RM0022
Reference manual

32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications

Introduction

The SPC560P44Lx, SPC560P50Lx microcontroller is built on the Power Architecture® platform and targets chassis and safety market segment, specifically the Electrical Hydraulic Power Steering (EHPS), the lower end of Electric Power Steering (EPS) and the airbag application space. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers.

EHPS and EPS systems typically feature sophisticated and advanced electrical motor control periphery with special enhancements in the area of pulse width modulation, highly flexible timers, and functional safety.

The safety features included in SPC560P44Lx, SPC560P50Lx (like fault collection unit, safety port or flash and SRAM with ECC) support the design of system applications where safety is a requirement.

Contents

List of tables

List of figures

Preface

Overview

The primary objective of this document is to define the functionality of the SPC560P44Lx, SPC560P50Lx family of microcontrollers for use by software and hardware developers. The SPC560P44Lx, SPC560P50Lx family is built on Power Architecture® technology and integrates technologies that are important for today's electrical hydraulic power steering (EHPS), electric power steering (EPS), and airbag applications.

As with any technical documentation, it is the reader's responsibility to be sure he or she is using the most recent version of the documentation.

To locate any published errata or updates for this document, visit the ST Web site at www.st.com.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the SPC560P44Lx, SPC560P50Lx device. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the Power Architecture.

Chapter organization and device-specific information

This document includes chapters that describe:

- The device as a whole
- The functionality of the individual modules on the device

In the latter, any device-specific information is presented in the section "Information Specific to This Device" at the beginning of the chapter.

References

In addition to this reference manual, the following documents provide additional information on the operation of the SPC560P44Lx, SPC560P50Lx:

- IEEE-ISTO 5001™ - 2003 and 2010, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface
- IEEE 1149.1-2001 standard - IEEE Standard Test Access Port and Boundary-Scan Architecture

1 Introduction

1.1 The SPC560P44Lx, SPC560P50Lx microcontroller family

The SPC560P44Lx, SPC560P50Lx microcontroller is built on the Power Architecture® platform and targets chassis and safety market segment, specifically the Electrical Hydraulic Power Steering (EHPS), the lower end of Electric Power Steering (EPS) and the airbag application space. The Power Architecture based 32-bit microcontrollers represent the latest achievement in integrated automotive application controllers.

EHPS and EPS systems typically feature sophisticated and advanced electrical motor control periphery with special enhancements in the area of pulse width modulation, highly flexible timers, and functional safety.

The safety features included in SPC560P44Lx, SPC560P50Lx (such as fault collection unit, safety port or flash and SRAM with ECC) support the design of system applications where safety is a requirement.

The core selected for the device is the Harvard bus interface version of the e200z0 to cover the low-end chassis application space.

The e200 processor family is a set of CPU cores that implement low-cost versions of the Power Architecture technology. The e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance. The e200z0 processor integrates an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable to sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by branch unit to allow single-cycle branches in some cases. The e200z0 core is a single-issue, 32-bit Power Architecture technology VLE only design with 32-bit general purpose registers (GPRs). All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). Instead of the base Power Architecture instruction set support, the e200z0 core only implements the VLE (variable length encoding) APU, providing improved code density.

The SPC560P44Lx, SPC560P50Lx has a single level of memory hierarchy consisting of 40 KB on-chip SRAM and 578 KB (512 KB program + 64 KB data) of on-chip flash memory. Both the SRAM and the flash memory can hold instructions and data.

The timer functions of SPC560P44Lx, SPC560P50Lx are performed by the eTimer Modular Timer System and FlexPWM. The two eTimer modules implement enhanced timer features (six channels each for a total of 12) including dedicated motor control quadrature decode functionality and DMA support; the FlexPWM module consists of four submodules controlling a pair of PWM channels each: three submodules may be used to control the three phases of a motor and the additional pair to support DC-DC converter width modulation control.

Off-chip communication is performed by a suite of serial protocols including FlexRay, CANs, enhanced SPIs (DSPI), and SCIs (LINFlex).

The System Integration Unit Lite (SIUL) performs several chip-wide configuration functions. Pad configuration and general-purpose input/output (GPIO) are controlled from the SIUL. External interrupts and reset control are also found in the SIUL. The internal multiplexer sub-block (IOMUX) provides multiplexing of daisy chaining the DSPIs and external interrupt signal.

As the SPC560P44Lx, SPC560P50Lx is built on a wider legacy of Power Architecture-based devices, when applicable and possible, reusing or enhancement of existing IP, design and concepts is adopted.

1.2 Target applications

The SPC560P44Lx, SPC560P50Lx targets the automotive chassis and safety market; specifically the electrical hydraulic power steering, the lower end of the electric power steering and the airbag application space.

1.2.1 Application examples

1.2.1.1 Electric power steering

Figure 1 outlines a typical electric power steering application built around the SPC560P44Lx, SPC560P50Lx microcontroller.

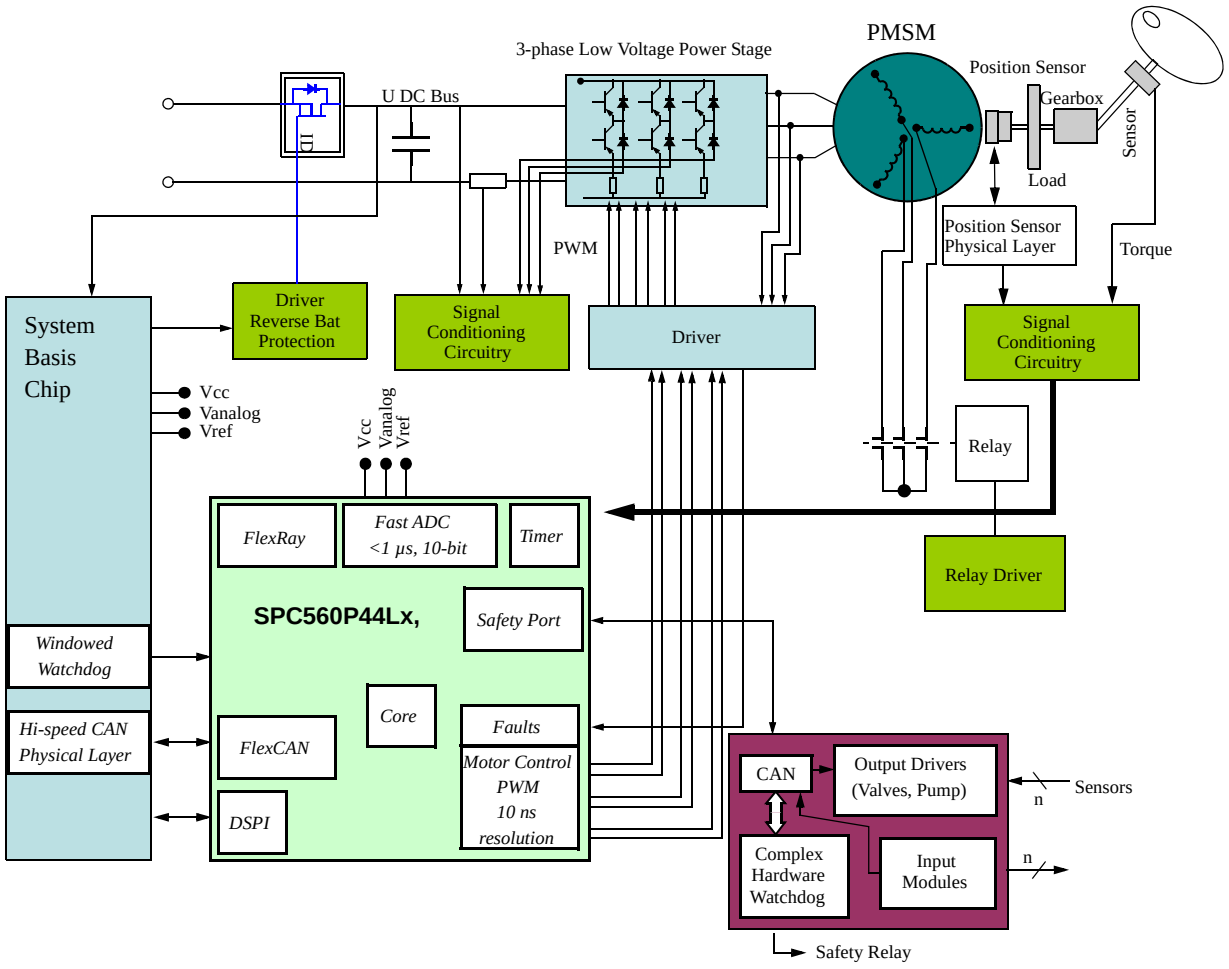


Figure 1. Electric power steering application

1.2.1.2 Airbag

Figure 2 outlines a typical airbag application built around the SPC560P44Lx, SPC560P50Lx microcontroller.

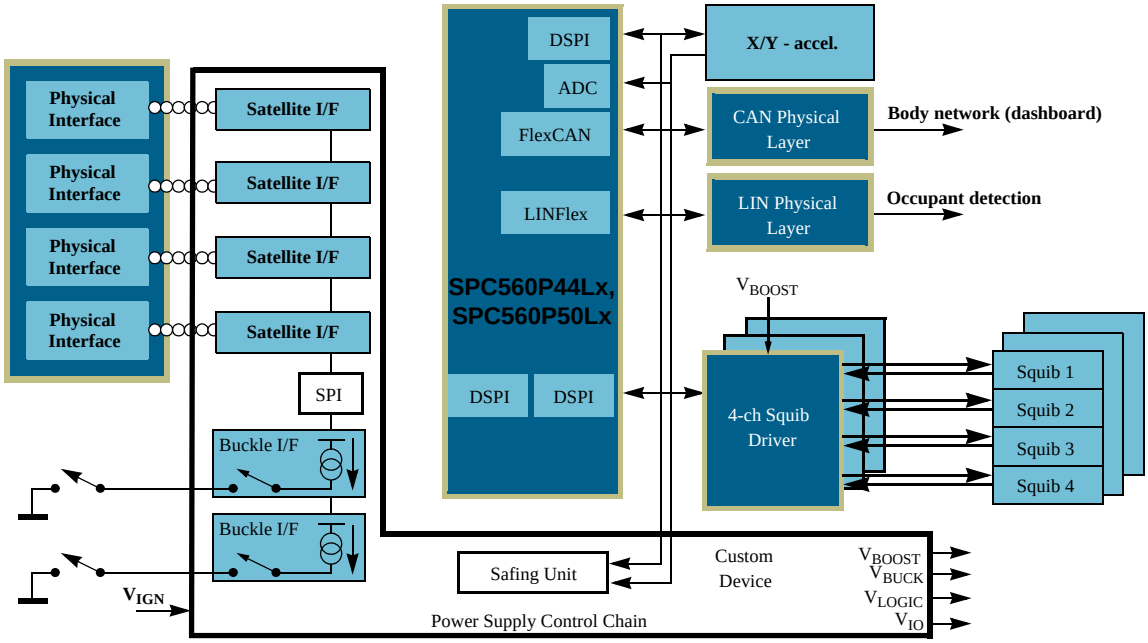


Figure 2. Airbag application

1.3 Features

Table 1 provides a summary of different members of the SPC560P44Lx, SPC560P50Lx family and their features—relative to full-featured version—to enable a comparison among the family members and an understanding of the range of functionality offered within this family.

Table 1. SPC560P44Lx, SPC560P50Lx device comparison

Feature	SPC560P44	SPC560P50
Code flash memory (with ECC)	384 KB	512 KB
Data flash memory / EE option (with ECC)	64 KB	
SRAM (with ECC)	36 KB	40 KB
Processor core	32-bit e200z0h	
Instruction set	VLE (variable length encoding)	
CPU performance	0–64 MHz	

Table 1. SPC560P44Lx, SPC560P50Lx device comparison(Continued)

Feature		SPC560P44	SPC560P50
FMPLL (frequency-modulated phase-locked loop) module		2	
INTC (interrupt controller) channels		147	
PIT (periodic interrupt timer)		1 (includes four 32-bit timers)	
eDMA (enhanced direct memory access) channels		16	
FlexRay		Yes(1)	
FlexCAN (controller area network)		2(2),(3)	
Safety port		Yes (via second FlexCAN module)	
FCU (fault collection unit)		Yes	
CTU (cross triggering unit)		Yes	
eTimer		2 (16-bit, 6 channels)	
FlexPWM (pulse-width modulation) channels		8 (capturing on X-channels)	
ADC (analog-to-digital converter)		2 (10-bit, 15-channel(4))	
LINFlex		2	
DSPI (deserial serial peripheral interface)		4	
CRC (cyclic redundancy check) unit		Yes	
JTAG controller		Yes	
Nexus port controller (NPC)		Yes (Level 2+)	
Supply	Digital power supply(5)	3.3 V or 5 V single supply with external transistor	
	Analog power supply	3.3 V or 5 V	
	Internal RC oscillator	16 MHz	
	External crystal oscillator	4–40 MHz	
Packages		LQFP100 LQFP144	
Temperature	Standard ambient temperature	–40 to 125 °C	

1. 32 message buffers, selectable single or dual channel support
2. Each FlexCAN module has 32 message buffers.
3. One FlexCAN module can act as a Safety Port with a bit rate as high as 7.5 Mbit/s.
4. Four channels shared between the two ADCs
5. The different supply voltages vary according to the part number ordered.

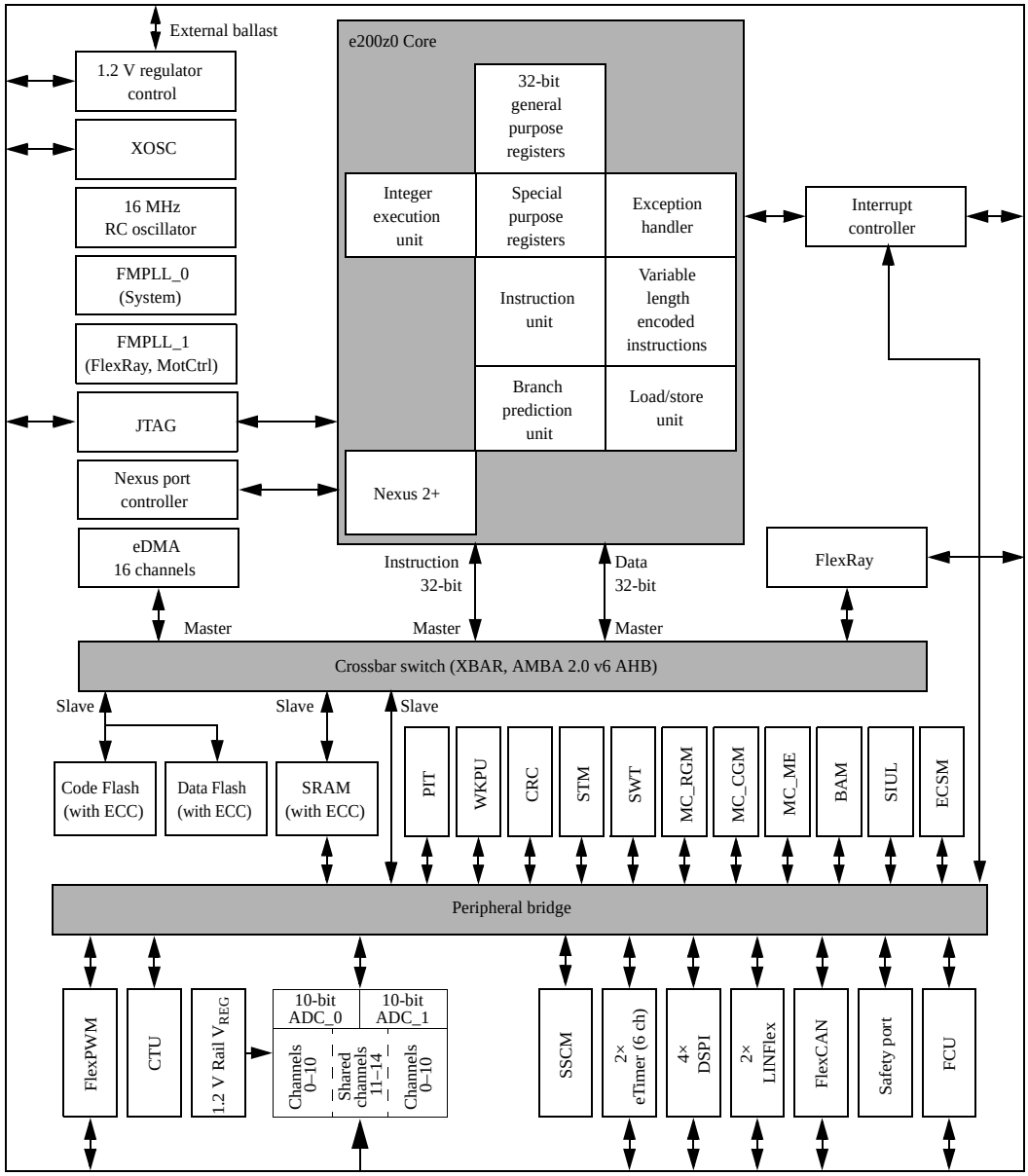
SPC560P44Lx, SPC560P50Lx is available in two configurations having different features: full-featured and airbag. *Table 2* shows the main differences between the two versions.

Table 2. SPC560P44Lx, SPC560P50Lx device configuration differences

Feature	Full-featured	Airbag
CTU (cross triggering unit)	Yes	No
FlexPWM	Yes	No
FlexRay	Yes	No
FMPLL (frequency-modulated phase-locked loop) module	2 (one FMPLL, one for FlexRay)	1 (only FMPLL)

1.4 Block diagram

Figure 3 shows a top-level block diagram of the SPC560P44Lx, SPC560P50Lx microcontroller.



Legend:

ADC	Analog-to-digital converter
BAM	Boot assist module
CRC	Cyclic redundancy check
CTU	Cross triggering unit
DSPI	Deserial serial peripheral interface
ECSM	Error correction status module
eDMA	Enhanced direct memory access
eTimer	Enhanced timer
FCU	Fault collection unit
Flash	Flash memory
FlexCAN	Controller area network
FlexPWM	Flexible pulse width modulation
FMPLL	Frequency-modulated phase-locked loop
INTC	Interrupt controller
JTAG	JTAG controller
LINFlex	Serial communication interface (LIN support)
MC_CGM	Clock generation module
MC_ME	Mode entry module
MC_PCU	Power control unit
MC_RGM	Reset generation module
PIT	Periodic interrupt timer
SIUL	System integration unit Lite
SRAM	Static random-access memory
SSCM	System status and configuration module
STM	System timer module
SWT	Software watchdog timer
WKPU	Wakeup unit
XOSC	External oscillator
XBAR	Crossbar switch

Figure 3. SPC560P44Lx, SPC560P50Lx block diagram

1.5 Critical performance parameters

- Fully static operation, 0 – 64 MHz
- –40 °C to 150 °C junction temperature
- Low power design
  - Halt and STOP mode available for power reduction
  - Resuming from Halt/STOP mode can be initiated via external pin
- Fabricated in 90 nm process
- 1.2 V nominal internal logic
- Nexus pins operate at V_{DDIO} (no dedicated power supply)
  - Unused pins configurable as GPIO
- 10-bit ADC conversion time < 1 µs
- Internal voltage regulator (VREG) with external ballast transistor enables control with a single input rail
  - 3.0 V–3.6 V or 4.5 V–5.5 V input supply voltage
- Configurable pins
  - Selectable slew rate for EMI reduction
  - Selectable pull-up, pull-down, or no pull on all pins
  - Selectable open drain
  - Support for 3.3 V or 5 V I/O levels

1.6 Chip-level features

On-chip modules available within the family include the following features:

1.6.1 Features

- 64 MHz, single issue, 32-bit CPU core complex (e200z0h)
  - Compliant with Power Architecture® embedded category
  - Variable Length Encoding (VLE)
- Memory organization
  - Up to 512 KB on-chip code flash memory with ECC and erase/program controller
  - Additional 64 (4 × 16) KB on-chip data flash memory with ECC for EEPROM emulation
  - Up to 40 KB on-chip SRAM with ECC
- Fail safe protection
  - Programmable watchdog timer
  - Non-maskable interrupt
  - Fault collection unit
- Nexus L2+ interface
- Interrupts
  - 16-channel eDMA controller
  - 16 priority level controller
- General purpose I/Os individually programmable as input, output or special function
- 2 general purpose eTimer units
  - 6 timers each with up/down count capabilities
  - 16-bit resolution, cascadable counters
  - Quadrature decode with rotation direction flag
  - Double buffer input capture and output compare
- Communications interfaces
  - 2 LINFlex channels (LIN 2.1)
  - 4 DSPI channels with automatic chip select generation
  - 1 FlexCAN interface (2.0B Active) with 32 message objects
  - 1 safety port based on FlexCAN with 32 message objects and up to 7.5 Mbit/s capability; usable as second CAN when not used as safety port
  - 1 FlexRay™ module (V2.1) with selectable dual or single channel support, 32 message objects and up to 10 Mbit/s (512 KB device only)
- Two 10-bit analog-to-digital converters (ADC)
  - 2 × 11 input channels, + 4 shared channels
  - Conversion time < 1 µs including sampling time at full precision
  - Programmable ADC Cross Triggering Unit (CTU)
  - 4 analog watchdogs with interrupt capability
- On-chip CAN/UART bootstrap loader with Boot Assist Module (BAM)
- 1 FlexPWM unit: 8 complementary or independent outputs with ADC synchronization signals

Table 3. Device summary

Package	Part number	
	448 KB Flash	576 KB Flash
LQFP144	SPC560P44L5	SPC560P50L5
LQFP100	SPC560P44L3	SPC560P50L3

1.7 Module features

1.7.1 High performance e200z0 core processor

The e200z0 Power Architecture core provides the following features:

- High performance e200z0 core processor for managing peripherals and interrupts
- Single issue 4-stage pipeline in-order execution 32-bit Power Architecture CPU
- Harvard architecture
- Variable length encoding (VLE), allowing mixed 16-bit and 32-bit instructions
  - Results in smaller code size footprint
  - Minimizes impact on performance
- Branch processing acceleration using lookahead instruction buffer
- Load/store unit
  - 1 cycle load latency
  - Misaligned access support
  - No load-to-use pipeline bubbles
- Thirty-two 32-bit general purpose registers (GPRs)
- Separate instruction bus and load/store bus Harvard architecture
- Hardware vectored interrupt support
- Reservation instructions for implementing read-modify-write constructs
- Long cycle time instructions, except for guarded loads, do not increase interrupt latency
- Extensive system development support through Nexus debug port
- Non-maskable interrupt support

1.7.2 Crossbar switch (XBAR)

The XBAR multi-port crossbar switch supports simultaneous connections between four master ports and three slave ports. The crossbar supports a 32-bit address bus width and a 32-bit data bus width.

The crossbar allows for two concurrent transactions to occur from any master port to any slave port; but one of those transfers must be an instruction fetch from internal flash memory. If a slave port is simultaneously requested by more than one master port, arbitration logic will select the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port will be stalled until the higher priority master completes its transactions. Requesting masters will be treated with equal priority and will be

granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access.

The crossbar provides the following features:

- 4 master ports:
  - e200z0 core complex Instruction port
  - e200z0 core complex Load/Store Data port
  - eDMA
  - FlexRay
- 3 slave ports:
  - Flash memory (code flash and data flash)
  - SRAM
  - Peripheral bridge
- 32-bit internal address, 32-bit internal data paths
- Fixed Priority Arbitration based on Port Master
- Temporary dynamic priority elevation of masters

1.7.3 Enhanced direct memory access (eDMA)

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data movements via 16 programmable channels, with minimal intervention from the host processor. The hardware micro architecture includes a DMA engine which performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels. This implementation is utilized to minimize the overall block size.

The eDMA module provides the following features:

- 16 channels support independent 8, 16 or 32-bit single value or block transfers
- Supports variable sized queues and circular queues
- Source and destination address registers are independently configured to either post-increment or to remain constant
- Each transfer is initiated by a peripheral, CPU, or eDMA channel request
- Each eDMA channel can optionally send an interrupt request to the CPU on completion of a single value or block transfer
- DMA transfers possible between system memories, DSPIs, ADC, FlexPWM, eTimer and CTU
- Programmable DMA channel multiplexer for assignment of any DMA source to any available DMA channel with as many as 30 request sources
- eDMA abort operation through software

1.7.4 Flash memory

The SPC560P44Lx, SPC560P50Lx provides as much as 576 KB of programmable, non-volatile, flash memory. The non-volatile memory (NVM) can be used for instruction and/or data storage. The flash memory module interfaces the system bus to a dedicated flash memory array controller. It supports a 32-bit data bus width at the system bus port, and a 128-bit read data interface to flash memory. The module contains four 128-bit wide prefetch

buffers. Prefetch buffer hits allow no-wait responses. Normal flash memory array accesses are registered and are forwarded to the system bus on the following cycle, incurring two wait-states.

The flash memory module provides the following features:

- As much as 576 KB flash memory
  - 8 blocks (32 KB + 2×16 KB + 32 KB + 32 KB + 3×128 KB) code flash
  - 4 blocks (16 KB + 16 KB + 16 KB + 16 KB) data flash
  - Full Read While Write (RWW) capability between code and data flash
- Four 128-bit wide prefetch buffers to provide single cycle in-line accesses (prefetch buffers can be configured to prefetch code or data or both)
- Typical flash memory access time: 0 wait states for buffer hits, 2 wait states for page buffer miss at 64 MHz
- Hardware managed flash memory writes handled by 32-bit RISC Krypton engine
- Hardware and software configurable read and write access protections on a per-master basis
- Configurable access timing allowing use in a wide range of system frequencies
- Multiple-mapping support and mapping-based block access timing (up to 31 additional cycles) allowing use for emulation of other memory types.
- Software programmable block program/erase restriction control
- Erase of selected block(s)
- Read page sizes
  - Code flash memory: 128 bits (4 words)
  - Data flash memory: 32 bits (1 word)
- ECC with single-bit correction, double-bit detection for data integrity
  - Code flash memory: 64-bit ECC
  - Data flash memory: 64-bit ECC
- Embedded hardware program and erase algorithm
- Erase suspend, program suspend and erase-suspended program
- Censorship protection scheme to prevent flash memory content visibility
- Hardware support for EEPROM emulation

1.7.5 Static random access memory (SRAM)

The SPC560P44Lx, SPC560P50Lx SRAM module provides up to 40 KB of general-purpose memory.

The SRAM module provides the following features:

- Supports read/write accesses mapped to the SRAM from any master
- Up to 40 KB general purpose SRAM
- Supports byte (8-bit), half word (16-bit), and word (32-bit) writes for optimal use of memory
- Typical SRAM access time: 0 wait-state for reads and 32-bit writes; 1 wait state for 8- and 16-bit writes if back to back with a read to same memory block

1.7.6 Interrupt controller (INTC)

The interrupt controller (INTC) provides priority-based preemptive scheduling of interrupt requests, suitable for statically scheduled hard real-time systems. The INTC handles 147 selectable-priority interrupt sources.

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is executing the interrupt service routine (ISR) has been minimized. The INTC provides a unique vector for each interrupt request source for quick determination of which ISR has to be executed. It also provides a wide number of priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. To allow the appropriate priorities for each source of interrupt request, the priority of each interrupt request is software configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol (PCP) for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the same resource can not preempt each other.

The INTC provides the following features:

- Unique 9-bit vector for each separate interrupt source
- 8 software triggerable interrupt sources
- 16 priority levels with fixed hardware arbitration within priority levels for each interrupt source
- Ability to modify the ISR or task priority: modifying the priority can be used to implement the Priority Ceiling Protocol for accessing shared resources.
- 2 external high priority interrupts directly accessing the main core and I/O processor (IOP) critical interrupt mechanism

1.7.7 System status and configuration module (SSCM)

The system status and configuration module (SSCM) provides central device functionality.

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search code flash for bootable sector
  - DMA status
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

1.7.8 System clocks and clock generation

The following list summarizes the system clock and clock generation on the SPC560P44Lx, SPC560P50Lx:

- Lock detect circuitry continuously monitors lock status
- Loss of clock (LOC) detection for PLL outputs
- Programmable output clock divider (÷1, ÷2, ÷4, ÷8)
- FlexPWM module and eTimer module can run on an independent clock source
- On-chip oscillator with automatic level control
- Internal 16 MHz RC oscillator for rapid start-up and safe mode: supports frequency trimming by user application

1.7.9 Frequency-modulated phase-locked loop (FMPLL)

The FMPLL allows the user to generate high speed system clocks from a 4–40 MHz input clock. Further, the FMPLL supports programmable frequency modulation of the system clock. The PLL multiplication factor, output clock divider ratio are all software configurable.

The PLL has the following major features:

- Input clock frequency: 4–40 MHz
- Maximum output frequency: 64 MHz
- Voltage controlled oscillator (VCO)—frequency 256–512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the PLL to relock
- Frequency-modulated PLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth (±0.25% to ±4% deviation from center frequency): programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation

1.7.10 Main oscillator

The main oscillator provides these features:

- Input frequency range: 4–40 MHz
- Crystal input mode or oscillator input mode
- PLL reference

1.7.11 Internal RC oscillator

This device has an RC ladder phase-shift oscillator. The architecture uses constant current charging of a capacitor. The voltage at the capacitor is compared by the stable bandgap reference voltage.

The RC oscillator provides these features:

- Nominal frequency 16 MHz
- ±5% variation over voltage and temperature after process trim
- Clock output of the RC oscillator serves as system clock source in case loss of lock or loss of clock is detected by the PLL
- RC oscillator is used as the default system clock during startup

1.7.12 Periodic interrupt timer (PIT)

The PIT module implements these features:

- 4 general purpose interrupt timers
- 32-bit counter resolution
- Clocked by system clock frequency
- Each channel can be used as trigger for a DMA request

1.7.13 System timer module (STM)

The STM module implements these features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

1.7.14 Software watchdog timer (SWT)

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of system or oscillator clock for timer operation
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- Reset configuration inputs allow timer to be enabled out of reset

1.7.15 Fault collection unit (FCU)

The FCU provides an independent fault reporting mechanism even if the CPU is malfunctioning.

The FCU module has the following features:

- FCU status register reporting the device status
- Continuous monitoring of critical fault signals
- User selection of critical signals from different fault sources inside the device
- Critical fault events trigger 2 external pins (user selected signal protocol) that can be used externally to reset the device and/or other circuitry (for example, safety relay or FlexRay transceiver)
- Faults are latched into a register

1.7.16 System integration unit – Lite (SIUL)

The SPC560P44Lx, SPC560P50Lx SIUL controls MCU pad configuration, external interrupt, general purpose I/O (GPIO), and internal peripheral multiplexing.

The pad configuration block controls the static electrical characteristics of I/O pins. The GPIO block provides uniform and discrete input/output control of the I/O pins of the MCU.

The SIU provides the following features:

- Centralized general purpose input output (GPIO) control of as many as 80 input/output pins and 26 analog input-only pads (package dependent)
- All GPIO pins can be independently configured to support pull-up, pull down, or no pull
- Reading and writing to GPIO supported both as individual pins and 16-bit wide ports
- All peripheral pins (except ADC channels) can be alternatively configured as both general purpose input or output pins
- ADC channels support alternative configuration as general purpose inputs
- Direct readback of the pin value is supported on all pins through the SIUL
- Configurable digital input filter that can be applied to some general purpose input pins for noise elimination: as many as 4 internal functions can be multiplexed onto 1 pin

1.7.17 Boot and censorship

Different booting modes are available in the SPC560P44Lx, SPC560P50Lx: booting from internal flash memory and booting via a serial link.

The default booting scheme uses the internal flash memory (an internal pull-down is used to select this mode). Optionally, the user can boot via FlexCAN or LINFlex (using the boot assist module software).

A censorship scheme is provided to protect the content of the flash memory and offer increased security for the entire device.

A password mechanism is designed to grant the legitimate user access to the non-volatile memory.

1.7.17.1 Boot assist module (BAM)

The BAM is a block of read-only one-time programmed memory and is identical for all SPC560Pxx devices that are based on the e200z0h core. The BAM program is executed every time the device is powered on if the alternate boot mode has been selected by the user.

The BAM provides the following features:

- Serial bootloading via FlexCAN or LINFlex
- Ability to accept a password via the used serial communication channel to grant the legitimate user access to the non-volatile memory

1.7.18 Error correction status module (ECSM)

The ECSM provides a myriad of miscellaneous control functions regarding program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, wakeup control for exiting sleep modes, and information on platform memory errors reported by error-correcting codes and/or generic access error information for certain processor cores.

The Error Correction Status Module supports a number of miscellaneous control functions for the platform. The ECSM includes these features:

- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- For test purposes, optional registers to specify the generation of double-bit memory errors are enabled on the SPC560P44Lx, SPC560P50Lx.

The sources of the ECC errors are:

- Flash memory
- SRAM

1.7.19 Peripheral bridge (PBRIDGE)

The PBRIDGE implements the following features:

- Duplicated periphery
- Master access privilege level per peripheral (per master: read access enable; write access enable)
- Write buffering for peripherals
- Checker applied on PBRIDGE output toward periphery
- Byte endianess swap capability

1.7.20 Controller area network (FlexCAN)

The SPC560P44Lx, SPC560P50Lx MCU contains one controller area network (FlexCAN) module. This module is a communication controller implementing the CAN protocol according to Bosch Specification version 2.0B. The CAN protocol was designed to be used primarily as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module contains 32 message buffers.

The FlexCAN module provides the following features:

- Full implementation of the CAN protocol specification, version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - Up to 8-bytes data length
  - Programmable bit rate up to 1 Mbit/s
- 32 message buffers of up to 8-bytes data length
- Each message buffer configurable as Rx or Tx, all supporting standard and extended messages
- Programmable loop-back mode supporting self-test operation
- 3 programmable mask registers
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)

- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages
- Transmit features
  - Supports configuration of multiple mailboxes to form message queues of scalable depth
  - Arbitration scheme according to message ID or message buffer number
  - Internal arbitration to guarantee no inner or outer priority inversion
  - Transmit abort procedure and notification
- Receive features
  - Individual programmable filters for each mailbox
  - 8 mailboxes configurable as a six-entry receive FIFO
  - 8 programmable acceptance filters for receive FIFO
- Programmable clock source
  - System clock
  - Direct oscillator clock to avoid PLL jitter

1.7.21 Safety port (FlexCAN)

The SPC560P44Lx, SPC560P50Lx MCU has a second CAN controller synthesized to run at high bit rates to be used as a safety port. The CAN module of the safety port provides the following features:

- Identical to the FlexCAN module
- Bit rate as fast as 7.5 Mbit/s at 60 MHz CPU clock using direct connection between CAN modules (no physical transceiver required)
- 32 message buffers of up to 8 bytes data length
- Can be used as a second independent CAN module

1.7.22 FlexRay

The FlexRay module provides the following features:

- Full implementation of FlexRay Protocol Specification 2.1
- 32 configurable message buffers can be handled
- Dual channel or single channel mode of operation, each as fast as 10 Mbit/s data rate
- Message buffers configurable as Tx, Rx or RxFIFO
- Message buffer size configurable
- Message filtering for all message buffers based on FrameID, cycle count and message ID
- Programmable acceptance filters for RxFIFO message buffers

1.7.23 Serial communication interface module (LINFlex)

The LINFlex (local interconnect network flexible) on the SPC560P44Lx, SPC560P50Lx features the following:

- Supports LIN Master mode, LIN Slave mode and UART mode
- LIN state machine compliant to LIN1.3, 2.0, and 2.1 specifications
- Handles LIN frame transmission and reception without CPU intervention
- LIN features
  - Autonomous LIN frame handling
  - Message buffer to store Identifier and as much as 8 data bytes
  - Supports message length as long as 64 bytes
  - Detection and flagging of LIN errors (sync field, delimiter, ID parity, bit framing, checksum, and time-out)
  - Classic or extended checksum calculation
  - Configurable Break duration as long as 36-bit times
  - Programmable baud rate prescalers (13-bit mantissa, 4-bit fractional)
  - Diagnostic features: Loop back; Self Test; LIN bus stuck dominant detection
  - Interrupt-driven operation with 16 interrupt sources
- LIN slave mode features
  - Autonomous LIN header handling
  - Autonomous LIN response handling
- UART mode
  - Full-duplex operation
  - Standard non return-to-zero (NRZ) mark/space format
  - Data buffers with 4-byte receive, 4-byte transmit
  - Configurable word length (8-bit or 9-bit words)
  - Error detection and flagging
  - Parity, Noise and Framing errors
  - Interrupt-driven operation with four interrupt sources
  - Separate transmitter and receiver CPU interrupt sources
  - 16-bit programmable baud-rate modulus counter and 16-bit fractional
  - 2 receiver wake-up methods

1.7.24 Deserial serial peripheral interface (DSPI)

The deserial serial peripheral interface (DSPI) module provides a synchronous serial interface for communication between the SPC560P44Lx, SPC560P50Lx MCU and external devices.

The DSPI modules provide these features:

- Full duplex, synchronous transfers
- Master or slave operation
- Programmable master bit rates
- Programmable clock polarity and phase
- End-of-transmission interrupt flag
- Programmable transfer baud rate
- Programmable data frames from 4 to 16 bits
- Up to 20 chip select lines available
  - 8 on DSPI_0
  - 4 each on DSPI_1, DSPI_2 and DSPI_3
- 8 clock and transfer attributes registers
- Chip select strobe available as alternate function on one of the chip select pins for deglitching
- FIFOs for buffering as many as 5 transfers on the transmit and receive side
- Queueing operation possible through use of the eDMA
- General purpose I/O functionality on pins when not used for SPI

1.7.25 Pulse width modulator (FlexPWM)

The pulse width modulator module (PWM) contains four PWM submodules, each capable of controlling a single half-bridge power stage. There are also four fault channels.

This PWM is capable of controlling most motor types: AC induction motors (ACIM), permanent magnet AC motors (PMAC), both brushless (BLDC) and brush DC motors (BDC), switched (SRM) and variable reluctance motors (VRM), and stepper motors.

The FlexPWM block implements the following features:

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- Maximum operating clock frequency of 120 MHz
- PWM outputs can operate as complementary pairs or independent channels
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half cycle reload capability
- Multiple ADC trigger events can be generated per PWM cycle via hardware
- Write protection for critical registers
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a “Force Out” event
- PWMX pin can optionally output a third PWM signal from each submodule
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual-edge capture functionality
- eDMA support with automatic reload
- 2 fault inputs
- Capture capability for PWMA, PWMB, and PWMX channels not supported

1.7.26 eTimer

The SPC560P44Lx, SPC560P50Lx includes two eTimer modules. Each module provides six 16-bit general purpose up/down timer/counter units with the following features:

- Maximum operating clock frequency of 120 MHz
- Individual channel capability
  - Input capture trigger
  - Output compare
  - Double buffer (to capture rising edge and falling edge)
  - Separate prescaler for each counter
  - Selectable clock source
  - 0–100% pulse measurement
  - Rotation direction flag (Quad decoder mode)
- Maximum count rate
  - External event counting: max. count rate = peripheral clock/2
  - Internal clock counting: max. count rate = peripheral clock
- Counters are:
  - Cascadable
  - Preloadable
- Programmable count modulo
- Quadrature decode capabilities
- Counters can share available input pins
- Count once or repeatedly
- Pins available as GPIO when timer functionality not in use

1.7.27 Analog-to-digital converter (ADC) module

The ADC module provides the following features:

Analog part:

- 2 on-chip AD converters
  - 10-bit AD resolution
  - 1 sample and hold unit per ADC
  - Conversion time, including sampling time, less than 1 µs (at full precision)
  - Typical sampling time is 150 ns min. (at full precision)
  - Differential non-linearity error (DNL) ±1 LSB
  - Integral non-linearity error (INL) ±1.5 LSB
  - TUE <3 LSB
  - Single-ended input signal up to 5.0 V
  - The ADC and its reference can be supplied with a voltage independent from V_{DDIO}
  - The ADC supply can be equal or higher than V_{DDIO}
  - The ADC supply and the ADC reference are not independent from each other (they are internally bonded to the same pad)
  - Sample times of 2 (default), 8, 64, or 128 ADC clock cycles

Digital part:

- 2 × 13 input channels including 4 channels shared between the 2 converters
- 4 analog watchdogs comparing ADC results against predefined levels (low, high, range) before results are stored in the appropriate ADC result location,
- 2 modes of operation: Normal mode or CTU control mode
- Normal mode features
  - Register-based interface with the CPU: control register, status register, 1 result register per channel
  - ADC state machine managing 3 request flows: regular command, hardware injected command, software injected command
  - Selectable priority between software and hardware injected commands
  - 4 analog watchdogs comparing ADC results against predefined levels (low, high, range)
  - DMA compatible interface
- CTU control mode features
  - Triggered mode only
  - 4 independent result queues (2 × 16 entries, 2 × 4 entries)
  - Result alignment circuitry (left justified; right justified)
  - 32-bit read mode allows to have channel ID on one of the 16-bit part
  - DMA compatible interfaces

1.7.28 Cross triggering unit (CTU)

The cross triggering unit allows automatic generation of ADC conversion requests on user selected conditions without CPU load during the PWM period and with minimized CPU load for dynamic configuration.

It implements the following features:

- Double buffered trigger generation unit with as many as eight independent triggers generated from external triggers
- Trigger generation unit configurable in sequential mode or in triggered mode
- Each Trigger can be appropriately delayed to compensate the delay of external low pass filter
- Double buffered global trigger unit allowing eTimer synchronization and/or ADC command generation
- Double buffered ADC command list pointers to minimize ADC-trigger unit update
- Double buffered ADC conversion command list with as many as 24 ADC commands
- Each trigger has the capability to generate consecutive commands
- ADC conversion command allows to control ADC channel from each ADC, single or synchronous sampling, independent result queue selection

1.7.29 Nexus development interface (NDI)

The NDI (Nexus Development Interface) block provides real-time development support capabilities for the SPC560P44Lx, SPC560P50Lx Power Architecture based MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility. The NDI

block is an integration of several individual Nexus blocks that are selected to provide the development support interface for this device. The NDI block interfaces to the host processor and internal busses to provide development support as per the IEEE-ISTO 5001-2003 Class 2+ standard. The development support provided includes access to the MCU's internal memory map and access to the processor's internal registers during run time.

The Nexus Interface provides the following features:

- Configured via the IEEE 1149.1
- All Nexus port pins operate at V_{DDIO} (no dedicated power supply)
- Nexus 2+ features supported
  - Static debug
  - Watchpoint messaging
  - Ownership trace messaging
  - Program trace messaging
  - Real time read/write of any internally memory mapped resources through JTAG pins
  - Overrun control, which selects whether to stall before Nexus overruns or keep executing and allow overwrite of information
  - Watchpoint triggering, watchpoint triggers program tracing
- Auxiliary Output Port
  - 4 MDO (Message Data Out) pins
  - MCKO (Message Clock Out) pin
  - 2 $\overline{\text{MSEO}}$ (Message Start/End Out) pins
  - $\overline{\text{EVTO}}$ (Event Out) pin
- Auxiliary Input Port
  - $\overline{\text{EVTI}}$ (Event In) pin

1.7.30 Cyclic redundancy check (CRC)

The CRC computing unit is dedicated to the computation of CRC off-loading the CPU. The CRC module features:

- Support for CRC-16-CCITT (*x*25 protocol):
  - $x^{16} + x^{12} + x^5 + 1$
- Support for CRC-32 (Ethernet protocol):
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
- Zero wait states for each write/read operations to the CRC_CFG and CRC_INP registers at the maximum frequency

1.7.31 IEEE 1149.1 JTAG controller

The JTAG controller (JTAGC) block provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. All data input to and output from the JTAGC block is communicated in serial format. The JTAGC block is compliant with the IEEE standard.

The JTAG controller provides the following features:

- IEEE Test Access Port (TAP) interface with 4 pins (TDI, TMS, TCK, TDO)
- Selectable modes of operation include JTAGC/debug or normal system operation.
- A 5-bit instruction register that supports the following IEEE 1149.1-2001 defined instructions:
  - BYPASS, IDCODE, EXTEST, SAMPLE, SAMPLE/PRELOAD
- A 5-bit instruction register that supports the additional following public instructions:
  - ACCESS_AUX_TAP_NPC, ACCESS_AUX_TAP_ONCE
- 3 test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

1.7.32 On-chip voltage regulator (VREG)

The on-chip voltage regulator module provides the following features:

- Uses external NPN (negative-positive-negative) transistor
- Regulates external 3.3 V /5.0 V down to 1.2 V for the core logic
- Low voltage detection on the internal 1.2 V and I/O voltage 3.3 V

1.8 Developer environment

The SPC560P44Lx, SPC560P50Lx MCU tools and third-party developers offer a widespread, established network of tool and software vendors.

The following development support is available:

- Automotive Evaluation Boards (EVB) featuring CAN, LIN interfaces, and more
- Compilers
- Debuggers
- JTAG and Nexus interfaces
- Autocode generation tools
- Initialization tools

1.9 Package

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: www.st.com. ECOPACK® is an ST trademark.

SPC560P44Lx, SPC560P50Lx family members are offered in the following package types:

- 100-pin LQFP, 0.5 mm pitch, 14 mm × 14 mm outline
- 144-pin LQFP 0.5 mm pitch, 20 mm × 20 mm outline

Table 4. SPC560P44Lx, SPC560P50Lx packages

Device	LQFP100	LQFP144
SPC560P44Lx, SPC560P50Lx	X	X

2 SPC560P44Lx, SPC560P50Lx memory map

Table 5 shows the memory map for the SPC560P44Lx, SPC560P50Lx. All addresses on the SPC560P44Lx, SPC560P50Lx, including those that are reserved, are identified in the table. The addresses represent the physical addresses assigned to each IP block.

Please see *Appendix B: Memory Map* for detailed memory map.

Table 5. Memory map

Start address	End address	Size (KB)	Region name
On-chip memory			
0x0000_0000	0x0007_FFFF	512	Code Flash Array 0
0x0008_0000	0x001F_FFFF	1536	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash Array 0 Shadow Sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash Array 0 Test Sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash Array 0
0x0081_0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C0_3FFF	16	Data Flash Array 0 Test Sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Flash Emulation Mapping
0x2000_0000	0x3FFF_FFFF	524288	Reserved
0x4000_0000	0x4000_9FFF	40	SRAM
0x4000_A000	0xC3F8_0000	1048536	Reserved
On-chip peripherals(1)			
0xC3F8_0000	0xC3F8_7FFF	32	Reserved
0xC3F8_8000	0xC3F8_BFFF	16	Code Flash 0 Configuration (CFLASH_0)
0xC3F8_C000	0xC3F8_FFFF	16	Data Flash 0 Configuration (DFLASH_0)
0xC3F9_0000	0xC3F9_3FFF	16	System Integration Unit Lite (SIUL)
0xC3F9_4000	0xC3F9_7FFF	16	WakeUp Unit (WKUP)
0xC3F9_8000	0xC3FD_7FFF	256	Reserved
0xC3FD_8000	0xC3FD_BFFF	16	System Status and Configuration Module (SSCM)
0xC3FD_C000	0xC3FD_FFFF	16	Mode Entry module (ME)
0xC3FE_0000	0xC3FE_3FFF	16	Clock Generation Module (CGM, XOSC, IRC, FMPLL_0, FMPLL_1, CMU_0, CMU_1)
0xC3FE_4000	0xC3FE_7FFF	16	Reset Generation Module (RGM)
0xC3FE_8000	0xC3FE_BFFF	16	Power Control Unit (PCU)(2)
0xC3FE_C000	0xC3FE_FFFF	16	Reserved
0xC3FF_0000	0xC3FF_3FFF	16	Periodic Interrupt Timer (PIT)

Table 5. Memory map(Continued)

Start address	End address	Size (KB)	Region name
0xC3FF_4000	0xC3FF_FFFF	48	Reserved
0xFFE0_0000	0xFFE0_3FFF	16	Analog to Digital Converter 0 (ADC_0)
0xFFE0_4000	0xFFE0_7FFF	16	Analog to Digital Converter 1 (ADC_1)
0xFFE0_8000	0xFFE0_BFFF	16	Reserved
0xFFE0_C000	0xFFE0_FFFF	16	CTU_0
0xFFE1_0000	0xFFE1_7FFF	32	Reserved
0xFFE1_8000	0xFFE1_BFFF	16	eTimer_0
0xFFE1_C000	0xFFE1_FFFF	16	eTimer_1
0xFFE2_0000	0xFFE2_3FFF	16	Reserved
0xFFE2_4000	0xFFE2_7FFF	16	FlexPWM_0
0xFFE2_8000	0xFFE3_FFFF	96	Reserved
0xFFE4_0000	0xFFE4_3FFF	16	LINFlex_0
0xFFE4_4000	0xFFE4_7FFF	16	LINFlex_1
0xFFE5_0000	0xFFE6_7FFF	128	Reserved
0xFFE6_8000	0xFFE6_BFFF	16	Cyclic Redundancy Check (CRC)
0xFFE6_C000	0xFFE6_FFFF	16	Fault Collection Unit (FCU)
0xFFE7_0000	0xFFE7_FFFF	64	Reserved
0xFFE8_0000	0xFFEF_FFFF	512	Mirrored (range 0xC3F8_0000 – 0xC3FF_FFFF)
0xFFF0_0000	0xFFF3_7FFF	224	Reserved
0xFFF3_8000	0xFFF3_BFFF	16	Software Watchdog (SWT_0)
0xFFF3_C000	0xFFF3_FFFF	16	System Timer Module (STM_0)
0xFFF4_0000	0xFFF4_3FFF	16	Error Correction Status Module (ECSM)
0xFFF4_4000	0xFFF4_7FFF	16	Enhanced Direct Memory Access Controller (eDMA)
0xFFF4_8000	0xFFF4_BFFF	16	Interrupt Controller (INTC)
0xFFF4_C000	0xFFF8_FFFF	272	Reserved
0xFFF9_0000	0xFFF9_3FFF	16	DSPI_0
0xFFF9_4000	0xFFF9_7FFF	16	DSPI_1
0xFFF9_8000	0xFFF9_BFFF	16	DSPI_2
0xFFF9_C000	0xFFF9_FFFF	16	DSPI_3
0xFFFA_0000	0xFFFB_FFFF	128	Reserved
0xFFFC_0000	0xFFFC_3FFF	16	FlexCAN_0 (CAN0)
0xFFFC_4000	0xFFFD_BFFF	96	Reserved
0xFFFD_C000	0xFFFD_FFFF	16	DMA Multiplexer (DMA_MUX)
0xFFFE_0000	0xFFFE_3FFF	16	FlexRay Controller (FlexRay)

Table 5. Memory map(Continued)

Start address	End address	Size (KB)	Region name
0xFFFE_4000	0xFFFE_7FFF	16	Reserved
0xFFFE_8000	0xFFFE_BFFF	16	Safety Port (FlexCAN)
0xFFFE_C000	0xFFFF_BFFF	64	Reserved
0xFFFF_C000	0xFFFF_FFFF	16	Boot Assist Module (BAM)

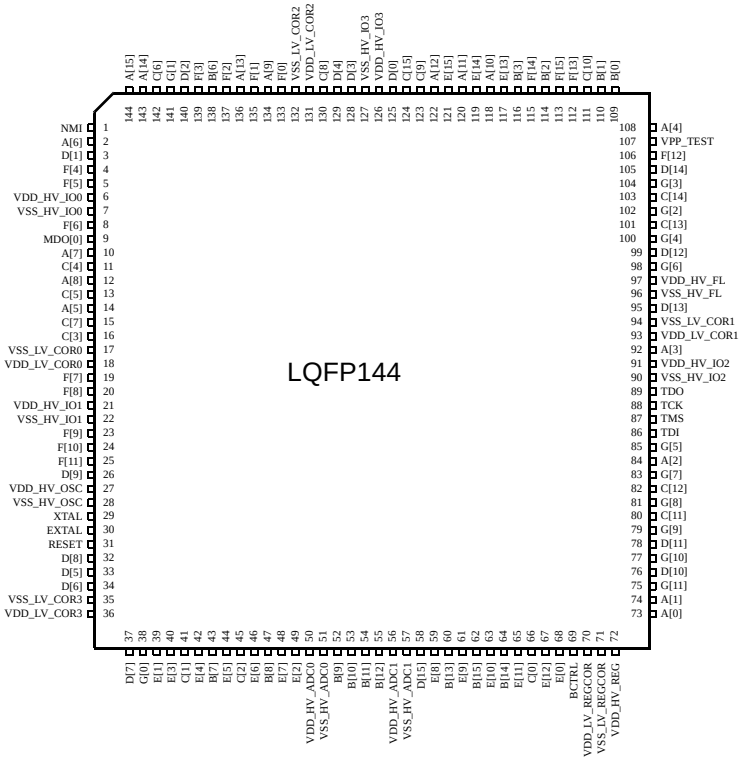
1. Running and Low Power Mode configurations for peripherals listed on *Table 45: ME registers* is related to Peripheral Control registers (ME_PCTL)
2. This address space contains also VREG registers. See *Chapter 35: Voltage Regulators and Power Supplies*.

3 Signal Description

This chapter describes the signals of the SPC560P44Lx, SPC560P50Lx. It includes a table of signal properties and detailed descriptions of signals.

3.1 144-pin LQFP pinout

Figure 4 shows the pinout of the 144-pin LQFP.

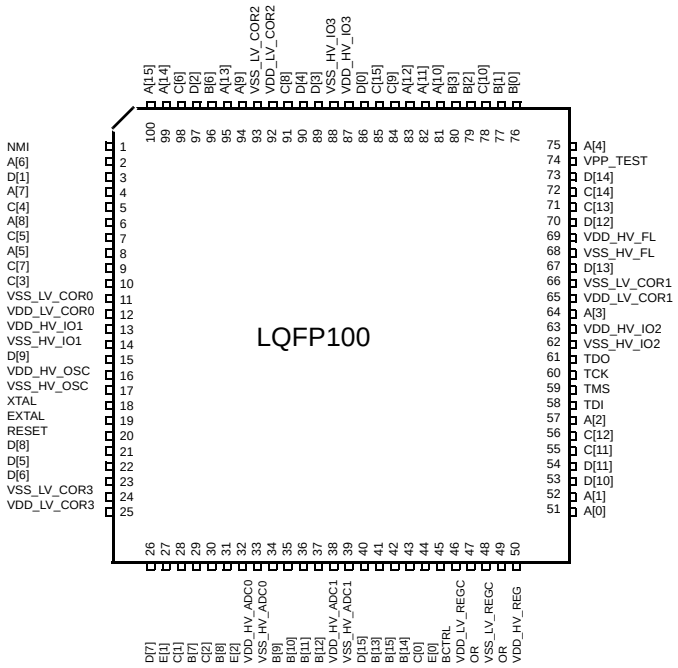


Note: Availability of port pin alternate functions depends on product selection.

Figure 4. 144-pin LQFP pinout – Full featured configuration (top view)

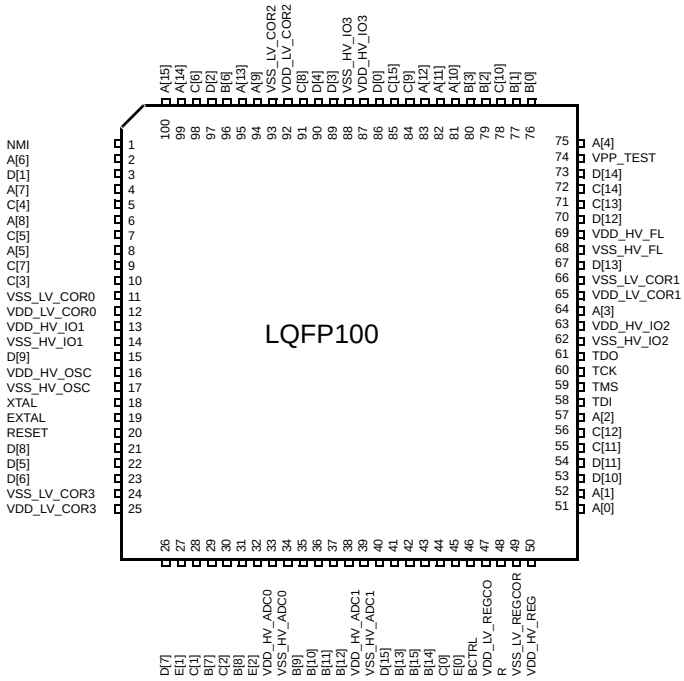
3.2 100-pin LQFP pinout

Figure 5 and *Figure 6* shows the pinout of the 100-pin LQFP.



Note: Availability of port pin alternate functions depends on product selection.

Figure 5. 100-pin LQFP pinout – Airbag configuration (top view)



Note: Availability of port pin alternate functions depends on product selection.

Figure 6. 100-pin LQFP pinout – Full featured configuration (top view)

3.3 Pin description

The following sections provide signal descriptions and related information about the functionality and configuration of the SPC560P44Lx, SPC560P50Lx devices.

3.3.1 Power supply and reference voltage pins

Table 6 lists the power supply and reference voltage for the SPC560P44Lx, SPC560P50Lx devices.

Table 6. Supply pins

Supply		Pin	
Symbol	**Description**	**100-pin**	**144-pin**
VREG control and power supply pins. Pins available on 100-pin and 144-pin package.			
BCTRL	Voltage regulator external NPN ballast base control pin	47	69
$V_{DD\_HV\_REG}$ (3.3 V or 5.0 V)	Voltage regulator supply voltage	50	72
$V_{DD\_LV\_REGCOR}$	1.2 V decoupling pins for core logic and regulator feedback. Decoupling capacitor must be connected between this pins and $V_{SS\_LV\_REGCOR}$.	48	70
$V_{SS\_LV\_REGCOR}$	1.2 V decoupling pins for core logic and regulator feedback. Decoupling capacitor must be connected between this pins and $V_{DD\_LV\_REGCOR}$.	49	71
ADC_0/ADC_1 reference and supply voltage. Pins available on 100-pin and 144-pin package.			
$V_{DD\_HV\_ADC0}$(1)	ADC_0 supply and high reference voltage	33	50
$V_{SS\_HV\_ADC0}$	ADC_0 ground and low reference voltage	34	51
$V_{DD\_HV\_ADC1}$	ADC_1 supply and high reference voltage	39	56
$V_{SS\_HV\_ADC1}$	ADC_1 ground and low reference voltage	40	57
Power supply pins (3.3 V or 5.0 V). All pins available on 144-pin package. Five pairs (V_{DD}; V_{SS}) available on 100-pin package.			
$V_{DD\_HV\_IO0}$(2)	Input/Output supply voltage	—	6
$V_{SS\_HV\_IO0}$(2)	Input/Output ground	—	7
$V_{DD\_HV\_IO1}$	Input/Output supply voltage	13	21
$V_{SS\_HV\_IO1}$	Input/Output ground	14	22
$V_{DD\_HV\_IO2}$	Input/Output supply voltage	63	91
$V_{SS\_HV\_IO2}$	Input/Output ground	62	90
$V_{DD\_HV\_IO3}$	Input/Output supply voltage	**87**	126
$V_{SS\_HV\_IO3}$	Input/Output ground	**88**	127
$V_{DD\_HV\_FL}$	Code and data flash supply voltage	69	97
$V_{SS\_HV\_FL}$	Code and data flash supply ground	68	96
$V_{DD\_HV\_OSC}$	Crystal oscillator amplifier supply voltage	16	27
$V_{SS\_HV\_OSC}$	Crystal oscillator amplifier ground	17	28
Power supply pins (1.2 V). All pins available on 100-pin and 144-pin package.			
$V_{DD\_LV\_COR0}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	12	18
$V_{SS\_LV\_COR0}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR}$ pin.	11	17

Table 6. Supply pins(Continued)

Supply		Pin	
Symbol	**Description**	**100-pin**	**144-pin**
$V_{DD\_LV\_COR1}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	65	93
$V_{SS\_LV\_COR1}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR}$ pin.	66	94
$V_{DD\_LV\_COR2}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{SS\_LV\_COR}$ pin.	92	131
$V_{SS\_LV\_COR2}$	1.2 V Decoupling pins for core logic. Decoupling capacitor must be connected between these pins and the nearest $V_{DD\_LV\_COR}$ pin.	93	132
$V_{DD\_LV\_COR3}$	1.2 V Decoupling pins for on-chip PLL modules. Decoupling capacitor must be connected between this pin and $V_{SS\_LV\_COR3}$.	25	36
$V_{SS\_LV\_COR3}$	1.2 V Decoupling pins for on-chip PLL modules. Decoupling capacitor must be connected between this pin and $V_{DD\_LV\_COR3}$.	24	35

1. Analog supply/ground and high/low reference lines are internally physically separate, but are shorted via a double-bonding connection on $V_{DD\_HV\_ADCx}$/$V_{SS\_HV\_ADCx}$ pins.
2. Not available on 100-pin package.

3.3.2 System pins

Table 6 and *Table 7* contain information on pin functions for the SPC560P44Lx, SPC560P50Lx devices. The pins listed in *Table 7* are single-function pins. The pins shown in *Table 8* are multi-function pins, programmable via their respective Pad Configuration Register (PCR) values.

Table 7. System pins

Symbol	Description	Direction	Pad speed(1)		Pin	
			SRC = 0	SRC = 1	100-pin	144-pin
Dedicated pins. Available on 100-pin and 144-pin package.						
MDO[0]	Nexus Message Data Output—line 0	Output only	Fast		—	9
NMI	Non-Maskable Interrupt	Input only	Slow	—	1	1
XTAL	Analog output of the oscillator amplifier circuit; needs to be grounded if oscillator is used in bypass mode	—	—	—	18	29

Table 7. System pins(Continued)

Symbol	Description	Direction	Pad speed(1)		Pin	
			SRC = 0	SRC = 1	100-pin	144-pin
EXTAL	– Analog input of oscillator amplifier circuit, when oscillator not in bypass mode – Analog input for clock generator when oscillator in bypass mode	—	—	—	19	30
TMS	JTAG state machine control	Input only	Slow	Fast	59	87
TCK	JTAG clock	Input only	Slow	—	60	88
TDI	Test Data In	Input only	Slow	Medium	58	86
TDO	Test Data Out	Output only	Slow	Fast	61	89
Reset pin, available on 100-pin and 144-pin package.						
RESET	Bidirectional reset with Schmitt trigger characteristics and noise filter	Bidirectional	Medium	—	20	31
Test pin, available on 100-pin and 144-pin package.						
VPP_TEST	Pin for testing purpose only. To be tied to ground in normal operating mode.	—	—	—	74	107

1. SCR values refer to the value assigned to the Slew Rate Control bits of the pad configuration register.

3.3.3 Pin muxing

Table 8 defines the pin list and muxing for the SPC560P44Lx, SPC560P50Lx devices.

Each row of *Table 8* shows all the possible ways of configuring each pin, via alternate functions. The default function assigned to each pin after reset is the ALT0 function.

SPC560P44Lx, SPC560P50Lx devices provide four main I/O pad types, depending on the associated functions:

- *Slow pads* are the most common, providing a compromise between transition time and low electromagnetic emission.
- *Medium pads* provide fast enough transition for serial communication channels with controlled current to reduce electromagnetic emission.
- *Fast pads* provide maximum speed. They are used for improved NEXUS debugging capability.
- *Symmetric pads* are designed to meet FlexRay requirements.

Medium and Fast pads can use slow configuration to reduce electromagnetic emission, at the cost of reducing AC performance. For more information, see the datasheet's "Pad AC Specifications" section.

Table 8. Pin muxing

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
Port A (16-bit)									
A[0]	PCR[0]	ALT0 ALT1 ALT2 ALT3 —	GPIO[0] ETC[0] SCK F[0] EIRQ[0]	SIUL eTimer_0 DSPI_2 FCU_0 SIUL	I/O I/O O O I	Slow	Medium	51	73
A[1]	PCR[1]	ALT0 ALT1 ALT2 ALT3 —	GPIO[1] ETC[1] SOUT F[1] EIRQ[1]	SIUL eTimer_0 DSPI_2 FCU_0 SIUL	I/O I/O O O I	Slow	Medium	52	74
A[2] [6]	PCR[2]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[2] ETC[2] — A[3] SIN ABS[0] EIRQ[2]	SIUL eTimer_0 — FlexPWM_0 DSPI_2 MC_RGM SIUL	I/O I/O — O I I I	Slow	Medium	57	84
A[3][6]	PCR[3]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[3] ETC[3] CS0 B[3] ABS[2] EIRQ[3]	SIUL eTimer_0 DSPI_2 FlexPWM_0 MC_RGM SIUL	I/O I/O I/O O I I	Slow	Medium	64	92
A[4][6]	PCR[4]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[4] ETC[0] CS1 ETC[4] FAB EIRQ[4]	SIUL eTimer_1 DSPI_2 eTimer_0 MC_RGM SIUL	I/O I/O O I/O I I	Slow	Medium	75	108
A[5]	PCR[5]	ALT0 ALT1 ALT2 ALT3 —	GPIO[5] CS0 ETC[5] CS7 EIRQ[5]	SIUL DSPI_1 eTimer_1 DSPI_0 SIUL	I/O I/O I/O O I	Slow	Medium	8	14
A[6]	PCR[6]	ALT0 ALT1 ALT2 ALT3 —	GPIO[6] SCK — — EIRQ[6]	SIUL DSPI_1 — — SIUL	I/O I/O — — I	Slow	Medium	2	2
A[7]	PCR[7]	ALT0 ALT1 ALT2 ALT3 —	GPIO[7] SOUT — — EIRQ[7]	SIUL DSPI_1 — — SIUL	I/O O — — I	Slow	Medium	4	10

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
A[8]	PCR[8]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[8] — — — SIN EIRQ[8]	SIUL — — — DSPI_1 SIUL	I/O — — — I I	Slow	Medium	6	12
A[9]	PCR[9]	ALT0 ALT1 ALT2 ALT3 —	GPIO[9] CS1 — B[3] FAULT[0]	SIUL DSPI_2 — FlexPWM_0 FlexPWM_0	I/O O — O I	Slow	Medium	94	134
A[10]	PCR[10]	ALT0 ALT1 ALT2 ALT3 —	GPIO[10] CS0 B[0] X[2] EIRQ[9]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O I/O O I/O I	Slow	Medium	81	118
A[11]	PCR[11]	ALT0 ALT1 ALT2 ALT3 —	GPIO[11] SCK A[0] A[2] EIRQ[10]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O I/O O O I	Slow	Medium	82	120
A[12]	PCR[12]	ALT0 ALT1 ALT2 ALT3 —	GPIO[12] SOUT A[2] B[2] EIRQ[11]	SIUL DSPI_2 FlexPWM_0 FlexPWM_0 SIUL	I/O O O O I	Slow	Medium	83	122
A[13]	PCR[13]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[13] — B[2] — SIN FAULT[0] EIRQ[12]	SIUL — FlexPWM_0 — DSPI_2 FlexPWM_0 SIUL	I/O — O — I I I	Slow	Medium	95	136
A[14]	PCR[14]	ALT0 ALT1 ALT2 ALT3 —	GPIO[14] TXD ETC[4] — EIRQ[13]	SIUL Safety Port_0 eTimer_1 — SIUL	I/O O I/O — I	Slow	Medium	99	143
A[15]	PCR[15]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[15] — ETC[5] — RXD EIRQ[14]	SIUL — eTimer_1 — Safety Port_0 SIUL	I/O — I/O — I I	Slow	Medium	100	144

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
Port B (16-bit)									
B[0]	PCR[16]	ALT0 ALT1 ALT2 ALT3 —	GPIO[16] TXD ETC[2] DEBUG[0] EIRQ[15]	SIUL FlexCAN_0 eTimer_1 SSCM SIUL	I/O O I/O — I	Slow	Medium	76	109
B[1]	PCR[17]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[17] — ETC[3] DEBUG[1] RXD EIRQ[16]	SIUL — eTimer_1 SSCM FlexCAN_0 SIUL	I/O — I/O — I I	Slow	Medium	77	110
B[2]	PCR[18]	ALT0 ALT1 ALT2 ALT3 —	GPIO[18] TXD — DEBUG[2] EIRQ[17]	SIUL LIN_0 — SSCM SIUL	I/O O — — I	Slow	Medium	79	114
B[3]	PCR[19]	ALT0 ALT1 ALT2 ALT3 —	GPIO[19] — — DEBUG[3] RXD	SIUL — — SSCM LIN_0	I/O — — — I	Slow	Medium	80	116
B[6]	PCR[22]	ALT0 ALT1 ALT2 ALT3 —	GPIO[22] CLKOUT CS2 — EIRQ[18]	SIUL MC_CGL DSPI_2 — SIUL	I/O O O — I	Slow	Medium	96	138
B[7]	PCR[23]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[23] — — — AN[0] RXD	SIUL — — — ADC_0 LIN_0	Input only	—	—	29	43
B[8]	PCR[24]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[24] — — — AN[1] ETC[5]	SIUL — — — ADC_0 eTimer_0	Input only	—	—	31	47
B[9]	PCR[25]	ALT0 ALT1 ALT2 ALT3 —	GPIO[25] — — — AN[11]	SIUL — — — ADC_0 / ADC_1	Input only	—	—	35	52

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
B[10]	PCR[26]	ALT0 ALT1 ALT2 ALT3 —	GPIO[26] — — — AN[12]	SIUL — — — ADC_0 / ADC_1	Input only	—	—	36	53
B[11]	PCR[27]	ALT0 ALT1 ALT2 ALT3 —	GPIO[27] — — — AN[13]	SIUL — — — ADC_0 / ADC_1	Input only	—	—	37	54
B[12]	PCR[28]	ALT0 ALT1 ALT2 ALT3 —	GPIO[28] — — — AN[14]	SIUL — — — ADC_0 / ADC_1	Input only	—	—	38	55
B[13]	PCR[29]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[29] — — — AN[0] RXD	SIUL — — — ADC_1 LIN_1	Input only	—	—	42	60
B[14]	PCR[30]	ALT0 ALT1 ALT2 ALT3 — — —	GPIO[30] — — — AN[1] ETC[4] EIRQ[19]	SIUL — — — ADC_1 eTimer_0 SIUL	Input only	—	—	44	64
B[15]	PCR[31]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[31] — — — AN[2] EIRQ[20]	SIUL — — — ADC_1 SIUL	Input only	—	—	43	62
Port C (16-bit)									
C[0]	PCR[32]	ALT0 ALT1 ALT2 ALT3 —	GPIO[32] — — — AN[3]	SIUL — — — ADC_1	Input only	—	—	45	66
C[1]	PCR[33]	ALT0 ALT1 ALT2 ALT3 —	GPIO[33] — — — AN[2]	SIUL — — — ADC_0	Input only	—	—	28	41

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
C[2]	PCR[34]	ALT0 ALT1 ALT2 ALT3 —	GPIO[34] — — — AN[3]	SIUL — — — ADC_0	Input only	—	—	30	45
C[3]	PCR[35]	ALT0 ALT1 ALT2 ALT3 —	GPIO[35] CS1 ETC[4] TXD EIRQ[21]	SIUL DSPI_0 eTimer_1 LIN_1 SIUL	I/O O I/O O I	Slow	Medium	10	16
C[4]	PCR[36]	ALT0 ALT1 ALT2 ALT3 —	GPIO[36] CS0 X[1] DEBUG[4] EIRQ[22]	SIUL DSPI_0 FlexPWM_0 SSCM SIUL	I/O I/O I/O — I	Slow	Medium	5	11
C[5]	PCR[37]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[37] SCK — DEBUG[5] FAULT[3] EIRQ[23]	SIUL DSPI_0 — SSCM FlexPWM_0 SIUL	I/O I/O — — I I	Slow	Medium	7	13
C[6]	PCR[38]	ALT0 ALT1 ALT2 ALT3 —	GPIO[38] SOUT B[1] DEBUG[6] EIRQ[24]	SIUL DSPI_0 FlexPWM_0 SSCM SIUL	I/O I/O O — I	Slow	Medium	98	142
C[7]	PCR[39]	ALT0 ALT1 ALT2 ALT3 —	GPIO[39] — A[1] DEBUG[7] SIN	SIUL — FlexPWM_0 SSCM DSPI_0	I/O — O — I	Slow	Medium	9	15
C[8]	PCR[40]	ALT0 ALT1 ALT2 ALT3 —	GPIO[40] CS1 — CS6 FAULT[2]	SIUL DSPI_1 — DSPI_0 FlexPWM_0	I/O O — O I	Slow	Medium	91	130
C[9]	PCR[41]	ALT0 ALT1 ALT2 ALT3 —	GPIO[41] CS3 — X[3] FAULT[2]	SIUL DSPI_2 — FlexPWM_0 FlexPWM_0	I/O O — I/O I	Slow	Medium	84	123

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
C[10]	PCR[42]	ALT0 ALT1 ALT2 ALT3 —	GPIO[42] CS2 — A[3] FAULT[1]	SIUL DSPI_2 — FlexPWM_0 FlexPWM_0	I/O O — O I	Slow	Medium	78	111
C[11]	PCR[43]	ALT0 ALT1 ALT2 ALT3	GPIO[43] ETC[4] CS2 CS0	SIUL eTimer_0 DSPI_2 DSPI_3	I/O I/O O I/O	Slow	Medium	55	80
C[12]	PCR[44]	ALT0 ALT1 ALT2 ALT3	GPIO[44] ETC[5] CS3 CS1	SIUL eTimer_0 DSPI_2 DSPI_3	I/O I/O O O	Slow	Medium	56	82
C[13]	PCR[45]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[45] ETC[1] — — EXT_IN EXT_SYNC	SIUL eTimer_1 — — CTU_0 FlexPWM_0	I/O I/O — — I I	Slow	Medium	71	101
C[14]	PCR[46]	ALT0 ALT1 ALT2 ALT3	GPIO[46] ETC[2] EXT_TGR —	SIUL eTimer_1 CTU_0 —	I/O I/O O —	Slow	Medium	72	103
C[15]	PCR[47]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[47] CA_TR_EN ETC[0] A[1] EXT_IN EXT_SYNC	SIUL FlexRay_0 eTimer_1 FlexPWM_0 CTU_0 FlexPWM_0	I/O O I/O O I I	Slow	Symmetric	85	124
Port D (16-bit)									
D[0]	PCR[48]	ALT0 ALT1 ALT2 ALT3	GPIO[48] CA_TX ETC[1] B[1]	SIUL FlexRay_0 eTimer_1 FlexPWM_0	I/O O I/O O	Slow	Symmetric	86	125
D[1]	PCR[49]	ALT0 ALT1 ALT2 ALT3 —	GPIO[49] — ETC[2] EXT_TRG CA_RX	SIUL — eTimer_1 CTU_0 FlexRay_0	I/O — I/O O I	Slow	Medium	3	3
D[2]	PCR[50]	ALT0 ALT1 ALT2 ALT3 —	GPIO[50] — ETC[3] X[3] CB_RX	SIUL — eTimer_1 FlexPWM_0 FlexRay_0	I/O — I/O I/O I	Slow	Medium	97	140

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
D[3]	PCR[51]	ALT0 ALT1 ALT2 ALT3	GPIO[51] CB_TX ETC[4] A[3]	SIUL FlexRay_0 eTimer_1 FlexPWM_0	I/O O I/O O	Slow	Symmetric	89	128
D[4]	PCR[52]	ALT0 ALT1 ALT2 ALT3	GPIO[52] CB_TR_EN ETC[5] B[3]	SIUL FlexRay_0 eTimer_1 FlexPWM_0	I/O O I/O O	Slow	Symmetric	90	129
D[5]	PCR[53]	ALT0 ALT1 ALT2 ALT3	GPIO[53] CS3 F[0] SOUT	SIUL DSPI_0 FCU_0 DSPI_3	I/O O O O	Slow	Medium	22	33
D[6]	PCR[54]	ALT0 ALT1 ALT2 ALT3 —	GPIO[54] CS2 SCK — FAULT[1]	SIUL DSPI_0 DSPI_3 — FlexPWM_0	I/O O I/O — I	Slow	Medium	23	34
D[7]	PCR[55]	ALT0 ALT1 ALT2 ALT3 —	GPIO[55] CS3 F[1] CS4 SIN	SIUL DSPI_1 FCU_0 DSPI_0 DSPI_3	I/O O O O I	Slow	Medium	26	37
D[8]	PCR[56]	ALT0 ALT1 ALT2 ALT3 —	GPIO[56] CS2 — CS5 FAULT[3]	SIUL DSPI_1 — DSPI_0 FlexPWM_0	I/O O — O I	Slow	Medium	21	32
D[9]	PCR[57]	ALT0 ALT1 ALT2 ALT3	GPIO[57] X[0] TXD —	SIUL FlexPWM_0 LIN_1 —	I/O I/O O —	Slow	Medium	15	26
D[10]	PCR[58]	ALT0 ALT1 ALT2 ALT3	GPIO[58] A[0] CS0 —	SIUL FlexPWM_0 DSPI_3 —	I/O O I/O —	Slow	Medium	53	76
D[11]	PCR[59]	ALT0 ALT1 ALT2 ALT3	GPIO[59] B[0] CS1 SCK	SIUL FlexPWM_0 DSPI_3 DSPI_3	I/O O O I/O	Slow	Medium	54	78
D[12]	PCR[60]	ALT0 ALT1 ALT2 ALT3 —	GPIO[60] X[1] — — RXD	SIUL FlexPWM_0 — — LIN_1	I/O I/O — — I	Slow	Medium	70	99

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
D[13]	PCR[61]	ALT0 ALT1 ALT2 ALT3	GPIO[61] A[1] CS2 SOUT	SIUL FlexPWM_0 DSPI_3 DSPI_3	I/O O O O	Slow	Medium	67	95
D[14]	PCR[62]	ALT0 ALT1 ALT2 ALT3 —	GPIO[62] B[1] CS3 — SIN	SIUL FlexPWM_0 DSPI_3 — DSPI_3	I/O O O — I	Slow	Medium	73	105
D[15]	PCR[63]	ALT0 ALT1 ALT2 ALT3 —	GPIO[63] — — — AN[4]	SIUL — — — ADC_1	Input only	—	—	41	58
Port E(16-bit)									
E[0]	PCR[64]	ALT0 ALT1 ALT2 ALT3 —	GPIO[64] — — — AN[5]	SIUL — — — ADC_1	Input only	—	—	46	68
E[1]	PCR[65]	ALT0 ALT1 ALT2 ALT3 —	GPIO[65] — — — AN[4]	SIUL — — — ADC_0	Input only	—	—	27	39
E[2]	PCR[66]	ALT0 ALT1 ALT2 ALT3 —	GPIO[66] — — — AN[5]	SIUL — — — ADC_0	Input only	—	—	32	49
E[3]	PCR[67]	ALT0 ALT1 ALT2 ALT3 —	GPIO[67] — — — AN[6]	SIUL — — — ADC_0	Input only	—	—	—	40
E[4]	PCR[68]	ALT0 ALT1 ALT2 ALT3 —	GPIO[68] — — — AN[7]	SIUL — — — ADC_0	Input only	—	—	—	42
E[5]	PCR[69]	ALT0 ALT1 ALT2 ALT3 —	GPIO[69] — — — AN[8]	SIUL — — — ADC_0	Input only	—	—	—	44

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
E[6]	PCR[70]	ALT0 ALT1 ALT2 ALT3 —	GPIO[70] — — — AN[9]	SIUL — — — ADC_0	Input only	—	—	—	46
E[7]	PCR[71]	ALT0 ALT1 ALT2 ALT3 —	GPIO[71] — — — AN[10]	SIUL — — — ADC_0	Input only	—	—	—	48
E[8]	PCR[72]	ALT0 ALT1 ALT2 ALT3 —	GPIO[72] — — — AN[6]	SIUL — — — ADC_1	Input only	—	—	—	59
E[9]	PCR[73]	ALT0 ALT1 ALT2 ALT3 —	GPIO[73] — — — AN[7]	SIUL — — — ADC_1	Input only	—	—	—	61
E[10]	PCR[74]	ALT0 ALT1 ALT2 ALT3 —	GPIO[74] — — — AN[8]	SIUL — — — ADC_1	Input only	—	—	—	63
E[11]	PCR[75]	ALT0 ALT1 ALT2 ALT3 —	GPIO[75] — — — AN[9]	SIUL — — — ADC_1	Input only	—	—	—	65
E[12]	PCR[76]	ALT0 ALT1 ALT2 ALT3 —	GPIO[76] — — — AN[10]	SIUL — — — ADC_1	Input only	—	—	—	67
E[13]	PCR[77]	ALT0 ALT1 ALT2 ALT3 —	GPIO[77] SCK — — EIRQ[25]	SIUL DSPI_3 — — SIUL	I/O I/O — — I	Slow	Medium	—	117
E[14]	PCR[78]	ALT0 ALT1 ALT2 ALT3 —	GPIO[78] SOUT — — EIRQ[26]	SIUL DSPI_3 — — SIUL	I/O O — — I	Slow	Medium	—	119

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
E[15]	PCR[79]	ALT0 ALT1 ALT2 ALT3 — —	GPIO[79] — — — SIN EIRQ[27]	SIUL — — — DSPI_3 SIUL	I/O — — — I I	Slow	Medium	—	121
Port F (16-bit)									
F[0]	PCR[80]	ALT0 ALT1 ALT2 ALT3 —	GPIO[80] DBG0 CS3 — EIRQ[28]	SIUL FlexRay_0 DSPI_3 — SIUL	I/O O O — I	Slow	Medium	—	133
F[1]	PCR[81]	ALT0 ALT1 ALT2 ALT3 —	GPIO[81] DBG1 CS2 — EIRQ[29]	SIUL FlexRay_0 DSPI_3 — SIUL	I/O O O — I	Slow	Medium	—	135
F[2]	PCR[82]	ALT0 ALT1 ALT2 ALT3	GPIO[82] DBG2 CS1 —	SIUL FlexRay_0 DSPI_3 —	I/O O O —	Slow	Medium	—	137
F[3]	PCR[83]	ALT0 ALT1 ALT2 ALT3	GPIO[83] DBG3 CS0 —	SIUL FlexRay_0 DSPI_3 —	I/O O I/O —	Slow	Medium	—	139
F[4]	PCR[84]	ALT0 ALT1 ALT2 ALT3	GPIO[84] MDO[3] — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	4
F[5]	PCR[85]	ALT0 ALT1 ALT2 ALT3	GPIO[85] MDO[2] — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	5
F[6]	PCR[86]	ALT0 ALT1 ALT2 ALT3	GPIO[86] MDO[1] — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	8
F[7]	PCR[87]	ALT0 ALT1 ALT2 ALT3	GPIO[87] MCKO — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	19

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
F[8]	PCR[88]	ALT0 ALT1 ALT2 ALT3	GPIO[88] MSEO1 — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	20
F[9]	PCR[89]	ALT0 ALT1 ALT2 ALT3	GPIO[89] MSEO0 — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	23
F[10] [7]	PCR[90]	ALT0 ALT1 ALT2 ALT3	GPIO[90] EVTO — —	SIUL NEXUS_0 — —	I/O O — —	Slow	Fast	—	24
F[11]	PCR[91]	ALT0 ALT1 ALT2 ALT3 —	GPIO[91] — — — EVTI	SIUL — — — NEXUS_0	I/O — — — I	Slow	Medium	—	25
F[12]	PCR[92]	ALT0 ALT1 ALT2 ALT3	GPIO[92] ETC[3] — —	SIUL eTimer_1 — —	I/O I/O — —	Slow	Medium	—	106
F[13]	PCR[93]	ALT0 ALT1 ALT2 ALT3	GPIO[93] ETC[4] — —	SIUL eTimer_1 — —	I/O I/O — —	Slow	Medium	—	112
F[14]	PCR[94]	ALT0 ALT1 ALT2 ALT3	GPIO[94] TXD — —	SIUL LIN_1 — —	I/O O — —	Slow	Medium	—	115
F[15]	PCR[95]	ALT0 ALT1 ALT2 ALT3 —	GPIO[95] — — — RXD	SIUL — — — LIN_1	I/O — — — I	Slow	Medium	—	113
Port G (12-bit)									
G[0]	PCR[96]	ALT0 ALT1 ALT2 ALT3 —	GPIO[96] F[0] — — EIRQ[30]	SIUL FCU_0 — — SIUL	I/O O — — I	Slow	Medium	—	38

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
G[1]	PCR[97]	ALT0 ALT1 ALT2 ALT3 —	GPIO[97] F[1] — — EIRQ[31]	SIUL FCU_0 — — SIUL	I/O O — — I	Slow	Medium	—	141
G[2]	PCR[98]	ALT0 ALT1 ALT2 ALT3	GPIO[98] X[2] — —	SIUL FlexPWM_0 — —	I/O I/O — —	Slow	Medium	—	102
G[3]	PCR[99]	ALT0 ALT1 ALT2 ALT3	GPIO[99] A[2] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	104
G[4]	PCR[100]	ALT0 ALT1 ALT2 ALT3	GPIO[100] B[2] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	100
G[5]	PCR[101]	ALT0 ALT1 ALT2 ALT3	GPIO[101] X[3] — —	SIUL FlexPWM_0 — —	I/O I/O — —	Slow	Medium	—	85
G[6]	PCR[102]	ALT0 ALT1 ALT2 ALT3	GPIO[102] A[3] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	98
G[7]	PCR[103]	ALT0 ALT1 ALT2 ALT3	GPIO[103] B[3] — —	SIUL FlexPWM_0 — —	I/O O — —	Slow	Medium	—	83

Table 8. Pin muxing(Continued)

Port pin	Pad configuration register (PCR)	Alternate function[1], [2]	Functions	Peripheral[3]	I/O direction [4]	Pad speed[5]		Pin No.	
						SRC = 0	SRC = 1	100-pin	144-pin
G[8]	PCR[104]	ALT0 ALT1 ALT2 ALT3 —	GPIO[104] — — — FAULT[0]	SIUL — — — FlexPWM_0	I/O — — — I	Slow	Medium	—	81
G[9]	PCR[105]	ALT0 ALT1 ALT2 ALT3 —	GPIO[105] — — — FAULT[1]	SIUL — — — FlexPWM_0	I/O — — — I	Slow	Medium	—	79
G[10]	PCR[106]	ALT0 ALT1 ALT2 ALT3 —	GPIO[106] — — — FAULT[2]	SIUL — — — FlexPWM_0	I/O — — — I	Slow	Medium	—	77
G[11]	PCR[107]	ALT0 ALT1 ALT2 ALT3 —	GPIO[107] — — — FAULT[3]	SIUL — — — FlexPWM_0	I/O — — — I	Slow	Medium	—	75

1. ALT0 is the primary (default) function for each port after reset.
2. Alternate functions are chosen by setting the values of the PCR[PA] bitfields inside the SIU module. PCR[PA] = 00 → ALT0; PCR[PA] = 01 → ALT1; PCR[PA] = 10 → ALT2; PCR[PA] = 11 → ALT3. This is intended to select the output functions; to use one of the input-only functions, the PCR[IBE] bit must be written to '1', regardless of the values selected in the PCR[PA] bitfields. For this reason, the value corresponding to an input only function is reported as "—".
3. Module included on the MCU.
4. Multiple inputs are routed to all respective modules internally. The input of some modules must be configured by setting the values of the PSMI[PADSEL*x*] bitfields inside the SIUL module.
5. Programmable via the SRC (Slew Rate Control) bits in the respective Pad Configuration Register.
6. Weak pull down during reset.
7. The port pin F[10] is internally connect to EVTO signal and it is not possible to use it as GPIO pin when the debugger is connected.

3.4 CTU / ADCs / FlexPWM / eTimers connections

Figure 7 shows the interconnections between the CTU, ADCs, FlexPWM, and eTimers.

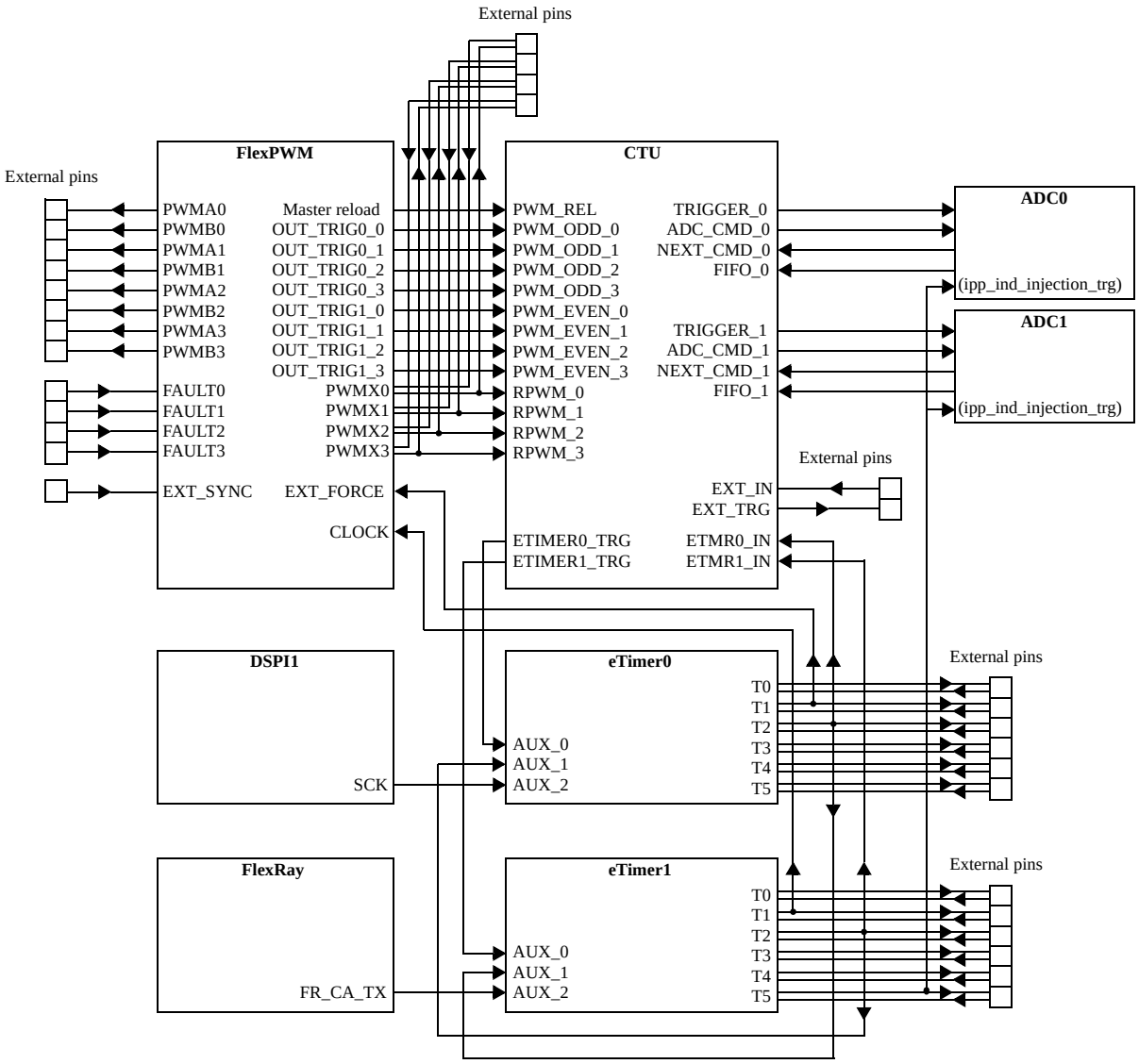


Figure 7. CTU / ADCs / FlexPWM / eTimers connections

Table 9. CTU / ADCs / FlexPWM / eTimers connections

Source module (Signal)	Target module (Signal)	Comment
PWM (Master Reload)	CTU (PWM Reload)	From PWM sub-module 0
PWM (OUT_TRIG0_0)	CTU (PWM_ODD_0)	OUT_TRIG0 sub-module 0
PWM (OUT_TRIG1_0)	CTU (PWM_EVEN_0)	OUT_TRIG1 sub-module 0
PWM (PWMX0)	CTU (PWM_REAL_0)	—
PWM (OUT_TRIG0_1)	CTU (PWM_ODD_1)	OUT_TRIG0 sub-module 1
PWM (OUT_TRIG1_1)	CTU (PWM_EVEN_1)	OUT_TRIG1 sub-module 1

Table 9. CTU / ADCs / FlexPWM / eTimers connections(Continued)

Source module (Signal)	Target module (Signal)	Comment
PWM (PWMX1)	CTU (PWM_REAL_1)	—
PWM (OUT_TRIG0_2)	CTU (PWM_ODD_2)	OUT_TRIG0 sub-module 2
PWM (OUT_TRIG1_2)	CTU (PWM_EVEN_2)	OUT_TRIG1 sub-module 2
PWM (PWMX2)	CTU (PWM_REAL_2)	—
PWM (OUT_TRIG0_3)	CTU (PWM_ODD_3)	OUT_TRIG0 sub-module 3
PWM (OUT_TRIG1_3)	CTU (PWM_EVEN_3)	OUT_TRIG1 sub-module 3
PWM (PWMX3)	CTU (PWM_REAL_3)	—
PWM (PWMA0)	SIU lite	—
PWM (PWMB0)	SIU lite	—
PWM (PWMX1)	SIU lite	—
PWM (PWMA1)	SIU lite	—
PWM (PWMB1)	SIU lite	—
PWM (PWMX2)	SIU lite	—
PWM (PWMA2)	SIU lite	—
PWM (PWMB2)	SIU lite	—
PWM (PWMX3)	SIU lite	—
PWM (PWMA3)	SIU lite	—
PWM (PWMB3)	SIU lite	—
PWM (PWMX3)	SIU lite	—
eTimer_0 (T1)	PWM (EXT_FORCE)	—
eTimer_0 (T2)	CTU (ETMR0_IN)	—
eTimer_0(T2)	eTimer_1(AUX_1)	—
eTimer_1(T1)	PWM(CLOCK)	—
eTimer_1(T2)	CTU(ETMR1_IN)	—
eTimer_1(T2)	eTimer_0(AUX_1)	—
eTimer_1(T5)	eTimer_1(T5)	A/D conversion injection request signal (for non CTU mode of operation)
CTU (ETIMER0_TRG)	eTimer_0 (AUX_0)	—
CTU(ETIMER1_TRG)	eTimer_1(AUX_0)	—
CTU (TRIGGER_0)	ADC_0	—
CTU (TRIGGER_1)	ADC_1	—
CTU (ADC_CMD_0)	ADC_0	16-bit signal
CTU (ADC_CMD_1)	ADC_1	16-bit signal
CTU (EXT_TGR)	SIU lite	—

Table 9. CTU / ADCs / FlexPWM / eTimers connections(Continued)

Source module (Signal)	Target module (Signal)	Comment
ADC_0 (EOC)	CTU (NEXT_CMD_0)	End Of Conversion should be used as next command request signal
ADC_1 (EOC)	CTU (NEXT_CMD_1)	End Of Conversion should be used as next command request signal
ADC_0	CTU (FIFO_0)	18-bit signal
ADC_1	CTU (FIFO_1)	18-bit signal
SIU lite	CTU (EXT_IN)	The same GPIO pin as used for CTU (EXT_IN) and the PWM (EXT_SYNC)
SIU lite	PWM (EXT_SYNC)	The same GPIO pin as used for CTU (EXT_IN) and the PWM (EXT_SYNC)
SIU lite	PWM (FAULT0)	—
SIU lite	PWM (FAULT1)	—
SIU lite	PWM (FAULT2)	—
SIU lite	PWM (FAULT3)	—
DSPI_1 (SCK)	eTimer_0 (AUX_2)	—
FlexRAY(FR_CA_TX)	eTimer_1(AUX_2)	—

4 Clock Description

This chapter describes the clock architectural implementation for SPC560P44Lx, SPC560P50Lx.

The following clock related modules are implemented on the SPC560P44Lx, SPC560P50Lx:

- Clock, Reset, and Mode Handling
  - Clock Generation Module (CGM) (see *Chapter 5: Clock Generation Module (CGM)*)
  - Reset Generation Module (RGM) (see *Chapter 8: Reset Generation Module (RGM)*)
  - Mode Entry Module (ME) (see *Chapter 6: Mode Entry Module (ME)*)
- High Frequency Oscillator (XOSC) (see *Section 4.8: XOSC external crystal oscillator*)
- High Frequency RC Oscillator (IRC) (see *Section 4.7: IRC 16 MHz internal RC oscillator (RC_CTL)*)
- FMPLL (FMPLL_0) (see *Section 4.9: Frequency Modulated Phase Locked Loop (FMPLL)*)
- FMPLL (FMPLL_1) (see *Section 4.9: Frequency Modulated Phase Locked Loop (FMPLL)*)
- CMU (CMU_0) (see *Section 4.10: Clock Monitor Unit (CMU)*)
- CMU (CMU_1) (see *Section 4.10: Clock Monitor Unit (CMU)*)
- Periodic Interrupt Timer (PIT) (see *Chapter 31: Periodic Interrupt Timer (PIT)*)
- System Timer Module (STM_0) (see *Chapter 32: System Timer Module (STM)*)
- Software Watchdog Timer (SWT_0) (see *Section 28.3: Software Watchdog Timer (SWT)*)

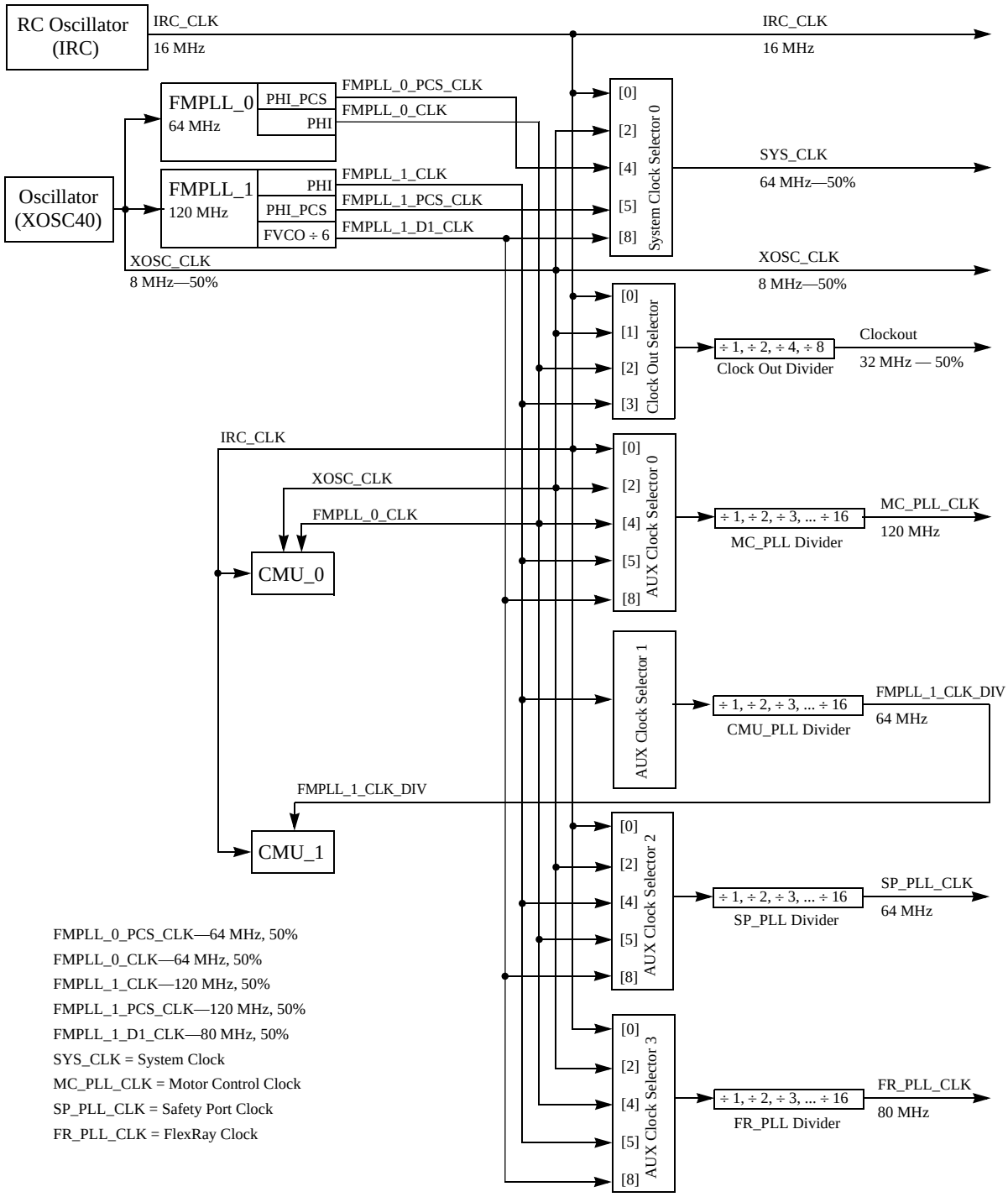
4.1 Clock architecture

The system and peripheral clocks are generated from four sources:

- IRC—internal RC oscillator clock
- XOSC—oscillator clock
- FMPLL_0 clock output
- FMPLL_1 clock output

The clock architecture is shown in *Figure 8*, *Figure 9*, and *Figure 10*.

The frequencies shown in *Figure 8* represent only one possible setting.



NOTE: FlexRay protocol clock does not support IRC and OSC as a clock source.

Figure 8. SPC560P44Lx, SPC560P50Lx system clock generation

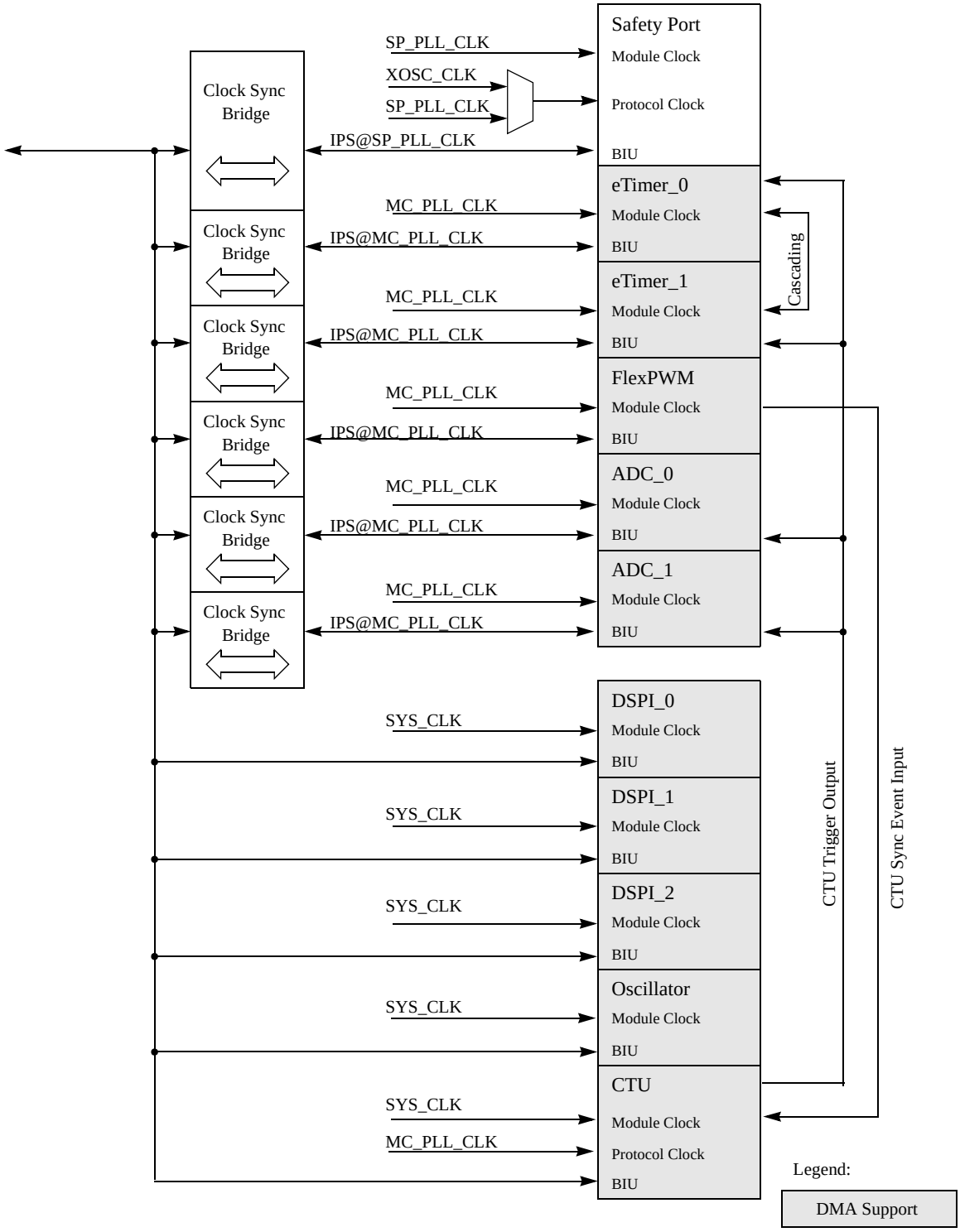


Figure 9. SPC560P44Lx, SPC560P50Lx system clock distribution Part A

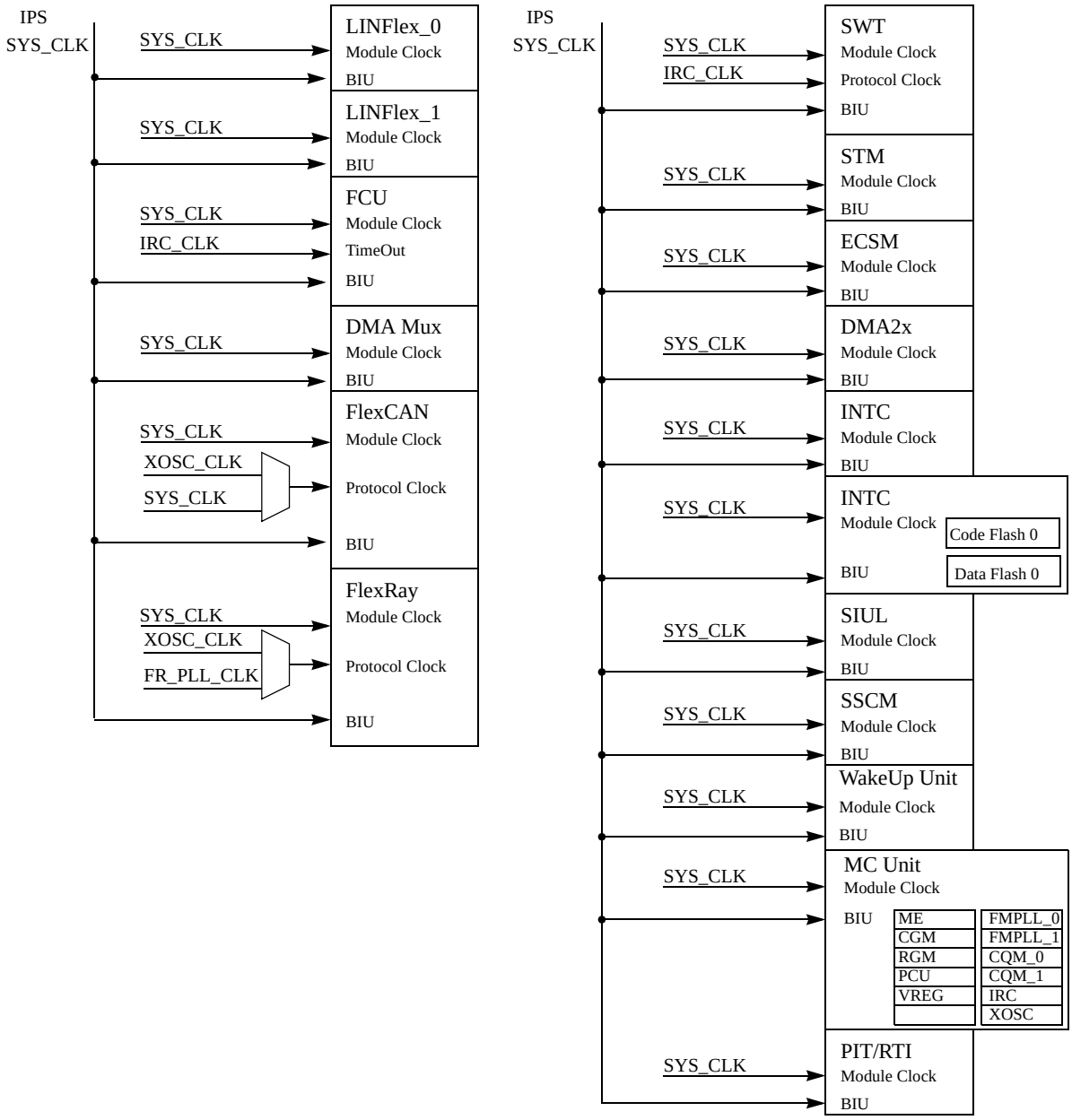


Figure 10. SPC560P44Lx, SPC560P50Lx system clock distribution Part B

4.2 Available clock domains

This section describes the various clock domains available on SPC560P44Lx, SPC560P50Lx.

4.2.1 FMPLL_*n* input reference clock

The input reference clock for both the FMPLL_0 and FMPLL_1 is always the external crystal oscillator clock (XOSC).

4.2.2 Clock selectors

4.2.2.1 System clock selector 0 for SYS_CLK

The system clock selector 0 selects the clock source for the system clock (SYS_CLK) from clock signals:

- Internal RC oscillator clock (IRC)
- Progressive output clock of the first PLL (FMPLL_0)
- Progressive output clock of the second PLL (FMPLL_1)
- Directly from the oscillator clock (XOSC)

Its behavior is configured via software through ME_*x*_MC register of the ME module.

When the standard boot from internal flash is selected via the boot configuration pins, the clock source for the system clock (SYS_CLK) after reset (DRUN mode) is the internal RC oscillator (IRC).

4.2.2.2 Auxiliary clock selector 0 for MC_PLL_CLK divider (Motor Control clock)

The auxiliary clock selector 0 provides the clock source for the MC_PLL divider. The MC_PLL divider generates the clock to drive the Motor Control subsystem. The following clocks can be selected by the auxiliary clock selector 0:

- Internal RC oscillator clock (IRC)
- Progressive output clock of the first PLL (FMPLL_0)
- Progressive output clock of the second PLL (FMPLL_1)
- Directly from the oscillator clock (XOSC)

Its behavior is configured via software through the CGM_AC[0]_SC register of the CGM module.

When the standard boot from internal flash is selected via the boot configuration pins, the auxiliary clock divider 0 selects the IRC as the default clock after reset (DRUN mode).

4.2.2.3 Auxiliary clock selector 1 for CMU_PLL divider (CMU_1 clock)

The auxiliary clock selector 1 provides the clock source for the CMU_PLL divider. The CMU_PLL divider generates the clock to drive the CMU_1. Only the non-progressive output from FMPLL_1 can be selected at auxiliary clock selector 1.

With the possibility of FMPLL_1 generating a clock frequency of 120 MHz, the CMU_PLL divider is required to reduce the frequency of the clock at the CMU_1 to a frequency that is equal to or less than 64 MHz.

The auxiliary clock selector 1 is fed by the non-progressive output of FMPLL_1, CGM_AC[1]_SC register of the CGM module.

The output of the clock selector feeds the CMU_PLL Divider, which is configured through CGM_AC[1]_DC[0].

The user needs to select an appropriate divide ratio between the FMPLL_1 output and the FMPLL_1_CLK_DIV clock to allow for proper operation of CMU_1.

4.2.2.4 Auxiliary clock selector 2 for SP_PLL divider (Safety Port clock)

The auxiliary clock selector 2 provides the clock source for the SP_PLL Divider. The SP_PLL Divider generates the clock to drive the Safety Port. The following clocks can be selected by the auxiliary clock selector 2:

- Internal RC oscillator clock (IRC)
- Progressive output clock of the first PLL (FMPLL_0)
- Progressive output clock of the second PLL (FMPLL_1)
- Directly from the oscillator clock (XOSC).

Its behavior is configured through CGM_AC[2]_SC register of the CGM module.

When the standard boot from internal flash is selected via the boot configuration pins, the auxiliary clock divider 2 selects the clock IRC as the default clock after reset (DRUN mode).

4.2.2.5 Auxiliary Clock Selector 3 for FR_PLL divider (FlexRay clock)

Software uses the clock output selector to select one of the following clock sources:

- Internal RC oscillator clock (IRC)
- Progressive output clock of the first PLL (FMPLL_0)
- Progressive output clock of the second PLL (FMPLL_1)
- Directly from the oscillator clock (XOSC)

The default clock source after reset is the IRC, but the default status is disabled.

The clock output can be enabled or disabled by software through the CGM module (CGM_OC_EN and CGM_OCDS_SC registers).

4.2.3 Auxiliary clock dividers

Four auxiliary clock dividers are implemented to generate the MC_PLL_CLK, the FR_PLL_CLK, the CMU_PLL clock and the SP_PLL_CLK. All four dividers support these divide options: ÷ 1, ÷ 2, ÷ 3, ÷ 4, ÷5 , ÷ 6, ÷ 7, ÷ 8, ÷ 9, ÷ 10, ÷ 11, ÷ 12, ÷ 13, ÷ 14, ÷ 15.

The clock dividers are configured by CGM_AC[x]_DC[y] registers of the CGM module.

4.2.4 External clock divider

The output clock divider provides a nominal 50% duty cycle clock and allows the selected output clock source to be divided with these divide options:

- ÷ 1, ÷ 2, ÷ 4, ÷ 8

4.3 Alternate module clock domains

This section lists the different clock domains for each module. If not otherwise noted, all modules on the SPC560P44Lx, SPC560P50Lx device are clocked on the SYS_CLK.

4.3.1 FlexCAN clock domains

The FlexCAN modules have two distinct software controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be either the system clock (SYS_CLK) or a direct feed from the oscillator pin XOSC_CLK. The logic in the

second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures very low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR. *Figure 420: FlexCAN block diagram* shows the two clock domains in the FlexCAN modules.

Refer to *Chapter 23: FlexCAN* for more information on the FlexCAN modules.

4.3.2 FlexRay clock domains

The FlexRay block has two distinct clock domains. The first clock domain (Module Clock or CHI clock) is always derived from the SYS_CLK clock.

The source for the second clock domain can either be the FR_PLL_CLK clock or the XOSC_CLK clock. The logic in the second clock domain controls the FlexRay interface pins (Protocol Clock or PE clock).

4.3.3 SWT clock domains

The SWT module has two distinct clock domains. The first clock domain (Module Clock) is always supplied from the SYS_CLK. This clock domain includes the register interface.

The source for the second clock domain (Protocol Clock) is always the IRC generated by the internal RC oscillator.

4.3.4 Nexus Message Clock (MCKO)

The Nexus message clock (MCKO) divider can be programmed to divide the system clock by one, two, four or eight based on the MCKO_DIV bit field in the port configuration register (PCR) in the Nexus port controller (NPC). The reset value of the MCKO_DIV selects an MCKO clock frequency one half of the system clock frequency. The MCKO divider is configured by writing to the NPC through the JTAG port. Refer to *Chapter 37: Nexus Development Interface (NDI)* for more information.

4.3.5 Cross Triggering Unit (CTU) clock domains

The CTU module has two distinct clock domains. The first clock domain (Module Clock) is supplied from the SYS_CLK. This clock domain includes the Command Buffer logic.

The source for the second clock domain (Protocol Clock) is the MC_PLL_CLK. The logic in the Protocol Clock domain controls the CTU interface pins to the eTimer modules and the ADC modules.

4.3.6 IPS bus clock sync bridge

The sync bridge module synchronizes all transactions on the PBRIDGE between peripherals clocked from the MC_PLL_CLK and the CPU and eDMA, clocked from SYS_CLK.

For motor control applications, this bridge introduces an access delay of as many as five additional SYS_CLK cycles.

4.3.7 Peripherals behind the IPS bus clock sync bridge

4.3.7.1 FlexPWM clock domain

The FlexPWM module has only one clock domain. The FlexPWM module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

4.3.7.2 eTimer_0 clock domain

The eTimer_0 module has only one clock domain. The eTimer_0 module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

4.3.7.3 eTimer_1 clock domain

The eTimer_1 module has only one clock domain. The eTimer_1 module is clocked from the MC_PLL_CLK. Therefore it is placed behind the IPS bus clock sync bridge.

4.3.7.4 ADC_0 clock domain

The ADC_0 module has only one clock domain. The ADC_0 module is clocked from the MC_PLL_CLK. Therefore, it is placed behind the IPS bus clock sync bridge.

4.3.7.5 ADC_1 clock domain

The ADC_1 module has only one clock domain. The ADC_1 module is clocked from the MC_PLL_CLK. Therefore it is placed behind the IPS bus clock sync bridge.

4.3.7.6 Safety Port clock domains

The Safety Port module has two distinct software-controlled clock domains. The first clock domain (Module Clock) is always supplied from the SP_PLL_CLK. The source for the second clock domain (Protocol Clock) can either be the SP_PLL_CLK or the XOSC_CLK.

The user must ensure that the frequency of the first clock domain (Module Clock) clocked from the MC_PLL_CLK is always the same or greater than the clock selected for the second clock domain (Protocol Clock).

4.4 Clock behavior in STOP and HALT mode

In this section the term “resume” is used to describe the transition from STOP and HALT mode back to a RUN mode.

The SPC560P44Lx, SPC560P50Lx supports the STOP and the HALT modes. These two modes allow to put the device into a power saving mode with the configuration options defined in the ME module.

The following constraints are applied on SPC560P44Lx, SPC560P50Lx to guarantee that in all modes of operation a resume from STOP or HALT mode is always possible without the need to reset:

- STOP and HALT mode:
  - SIUL clock is not gateable
  - SIUL filter for external interrupt capable pins is always clocked with IRC
  - Resume via interrupt that can be generated by any peripheral that clock is not gated
  - Resume via $\overline{\text{NMI}}$ pin is always possible if once enabled after reset (no software configuration that could block resume afterwards)
- STOP mode:
  - IRC can NOT be switched off
  - The System Clock Selector 0 is switched to the IRC and therefore the SYS_CLK is feed by the IRC signal
  - Resume via external interrupt pin is always possible (if not masked)
- HALT mode:
  - The output of the System Clock Selector 0 can only be switched to a running clock input
  - Resume via external interrupt pin is always possible (if not masked) and IRC is not switched off

4.5 Software controlled power management/clock gating

Some of the IP modules on this device support software controlled power management/clock gating whereby the application software can disable the non-memory-mapped portions of the modules by writing to module disable (MDIS) bits in registers within the modules. The memory-mapped portions of the modules are clocked by the system clock when they are being accessed. The NPC can be configured to disable the MCKO signal when there are no Nexus messages pending.

The modules that implemented software controlled power management/clock gating are listed in *Table 10* along with the registers and bits that disable each module. The software controlled clocks are enabled when the MCU comes out of reset.

Table 10. Software controlled power management/clock gating support

Module name	Register name	Bit names
NPC	NPC_PCR	MCKO_EN, MCKO_GT(1)

1. Refer to *Chapter 37: Nexus Development Interface (NDI)*.

4.6 System clock functional safety

This section shows the SPC560P44Lx, SPC560P50Lx modules used to detect clock failures:

- The Clock Monitoring Unit (CMU_0) monitors the clock frequency of the FMPLL_0 and the XOSC signal against the IRC and provides clock out of range information about the monitored clock signals.
- The Clock Monitoring Unit (CMU_1) monitors the clock frequency of the FMPLL_1 signal against the IRC and provides clock out of range information about the monitored clock signal.
- Both PLLs (FMPLL_0 and FMPLL_1) provide a signal that indicates a loss of lock. Each loss of lock signal is sent to the CGM module.

Upon the detection of one of the above mentioned failures, the SPC560P44Lx, SPC560P50Lx device either asserts a reset, generates an interrupt, or sends the device into the SAFE state.

The reaction to each of the clock failures and system parameters (like active clocks and SYS_CLK clock source) that become active in SAFE state are under software control and can be configured in the ME module.

4.7 IRC 16 MHz internal RC oscillator (RC_CTL)

The IRC output frequency can be trimmed using RCTRIM bits. After a power-on reset, the IRC is trimmed using a factory test value stored in test flash memory. However, after a power-on reset the test flash memory value is not visible at RC_CTL[RCTRIM], and this field shows a value of zero. Therefore, be aware that the RC_CTL[RCTRIM] field does not reflect the current trim value until you have written to it. Pay particular attention to this feature when you initiate a read-modify-write operation on RC_CTL, because a RCTRIM value of zero may be unintentionally written back and this may alter the IRC frequency. In this case, you should calibrate the IRC using the CMU.

In this oscillator, two's complement trimming method is implemented. So the trimming code increases from -32 to 31. As the trimming code increases, the internal time constant increases and frequency reduces. Please refer to device datasheet for average frequency variation of the trimming step.

Address: 0xC3FE_0060 (Base + 0x0000) Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	RCTRIM[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 11. RC Control register (RC_CTL)

Table 11. RC_CTL field descriptions

Field	Description
RCTRIM[5:0]	Main RC trimming bits

4.8 XOSC external crystal oscillator

The external crystal oscillator (XOSC) operates in the range of 4 MHz to 40 MHz. The XOSC digital interface contains the control and status registers accessible for the external crystal oscillator.

Main features are:

- Oscillator clock available interrupt
- Oscillator bypass mode

4.8.1 Functional description

The crystal oscillator circuit includes an internal oscillator driver and an external crystal circuitry. The XOSC provides an output clock to the PLL or it is used as a reference clock to specific modules depending on system needs.

The crystal oscillator can be controlled by the ME:

- Control by ME module. The OSCON bit of the ME_xxx_MCRs controls the powerdown of oscillator based on the current device mode while S_OSC of ME_GS register provides the oscillator clock available status.

After system reset, the oscillator is put to power down state and software has to switch on when required. Whenever the crystal oscillator is switched on from off state, OSCCNT counter starts and when it reaches the value EOCV[7:0] × 512, oscillator clock is made available to the system. Also an interrupt pending bit I_OSC of OSC_CTL register is set. An interrupt will be generated if the interrupt mask bit M_OSC is set.

The oscillator circuit can be bypassed by setting OSC_CTL[OSCBYP]. This bit can only be set by the software. System reset is needed to reset this bit. In this bypass mode, the output clock has the same polarity as external clock applied on EXTAL pin and the oscillator status is forced to ‘1’. The bypass configuration is independent of the powerdown mode of the oscillator.

Table 12 shows the truth table of different configurations of oscillator.

Table 12. Crystal oscillator truth table

ENABLE	BYP	XTALIN	EXTAL	CK_OSCM	XOSC Mode
0	0	No crystal, High Z	No crystal, High Z	0	Power down, IDDQ
x	1	x	Ext clock	EXTAL	Bypass, XOSC disabled
1	0	Crystal	Crystal	EXTAL	Normal, XOSC enabled

4.8.2 Register description

Table 13. OSC_CTL memory map

Offset from OSC_CTL_BASE (0xC3FE_0000)	Register	Location
0x0000	OSC_CTL—Oscillator control register	*on page 114*
0x0004–0x005F	Reserved	

Address: 0xC3FE_0000
(Base + 0x0000)

Access: Supervisor read/write; User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OSC BYP	0	0	0	0	0	0	0	EOCV[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M_ OSC	0	0	0	0	0	0	0	I_ OSC	0	0	0	0	0	0	0
W									w1c							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12. Crystal Oscillator Control register (OSC_CTL)

Table 14. OSC_CTL field descriptions

Field	Description
OSCBYP	Crystal Oscillator bypass This bit specifies whether the oscillator should be bypassed or not. Software can only set this bit. System reset is needed to reset this bit. 0: Oscillator output is used as root clock. 1: EXTAL is used as root clock.
EOCV[7:0]	End of Count Value These bits specify the end of count value to be used for comparison by the oscillator stabilization counter OSCCNT after reset or whenever it is switched on from the off state. This counting period ensures that external oscillator clock signal is stable before it can be selected by the system. When oscillator counter reaches the value EOCV[7:0]*512, oscillator available interrupt request is generated. The reset value of this field depends on the device specification. The OSCCNT counter will be kept under reset if oscillator bypass mode is selected.
M_OSC	Crystal oscillator clock interrupt mask 0: Crystal oscillator clock interrupt masked 1: Crystal oscillator clock interrupt enabled
I_OSC	Crystal oscillator clock interrupt This bit is set by hardware when OSCCNT counter reaches the count value EOCV[7:0]*512. It is cleared by software by writing 1. 0: No oscillator clock interrupt occurred 1: Oscillator clock interrupt pending

4.9 Frequency Modulated Phase Locked Loop (FMPLL)

4.9.1 Introduction

This section describes the features and functions of the two independent FMPLL modules implemented in SPC560P44Lx, SPC560P50Lx.

4.9.2 Overview

The FMPLLs enable the generation of high speed system clocks from a common 4–40 MHz input clock. Further, the FMPLLs support programmable frequency modulation of the system clock. The PLL multiplication factor and output clock divider ratio are all software configurable.

SPC560P44Lx, SPC560P50Lx has two PLLs so that one can be used as the system clock and take advantage of the FM modulation, while the other one can be used for motor control peripherals.

The FMPLL block diagram is shown in *Figure 13*.

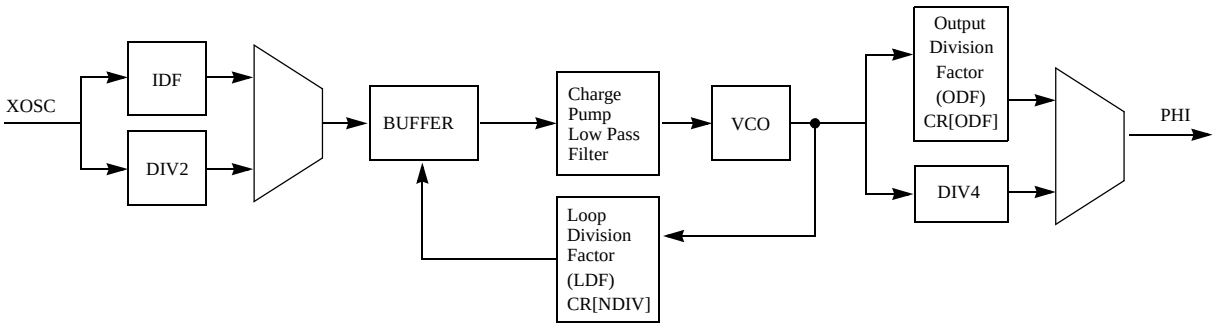


Figure 13. FMPLL block diagram

4.9.3 Features

The FMPLL has the following major features:

- Input clock frequency 4–40 MHz
- Voltage controlled oscillator (VCO) range from 256 MHz to 512 MHz
- Reduced frequency divider (RFD) for reduced frequency operation without forcing the FMPLL to relock
- Frequency modulated PLL
  - Modulation enabled/disabled through software
  - Triangle wave modulation
- Programmable modulation depth
  - ±0.25% to ±4% deviation from center spread frequency
  - –0.5% to +8% deviation from down spread frequency
  - Programmable modulation frequency dependent on reference frequency
- Self-clocked mode (SCM) operation
- 4 available modes
  - Normal mode
  - Progressive clock switching
  - Normal Mode with SSCG
  - Powerdown mode

4.9.4 Memory map

Table 15 shows the memory map locations. Addresses are given as offsets of the module base address.

Table 15. FMPLL memory map

Offset from ME_CGM_BASE(1) FMPLL_0: 0xC3FE_00A0 FMPLL_1: 0xC3FE_00C0	Register	Location
0x0000	CR—Control Register	*on page 117*
0x0004	MR—Modulation register	*on page 118*
0x0004–0x001F	Reserved	

1. FMPLL_x are mapped through the ME_CGM Register Slot

4.9.5 Register description

The PLL operation is controlled by two registers. Those registers can only be written in supervisor mode.

4.9.5.1 Control Register (CR)

Address: Base + 0x0000

Access: Supervisor read/write
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	IDF[3:0]				ODF[1:0]		0	NDIV[6:0]						
W																
Reset	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	en_pll_sw	0	unlock_once	0	i_lock	s_lock	pll_fail_mask	pll_fail_flag	1
W												w1c			w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 14. Control Register (CR)

Table 16. CR field descriptions

Field	Description
IDF[3:0]	Input Division Factor The value of this field sets the PLL input division factor. 0000: Divide by 1 0001: Divide by 2 0010: Divide by 3 0011: Divide by 4 0100: Divide by 5 0101: Divide by 6 0110: Divide by 7 0111: Divide by 8 1000: Divide by 9 1001: Divide by 10 1010: Divide by 11 1011: Divide by 12 1100: Divide by 13 1101: Divide by 14 1110: Divide by 15 1111: Clock Inhibit
ODF[1:0]	Output Division Factor The value of this field sets the PLL output division factor. 00: Divide by 2 01: Divide by 4 10: Divide by 8 11: Divide by 16

Table 16. CR field descriptions(Continued)

Field	Description
NDIV[6:0]	Loop Division Factor The value of this field sets the PLL loop division factor. 0000000–0011111: Reserved 0100000: Divide by 32 0100001: Divide by 33 0100010: Divide by 34 ... 1011111: Divide by 95 1100000: Divide by 96 1100001–1111111: Reserved
en_pll_sw	This bit is used to enable progressive clock switching. After the PLL locks, the PLL output initially is divided by 8, and then progressively decreases until it reaches divide-by-1. **Note:** Note: The PLL output should not be used if a non-changing clock is needed, such as for serial communications, until the division has finished. 0: Progressive clock switching disabled 1: Progressive clock switching enabled
unlock_once	This bit is a sticky indication of PLL loss of lock condition. Unlock_once is set when the PLL loses lock. Whenever the PLL reacquires lock, unlock_once remains set. unlock_once is cleared after a POR event.
i_lock	This bit is set by hardware whenever there is a lock/unlock event.It is cleared by software, writing 1.
s_lock	This bit indicates whether the PLL has acquired lock. 0: PLL unlocked 1: PLL locked
pll_fail_mask	This bit masks the pll_fail output. 0: pll_fail not masked 1: pll_fail masked
pll_fail_flag	This bit is asynchronously set by hardware whenever a loss of lock event occurs while PLL is switched on. It is cleared by software, writing 1.

4.9.5.2 Modulation Register (MR)

Address: Base + 0x0004

Access: Supervisor read/write
User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	STRB_BYPASS	0	SPRD_SEL	MOD_PERIOD												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	FM_EN	INC_STEP														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15. Modulation Register (MR)

Table 17. MR field descriptions

Field	Description
STRB_BYPASS	Strobe bypass The STRB_BYPASS signal bypasses the STRB signal used inside the PLL to latch the correct values for control bits (INC_STEP, MOD_PERIOD and SPRD_SEL). 0: STRB latches the PLL modulation control bits. 1: STRB is bypassed. In this case, the control bits need to be static. The control bits must be changed only when PLL is in power down mode.
SPRD_SEL	Spread type selection The SPRD_SEL bit selects the spread type in Frequency Modulation mode. 0: Center spread 1: Down spread
MOD_PERIOD	Modulation period The MOD_PERIOD field is the binary equivalent of the value modperiod derived from following formula: $modperiod = \frac{f_{ref}}{4 \times f_{mod}}$ where: *fref:* represents the frequency of the feedback divider *fmod*: represents the modulation frequency
FM_EN	Frequency modulation enable The FM_EN bit enables the frequency modulation. 0: Frequency Modulation disabled 1: Frequency Modulation enabled
INC_STEP	Increment step The INC_STEP field is the binary equivalent of the value incstep derived from following formula: $incstep = round\left(\frac{(2^{15}-1) \times md \times MDF}{100 \times 5 \times MODPERIOD}\right)$ where: *md*: represents the peak modulation depth in percentage (Center spread — pk-pk = ±md, Downspread — pk-pk = –2 × md) *MDF*: represents the nominal value of loop divider (NDIV in PLL Control Register).

4.9.6 Functional description

4.9.6.1 Normal mode

In Normal mode, the PLL inputs are driven by the Control Register (CR). This means that when the PLL is locked, the PLL output clock (PHI) is derived from the reference clock (XOSC) through this relationship:

Equation 1

$$phi = \frac{xosc \cdot ldf}{idf \cdot odf}$$

where the value of *idf* (Input Division Factor), *ldf* (Loop Division Factor), and *odf* (Output Division Factor) are set in the CR as shown in *Table 16*. *idf* and *odf* are specified in the IDF and ODF bitfields, respectively; *ldf* is specified in the NDIV bitfield.

4.9.6.2 Progressive clock switching

Progressive clock switching allows to switch system clock to PLL output clock stepping through different division factors. This means that the current consumption gradually increases and, in turn, voltage regulator response is improved.

This feature can be enabled by programming bit en_pll_sw in the CR. Then, when the PLL is selected as the system clock, the output clock progressively increases its frequency as shown in *Table 18*.

Table 18. Progressive clock switching on pll_select rising edge

Number of PLL output clock cycles	ck_pll_div frequency (PLL output frequency)
8	(ck_pll_out frequency) ÷ 8
16	(ck_pll_out frequency) ÷ 4
32	(ck_pll_out frequency) ÷ 2
onward	(ck_pll_out frequency)

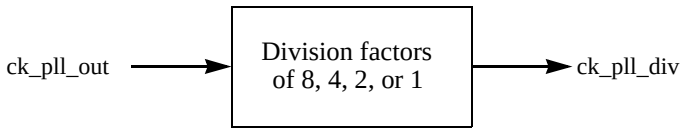


Figure 16. Progressive clock switching scheme

4.9.6.3 Normal Mode with frequency modulation

The FMPLL default mode is without frequency modulation enabled. When frequency modulation is enabled, however, two parameters must be set to generate the desired level of modulation: the PERIOD, and the STEP. The modulation waveform is always a triangle wave and its shape is not programmable.

Frequency modulation is activated as follows:

1. Configure the FM modulation characteristics: MOD_PERIOD, INC_STEP.
2. Enable the FM modulation by programming bit SSCG_EN of the MR to ‘1’. FM modulated mode can be enabled only when PLL is in lock state.

There are two ways to latch these values inside the FMPLL, depending on the value of bit STRB_BYPASS in the MR.

If STRB_BYPASS is low, the modulation parameters are latched in the PLL only when the strobe signal goes high. The strobe signal is automatically generated in the FMPLL when the modulation is enabled (SSCG_EN goes high) if the PLL is locked (s_lock = 1) or when the modulation has been enabled (SSCG_EN = 1) and PLL enters in lock state (s_lock goes high).

If STRB_BYPASS is high, the strobe signal is bypassed. In this case, control bits (MOD_PERIOD[12:0], INC_STEP[14:0], SPREAD_CONTROL) must be changed only when the PLL is in power down mode.

The modulation depth in % is

Equation 2

$$ModulationDepth = \left(\frac{100 \times 5 \times INCSTEPxMODPERIOD}{(2^{15} - 1) \times MDF}\right)$$

Note: The user must ensure that the product of INCSTEP and MODPERIOD is less than (2^{15} – 1).

The following values show the input setting for one possible configuration of the PLL:

- PLL input frequency: 4 MHz
- Loop divider (LDF): 64
- Input divider (IDF): 1
- VCO frequency = 4 MHz × 64 = 256 MHz
- PLL output frequency = 256 MHz/ODF
- Spread: Center spread (SPREAD_CONTROL = 0)
- Modulation frequency = 24 kHz
- Modulation depth = ±2.0% (4% pk-pk)

Using the formulae for MODPERIOD and INCSTEP:

Equation 3
MODPERIOD = Round [(4e06) / (4 × 24e03)] = Round [41.66] = 42

Equation 4
INCSTEP = Round [((2^{15} – 1) × 2 × 64) / (100 × 5 × 42)] = Round [199.722] = 200

Equation 5
MODPERIOD × INCSTEP = 42 × 200 = 8400 (which is less than 2^{15})

Equation 6
md(quantized)% = ((42*200*100*5) / ((2^15-1)*64) = 2.00278% (peak)

Equation 7
Error in modulation depth = 2.00278 - 2.0 = 0.00278%

If we choose MODPERIOD = 41,

Equation 8
INCSTEP = Round [((2^{15} – 1) × 2 × 64) / (100 × 5 × 41)] = Round [204.878] = 205

Equation 9
MODPERIOD × INCSTEP = 41 × 205 = 8405 (which is less than 2^{15})

Equation 10
md(quantized)% = ((41 × 205 × 100 × 5) / ((2^{15} – 1) × 64) = 2.00397% (peak)

Equation 11
Error in modulation depth = 2.00397 – 2.0 = 0.00397%

The above calculations show that the quantization error in the modulation depth depends on the flooring and rounding of MODPERIOD and INCSTEP. For this reason, the MODPERIOD and INCSTEP should be judiciously rounded/floored to minimize the quantization error in the modulation depth.

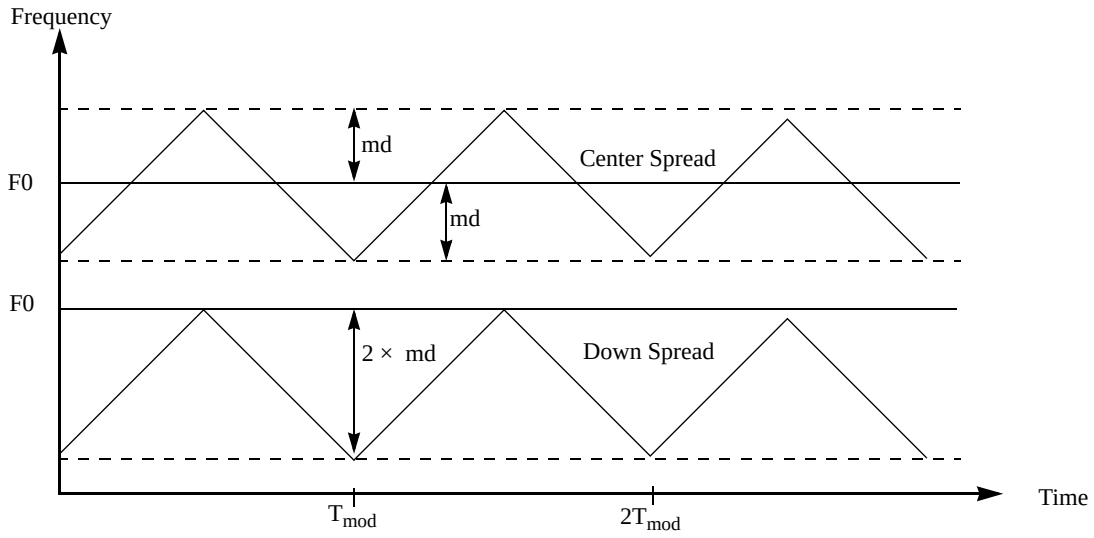


Figure 17. Frequency modulation depth spreads

4.9.6.4 Powerdown mode

To reduce consumption, the FMPLL can be switched off when not required by programming the registers ME_x_MC on the ME module.

4.9.7 Recommendations

To avoid any unpredictable behavior of the PLL clock, it is recommended to follow these guidelines:

- The PLL VCO frequency should reside in the range 256 MHz to 512 MHz. Care is required when programming the multiplication and division factors to respect this requirement.
- The user must change the multiplication, division factors only when the PLL output clock is not selected as system clock. MOD_PERIOD, INC_STEP, SPREAD_SEL bits should be modified before activating the FM modulated mode. Then strobe has to be generated to enable the new settings. If STRB_BYP is set to ‘1’ then MOD_PERIOD, INC_STEP and SPREAD_SEL can be modified only when PLL is in power down mode.
- Use progressive clock switching.

4.10 Clock Monitor Unit (CMU)

4.10.1 Overview

The Clock Monitor Unit (CMU) serves three purposes:

- PLL clock monitoring: detects if PLL leaves an upper or lower frequency boundary
- XOSC clock monitoring: monitor the XOSC clock, which must be greater than the IRCOSC clock divided by a division factor given by CMU_CSR[RCDIV]
- Frequency meter: measure the frequency of the IRCOSC clock versus the reference XOSC clock frequency

When mismatch occurs in the CMU either with the PLL monitor or the XOSC monitor, the CMU notifies the RGM, ME and the FCU (Fault Collection Unit) modules. The default behavior is such that a reset occurs and a status bit is set in the RGM. The user also has the option to change the behavior of the action by disabling the reset and selecting an alternate action. The alternate action can be either entering safe mode or generating an interrupt.

Since the SPC560P44Lx, SPC560P50Lx device has two PLLs, two CMU subunits are implemented. CMU_1 monitors only the FMPLL_1 output (see *Table 19* and *Figure 18*).

Table 19. CMU module summary

Module	Monitored clocks
CMU_0	– XOSC integrity supervisor – FMPLL_0 integrity supervisor – IRCOSC frequency meter
CMU_1	FMPLL_1 integrity supervisor

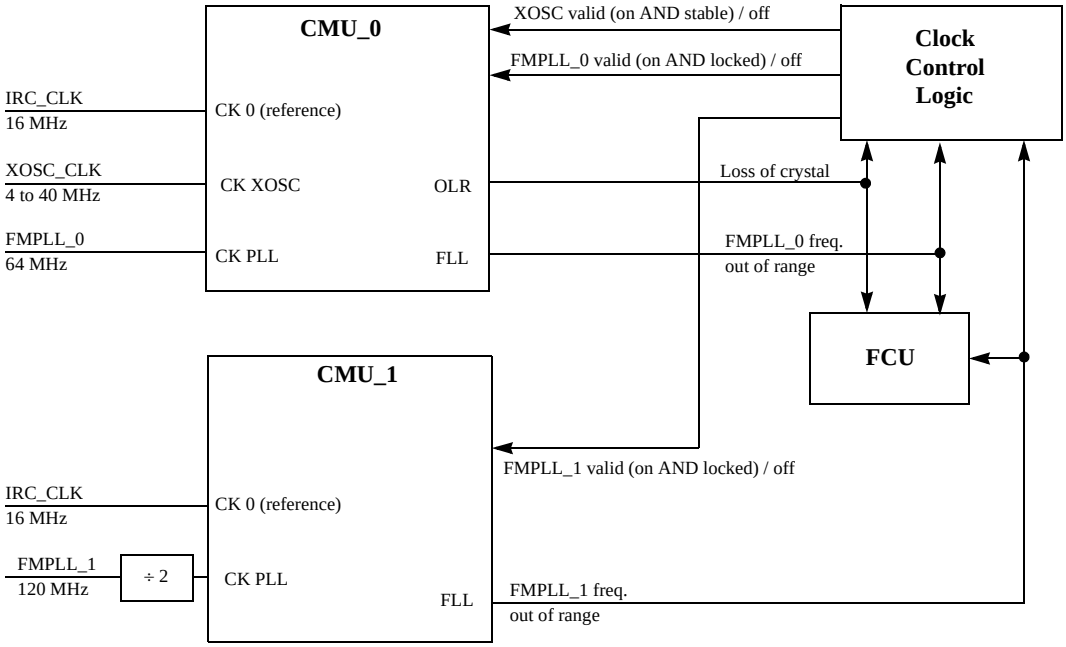


Figure 18. SPC560P44Lx, SPC560P50Lx with two CMUs

4.10.2 Main features

- RC oscillator frequency measurement
- External oscillator clock monitoring with respect to CK_IRC/*n* clock
- PLL clock frequency monitoring with respect to CK_IRC/4 clock
- Event generation for various failures detected inside monitoring unit

4.10.3 Functional description

The clock and frequency names referenced in this block are defined as follows:

- CK_XOSC: clock coming from the external crystal oscillator
- CK_IRC: clock coming from the low frequency internal RC oscillator
- CK_PLL: clock coming from the PLL[(a)]
- f_{XOSC}: frequency of external crystal oscillator clock
- f_{RC}: frequency of low frequency internal RC oscillator
- f_{PLL}: frequency of FMPLL clock

4.10.3.1 Crystal clock monitor[(b)]

If f_{XOSC} is smaller than f_{RC} divided by 2^{RCDIV} bits of CMU_0_CSR and the CK_XOSC is 'ON' and stable as signaled by the ME, then:

- An event pending bit OLRI in CMU_0_ISR is set
- A failure event OLR is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

4.10.3.2 PLL clock monitor

The PLL clock CK_PLL frequency can be monitored by programming bit CME_*n* of the CMU_*n*_CSR to '1'. The CK_PLL monitor starts as soon as bit CME_*n* is set. This monitor can be disabled at any time by writing bit CME_*n* to '0'.

If the CK_PLL frequency (f_{PLL}) is greater than a reference value determined by bits HFREF[11:0] of the CMU_HFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME then:

- An event pending bit FHHI_*n* in the CMU_*n*_ISR is set.
- A failure event FHH is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

If f_{PLL} is less than a reference clock frequency ($f_{RC}/4$) and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then:

- An event pending bit FLCI_*n* in the CMU_*n*_ISR is set.
- A failure event FLC is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

If f_{PLL} is less than a reference value determined by bits LFREF[11:0] of the CMU_LFREFR and the CK_PLL is 'ON' and the PLL locked as signaled by the ME, then:

- An event pending bit FLLI_*n* in the CMU_*n*_ISR is set.
- A failure event FLL is signaled to the RGM and FCU, which in turn can generate either an interrupt, a reset, or a SAFE mode request.

Note: It is possible for either the XOSC or PLL monitors to produce a false event when the XOSC or PLL frequency is too close to RC/2^{RCDIV} frequency due to an accuracy limitation of the compare circuitry.

a. CMU_0 refers to FMPLL_0 and CMU_1 refers to FMPLL_1.

b. Enabled only in CMU_0.

4.10.3.3 System clock monitor

The system clock is monitored by CMU_1. The F_{SYS_CLK} frequency can be monitored by programming CMU_1_CSR[CME] = 1. SYS_CLK monitoring starts as soon as CMU_1_CSR[CME] = 1. This monitor can be disabled at any time by writing CME bit to 0.

If F_{SYS_CLK} is greater than a reference value determined by the CMU_1_HFREFR_A[HFREF_A] bits and the system clock is enabled, then:

- CMU_1_ISR[FHHI] is set
- A failure event is signaled to the MC_RGM and FCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

If F_{SYS_CLK} is less than a reference clock frequency (F_{IRCOSC_CLK}÷4) and the system clock is enabled, then:

- CMU_1_ISR[FLCI] is set
- A failure event FLC is signaled to the MC_RGM and Fault Collection Unit, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

If F_{SYS_CLK} is less than a reference value determined by the CMU_1_LFREFR_A[LFREF_A] bits and the system clock is enabled, then:

- CMU_1_ISR[FLLI] is set
- A failure event is signaled to the MC_RGM and FCU, which in turn can generate a 'functional' reset, a SAFE mode request, or an interrupt

Note: The system clock monitor may produce a false event when F_{SYS_CLK} is less than $2 \times F_{IRCOSC\_CLK}/2^{CMU\_1\_CSR[RCDIV]}$ due to an accuracy limitation of the compare circuitry.

4.10.3.4 Frequency meter

The frequency meter calibrates the internal RC oscillator (CK_IRC) using a known frequency. The frequency meter is implemented only on CMU_0.

Note: This value can then be stored into the flash so that application software can reuse it later on.

The reference clock will be always the XOSC. A simple frequency meter returns a draft value of CK_IRC. The measure starts when bit SFM (Start Frequency Measure) in the CMU_CSR is set to '1'. The measurement duration is given by the CMU_MDR in numbers of IRC clock cycles with a width of 20 bits. Bit SFM is reset to '0' by hardware once the frequency measurement is done and the count is loaded in the CMU_FDR. The frequency f_{RC} can be derived from the value loaded in the CMU_FDR as follows:

Equation 12

$$f_{RC} = (f_{OSC} \times MD) / n$$

where *n* is the value in the CMU_FDR and MD is the value in the CMU_MDR.

4.10.4 Memory map and register description

Table 20 shows the memory map of the CMU.

Table 20. CMU memory map

Offset from CMU_BASE (0xC3FE_0100)	Register	Location
0x0000	Control Status Register (CMU_0_CSR)	*on page 126*
0x0004	Frequency Display Register (CMU_0_FDR)	*on page 127*
0x0008	High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_A)	*on page 128*
0x000C	Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)	*on page 128*
0x0010	Interrupt Status Register (CMU_0_ISR)	*on page 129*
0x0014	Reserved	
0x0018	Measurement Duration Register (CMU_0_MDR)	*on page 130*
0x001C	Reserved	
0x0020	*Control Status Register (CMU_1_CSR)*	*on page 126*
0x0024	Reserved	
0x0028	*High Frequency Reference Register FMPLL_1 (CMU_1_HFREFR_A)*	*on page 128*
0x002C	*Low Frequency Reference Register FMPLL_1 (CMU_1_LFREFR_A)*	*on page 128*
0x0030	*Interrupt Status Register (CMU_1_ISR)*	*on page 129*
0x0034–0x026F	Reserved	

4.10.4.1 Control Status Register (CMU_0_CSR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	SFM	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	RCDIV[1:0]		CME_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19. Control Status Register (CMU_0_CSR)

Table 21. CMU_0_CSR field descriptions

Field	Description
SFM	Start frequency measure The software can only set this bit to start a clock frequency measure. It is reset by hardware when the measure is ready in the CMU_FDR. 0: Frequency measurement completed or not yet started 1: Frequency measurement not completed
RCDIV[1:0]	RC clock division factor These bits specify the RC clock division factor. The output clock is CK_IRC divided by the factor 2^{RCDIV}. This output clock is compared with CK_XOSC for crystal clock monitor feature.The clock division coding is as follows. 00: Clock divided by 1 (no division) 01: Clock divided by 2 10: Clock divided by 4 11: Clock divided by 8
CME_0	FMPLL_0 clock monitor enable 0: FMPLL_0 monitor disabled 1: FMPLL_0 monitor enabled

4.10.4.2 Frequency Display Register (CMU_0_FDR)

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	FD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20. Frequency Display Register (CMU_0_FDR)

Table 22. CMU_0_FDR field descriptions

Field	Description
FD[19:0]	Measured frequency bits This register displays the measured frequency f_{RC} with respect to f_{OSC}. The measured value is given by the following formula: $f_{RC} = (f_{OSC} \times MD) / n$, where n is the value in CMU_0_FDR.

4.10.4.3 High Frequency Reference Register FMPLL_0 (CMU_0_HFREFR_A)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	HFREF_A[11:0]											
W																
Reset	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Figure 21. High Frequency Reference register FMPLL_0 (CMU_0_HFREFR_A)

Table 23. CMU_0_HFREFR_A field descriptions

Field	Description
HFREF_A	High Frequency reference value These bits determine the high reference value for the FMPLL_0 clock. The reference value is given by: (HFREF_A[11:0]/16) × (f_{RC}/4).

4.10.4.4 Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF_A[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22. Low Frequency Reference Register FMPLL_0 (CMU_0_LFREFR_A)

Table 24. CMU_0_LFREFR_A fields descriptions

Field	Description
LFREF_A	Low Frequency reference value These bits determine the low reference value for the FMPLL_0. The reference value is given by: (LFREF_A[11:0]/16) * (f_{RC}/4).

4.10.4.5 Interrupt Status Register (CMU_0_ISR)

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_0	FHHI_0	FLLI_0	OLRI
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23. Interrupt Status Register (CMU_0_ISR)

Table 25. CMU_0_ISR field descriptions

Field	Description
FLCI_0	FMPLL_0 Clock frequency less than reference clock interrupt This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than reference clock frequency (f_{RC}/4) value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLC event 1: FLC event pending
FHHI_0	FMPLL_0 Clock frequency higher than high reference interrupt This bit is set by hardware when CK_FMPLL_ 0 frequency becomes higher than HFREF_A value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FHH event 1: FHH event pending
FLLI_0	FMPLL_0 Clock frequency less than low reference event This bit is set by hardware when CK_FMPLL_0 frequency becomes lower than LFREF_A value and CK_FMPLL_0 is 'ON' and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0: No FLL event 1: FLL event pending
OLRI	Oscillator frequency less than RC frequency event This bit is set by hardware when the frequency of CK_XOSC is less than CK_IRC/2^{RCDIV} frequency and CK_XOSC is 'ON' and stable as signaled by the ME. It can be cleared by software by writing 1. 0: No OLR event 1: OLR event pending

4.10.4.6 Measurement Duration Register (CMU_0_MDR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	MD[19:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MD[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24. Measurement Duration Register (CMU_0_MDR)

Table 26. CMU_0_MDR field descriptions

Field	Description
MD[19:0]	Measurement duration bits This register displays the measured duration in term of IRC clock cycles. This value is loaded in the frequency meter downcounter. When SFM bit is set to ‘1’, downcounter starts counting.

4.10.4.7 Control Status Register (CMU_1_CSR)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CME_1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25. Control Status Register (CMU_1_CSR)

Table 27. CMU_1_CSR field descriptions

Field	Description
CME_1	FMPLL_1 clock monitor enable 0 FMPLL_1 monitor disabled 1 FMPLL_1 monitor enabled

4.10.4.8 High Frequency Reference Register FMPLL_1 (CMU_1_HFREFR_A)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	HFREF_A[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26. High Frequency Reference Register FMPLL_1 (CMU_1_HFREFR_A)

Table 28. CMU_1_HFREFR_A field descriptions

Field	Description
HFREF_A[11:0]	High Frequency reference value These bits determine the high reference value for the FMPLL_1 clock. The reference value is given by: (HFREF_A[11:0]/16) × (F_{RC}/4).

4.10.4.9 Low Frequency Reference Register FMPLL_1 (CMU_1_LFREFR_A)

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	LFREF_A[11:0]											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27. Low Frequency Reference Register FMPLL_1 (CMU_1_LFREFR_A)

Table 29. CMU_1_LFREFR_A field descriptions

Field	Description
LFREF_A[11:0]	Low Frequency reference value These bits determine the low reference value for the FMPLL_1 clock. The reference value is given by: (LFREF_A[11:0]/16) × (f_{RC}/4).

4.10.4.10 Interrupt Status Register (CMU_1_ISR)

Address Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FLCI_1	FHHI_1	FLLI_1	0
W													w1c	w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28. Interrupt Status register (CMU_1_ISR)

Table 30. CMU_1_ISR field descriptions

Field	Description
FLCI_1	FMPLL_1 clock frequency less than reference clock event. This bit is set by hardware when the FMPLL_1 clock frequency is lower than the reference clock frequency (FRC/4) value and FMPLL_1 is ‘ON’ and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FLC event. 1 FLC event is pending.
FHHI_1	FMPLL_1 clock frequency higher than high reference event. This bit is set by hardware when the CK_FMPLL_1 frequency is higher than the HFREF_A value and CK_FMPLL_1 is ‘ON’ and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FHH event. 1 FHH event is pending.
FLLI_1	FMPLL_1 clock frequency less than low reference event. This bit is set by hardware when the CK_FMPLL_1 frequency becomes lower than LFREF_A value and CK_FMPLL_1 is ‘ON’ and the PLL locked as signaled by the ME. It can be cleared by software by writing 1. 0 No FLL event. 1 FLL event is pending.

5 Clock Generation Module (CGM)

5.1 Introduction

The clock generation module (CGM) generates reference clocks for all other modules on the device. The mode entry module (ME) controls the system clock selection (see *Section 6.1: Introduction*, for more details). Peripheral clock selection is controlled by CGM control registers. A set of CGM registers controls the clock dividers that are utilized for divided system and peripheral clock generation. The memory spaces of system and peripheral clock sources with addressable memory spaces are accessed through the CGM memory space. The CGM also selects and generates an output clock.

Figure 29 shows the CGM block diagram.

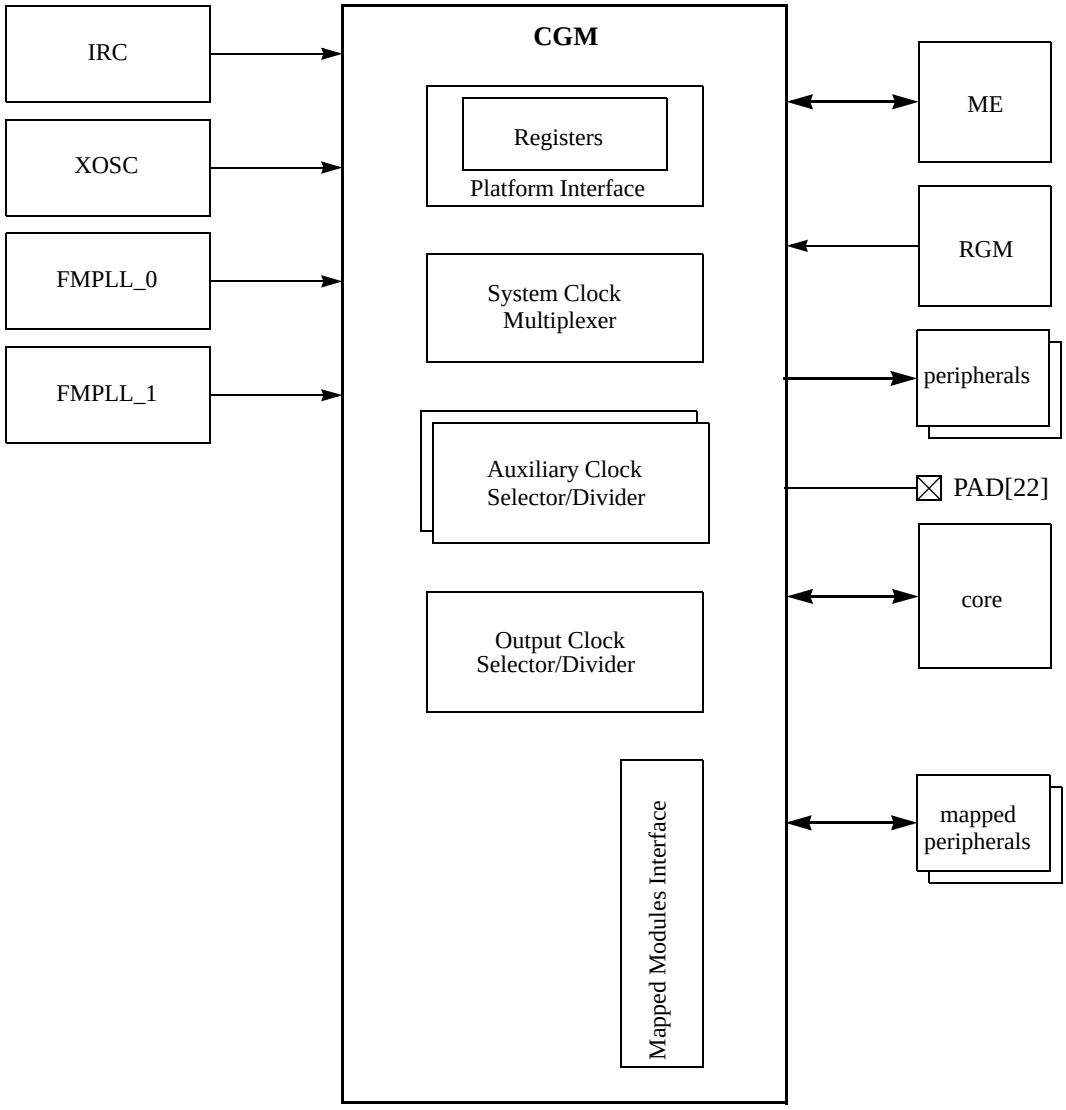


Figure 29. CGM block diagram

5.2 Features

The CGM includes the following features:

- Generates system and peripheral clocks
- Selects and enables/disables the system clock supply from system clock sources according to ME control
- Contains a set of registers to control clock dividers for divided clock generation
- Contains a set of registers to control peripheral clock selection
- Supports multiple clock sources and maps their address spaces to its memory map
- Generates an output clock
- Guarantees glitch-less clock transitions when changing the system clock selection
- Supports 8, 16, and 32-bit wide read/write accesses

5.3 Modes of operation

This section describes the basic functional modes of the CGM.

5.3.1 Normal and Reset Modes of Operation

During normal and reset modes of operation, the clock selection for the system clock is controlled by the ME.

5.4 External signal description

The CGM delivers an output clock to the PAD[22] pin for off-chip use and/or observation.

5.5 Memory map and registers description

Table 31. CGM memory map

Offset from CGM_BASE (0xC3FE_0000)	Register	Location
0x0370	CGM_OC_EN—Output Clock Enable	*on page 137*
0x0374	CGM_OCDS_SC—Output Clock Division Select	*on page 138*
0x0378	CGM_SC_SS—System Clock Select Status	*on page 139*
0x037C–0x037F	Reserved	
0x0380	CGM_AC0_SC—Aux Clock 0 Select Control	*on page 139*
0x0384	CGM_AC0_DC0—Aux Clock 0 Divider Configuration 0	*on page 141*
0x0388	CGM_AC1_SC—Aux Clock 1 Select Control	*on page 141*
0x038C	CGM_AC1_DC0—Aux Clock 1 Divider Configuration 0	*on page 142*
0x0390	CGM_AC2_SC—Aux Clock 2 Select Control	*on page 143*

Table 31. CGM memory map(Continued)

Offset from CGM_BASE (0xC3FE_0000)	Register	Location
0x0394	CGM_AC2_DC0—Aux Clock 2 Divider Configuration 0	*on page 144*
0x0398	CGM_AC3_SC—Aux Clock 3 Select Control	*on page 145*
0x039C	CGM_AC3_DC0—Aux Clock 3 Divider Configuration 0	*on page 145*
0x037A0–0x3FFF	Reserved	

Note: Any access to unused registers as well as write accesses to read-only registers will not change register content, and can cause a transfer error .

Table 32. CGM registers

Address	Name	0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_0000 … 0xC3FE_001C	XOSC0 registers																
0xC3FE_0020 … 0xC3FE_005C	Reserved																
0xC3FE_0060 … 0xC3FE_007C	IRC registers																
0xC3FE_0080 … 0xC3FE_009C	Reserved																
0xC3FE_00A0 … 0xC3FE_00BC	FMPLL_0 registers																
0xC3FE_00C0 … 0xC3FE_00DC	FMPLL_1 registers																
0xC3FE_00E0 … 0xC3FE_00FC	Reserved																
0xC3FE_0100 … 0xC3FE_011C	CMU0 registers																
0xC3FE_0120 … 0xC3FE_013C	CMU1 registers																

Table 32. CGM registers (Continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	27 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FE_0140 … 0xC3FE_036C	Reserved																	
0xC3FE_0370	CGM_OC_EN	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
		W																
0xC3FE_0374	CGM_OCDS_SC	R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0378	CGM_SC_SS	R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_037C	Reserved																	
0xC3FE_0380	CGM_AC0_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0384	CGM_AC0_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0388	CGM_AC1_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_038C	CGM_AC1_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 32. CGM registers (Continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	27 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FE_0390	CGM_AC2_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0394	CGM_AC2_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0398	CGM_AC3_SC	R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_039C	CGM_AC3_DC0	R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_0400 … 0xC3FE_3FFC	Reserved																	

5.6 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the CGM_OC_EN register may be accessed as a word at address 0xC3FE_0370, as a half-word at address 0xC3FE_0372, or as a byte at address 0xC3FE_0373.

5.6.1 Output Clock Enable register (CGM_OC_EN)

The Output Clock Enable register (CGM_OC_EN) enables and disables the output clock.

Address: Base + 0x0370 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 30. Output Clock Enable register (CGM_OC_EN)

Table 33. CGM_OC_EN field descriptions

Field	Description
EN	Output Clock Enable control 0 Output Clock is disabled. 1 Output Clock is enabled.

5.6.2 Output Clock Division Select register (CGM_OCDS_SC)

The Output Clock Division Select register (CGM_OCDS_SC) selects the current output clock source and the factor by which it is divided before being delivered at the output clock.

Address: Base + 0x0374 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SELDIV		SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31. Output Clock Division Select register (CGM_OCDS_SC)

Table 34. CGM_OCDS_SC field descriptions

Field	Description
SELDIV	Output Clock Division Select 00 Output selected Output Clock without division. 01 Output selected Output Clock divided by 2. 10 Output selected Output Clock divided by 4. 11 Output selected Output Clock divided by 8.
SELCTL	Output Clock Source Selection Control — This value selects the current source for the output clock. 0000 IRC 0001 XOSC 0010 FMPLL_0 0011 FMPLL_1 All other values are reserved.

5.6.3 System Clock Select Status register (CGM_SC_SS)

The System Clock Select Status register (CGM_SC_SS) provides the current system clock source selection.

Address: Base + 0x0378 Access: User read-only, Supervisor read-only, Test read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELSTAT				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	C	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32. System Clock Select Status register (CGM_SC_SS)

Table 35. CGM_SC_SS field descriptions

Field	Description
SELSTAT	System Clock Source Selection Status — This value indicates the current source for the system clock. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 0101 FMPLL_1 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 System clock is disabled.

5.6.4 Auxiliary Clock 0 Select Control register (CGM_AC0_SC)

The Auxiliary Clock 0 Select Control register (CGM_AC0_SC) selects the current auxiliary clock 0 source.

Address: Base + 0x0380 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33. Auxiliary Clock 0 Select Control register (CGM_AC0_SC)

Table 36. CGM_AC0_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 0 Source Selection Control — This value selects the current source for auxiliary clock 0. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 0101 FMPLL_1 0110 reserved 0111 reserved 1000 FMPLL_1_D1 All other values are reserved.

5.6.5 Auxiliary Clock 0 Divider Configuration register (CGM_AC0_DC0)

The Auxiliary Clock 0 Divider Configuration register (CGM_AC0_DC0) controls the auxiliary clock 0 divider.

Address: Base + 0x0384 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34. Auxiliary Clock 0 Divider Configuration register (CGM_AC0_DC0)

Table 37. CGM_AC0_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 0 divider 0. 1 Enable auxiliary clock 0 divider 0.
DIV0	Divider 0 Division Value — The resultant motor control clock will have a period DIV0 + 1 times that of auxiliary clock 0. If DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the motor control clock remains disabled.

5.6.6 Auxiliary Clock 1 Select Control register (CGM_AC1_SC)

The Auxiliary Clock 1 Select Control register (CGM_AC1_SC) selects the current auxiliary clock 1 source.

Address: Base + 0x0388 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35. Auxiliary Clock 1 Select Control register (CGM_AC1_SC)

Table 38. CGM_AC1_SC) field descriptions

Field	Description
SELCTL	Auxiliary Clock 1 Source Selection Control — This value selects the current source for auxiliary clock 1. 0000 FMPLL_1 0001 reserved 0010 reserved 0011 reserved 0100 reserved 0101 reserved 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

5.6.7 Auxiliary Clock 1 Divider Configuration register (CGM_AC1_DC0)

The Auxiliary Clock 1 Divider Configuration register (CGM_AC1_DC0) controls the auxiliary clock 1 divider.

Address: Base + 0x038C Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36. Auxiliary Clock 1 Divider Configuration register (CGM_AC1_DC0)

Table 39. CGM_AC1_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 1 divider 0. 1 Enable auxiliary clock 1 divider 0.
DIV0	Divider 0 Division Value — The resultant CMU1 monitored clock will have a period DIV0 + 1 times that of auxiliary clock 1. If DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV0 field is ignored and the CMU1 monitored clock remains disabled.

5.6.8 Auxiliary Clock 2 Select Control register (CGM_AC2_SC)

The Auxiliary Clock 2 Select Control register (CGM_AC2_SC) selects the current auxiliary clock 2 source.

Address: Base + 0x0390 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37. Auxiliary Clock 2 Select Control register (CGM_AC2_SC)

Table 40. CGM_AC2_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 2 Source Selection Control — This value selects the current source for auxiliary clock 2. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 0101 FMPLL_1 0110 reserved 0111 reserved 1000 FMPLL_1_D1 All other values are reserved.

5.6.9 Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

The Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0) controls the auxiliary clock 2 divider.

Address: Base + 0x0394 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 38. Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)

Table 41. CGM_AC2_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 2 divider 0. 1 Enable auxiliary clock 2 divider 0.
DIV0	Divider 0 Division Value — The resultant Safety Port clock will have a period DIV0 + 1 times that of auxiliary clock 2. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV 0 field is ignored and the SafetyPort clock remains disabled.

5.6.10 Auxiliary Clock 3 Select Control register (CGM_AC3_SC)

The Auxiliary Clock 3 Select Control register (CGM_AC3_SC) selects the current auxiliary clock 3 source.

Address: Base + 0x0398 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SELCTL				0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 39. Auxiliary Clock 3 Select Control register (CGM_AC3_SC)

Table 42. CGM_AC3_SC field descriptions

Field	Description
SELCTL	Auxiliary Clock 3 Source Selection Control — This value selects the current source for auxiliary clock 2. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 FMPLL_0 0101 FMPLL_1 0110 reserved 0111 reserved 1000 FMPLL_1_D1 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

5.6.11 Auxiliary Clock 3 Divider Configuration register (CGM_AC3_DC0)

The Auxiliary Clock 3 Divider Configuration (CGM_AC3_DC023) register controls the auxiliary clock 3 dividers.

Address: Base + 0x039C Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DE0	0	0	0	DIV0				0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 40. Auxiliary Clock 3 Divider Configuration registers (CGM_AC3_DC0)

Table 43. CGM_AC3_DC0 field descriptions

Field	Description
DE0	Divider 0 Enable 0 Disable auxiliary clock 3 divider 0. 1 Enable auxiliary clock 3 divider 0.
DIV0	Divider 0 Division Value — The resultant FlexRay clock will have a period DIV0 + 1 times that of auxiliary clock 3. If the DE0 is set to 0 (Divider 0 is disabled), any write access to the DIV 0 field is ignored and the FlexRay clock remains disabled.

5.7 Functional description

5.8 System clock generation

Figure 41 shows the block diagram of the system clock generation logic. The ME provides the system clock select and switch mask (see *Chapter 6: Mode Entry Module (ME)* for more details), and the MC_RGM provides the safe clock request (see *Chapter 8: Reset Generation Module (RGM)* for more details). The safe clock request forces the selector to select the IRC as the system clock and to ignore the system clock select.

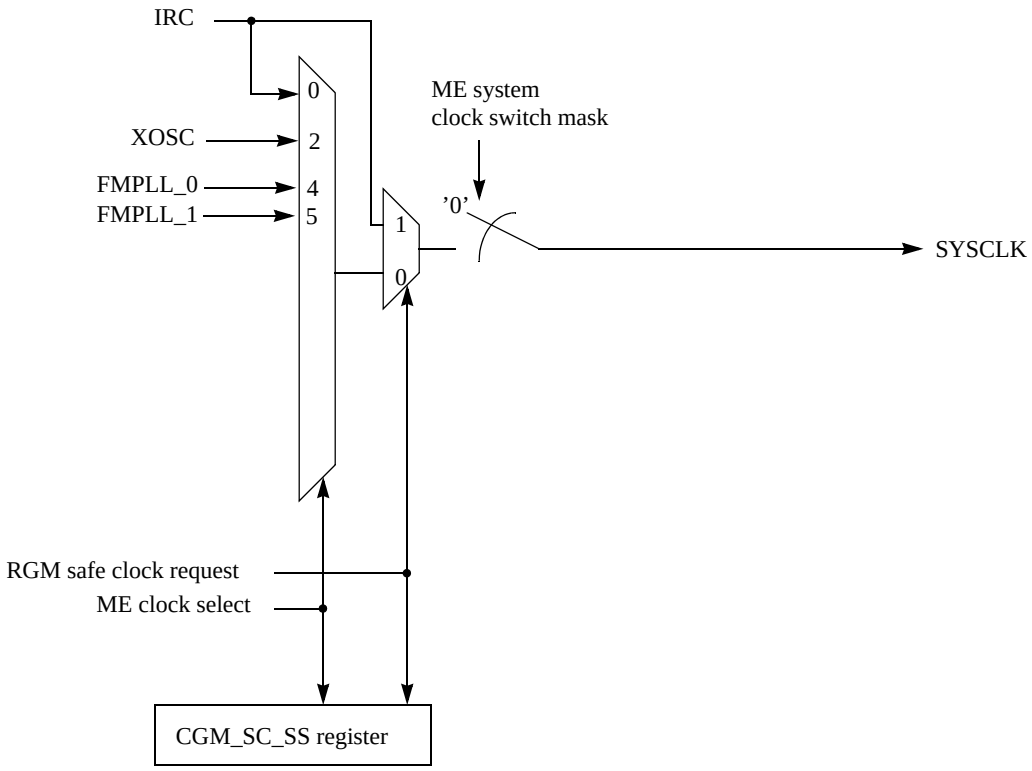


Figure 41. CGM system clock generation overview

5.8.1 System clock source selection

During normal operation, the system clock selection is controlled as follows:

- On a SAFE mode event, by RGM
- Otherwise, by the ME

5.8.2 System clock disable

During normal operation, the system clock can be disabled by the ME.

5.9 Auxiliary clock generation

Figure 42 shows the block diagram of the auxiliary clock generation logic. See the following sections for auxiliary clock selection control.

- *Section 5.6.4: Auxiliary Clock 0 Select Control register (CGM_AC0_SC)*
- *Section 5.6.6: Auxiliary Clock 1 Select Control register (CGM_AC1_SC)*
- *Section 5.6.8: Auxiliary Clock 2 Select Control register (CGM_AC2_SC)*
- *Section 5.6.10: Auxiliary Clock 3 Select Control register (CGM_AC3_SC)*

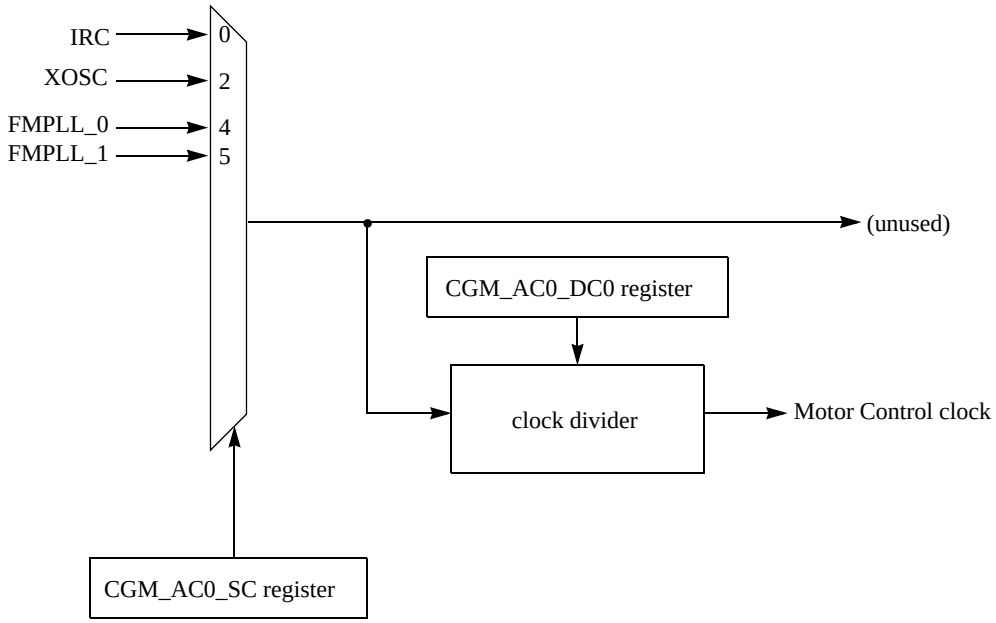


Figure 42. CGM auxiliary clock 0 generation overview

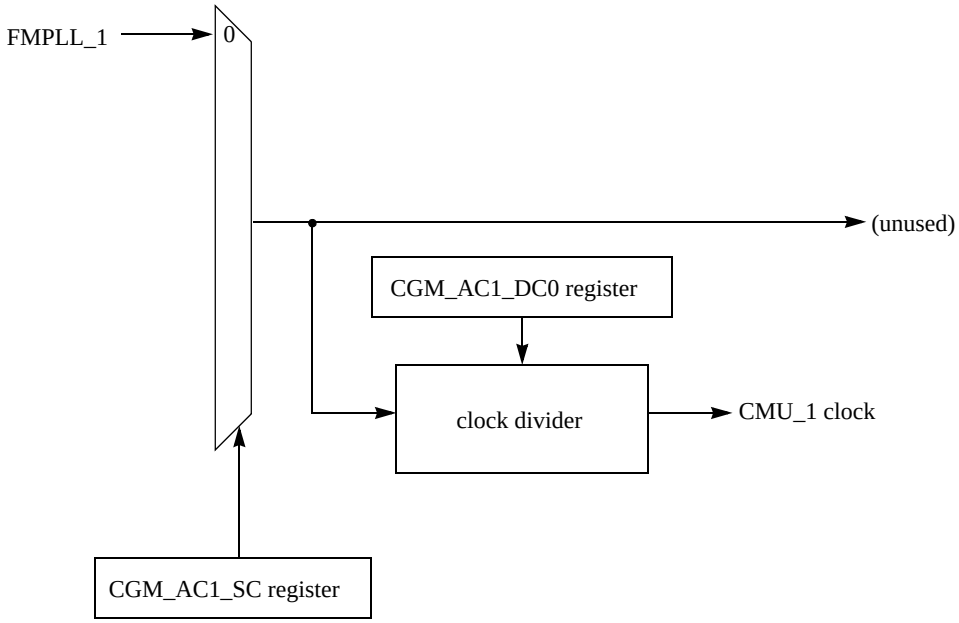


Figure 43. CGM auxiliary clock 1 generation overview

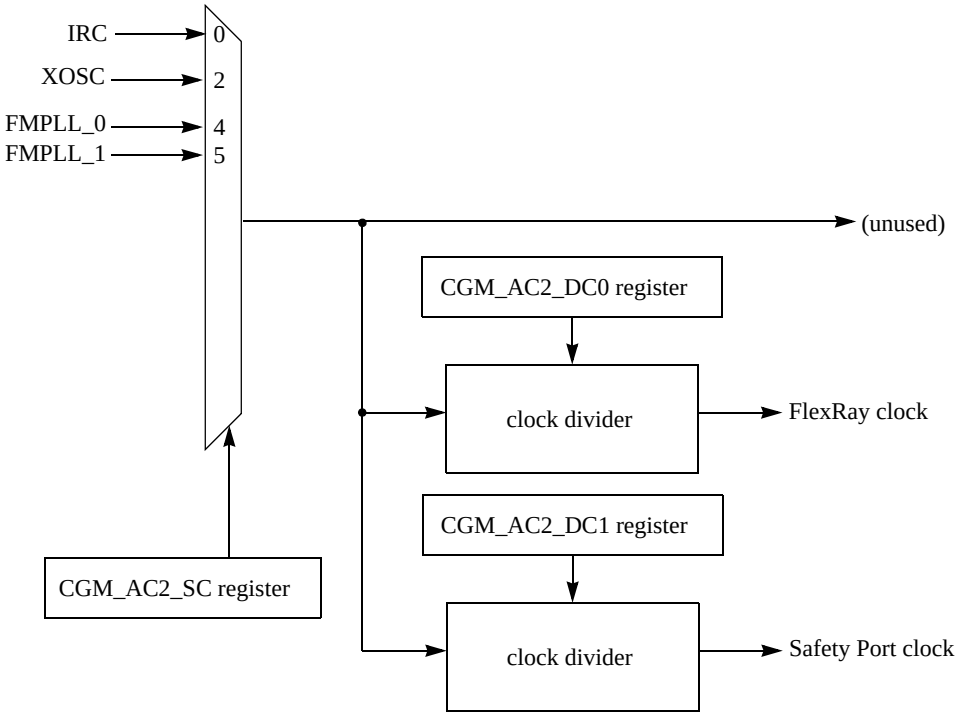


Figure 44. CGM auxiliary clock 2 generation overview

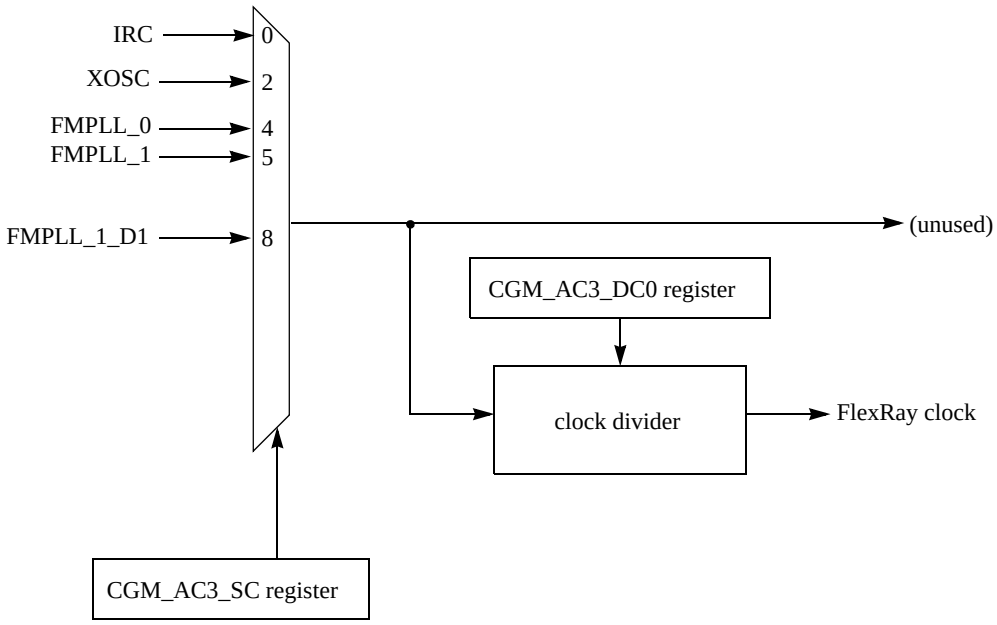


Figure 45. CGM auxiliary clock 3 generation overview

5.9.1 Auxiliary clock source selection

The auxiliary clock selection is done via the CGM_AC0…3_SC registers. If software selects an unavailable source, the previous selection remains, and the register content does not change.

5.9.2 Dividers functional description

Dividers are utilized for the generation of divided system and peripheral clocks. The CGM has the CGM_SC-DC control registers for built-in dividers:

- *Section 5.6.5: Auxiliary Clock 0 Divider Configuration register (CGM_AC0_DC0)*
- *Section 5.6.7: Auxiliary Clock 1 Divider Configuration register (CGM_AC1_DC0)*
- *Section 5.6.9: Auxiliary Clock 2 Divider Configuration Register (CGM_AC2_DC0)*
- *Section 5.6.11: Auxiliary Clock 3 Divider Configuration register (CGM_AC3_DC0)*

The reset value of all counters is 1. If a divider has its DE bit in the respective configuration register set to 0 (the divider is disabled), any value in its DIV*n* field is ignored.

5.10 Output clock multiplexing

The CGM contains a multiplexing function for a number of clock sources that can then be utilized as output clock sources. The selection is done via the CGM_OCDS_SC register.

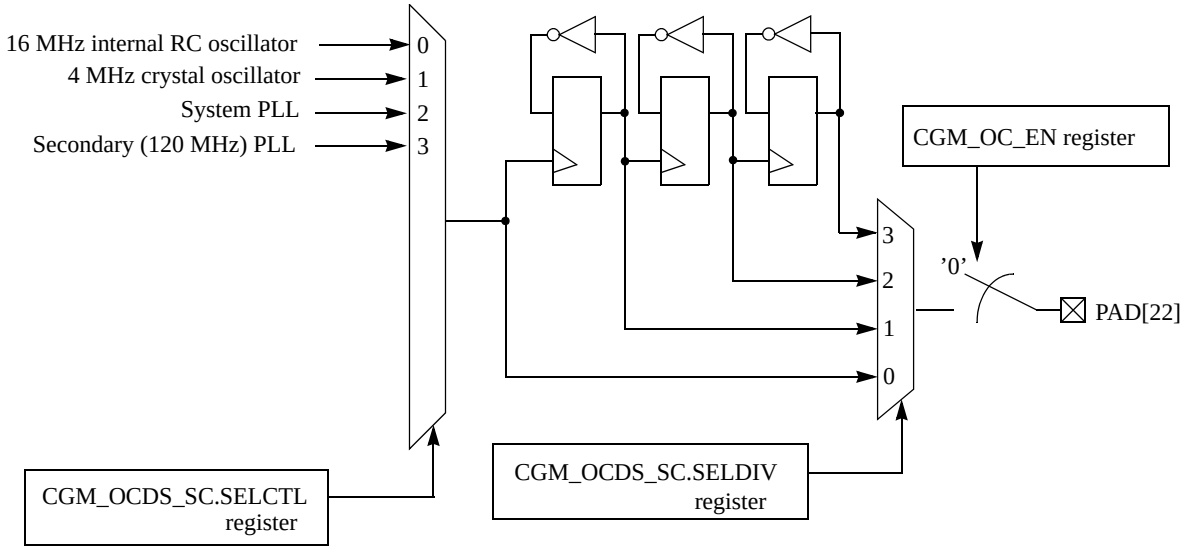


Figure 46. CGM output clock multiplexer and PAD[22] generation

5.11 Output clock division selection

The CGM provides the following output signals for the output clock generation:

- PAD[22] (see *Figure 46*). This signal is generated by utilizing one of the 3-stage ripple counter outputs or the selected signal without division. The non-divided signal is not guaranteed to be 50% duty cycle by the CGM.
- The CGM also has an Output Clock Enable register (see *Section 5.6.1: Output Clock Enable register (CGM_OC_EN)*), which contains the output clock enable/disable control bit.

6 Mode Entry Module (ME)

6.1 Introduction

This chapter describes the Introduction, which includes the functionality, pin description, and registers of the ME module.

6.1.1 Overview

The ME controls the device mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application.

Figure 47 shows the ME block diagram.

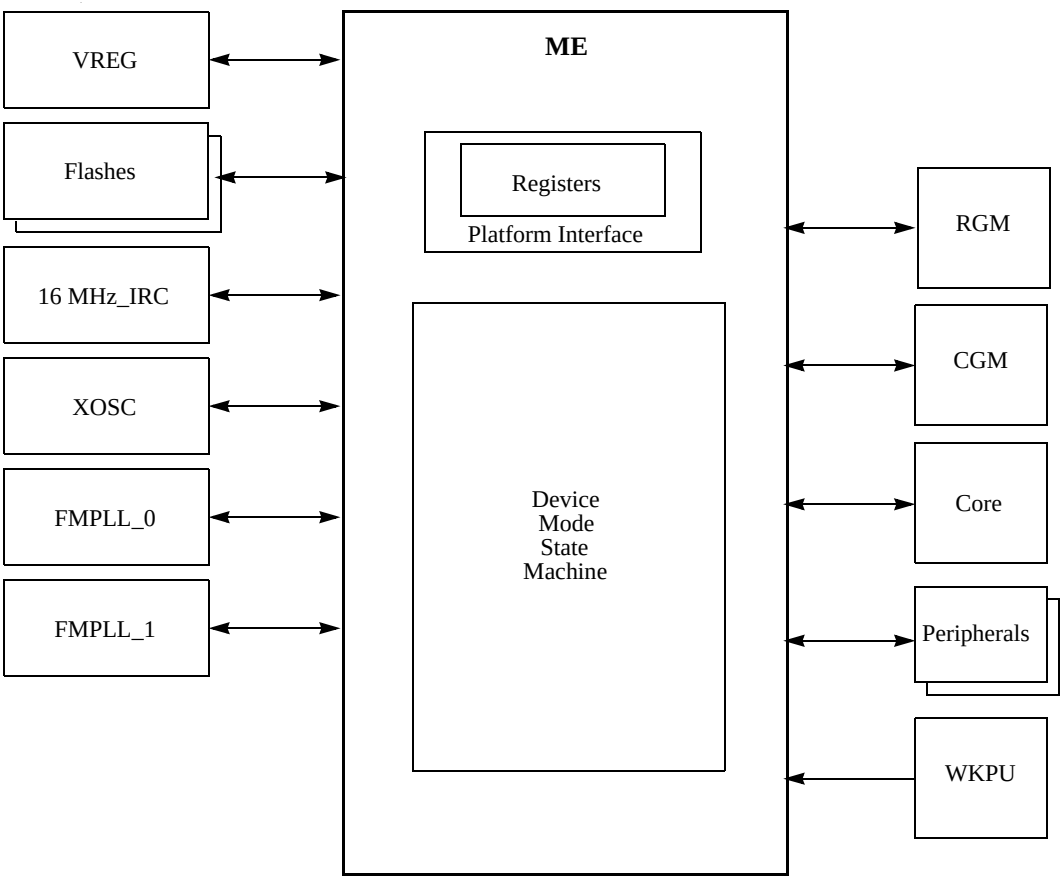


Figure 47. ME block diagram

6.1.2 Features

The ME includes the following features:

- Control of the available modes by the ME_ME register
- Definition of various device mode configurations by the ME_<*mode*>_MC registers
- Control of the actual device mode by the ME_MCTL register
- Capture of the current mode and various resource status within the contents of the ME_GS register
- Optional generation of various mode transition interrupts
- Status bits for each cause of invalid mode transitions
- Peripheral clock gating control based on the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL0…143 registers
- Capture of current peripheral clock gated/enabled status

6.1.3 Modes of Operation

6.1.3.1 Modes Overview

The ME is based on several device modes corresponding to different usage models of the device. Each mode is configurable and can define a policy for energy and processing power management to fit particular system requirements. An application can easily switch from one mode to another depending on the current needs of the system. The operating modes controlled by the ME are divided into system and user modes. The system modes are modes such as RESET, DRUN, SAFE, and TEST. These modes aim to ease the configuration and monitoring of the system. The user modes are modes such as RUN0…3, HALT0, and STOP0, which can be configured to meet the application requirements in terms of energy management and available processing power. The modes DRUN, SAFE, TEST, and RUN0…3 are the device software running modes.

Table 44 describes the ME modes.

Table 44. ME Mode Descriptions

Name	Description	Entry	Exit
RESET	This is a device-wide virtual mode in which the application is not active. The system remains in this mode until all resources are available for the embedded software to take control of the device. It manages hardware initialization of device configuration, voltage regulators, oscillators, PLLs, and flash modules.	System reset assertion from MC_RGM	System reset deassertion from MC_RGM
DRUN	This is the entry mode for the embedded software. It provides full accessibility to the system and enables the configuration of the system at startup. It provides the unique gate to enter USER modes. BAM when present is executed in DRUN mode.	System reset deassertion from MC_RGM, software request from SAFE, TEST and RUN0…3	System reset assertion, RUN0…3, TEST via software, SAFE via software or hardware failure.
SAFE	This is a device-wide service mode that can be entered on the detection of a recoverable error. It forces the system into a pre-defined safe configuration from which the system may try to recover.	Hardware failure, software request from DRUN, TEST, and RUN0…3	System reset assertion, DRUN via software

Table 44. ME Mode Descriptions(Continued)

Name	Description	Entry	Exit
TEST	This is a device-wide service mode that provides a control environment for device self-test. It may enable the application to run its own self-test like flash checksum, memory BIST etc.	Software request from DRUN	System reset assertion, DRUN via software
RUN 0…3	These are software running modes where most processing activity is done. These various run modes allow to enable different clock and power configurations of the system with respect to each other.	Software request from DRUN, interrupt event from HALT0, interrupt or wakeup event from STOP0	System reset assertion, SAFE via software or hardware failure, other RUN0…3 modes, HALT0, STOP0
HALT0	This is a reduced-activity low-power mode during which the clock to the core is disabled. It can be configured to switch off analog peripherals like PLL, flash, main regulator etc. for efficient power management at the cost of higher wakeup latency.	Software request from RUN0…3	System reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event
STOP0	This is an advanced low-power mode during which the clock to the core is disabled. It may be configured to switch off most of the peripherals including oscillator for efficient power management at the cost of higher wakeup latency.	Software request from RUN0…3	System reset assertion, SAFE on hardware failure, RUN0…3 on interrupt event or wakeup event

6.2 External signal description

The ME has no connections to any external pins.

6.3 Memory map and registers description

The ME base address is 0xC3FD_C000. The ME contains registers for:

- Mode selection and status reporting
- Mode configuration
- Mode transition interrupts status and mask control
- Scalable number of peripheral sub-mode selection and status reporting

6.3.1 Register summary

Table 45. ME registers

Offset from ME_BASE (0xC3FD_C000)	Register	Location
0x0000	ME_GS—Global Status register	*on page 160*
0x0004	ME_MCTL—Mode Control register	*on page 162*
0x0008	ME_ME—Mode Enable register	*on page 163*

Table 45. ME registers(Continued)

Offset from ME_BASE (0xC3FD_C000)	Register	Location
0x000C	ME_IS—Interrupt Status register	*on page 165*
0x0010	ME_IM—Interrupt Mask register	*on page 165*
0x0014	ME_IMTS—Invalid Mode Transition Status register	*on page 166*
0x0018	ME_DMTS—Debug Mode Transition Status register	*on page 166*
0x001C	Reserved	
0x0020	ME_RESET_MC—RESET Mode Configuration register	*on page 170*
0x00024	ME_TEST_MC—TEST Mode Configuration register	*on page 171*
0x0028	ME_SAFE_MC—SAFE Mode Configuration register	*on page 171*
0x002C	ME_DRUN_MC—DRUN Mode Configuration register	*on page 172*
0x0030	ME_RUN0_MC—RUN0 Mode Configuration register	*on page 172*
0x0034	ME_RUN1_MC—RUN1 Mode Configuration register	*on page 172*
0x0038	ME_RUN2_MC—RUN2 Mode Configuration register	*on page 172*
0x003C	ME_RUN3_MC—RUN3 Mode Configuration register	*on page 172*
0x0040	ME_HALT0_MC—HALT0 Mode Configuration register	*on page 173*
0x0044	Reserved	
0x0048	ME_STOP0_MC—STOP0 Mode Configuration register	*on page 174*
0x004C–0x005F	Reserved	
0x0060	ME_PS0—Peripheral Status 0 register	*on page 176*
0x0064	ME_PS1—Peripheral Status 1 register	*on page 176*
0x0068	ME_PS2—Peripheral Status 2 register	*on page 177*
0x006C–0x007F	Reserved	
0x0080	ME_RUN_PC0—Run Peripheral Configuration 0 register	*on page 178*
0x0084	ME_RUN_PC1—Run Peripheral Configuration 1 register	*on page 178*
0x0088	ME_RUN_PC2—Run Peripheral Configuration 2 register	*on page 178*
0x008C	ME_RUN_PC3—Run Peripheral Configuration 3 register	*on page 178*
0x0090	ME_RUN_PC4—Run Peripheral Configuration 4 register	*on page 178*
0x0094	ME_RUN_PC5—Run Peripheral Configuration 5 register	*on page 178*
0x0098	ME_RUN_PC6—Run Peripheral Configuration 6 register	*on page 178*
0x009C	ME_RUN_PC7—Run Peripheral Configuration 7 register	*on page 178*
0x00A0	ME_LP_PC0—Low-Power Peripheral Configuration 0 register	*on page 179*
0x00A4	ME_LP_PC1—Low-Power Peripheral Configuration 1 register	*on page 179*

Table 45. ME registers(Continued)

Offset from ME_BASE (0xC3FD_C000)	Register	Location
0x00A8	ME_LP_PC2—Low-Power Peripheral Configuration 2 register	*on page 179*
0x00AC	ME_LP_PC3—Low-Power Peripheral Configuration 3 register	*on page 179*
0x00B0	ME_LP_PC4—Low-Power Peripheral Configuration 4 register	*on page 179*
0x00B4	ME_LP_PC5—Low-Power Peripheral Configuration 5 register	*on page 179*
0x00B8	ME_LP_PC6—Low-Power Peripheral Configuration 6 register	*on page 179*
0x00BC	ME_LP_PC7—Low-Power Peripheral Configuration 7 register	*on page 179*
0x00C0–0x00C3	Reserved	
0x00C4	ME_PCTL4—DSPI0 Control register	*on page 179*
0x00C5	ME_PCTL5—DSPI1 Control register	*on page 179*
0x00C6	ME_PCTL6—DSPI2 Control register	*on page 179*
0x00C7	ME_PCTL7—DSPI3 Control register	*on page 179*
0x00C8–0x00CF	Reserved	
0x00D0	ME_PCTL16—FlexCAN0 Control register	*on page 179*
0x00D1–0x00D7	Reserved	
0x00D8	ME_PCTL24—FlexRay Control register	*on page 179*
0x00D9	Reserved	
0x00DA	ME_PCTL26—SafetyPort Control register	*on page 179*
0x00DB–0x00DF	Reserved	
0x00E0	ME_PCTL32—ADC0 Control register	*on page 179*
0x00E1	ME_PCTL33—ADC1 Control register	*on page 179*
0x00E2	Reserved	
0x00E3	ME_PCTL35—CTU0 Control register	*on page 179*
0x00E4–0x00E5	Reserved	
0x00E6	ME_PCTL38—eTimer0 Control register	*on page 179*
0x00E7	ME_PCTL39—eTimer1 Control register	*on page 179*
0x00E8	Reserved	
0x00E9	ME_PCTL41—FlexPWM0 Control register	*on page 179*
0x00EA–0x00EF	Reserved	
0x00F0	ME_PCTL48—LINFlex0 Control register	*on page 179*
0x00F1	ME_PCTL49—LINFlex1 Control register	*on page 179*
0x00F2–0x010B	Reserved	

Table 45. ME registers(Continued)

Offset from ME_BASE (0xC3FD_C000)	Register	Location
0x011C	ME_PCTL92—PIT_RTI Control register	*on page 179*
0x011D–0x3FFF	Reserved	

Note: Any access to unused registers as well as write accesses to read-only registers will not change register content, and will cause a transfer error.

6.3.2 Memory Map

Table 46. ME Memory Map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C000	ME_GS	R	S_CURRENT_MODE				S_MTRANS	0	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
		W																
		R	0	0	0	0	0	0	0	0	0	S_PLL0	S_XOSC0	S_16 MHz_IRC	S_SYSCLK			
		W																
0xC3FD_C004	ME_MCTL	R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	KEY															
		W																
0xC3FD_C008	ME_ME	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		W																
0xC3FD_C00C	ME_IS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
		W													w1c	w1c	w1c	w1c

Table 46. ME Memory Map(Continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C010	ME_IM	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
		W																
0xC3FD_C014	ME_IMTS	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
		W												w1c	w1c	w1c	w1c	w1c
0xC3FD_C018	ME_DMTS	R	0	0	0	0	0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
		W																
		R	0	PLL0_SC	XOSC0_SC		SSCLK_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0			CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
		W																
0xC3FD_C01C	Reserved																	
0xC3FD_C020	ME_RESET_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	0	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C024	ME_TEST_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																

Table 46. ME Memory Map(Continued)

Address	Name		0 / 16	1 / 17	2 / 18	3 / 19	4 / 20	5 / 21	6 / 22	7 / 23	8 / 24	9 / 25	10 / 26	11 / 27	12 / 28	13 / 29	14 / 30	15 / 31
0xC3FD_C028	ME_SAFE_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL01N	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C02C	ME_DRUN_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C030 … 0xC3FD_C03C	ME_RUN0…3_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C040	ME_HALT0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C044	Reserved																	

Table 46. ME Memory Map(Continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C048	ME_STOP0_MC	R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
		W																
		R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
		W																
0xC3FD_C04C … 0xC3FD_C05C	Reserved																	
0xC3FD_C060	ME_PS0	R						S_SafetyPort		S_FlexRay								S_FlexCAN0
		W																
		R									S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
		W																
0xC3FD_C064	ME_PS1	R	0	0	0		0	0		0	0	0	0	0			S_LIN_FLEX1	S_LIN_FLEX0
		W																
		R							S_FlexPWM0		S_eTimer1	S_eTimer0			S_CTU0		S_ADC1	S_ADC0
		W																
0xC3FD_C068	ME_PS2	R	0	0	0	S_PIT_RTI		0	0	0	0	S_SSCM	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0			0	0			0	0	0	0
		W																
0xC3FD_C06C	Reserved																	
0xC3FD_C070	Reserved																	
0xC3FD_C074 … 0xC3FD_C07C	Reserved																	

Table 46. ME Memory Map(Continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FD_C080 … 0xC3FD_C09C	ME_RUN_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
		W																
0xC3FD_C0A0 … 0xC3FD_C0BC	ME_LP_PC0…7	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
		W																
0xC3FD_C0C0 … 0xC3FD_C14F	ME_PCTL0…143	R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
		R	0	DBG_F	LP_CFG			RUN_CFG			0	DBG_F	LP_CFG			RUN_CFG		
		W																
0xC3FD_C150 … 0xC3FD_	Reserved																	

6.3.3 Registers description

Unless otherwise noted, all registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the ME_RUN_PC0 register may be accessed as a word at address offset 0xC3FD_C080, as a half-word at address offset 0xC3FD_C082, or as a byte at address offset 0xC3FD_C083.

6.3.3.1 Global Status register (ME_GS)

This register contains global mode status.

Address: Base + 0x0000 Access: User read, Supervisor read, Test read

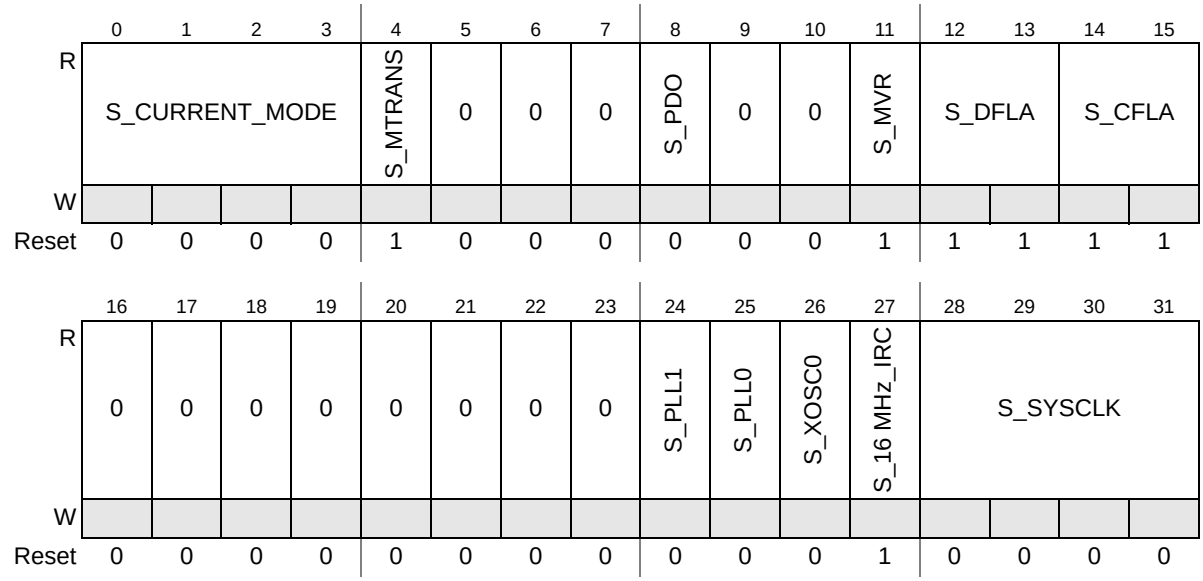


Figure 48. Global Status register (ME_GS)

Table 47. ME_GS field descriptions

Field	Description
S_CURRENT_MODE	Current device mode status 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT0 1001 reserved 1010 STOP0 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved
S_MTRANS	Mode transition status 0 Mode transition process is not active 1 Mode transition is ongoing
S_PDO	Output power-down status — This bit specifies output power-down status of I/Os. This bit is asserted whenever outputs of pads are forced to high impedance state or the pads power sequence driver is switched off. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state and pads power sequence driver is disabled. The inputs are level unchanged. In STOP0 mode, only pad power sequence driver is disabled but the state of the output is kept.

Table 47. ME_GS field descriptions(Continued)

Field	Description
S_MVR	Main voltage regulator status 0 Main voltage regulator is not ready 1 Main voltage regulator is ready for use
S_DFLA	Data flash availability status 00 Data flash is not available. 01 Data flash is in power-down mode. 10 Data flash is in low-power mode. 11 Data flash is in normal mode and available for use.
S_CFLA	Code flash availability status 00 Code flash is not available. 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode and available for use.
S_PLL1	Secondary PLL status 0 Secondary PLL is not stable. 1 Secondary PLL is providing a stable clock.
S_PLL0	System PLL status 0 System PLL is not stable. 1 System PLL is providing a stable clock.
S_XOSC0	XOSC status 0 XOSC is not stable. 1 XOSC is providing a stable clock.
S_16 MHz_IRC	IRC status 0 IRC is not stable. 1 IRC is providing a stable clock.
S_SYSCLK	System clock switch status — These bits specify the system clock currently used by the system. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 System PLL 0101 Secondary (120 MHz) PLL 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 System clock is disabled.

6.3.3.2 Mode Control register (ME_MCTL)

This register triggers software-controlled mode changes. Depending on the modes as enabled by the ME_ME register bits, configurations corresponding to unavailable modes are reserved and access to ME_<*mode*>_MC registers must respect this for successful mode requests.

Note: Byte and half-word write accesses are not allowed for this register as a predefined key is required to change its value.

Address : Base + 0x0004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	KEY															
W																
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

Figure 49. Mode Control register (ME_MCTL)

Table 48. ME_MCTL field descriptions

Field	Description
TARGET _MODE	Target device mode — These bits provide the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. These bits are automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT0 and STOP0 modes on hardware exit events, these are updated with the appropriate RUN0…3 mode value. 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT0 1001 reserved 1010 STOP0 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved
KEY	Control key — These bits enable write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return inverted key. KEY: 0101101011110000 (0x5AF0) INVERTED KEY: 1010010100001111 (0xA50F)

6.3.3.3 Mode Enable register (ME_ME)

This register allows a way to disable the device modes that are not required for a given device. RESET, SAFE, DRUN, and RUN0 modes are always enabled.

Address: Base + 0x0008 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 50. Mode Enable register (ME_ME)

Table 49. ME_ME field descriptions

Field	Description
STOP0	STOP0 mode enable 0 STOP0 mode is disabled. 1 STOP0 mode is enabled.
HALT0	HALT0 mode enable 0 HALT0 mode is disabled. 1 HALT0 mode is enabled.
RUN3	RUN3 mode enable 0 RUN3 mode is disabled. 1 RUN3 mode is enabled.
RUN2	RUN2 mode enable 0 RUN2 mode is disabled. 1 RUN2 mode is enabled.
RUN1	RUN1 mode enable 0 RUN1 mode is disabled. 1 RUN1 mode is enabled.
RUN0	RUN0 mode enable 0 RUN0 mode is disabled. 1 RUN0 mode is enabled.
DRUN	DRUN mode enable 0 DRUN mode is disabled. 1 DRUN mode is enabled.
SAFE	SAFE mode enable 0 SAFE mode is disabled. 1 SAFE mode is enabled.
TEST	TEST mode enable 0 TEST mode is disabled. 1 TEST mode is enabled.
RESET	RESET mode enable 0 RESET mode is disabled. 1 RESET mode is enabled.

6.3.3.4 Interrupt Status register (ME_IS)

This register provides the current interrupt status.

Address : Base + 0x000C Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	I_ICONF	I_IMODE	I_SAFE	I_MTC
W													w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 51. Interrupt Status register (ME_IS)

Table 50. ME_IS field descriptions

Field	Description
I_ICONF	Invalid mode configuration interrupt — This bit is set whenever a write operation to ME_<*mode*>_MC registers with invalid mode configuration is attempted. It is cleared by writing a 1 to this bit. 0 No invalid mode configuration interrupt occurred 1 Invalid mode configuration interrupt is pending
I_IMODE	Invalid mode interrupt — This bit is set whenever an invalid mode transition is requested. It is cleared by writing a 1 to this bit. 0 No invalid mode interrupt occurred 1 Invalid mode interrupt is pending
I_SAFE	SAFE mode interrupt — This bit is set whenever the device enters SAFE mode on hardware requests generated in the system. It is cleared by writing a 1 to this bit. 0 No SAFE mode interrupt occurred 1 SAFE mode interrupt is pending
I_MTC	Mode transition complete interrupt — This bit is set whenever the mode transition process completes (S_MTRANS transitions from 1 to 0). It is cleared by writing a 1 to this bit. This mode transition interrupt bit will not be set while entering low-power modes HALT0, or STOP0. 0 No mode transition complete interrupt occurred 1 Mode transition complete interrupt is pending

6.3.3.5 Interrupt Mask register (ME_IM)

This register controls whether an event generates an interrupt.

Address Base + 0x0010 Access: User read, Supervisor read/write, Test read/write
:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	M_ICONF	M_IMODE	M_SAFE	M_MTC
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 52. Interrupt Mask register (ME_IM)

Table 51. ME_IM field descriptions

Field	Description
M_ICONF	Invalid mode configuration interrupt mask 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled.
M_IMODE	Invalid mode interrupt mask 0 Invalid mode interrupt is masked 1 Invalid mode interrupt is enabled.
M_SAFE	SAFE mode interrupt mask 0 SAFE mode interrupt is masked 1 SAFE mode interrupt is enabled.
M_MTC	Mode transition complete interrupt mask 0 Mode transition complete interrupt is masked 1 Mode transition complete interrupt is enabled.

6.3.3.6 Invalid Mode Transition Status register (ME_IMTS)

This register provides the status bits for each cause of invalid mode interrupt.

Address Base + 0x0014 Access: User read, Supervisor read/write, Test read/write
:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	S_MTI	S_MRI	S_DMA	S_NMA	S_SEA
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 53. Invalid Mode Transition Status register (ME_IMTS)

Table 52. ME_IMTS field descriptions

Field	Description
S_MTI	Mode Transition Illegal status — This bit is set whenever a new mode is requested while some other mode transition process is active (S_MTRANS is 1). Please refer to *Section 6.4.5: Mode transition interrupts* for the exceptions to this behavior. It is cleared by writing a 1 to this bit. 0 Mode transition requested is not illegal. 1 Mode transition requested is illegal.
S_MRI	Mode Request Illegal status — This bit is set whenever the target mode requested is not a valid mode with respect to current mode. It is cleared by writing a 1 to this bit. 0 Target mode requested is not illegal with respect to current mode. 1 Target mode requested is illegal with respect to current mode.
S_DMA	Disabled Mode Access status — This bit is set whenever the target mode requested is one of those disabled modes determined by ME_ME register. It is cleared by writing a 1 to this bit. 0 Target mode requested is not a disabled mode. 1 Target mode requested is a disabled mode.
S_NMA	Non-existing Mode Access status — This bit is set whenever the target mode requested is one of those non-existing modes determined by ME_ME register. It is cleared by writing a 1 to this bit. 0 Target mode requested is an existing mode. 1 Target mode requested is a non-existing mode.
S_SEA	SAFE Event Active status — This bit is set whenever the device is in SAFE mode, SAFE event bit is pending and a new mode requested other than RESET/SAFE modes. It is cleared by writing a 1 to this bit. 0 No new mode requested other than RESET/SAFE while SAFE event is pending. 1 New mode requested other than RESET/SAFE while SAFE event is pending.

6.3.3.7 Debug Mode Transition Status register (ME_DMTS)

Address: Base + 0x0018 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MPH_BUSY	0	0	PMC_PROG	CORE_DBG	0	0	SMR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	PLL0_SC	XOSC0_SC	0	SSCLK_SC	SYSCLK_SW	DFLASH_SC	CFLASH_SC	CDP_PRPH_0_143	0	0	0	0	CDP_PRPH_64_95	CDP_PRPH_32_63	CDP_PRPH_0_31
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 54. Debug Mode Transition Status register (ME_DMTS)

This register provides the status of different factors which influence mode transitions. It is used to give an indication of why a mode transition indicated by ME_GS[S_MTRANS] may be taking longer than expected.

Note: The ME_DMTS register does not indicate whether a mode transition is ongoing. Therefore, some ME_DMTS bits may still be asserted after the mode transition has completed.

Table 53. ME_DMTS Field Descriptions

Field	Description
MPH_BUSY	ME/MC_PCU Handshake Busy indicator — This bit is set if the ME has requested a mode change from the MC_PCU and the MC_PCU has not yet responded. It is cleared when the MC_PCU has responded. 0 Handshake is not busy. 1 Handshake is busy.
PMC_PROG	MC_PCU Mode Change in Progress indicator — This bit is set if the MC_PCU is in the process of powering up or down power domains. It is cleared when all power-up/down processes have completed. 0 Power-up/down transition is not in progress. 1 Power-up/down transition is in progress.
CORE_DBG	Processor is in Debug mode indicator — This bit is set while the processor is in debug mode. 0 The processor is not in debug mode. 1 The processor is in debug mode.
SMR	SAFE mode request from MC_RGM is active indicator — This bit is set if a hardware SAFE mode request has been triggered. It is cleared when the hardware SAFE mode request has been cleared. 0 A SAFE mode request is not active. 1 A SAFE mode request is active.

Table 53. ME_DMTS Field Descriptions(Continued)

Field	Description
PLL0_SC	PLL0 State Change during mode transition indicator — This bit is set when the System PLL is requested to change its power up/down state. It is cleared when the System PLL has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
XOSC0_SC	XOSC0 State Change during mode transition indicator — This bit is set when the XOSC is requested to change its power up/down state. It is cleared when the XOSC has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
16 MHz_IRC_SC	16 MHz_IRC State Change during mode transition indicator — This bit is set when the IRC is requested to change its power up/down state. It is cleared when the IRC has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
SSCLK_SC	Secondary System Clock Sources State Change during mode transition indicator — This bit is set when a secondary system clock source is requested to change its power up/down state. It is cleared when all secondary system clock sources have completed their state changes. (A secondary system clock source is a system clock source other than 16 MHz_IRC, XOSC0, or PLL0.) 0 No state change is taking place. 1 A state change is taking place.
SYSCLK_SW	System Clock Switching pending status — 0 No system clock source switching is pending. 1 A system clock source switching is pending.
DFLASH_SC	DFLASH State Change during mode transition indicator — This bit is set when the DFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
CFLASH_SC	CFLASH State Change during mode transition indicator — This bit is set when the CFLASH is requested to change its power up/down state. It is cleared when the DFLASH has completed its state change. 0 No state change is taking place. 1 A state change is taking place.
CDP_PRPH_0_143	Clock Disable Process Pending status for Peripherals 0…143 — This bit is set when any peripheral has been requested to have its clock disabled. It is cleared when all the peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.
CDP_PRPH_64_95	Clock Disable Process Pending status for Peripherals 64…95 — This bit is set when any peripheral appearing in ME_PS2 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.

Table 53. ME_DMTS Field Descriptions(Continued)

Field	Description
CDP_PRPH_32_63	Clock Disable Process Pending status for Peripherals 32…63 — This bit is set when any peripheral appearing in ME_PS1 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.
CDP_PRPH_0_31	Clock Disable Process Pending status for Peripherals 0…31 — This bit is set when any peripheral appearing in ME_PS0 has been requested to have its clock disabled. It is cleared when all these peripherals which have been requested to have their clocks disabled have entered the state in which their clocks may be disabled. 0 No peripheral clock disabling is pending. 1 Clock disabling is pending for at least one peripheral.

6.3.3.8 RESET Mode Configuration register (ME_RESET_MC)

Address : Base + 0x0020 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 55. Invalid Mode Transition Status register (ME_IMTS)

This register configures system behavior during RESET mode. Please refer to *Table 54* for details.

6.3.3.9 TEST Mode Configuration register (ME_TEST_MC)

Address: Base + 0x0024 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 56. TEST Mode Configuration register (ME_TEST_MC)

This register configures system behavior during TEST mode. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses are not allowed to this register.

6.3.3.10 SAFE Mode Configuration register (ME_SAFE_MC)

Address : Base + 0x0028 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RCPLL1	RCPLL0	RCXOSC0	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 57. SAFE Mode Configuration register (ME_SAFE_MC)

This register configures system behavior during SAFE mode. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses are not allowed to this register.

6.3.3.11 DRUN Mode Configuration register (ME_DRUN_MC)

Address : Base + 0x002C

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 58. DRUN Mode Configuration register (ME_DRUN_MC)

This register configures system behavior during DRUN mode. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses are not allowed to this register.

6.3.3.12 RUN0…3 Mode Configuration registers (ME_RUN0…3_MC)

Address : Base + 0x0030 (ME_RUN0)
Base + 0x0034 (ME_RUN1)
Base + 0x0038 (ME_RUN2)
Base + 0x003C (ME_RUN3)

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 59. RUN0…3 Mode Configuration registers (ME_RUN0…3_MC)

This register configures system behavior during RUN0…3 modes. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses to this register are not allowed.

6.3.3.13 HALT0 Mode Configuration register (ME_HALT0_MC)

Address : Base + 0x0040 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 60. HALT0 Mode Configuration register (ME_HALT0_MC)

This register configures system behavior during HALT0 mode. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses are not allowed to this register.

6.3.3.14 STOP0 Mode Configuration register (ME_STOP0_MC)

Address : Base + 0x0048 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PLL1ON	PLL0ON	XOSC0ON	16 MHz_IRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 61. STOP0 Mode Configuration register (ME_STOP0_MC)

This register configures system behavior during STOP0 mode. Please refer to *Table 54* for details.

Note: Byte and half-word write accesses are not allowed to this register.

Table 54. ME_STOP0_MC field descriptions

Field	Description
PDO	I/O output power-down control — This bit controls the output power-down of I/Os. 0 No automatic safe gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high impedance state. The inputs are level unchanged.
MVRON	Main voltage regulator control — This bit specifies whether main voltage regulator is switched off or not while entering this mode. 0 Main voltage regulator is switched off. 1 Main voltage regulator is switched on.
DFLAON	Data flash power-down control — This bit specifies the operating mode of the data flash after entering this mode. 00 Reserved. 01 Data flash is in power-down mode. 10 Data flash is in low-power mode. 11 Data flash is in normal mode.
CFLAON	Code flash power-down control — This bit specifies the operating mode of the code flash after entering this mode. 00 Reserved. 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode.

Table 54. ME_STOP0_MC field descriptions(Continued)

Field	Description
PLL1ON	Secondary PLL control 0 Secondary PLL is switched off. 1 Secondary PLL is switched on.
PLL0ON	System PLL control 0 System PLL is switched off. 1 System PLL is switched on.
XOSC0ON	XOSC control 0 XOSC is switched off. 1 XOSC is switched on.
16 MHz_IRCON	IRC control 0 IRC is switched off. 1 IRC is switched on
SYSCLK	System clock switch control — These bits specify the s. ystem clock to be used by the system. 0000 IRC 0001 reserved 0010 XOSC 0011 reserved 0100 System PLL 0101 Secondary (120 MHz) PLL 0110 reserved 0111 reserved 1000 reserved 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 system clock is disabled.

6.3.3.15 Peripheral Status register 0 (ME_PS0)

Address: Base + 0x0060 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	S_SafetyPort	0	S_FlexRay	0	0	0	0	0	0	0	S_FlexCAN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 62. Peripheral Status register 0 (ME_PS0)

This register provides the status of the peripherals. Please refer to *Table 55* for details.

6.3.3.16 Peripheral Status register 1 (ME_PS1)

Address: Base + 0x0064 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	S_LIN_FLEX1	S_LIN_FLEX0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	S_FlexPWM0	0	S_eTimer1	S_eTimer0	0	0	S_CTU0	0	S_ADC1	S_ADC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 63. Peripheral Status register 1 (ME_PS1)

This register provides the status of the peripherals. Please refer to *Table 55* for details.

6.3.3.17 Peripheral Status register 2 (ME_PS2)

Address: Base + 0x0068 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	S_PIT_RTI	0	0	0	0	0	S_SSCM	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 64. Peripheral Status register 2 (ME_PS2)

This register provides the status of the peripherals. Please refer to *Table 55* for details.

Table 55. ME_PS0…4 field descriptions

Field	Description
S_<*periph*>	Peripheral status — These bits specify the current status of the peripherals in the system. If no peripheral is mapped on a particular position, the corresponding bit is always read as 0. 0 Peripheral is frozen. 1 Peripheral is active.

6.3.3.18 Run Peripheral Configuration registers (ME_RUN_PC0…7)

Address: Base + 0x0080 (ME_RUN_PC0)
Base + 0x0084 (ME_RUN_PC1)
Base + 0x0088 (ME_RUN_PC2)
Base + 0x008C (ME_RUN_PC3)
Base + 0x0090 (ME_RUN_PC0)
Base + 0x0094 (ME_RUN_PC1)
Base + 0x0098 (ME_RUN_PC2)
Base + 0x009C (ME_RUN_PC3)

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 65. Run Peripheral Configuration registers (ME_RUN_PC0…7)

These registers configure eight different types of peripheral behavior during run modes.

Table 56. ME_RUN_PC0…7 field descriptions

Field	Description
RUN3	Peripheral control during RUN3 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN2	Peripheral control during RUN2 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN1	Peripheral control during RUN1 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RUN0	Peripheral control during RUN0 0 Peripheral is frozen with clock gated 1 Peripheral is active
DRUN	Peripheral control during DRUN 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
SAFE	Peripheral control during SAFE 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

Table 56. ME_RUN_PC0…7 field descriptions(Continued)

Field	Description
TEST	Peripheral control during TEST 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
RESET	Peripheral control during RESET 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

6.3.3.19 Low-Power Peripheral Configuration registers (ME_LP_PC0…7)

Address: Base + 0x00A0 (ME_LP_PC0) Base + 0x00B0 (ME_LP_PC0)
Base + 0x00A4 (ME_LP_PC1) Base + 0x00B4 (ME_LP_PC1)
Base + 0x00A8 (ME_LP_PC2) Base + 0x00B8 (ME_LP_PC2)
Base + 0x00AC (ME_LP_PC3) Base + 0x00BC (ME_LP_PC3)
Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	STOP0	0	HALT0	0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 66. Low-Power Peripheral Configuration registers (ME_LP_PC0…7)

These registers configure eight different types of peripheral behavior during non-run modes.

Table 57. ME_LP_PC0…7 field descriptions

Field	Description
STOP0	Peripheral control during STOP0 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
HALT0	Peripheral control during HALT0 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

6.3.3.20 Peripheral Control registers (ME_PCTL0…143)

These registers select the configurations during run and non-run modes for each peripheral.

Address: See *Table 45* for list of implemented registers.

Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7
R	0	DBG_F	LP_CFG			RUN_CFG		
W								
Reset	0	0	0	0	0	0	0	0

Figure 67. Peripheral Control registers (ME_PCTL0…143)

Table 58. ME_PCTL0…143 field descriptions

Field	Description
DBG_F	Peripheral control in debug mode — This bit controls the state of the peripheral in debug mode. 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the device mode. 1 Peripheral is frozen if not already frozen in device modes. **Note:** This feature is useful to freeze the peripheral state while entering debug. For example, this may be used to prevent a reference timer from running while making a debug accesses.
LP_CFG	Peripheral configuration select for non-run modes — These bits associate a configuration as defined in the ME_LP_PC0...7 registers to the peripheral. 000 Selects ME_LP_PC0 configuration. 001 Selects ME_LP_PC1 configuration. 010 Selects ME_LP_PC2 configuration. 011 Selects ME_LP_PC3 configuration. 100 Selects ME_LP_PC4 configuration. 101 Selects ME_LP_PC5 configuration. 110 Selects ME_LP_PC6 configuration. 111 Selects ME_LP_PC7 configuration.
RUN_CFG	Peripheral configuration select for run modes — These bits associate a configuration as defined in the ME_RUN_PC0...7 registers to the peripheral. 000 Selects ME_RUN_PC0 configuration. 001 Selects ME_RUN_PC1 configuration. 010 Selects ME_RUN_PC2 configuration. 011 Selects ME_RUN_PC3 configuration. 100 Selects ME_RUN_PC4 configuration. 101 Selects ME_RUN_PC5 configuration. 110 Selects ME_RUN_PC6 configuration. 111 Selects ME_RUN_PC7 configuration.

6.4 Functional description

6.4.1 Mode transition request

The transition from one mode to another mode is normally handled by software by accessing the mode control ME_MCTL register. But in case of special events, mode transition can be automatically managed by hardware. In order to switch from one mode to another, the application should access ME_MCTL register twice by writing:

- The first time with the value of the key (0x5AF0) into the KEY bit field and the required target mode into the TARGET_MODE bit field
- the second time with the inverted value of the key (0xA50F) into the KEY bit field and the required target mode into the TARGET_MODE bit field.

Once a valid mode transition request is detected, the target mode configuration information is loaded from the corresponding ME_<*mode*>_MC register. The mode transition request may require a number of cycles depending on the programmed configuration, and software should check the S_CURRENT_MODE bit field and the S_MTRANS bit of the global status register ME_GS to verify when the mode has been correctly entered and the transition process has completed. For a description of valid mode requests, please refer to *Section 6.4.5: Mode transition interrupts*.

Any modification of the mode configuration register of the currently selected mode will not be taken into account immediately but on the next request to enter this mode. This means that transition requests such as RUN0…3 → RUN0…3, DRUN → DRUN, SAFE → SAFE, and TEST → TEST are considered valid mode transition requests. As soon as the mode request is accepted as valid, the S_MTRANS bit is set till the status in the ME_GS register matches the configuration programmed in the respective ME_<*mode*>_MC register.

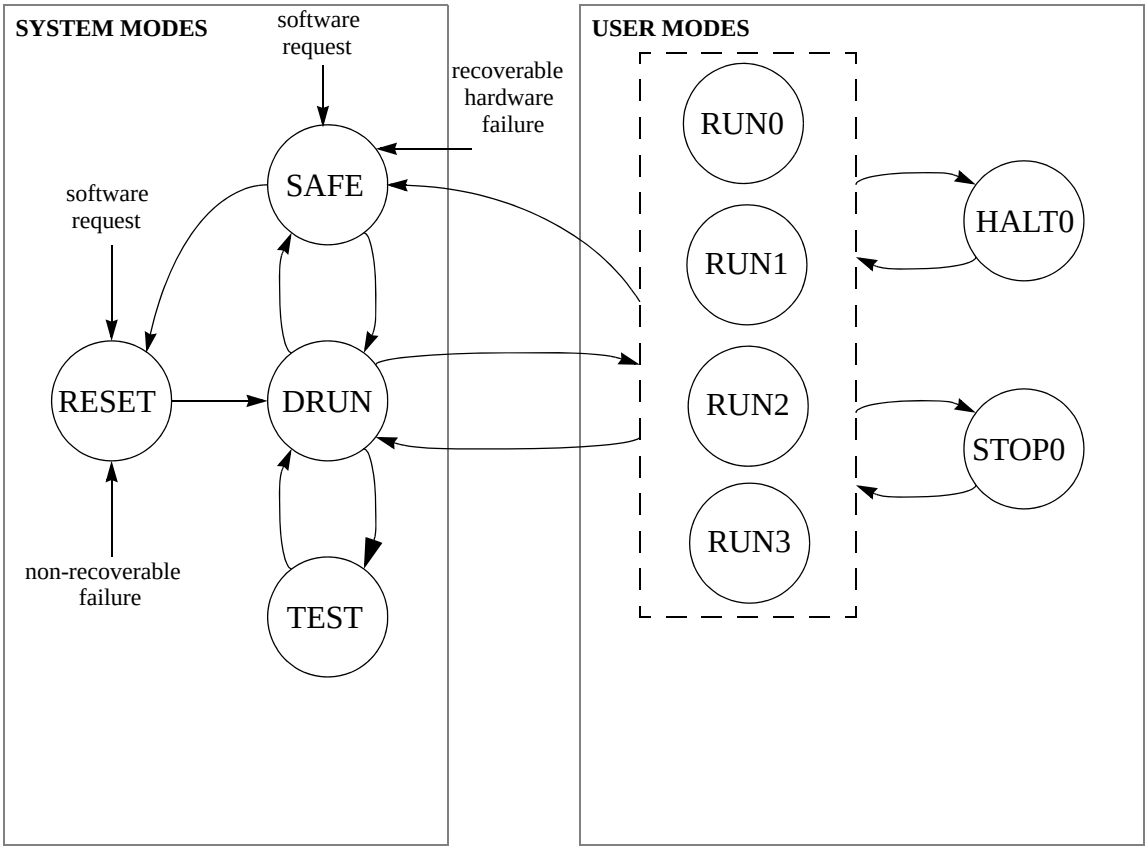


Figure 68. ME mode diagram

6.4.2 Mode details

6.4.2.1 RESET mode

The device enters this mode whenever the system reset is asserted by the MC_RGM. Transition to this mode is instantaneous, and the system remains in this mode until the reset sequence is finished.

6.4.2.2 DRUN mode

The device enters this mode on the following events:

- Automatically from RESET mode after completion of the reset sequence
- From RUN0…3, SAFE, or TEST mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0b0011

As soon as any of the above events has occurred, a DRUN mode transition request is generated. The mode configuration information for this mode is provided by the ME_DRUN_MC register. In this mode, the flashes, all clock sources, and the system clock configuration can be controlled by software as required. After system reset, the software execution starts with the default configuration selecting the IRC as the system clock.

This mode is intended to be used by software to initialize all registers as per the system needs.

Note: As flashes can be configured in low-power or power-down state in this mode, software must ensure that the code executes from RAM before changing to this mode.

6.4.2.3 SAFE mode

The device enters this mode on the following events:

- From DRUN, RUN0…3, or TEST mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0b0011
- From any mode except RESET due to a SAFE mode request generated by the MC_RGM because of some hardware failure in the system (see *Chapter 8: Reset Generation Module (RGM)* for details)

As soon as any of the above events has occurred, a SAFE mode transition request is generated. The mode configuration information for this mode is provided by the ME_SAFE_MC register. This mode has a pre-defined configuration, and the IRC is selected as the system clock.

If the SAFE mode is requested by software while some other mode transition process is ongoing, the new target mode becomes the SAFE mode regardless of other pending requests. In this case, the new mode request is not interpreted as an invalid request.

Note: If software requests to change to the SAFE mode and then requests to change back to the parent mode before the mode transition is completed, the device's final mode after mode transition will be the parent mode. However, this is not recommended software behavior. It is recommended for software to wait until the S_MTRANS bit is cleared after requesting a change to SAFE before requesting another mode change.

As long as a SAFE event is active, the system remains in the SAFE mode and no write access is allowed to the ME_MCTL register.

This mode is intended to be used by software to assess the severity of the cause of failure and then to either

- Re-initialize the device via the DRUN mode, or
- Completely reset the device via the RESET mode.

If the outputs of the system I/Os need to be forced to a high impedance state upon entering this mode, the ME_SAFE_MC[PDO] bit should be set. The input levels remain unchanged.

6.4.2.4 TEST mode

The device enters this mode from the DRUN mode when the ME_MCTL[TARGET_MODE] bit field is written with 0b0001.

As soon as this event occurs, a TEST mode transition request is generated. The mode configuration information for this mode is provided by the ME_TEST_MC register. Except for the main voltage regulator, all resources of the system are configurable in this mode. The system clock to the whole system can be stopped by programming the SYSCLK bit field to 0b1111, and in this case, the only way to exit this mode is via a device reset.

This mode is intended to be used by software to execute on-chip test routines.

Note: As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

6.4.2.5 RUN0…3 modes

The device enters one of these modes on the following events:

- From the DRUN another RUN0…3 mode when the TARGET_MODE bit field of the ME_MCTL register is written with 0b0100…0111
- From the HALT0 mode when an interrupt event occurs
- From the STOP0 mode when an interrupt or wakeup event occurs

As soon as any of the above events occur, a RUN0…3 mode transition request is generated. The mode configuration information for these modes is provided by ME_RUN0…3_MC registers. In these modes, the flashes, all clock sources, and the system clock configuration can be controlled by software as required.

These modes are intended to be used by software to execute application routines.

Note: As flash modules can be configured to a low-power or power-down state in these modes, software must ensure that the code will execute from RAM before it changes to this mode.

6.4.2.6 HALT0 mode

The device enters this mode from one of the RUN0…3 modes when the TARGET_MODE bit field of the ME_MCTL register is written with 0b1000.

As soon as this occurs, a HALT0 mode transition request is generated. The mode configuration information for this mode is provided by ME_HALT0_MC register. This mode is quite configurable, and the ME_HALT0_MC register should be programmed according to the system needs. The flashes can be put in power-down mode as needed. If there is a HALT0 mode request while an interrupt request is active, the device mode does not change, and an invalid mode interrupt is not generated.

This mode is intended as a first level low-power mode with

- The core clock frozen
- Only a few peripherals running

and to be used by software to wait until it is required to do something and then to react quickly (i.e., within a few system clock cycles of an interrupt event).

6.4.2.7 STOP0 mode

The device enters this mode from one of the RUN0…3 modes when the TARGET_MODE bit field of the ME_MCTL register is written with 0b1010.

As soon as this occurs, a STOP0 mode transition request is generated. The mode configuration information for this mode is provided by the ME_STOP0_MC register. This mode is fully configurable, and the ME_STOP0_MC register should be programmed according to the system needs. The PLL0 is switched off in this mode. The flashes can be put in power-down mode as needed. If there is a STOP0 mode request while any interrupt or wakeup event is active, the device mode does not change, and an invalid mode interrupt is not generated.

This can be used as an advanced low-power mode with the core clock frozen and almost all peripherals stopped.

This mode is intended as an advanced low-power mode with

- The core clock frozen
- Almost all peripherals stopped

and to be used by software to wait until it is required to do something with no need to react quickly (e.g., allow for the system clock source to be re-started).

This mode can be used to stop all clock sources, thus preserving the device status. When exiting the STOP0 mode, the IRC clock is selected as the system clock until the target clock is available.

6.4.3 Mode transition process

The process of mode transition follows the following steps in a pre-defined manner depending on the current device mode and the requested target mode. In many cases of mode transition, not all steps need to be executed based on the mode control information, and some steps may not be valid according to the mode definition itself.

6.4.3.1 Target mode request

The target mode is requested by accessing the ME_MCTL register with the required keys. This mode transition request by software must be a valid request satisfying a set of pre-defined rules to initiate the process. If the request fails to satisfy these rules, it is ignored, and the TARGET_MODE bit field is not updated. An optional interrupt can be generated for invalid mode requests. Refer to *Section 6.4.5: Mode transition interrupts* for details.

In the case of mode transitions occurring because of hardware events such as a reset, a SAFE mode request, or interrupt requests and wakeup events to exit from low-power modes, the TARGET_MODE bit field of the ME_MCTL register is automatically updated with the appropriate target mode. The mode change process start is indicated by the setting of the mode transition status bit S_MTRANS of the ME_GS register.

A RESET mode requested via the ME_MCTL register is passed to the MC_RGM, which generates a global system reset and initiates the reset sequence. The RESET mode request has the highest priority, and the ME is kept in the RESET mode during the entire reset sequence.

The SAFE mode request has the next highest priority (after reset) that can be generated by software via the ME_MCTL register from all software running modes, including DRUN, RUN0…3, and TEST; or by the MC_RGM after the detection of system hardware failures, which can occur in any mode.

6.4.3.2 Target mode configuration loading

On completion of the target mode request (see *Section 6.4.3.1: Target mode request*), the target mode configuration from the ME_<*target mode*>_MC register is loaded to start the resources (voltage sources, clock sources, flashes, pads, etc.) control process.

An overview of resource control possibilities for each mode is shown in *Table 59*. A '√' indicates that a given resource is configurable for a given mode.

Table 59. ME resource control overview

Resource	Mode						
	RESET	TEST	SAFE	DRUN	RUN0…3	HALT0	STOP0
16 MHz_IRC	on	√ on	on	on	on	√ on	√ on
XOSC0	off	√ off	off	√ off	√ off	√ off	√ off
PLL0	off	√ off	off	√ off	√ off	√ off	off
PLL1	off	√ off	off	√ off	√ off	√ off	√ off
CFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
DFLASH	normal	√ normal	normal	√ normal	√ normal	√ low-power	√ power-down
MVREG	on	on	on	on	on	√ on	√ on
PDO	off	√ off	√ on	off	off	off	√ off

6.4.3.3 Peripheral clocks disable

On completion of the target mode request (see *Section 6.4.3.1: Target mode request*), the ME requests each peripheral to enter its stop mode when the peripheral is configured to be disabled via the target mode, the peripheral configuration registers ME_RUN_PC0…7 and ME_LP_PC0…7, and the peripheral control registers ME_PCTL0…143.

Each peripheral acknowledges its stop mode request after closing its internal activity. The ME then disables the corresponding clock(s) to this peripheral.

In the case of a SAFE mode transition request, the ME does not wait for the peripherals to acknowledge the stop requests. The SAFE mode clock gating configuration is applied immediately regardless of the status of the peripherals' stop acknowledges.

Please refer to *Section 6.4.6: Peripheral clock gating* for more details.

Each peripheral that may block or disrupt a communication bus to which it is connected ensures that these outputs are forced to a safe or recessive state when the device enters the SAFE mode.

6.4.3.4 Processor low-power mode entry

If, on completion of the peripheral clocks disable (see *Section 6.4.3.3: Peripheral clocks disable*), the mode transition is to the HALT0 mode, the ME requests the processor to enter its halted state. The processor acknowledges its halt state request after completing all outstanding bus transactions.

If, on completion of the peripheral clocks disable (see *Section 6.4.3.3: Peripheral clocks disable*), the mode transition is to the STOP0 mode, the ME requests the processor to enter its stopped state. The processor acknowledges its stop state request after completing all outstanding bus transactions.

6.4.3.5 Processor and system memory clock disable

If, on completion of the processor low-power mode entry (see *Section 6.4.3.4: Processor low-power mode entry*), the mode transition is to the HALT0 or STOP0 mode and the processor is in its appropriate halted or stopped state, the ME disables the processor and system memory clocks to achieve further power saving.

The clocks to the processor and system memories are unaffected for all transitions between software running modes including DRUN, RUN0…3, and SAFE.

Warning: Clocks to the whole device including the processor and system memories can be disabled in TEST mode.

6.4.3.6 Clock sources switch-on

On completion of the processor low-power mode entry (see *Section 6.4.3.4: Processor low-power mode entry*), the ME controls all clock sources that affect the system clock based on the <*clock source*>ON bits of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers. The following system clock sources are controlled at this step:

- IRC
- XOSC
- Secondary PLL

Note: The system PLLneeds the main voltage regulator to be stable. Therefore, it is not controlled by this step.

The clock sources that are required by the target mode are switched on. The duration required for the output clocks to be stable depends on the type of source, and all further steps of mode transition depending on one or more of these clocks waits for the stable status of the respective clocks. The availability status of these system clocks is updated in the S_<*clock source*> bits of ME_GS register.

The clock sources that need to be switched off are unaffected during this process in order to not disturb the system clock, which might require one of these clocks before switching to a different target clock.

6.4.3.7 Flash modules switch-on

On completion of the main voltage regulator switch-on, if a flash module needs to be switched to normal mode from its low-power or power-down mode based on the CFLAON and DFLAON bit fields of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, the ME requests the flash to exit from its low-power/power-down mode. When the flash modules are available for access, the S_CFLA and S_DFLA bit fields of the ME_GS register are updated to “11” by hardware.

6.4.3.8 PLL0 switch-on

On completion of the clock sources switch-on and main voltage regulator switch-on, if the PLL0 is to be switched on from the off state based on the PLL0ON bit of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, the ME requests the PLL0 digital interface to start the phase locking process and waits for the PLL0 to enter into the locked state. When the PLL0 enters the locked state and starts providing a stable output clock, the ME_GS[S_PLL0] bit is set.

6.4.3.9 Pad outputs on

On completion of the main voltage regulator switch, if the ME_<*target mode*>_MC[PDO] bit is cleared, then

- All pad outputs are enabled to return to their previous state
- The I/O pads power sequence driver is switched on

6.4.3.10 Peripheral clocks enable

Based on the current and target device modes, the peripheral configuration registers ME_RUN_PC0…7, ME_LP_PC0…7, and the peripheral control registers ME_PCTL0…143, the ME enables the clocks for selected modules as required. This step is executed only after the main voltage regulator switch-on process is completed.

6.4.3.11 Processor and memory clock enable

If the mode transition is from any of the low-power modes HALT0 orSTOP0 to RUN0…3, the clocks to the processor and system memories are enabled. The process of enabling these clocks is executed only after the flash modules switch-on (see *Section 6.4.3.7: Flash modules switch-on*) process is completed.

6.4.3.12 Processor low-power mode exit

If the mode transition is from any of the low-power modes HALT0or STOP0 to RUN0…3, the ME requests the processor to exit from its halted or stopped state. This step is executed only after the processor and memory clock enable (see *Section 6.4.3.11: Processor and memory clock enable*) process is completed.

6.4.3.13 System clock switching

Based on the SYSCLK bit field of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, if the target and current system clock configurations differ, the following method is implemented for clock switching.

- The target clock configuration for the IRC is effective only when the ME_GS[S_16 MHz_IRC] bit is set by hardware (i.e., the IRC has stabilized).
- The target clock configuration for the XOSC is effective only when the ME_GS[S_XOSC0] bit is set by hardware (i.e., the XOSC has stabilized).
- The target clock configuration for the system PLL is effective only when the ME_GS[S_PLL0] bit is set by hardware (i.e., the System PLL has stabilized).
- The target clock configuration for the Secondary (120 MHz) PLL is effective only when the ME_GS[S_PLL1] bit is set by hardware (i.e., the secondary PLL has stabilized).
- If the clock is to be disabled, the SYSCLK bit field should be programmed with 0b1111. This is possible only in the TEST mode.

The current system clock configuration can be observed by reading the ME_GS[S_SYSCLK] bit field, which is updated after every system clock switching. Until the target clock is available, the system uses the previous clock configuration.

System clock switching starts only after the following events occur:

- The clock sources switch-on process has completed if the target system clock source needs to be switched on (see *Section 6.4.3.6: Clock sources switch-on*)
- The FMPLL_0 switch-on process has completed if the target system clock is the system PLL (see "*Section 6.4.3.8: PLL0 switch-on*)
- The peripheral clocks disable process is completed in order not to change the system clock frequency before peripherals close their internal activities (see *Section 6.4.3.3: Peripheral clocks disable*)

An overview of system clock source selection possibilities for each mode is shown in *Table 60*. A '√' indicates that a given clock source is selectable for a given mode.

Table 60. ME system clock selection overview

System Clock Source	Mode						
	RESET	**TEST**	**SAFE**	**DRUN**	**RUN0…3**	**HALT0**	**STOP0**
IRC	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)	√ (default)
XOSC		√		√	√	√	√
System PLL		√		√	√	√	
Secondary (120 MHz) PLL		√		√	√	√	√
System clock is disabled		√(1)					√

1. Disabling the system clock during ***TEST*** mode will require a reset in order to exit ***TEST*** mode

6.4.3.14 Pad switch-off

If the ME_<*target mode*>_MC[PDO] bit = 1, then the outputs of the pads are forced to the high impedance state if the target mode is SAFE or TEST.

This step is executed only after the peripheral clocks disable process is completed (see *Section 6.4.3.3: Peripheral clocks disable*).

6.4.3.15 PLL0 switch-off

Based on the PLL0ON bit of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, if PLL0 is to be switched off, the ME requests the PLL0 to power down and updates its availability status bit S_PLL0 of the ME_GS register to 0. This step is executed only after the system clock switching process is completed (see *Section 6.4.3.13: System clock switching*).

6.4.3.16 Clock sources switch-off

Based on the device mode and the <*clock source*>ON bits of the ME_<*mode*>_MC registers, if a given clock source is to be switched off, the ME requests the clock source to power down and updates its availability status bit S_<*clock source*> of the ME_GS register to 0.

This step is executed only after

- System clock switching (see *Section 6.4.3.13: System clock switching*) process is completed in order not to lose the current system clock during mode transition.
- FMPLL_0 switch-off (see *Section 6.4.3.15: PLL0 switch-off*) as the input reference clock of the PLL0 can be among these clock sources. This is needed to prevent an unwanted lock transition when the PLL0 is switched on.

6.4.3.17 Flash switch-off

Based on the CFLAON and DFLAON bit fields of the ME_<*current mode*>_MC and ME_<*target mode*>_MC registers, if any of the flash modules is to be put in a low-power state, the ME requests the flash to enter the corresponding low-power state and waits for the deassertion of flash ready status signal. The exact low-power mode status of the flash modules is updated in the S_CFLA and S_DFLA bit fields of the ME_GS register. This step is executed only when processor and system memory clock disable (see *Section 6.4.3.5: Processor and system memory clock disable*) process is completed.

6.4.3.18 Current Mode Update

The current mode status bit field S_CURRENT_MODE of the ME_GS register is updated with the target mode bit field TARGET_MODE of the ME_MCTL register when:

- All the updated status bits in the ME_GS register match the configuration specified in the ME_<*target mode*>_MC register
- Power sequences are done
- Clock disable/enable process is finished
- Processor low-power mode (halt/stop) entry and exit processes are finished

Software can monitor the mode transition status by reading the ME_GS[S_MTRANS] bit. The mode transition latency can differ from one mode to another depending on the resources' availability before the new mode request and the target mode's requirements.

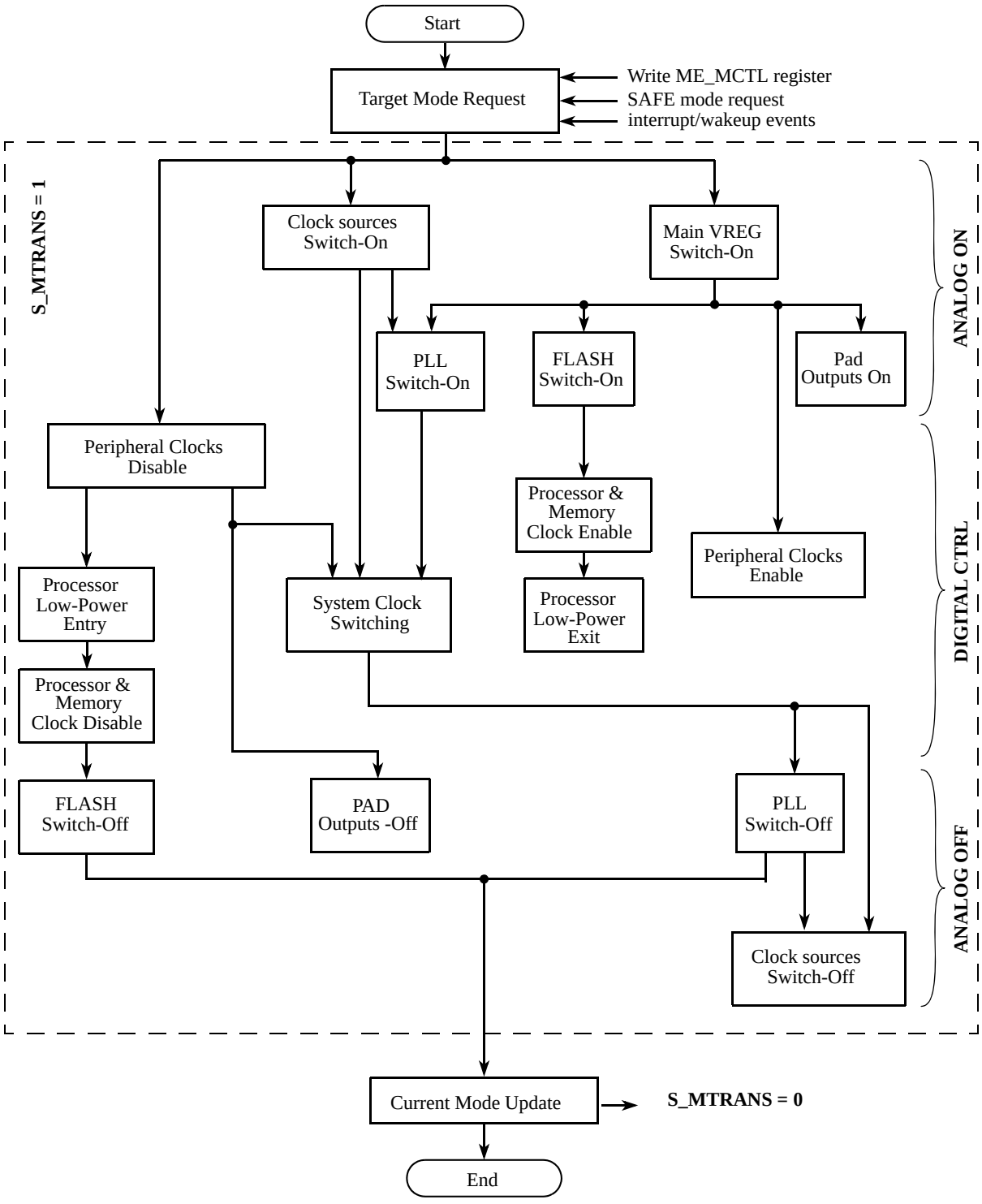


Figure 69. ME transition diagram

6.4.4 Protection of mode configuration registers

While programming the mode configuration registers ME_<*mode*>_MC, the following rules must be respected. Otherwise, the write operation is ignored and an invalid mode configuration interrupt may be generated.

- 16 MHz_IRC must be on if the system clock is one the following:
  - IRC
- XOSC0 must be on if the system clock is one of the following:
  - XOSC

Note: Software must ensure to switch on the clock source that provides the input reference clock to the PLL0. There is no automatic protection mechanism to check this in the ME.

- PLL0 must be on if the system clock is the System PLL.
- PLL1 must be on if the system clock is the Secondary (120 MHz) PLL.
- Configuration 0b00 for the CFLAON and DFLAON bit fields are reserved.
- System clock configurations marked as 'reserved' may not be selected.
- Configuration 0b1111 for the SYSCLK bit field is allowed only for TEST mode, and only in this case may all system clock sources be turned off.

Warning: If the system clock is stopped during TEST mode, the device can exit only via a system reset.

6.4.5 Mode transition interrupts

The following are the three interrupts related to mode transition implemented in the ME.

6.4.5.1 Invalid mode configuration interrupt

Whenever a write operation is attempted to the ME_<*mode*>_MC registers that violates the protection rules mentioned in *Section 6.4.4: Protection of mode configuration registers* the interrupt pending bit I_ICONF of the ME_IS register is set and an interrupt request is generated if the mask bit ME_IM[M_ICONF] = 1.

6.4.5.2 Invalid mode transition interrupt

The mode transition request is considered invalid under the following conditions:

- If the system is in the SAFE mode and the SAFE mode request from MC_RGM is active, and if the target mode requested is other than RESET or SAFE, then this new mode request is considered to be invalid, and the S_SEA bit of the ME_IMTS register is set.
- If the TARGET_MODE bit field of the ME_MCTL register is written with a value different from the specified mode values (i.e., a non-existing mode), an invalid mode transition event is generated. When such a non-existing mode is requested, the ME_IMTS[S_NMA] bit is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME_MCTL register.
- If some of the device modes are disabled as programmed in the ME_ME register, their respective configurations are considered reserved, and any access to the ME_MCTL register with those values results in an invalid mode transition request. When such a

disabled mode is requested, the ME_IMTS[S_DMA] bit is set. This condition is detected regardless of whether the proper key mechanism is followed while writing the ME_MCTL register.

- If the target mode is not a valid mode with respect to current mode, the mode request illegal status bit ME_IMTS[S_MRI] is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.
- If further new mode requests occur while a mode transition is in progress (ME_GS[S_MTRANS] = 1), the mode transition illegal status bit ME_IMTS[S_MTI] is set. This condition is detected only when the proper key mechanism is followed while writing the ME_MCTL register. Otherwise, the write operation is ignored.

Note: As the causes of invalid mode transitions may overlap at the same time, the priority implemented for invalid mode transition status bits of the ME_IMTS register in the order from highest to lowest is S_SEA, S_NMA, S_DMA, S_MRI, and S_MTI.

As an exception, the mode transition request is not considered as invalid under the following conditions:

- A new request is allowed to enter the RESET or SAFE mode irrespective of the mode transition status.
- As the exit of HALT0 and STOP0 modes depends on the interrupts of the system (which can occur at any instant), these requests to return to RUN0…3 modes are always valid.
- In order to avoid any unwanted lockup of the device modes, software can abort a mode transition by requesting the parent mode if, for example, the mode transition has not completed after a software determined ‘reasonable’ amount of time for whatever reason. The parent mode is the device mode before a valid mode request was made.
- Self-transition requests (e.g., RUN0 → RUN0) are not considered as invalid even when the mode transition process is active (i.e., S_MTRANS = 1). During the low-power mode exit process, if the system is not able to enter the respective RUN0…3 mode properly (i.e., all status bits of the ME_GS register match with configuration bits in the ME_<*mode*>_MC register), then software can only request the SAFE or RESET mode. Its not possible to request any other mode or to go back to the low-power mode again.

Note: If software requests to change to the SAFE or RESET mode and then requests to change back to the parent mode before the mode transition is completed, the device’s final mode after mode transition will be the parent mode. However, this is not recommended software behavior. It is recommended for software to wait until the S_MTRANS bit is cleared after requesting a change to SAFE or RESET before requesting another mode change.

Whenever an invalid mode request is detected, the interrupt pending bit ME_IS[I_IMODE] is set, and an interrupt request is generated if the mask bit ME_IM[M_IMODE] = 1.

6.4.5.3 SAFE mode transition interrupt

Whenever the system enters the SAFE mode as a result of a SAFE mode request from the MC_RGM due to a hardware failure, the interrupt pending bit ME_IS[I_SAFE] is set, and an interrupt is generated if the mask bit ME_IM[M_SAFE] = 1.

The SAFE mode interrupt pending bit can be cleared only when the SAFE mode request is deasserted by the MC_RGM (see the *Chapter 8: Reset Generation Module (RGM)* for details on how to clear a SAFE mode request). If the system is already in SAFE mode, any new SAFE mode request by the MC_RGM also sets the interrupt pending bit I_SAFE.

However, the SAFE mode interrupt pending bit is not set when the SAFE mode is entered by a software request (i.e., programming of ME_MCTL register).

6.4.5.4 Mode transition complete interrupt

Whenever the system completes a mode transition fully (the ME_IS[S_MTRANS] bit transitions from 1 to 0), the interrupt pending bit ME_IS[I_MTC] is set, and interrupt request is generated if the mask bit ME_IM[M_MTC] = 1. The interrupt bit I_MTC is not set when entering low-power modes HALT0 and STOP0 in order to avoid the same event requesting the exit of these low-power modes.

6.4.6 Peripheral clock gating

During all device modes, each peripheral can be associated with a particular clock gating policy determined by two groups of peripheral configuration registers.

The run peripheral configuration registers ME_RUN_PC0…7 are chosen only during the software running modes DRUN, TEST, SAFE, and RUN0…3. All configurations are programmable by software according to the needs of application. Each configuration register contains a mode bit that determines whether or not a peripheral clock is to be gated. Run configuration selection for each peripheral is done by the RUN_CFG bit field of the ME_PCTL0…143 registers.

The low-power peripheral configuration registers ME_LP_PC0…7 are chosen only during the low-power modes HALT0 and STOP0. All configurations are programmable by software according to the needs of the application. Each configuration register contains a mode bit that determines whether or not a peripheral clock is to be gated. Low-power configuration selection for each peripheral is done by the LP_CFG bit field of the ME_PCTL0…143 registers.

Any modifications to the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL0…143 registers do not affect the clock gating behavior until a new mode transition request is generated.

Whenever the processor enters a debug session during any mode, the following occurs for each peripheral:

- The clock is gated if the DBG_F bit of the associated ME_PCTL0…143 register is set. Otherwise, the peripheral clock gating status depends on the RUN_CFG and LP_CFG bits. Any further modifications of the ME_RUN_PC0…7, ME_LP_PC0…7, and ME_PCTL0…143 registers during a debug session will take affect immediately without requiring any new mode request.

6.4.7 Application example

Figure 70 shows an example application flow for requesting a mode change and then waiting until the mode transition has completed.

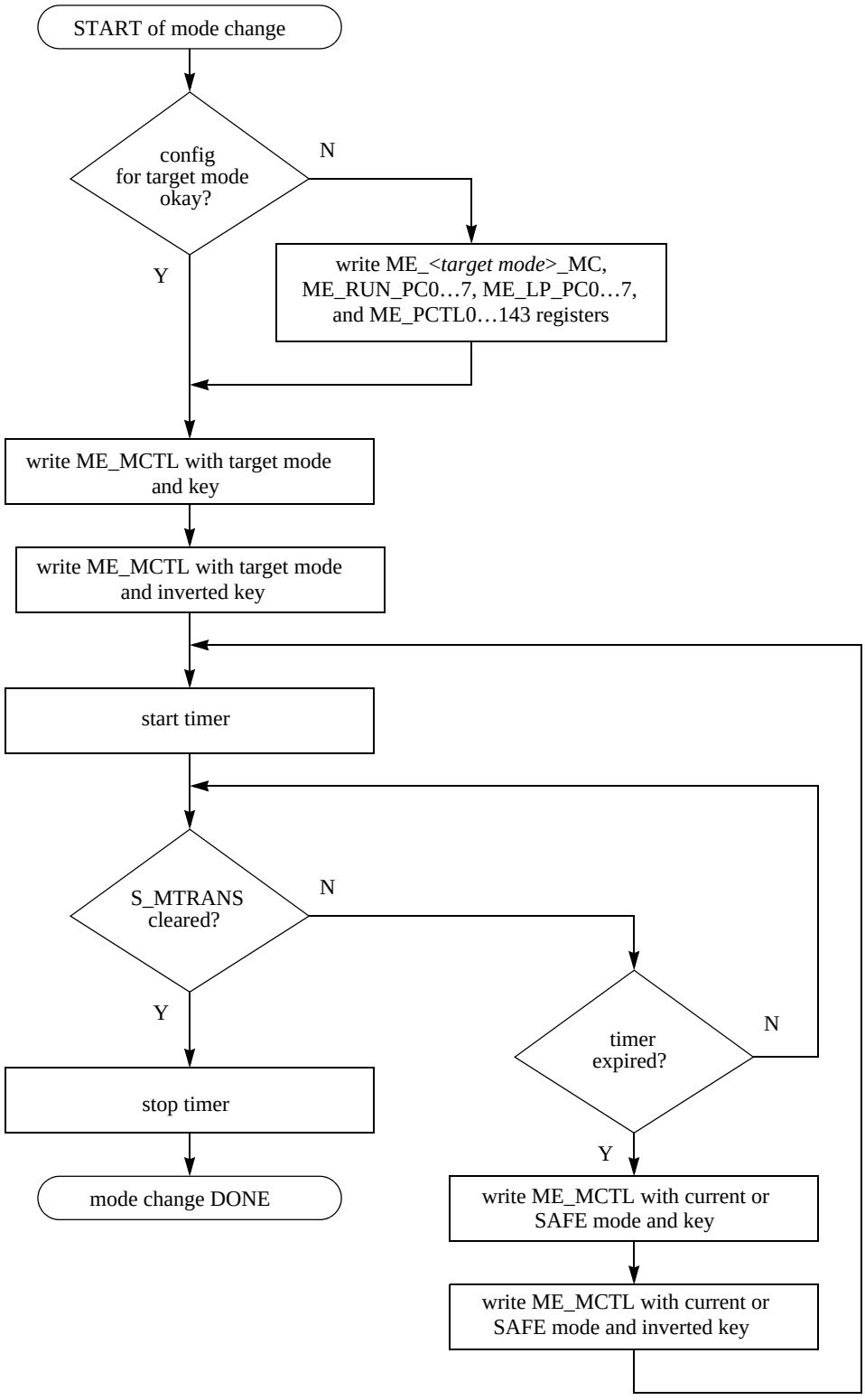


Figure 70. ME application example flow diagram

7 Power Control Unit (PCU)

7.1 Introduction

This chapter describes the Power Control Unit (PCU).

The power control unit (PCU) is used to reduce the overall device power consumption. Power can be saved by disconnecting parts of the device from the power supply via a power switching device. The modules on the device are grouped into sections called *power domains*.

When a power domain is disconnected from the supply, the power consumption is reduced to zero in that domain. Any status information of such a power domain is lost. When re-connecting a power domain to the supply voltage, the domain draws an increased current until the power domain reaches its operational voltage.

Power domains are controlled on a device mode basis. For each mode, software can configure whether a power domain is connected to the supply voltage (power-up state) or disconnected (power-down state).

On each mode change request, the PCU evaluates the power domain settings in the power domain configuration registers and initiates a power-down or a power-up sequence for each individual power domain. The power-up/down sequences are handled by finite state machines to ensure a smooth and safe transition from one power state to the other.

The power control unit (PCU) acts as a bridge for mapping the VREG peripheral to the PCU address space.

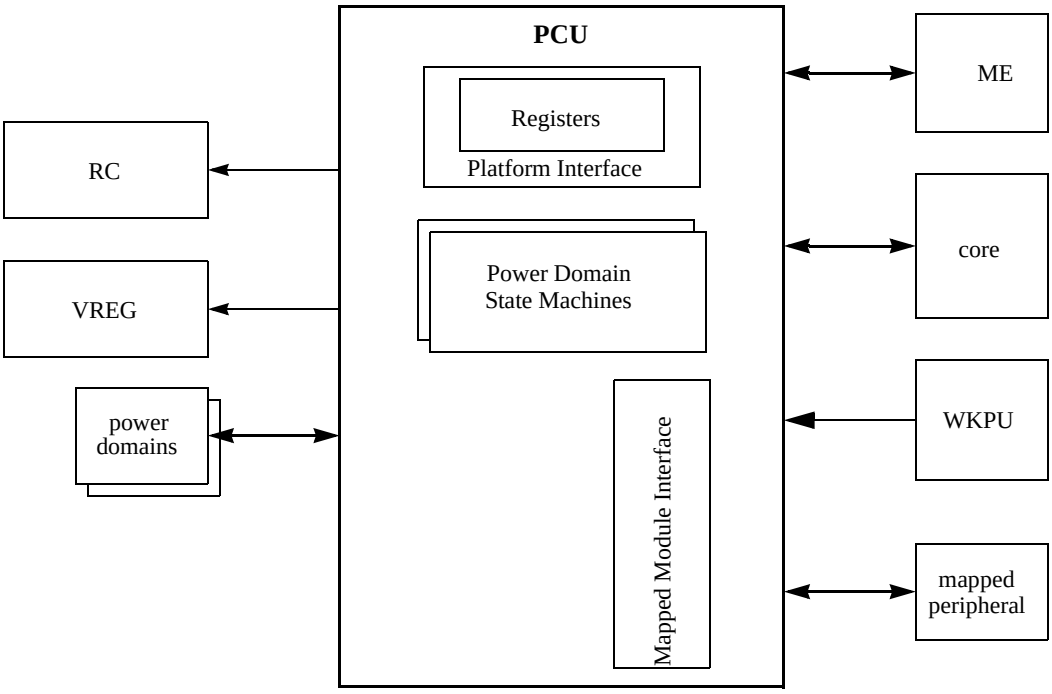


Figure 71. PCU block diagram

7.1.1 Features

The PCU includes the following features:

- Support for as many as 16 power domains consisting of one or more of the following:
  - Standard logic
  - State-retention logic
  - Memories
- Support for device modes RESET, DRUN, SAFE, TEST, RUN0…3, HALT0, and STOP0 (for more mode details, see *Chapter 6: Mode Entry Module (ME)*)
- Power state updating on each mode change and on system wakeup
- Handshake mechanism for power state changes thus guaranteeing operable voltage
- Maps the mapped peripheral registers to the PCU address space

7.1.2 Modes of operation

The PCU is available in all device modes.

7.2 External signal description

The PCU has no connections to any external pins.

7.3 Memory map and register definition

7.3.1 Memory map

Table 61. PCU memory map

Offset from PCU_BASE (0xC3FE_8000)	Register	Location
0x0000	PCU_PCONF0—Power Domain #0 Configuration register	*on page 198*
0x0004–0x003F	Reserved	
0x0040	PCU_PSTAT—Power Domain Status register	*on page 199*
0x0044–0x007F	Reserved	
0x0080	VREG_CTL—Voltage Regulator Control register	*on page 1018*
0x0084	VREG_STATUS—Voltage Regulator Status register	*on page 1019*
0x0088–0x3FFC	Reserved	

Note: Accesses to unused registers as well as write accesses to read-only registers do not change register content, and can cause a transfer error.

Table 62. PCU registers

Address	Name		0	1	2	3	27	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_8000	PCU_PCONF0	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	STBY0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
		W																
0xC3FE_8040	PCU_PSTAT	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PD0
		W																
0xC3FE_8044 … 0xC3FE_807C	reserved																	
0xC3FE_8080	VREG_CTL(1)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V LVD MASK
		W																
0xC3FE_8084	VREG_STATUS(2)	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
		R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V LVD STATUS
		W																
0xC3FE_8088 … 0xC3FE_BFFC	reserved																	

1. See *Section 35.1.4.1: Voltage Regulator Control Register (VREG_CTL)*.
2. See *Section 35.1.4.2: Voltage Regulator Status register (VREG_STATUS)*.

7.3.2 Register descriptions

All registers may be accessed as 32-bit words, 16-bit half-words, or 8-bit bytes. The bytes are ordered according to big endian. For example, the PD0 field of the PCU_PSTAT register may be accessed as a word at address 0xC3FE_8040, as a half-word at address 0xC3FE_8042, or as a byte at address 0xC3FE_8043.

7.3.2.1 Power Domain #0 Configuration register (PCU_PCONF0)

Address: Base + 0x0000 Access: User read-only, Supervisor read-only, Test read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STBY0	0	0	STOP0	0	HALT0	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RST
W																
Reset	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1

Figure 72. Power Domain #0 Configuration register (PCU_PCONF0)

This register defines for power domain #0 whether it is on or off in each device mode. As power domain #0 is the always-on power domain (and includes the PCU), none of its bits are programmable. This register is available for completeness reasons.

Table 63. PCU_PCONF0 field descriptions

Field	Description
RST	Power domain control during RESET mode 0 Power domain off. 1 Power domain on.
TEST	Power domain control during TEST mode 0 Power domain off. 1 Power domain on.
SAFE	Power domain control during SAFE mode 0 Power domain off. 1 Power domain on.
DRUN	Power domain control during DRUN mode 0 Power domain off. 1 Power domain on.
RUN0	Power domain control during RUN0 mode 0 Power domain off. 1 Power domain on.
RUN1	Power domain control during RUN1 mode 0 Power domain off. 1 Power domain on.

Table 63. PCU_PCONF0 field descriptions(Continued)

Field	Description
RUN2	Power domain control during RUN2 mode 0 Power domain off. 1 Power domain on.
RUN3	Power domain control during RUN3 mode 0 Power domain off. 1 Power domain on.
HALT0	Power domain control during HALT0 mode 0 Power domain off. 1 Power domain on.
STOP0	Power domain control during STOP0 mode 0 Power domain off. 1 Power domain on.
STBY0	Power domain control during STANDBY0 mode 0 Power domain off. 1 Power domain on.

7.3.2.2 Power Domain Status register (PCU_PSTAT)

Address: Base + 0x0004 Access: User read-only, Supervisor read-only, Test read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PD15	PD14	PD13	PD12	PD11	PD10	PD9	PD8	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 73. Power Domain Status register (PCU_PSTAT)

This register reflects the power status of all available power domains.

Table 64. PCU_PSTAT field descriptions

Field	Description
PD*n*	Power status for power domain #*n* 0 Power domain is inoperable. 1 Power domain is operable.

7.4 Functional description

7.4.1 General

The PCU controls all available power domains on a device mode basis. The PCU_PCONF*n* registers specify during which system/user modes a power domain is powered up. The power state for each individual power domain is reflected by the bits in the PCU_PSTAT register.

On a mode change, the PCU evaluates which power domain(s) must change power state. The power state is controlled by a state machine (FSM) for each individual power domain, which ensures a clean and safe state transition.

7.4.2 Reset / Power-on reset

After any reset, the SPC560P44Lx, SPC560P50Lx device transitions to the RESET mode, during which all power domains are powered up (see the *Chapter 6: Mode Entry Module (ME)*). Once the reset sequence has been completed, the DRUN mode is entered and software can begin the PCU configuration.

7.4.3 PCU configuration

Per default, all power domains are powered in all modes other than STANDBY0. Software can change the configuration for each power domain on a mode basis by programming the PCU_PCONF*n* registers.

Each power domain which is powered down is held in a reset state. Read/write accesses to peripherals in those power domains will result in a transfer error.

7.4.4 Mode transitions

On a mode change requested by the ME, the PCU evaluates the power configurations for all power domains. It compares the settings in the PCU_PCONF*n* registers for the new mode with the settings for the current mode. If the configuration for a power domain differs between the modes, a power state change request is generated. These requests are handled by a finite state machine to ensure a smooth and safe transition from one power state to another.

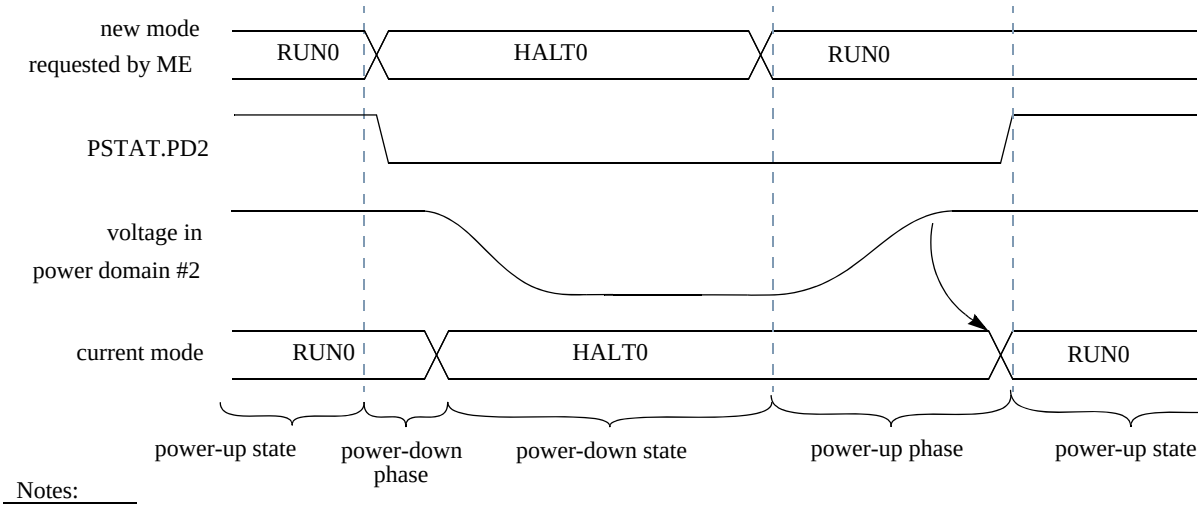
7.4.4.1 DRUN, SAFE, TEST, RUN0…3, HALT0, and STOP0 mode transition

The DRUN, SAFE, TEST, RUN0…3, HALT0, and STOP0 modes allow an increased power saving. The level of power saving is software-controllable via the settings in the PCU_PCONF*n* registers for power domain #2 onwards. The settings for power domains #0 and #1 can not be changed. Therefore, power domains #0 and #1 remain connected to the power supply for all modes beside STANDBY0.

Figure 74 shows an example for a mode transition from RUN0 to HALT0 and back, which will result in power domain #2 being powered down during the HALT0 mode. In this case, PCU_PCONF2[HALT0] is cleared.

When the PCU receives the mode change request to HALT0 mode, it starts its power-down phase. During the power-down phase, clocks are disabled and the reset is asserted resulting in a loss of all information for this power domain.

Then the power domain is disconnected from the power supply (power-down state).



Notes:

Not drawn to scale; PCONF2[RUN0] = 1; PCONF2[HALT0] = 0

Figure 74. PCU events during power sequences

When the PCU receives a mode change request to RUN0, it starts its power-up phase if PCU_PCONF2[RUN0] = 1. The power domain is re-connected to the power supply, and the voltage in power domain #2 will increase slowly. Once the voltage of power domain #2 is within an operable range, its clocks are enabled, and its resets are deasserted (power-up state).

Note: It is possible that, due to a mode change, power-up is requested before a power domain completed its power-down sequence. In this case, the information in that power domain is lost.

7.4.5 Power domain control state machine

The MC_PCU controls the power cells (isolation cells, switches, etc.) for each power domain. Each power domain has its own finite state machine (FSM) in the MC_PCU. The internal structure of each state machine is the same and is described in *Figure 75*. The state machine is divided into three phases: idle, power-down, and power-up.

Unless otherwise noted, the term “power domain” refers to the power domain controlled by this FSM.

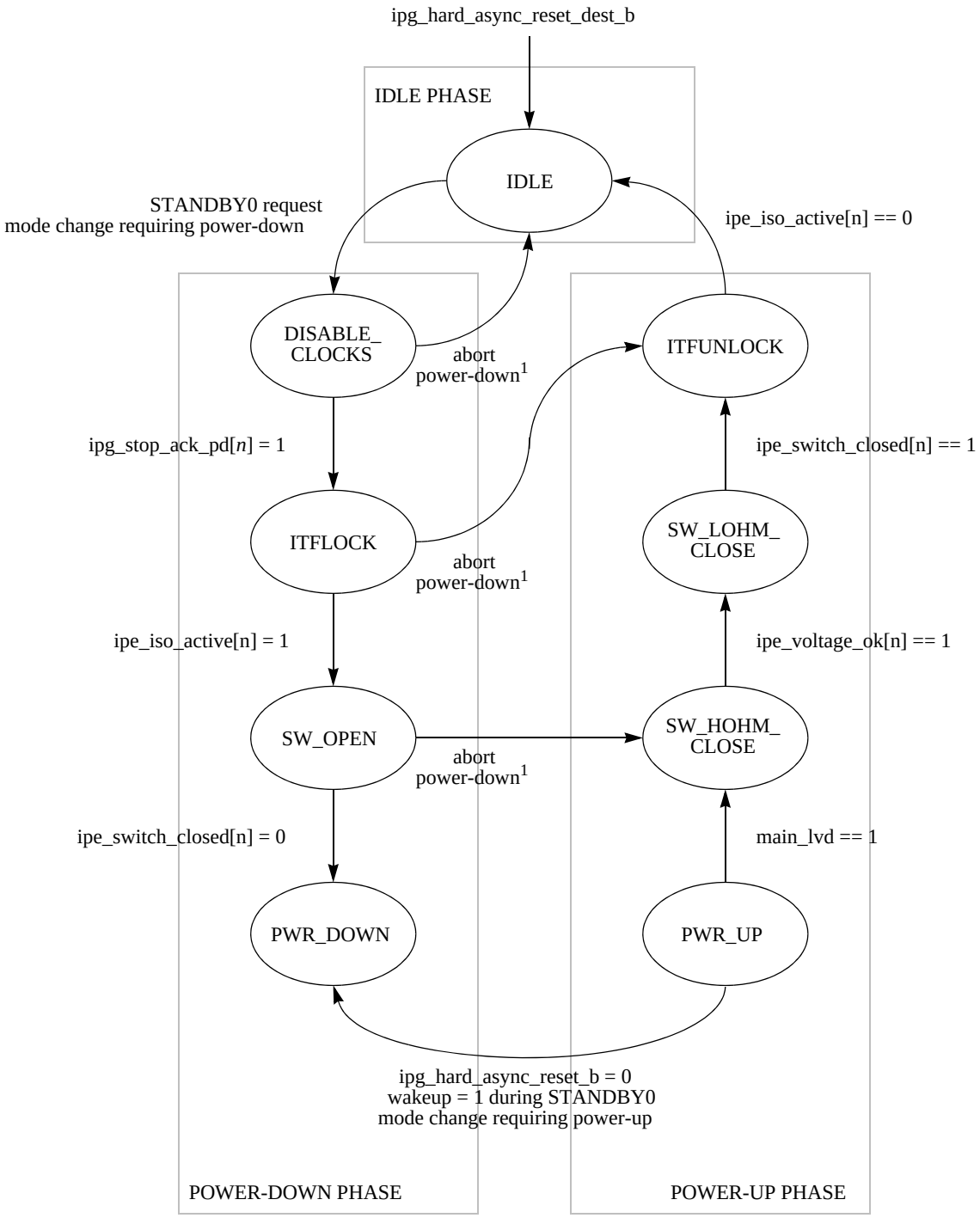


Figure 75. PCU state machine for one power domain

c. See *Section 7.4.5.2.4: Aborting power-down* for details on when a power-down is aborted.

7.4.5.1 Idle phase

During the idle phase, the power domain is fully powered, and the peripherals residing in that power domain are operational. The state for this phase and the conditions for transitioning to and from this state are described below.

7.4.5.1.1 IDLE state

This state is entered from any state on a destructive reset, via the assertion of ipg_hard_async_reset_dest_b, on exit from ITFUNLOCK, or on a power-down abort from DISABLE_CLOCKS. The state machine exits IDLE and enters DISABLE_CLOCKS on one of the following events:

- The ME indicates a change to a new mode during which requires the power domain to be powered-down

During this state, the power domain is fully powered. In the case of power domain #0, the MC_PCU requests one of the following:

- The 16 MHz internal RC oscillator to be disabled by asserting rc_disable (the ME can override the 16 MHz internal RC oscillator control directly)
- The high-power voltage regulator to be disabled based on configuration data from the ME by passing the high_power_vreg_dis_me value to high_power_vreg_dis

7.4.5.2 Power-down phase

During the power-down phase, the power domain is powered down in the following order:

1. Disable the clocks for all peripherals residing in the power domain.
2. Lock the interface between the power domain and other power domains to a fixed state.
3. Disconnect the power domain from the main power supply.

The states for this sequence and the conditions for transitioning to and from these states are described below.

7.4.5.2.1 DISABLE_CLOCKS state

This state is entered on exit from IDLE. The state machine exits DISABLE_CLOCKS and either

- Enters ITFLOCK on the completion of peripheral clock disabling, indicated by the assertion of ipg_stop_ack_pd[*n*], or
- Enters IDLE on a power-down abort (see *Section 7.4.5.2.4: Aborting power-down*).

During this state, the MC_PCU requests

- The ME to disable the clocks to all peripherals residing in the power domain by asserting ipg_stop_pd[*n*]
- In the case of power domain #0, the 16 MHz internal RC oscillator to be enabled by deasserting rc_disable
- In the case of power domain #0, the high-power voltage regulator to be enabled by deasserting high_power_vreg_dis

  ITFLOCK state

This state is entered on exit from DISABLE_CLOCKS. The state machine exits ITFLOCK and either

- Enters SW_OPEN on the completion of isolation cell activation, indicated by the assertion of ipe_iso_active[*n*], or
- Enters ITFUNLOCK on a power-down abort (see *Section 7.4.5.2.4: Aborting power-down*).

Note: In the case of power domain #0, the minimum duration of ITFLOCK before transitioning to SW_OPEN is DELAY_ST_TR ipg_clk_p cycles.

During this state, the MC_PCU requests all isolation cells associated with the power domain to be activated by asserting ipe_iso[*n*].

7.4.5.2.2 SW_OPEN state

This state is entered on exit from ITFLOCK. The state machine exits SW_OPEN and either

- Enters PWR_DOWN on the completion of power gate opening, indicated by the assertion of ipe_switch_closed[*n*], or
- Enters SW_HOHM_CLOSE on a power-down abort (see *Section 7.4.5.2.4: Aborting power-down*).

Note: In the case of power domain #0, the minimum duration of ITFLOCK + SW_OPEN before transitioning to PWR_DOWN is DELAY_END_TR ipg_clk_p cycles.

During this state, the MC_PCU requests

- All power gates connecting the power domain to the main power supply to be opened by deasserting ipe_switch_b[*n*] and ipe_short_b[*n*]
- All embedded memories in the power domain to exit full-power mode by asserting ipe_pd[*n*]

Note: In the case of power domain #0, ipe_pd[0] is asserted when entering STOP0 mode and ipg_clk_sys_en has been deasserted. This may occur before entering the SW_OPEN state.

- In the case of power domain #0, the main low-voltage detector to be disabled by asserting main_lvd_dis

7.4.5.2.3 PWR_DOWN state

This state is entered on exit from SW_OPEN. The state machine exits PWR_DOWN and enters PWR_UP on any of the following events:

- A PHASE3 reset via the assertion of the ipg_hard_async_reset_b input
- A mode change request during one of the non-powered-down states requiring a power-up

During this state, the MC_PCU requests

- All embedded memories in the power domain to enter low-power mode by asserting ipe_pd_mode[*n*]

Note: In the case of power domain #0, ipe_pd_mode[0] is asserted when entering STOP0 mode and ipg_clk_sys_en has been deasserted. This may occur before entering the PWR_DOWN state.

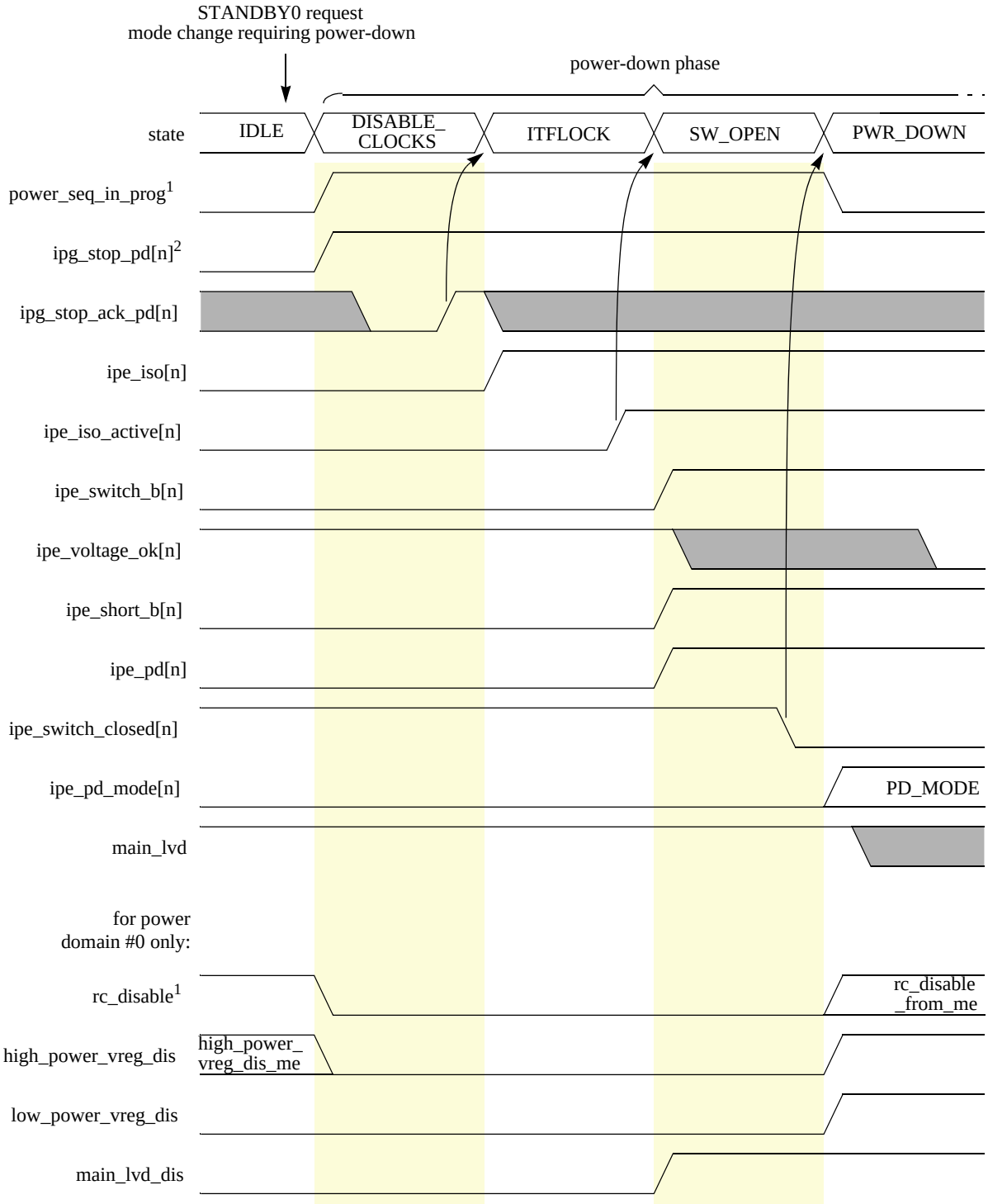
- In the case of power domain #0, the 16 MHz internal RC oscillator to be disabled based on configuration data from the ME by passing the captured value of rc_disable_from_me to rc_disable
- In the case of power domain #0, the high- and low-power voltage regulators to be disabled by asserting high_power_vreg_dis and low_power_vreg_dis

7.4.5.2.4 Aborting power-down

The power-down phase is immediately terminated and the power-up phase entered at the appropriate state in the case of one of the following events:

- A PHASE3 reset via the assertion of the ipg_hard_async_reset_b input
- A mode change request during one of the non-powered-down states requiring a power-up

Note: A power-down abort always has priority over other state changing events.



[1]power_seq_in_prog is deasserted and rc_disable asserted only when the last power domain transitions to PWR_DOWN.
[2]ipg_stop_pd[0] is always deasserted (i.e., power domain #0 is never powered down in a low-power mode).

Figure 76. PCU power-down sequence

7.4.5.3 Power-up phase

During the power-up phase, the power domain is powered up in the following order:

1. Ensure that the main power supply voltage is sufficient.
2. Connect the power domain to the main power supply.
3. Unlock the interface between the power domain and other power domains to allow normal communication between them.

The states for this sequence and the conditions for transitioning to and from these states are described below.

7.4.5.3.1 PWR_UP state

This state is entered on exit from PWR_DOWN. The state machine exits PWR_UP and enters SW_HOHM_CLOSE on confirmation that the main power supply voltage is okay, indicated by the assertion of main_lvd.

During this state, the MC_PCU requests

- In the case of power domain #0, the high- and low-power voltage regulators to be enabled by deasserting high_power_vreg_dis and low_power_vreg_dis
- In the case of power domain #0, the main low-voltage detector to be enabled by deasserting main_lvd_dis

7.4.5.3.2 SW_HOHM_CLOSE state

This state is entered either on exit from PWR_UP or on a power-down abort from SW_OPEN. The state machine exits SW_HOHM_CLOSE and enters SW_LOHM_CLOSE on confirmation that the power domain's voltage is okay, indicated by the assertion of ipe_voltage_ok[*n*].

During this state, the MC_PCU requests all power gates (in the two-stage gate, only the high-impedance part) connecting

the power domain to the main power supply to be closed by asserting ipe_switch_b[*n*].

7.4.5.3.3 SW_LOHM_CLOSE state

This state is entered on exit from SW_HOHM_CLOSE. The state machine exits SW_LOHM_CLOSE and enters ITFUNLOCK on confirmation that all the power domain's power gates have completely closed, indicated by the assertion of ipe_switch_closed[*n*].

During this state, the MC_PCU requests the following:

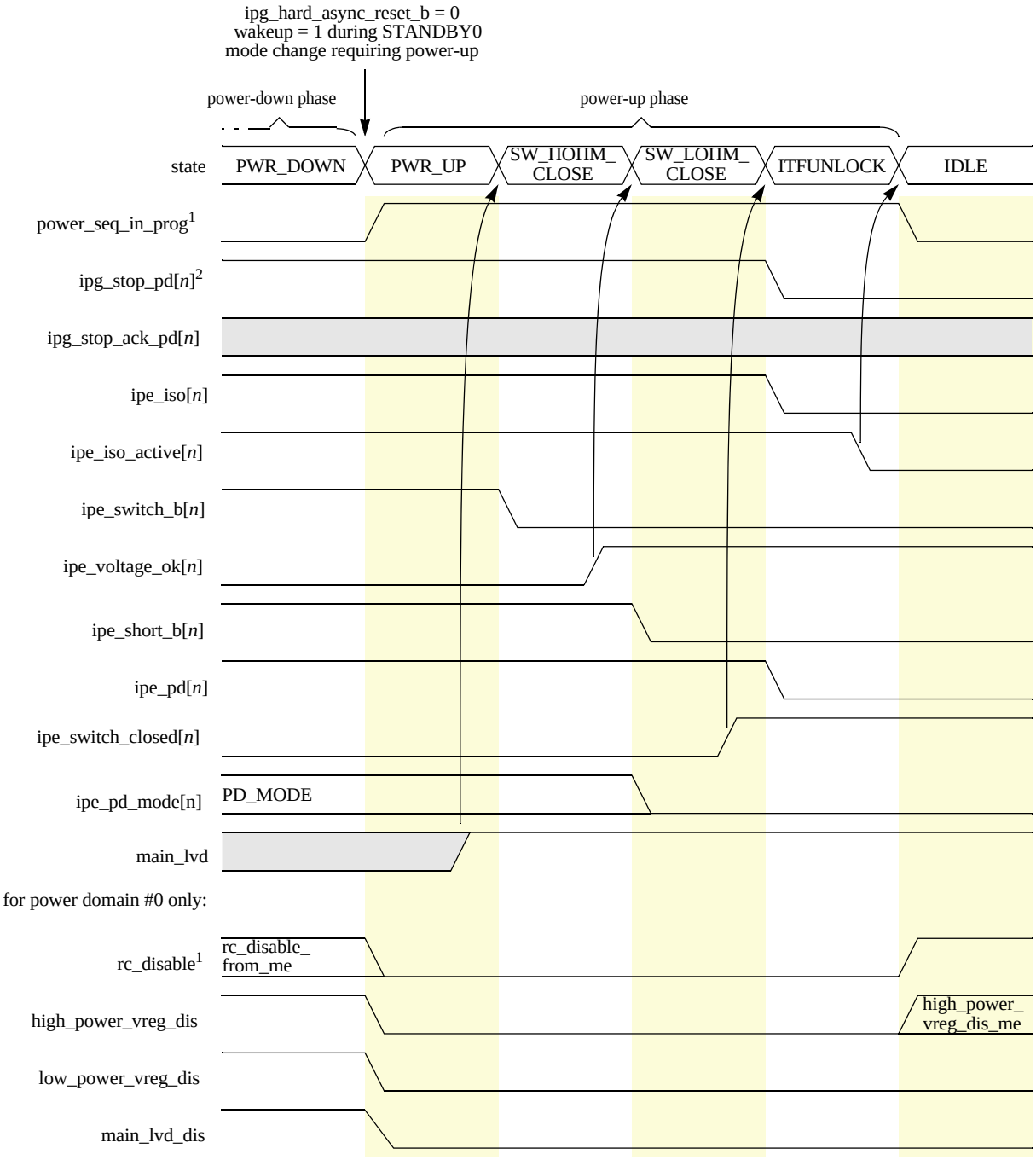
- The low-impedance part of all two-stage power gates connecting the power domain to the main power supply to be closed by asserting ipe_short_b[*n*]
- All embedded memories in the power domain to exit low-power mode by deasserting ipe_pd_mode

7.4.5.3.4 ITFUNLOCK state

This state is entered either on exit from SW_LOHM_CLOSE or on a power-down abort from ITFLOCK. The state machine exits ITFUNLOCK and enters IDLE on completion of isolation cell deactivation, indicated by the deassertion of ipe_iso_active[*n*].

During this state, the MC_PCU requests

- All isolation cells associated with the power domain to be deactivated by deasserting ipe_iso[*n*]
- All clocks to the peripherals residing in the power domain to be released for enabling by deasserting ipg_stop_pd[*n*]
- All embedded memories in the power domain to enter full-power mode by deasserting ipe_pd



[1]power_seq_in_prog is deasserted and rc_disable asserted only when the last power domain transitions to IDLE.
[2]ipg_stop_pd[0] is always deasserted (i.e., power domain #0 is never powered down in a low-power mode).

Figure 77. PCU power-up sequence

7.5 Initialization information

To initialize the PCU, the registers PCU_PCONF2…15 should be programmed. After programming is done, those registers should no longer be changed.

8 Reset Generation Module (RGM)

8.1 Introduction

The reset generation module (RGM) centralizes the different reset sources and manages the reset sequence of the device. It provides a register interface and the reset sequencer. The different registers are available to monitor and control the device reset sequence. The reset sequencer is a state machine that controls the different phases (PHASE0, PHASE1, PHASE2, PHASE3, and IDLE) of the reset sequence and control the reset signals generated in the system. *Figure 78* depicts the RGM block diagram.

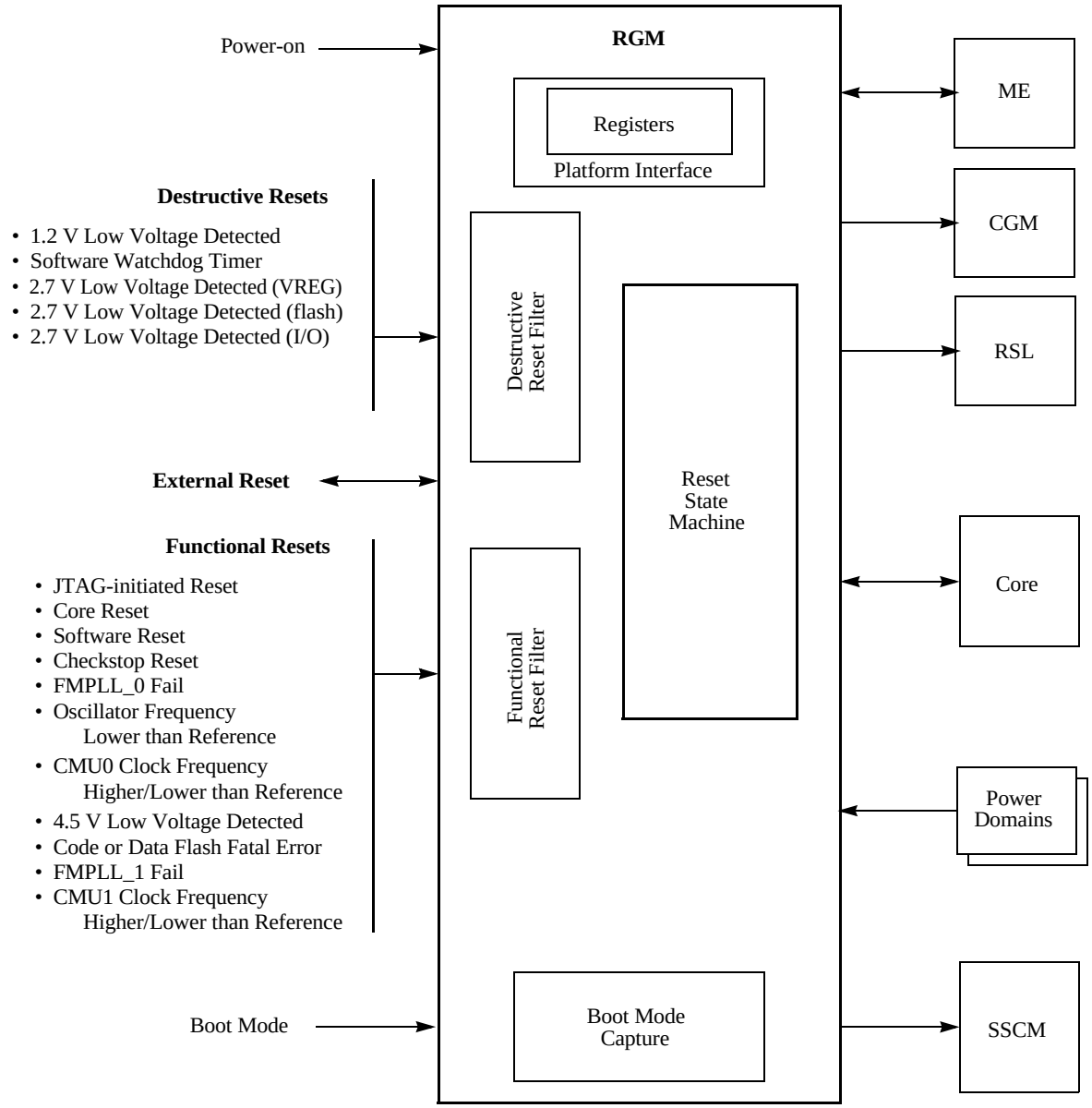


Figure 78. RGM Block Diagram

8.2 Features

The RGM contains the functionality for the following features:

- Destructive reset management
- Functional resets management
- Signalling of reset events after each reset sequence (reset status flags)
- Conversion of reset events to SAFE mode or interrupt request events (for further mode details, please see *Chapter 6: Mode Entry Module (ME)*)
- Short reset sequence configuration
- Bidirectional reset behavior configuration
- Boot mode capture on ipg_rst_external_b deassertion

Note: The RGM also provides test features for reset controlling. While the device is in test mode, it is possible to directly control the reset sources, thus avoiding unwanted reset generation, or to bypass the reset control.

8.3 Modes of operation

The different reset sources are organized into two families: destructive and functional.

- A destructive reset source is associated with an event related to a critical error or dysfunction, usually in hardware. When a destructive reset event occurs, the full reset sequence is applied to the device starting from PHASE0. This resets the full device ensuring a safe start-up state for both digital and analog modules. Destructive resets typically include:
  - Power-on reset
  - Low voltage detection
  - Watchdog timer
- A functional reset source is associated with an event related to a less critical error or dysfunction, usually non-hardware. When a functional reset event occurs, a partial reset sequence is applied to the device starting from PHASE1. In this case, most digital modules are reset normally, while the state of analog modules or specific digital modules (e.g., debug modules, flash modules) is preserved. Functional resets are typically
  - External reset
  - Clock monitor failure (e.g., PLL lock, clock quality)
  - Machine check
  - Software reset from mode entry
  - Boundary scan instructions

When a reset is triggered, the RGM state machine is activated and proceeds through the different phases (i.e., PHASE*n* states). Each phase is associated with a particular device reset being provided to the system. A phase is completed when all corresponding phase completion gates from either the system or internal to the RGM are acknowledged. The device reset associated with the phase is then released, and the state machine proceeds to the next phase up to entering the IDLE phase. During this entire process, the MC_ME state machine is held in RESET mode. Only at the end of the reset sequence, when the IDLE phase is reached, does the MC_ME enter the DRUN mode.

Alternatively, it is possible for software to configure some reset source events to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt issued to the core (see *Section 8.5.1.4: Destructive Event Reset Disable register (RGM_DERD)*, and *Section 8.5.1.3: Functional Event Reset Disable register (RGM_FERD)*, and *Section 8.5.1.5: Functional Event Alternate Request register (RGM_FEAR)* for functional resets).

8.4 External signal description

Table 65 describes the signals that are connected to the I/O pad ring.

Table 65. RGM external signals

Signal Name	Direction	Reset value	Description
ipg_rst_external_b	in	—	Active low external reset
ipg_rst_external_out	out	—	Active high external reset output, used to assert the external reset pad (in case of bidirectional reset)
chip_mode_in[CM_SIZE-1:0]	in	—	Boot mode from external pin(s)
ipt_test	in	—	Test mode from external test pin

The RGM interfaces to the bidirectional reset pin $\overline{\text{RESET}}$ and the boot mode PAD[4:2].

8.5 Memory map and registers description

Table 66 lists the registers for the RGM. *Table 67* provides the memory map for the RGM.

Table 66. RGM registers

Offset from RGM_BASE (0xC3FE_4000)	Register	Location
0x0000	RGM_FES—Functional Event Status register	*on page 215*
0x0002	RGM_DES—Destructive Event Status	*on page 217*
0x0004	RGM_FERD—Functional Event Reset Disable	*on page 218*
0x0006	RGM_DERD—Destructive Event Reset Disable	*on page 220*
0x0008–0x000F	Reserved	
0x0010	RGM_FEAR—Functional Event Alternate Request	*on page 220*
0x0012–0x0017	Reserved	
0x0018	RGM_FESS—Functional Event Short Sequence	*on page 222*
0x001A–0x001B	Reserved	
0x001C	RGM_FBRE—Functional Bidirectional Reset Enable	*on page 223*
0x001E–0x3FFF	Reserved	

Note: Any access to unused registers as well as write accesses to read-only registers will not change register content and can cause a transfer error.

Table 67. RGM memory map

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4000	RGM_FES / RGM_DES	R	F_EXR	0	0	0	F_CMU1_FHL	reserved	F_PLL1	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
		W	w1c				w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
		R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	reserved	F_SWT	reserved	F_LVD12_PD0
		W	w1c									w1c	w1c	w1c	w1c	w1c	w1c	w1c
0xC3FE_4004	RGM_FERD / RGM_DERD	R	D_EXR	0	0	0	D_CMU1_FHL	0	D_PLL1	D_FLASH	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	D_SWT	0	D_LVD12_PD0
		W																
0xC3FE_4008 … 0xC3FE_400F	reserved																	
0xC3FE_4010	RGM_FEAR	R	0	0	0	0	AR_CMU1_FHL	reserved	AR_PLL1	AR_FLASH	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CHKSTOP	AR_SOFT	AR_CORE	AR_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																

Table 67. RGM memory map(Continued)

Address	Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0xC3FE_4014 ... 0xC3FE_4017	reserved																	
0xC3FE_4018	RGM_FESS	R	0	0	0	0	SS_CMU1_FHL	reserved	SS_PLL1	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401C	RGM_FBRE	R	0	0	0	0	BE_CMU1_FHL	reserved	BE_PLL1	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
		W																
		R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		W																
0xC3FE_401E ... 0xC3FE_7FFF	reserved																	

8.5.1 Registers description

8.5.1.1 Functional Event Status register (RGM_FES)

Address: Base + 0x0000 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_EXR	0	0	0	F_CMU1_FHL	reserved	F_PLL1	F_FLASH	F_LVD45	F_CMU0_FHL	F_CMU0_OLR	F_PLL0	F_CHKSTOP	F_SOFT	F_CORE	F_JTAG
W	w1c				w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 79. Functional Event Status register (RGM_FES)

This register contains the status of the last asserted functional reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared when written with 1.

Table 68. RGM_FES field descriptions

Field	Description
F_EXR	Flag for External Reset 0 No external reset event has occurred since either the last clear or the last destructive reset assertion. 1 An external reset event has occurred.
F_CMU1_FHL	Flag for CMU1 clock frequency higher/lower than reference 0 No CMU1 clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion. 1 A CMU1 clock frequency higher/lower than reference event has occurred.
F_PLL1	Flag for PLL1 fail 0 No PLL1 fail event has occurred since either the last clear or the last destructive reset assertion. 1 A PLL1 fail event has occurred.
F_FLASH	Flag for code or data flash fatal error 0 No code or data flash fatal error event has occurred since either the last clear or the last destructive reset assertion. 1 A code or data flash fatal error event has occurred.
F_LVD45	Flag for 4.5 V low-voltage detected 0 No 4.5 V low-voltage detected event has occurred since either the last clear or the last destructive reset assertion. 1 A 4.5 V low-voltage detected event has occurred.
F_CMU0_FHL	Flag for CMU0 clock frequency higher/lower than reference 0 No CMU0 clock frequency higher/lower than reference event has occurred since either the last clear or the last destructive reset assertion. 1 A CMU0 clock frequency higher/lower than reference event has occurred.
F_CMU0_OLR	Flag for oscillator frequency lower than reference 0 No oscillator frequency lower than reference event has occurred since either the last clear or the last destructive reset assertion. 1 A oscillator frequency lower than reference event has occurred.
F_PLL0	Flag for PLL0 fail 0 No PLL0 fail event has occurred since either the last clear or the last destructive reset assertion. 1 A PLL0 fail event has occurred.
F_CHKSTOP	Flag for checkstop reset 0 No checkstop reset event has occurred since either the last clear or the last destructive reset assertion. 1 A checkstop reset event has occurred.
F_SOFT	Flag for software reset 0 No software reset event has occurred since either the last clear or the last destructive reset assertion. 1 A software reset event has occurred.

Table 68. RGM_FES field descriptions(Continued)

Field	Description
F_CORE	Flag for core reset 0 No core reset event has occurred since either the last clear or the last destructive reset assertion. 1 A core reset event has occurred.
F_JTAG	Flag for JTAG initiated reset 0 No JTAG initiated reset event has occurred since either the last clear or the last destructive reset assertion. 1 A JTAG initiated reset event has occurred.

8.5.1.2 Destructive Event Status register (RGM_DES)

Address: Base + 0x0002 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F_POR	0	0	0	0	0	0	0	0	F_LVD27_IO	F_LVD27_FLASH	F_LVD27_VREG	reserved	F_SWT	reserved	F_LVD12_PD0
W	w1c									w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 80. Destructive Event Status register (RGM_DES)

This register contains the status of the last asserted destructive reset sources. It can be accessed in read/write on either supervisor mode or test mode. Register bits are cleared when written with 1.

Table 69. RGM_DES field descriptions

Field	Description
F_POR	Flag for Power-On reset 0 No power-on event has occurred since the last clear. 1 A power-on event has occurred.
F_LVD27_IO	Flag for 2.7 V low-voltage detected (I/O) 0 No 2.7 V low-voltage detected (I/O) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage detected (I/O) event has occurred.
F_LVD27_FLASH	Flag for 2.7 V low-voltage detected (flash) 0 No 2.7 V low-voltage detected (flash) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage detected (flash) event has occurred.
F_LVD27_VREG	Flag for 2.7 V low-voltage detected (VREG) 0 No 2.7 V low-voltage detected (VREG) event has occurred since either the last clear or the last power-on reset assertion. 1 A 2.7 V low-voltage detected (VREG) event has occurred.

Table 69. RGM_DES field descriptions(Continued)

Field	Description
F_SWT	Flag for software watchdog timer 0 No software watchdog timer event has occurred since either the last clear or the last power-on reset assertion. 1 A software watchdog timer event has occurred.
F_LVD12_PD0	Flag for 1.2 V low-voltage detected 0 No 1.2 V low-voltage detected event has occurred since either the last clear or the last power-on reset assertion. 1 A 1.2 V low-voltage detected event has occurred.

*Note: The **F_POR** flag is automatically cleared on a 1.2 V low-voltage detected or a 2.7 V low-voltage detected (VREG). This means that if the power-up sequence is not monotonic (i.e the voltage rises and then drops enough to trigger a low-voltage detection), the **F_POR** flag may not be set but instead the F_LVD12_PD0 or F_LVD27_VREG flag is set on exiting the reset sequence. Therefore, if the **F_POR**, F_LVD12_PD0 or F_LVD27_VREG flags are set on reset exit, software should interpret the reset cause as power-on.*

*Note: In contrast to all other reset sources, the 1.2 V low-voltage detected event is captured on its deassertion. Therefore, the status bit **F_LVD12_PD0** is also asserted on the reset's deassertion. In case an alternate event is selected, the **SAFE** mode or interrupt request are similarly asserted on the reset's deassertion.*

8.5.1.3 Functional Event Reset Disable register (RGM_FERD)

Address: Base + 0x0004 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	D_EXR	0	0	0	D_CMU1_FHL	0	D_PLL1	D_FLASH	D_LVD45	D_CMU0_FHL	D_CMU0_OLR	D_PLL0	D_CHKSTOP	D_SOFT	D_CORE	D_JTAG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 81. Functional Event Reset Disable register (RGM_FERD)

This register provides dedicated bits to disable functional reset sources. When a functional reset source is disabled, the associated destructive event will trigger either a SAFE mode request or an interrupt request (see *Section 8.5.1.5: Functional Event Alternate Request register (RGM_FEAR)*). It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode. Register bits can be written only once after power-on reset.

Table 70. RGM_FERD field descriptions

Field	Description
D_EXR	Disable External Reset 0 An external reset event triggers a reset sequence. 1 An external reset event generates a SAFE mode request.
D_CMU1_FHL	Disable CMU1 clock frequency higher/lower than reference 0 A CMU1 clock frequency higher/lower than reference event triggers a reset sequence. 1 A CMU1 clock frequency higher/lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CMU1_FHL].
D_PLL1	Disable PLL1 fail 0 A PLL1 fail event triggers a reset sequence. 1 A PLL1 fail event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_PLL1].
D_FLASH	Disable code or data flash fatal error 0 A code or data flash fatal error event triggers a reset sequence. 1 A code or data flash fatal error event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_FLASH].
D_LVD45	Disable 4.5 V low-voltage detected 0 A 4.5 V low-voltage detected event triggers a reset sequence. 1 A 4.5 V low-voltage detected event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_LVD45].
D_CMU0_FHL	Disable CMU0 clock frequency higher/lower than reference 0 A CMU0 clock frequency higher/lower than reference event triggers a reset sequence 1 A CMU0 clock frequency higher/lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CMU0_FHL]
D_CMU0_OLR	Disable oscillator frequency lower than reference 0 A oscillator frequency lower than reference event triggers a reset sequence 1 A oscillator frequency lower than reference event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CMU0_OLR]
D_PLL0	Disable PLL0 fail 0 A PLL0 fail event triggers a reset sequence. 1 A PLL0 fail event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_PLL0].
D_CHKSTOP	Disable checkstop reset 0 A checkstop reset event triggers a reset sequence. 1 A checkstop reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CHKSTOP].
D_SOFT	Disable software reset 0 A software reset event triggers a reset sequence. 1 A software reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_SOFT].
D_CORE	Disable core reset 0 A core reset event triggers a reset sequence. 1 A core reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_CORE].
D_JTAG	Disable JTAG initiated reset 0 A JTAG initiated reset event triggers a reset sequence. 1 A JTAG initiated reset event generates either a SAFE mode or an interrupt request depending on the value of RGM_FEAR[AR_JTAG].

8.5.1.4 Destructive Event Reset Disable register (RGM_DERD)

Address Base + 0x0006 Access: read-only

:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	D_LVD27_IO	D_LVD27_FLASH	D_LVD27_VREG	0	D_SWT	0	D_LVD12_PD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 82. Destructive Event Reset Disable register (RGM_DERD)

This register provides dedicated bit to disable a particular destructive reset source. It can be accessed in read-only in supervisor mode, test mode, and user mode.

Table 71. RGM_DERD field descriptions

Field	Description
D_LVD27_IO	Disable 2.7 V low-voltage detected (I/O) 0 A 2.7 V low-voltage detected (I/O) event triggers a reset sequence.
D_LVD27_FLASH	Disable 2.7 V low-voltage detected (flash) 0 A 2.7 V low-voltage detected (flash) event triggers a reset sequence.
D_LVD27_VREG	Disable 2.7 V low-voltage detected (VREG) 0 A 2.7 V low-voltage detected (VREG) event triggers a reset sequence.
D_SWT	Disable software watchdog timer 0 A software watchdog timer event triggers a reset sequence.
D_LVD12_PD0	Disable 1.2 V low-voltage detected 0 A 1.2 V low-voltage detected event triggers a reset sequence.

8.5.1.5 Functional Event Alternate Request register (RGM_FEAR)

Address: Base + 0x0010 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	AR_CMU1_FHL	reserved	AR_PLL1	AR_FLASH	AR_LVD45	AR_CMU0_FHL	AR_CMU0_OLR	AR_PLL0	AR_CHKSTOP	AR_SOFT	AR_CORE	AR_JTAG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 83. Functional Event Alternate Request register (RGM_FEAR)

This register defines alternate request to be generated when reset on functional event has been disabled. Alternate request can be either a SAFE mode request to ME or an interrupt request to the system. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read only in user mode.

Table 72. RGM_FEAR field descriptions

Field	Description
AR_CMU1_FHL	Alternate Request for CMU1 clock frequency higher/lower than reference 0 Generate a SAFE mode request on a CMU1 clock frequency higher/lower than reference event if the reset is disabled. 1 Generate an interrupt request on a CMU1 clock frequency higher/lower than reference event if the reset is disabled.
AR_PLL1	Alternate Request for PLL1 fail 0 Generate a SAFE mode request on a PLL1 fail event if the reset is disabled. 1 Generate an interrupt request on a PLL1 fail event if the reset is disabled.
AR_FLASH	Alternate Request for code or data flash fatal error 0 Generate a SAFE mode request on a code or data flash fatal error event if the reset is disabled. 1 Generate an interrupt request on a code or data flash fatal error event if the reset is disabled.
AR_LVD45	Alternate Request for 4.5 V low-voltage detected 0 Generate a SAFE mode request on a 4.5 V low-voltage detected event if the reset is disabled. 1 Generate an interrupt request on a 4.5 V low-voltage detected event if the reset is disabled.
AR_CMU0_FHL	Alternate Request for CMU0 clock frequency higher/lower than reference 0 Generate a SAFE mode request on a CMU0 clock frequency higher/lower than reference event if the reset is disabled. 1 Generate an interrupt request on a CMU0 clock frequency higher/lower than reference event if the reset is disabled.
AR_CMU0_OLR	Alternate Request for oscillator frequency lower than reference 0 Generate a SAFE mode request on a oscillator frequency lower than reference event if the reset is disabled. 1 Generate an interrupt request on a oscillator frequency lower than reference event if the reset is disabled.
AR_PLL0	Alternate Request for PLL0 fail 0 Generate a SAFE mode request on a PLL0 fail event if the reset is disabled. 1 Generate an interrupt request on a PLL0 fail event if the reset is disabled.
AR_CHKSTOP	Alternate Request for checkstop reset 0 Generate a SAFE mode request on a checkstop reset event if the reset is disabled. 1 Generate an interrupt request on a checkstop reset event if the reset is disabled.
AR_SOFT	Alternate Request for software reset 0 Generate a SAFE mode request on a software reset event if the reset is disabled. 1 Generate an interrupt request on a software reset event if the reset is disabled.
AR_CORE	Alternate Request for core reset 0 Generate a SAFE mode request on a core reset event if the reset is disabled. 1 Generate an interrupt request on a core reset event if the reset is disabled.
AR_JTAG	Alternate Request for JTAG initiated reset 0 Generate a SAFE mode request on a JTAG initiated reset event if the reset is disabled. 1 Generate an interrupt request on a JTAG initiated reset event if the reset is disabled.

8.5.1.6 Functional Event Short Sequence register (RGM_FESS)

Address: Base + 0x0018 Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SS_CMU1_FHL	reserved	SS_PLL1	SS_FLASH	SS_LVD45	SS_CMU0_FHL	SS_CMU0_OLR	SS_PLL0	SS_CHKSTOP	SS_SOFT	SS_CORE	SS_JTAG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 84. Functional Event Short Sequence register (RGM_FESS)

This register defines which reset sequence is performed when a functional reset sequence is triggered. The functional reset sequence can either start from PHASE1, enforcing reset of all modules or from PHASE3, skipping PHASE1 and PHASE2 and thus asserting reset only on modules associated with PHASE3.

Note: This could be useful for fast reset sequence, for example to skip flash reset.

It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 73. RGM_FESS field descriptions

Field	Description
SS_CMU1_FHL	Short Sequence for CMU1 clock frequency higher/lower than reference 0 The reset sequence triggered by a CMU1 clock frequency higher/lower than reference event will start from PHASE1. 1 The reset sequence triggered by a CMU1 clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_PLL1	Short Sequence for PLL1 fail 0 The reset sequence triggered by a PLL1 fail event will start from PHASE1. 1 The reset sequence triggered by a PLL1 fail event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_FLASH	Short Sequence for code or data flash fatal error 0 The reset sequence triggered by a code or data flash fatal error event will start from PHASE1. 1 The reset sequence triggered by a code or data flash fatal error event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_LVD45	Short Sequence for 4.5 V low-voltage detected 0 The reset sequence triggered by a 4.5 V low-voltage detected event will start from PHASE1. 1 The reset sequence triggered by a 4.5 V low-voltage detected event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_CMU0_FHL	Short Sequence for CMU0 clock frequency higher/lower than reference 0 The reset sequence triggered by a CMU0 clock frequency higher/lower than reference event will start from PHASE1. 1 The reset sequence triggered by a CMU0 clock frequency higher/lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2.

Table 73. RGM_FESS field descriptions(Continued)

Field	Description
SS_CMU0_OLR	Short Sequence for oscillator frequency lower than reference 0 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE1. 1 The reset sequence triggered by a oscillator frequency lower than reference event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_PLL0	Short Sequence for PLL0 fail 0 The reset sequence triggered by a PLL0 fail event will start from PHASE1. 1 The reset sequence triggered by a PLL0 fail event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_CHKSTOP	Short Sequence for checkstop reset 0 The reset sequence triggered by a checkstop reset event will start from PHASE1. 1 The reset sequence triggered by a checkstop reset event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_SOFT	Short Sequence for software reset 0 The reset sequence triggered by a software reset event will start from PHASE1. 1 The reset sequence triggered by a software reset event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_CORE	Short Sequence for core reset 0 The reset sequence triggered by a core reset event will start from PHASE1. 1 The reset sequence triggered by a core reset event will start from PHASE3, skipping PHASE1 and PHASE2.
SS_JTAG	Short Sequence for JTAG initiated reset 0 The reset sequence triggered by a JTAG initiated reset event will start from PHASE1. 1 The reset sequence triggered by a JTAG initiated reset event will start from PHASE3, skipping PHASE1 and PHASE2.

8.5.1.7 Functional Bidirectional Reset Enable register (RGM_FBRE)

Address Base + 0x001C : Access: User read-only, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	BE_CMU1_FHL	reserved	BE_PLL1	BE_FLASH	BE_LVD45	BE_CMU0_FHL	BE_CMU0_OLR	BE_PLL0	BE_CHKSTOP	BE_SOFT	BE_CORE	BE_JTAG
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 85. Functional Bidirectional Reset Enable register (RGM_FBRE)

This register enables the generation of an external reset on functional reset. It can be accessed in read/write in either supervisor mode or test mode. It can be accessed in read in user mode.

Table 74. RGM_FBRE field descriptions

Field	Description
BE_CMU1_FHL	Bidirectional Reset Enable for CMU1 clock frequency higher/lower than reference 0 RESET_B is asserted on a CMU1 clock frequency higher/lower than reference event if the reset is enabled. 1 RESET_B is not asserted on a CMU1 clock frequency higher/lower than reference event.
BE_PLL1	Bidirectional Reset Enable for PLL1 fail 0 RESET_B is asserted on a PLL1 fail event if the reset is enabled. 1 RESET_B is not asserted on a PLL1 fail event.
BE_FLASH	Bidirectional Reset Enable for code or data flash fatal error 0 RESET_B is asserted on a code or data flash fatal error event if the reset is enabled. 1 RESET_B is not asserted on a code or data flash fatal error event.
BE_LVD45	Bidirectional Reset Enable for 4.5 V low-voltage detected 0 RESET_B is asserted on a 4.5 V low-voltage detected event if the reset is enabled. 1 RESET_B is not asserted on a 4.5 V low-voltage detected event.
BE_CMU0_FHL	Bidirectional Reset Enable for CMU0 clock frequency higher/lower than reference 0 RESET_B is asserted on a CMU0 clock frequency higher/lower than reference event if the reset is enabled. 1 RESET_B is not asserted on a CMU0 clock frequency higher/lower than reference event.
BE_CMU0_OLR	Bidirectional Reset Enable for oscillator frequency lower than reference 0 RESET_B is asserted on a oscillator frequency lower than reference event if the reset is enabled. 1 RESET_B is not asserted on a oscillator frequency lower than reference event.
BE_PLL0	Bidirectional Reset Enable for PLL0 fail 0 RESET_B is asserted on a PLL0 fail event if the reset is enabled. 1 RESET_B is not asserted on a PLL0 fail event.
BE_CHKSTOP	Bidirectional Reset Enable for checkstop reset 0 RESET_B is asserted on a checkstop reset event if the reset is enabled. 1 RESET_B is not asserted on a checkstop reset event.
BE_SOFT	Bidirectional Reset Enable for software reset 0 RESET_B is asserted on a software reset event if the reset is enabled. 1 RESET_B is not asserted on a software reset event.
BE_CORE	Bidirectional Reset Enable for core reset 0 RESET_B is asserted on a core reset event if the reset is enabled. 1 RESET_B is not asserted on a core reset event.
BE_JTAG	Bidirectional Reset Enable for JTAG initiated reset 0 RESET_B is asserted on a JTAG initiated reset event if the reset is enabled. 1 RESET_B is not asserted on a JTAG initiated reset event.

8.6 Functional description

8.6.1 Reset state machine

The main role of RGM is the generation of the reset sequence that ensures that the correct parts of the device are reset based on the reset source event. This is summarized in *Table 75*.

Table 75. RGM reset implications

Source	What Gets Reset	External Reset Assertion	Boot Mode Capture
Power-on reset	All	yes	yes
Destructive resets	All except some clock/reset management	yes	yes
External reset	All except some clock/reset management and debug	yes	yes
Functional resets	All except some clock/reset management and debug	programmable (1)	programmable(2)
Shortened functional resets(3)	Flip-flops except some clock/reset management	programmable(1)	programmable(2)

1. the assertion of the external reset is controlled via the RGM_FBRE register
2. the boot mode is captured if the external reset is asserted
3. the short sequence is enabled via the RGB_FESS register

The reset sequence is composed of five phases managed by a state machine, which ensures that all phases are correctly processed through waiting for a minimum duration and until all processes that need to occur during that phase have been completed before proceeding to the next phase.

The state machine used to produce the reset sequence is shown in *Figure 86*.

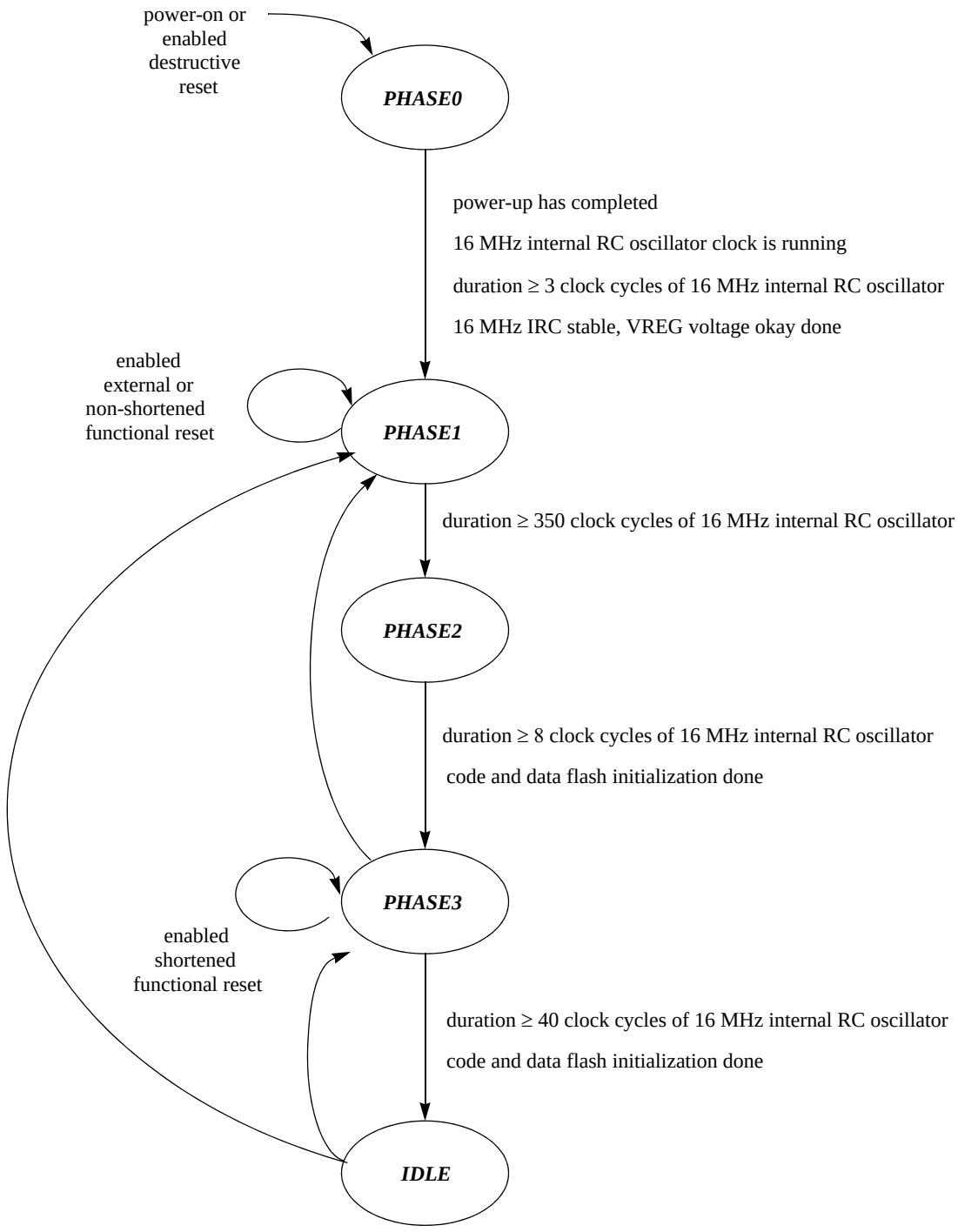


Figure 86. RGM state machine

8.6.1.1 PHASE0

This phase is entered immediately from any phase on a power-on or enabled destructive reset event. The reset state machine exits PHASE0 and enters PHASE1 on verification of the following:

- Power-up has completed
- 16 MHz internal RC oscillator clock is running
- All enabled destructive resets have been processed
- All processes that need to be done in PHASE0 are completed
  - 16 MHz IRC stable, VREG voltage okay
- A minimum of three 16 MHz internal RC oscillator clock cycles have elapsed since power-up completion and the last enabled destructive reset event

8.6.1.2 PHASE1

This phase is entered either on exit from PHASE0 or immediately from PHASE2, PHASE3, or IDLE on a non-masked external or functional reset event if it has not been configured to trigger a ‘short’ sequence. The reset state machine exits PHASE1 and enters PHASE2 on verification of the following:

- All enabled, non-shortened functional resets have been processed
- A minimum of 350 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled external or non-shortened functional reset event

8.6.1.3 PHASE2

This phase is entered on exit from PHASE1. The reset state machine exits PHASE2 and enters PHASE3 on verification of the following:

- All processes that need to be done in PHASE2 are completed
  - code and data flash initialization
- A minimum of eight 16 MHz internal RC oscillator clock cycles have elapsed since entering PHASE2

8.6.1.4 PHASE3

This phase is a entered either on exit from PHASE2 or immediately from IDLE on an enabled, shortened functional reset event. The reset state machine exits PHASE3 and enters IDLE on verification of the following:

- All processes that need to be done in PHASE3 are completed
  - code and data flash initialization
- A minimum of forty 16 MHz internal RC oscillator clock cycles have elapsed since the last enabled, shortened functional reset event

8.6.1.5 IDLE

This is a the final phase and is entered on exit from PHASE3. When this phase is reached, the RGM releases control of the system to the platform and waits for new reset events that can trigger a reset sequence.

8.6.2 Destructive resets

A destructive reset indicates that an event has occurred after which critical register or memory content can no longer be guaranteed.

The status flag associated with a given destructive reset event (RGM_DES[F_<*destructive reset*>] bit) is set when the destructive reset is asserted and the power-on reset is not asserted.

The destructive reset can be optionally disabled by writing bit RGM_DERD[D_<*destructive reset*>].

Note: The RGM_DERD register can be written only once between two power-on reset events.

The device's low-voltage detector threshold ensures that, when 1.2 V low-voltage detected is enabled, the supply is sufficient to have the destructive event correctly propagated through the digital logic. Therefore, if a given destructive reset is enabled, the RGM ensures that the associated reset event will be correctly triggered to the full system. However, if the given destructive reset is disabled and the voltage goes below the digital functional threshold, functionality can no longer be ensured, and the reset may or may not be asserted.

An enabled destructive reset will trigger a reset sequence starting from the beginning of PHASE0.

8.6.3 External reset

The RGM manages the external reset coming from <Overbar>RESET. The detection of a falling edge on <Overbar>RESET will start the reset sequence from the beginning of PHASE1.

The status flag associated with the external reset falling edge event (the RGM_FES[F_EXR] bit) is set when the external reset is asserted and the power-on reset is not asserted.

The external reset can optionally be disabled by writing the RGM_FERD[D_EXR] bit.

Note: The RGM_FERD register can be written only once between two power-on reset events.

An enabled external reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by the external reset. When RGM_FESS.SS_EXR is set, the external reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially when an external reset should not reset the flash.

The RGM may also assert the external reset if the reset sequence was triggered by one of the following:

- A power-on reset
- A destructive reset event
- An external reset event
- A functional reset event configured via the RGM_FBRE register to assert the external reset

In this case, the external reset is asserted until the end of PHASE3.

8.6.4 Functional resets

A functional reset indicates that an event has occurred after which it can be guaranteed that critical register and memory content is still intact.

The status flag associated with a given functional reset event (the RGM_FES[F_<*functional reset*>] bit) is set when the functional reset is asserted and the power-on reset is not asserted.

The functional reset can be optionally disabled by software writing bit RGM_FERD[D_<*functional reset*>].

Note: The RGM_FERD register can be written only once between two power-on reset events.

An enabled functional reset will normally trigger a reset sequence starting from the beginning of PHASE1. Nevertheless, the RGM_FESS register enables the further configuring of the reset sequence triggered by a functional reset. When RGM_FESS[SS_<*functional reset*>] is set, the associated functional reset will trigger a reset sequence starting directly from the beginning of PHASE3, skipping PHASE1 and PHASE2. This can be useful especially in case a functional reset should not reset the flash module.

8.6.5 Alternate event generation

The RGM provides alternative events to be generated on reset source assertion. When a reset source is asserted, the RGM normally enters the reset sequence. Alternatively, it is possible for each reset source event (except the power-on reset event) to be converted from a reset to either a SAFE mode request issued to the MC_ME or to an interrupt request issued to the core.

Alternate event selection for a given reset source is made via the RGM_DERD and RGM_FEAR registers as shown in *Table 76*.

Table 76. RGM alternate event selection

RGM_FERD bit value	RGM_FEAR bit value	Generated event
0	*x*	Reset
1	0	SAFE mode request
1	1	Interrupt request

The alternate event is cleared by deasserting the source of the request (i.e., at the reset source that caused the alternate request) and also clearing the appropriate RGM_FES status bit.

Note: Alternate requests (SAFE mode as well as interrupt requests) are generated asynchronously.

8.6.6 Boot mode capturing

The RGM samples PAD[4:2] whenever <Overbar>RESET is asserted until five 16 MHz internal RC oscillator clock cycles before its deassertion edge. The result of the sampling is used at the beginning of reset PHASE3 for boot mode selection and is retained after <Overbar>RESET has been deasserted for subsequent boots after reset sequences during which <Overbar>RESET is not asserted.

Note: In order to ensure that the boot mode is correctly captured, the application needs to apply the valid boot mode value the entire time that <Overbar>RESET is asserted.

Note: <Overbar>RESET can be asserted as a consequence of the internal reset generation. This will force re-sampling of the boot mode pins. (See Table 75 for details.)

9 Interrupt Controller (INTC)

9.1 Introduction

The INTC provides priority-based preemptive scheduling of interrupt service requests (ISRs). This scheduling scheme is suitable for statically scheduled hard real-time systems. The INTC supports 147 interrupt requests. It is targeted to work with Power Architecture technology and automotive applications where the ISRs nest to multiple levels, but it also can be used with other processors and applications.For high-priority interrupt requests in these target applications, the time from the assertion of the peripheral's interrupt request to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC supports this goal by providing a unique vector for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Because each individual application will have different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that tasks sharing the resource will not preempt each other.

Multiple processors can assert interrupt requests to each other through software configurable interrupt requests. These software configurable interrupt requests can also be used to separate the work involved in servicing an interrupt request into a high-priority portion and a low-priority portion. The high-priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software configurable interrupt request to finish the servicing in a lower priority ISR. Therefore these software configurable interrupt requests can be used instead of the peripheral ISR scheduling a task through the RTOS.

9.2 Features

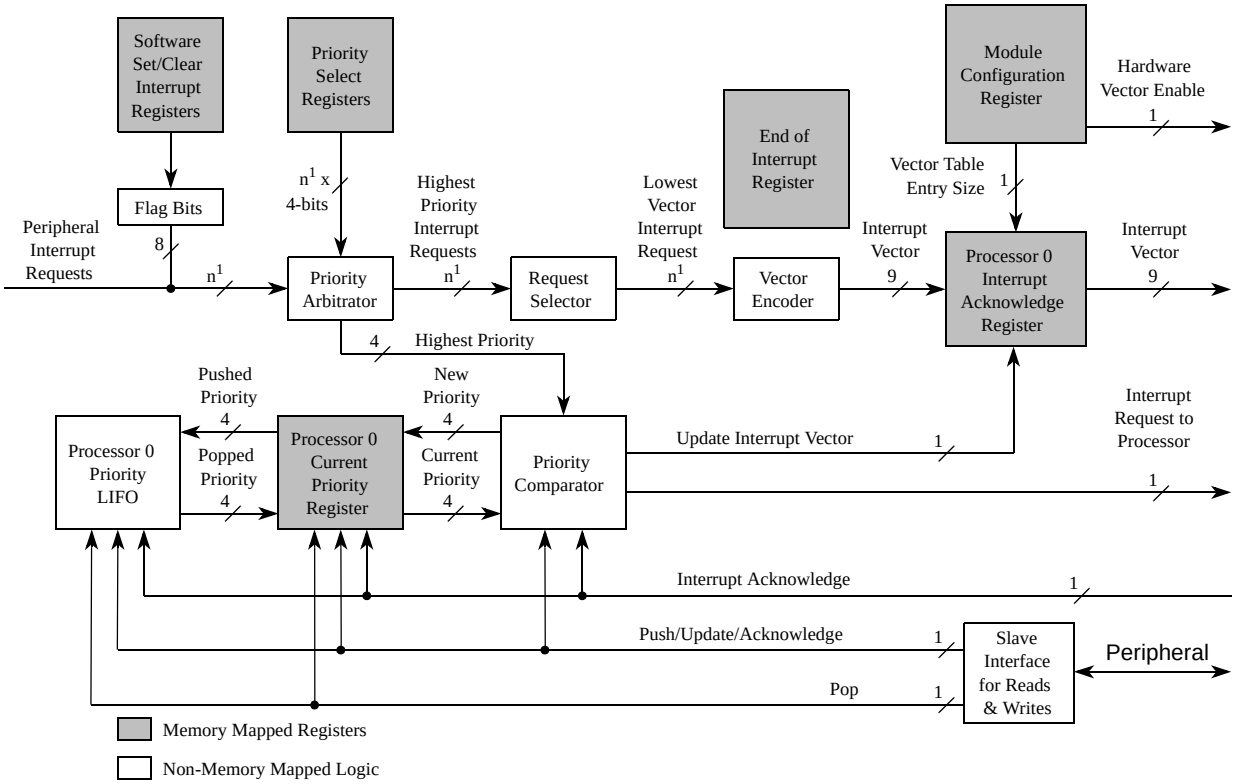
- Supports 139 peripheral interrupt and 8 software-configurable interrupt request sources
- Unique 9-bit vector per interrupt source
- Each interrupt source programmable to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency—3 clock cycles from receipt of interrupt request from peripheral to interrupt request to processor

Table 77. Interrupt sources available

Interrupt sources (147)	Number available
Software	8
ECSM	3
eDMA2x	17
SWT	1
STM	4
SIUL	4
MC_ME	4
MC_RGM	1
XOSC	1
PIT	4
ADC	6
FlexCAN	8
FlexRay	8
eTimer	15
FlexPWM	14
CTU	15
Safety Port	8
DSPI	20
LINFlex	6

9.3 Block diagram

Figure 87 shows a block diagram of the interrupt controller (INTC).



1 The total number of interrupt sources is 147, which includes 16 reserved sources and 8 software sources.

Figure 87. INTC block diagram

9.4 Modes of operation

9.4.1 Normal mode

In normal mode, the INTC has two handshaking modes with the processor: software vector mode and hardware vector mode.

Note: To correctly configure the interrupts in both software and hardware vector mode, the user must also configure the IVPR. The core register IVPR contains the base address for the interrupt handlers. Please refer to the core reference manual for more information.

9.4.1.1 Software vector mode

In software vector mode, the interrupt exception handler software must read a register in the INTC to obtain the vector associated with the interrupt request to the processor. The INTC will use software vector mode for a given processor when its associated HVEN bit in INTC_MCR is negated. The hardware vector enable signal to processor 0 or processor 1 is driven as negated when its associated HVEN bit is negated. The vector is read from

INC_IACKR. Reading the INTC_IACKR negates the interrupt request to the associated processor. Even if a higher priority interrupt request arrived while waiting for this interrupt acknowledge, the interrupt request to the processor will negate for at least one clock. The reading also pushes the PRI value in INTC_CPR onto the associated LIFO and updates PRI in the associated INTC_CPR with the new priority.

Furthermore, the interrupt vector to the processor is driven as all 0s. The interrupt acknowledge signal from the associated processor is ignored.

9.4.1.2 Hardware vector mode

In hardware vector mode, the hardware signals the interrupt vector from the INTC in conjunction with a processor that can use that vector. This hardware causes the first instruction to be executed in handling the interrupt request to the processor to be specific to that vector. Therefore, the interrupt exception handler is specific to a peripheral or software configurable interrupt request rather than being common to all of them. The INTC uses hardware vector mode for a given processor when the associated HVEN bit in the INTC_MCR is asserted. The hardware vector enable signal to the associated processor is driven as asserted. When the interrupt request to the associated processor asserts, the interrupt vector signal is updated. The value of that interrupt vector is the unique vector associated with the preempting peripheral or software configurable interrupt request. The vector value matches the value of the INTVEC field in the INTC_IACKR field in the INTC_IACKR, depending on which processor was assigned to handle a given interrupt source.

The processor negates the interrupt request to the processor driven by the INTC by asserting the interrupt acknowledge signal for one clock. Even if a higher priority interrupt request arrived while waiting for the interrupt acknowledge, the interrupt request to the processor will negate for at least one clock.

The assertion of the interrupt acknowledge signal for a given processor pushes the associated PRI value in the associated INTC_CPR register onto the associated LIFO and updates the associated PRI in the associated INTC_CPR register with the new priority. This pushing of the PRI value onto the associated LIFO and updating PRI in the associated INTC_CPR does not occur when the associated interrupt acknowledge signal asserts and INTC_SSCIR0_3–INTC_SSCIR4_7 is written at a time such that the PRI value in the associated INTC_CPR register would need to be pushed and the previously last pushed PRI value would need to be popped simultaneously. In this case, PRI in the associated INTC_CPR is updated with the new priority, and the associated LIFO is neither pushed or popped.

9.4.1.3 Debug mode

The INTC operation in debug mode is identical to its operation in normal mode.

9.4.1.4 Stop mode

The INTC supports stop mode. The INTC can have its clock input disabled at any time by the clock driver on the device. While its clocks are disabled, the INTC registers are not accessible.

The INTC requires clocking in order for a peripheral interrupt request to generate an interrupt request to the processor.

9.5 Memory map and registers description

9.5.1 Module memory map

Table 78 shows the INTC memory map.

Table 78. INTC memory map

Offset from INTC_BASE 0xFFF4_8000	Register	Location
0x0000	*INTC Module Configuration Register (INTC_MCR)*	*on page 236*
0x0004	Reserved	
0x0008	INTC Current Priority Register (INTC_CPR)	*on page 236*
0x000C	Reserved	
0x0010	INTC Interrupt Acknowledge Register(INTC_IACKR)	*on page 238*
0x0014	Reserved	
0x0018	INTC End-of-Interrupt Register (INTC_EOIR)	*on page 239*
0x001C	Reserved	
0x0020–0x0027	INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)	*on page 239*
0x0028– 0x003C	Reserved	
0x0040–0x011C	*INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)*(1)	*on page 240*
0x0120–0x3FFF	Reserved	

1. The PRI fields are “reserved” for peripheral interrupt requests whose vectors are labeled as Reserved in *Table 85*.

9.5.2 Registers description

With exception of the INTC_SSCI*n* and INTC_PSR*n*, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of 8 bits, aligned 16 bits, misaligned 16 bits to the middle 2 bytes, and aligned 32 bits.

Although INTC_SSCI*n* and INTC_PSR*n* are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC_IACKR are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_SSCIR0_3–INTC_SSCIR4_7 or INTC_EOIR does not affect the operation of the write.

INTC registers are accessible only when the core is in supervisor mode (see *Section 15.4.3: ECSM_reg_protection*).

9.5.2.1 INTC Module Configuration Register (INTC_MCR)

The module configuration register configures options of the INTC.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	VTES	0	0	0	0	HVEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 88. INTC Module Configuration Register (INTC_MCR)

Table 79. INTC_MCR field descriptions

Field	Description
26 VTES	Vector table entry size Controls the number of 0s to the right of INTVEC in *Section 9.5.2.3: INTC Interrupt Acknowledge Register(INTC_IACKR)*. If the contents of INTC_IACKR are used as an address of an entry in a vector table as in software vector mode, then the number of right most 0s will determine the size of each vector table entry. VTES impacts software vector mode operation but also affects INTC_IACKR[INTVEC] position in both hardware vector mode and software vector mode. 0 4 bytes 1 8 bytes
31 HVEN	Hardware vector enable Controls whether the INTC is in hardware vector mode or software vector mode. Refer to *Section 9.4: Modes of operation*, for the details of the handshaking with the processor in each mode. 0 Software vector mode 1 Hardware vector mode

9.5.2.2 INTC Current Priority Register (INTC_CPR)

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 89. INTC Current Priority Register (INTC_CPR)

Table 80. INTC_CPR field descriptions

Field	Description
28–31 PRI[0:3]	Priority PRI is the priority of the currently executing ISR according to the following: 1111 Priority 15—highest priority 1110 Priority 14 1101 Priority 13 1100 Priority 12 1011 Priority 11 1010 Priority 10 1001 Priority 9 1000 Priority 8 0111 Priority 7 0110 Priority 6 0101 Priority 5 0100 Priority 4 0011 Priority 3 0010 Priority 2 0001 Priority 1 0000 Priority 0—lowest priority

The INTC_CPR masks any peripheral or software settable interrupt request set at the same or lower priority as the current value of the INTC_CPR[PRI] field from generating an interrupt request to the processor. When the INTC interrupt acknowledge register (INTC_IACKR) is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When the INTC end-of-interrupt register (INTC_EOIR) is written, the LIFO is popped into the INTC_CPR's PRI field.

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to *Section 9.7.5: Priority ceiling protocol*.

Note: A store to modify the PRI field that closely precedes or follows an access to a shared resource can result in a non-coherent access to the resource. Refer to Section 9.7.5.2: Ensuring coherency, for example code to ensure coherency.

9.5.2.3 INTC Interrupt Acknowledge Register(INTC_IACKR)

Address Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA (least significant 5 bits)					INTVEC[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1 When the VTES bit in INTC_MCR is asserted, INTVEC is shifted to the left one bit. Bit 29 is read as a '0'. VTBA is narrowed to 20 bits in width.

Figure 90. INTC Interrupt Acknowledge Register (INTC_IACKR)

Table 81. INTC_IACKR field descriptions

Field	Description
0–20 or 0–19 VTBA	Vector Table Base Address Can be the base address of a vector table of addresses of ISRs. The VTBA only uses the leftmost 20 bits when the VTES bit in INTC_MCR is asserted.
21–29 or 20–28 INTVEC	Interrupt Vector It is the vector of the peripheral or software configurable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC is updated, whether the INTC is in software or hardware vector mode. **Note:** If INTC_MCR[VTES] = 1, then the INTVEC field is shifted left one position to bits 20–28. VTBA is then shortened by one bit to bits 0–19.

The interrupt acknowledge register provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, the INTC_IACKR has side effects from reads. Therefore, it must not be speculatively read while in this mode. The side effects are the same regardless of the size of the read. Reading the INTC_IACKR does not have side effects in hardware vector mode.

9.5.2.4 INTC End-of-Interrupt Register (INTC_EOIR)

Address Base + 0x0018 Access: Write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 91. INTC End-of-Interrupt Register (INTC_EOIR)

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR is written, the priority last pushed on the LIFO is popped into INTC_CPR. An exception to this behavior is described in *Section 9.4.1.2: Hardware vector mode*. The values and size of data written to the INTC_EOIR are ignored. The values and sizes written to this register neither update the INTC_EOIR contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR.

Reading the INTC_EOIR has no effect on the LIFO.

9.5.2.5 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

Address Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 92. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Address Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 93. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Table 82. INTC_SSCIR[0:7] field descriptions

Field	Description
6, 14, 22, 30 SET[0:7]	Set Flag Bits Writing a ‘1’ sets the corresponding CLR*x* bit. Writing a ‘0’ has no effect. Each SET*x* always will be read as a ‘0’.
7, 15, 23, 31 CLR[0:7]	Clear Flag Bits CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* clears it provided that a ‘1’ is not written simultaneously to its corresponding SET*x* bit. Writing a ‘0’ to CLR*x* has no effect. 0 Interrupt request not pending within INTC 1 Interrupt request pending within INTC

The software set/clear interrupt registers support the setting or clearing of software configurable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a ‘1’ to SET*x* will leave SET*x* unchanged at 0 but sets CLR*x*. Writing a ‘0’ to SET*x* has no effect. CLR*x* is the flag bit. Writing a ‘1’ to CLR*x* clears it. Writing a ‘0’ to CLR*x* has no effect. If a ‘1’ is written simultaneously to a pair of SET*x* and CLR*x* bits, CLR*x* will be asserted, regardless of whether CLR*x* was asserted before the write.

9.5.2.6 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)

Address Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI0				0	0	0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PRI2				0	0	0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 94. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

Address Base + 0x011C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PRI220				0	0	0	0	PRI221			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 95. INTC Priority Select Register 220–221 (INTC_PSR[220:221])

Table 83. INTC_PSR0_3–INTC_PSR220–221 field descriptions

Field	Description
4–7, 12–15, 20–23, 28–31 PRI[0:3]–PRI220:221	Priority Select PRI*x* selects the priority for interrupt requests. Refer to *Section 9.6: Functional description*.

Table 84. INTC Priority Select Register address offsets

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR0_3	0x0040	INTC_PSR112_115	0x00B0
INTC_PSR4_7	0x0044	INTC_PSR116_119	0x00B4
INTC_PSR8_11	0x0048	INTC_PSR120_123	0x00B8
INTC_PSR12_15	0x004C	INTC_PSR124_127	0x00BC
INTC_PSR16_19	0x0050	INTC_PSR128_131	0x00C0
INTC_PSR20_23	0x0054	INTC_PSR132_135	0x00C4
INTC_PSR24_27	0x0058	INTC_PSR136_139	0x00C8
INTC_PSR28_31	0x005C	INTC_PSR140_143	0x00CC
INTC_PSR32_35	0x0060	INTC_PSR144_147	0x00D0
INTC_PSR36_39	0x0064	INTC_PSR148_151	0x00D4
INTC_PSR40_43	0x0068	INTC_PSR152_155	0x00D8
INTC_PSR44_47	0x006C	INTC_PSR156_159	0x00DC
INTC_PSR48_51	0x0070	INTC_PSR160_163	0x00E0
INTC_PSR52_55	0x0074	INTC_PSR164_167	0x00E4
INTC_PSR56_59	0x0078	INTC_PSR168_171	0x00E8
F60_63	0x007C	INTC_PSR172_175	0x00EC
INTC_PSR64_67	0x0080	INTC_PSR176_179	0x00F0

Table 84. INTC Priority Select Register address offsets(Continued)

INTC_PSR*x*_*x*	Offset Address	INTC_PSR*x*_*x*	Offset Address
INTC_PSR68_71	0x0084	INTC_PSR180_183	0x00F4
INTC_PSR72_75	0x0088	INTC_PSR184_187	0x00F8
INTC_PSR76_79	0x008C	INTC_PSR188_191	0x00FC
INTC_PSR80_83	0x0090	INTC_PSR192_195	0x0100
INTC_PSR84_87	0x0094	INTC_PSR196_199	0x0104
INTC_PSR88_91	0x0098	INTC_PSR200_203	0x0108
INTC_PSR92_95	0x009C	INTC_PSR204_207	0x010C
INTC_PSR96_99	0x00A0	INTC_PSR208_211	0x0110
INTC_PSR100_103	0x00A4	INTC_PSR212_215	0x0114
INTC_PSR104_107	0x00A8	INTC_PSR216_219	0x0118
INTC_PSR108_111	0x00AC	INTC_PSR220_221	0x011C

9.6 Functional description

The functional description involves the areas of interrupt request sources, priority management, and handshaking with the processor.

Note: The INTC has no spurious vector support. Therefore, if an asserted peripheral or software settable interrupt request, whose PRIn value in INTC_PSR0–INTC_PSR221 is higher than the PRI value in INTC_CPR, negates before the interrupt request to the processor for that peripheral or software settable interrupt request is acknowledged, the interrupt request to the processor still can assert or will remain asserted for that peripheral or software settable interrupt request. In this case, the interrupt vector will correspond to that peripheral or software settable interrupt request. Also, the PRI value in the INTC_CPR will be updated with the corresponding PRIn value in INTC_PSRn. Furthermore, clearing the peripheral interrupt request's enable bit in the peripheral or, alternatively, setting its mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

Table 85. Interrupt vector table

IRQ #	Offset	Interrupt	Module
On-Platform Peripherals			
Software Interrupts			
0	0x0800	Software configurable flag 0	Software
1	0x0804	Software configurable flag 1	Software
2	0x0808	Software configurable flag 2	Software
3	0x080C	Software configurable flag 3	Software

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
4	0x0810	Software configurable flag 4	Software
5	0x0814	Software configurable flag 5	Software
6	0x0818	Software configurable flag 6	Software
7	0x081C	Software configurable flag 7	Software
8	0x0820	Reserved	
ECSM			
9	0x0824	Platform Flash Bank 0 Abort | Platform Flash Bank 0 Stall | Platform Flash Bank 1 Abort | Platform Flash Bank 1 Stall | Platform Flash Bank 2 Abort | Platform Flash Bank 2 Stall | Platform Flash Bank 3 Abort | Platform Flash Bank 3 Stall	ECSM
DMA2x			
10	0x0828	Combined Error	DMA2x
11	0x082C	Channel 0	DMA2x
12	0x0830	Channel 1	DMA2x
13	0x0834	Channel 2	DMA2x
14	0x0838	Channel 3	DMA2x
15	0x083C	Channel 4	DMA2x
16	0x0840	Channel 5	DMA2x
17	0x0844	Channel 6	DMA2x
18	0x0848	Channel 7	DMA2x
19	0x084C	Channel 8	DMA2x
20	0x0850	Channel 9	DMA2x
21	0x0854	Channel 10	DMA2x
22	0x0858	Channel 11	DMA2x
23	0x085C	Channel 12	DMA2x
24	0x0860	Channel 13	DMA2x
25	0x0864	Channel 14	DMA2x
26	0x0868	Channel 15	DMA2x
27	0x086C	Reserved	
SWT			
28	0x0870	Timeout	Software Watchdog (SWT)
29	0x0874	Reserved	

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
STM			
30	0x0878	Match on channel 0	STM
31	0x087C	Match on channel 1	STM
32	0x0880	Match on channel 2	STM
33	0x0884	Match on channel 3	STM
34	0x0888	Reserved	
ECSM			
35	0x088C	ECC_DBD_PlatformFlash | ECC_DBD_PlatformRAM	ECSM
36	0x0890	ECC_SBC_PlatformFlash | ECC_SBC_PlatformRAM	ECSM
37	0x0894	Reserved	
38	0x0898	Reserved	
39	0x089C	Reserved	
40	0x08A0	Reserved	
SIUL			
41	0x08A4	SIU External IRQ_0	SIUL
42	0x08A8	SIU External IRQ_1	SIUL
43	0x08AC	SIU External IRQ_2	SIUL
44	0x08B0	SIU External IRQ_3	SIUL
45	0x08B4	Reserved	
46	0x08B8	Reserved	
47	0x08BC	Reserved	
48	0x08C0	Reserved	
49	0x08C4	Reserved	
50	0x08C8	Reserved	
MC_ME			
51	0x08CC	Safe Mode Interrupt	Mode Entry module (MC_ME)
52	0x08D0	Mode Transition Interrupt	Mode Entry module (MC_ME)
53	0x08D4	Invalid Mode Interrupt	Mode Entry module (MC_ME)
54	0x08D8	Invalid Mode Configuration	Mode Entry module (MC_ME)
55	0x08DC	Reserved	
MC_RGM			

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
56	0x08E0	Functional and destructive reset alternate event interrupt	Reset Generation Module (MC_RGM)
XOSC			
57	0x08E4	XOSC counter expired	XOSC
PIT			
58	0x08E8	Reserved	
59	0x08EC	PITimer Channel 0	PIT
60	0x08F0	PITimer Channel 1	PIT
61	0x08F4	PITimer Channel 2	PIT
ADC0			
62	0x08F8	ADC_EOC	ADC_0
63	0x08FC	ADC_ER	ADC_0
64	0x0900	ADC_WD	ADC_0
FlexCAN0			
65	0x0904	FLEXCAN_ESR[ERR_INT]	FlexCAN_0
66	0x0908	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_0
67	0x090C	Reserved	
68	0x0910	FLEXCAN_BUF_00_03	FlexCAN_0
69	0x0914	FLEXCAN_BUF_04_07	FlexCAN_0
70	0x0918	FLEXCAN_BUF_08_11	FlexCAN_0
71	0x091C	FLEXCAN_BUF_12_15	FlexCAN_0
72	0x0920	FLEXCAN_BUF_16_31	FlexCAN_0
73	0x0924	Reserved	
DSPI0			
74	0x0928	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_0
75	0x092C	DSPI_SR[EOQF]	DSPI_0
76	0x0930	DSPI_SR[TFFF]	DSPI_0
77	0x0934	DSPI_SR[TCF]	DSPI_0
78	0x0938	DSPI_SR[RFDF]	DSPI_0
LINFlex0			
79	0x093C	LINFlex_RXI	LINFlex_0

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
80	0x0940	LINFlex_TXI	LINFlex_0
81	0x0944	LINFlex_ERR	LINFlex_0
ADC1			
82	0x0948	ADC_EOC	ADC_1
83	0x094C	ADC_ER	ADC_1
84	0x0950	ADC_WD	ADC_1
FlexCAN1			
85	0x0954	FLEXCAN_ESR[ERR_INT]	FlexCAN_1
86	0x0958	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	FlexCAN_1
87	0x095C	FLEXCAN_ESR_WAK	FlexCAN_1
88	0x0960	FLEXCAN_BUF_00_03	FlexCAN_1
89	0x0964	FLEXCAN_BUF_04_07	FlexCAN_1
90	0x0968	FLEXCAN_BUF_08_11	FlexCAN_1
91	0x096C	FLEXCAN_BUF_12_15	FlexCAN_1
92	0x0970	FLEXCAN_BUF_16_31	FlexCAN_1
93	0x0974	Reserved	
DSPI1			
94	0x0978	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_1
95	0x097C	DSPI_SR[EOQF]	DSPI_1
96	0x0980	DSPI_SR[TFFF]	DSPI_1
97	0x0984	DSPI_SR[TCF]	DSPI_1
98	0x0988	DSPI_SR[RFDF]	DSPI_1
LINFlex1			
99	0x098C	LINFlex_RXI	LINFlex_1
100	0x0990	LINFlex_TXI	LINFlex_1
101	0x0994	LINFlex_ERR	LINFlex_1
102	0x0998	Reserved	
103	0x099C	Reserved	
104	0x09A0	Reserved	
105	0x09A4	Reserved	
106	0x09A8	Reserved	
107	0x09AC	Reserved	

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
108	0x09B0	Reserved	
109	0x09B4	Reserved	
110	0x09B8	Reserved	
111	0x09BC	Reserved	
112	0x09C0	Reserved	
113	0x09C4	Reserved	
DSPI2			
114	0x09C8	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_2
115	0x09CC	DSPI_SR[EOQF]	DSPI_2
116	0x09D0	DSPI_SR[TFFF]	DSPI_2
117	0x09D4	DSPI_SR[TCF]	DSPI_2
118	0x09D8	DSPI_SR[RFDF]	DSPI_2
119	0x09DC	Reserved	
120	0x09E0	Reserved	
121	0x09E4	Reserved	
122	0x09E8	Reserved	
123	0x09EC	Reserved	
124	0x09F0	Reserved	
125	0x09F4	Reserved	
126	0x09F8	Reserved	
PIT			
127	0x09FC	PITimer Channel 3	PIT
128	0x0A00	Reserved	
129	0x0A04	Reserved	
130	0x0A08	Reserved	
131	0x0A0C	Reserved	
132	0x0A10	Reserved	
FlexRay			
133	0x0A14	CIFRR.FNEAIF	FlexRay
134	0x0A18	CIFRR.FNEBIF	FlexRay
135	0x0A1C	CIFRR.WUPIF	FlexRay
136	0x0A20	CIFRR.PRIF	FlexRay
137	0x0A24	CIFRR.CHIF	FlexRay

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
138	0x0A28	CIFRR.TBIF	FlexRay
139	0x0A2C	CIFRR.RBIF	FlexRay
140	0x0A30	CIFRR.MIF	FlexRay
141	0x0A34	Reserved	
142	0x0A38	Reserved	
143	0x0A3C	Reserved	
144	0x0A40	Reserved	
145	0x0A44	Reserved	
146	0x0A48	Reserved	
147	0x0A4C	Reserved	
148	0x0A50	Reserved	
149	0x0A54	Reserved	
150	0x0A58	Reserved	
151	0x0A5C	Reserved	
152	0x0A60	Reserved	
153	0x0A64	Reserved	
154	0x0A68	Reserved	
155	0x0A6C	Reserved	
156	0x0A70	Reserved	
eTimer			
157	0x0A74	TC0IR	eTimer_0
158	0x0A78	TC1IR	eTimer_0
159	0x0A7C	TC2IR	eTimer_0
160	0x0A80	TC3IR	eTimer_0
161	0x0A84	TC4IR	eTimer_0
162	0x0A88	TC5IR	eTimer_0
163	0x0A8C	Reserved	
164	0x0A90	Reserved	
165	0x0A94	WTIF	eTimer_0
166	0x0A98	Reserved	
167	0x0A9C	RCF	eTimer_0
168	0x0AA0	TC0IR	eTimer_1
169	0x0AA4	TC1IR	eTimer_1
170	0x0AA8	TC2IR	eTimer_1

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
171	0x0AAC	TC3IR	eTimer_1
172	0x0AB0	TC4IR	eTimer_1
173	0x0AB4	TC5IR	eTimer_1
174	0x0AB8	Reserved	
175	0x0ABC	Reserved	
176	0x0AC0	Reserved	
177	0x0AC4	Reserved	
178	0x0AC8	RCF	eTimer_1
FlexPWM			
179	0x0ACC	RF0	FlexPWM_0
180	0x0AD0	COF0	FlexPWM_0
181	0x0AD4	CAF0	FlexPWM_0
182	0x0AD8	RF1	FlexPWM_0
183	0x0ADC	COF1	FlexPWM_0
184	0x0AE0	CAF1	FlexPWM_0
185	0x0AE4	RF2	FlexPWM_0
186	0x0AE8	COF2	FlexPWM_0
187	0x0AEC	CAF2	FlexPWM_0
188	0x0AF0	RF3	FlexPWM_0
189	0x0AF4	COF3	FlexPWM_0
190	0x0AF8	CAF3	FlexPWM_0
191	0x0AFC	FFLAG	FlexPWM_0
192	0x0B00	REF	FlexPWM_0
CTU			
193	0x0B04	MRS_I	CTU_0
194	0x0B08	T0_I	CTU_0
195	0x0B0C	T1_I	CTU_0
196	0x0B10	T2_I	CTU_0
197	0x0B14	T3_I	CTU_0
198	0x0B18	T4_I	CTU_0
199	0x0B1C	T5_I	CTU_0
200	0x0B20	T6_I	CTU_0
201	0x0B24	T7_I	CTU_0
202	0x0B28	FIFO1_I	CTU_0

Table 85. Interrupt vector table(Continued)

IRQ #	Offset	Interrupt	Module
203	0x0B2C	FIFO2_I	CTU_0
204	0x0B30	FIFO3_I	CTU_0
205	0x0B34	FIFO4_I	CTU_0
206	0x0B38	ADC_I	CTU_0
207	0x0B3C	ERR_I	CTU_0
SafetyPort			
208	0x0B40	FLEXCAN_ESR[ERR_INT]	SafetyPort (FlexCAN)
209	0x0B44	FLEXCAN_ESR_BOFF | FLEXCAN_Transmit_Warning | FLEXCAN_Receive_Warning	SafetyPort (FlexCAN)
210	0x0B48	Reserved	
211	0x0B4C	FLEXCAN_BUF_0_3	SafetyPort (FlexCAN)
212	0x0B50	FLEXCAN_BUF_4_7	SafetyPort (FlexCAN)
213	0x0B54	FLEXCAN_BUF_8_11	SafetyPort (FlexCAN)
214	0x0B58	FLEXCAN_BUF_12_15	SafetyPort (FlexCAN)
215	0x0B5C	FLEXCAN_BUF_16_31	SafetyPort (FlexCAN)
216	0x0B60	Reserved	
DSPI3			
217	0x0B64	DSPI_SR[TFUF] DSPI_SR[RFOF]	DSPI_3
218	0x0B68	DSPI_SR[EOQF]	DSPI_3
219	0x0B6C	DSPI_SR[TFFF]	DSPI_3
220	0x0B70	DSPI_SR[TCF]	DSPI_3
221	0x0B74	DSPI_SR[RFDF]	DSPI_3

9.6.1 Interrupt request sources

The INTC has two types of interrupt requests, peripheral and software configurable. These interrupt requests can assert on any clock cycle.

9.6.1.1 Peripheral interrupt requests

An interrupt event in a peripheral's hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

External interrupts are handled by the SIU (see *Section 11.6.4: External interrupts*).

9.6.1.2 Software configurable interrupt requests

An interrupt request is triggered by software by writing a '1' to a SET*x* bit in INTC_SSCIR0_3–INTC_SSCIR4_7. This write sets the corresponding flag bit, CLR*x*, resulting in the interrupt request. The interrupt request is cleared by writing a '1' to the CLR*x* bit.

The time from the write to the SET*x* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

9.6.1.3 Unique vector for each interrupt request source

Each peripheral and software configurable interrupt request is assigned a hardwired unique 9-bit vector. Software configurable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests. The peripheral interrupt request input ports at the boundary of the INTC block are assigned specific hardwired vectors within the INTC (see *Table 77*).

9.6.2 Priority management

The asserted interrupt requests are compared to each other based on their PRI*x* values set in *Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)*. The result is compared to PRI in the associated INTC_CPR. The results of those comparisons manage the priority of the ISR executed by the associated processor. The associated LIFO also assists in managing that priority.

9.6.2.1 Current priority and preemption

The priority arbitrator, selector, encoder, and comparator subblocks shown in *Figure 87* compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software configurable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for INTC interrupt acknowledge register (INTC_IACKR), and if in hardware vector mode, for the interrupt vector provided to the processor.

9.6.2.1.1 Priority arbitrator subblock

The priority arbitrator subblock for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software

configurable. The output of the priority arbitrator subblock is the highest of those priorities assigned to a given processor. Also, any interrupt requests that have this highest priority are output as asserted interrupt requests to the associated request selector subblock.

9.6.2.1.2 Request selector subblock

If only one interrupt request from the associated priority arbitrator subblock is asserted, then it is passed as asserted to the associated vector encoder subblock. If multiple interrupt requests from the associated priority arbitrator subblock are asserted, only the one with the lowest vector passes as asserted to the associated vector encoder subblock. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software configurable interrupt requests.

9.6.2.1.3 Vector encoder subblock

The vector encoder subblock generates the unique 9-bit vector for the asserted interrupt request from the request selector subblock for the associated processor.

9.6.2.1.4 Priority comparator subblock

The priority comparator submodule compares the highest priority output from the priority arbitrator submodule with PRI in INTC_CPR. If the priority comparator submodule detects that this highest priority is higher than the current priority, then it asserts the interrupt request to the processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in INTC_CPR or the PRI value in INTC_CPR is lowered below this highest priority. This highest priority then becomes the new priority that will be written to PRI in INTC_CPR when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n* are zero will not cause a preemption because their PRI*n* will not be higher than PRI in INTC_CPR.

9.6.2.2 Last-in first-out (LIFO)

The LIFO stores the preempted PRI values from the INTC_CPR. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the INTC_CPR does not need to be loaded from the INTC_CPR and stored onto the context stack. Likewise at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the INTC_CPR.

The PRI value in the INTC_CPR is pushed onto the LIFO when the INTC_IACKR is read in software vector mode or the interrupt acknowledge signal from the processor is asserted in hardware vector mode. The priority is popped into PRI in the INTC_CPR whenever the INTC_EOIR is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the priorities first pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than it is pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

9.6.3 Handshaking with processor

9.6.3.1 Software vector mode handshaking

This section describes handshaking in software vector mode.

9.6.3.1.1 Acknowledging interrupt request to processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode and the handshake near the end of the interrupt exception handler, is shown in *Figure 96*. The INTC examines the peripheral and software configurable interrupt requests. When it finds an asserted peripheral or software configurable interrupt request with a higher priority than PRI in the associated INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the associated INTC_IACKR is updated with the preempting interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The rest of handshaking process is described in *Section 9.4.1.1: Software vector mode*.

9.6.3.1.2 End of interrupt exception handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR) must be written.When written, the associated LIFO is popped so the preempted priority is restored into PRI of the INTC_CPR. Before it is written, the peripheral or software configurable flag bit must be cleared so that the peripheral or software configurable interrupt request is negated.

Note: To ensure proper operation across all eSys MCUs, execute an MBAR or MSYNC instruction between the access to clear the flag bit and the write to the INTC_EOIR.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer even be asserted. When PRI in INTC_CPR is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or any other asserted peripheral or software settable interrupt request at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor will return to the instruction address that it was to next execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler's prolog or epilog.

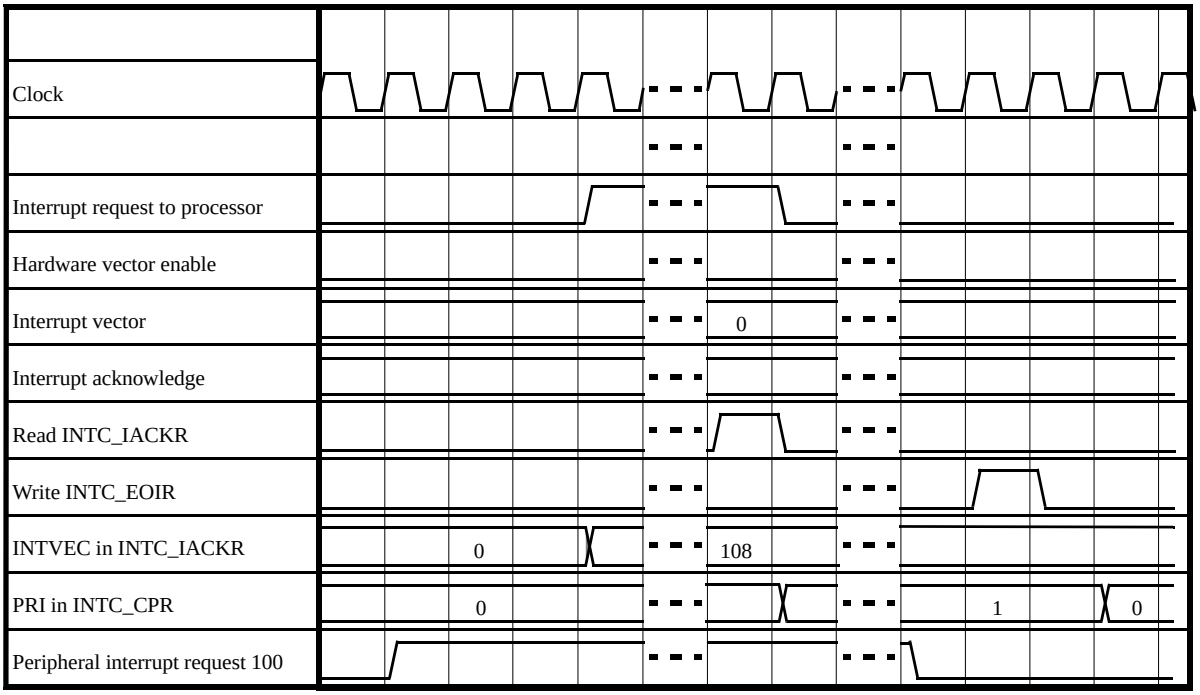


Figure 96. Software vector mode handshaking timing diagram

9.6.3.2 Hardware vector mode handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode, along with the handshaking near the end of the interrupt exception handler, is shown in *Figure 97*. As in software vector mode, the INTC examines the peripheral and software settable interrupt requests, and when it finds an asserted one with a higher priority than PRI in INTC_CPR, it asserts the interrupt request to the processor. The INTVEC field in the INTC_IACKR is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. In addition, the value of the interrupt vector to the processor matches the value of the INTVEC field in the INTC_IACKR. The rest of the handshaking is described in" *Section 9.4.1.2: Hardware vector mode*.

The handshaking near the end of the interrupt exception handler, that is the writing to the INTC_EOIR, is the same as in software vector mode. Refer to *Section 9.6.3.1.2: End of interrupt exception handler*.

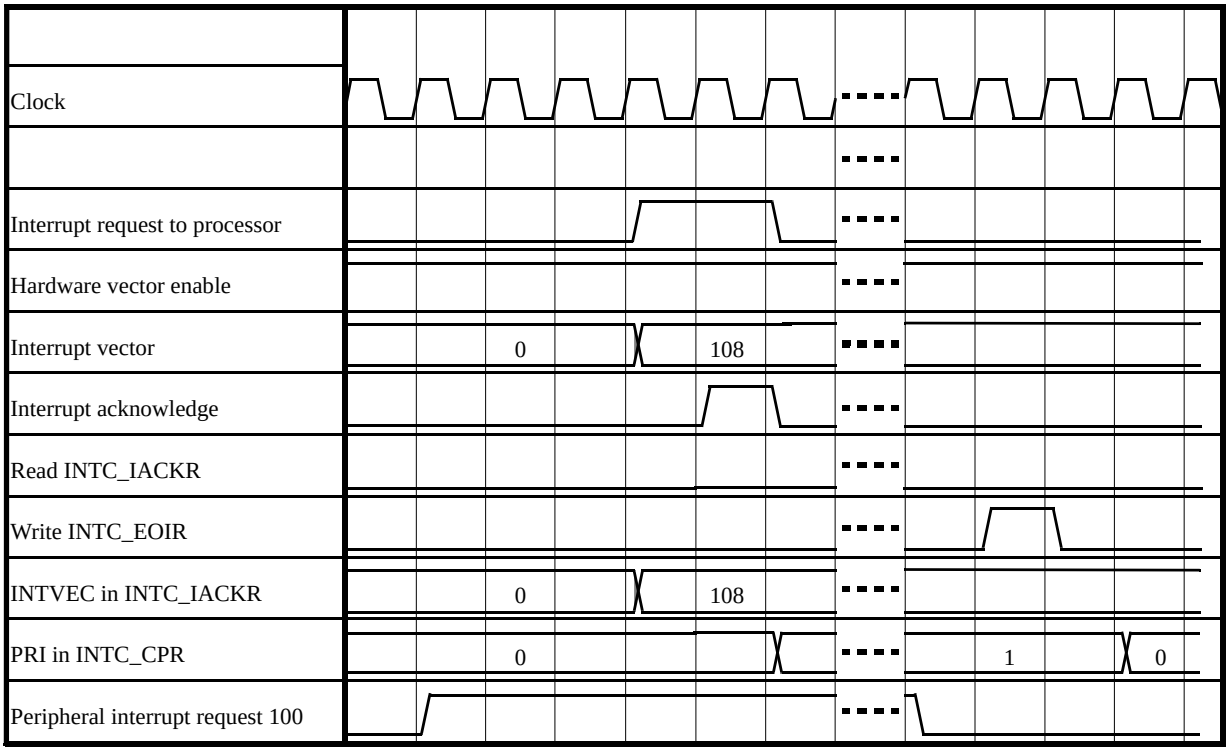


Figure 97. Hardware vector mode handshaking timing diagram

9.7 Initialization/application information

9.7.1 Initialization flow

After exiting reset, all of the PRI*n* fields in *Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)* will be zero, and PRI in INTC current priority register (INTC_CPR) will be 15. These reset values will prevent the INTC from asserting the interrupt request to the processor. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated. An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is: interrupt_request_initialization:

```
interrupt_request_initialization:
configure VTES and HVEN in INTC_MCR
configure VTBA in INTC_IACKR
raise the PRIn fields in INTC_PSRn
set the enable bits or clear the mask bits for the peripheral interrupt
requests
lower PRI in INTC_CPR to zero
enable processor recognition of interrupts
```

9.7.2 Interrupt exception handler

These example interrupt exception handlers use Power Architecture assembly code.

9.7.2.1 Software vector mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1
lis r3,INTC_IACKR@ha # form adjusted upper half of INTC_IACKR address
lwz r3,INTC_IACKR@l(r3) # load INTC_IACKR, which clears request to
processor
lwz r3,0x0(r3) # load address of ISR from vector table
wrteei 1   # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr r3    # move INTC_IACKR contents into link register
blrl       # branch to ISR; link register updated with epilog
           # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar       # ensure store to clear flag bit has completed
lis r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li r4,0x0  # form 0 to write to INTC_EOIR
wrteei 0   # disable processor recognition of interrupts
stw r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame

rfi



vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
.
.
.
```

```
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit that drives interrupt request to INTC

blr # return to epilog
```

9.7.2.2 Hardware vector mode

This interrupt exception handler is useful with processor and system bus implementations that support a hardware vector. This example assumes that each interrupt_exception_handler*x* only has space for four instructions, and therefore a branch to interrupt_exception_handler_continued*x* is needed.

```
interrupt_exception_handlerx:
b interrupt_exception_handler_continuedx# 4 instructions available, branch
to continue
interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei 1 # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl ISRx # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the
disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth
at the cost of
# postponing the servicing of the next interrupt request.
mbar    # ensure store to clear flag bit has completed
lis r3,INTC_EOIR@ha # form adjusted upper half of INTC_EOIR address
li r4,0x0 # form 0 to write to INTC_EOIR
wrteei 0 # disable processor recognition of interrupts
stw r4,INTC_EOIR@l(r3) # store to INTC_EOIR, informing INTC to lower
priority

code to restore SRR0 and SRR1, restore working registers, and delete stack
frame
```

```
rfi

ISRx:
code to service the interrupt event
code to clear flag bit that drives interrupt request to INTC
blr     # branch to epilog
```

9.7.3 ISR, RTOS, and task hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR) having a value of 0. The RTOS will execute the tasks according to whatever priority scheme that it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR while the shared resource is being accessed.

An ISR whose PRI*n* in *Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)* has a value of 0 will not cause an interrupt request to the processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit will cause it to remain negated, which consequently also will not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR will not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

9.7.4 Order of execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software configurable interrupt requests. However, if multiple peripheral or software configurable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software configurable interrupt requests asserted.

The example in *Table 86* shows the order of execution of both ISRs with different priorities and the same priority

Table 86. Order of ISR execution example

Step #	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108 (1)	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR.						X	0
12	RTOS continues execution.	X						0

1. ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software configurable interrupt requests.

9.7.5 Priority ceiling protocol

9.7.5.1 Elevating priority

The PRI field in INTC_CPR is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR to 3, the ceiling of all of the ISR priorities. After they

release the resource, the PRI value in INTC_CPR can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

9.7.5.2 Ensuring coherency

A scenario can cause non-coherent accesses to the shared resource. For example, ISR1 and ISR2 are both running on the same core and both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing and writes to the INTC_CPR. The instruction following this store is a store to a value in a shared coherent data block. Either immediately before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted. As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

OSEK uses the GetResource and ReleaseResource system services to manage access to a shared resource. To prevent corruption of a coherent data block, modifications to PRI in INTC_CPR can be made by those system services with the code sequence:

```
disable processor recognition of interrupts
PRI modification
enable processor recognition of interrupts
```

9.7.6 Selecting priorities according to request rates and deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs that have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2, which has a higher priority than ISR3; however, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which may be less than the number of ISRs. In this case, the ISRs should be grouped with other ISRs that have similar deadlines. For example, a priority could be allocated for every time the request rate doubles. ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 µs would share a priority, ISRs with request rates around 250 µs would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be included, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor's ability to meet its deadlines. However, reducing the number of priorities can reduce the size and latency through the interrupt controller. It also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

9.7.7 Software configurable interrupt requests

The software configurable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and they may also be used by processors to interrupt other processors in a multiple processor system.

9.7.7.1 Scheduling a lower priority portion of an ISR

A portion of an ISR needs to be executed at the PRI*x* value in *Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)*, which becomes the PRI value in INTC_CPR with the interrupt acknowledge. The ISR, however, can have a portion that does not need to be executed at this higher priority. Therefore, executing the later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor's ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*x* bit in INTC_SSCIR0_3–INTC_SSCIR4_7. Writing a '1' to SET*x* causes a software configurable interrupt request. This software configurable interrupt request will usually have a lower PRI*x* value in the INTC_PSR*x_x* and will not cause preemptive scheduling inefficiencies. After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

9.7.7.2 Scheduling an ISR on another processor

Because the SET*x* bits in the INTC_SSCIR*x_x* are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software configurable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR*x* bit in INTC_SSCIR*x_x* is asserted before again writing a '1' to the SET*x* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. To do this, the first processor writes a '1' to a SET*x* bit on the second processor. After accessing the block of data, the second processor clears the corresponding CLR*x* bit and then writes 1 to a SET*x* bit on the first processor, informing it that it can now access the block of data.

9.7.8 Lowering priority within an ISR

A common method for avoiding preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities (see *Section 9.7.7.1: Scheduling a lower priority portion of an ISR*) is to lower the current priority. However, the INTC has a LIFO whose depth is determined by the number of priorities.

Note: Lowering the PRI value in INTC_CPR within an ISR to below the ISR's corresponding PRI value in Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221) allows more preemptions than the LIFO depth can support.

Therefore, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

9.7.9 Negating an interrupt request outside of its ISR

9.7.9.1 Negating an interrupt request as a side effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and their corresponding interrupt requests. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

9.7.9.2 Negating multiple interrupt requests in one ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

9.7.9.3 Proper setting of interrupt request priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software configurable interrupt requests for these other flag bits must be selected properly. Their PRI*x* values in *Section 9.5.2.6: INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR220_221)* must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC_SSCIR0_3–INTC_SSCIR4_7 as the clearing of the flag bit that caused the present ISR to be executed (see *Section 9.6.3.1.2: End of interrupt exception handler*).

A flag bit whose enable bit or mask bit negates its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request's PRI*x* value in INTC_PSR*x*_*x*.

9.7.10 Examining LIFO contents

In normal mode, the user does not need to know the contents of the LIFO. He may not even know how deeply the LIFO is nested. However, if he wants to read the contents, such as in debug mode, they are not memory mapped. The contents can be read by popping the LIFO and reading the PRI field in either INTC_CPR. The code sequence is:

```
pop_lifo:
store to INTC_EOIR
load INTC_CPR, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to
pop_lifo
```

When the examination is complete, the LIFO can be restored using this code sequence:

```
push_lifo:
```

```
load stacked PRI value and store to INTC_CPR
load INTC_IACKR
if stacked PRI values are not depleted, branch to push_lifo
```

10 System Status and Configuration Module (SSCM)

10.1 Introduction

10.1.1 Overview

The System Status and Configuration Module (SSCM), pictured in *Figure 98*, provides central device functionality.

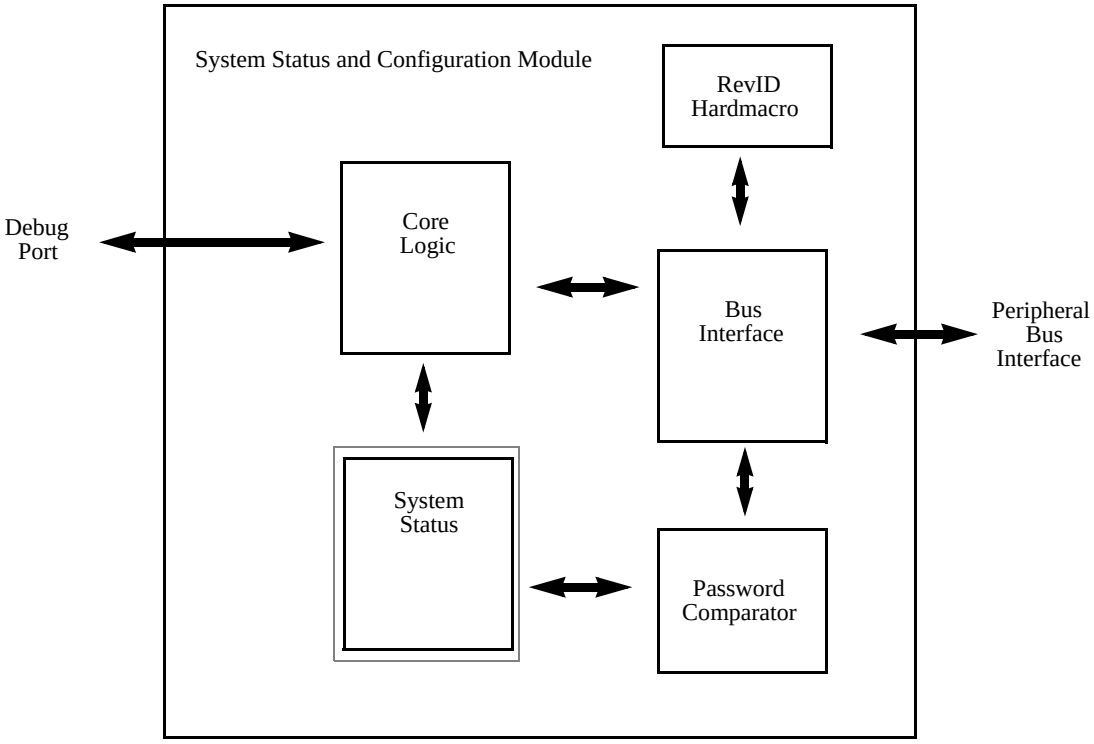


Figure 98. SSCM block diagram

10.1.2 Features

The SSCM includes these features:

- System configuration and status
  - Memory sizes/status
  - Device mode and security status
  - Determine boot vector
  - Search Code Flash for bootable sector
  - DMA status
- Debug status port enable and selection
- Bus and peripheral abort enable/disable

10.1.3 Modes of operation

The SSCM operates identically in all system modes.

10.2 Memory map and register description

This section provides a detailed description of all memory-mapped registers in the SSCM.

10.2.1 Memory map

Table 87 shows the memory map for the SSCM. Note that all addresses are offsets; the absolute address may be calculated by adding the specified offset to the base address of the SSCM.

Table 87. SSCM memory map

Offset from SSCM_BASE (0xC3FD_8000)	Register	Location
0x0000	STATUS—System Status register	*on page 266*
0x0002	MEMCONFIG—System Memory Configuration register	*on page 266*
0x0004	Reserved (Reads/Writes have no effect)	
0x0006	ERROR—Error Configuration register	*on page 267*
0x0008	DEBUGPORT—Debug Status Port register	*on page 268*
0x000A	Reserved (Reads/Writes have no effect)	
0x000C	PWCMPH—Password Comparison High Word register	*on page 269*
0x0010	PWCMPL—Password Comparison Low Word register	*on page 269*
0x0014–0x3FFF	Reserved	

All registers are accessible via 8, 16 or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, the MEMCONFIG register is accessible by a 16-bit read/write to address Base + 0x0002, but performing a 16-bit access to Base + 0x0003 is illegal.

10.2.2 Register description

Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of the registers is MSB = 0, however the numbering of the internal fields is LSB = 0, for example, register SSCM_STATUS[8] = BMODE[2].

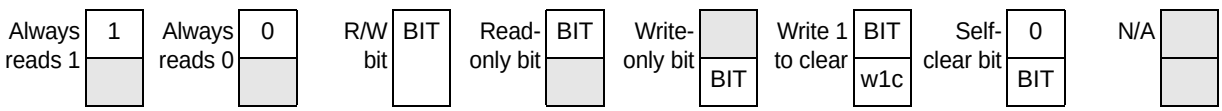


Figure 99. Key to register fields

10.2.2.1 System Status register (STATUS)

The system status register is a read-only register that reflects the current state of the system.

Address: Base + 0x0000 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	NXEN	PUB	SEC	0	BMODE[2:0]			0	ABD	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0/1	0/1	0/1	0	0	0	0	0

Figure 100. Status (STATUS) register

Table 88. STATUS allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8, or 0xC).

Table 89. STATUS field descriptions

Field	Description
NXEN	Nexus enabled
PUB	Public Serial Access Status This bit indicates whether serial boot mode with public password is allowed. 1: Serial boot mode with public password allowed 0: Serial boot mode with private Flash password allowed, provided the key has not been swallowed
SEC	Security Status This bit reflects the current security state of the Flash. 1: Flash secured 0: Flash not secured
BMODE [2:0]	Device Boot Mode 000: Reserved for FlexRay boot serial boot loader 001: Legacy bootstrap via CAN (no autobaud) 010: Legacy bootstrap via UART (no autobaud) 011: Single Chip 100–111: Reserved This field is updated only during reset.
ABD	Autobaud Indicates that autobaud detection is active when in SCI or CAN serial boot loader mode. No meaning in other modes.

10.2.2.2 System Memory Configuration register (MEMCONFIG)

The system memory configuration register is a read-only register that reflects the memory configuration of the system.

Address: Base + 0x0002 Access: Read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	1	0	0	0	1	0	0	1	1	IVLD	0	0	1	1	DVLD
W																
Reset	0	1	0	0	0	1	0	0	1	1	1	0	0	1	1	1

Figure 101. System memory configuration (MEMCONFIG) register

Table 90. MEMCONFIG field descriptions

Field	Description
0:9	Reserved (Read only) A write to these bits has no effect. A read of this bit field is always 0100010011.
IVLD	CFlash Valid This bit identifies whether or not the on-chip CFlash is accessible in the system memory map. The Flash may not be accessible due to security limitations. 1: CFlash accessible 0: CFlash not accessible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.
11:14	Reserved (Read only) A write to these bits has no effect. A read of this bit field is always 0011.
DVLD	DFlash Valid This bit identifies whether or not the on-chip DFlash is visible in the system memory map. The Flash may not be accessible due to security limitations. 1: DFlash visible 0: DFlash not visible Note: This is a status bit only and writing to this bit does not enable the CFlash if it has been disabled due to specific mode of operation.

Table 91. MEMCONFIG allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed (also reads STATUS register)
Write	Not allowed	Not allowed	Not allowed

10.2.2.3 Error Configuration (ERROR) register

The Error Configuration register is a read-write register that controls the error handling of the system.

Address: Base + 0x0006 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAE	RAE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 102. Error Configuration (ERROR) register

Table 92. ERROR field descriptions

Field	Description
PAE	Peripheral Bus Abort Enable This bit enables bus aborts on any access to a peripheral slot that is not used on the device. This feature is intended to aid in debugging when developing application code. 1: Illegal accesses to non-existing peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to non-existing peripherals do not produce a Prefetch or Data Abort exception.
RAE	Register Bus Abort Enable This bit enables bus aborts on illegal accesses to off-platform peripherals. Illegal accesses are defined as reads or writes to reserved addresses within the address space for a particular peripheral. This feature is intended to aid in debugging when developing application code. 1: Illegal accesses to peripherals produce a Prefetch or Data Abort exception. 0: Illegal accesses to peripherals do not produce a Prefetch or Data Abort exception.

Note: Transfers to Peripheral Bus resources may be aborted even before they reach the Peripheral Bus (i.e., at the PRIDGE level). In this case, the PAE and RAE register bits will have no effect on the abort.

Table 93. ERROR allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit
Read	Allowed	Allowed	Allowed
Write	Allowed	Allowed	Not allowed

10.2.2.4 Debug Status Port (DEBUGPORT) register

The Debug Status Port register provides debug data on a set of pins.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	DEBUG_MODE [2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 103. Debug Status Port (DEBUGPORT) register

Table 94. DEBUGPORT field descriptions

Field	Description
13-15 DEBUG_MODE[2:0]	Debug Status Port Mode This field selects the alternate debug functionality for the Debug Status Port. 000: No alternate functionality selected 001: Mode 1 selected 010: Mode 2 selected 011: Mode 3 selected 100: Mode 4 selected 101: Mode 5 selected 110: Mode 6 selected 111: Mode 7 selected *Table 95* describes the functionality of the Debug Status Port in each mode.

Table 95. Debug Status Port modes

Pin (1)	Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	Mode 6	Mode 7
0	STATUS[0]	STATUS[8]	MEMCONFIG[0]	MEMCONFIG[8]	Reserved	Reserved	Reserved
1	STATUS[1]	STATUS[9]	MEMCONFIG[1]	MEMCONFIG[9]	Reserved	Reserved	Reserved
2	STATUS[2]	STATUS[10]	MEMCONFIG[2]	MEMCONFIG[10]	Reserved	Reserved	Reserved
3	STATUS[3]	STATUS[11]	MEMCONFIG[3]	MEMCONFIG[11]	Reserved	Reserved	Reserved
4	STATUS[4]	STATUS[12]	MEMCONFIG[4]	MEMCONFIG[12]	Reserved	Reserved	Reserved
5	STATUS[5]	STATUS[13]	MEMCONFIG[5]	MEMCONFIG[13]	Reserved	Reserved	Reserved
6	STATUS[6]	STATUS[14]	MEMCONFIG[6]	MEMCONFIG[14]	Reserved	Reserved	Reserved
7	STATUS[7]	STATUS[15]	MEMCONFIG[7]	MEMCONFIG[15]	Reserved	Reserved	Reserved

1. All signals are active high, unless otherwise noted

Table 96. DEBUGPORT allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Not allowed
Write	Allowed	Allowed	Not allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

10.2.2.5 Password comparison registers

These registers allow to unsecure the device, if the correct password is known.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_HI[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 104. Password Comparison Register High Word (PWCMPH) register

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	PWD_LO[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 105. Password Comparison Register Low Word (PWCMPL) register

Table 97. PWCMPH/L field descriptions

Field	Description
PWD_HI[31:0]	Upper 32 bits of the password
PWD_LO[31:0]	Lower 32 bits of the password

Table 98. PWCMPH/L allowed register accesses

Access type	Access width		
	8-bit	16-bit	32-bit(1)
Read	Allowed	Allowed	Allowed
Write	Not allowed	Not allowed	Allowed

1. All 32-bit accesses must be aligned to 32-bit addresses (i.e., 0x0, 0x4, 0x8 or 0xC).

10.3 Functional description

The primary purpose of the SSCM is to provide information about the current state and configuration of the system that may be useful for configuring application software and for debug of the system.

10.4 Initialization/application information

10.4.1 Reset

The reset state of each individual bit is shown in *Section 10.2.2: Register description*.

11 System Integration Unit Lite (SIUL)

11.1 Introduction

This chapter describes the System Integration Unit Lite (SIUL), which is used for the management of the pads and their configuration. It controls the multiplexing of the alternate functions used on all pads and is responsible for managing the external interrupts to the device.

11.2 Overview

The System Integration Unit Lite (SIUL) controls the MCU pad configuration, ports, general-purpose input and output (GPIO) signals and external interrupts with trigger event configuration. *Figure 106* is a block diagram of the SIUL and its interfaces to other system components.

The module provides dedicated general-purpose pads that can be configured as either inputs or outputs. When configured as an output, you can write to an internal register to control the state driven on the associated output pad. When configured as an input, you can detect the state of the associated pad by reading the value from an internal register. When configured as an input and output, the pad value can be read back, which can be used a method of checking if the written value appeared on the pad.

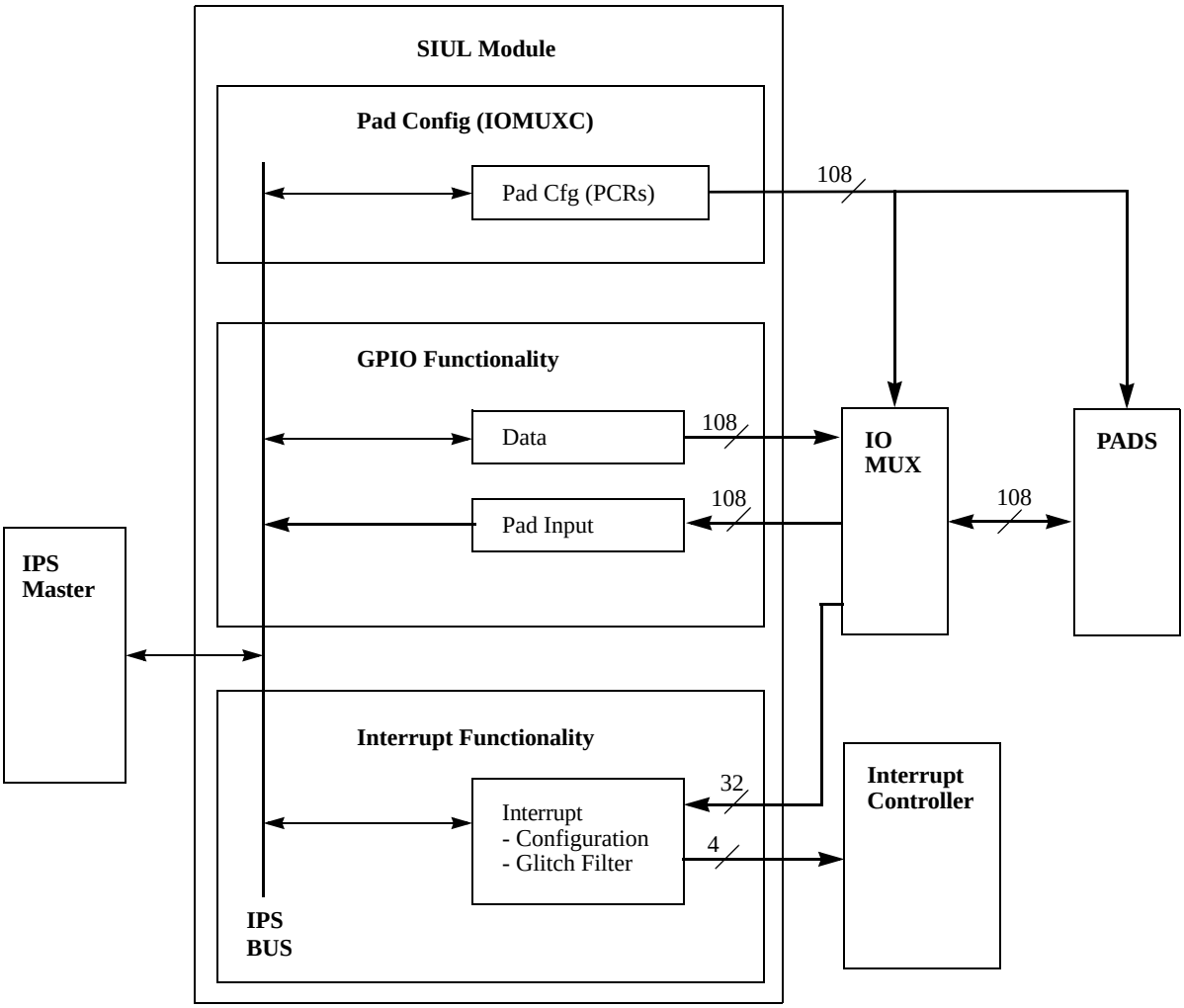


Figure 106. System Integration Unit Lite block diagram

11.3 Features

The System Integration Unit Lite provides these features:

- GPIO
  - GPIO function on up to 106 I/O pins
  - Dedicated input and output registers for each GPIO pin
- External interrupts
  - 4 system interrupt vectors for up to 32 interrupt sources
  - 32 programmable digital glitch filters
  - Independent interrupt mask
  - Edge detection
- System configuration
  - Pad configuration control

11.3.1 Register protection

Most of the configuration registers of the System Integration Unit Lite are protected from accidental writes, see *Appendix A: Registers Under Protection*.

11.4 External signal description

The pad configuration allows flexible, centralized control of the pin electrical characteristics of the MCU with the GPIO control providing centralized general purpose I/O for an MCU that multiplexes GPIO with other signals at the I/O pads. These other signals, or alternate functions, will normally be the peripherals functions. The internal multiplexing allows user selection of the input to chip-level signal multiplexers. Each GPIO port communicates via 16 I/O channels. In order to use the pad as a GPIO, the corresponding Pad Configuration Registers (PCR[0:107]) for all pads used in the port must be configured as GPIO rather than as the alternate pad function.

Table 99 lists the external pins used by the SIUL.

Table 99. SIUL signal properties

GPIO category	Name	I/O direction	Function
System configuration	GPIO[0:19], GPIO[22], GPIO[35:62] GPIO[77:107]	Input/Output	General-purpose input/output
	GPIO[23:34], GPIO[63:76]	Input	Analog precise channel pins
External interrupt	EIRQ[0:31]	Input	Pins with External Interrupt Request functionality. Please refer to the signal description chapter of this reference manual for details.

11.4.1 Detailed signal descriptions

11.4.1.1 General-purpose I/O pins (GPIO[0:107])

The GPIO pins provide general-purpose input and output function. The GPIO pins are generally multiplexed with other I/O pin functions. Each GPIO input and output is separately controlled by an input (GPDI*n_n*) or output (GPDO*n_n*) register.

See *Section 11.5.2.10: GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])* and *Section 11.5.2.11: GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])*.

11.4.1.2 External interrupt request input pins (EIRQ[0:31])

The EIRQ[0:31] are connected to the SIUL inputs. Rising or falling edge events are enabled by setting the corresponding bits in the “n” SIUL_IREER or the SIUL_IFEER register. See *Section 11.5.2.5: Interrupt Rising-Edge Event Enable Register (IREER)* and *Section 11.5.2.6: Interrupt Falling-Edge Event Enable Register (IFEER)*.

11.5 Memory map and register description

This section provides a detailed description of all registers accessible in the SIUL module.

11.5.1 SIUL memory map

Table 100 lists the SIUL registers.

Table 100. SIUL memory map

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x0000–0x0003	Reserved	
0x0004	MCU ID Register #1 (MIDR1)	*on page 276*
0x0008	MCU ID Register #2 (MIDR2)	*on page 277*
0x000C–0x0013	Reserved	
0x0014	Interrupt Status Flag Register (ISR)	*on page 278*
0x0018	Interrupt Request Enable Register (IRER)	*on page 279*
0x001C–0x0027	Reserved	
0x0028	Interrupt Rising-Edge Event Enable Register (IREER)	*on page 279*
0x002C	Interrupt Falling-Edge Event Enable Register (IFEER)	*on page 280*
0x0030	Interrupt Filter Enable Register (IFER)	*on page 281*
0x0034–0x003F	Reserved	
0x0040–0x0116	Pad Configuration Registers (PCR[0:107])	*on page 281*
0x0118–0x04FF	Reserved	
0x0500–0x0520	Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])	*on page 283*
0x0524–0x05FF	Reserved	

Table 100. SIUL memory map(Continued)

Offset from SIUL_BASE (0xC3F9_0000)	Register	Location
0x0600–0x0668	GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])	*on page 287*
0x066C–0x07FF	Reserved	
0x0800–0x0868	GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])	*on page 288*
0x086C–0x0BFF	Reserved	
0x0C00–0x0C0C	Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])	*on page 289*
0x0C10–0x0C3F	Reserved	
0x0C40–0x0C4C	Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])	*on page 289*
0x0C50–0x0C7F	Reserved	
0x0C80–0x0C98	Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])	*on page 290*
0x0C9C–0x0FFF	Reserved	
0x1000–0x107C	Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])	*on page 291*
0x1080	Interrupt Filter Clock Prescaler Register (IFCPR)	*on page 291*
0x1084–0x3FFF	Reserved	

Note: A transfer error will be issued when trying to access completely reserved register space.

11.5.2 Register description

This section describes in address order all the SIUL registers. Each description includes a standard register diagram. Details of register bit and field function follow the register diagrams, in bit order. The numbering convention of register is MSB = 0, however the numbering of internal field is LSB = 0, for example PARTNUM[5] = MIDR1[10].

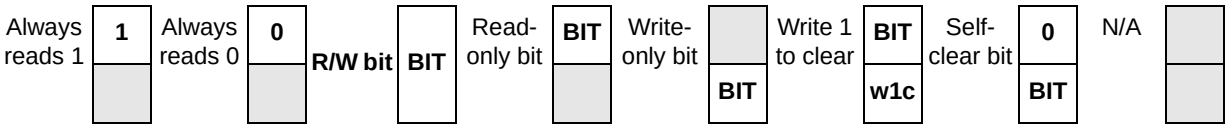


Figure 107. Key to register fields

11.5.2.1 MCU ID Register #1 (MIDR1)

This register contains the part number and the package ID of the device.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM[15:0]															
W																
Reset	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSP	PKG[4:0]					0	0	MAJOR_MASK[3:0](1)				MINOR_MASK[3:0]1			
W																
Reset	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0

Figure 108. MCU ID Register #1 (MIDR1)

1. See *Table 101*.

Table 101. MIDR1 field descriptions

Field	Description
PARTNUM[15:0]	MCU Part Number Device part number of the MCU. 0101_0110_0000_0001: 192 KB 0101_0110_0000_0010: 256 KB 0101_0110_0000_0011: 320/384 KB 0101_0110_0000_0100: 512 KB For the full part number this field needs to be combined with MIDR2.PARTNUM[23:16]
CSP	Always reads back 0
PKG[4:0]	Package Settings Can by read by software to determine the package type that is used for the particular device: 01001: 100-pin LQFP 01101: 144-pin LQFP
MAJOR_MASK[3:0]	Major Mask Revision Counter starting at 0x0. Incremented each time a resynthesis is done.
MINOR_MASK[3:0]	Minor Mask Revision Counter starting at 0x0. Incremented each time a mask change is done.

11.5.2.2 MCU ID Register #2 (MIDR2)

This register contains additional configuration information about the device.

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SF	FLASH_SIZE_1[3:0]				FLASH_SIZE_2[3:0]			0	0	0	0	0	0	0	0
W																
Reset	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PARTNUM[23:16]								0	0	0	EE	0	0	0	FF
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure 109. MCU ID Register #2 (MIDR2)

Table 102. MIDR2 field descriptions

Field	Description
SF	Manufacturer 0: Reserved 1: ST
FLASH_SIZE_1[3:0]	Coarse granularity for Flash memory size Needs to be combined with FLASH_SIZE_2 to calculate the actual memory size. 0100: 256 KB 0101: 512 KB Other values are reserved.
FLASH_SIZE_2[3:0]	Fine granularity for Flash memory size Needs to be combined with FLASH_SIZE_1 to calculate the actual memory size. 0000: 0 x (FLASH_SIZE_1 / 8) 0010: 2 x (FLASH_SIZE_1 / 8) 0100: 4 x (FLASH_SIZE_1 / 8) Other values are reserved.
PARTNUM[23:16]	ASCII character in MCU Part Number 0x50: P family (Steering)
EE	Data Flash present 0: No Data Flash present 1: Data Flash present
FF	Full-feature configuration 0: Airbag version 1: Full-featured version

11.5.2.3 Interrupt Status Flag Register (ISR)

This register holds the interrupt flags.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EIF[31:16]															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF[15:0]															
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 110. Interrupt Status Flag Register (ISR)

Table 103. ISR field descriptions

Field	Description
EIF*n*	External Interrupt Status Flag *n* This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (IRER*n*), EIF*n* causes an interrupt request. 0: No interrupt event has occurred on the pad. 1: An interrupt event as defined by IREER*n* and IFEER*n* has occurred.

11.5.2.4 Interrupt Request Enable Register (IRER)

This register enables the interrupt messaging to the interrupt controller.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IRE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IRE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 111. Interrupt Request Enable Register (IRER)

Table 104. IRER field descriptions

Field	Description
IRE*n*	External Interrupt Request Enable *n* 0: Interrupt requests from the corresponding EIF*n* bit are disabled. 1: A set EIF*n* bit causes an interrupt request.

11.5.2.5 Interrupt Rising-Edge Event Enable Register (IREER)

This register allows rising-edge triggered events to be enabled on the corresponding interrupt pads.

Address: Base + 0x0028 Access: User read/write

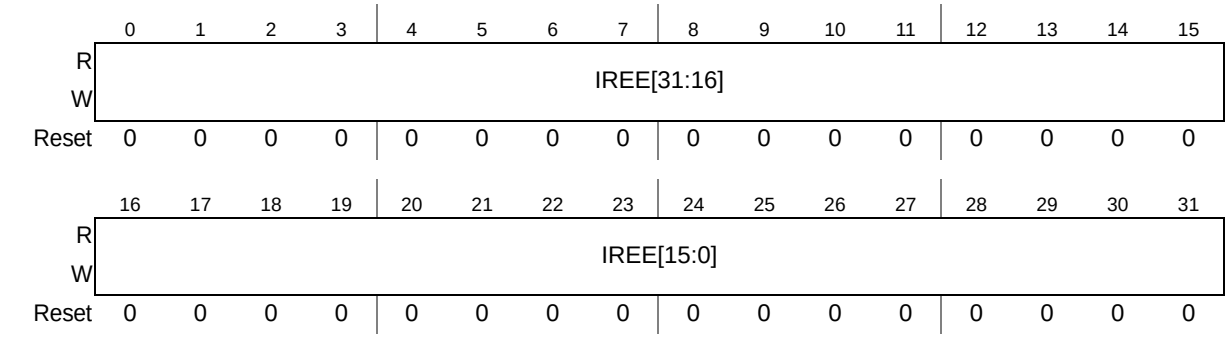


Figure 112. Interrupt Rising-Edge Event Enable Register (IREER)

Table 105. IREER field descriptions

Field	Description
IREE*n*	Enable rising-edge events to cause the EIF*n* bit to be set. 0: Rising-edge event disabled 1: Rising-edge event enabled

11.5.2.6 Interrupt Falling-Edge Event Enable Register (IFEER)

This register allows falling-edge triggered events to be enabled on the corresponding interrupt pads.

Address: Base + 0x002C Access: User read/write

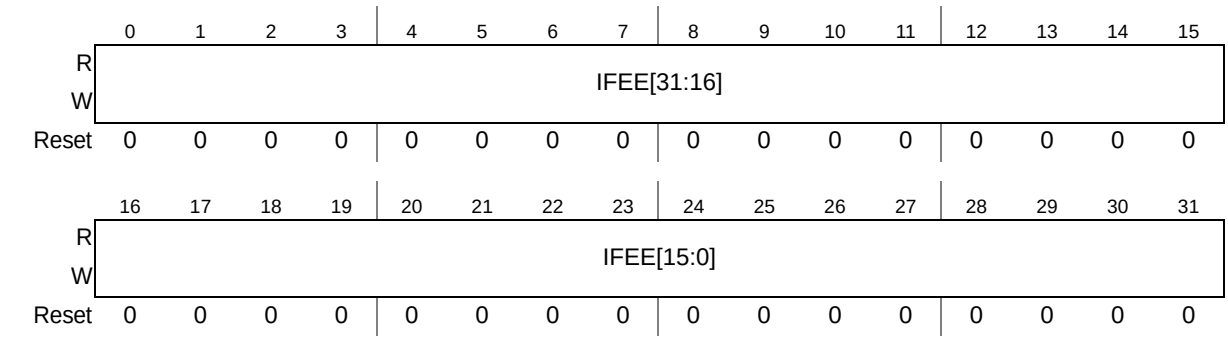


Figure 113. Interrupt Falling-Edge Event Enable Register (IFEER)

Table 106. IFEER field descriptions

Field	Description
IFEE*n*	Enable falling-edge events to cause the EIF*n* bit to be set. 0: Falling-edge event disabled 1: Falling-edge event enabled

Note: If both the IREER.IREE and IFEER.IFEE bits are cleared for the same interrupt source, the interrupt status flag for the corresponding external interrupt will never be set.

11.5.2.7 Interrupt Filter Enable Register (IFER)

This register enables a digital filter counter on the corresponding interrupt pads to filter out glitches on the inputs.

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IFE[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IFE[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 114. Interrupt Filter Enable Register (IFER)

Table 107. IFER field descriptions

Field	Description
IFE*n*	Enable digital glitch filter on the interrupt pad input. 0: Filter disabled 1: Filter enabled

11.5.2.8 Pad Configuration Registers (PCR[0:107])

The Pad Configuration Registers allow configuration of the static electrical and functional characteristics associated with I/O pads. Each PCR controls the characteristics of a single pad.

Address: Base + 0x0040 (PCR0) Access: User read/write

...

Base + 0x0116 (PCR107) 108 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SMC	APC	0	PA[1:0]		OBE	IBE	0	0	ODE	0	0	SRC	WPE	WPS
W																
Reset(1)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 115. Pad Configuration Registers 0–107 (PCR[0:107])

1. See *Table 109*.

Note: 16/32-bit access is supported for the PCR[0:107] registers.

Table 108. PCR[0:107] field descriptions

Field	Description
SMC	Safe Mode Control This bit supports the overriding of the automatic deactivation of the output buffer of the associated pad upon entering Safe mode of the device. 0: In Safe mode, output buffer of the pad disabled 1: In Safe mode, output buffer remains functional
APC	Analog Pad Control This bit enables the usage of the pad as analog input. 0: Analog input path from the pad is gated and cannot be used. 1: Analog input path switch can be enabled by the ADC.
PA[1:0]	Pad Output Assignment This field selects the function that is allowed to drive the output of a multiplexed pad. The PA field size can vary from 0 to 2 bits, depending on the number of output functions associated with this pad. 00: Alternative mode 0: GPIO 01: Alternative mode 1 (see *Chapter 3: Signal Description*) 10: Alternative mode 2 (see *Chapter 3: Signal Description*) 11: Alternative mode 3 (see *Chapter 3: Signal Description*) **Note:** The number of bits in the PA bitfield depends of the number of actual alternate functions provided for each pad. Please see the *SPC560P44Lx, SPC560P50Lx Datasheet (SPC560P44Lx, SPC560P50Lx)*.
OBE	Output Buffer Enable This bit enables the output buffer of the pad in case the pad is in GPIO mode. 0: Output buffer of the pad disabled when PA = 00 1: Output buffer of the pad enabled when PA = 00
IBE	Input Buffer Enable This bit enables the input buffer of the pad. 0: Input buffer of the pad disabled 1: Input buffer of the pad enabled
ODE	Open Drain Output Enable This bit controls output driver configuration for the pads connected to this signal. Either open drain or push/pull driver configurations can be selected. This feature applies to output pads only. 0: Open drain enable signal negated for the pad 1: Open drain enable signal asserted for the pad
SRC	Slew Rate Control 0: Slowest configuration 1: Fastest configuration
WPE	Weak Pull Up/Down Enable This bit controls whether the weak pull up/down devices are enabled/disabled for the pad connected to this signal. 0: Weak pull device enable signal negated for the pad 1: Weak pull device enable signal asserted for the pad
WPS	Weak Pull Up/Down Select This bit controls whether weak pull up or weak pull down devices are used for the pads connected to this signal when weak pull up/down devices are enabled. 0: Pull down enabled 1: Pull up enabled

Table 109. PCR[*n*] reset value exceptions

Field	Description
PCR[2] PCR[3] PCR[4]	These registers correspond to the ABS[0], ABS[1], and FAB boot pins, respectively. Their default state is input, pull enabled. Their reset value is 0x0102.
PCR[20]	This register corresponds to the TDO pin. Its default state is ALT1, slew rate = 1. Its reset value is 0x0604.
PCR[21]	This register corresponds to the TDI pin. Its default state is input, pull enabled, pull selected, slew enabled. So its reset value is 0x0107.
PCR[*n*]	For other PCR[*n*] registers, the reset value is 0x0000.

In addition to the bit map above, the following *Table 110: PCR bit implementation by pad type* describes the PCR depending on the pad type (refer to *Section 3.3.3: Pin muxing* for pad types description). The bits in shaded fields are not implemented for the particular I/O type. The PA field selecting the number of alternate functions may or may not be present depending on the number of alternate functions actually mapped on the pad.

Table 110. PCR bit implementation by pad type

Pad type	PCR bit No.															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S, M, F (Pad with GPIO and digital alternate functionality)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS
I (Pad with GPIO and analog functionality)		SMC	APC		PA [1:0]		OBE	IBE			ODE			SRC	WPE	WPS

11.5.2.9 Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])

The purpose of the PSMI[0_3:32_35] registers is to allow connecting a single input pad to one of several peripheral inputs. Thus, it is possible to define different pads to be possible inputs for a certain peripheral function.

Address: Base + 0x0500 (PSMI0_3)
Base + 0x0504 (PSMI4_7)
Base + 0x0508 (PSMI8_11)
Base + 0x050C (PSMI12_15)
Base + 0x0510 (PSMI16_19)
Base + 0x0514 (PSMI20_23)
Base + 0x0518 (PSMI024_27)
Base + 0x051C (PSMI28_31)
Base + 0x0520 (PSMI32_35)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PADSEL0[3:0]				0	0	0	0	PADSEL1[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	PADSEL2[3:0]				0	0	0	0	PADSEL3[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 116. Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])

Table 111. PSMI[0_3:32_35] field descriptions

Field	Description
PADSEL0–3 ... PADSEL32–35	Pad Selection Bits Each PADSEL field selects the pad currently used for a certain input function. See *Table 112: Pad selection* .

Table 112. Pad selection

Register	PADSEL	Module	Port	PADSEL field(1)	Port name	LQFP pin	
						100-pin	144-pin
PSMI0_3	PADSEL0	ctu0	EXT_IN	0b	C[13]	71	101
				1b	C[15]	85	124
	PADSEL1	dspi2	SCK	0b	A[0]	51	73
				1b	A[11]	82	120
	PADSEL2	dspi2	SIN	0b	A[2]	57	84
				1b	A[13]	95	136
	PADSEL3	dspi2	CS0	0b	A[3]	64	92
				1b	A[10]	81	118

Table 112. Pad selection(Continued)

Register	PADSEL	Module	Port	PADSEL field[1]	Port name	LQFP pin	
						100-pin	144-pin
PSMI4_7	PADSEL0	dspi3	SCK	00b	D[6]	23	34
				01b	D[11]	54	78
				10b	E[13]	—	117
	PADSEL1	dspi3	SIN	00b	D[7]	26	37
				01b	D[14]	73	105
				10b	E[15]	—	121
	PADSEL2	dspi3	CS0	00b	C[11]	55	80
				01b	D[10]	53	76
				10b	F[3]	—	139
	PADSEL3	eTimer0	ETC[4]	00b	A[4]	75	108
				01b	C[11]	55	80
				10b	B[14]	44	64
PSMI8_11	PADSEL0	eTimer0	ETC[5]	0b	C[12]	56	82
				1b	B[8]	31	47
	PADSEL1	eTimer1	ETC[0]	0b	A[4]	75	108
				1b	C[15]	85	124
	PADSEL2	eTimer1	ETC[1]	0b	C[13]	71	101
				1b	D[0]	86	125
	PADSEL3	eTimer1	ETC[2]	00b	B[0]	76	109
				01b	C[14]	72	103
				10b	D[1]	3	3
PSMI12_15	PADSEL0	eTimer1	ETC[3]	00b	B[1]	77	110
				01b	D[2]	97	140
				10b	F[12]	—	106
	PADSEL1	eTimer1	ETC[4]	00b	A[14]	99	143
				01b	C[3]	10	16
				10b	D[3]	89	128
				11b	F[13]	—	112
	PADSEL2	eTimer1	ETC[5]	00b	A[5]	8	14
				01b	A[15]	100	144
				10b	D[4]	90	129
	PADSEL3	flexpwm0	EXT_SYNC	0b	C[13]	71	101
				1b	C[15]	85	124

Table 112. Pad selection(Continued)

Register	PADSEL	Module	Port	PADSEL field[1]	Port name	LQFP pin	
						100-pin	144-pin
PSMI16_19	PADSEL0	flexpwm0	FAULT0	00b	A[9]	94	134
				01b	A[13]	95	136
				10b	G[8]	—	81
	PADSEL1	flexpwm0	FAULT1	00b	C[10]	78	111
				01b	D[6]	23	34
				10b	G[9]	—	79
	PADSEL2	flexpwm0	FAULT2	00b	C[8]	91	130
				01b	C[9]	84	123
				10b	G[10]	—	77
	PADSEL3	flexpwm0	FAULT3	00b	C[5]	7	13
				01b	D[8]	21	32
				10b	G[11]	—	75
PSMI20_23	PADSEL0	—	—	—	—	—	—
				—	—	—	—
	PADSEL1	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
	PADSEL2	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
	PADSEL3	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
				—	—	—	—

Table 112. Pad selection(Continued)

Register	PADSEL	Module	Port	PADSEL field[1]	Port name	LQFP pin	
						100-pin	144-pin
PSMI24_27	PADSEL0	—	—	—	—	—	—
				—	—	—	—
	PADSEL1	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
	PADSEL2	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
	PADSEL3	—	—	—	—	—	—
				—	—	—	—
				—	—	—	—
				—	—	—	—
PSMI28_31	PADSEL0	flexpwm0	PWMX_1	0b	C[4]	5	11
				1b	D[12]	70	99
	PADSEL1	flexpwm0	PWMX_2	0b	A[10]	81	118
				1b	G[2]	—	102
	PADSEL2	flexpwm0	PWMX_3	00b	C[9]	84	123
				01b	D[2]	97	140
				10b	G[5]		85
	PADSEL3	LINflex0	RXD Receive Data Input Line	0b	B[3]	80	116
				1b	B[7]	29	43
PSMI32_35	PADSEL0	LINflex1	RXD Receive Data Input Line	00b	B[13]	42	60
				01b	D[12]	70	99
				10b	F[15]	—	113

1. Values not listed are reserved.

11.5.2.10 GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])

These registers can be used to set or clear a single GPIO pad with a byte access.

Address: Base + 0x0600 (GPDO0_3) Access: User read/write

...

Base + 0x0668 (GPDO104_107) 27 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDO[0]	0	0	0	0	0	0	0	PDO[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDO[2]	0	0	0	0	0	0	0	PDO[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 117. Port GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])

Table 113. GPDO[0_3:104_107] field descriptions

Field	Description
PDO[*n*]	Pad Data Out This bit stores the data to be driven out on the external GPIO pad controlled by this register. 0: Logic low value is driven on the corresponding GPIO pad when the pad is configured as an output. 1: Logic high value is driven on the corresponding GPIO pad when the pad is configured as an output.

11.5.2.11 GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])

These registers can be used to read the GPIO pad data with a byte access.

Address: Base + 0x0800 (GPDI0_3) Access: User read-only

...

Base + 0x0868 (GPDI104_107) 27 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	PDI[0]	0	0	0	0	0	0	0	PDI[1]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	PDI[2]	0	0	0	0	0	0	0	PDI[3]
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 118. GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])

Table 114. GPDI[0_3:104_107] field descriptions

Field	Description
PDI[*x*]	Pad Data In This bit stores the value of the external GPIO pad associated with this register. 0: The value of the data in signal for the corresponding GPIO pad is logic low. 1: The value of the data in signal for the corresponding GPIO pad is logic high.

11.5.2.12 Parallel GPIO Pad Data Out register 0–3 (PGPDO[0:3])

These registers set or clear the respective pads of the device.

Address: Base + 0x0C00 (PGPDO0) Base + 0x0C05 (PGPDO2) Access: User read/write
Base + 0x0C04 (PGPDO1) Base + 0x0C0C (PGPDO3)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PPDO[*x*][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PPDO[*x* + 1][15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 119. Parallel GPIO Pad Data Out register 0–3(PGPDO[0:3])

Table 115. PGPDO0_3 field descriptions

Field	Description
PPDO[*x*]	Parallel Pad Data Out Write or read the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise *Section 11.5.2.10: GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])*. The *x* and bit index define which PPDO register bit is equivalent to which PDO register bit according to the following equation: PPDO[*x*][*y*] = PDO[(*x* * 16) + *y*]

Note: The PGPDO registers access the same physical resource as the PDO and MPGPDO address locations. Some examples of the mapping:

PPDO[0][0] = PDO[0]

PPDO[2][0] = PDO[32]

11.5.2.13 Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

These registers hold the synchronized input value from the pads.

Address: Base + 0x0C40 (PGPDI0) Base + 0x0C45 (PGPDI2) Access: User read-only
Base + 0x0C44 (PGPDI1) Base + 0x0C4C (PGPDI3)

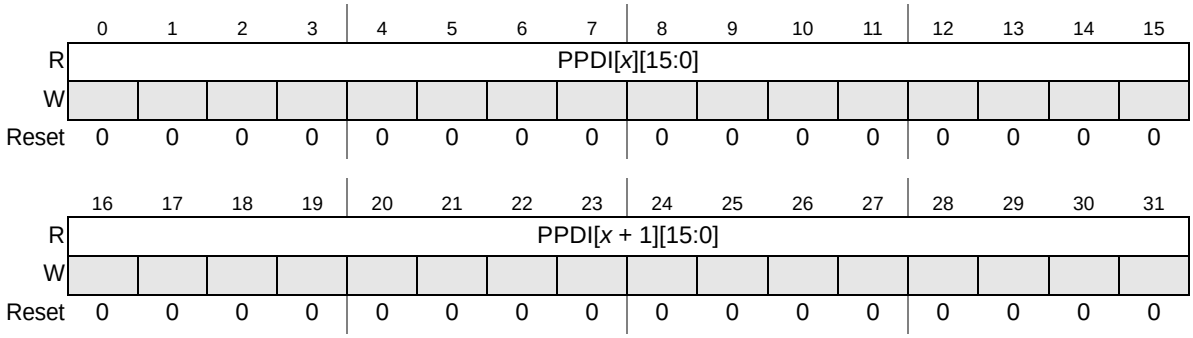


Figure 120. Parallel GPIO Pad Data In register 0–3 (PGPDI[0:3])

Table 116. PGPDI[0:3] field descriptions

Field	Description
PPDI[*x*]	Parallel Pad Data In Read the current pad value. Accesses to this register location are coherent with accesses to the bit-wise *Section 11.5.2.11: GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])*. The *x* and bit index define which PPDI register bit is equivalent to which PDI register bit according to the following equation: PPDI[*x*][*y*] = PDI[(*x* * 16) + *y*]

11.5.2.14 Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

This register can be used to selectively modify the pad values associated to PPDO[*x*][15:0]. The MPGPDO[*x*] register may only be accessed with 32-bit writes. 8-bit or 16-bit writes will not modify any bits in the register and cause a transfer error response by the module. Read accesses will return 0.

Address: Base + 0x0C80 (MPGPDO0) Base + 0x0C80 (MPGPDO4) Access: User write-only
Base + 0x0C84 (MPGPDO1) Base + 0x0C84 (MPGPDO5)
Base + 0x0C88 (MPGPDO2) Base + 0x0C98 (MPGPDO6)
Base + 0x0C8C (MPGPDO3)

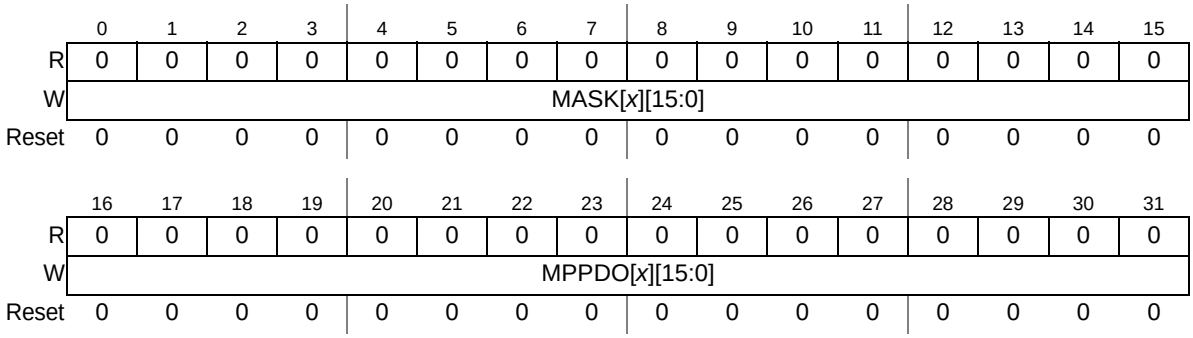


Figure 121. Masked Parallel GPIO Pad Data Out register 0–6 (MPGPDO[0:6])

Table 117. MPGPDO[0:6] field descriptions

Field	Description
MASK[*x*] [15:0]	Mask Field Each bit corresponds to one data bit in the MPPDO[x] field at the same bit location. 0: The associated bit value in the MPPDO[*x*] field is ignored. 1: The associated bit value in the MPPDO[*x*] field is written.
MPPDO[*x*] [15:0]	Masked Parallel Pad Data Out Write the data register that stores the value to be driven on the pad in output mode. Accesses to this register location are coherent with accesses to the bit-wise *Section 11.5.2.10: GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])*. The x and bit index define which MPPDO register bit is equivalent to which PDO register bit according to the following equation: MPPDO[*x*][*y*] = PDO[(*x* * 16) + *y*]

11.5.2.15 Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])

These registers configure the filter counter associated with each digital glitch filter.

Address: Base + 0x1000 (IFMC0) Access: User read/write

...

Base + 0x107C (IFMC31) 32 registers

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MAXCNT*x*[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 122. Interrupt Filter Maximum Counter registers 0–31 (IFMC[0:31])

Table 118. IFMC[0:31] field descriptions

Field	Description
MAXCNT*x* [3:0]	Maximum Interrupt Filter Counter setting. Filter Period = (T_{CK})*3 (for 2 < MAXCNT < 6) Filter Period = (T_{CK}) × MAXCNT*x)* (for MAXCNT = 6,7,.... 15) For MAXCNT = 0, 1, 2, the filter behaves as ALL PASS filter. MAXCNT*x* can be 0 to 15; T_{CK}: Prescaled Filter Clock Period, which is IRC clock prescaled to IFCP value; T_{IRC}: Basic Filter Clock Period: 62.5 ns (f_{IRC} = 16 MHz)

11.5.2.16 Interrupt Filter Clock Prescaler Register (IFCPR)

This register configures a clock prescaler that selects the clock for all digital filter counters in the SIUL.

Address: Base + 0x1080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	IFCP[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 123. Interrupt Filter Clock Prescaler Register (IFCPR)

Table 119. IFCPR field descriptions

Field	Description
IFCP [3:0]	Interrupt Filter Clock Prescaler setting Prescaled Filter Clock Period = T_{IRC} × (IFCP + 1) T_{IRC} is the internal oscillator period. IFCP can be 0 to 15.

11.6 Functional description

11.6.1 General

This section provides a functional description of the System Integration Unit Lite.

11.6.2 Pad control

The SIUL controls the configuration and electrical characteristic of the device pads. It provides a consistent interface for all pads, both on a by-port and a by-bit basis. The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output). The pad configuration registers (PCR*n*, see *Section 11.5.2.8: Pad Configuration Registers (PCR[0:107])*) allow software control of the static electrical characteristics of external pins with a single write. These configure the following pad features:

- Open drain output enable
- Slew rate control
- Pull control
- Pad assignment
- Control of analog path switches
- Safe mode behavior configuration

11.6.3 General purpose input and output pads (GPIO)

The SIUL allows each pad to be configured as either a General Purpose Input Output pad (GPIO), and as one or more alternate functions (input or output), the function of which is normally determined by the peripheral that uses the pad.

The SIUL manages up to 106 GPIO pads organized as ports that can be accessed for data reads and writes as 32-bit, 16-bit or 8-bit.

As shown in *Figure 124*, all port accesses are identical with each read or write being performed only at a different location to access a different port width.

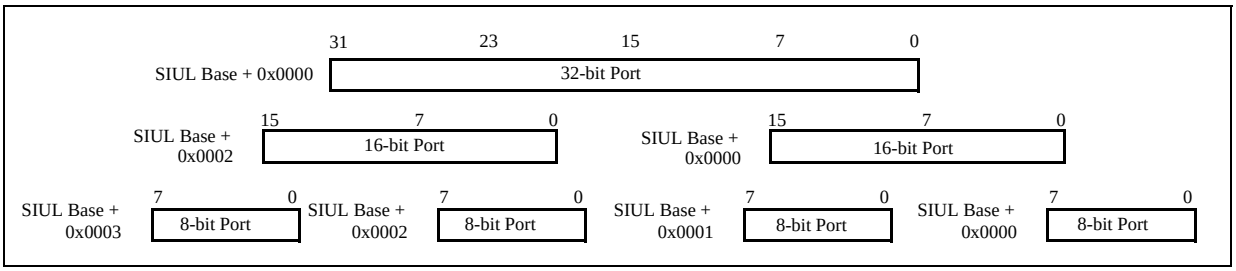


Figure 124. Data port example arrangement showing configuration for different port width accesses

This implementation requires that the registers are arranged in such a way as to support this range of port widths without having to split reads or writes into multiple accesses.

The SIUL has separate data input (GPDI*n_n*, see *Section 11.5.2.11: GPIO Pad Data Input registers 0_3–104_107 (GPDI[0_3:104_107])*) and data output (GPDO*n_n*, see *Section 11.5.2.10: GPIO Pad Data Output registers 0_3–104_107 (GPDO[0_3:104_107])*) registers for all pads, allowing the possibility of reading back an input or output value of a pad directly. This supports the ability to validate what is present on the pad rather than merely confirming the value that was written to the data register by accessing the data input registers.

The data output registers support both read and write operations to be performed.

The data input registers support read access only.

When the pad is configured to use one of its alternate functions, the data input value reflect the respective value of the pad. If a write operation is performed to the data output register for a pad configured as an alternate function (non GPIO), this write will not be reflected by the pad value until reconfigured to GPIO.

The allocation of what input function is connected to the pin is defined by the PSMI registers (see *Section 11.5.2.9: Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35])*).

11.6.4 External interrupts

The SIUL supports 32 external interrupts, EIRQ[0:31]. The signal description chapter of this reference manual provides a map of the external interrupts.

The SIUL supports four interrupt vectors to the interrupt controller. Each vector interrupt has eight external interrupts combined together with the presence of flag generating an interrupt for that vector if enabled. All of the external interrupt pads within a single group have equal priority.

Refer to *Figure 125* for an overview of the external interrupt implementation.

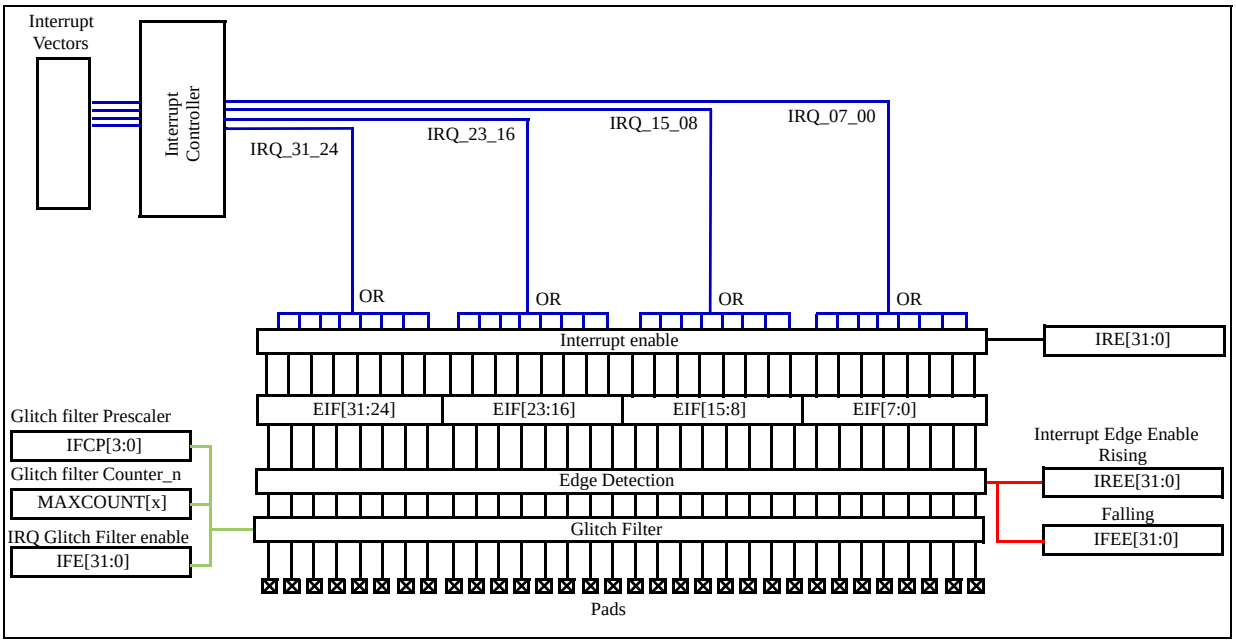


Figure 125. External interrupt pad diagram

11.6.4.1 External interrupt management

Each interrupt can be enabled or disabled independently. This can be performed using the IRER (see *Section 11.5.2.4: Interrupt Request Enable Register (IRER)*). A pad defined as an external interrupt can be configured to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled is reserved and should not be configured.

The active EIRQ edge is controlled through the configuration of the registers IREER and IFEER.

Each external interrupt supports an individual flag that is held in the ISR (see *Section 11.5.2.3: Interrupt Status Flag Register (ISR)*). This register is a write-1-to-clear register type, preventing inadvertent overwriting of other flags in the same register.

11.7 Pin muxing

For pin muxing, please refer to *Chapter 3: Signal Description* of this reference manual.

12 e200z0 and e200z0h Core

12.1 Overview

The SPC560P44Lx, SPC560P50Lx microcontroller implements the e200z0h core.

The e200 processor family is a set of CPU cores built on the Power Architecture technology. e200 processors are designed for deeply embedded control applications that require low cost solutions rather than maximum performance.

The e200z0 and e200z0h processors integrate an integer execution unit, branch control unit, instruction fetch and load/store units, and a multi-ported register file capable of sustaining three read and two write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in some cases.

The e200z0 core is a single-issue, 32-bit Power Architecture technology VLE-only design with 32-bit general purpose registers (GPRs). Implementing only the VLE (variable-length encoding) APU provides improved code density. All arithmetic instructions that execute in the core operate on data in the GPRs.

12.2 Features

The following is a list of some of the key features of the e200z0 and e200z0h cores:

- 32-bit Power Architecture technology VLE-only programmer's model
- Single issue, 32-bit CPU
- Implements the VLE APU for reduced code footprint
- In-order execution and retirement
- Precise exception handling
- Branch processing unit
  - Dedicated branch address calculation adder
  - Branch acceleration using Branch Target Buffer (e200z0h only)
- Supports independent instruction and data accesses to different memory subsystems, such as SRAM and Flash memory via independent Instruction and Data bus interface units (BIUs) (e200z0h only)
- Supports instruction and data access via a unified 32-bit Instruction/Data BIU (e200z0 only)
- Load/store unit
  - 1 cycle load latency
  - Fully pipelined
  - Big-endian support only
  - Misaligned access support
  - Zero load-to-use pipeline bubbles for aligned transfers
- Power management
  - Low power design
  - Dynamic power management of execution units

12.2.1 Microarchitecture summary

The e200z0 processor utilizes a four stage pipeline for instruction execution. The Instruction Fetch (stage 1), Instruction Decode/Register file Read/Effective Address Calculation (stage 2), Execute/Memory Access (stage 3), and Register Writeback (stage 4) stages operate in an overlapped fashion, allowing single clock instruction execution for most instructions.

The integer execution unit consists of a 32-bit Arithmetic Unit (AU), a Logic Unit (LU), a 32-bit Barrel shifter (Shifter), a Mask-Insertion Unit (MIU), a Condition Register manipulation Unit (CRU), a Count-Leading-Zeros unit (CLZ), an 8 × 32 Hardware Multiplier array, result feed-forward hardware, and a hardware divider.

Arithmetic and logical operations are executed in a single cycle with the exception of the divide and multiply instructions. A Count-Leading-Zeros unit operates in a single clock cycle.

The Instruction Unit contains a PC incrementer and a dedicated Branch Address adder to minimize delays during change of flow operations. Sequential prefetching is performed to ensure a supply of instructions into the execution pipeline. Branch target prefetching from the BTB is performed to accelerate certain taken branches in the e200z0. Prefetched instructions are placed into an instruction buffer with 4 entries (2 entries in e200z0), each capable of holding a single 32-bit instruction or a pair of 16-bit instructions.

Conditional branches that are not taken execute in a single clock. Branches with successful target prefetching have an effective execution time of one clock on e200z0h. All other taken branches have an execution time of two clocks.

Memory load and store operations are provided for byte, halfword, and word (32-bit) data with automatic zero or sign extension of byte and halfword load data as well as optional byte reversal of data. These instructions can be pipelined to allow effective single cycle throughput. Load and store multiple word instructions allow low overhead context save and restore operations. The load/store unit contains a dedicated effective address adder to allow effective address generation to be optimized. Also, a load-to-use dependency does not incur any pipeline bubbles for most cases.

The Condition Register unit supports the condition register (CR) and condition register operations defined by the Power Architecture architecture. The condition register consists of eight 4-bit fields that reflect the results of certain operations, such as move, integer and floating-point compare, arithmetic, and logical instructions, and provide a mechanism for testing and branching.

Vectored and autovectored interrupts are supported by the CPU. Vectored interrupt support is provided to allow multiple interrupt sources to have unique interrupt handlers invoked with no software overhead.

12.2.1.1 Block diagram

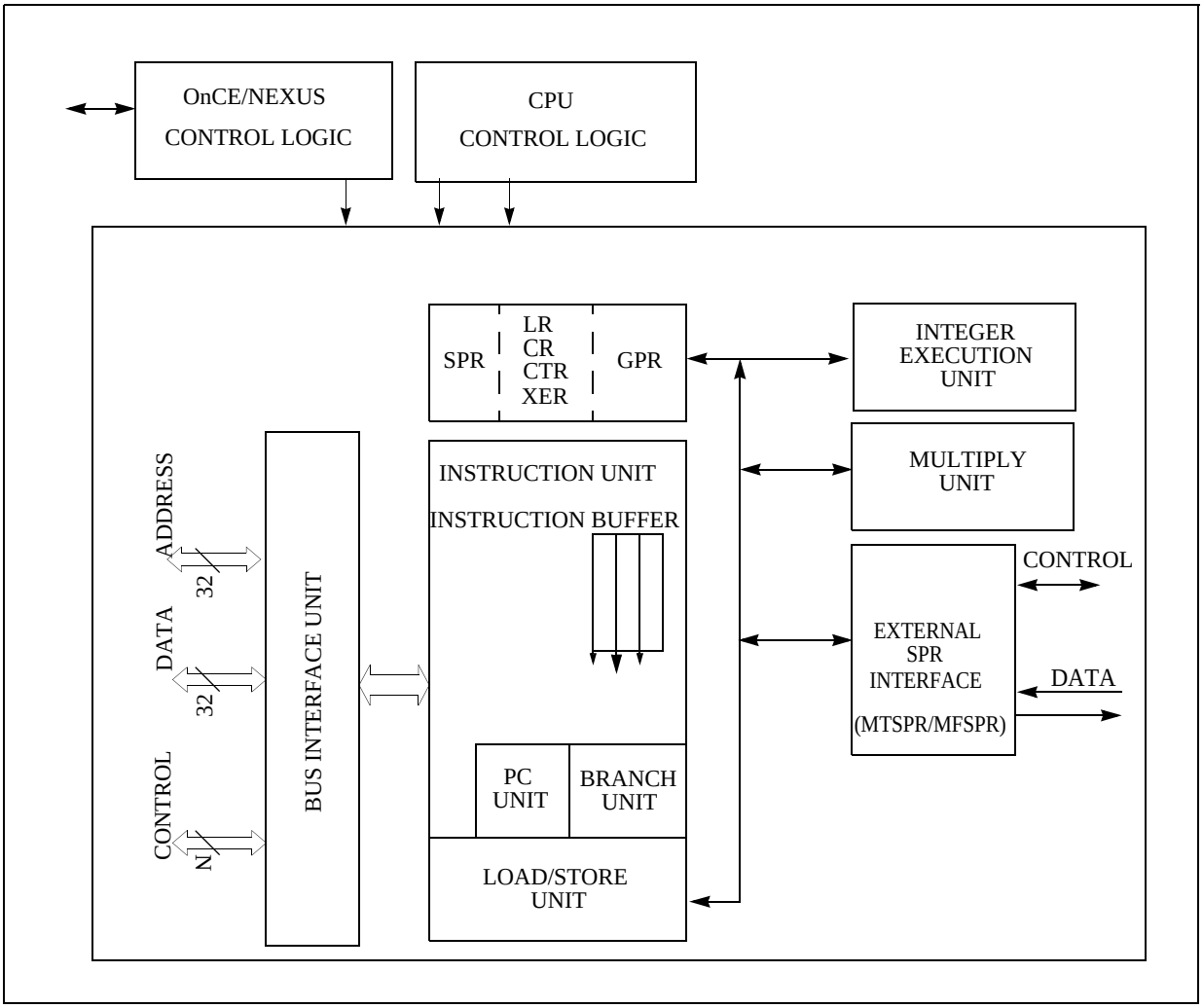


Figure 126. e200z0 block diagram

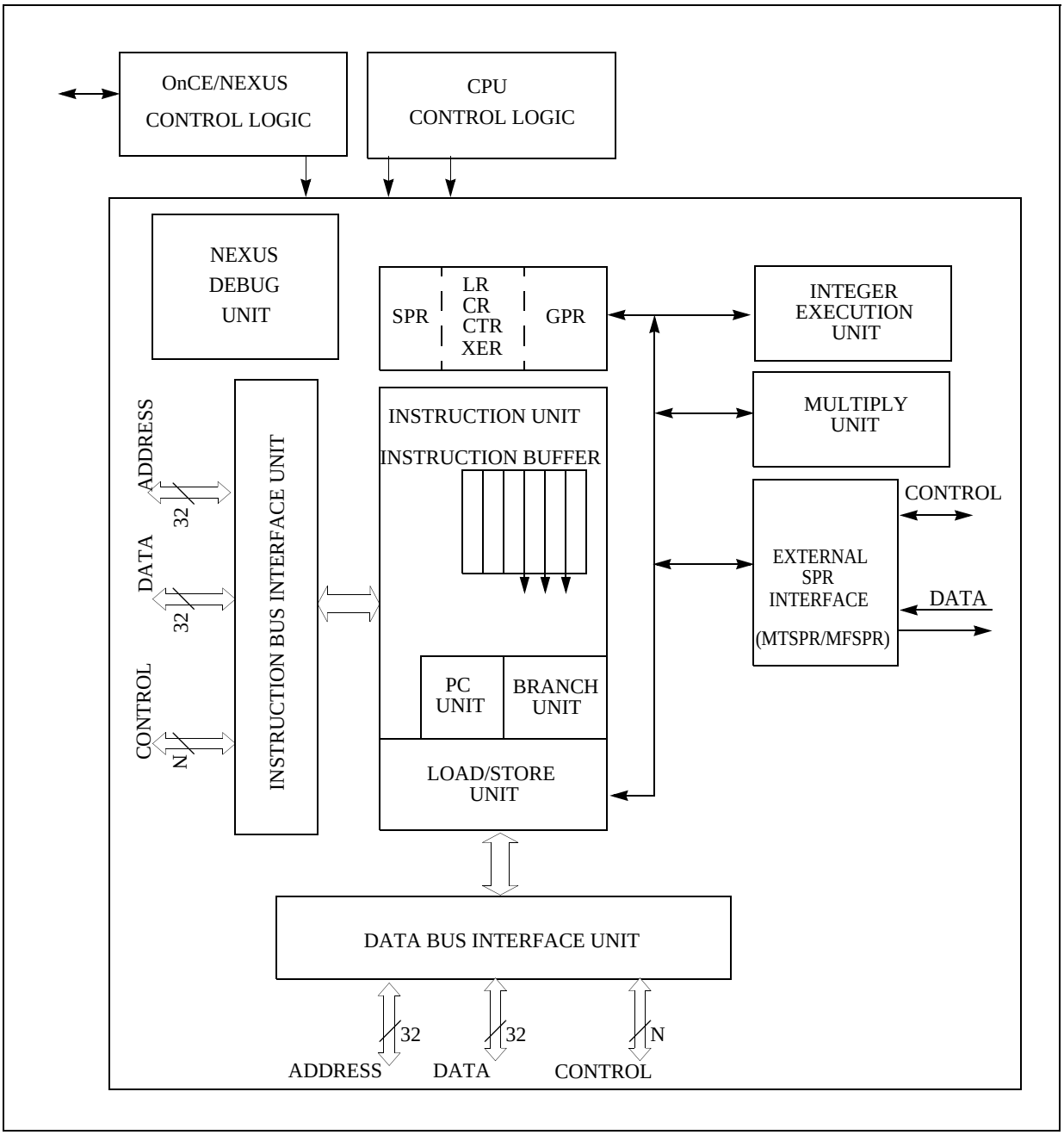


Figure 127. e200z0h block diagram

12.2.1.2 Instruction unit features

The features of the e200 Instruction unit are:

- 32-bit instruction fetch path supports fetching of one 32-bit instruction per clock, or as many as two 16-bit VLE instructions per clock
- Instruction buffer with 4 entries in e200z0h, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Instruction buffer with 2 entries in e200z0, each holding a single 32-bit instruction, or a pair of 16-bit instructions
- Dedicated PC incrementer supporting instruction prefetches
- Branch unit with dedicated branch address adder supporting single cycle of execution of certain branches, two cycles for all others

12.2.1.3 Integer unit features

The e200 integer unit supports single cycle execution of most integer instructions:

- 32-bit AU for arithmetic and comparison operations
- 32-bit LU for logical operations
- 32-bit priority encoder for count leading zero's function
- 32-bit single cycle barrel shifter for shifts and rotates
- 32-bit mask unit for data masking and insertion
- Divider logic for signed and unsigned divide in 5 to 34 clocks with minimized execution timing
- 8 × 32 hardware multiplier array supports 1 to 4 cycle 32 × 32 $\rightarrow$ 32 multiply (early out)

12.2.1.4 Load/Store unit features

The e200 load/store unit supports load, store, and the load multiple / store multiple instructions:

- 32-bit effective address adder for data memory address calculations
- Pipelined operation supports throughput of one load or store operation per cycle
- 32-bit interface to memory (dedicated memory interface on e200z0h)

12.2.1.5 e200z0h system bus features

The features of the e200z0h System Bus interface are as follows:

- Independent Instruction and Data Buses
- AMBA AHB Lite Rev 2.0 Specification with support for ARM v6 AMBA Extensions
  - Exclusive Access Monitor
  - Byte Lane Strobes
  - Cache Allocate Support
- 32-bit address bus plus attributes and control on each bus
- 32-bit read data bus for Instruction Interface
- Separate uni-directional 32-bit read data bus and 32-bit write data bus for Data Interface
- Overlapped, in-order accesses

12.2.1.6 Nexus 2+ features

The Nexus 2+ module is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program Trace via Branch Trace Messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code may be traced.
- Ownership Trace via Ownership Trace Messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An Ownership Trace Message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to the processor memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint Messaging via the auxiliary interface
- Watchpoint Trigger enable of Program Trace Messaging
- Auxiliary interface for higher data input/output
  - Configurable (min/max) Message Data Out pins (**nex_mdo[n:0]**)
  - One (1) or two (2) Message Start/End Out pins (**nex_mseo_b[1:0]**)
  - One (1) Read/Write Ready pin (**nex_rdy_b**) pin
  - One (1) Watchpoint Event pin (**nex_evto_b**)
  - One (1) Event In pin (**nex_evti_b**)
  - One (1) MCKO (Message Clock Out) pin
- Registers for Program Trace, Ownership Trace and Watchpoint Trigger control
- All features controllable and configurable via the JTAG port

12.3 Core registers and programmer's model

This section describes the registers implemented in the e200z0 and e200z0h cores. It includes an overview of registers defined by the Power Architecture technology, highlighting differences in how these registers are implemented in the e200 core, and provides a detailed description of e200-specific registers. Full descriptions of the architecture-defined register set are provided in Power Architecture Specification.

The Power Architecture defines register-to-register operations for all computational instructions. Source data for these instructions are accessed from the on-chip registers or are provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions. Data is transferred between memory and registers with explicit load and store instructions only.

Figure 128 and *Figure 130* show the e200 register set, including the registers that are accessible while in supervisor mode and the registers that are accessible in user mode. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).

Note: e200z0 and e200z0h is a 32-bit implementation of the Power Architecture specification.

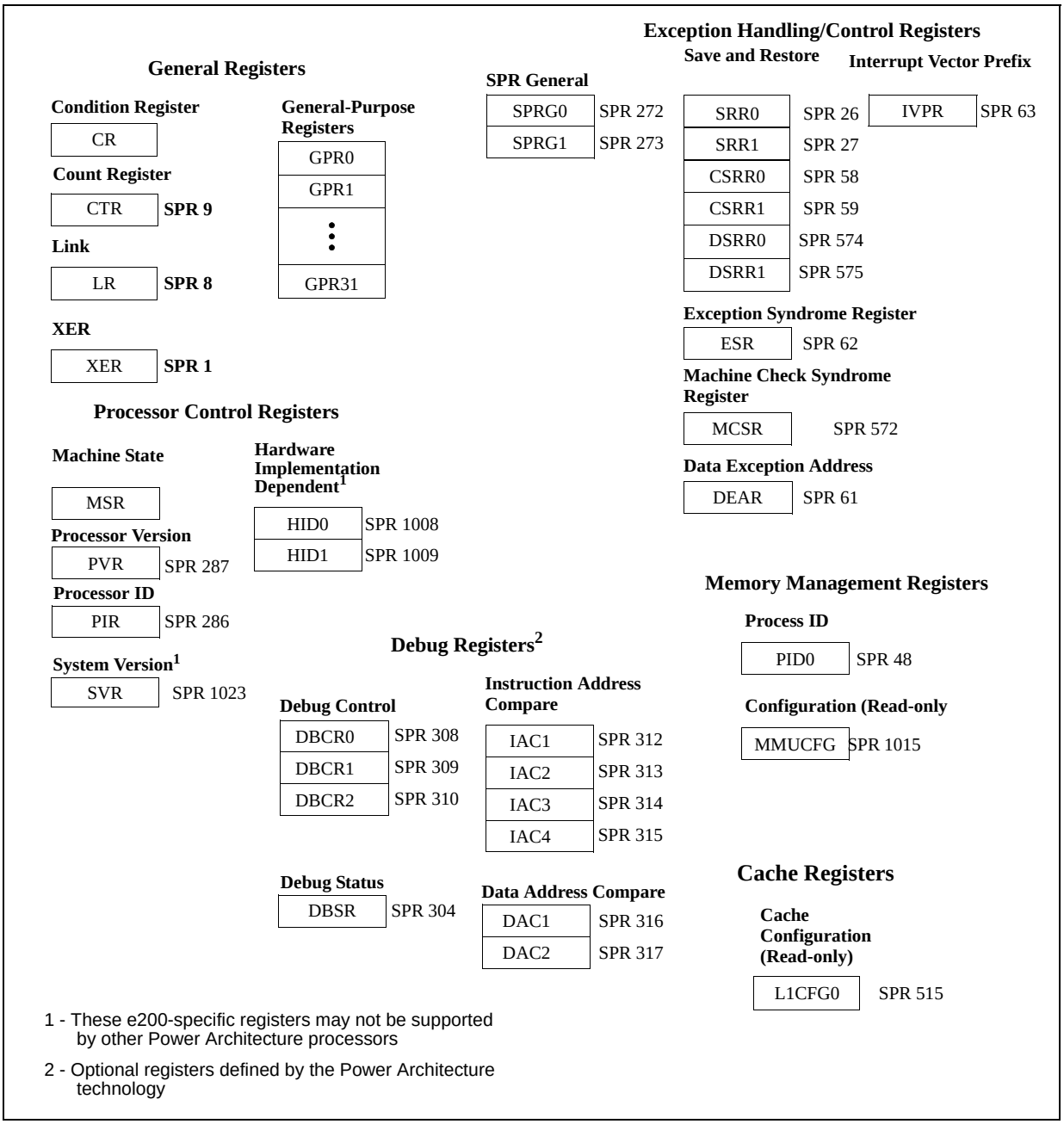


Figure 128. e200z0 Supervisor mode programmer's model

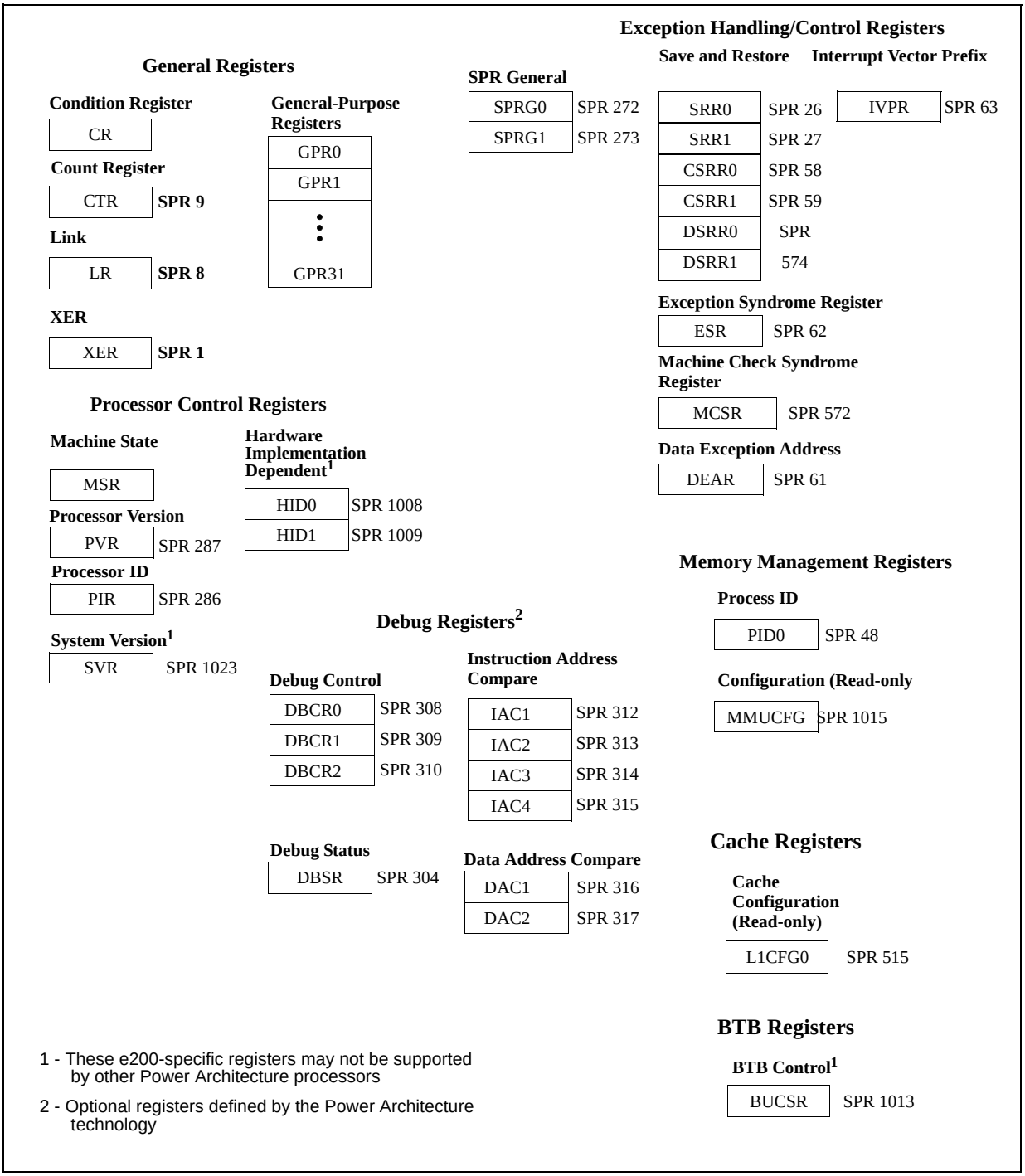


Figure 129. e200z0h Supervisor mode programmer's model

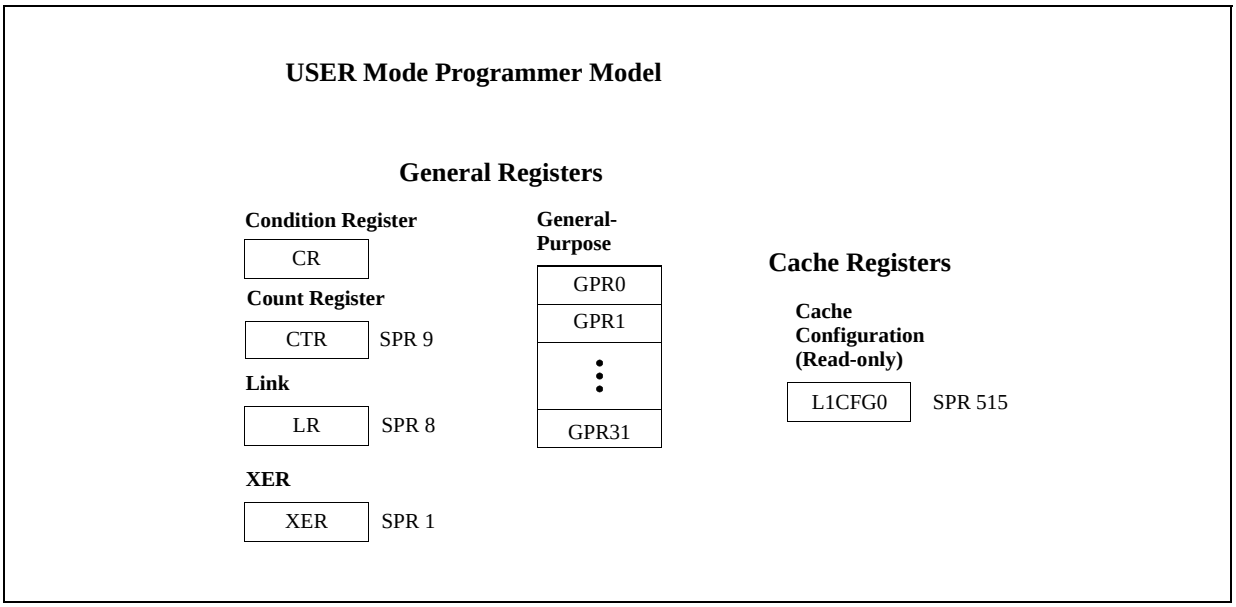


Figure 130. e200 User mode program model

12.3.1 Unimplemented SPRs and read-only SPRs

e200 fully decodes the SPR field of the **mfspr** and **mtspr** instructions. **If the SPR specified is undefined and not privileged, an illegal instruction exception** is generated**.** If the SPR specified is undefined and privileged and the CPU is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is undefined and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

For the **mtspr** instruction, if the SPR specified is read-only **and not privileged, an illegal instruction exception** is generated**.** If the SPR specified is read-only and privileged and the core is in user mode (MSR[PR=1]), a privileged instruction exception is generated. If the SPR specified is read-only and privileged and the core is in supervisor mode (MSR[PR=0]), an illegal instruction exception is generated.

12.4 Instruction summary

The e200z0 core supports Power Architecture technology VLE instructions..

13 Peripheral Bridge (PBRIDGE)

13.1 Introduction

The Peripheral Bridge (PBRIDGE) is the interface between the system bus and on-chip peripherals.

13.1.1 Block diagram

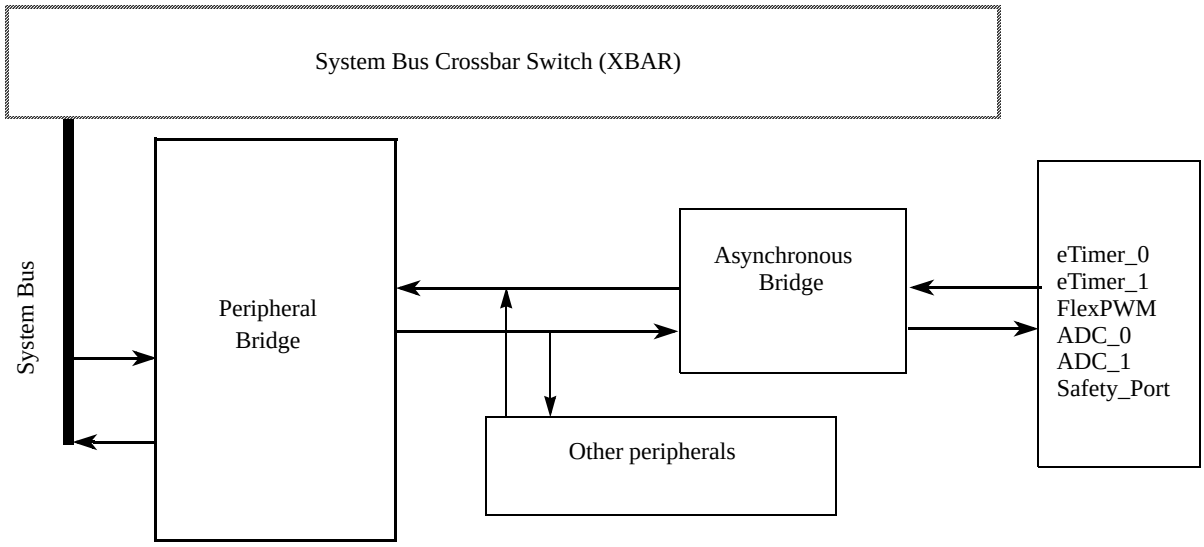


Figure 131. PBRIDGE interface

13.1.2 Overview

The PBRIDGE acts as interface between the system bus and lower bandwidth peripherals. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

As shown in *Figure 131*, the asynchronous bridge is a dedicated module that resynchronizes signals synchronous to the system clock (SYS_CLK) to the ones synchronous to the motor control clock (MC_PLL_CLK).

The PBRIDGE has the following key features:

- Supports the slave interface signals. This interface is only meant for slave peripherals.
- Supports 32-bit slave peripherals (byte, halfword, and word reads and writes are supported to each)
- Provides configurable per-master access protections

13.1.3 Modes of operation

The PBRIDGE has only one operating mode.

13.2 Functional description

The PBRIDGE serves as an interface between a system bus and the peripheral (slave) bus. It functions as a protocol translator. Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the slave bus interface.

13.2.1 Access support

Aligned 32-bit word accesses, halfword accesses, and byte accesses are supported for the peripherals. Peripheral registers must not be misaligned, although no explicit checking is performed by the PBRIDGE.

Note: Data accesses that cross a 32-bit boundary are not supported.

13.2.1.1 Peripheral Write Buffering

Buffered writes are not supported by the device PBRIDGE.

13.2.1.2 Read cycles

Two-clock read accesses are possible with the Peripheral Bridge when the requested access size is 32-bits or smaller, and is not misaligned across a 32-bit boundary.

13.2.1.3 Write cycles

Three clock write accesses are possible with the Peripheral Bridge when the requested access size is 32-bits or smaller. Misaligned writes that cross a 32-bit boundary are not supported.

13.2.2 General operation

Slave peripherals are modules that contain readable/writable control and status registers. The system bus master reads and writes these registers through the PBRIDGE. The PBRIDGE generates module enables, the module address, transfer attributes, byte enables, and write data as inputs to the slave peripherals. The PBRIDGE captures read data from the slave interface and drives it on the system bus.

The PBRIDGE occupies a 64 MB portion of the address space. The register maps of the slave peripherals are located on 16-KB boundaries. Each slave peripheral is allocated one 16-KB block of the memory map, and is activated by one of the module enables from the PBRIDGE.

The PBRIDGE is responsible for indicating to slave peripherals if an access is in supervisor or user mode. All eDMA and FlexRay transfers are done in supervisor mode.

14 Crossbar Switch (XBAR)

14.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between four master ports and three slave ports. XBAR supports a 32-bit address bus width and a 32-bit data bus width at all master and slave ports.

14.2 Block diagram

Figure 132 shows a block diagram of the crossbar switch.

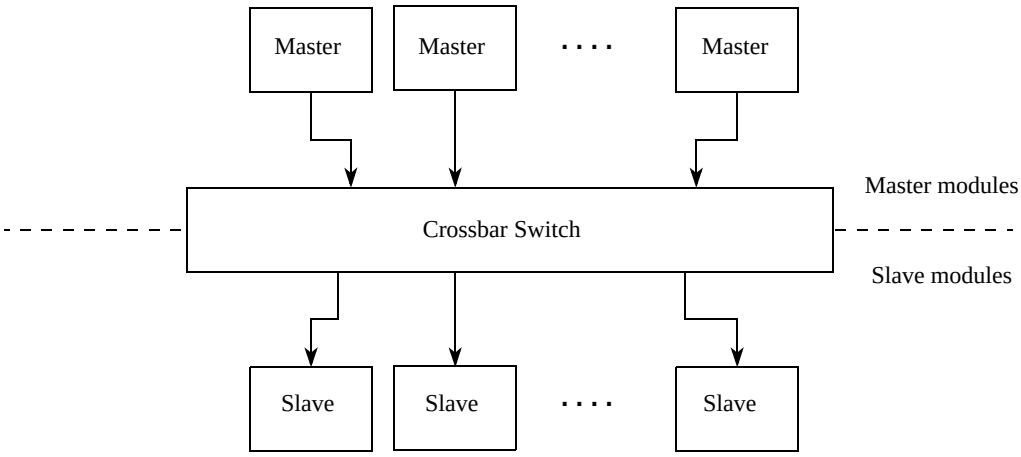


Figure 132. XBAR block diagram

Table 120 gives the crossbar switch port for each master and slave, the assigned and fixed ID number for each master and shows the master ID numbers as they relate to the master port numbers.

Table 120. Device XBAR switch ports

Module	Port		Physical master ID
	Type	Logical number	
e200z0 core—CPU instructions	Master	0	0
e200z0 core—Data	Master	0	1
eDMA	Master	2	2

Table 120. Device XBAR switch ports(Continued)

Module	Port		Physical master ID
	Type	Logical number	
FlexRay	Master	3	3
Flash Controller	Slave	0	—
Internal SRAM Controller	Slave	2	—
Peripheral bridge	Slave	7	—

14.3 Overview

The XBAR allows for concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grants it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

Requesting masters are granted access based on a fixed priority.

14.4 Features

- 4 Master ports
  - e200z0 core complex Instruction port
  - e200z0 core complex Load/Store Data port
  - eDMA
  - FlexRay
- 3 Slave ports
  - Flash memory (code and data)
  - Internal SRAM
  - Peripheral bridge
- 32-bit internal address, 32-bit internal data paths
- Fully concurrent transfers between independent master and slave ports
- Fixed priority scheme and fixed parking strategy

14.5 Modes of operation

14.5.1 Normal mode

In normal mode, the XBAR provides the register interface and logic that controls crossbar switch configuration.

14.5.2 Debug mode

The XBAR operation in debug mode is identical to operation in normal mode.

14.6 Functional description

This section describes the functionality of the XBAR in more detail.

14.6.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls masters, or inserts bubbles on the slave side.

14.6.2 General operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master's priority level and the responding slave's access time.

Because the XBAR appears to be just another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed.

When a slave bus is idled by the XBAR, it is parked on the master that did the last transfer.

14.6.3 Master ports

A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely

transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master.

If the slave port is currently parked on another master, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

14.6.4 Slave ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a handoff of bus ownership occurs and there are no wait states from the slave port. A requesting master that does not own the slave port is granted access after a one clock delay.

14.6.5 Priority assignment

Each master port is assigned a fixed 3-bit priority level (hard-wired priority). *Table 121* shows the priority levels assigned to each master (the lowest has highest priority).

Table 121. Hardwired bus master priorities

Module	Port		Priority level
	Type	Number	
e200z0 core–CPU instructions	Master	0	7
e200z0 core—Data	Master	1	6
eDMA	Master	2	5
FlexRay	Master	3	4

14.6.6 Arbitration

XBAR supports only a fixed-priority comparison algorithm.

14.6.6.1 Fixed priority operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

14.6.6.1.1 Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks always to the most recently requesting master (park-on-last). When parked on the last master, the slave port is passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.

15 Error Correction Status Module (ECSM)

15.1 Introduction

The Error Correction Status Module (ECSM) provides control functions for the device Standard Product Platform (SPP) including program-visible information about the platform configuration and revision levels, a reset status register, a software watchdog timer, and wakeup control for exiting sleep modes, and optional features such as an address map for the device's crossbar switch, information on memory errors reported by error-correcting codes and/or generic access error information for certain processor cores. It also provides with register access protection for the following slave modules: INTC, ECSM, STM, and SWT.

15.2 Overview

The Error Correction Status Module is mapped into the IPS space and supports a number of control functions for the platform device.

15.3 Features

The ECSM includes these features:

- Program-visible information on the platform device configuration and revision
- Reset status register (MRSR)
- Registers for capturing information on platform memory errors if error-correcting codes (ECC) are implemented
- Registers to specify the generation of single- and double-bit memory data inversions for test purposes if error-correcting codes are implemented
- Access address information for faulted memory accesses for certain processor core micro-architectures
- XBAR priority functions, including forcing round robin and high priority enabling
- Capability to restrict register access to supervisor mode to selected on-platform slave devices: INTC, ECSM, STM, and SWT

15.4 Memory map and registers description

This section details the programming model for the ECSM. This is an on-platform 128-byte space mapped to the region serviced by an IPS bus controller. Some of the control registers have a 64-bit width. These 64-bit registers are implemented as two 32-bit registers, and include an "H" and "L" suffixes, indicating the "high" and "low" portions of the control function.

The Error Correction Status Module does not include any logic that provides access control. Rather, this function is supported using the standard access control logic provided by the IPS controller.

ECSM registers are accessible only when the core is in supervisor mode (see *Section 15.4.3: ECSM_reg_protection*).

15.4.1 Memory map

Table 122 lists the registers in the ECSM.

Table 122. ECSM registers

Offset from ECSM_BASE 0xFFF4_0000	Register	Location	Size (bits)
0x0000	PCT—Processor Core Type register	*on page 313*	16
0x0002	REV—Revision register	*on page 313*	16
0x0004	PLAMC — Platform XBAR Master Configuration	*on page 314*	16
0x0006	PLASC — Platform XBAR Sleave Configuration	*on page 314*	16
0x0008	IMC—IPS Module Configuration register	*on page 315*	16
0x000C–0x000E	Reserved		
0x000F	MRSR—Miscellaneous Reset Status register	*on page 315*	8
0x0010–0x001E	Reserved		
0x001F	MIR—Miscellaneous Interrupt register	*on page 316*	8
0x0020–0x0023	Reserved		
0x0024	MUDCR—Miscellaneous User-Defined Control Register	*on page 316*	32
0x0028–0x0042	Reserved		
0x0043	ECR—ECC Configuration register	*on page 318*	8
0x0044–0x0046	Reserved		
0x0047	ESR—ECC Status register	*on page 319*	8
0x0048–0x0049	Reserved		
0x004A	EEGR—ECC Error Generation register	*on page 320*	16
0x004C–0x004F	Reserved		
0x0050	FEAR—Flash ECC Address register	*on page 322*	32
0x0054–0x0055	Reserved		
0x0056	FEMR—Flash ECC Master Number Register	*on page 323*	8
0x0057	FEAT—Flash ECC Attributes register	*on page 324*	8
0x0058–0x005B	Reserved		
0x005C	FEDR—Flash ECC Data register	*on page 324*	32
0x0060	REAR—RAM ECC Address register	*on page 325*	32
0x0064	Reserved		
0x0065	RESR—RAM ECC Syndrome register	*on page 326*	8
0x0066	REMR—RAM ECC Master register	*on page 328*	8
0x0067	REAT—RAM ECC Attributes register	*on page 328*	8

Table 122. ECSM registers(Continued)

Offset from ECSM_BASE 0xFFF4_0000	Register	Location	Size (bits)
0x0068–0x006B	Reserved		
0x006C	REDR—RAM ECC Data register	*on page 329*	32
0x0070–0x3FFF	Reserved		

15.4.2 Registers description

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, *writes to the programming model must match the size of the register*, e.g., an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

15.4.2.1 Processor core type (PCT) register

The PCT is a 16-bit read-only register that specifies the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PCT[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0

Figure 133. Processor core type (PCT) register

Table 123. PCT field descriptions

Name	Description
0-15 PCT[15:0]	Processor Core Type 0xE012 identifies the z0H Power Architecture.

15.4.2.2 Revision (REV) register

The REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address: Base + 0x0002 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REV[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 134. Revision (REV) register

Table 124. REV field descriptions

Name	Description
0-15 REV[15:0]	Revision The REV[15:0] field is specified by an input signal to define a software-visible revision number.

15.4.2.3 Platform XBAR Master Configuration (PLAMC)

The PLAMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the device's AMBA-AHB Crossbar Switch (XBAR). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address Base + 0x0004 Access: User read/-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	AMC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	AMC[7:0]							

Figure 135. Platform XBAR Master Configuration (PLAMC) register

Table 125. PLAMC field descriptions

Field	Description
AMC[7:0]	XBAR Master Configuration 0 Bus master connection to XBAR input port *n* is not present. 1 Bus master connection to XBAR input port *n* is present.

15.4.2.4 Platform XBAR Slave Configuration (PLASC)

The PLASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the device's AMBA-AHB Crossbar Switch (XBAR), plus a 1-bit flag defining the internal platform datapath width (DP64). The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address Base + 0x0006 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP64	0	0	0	0	0	0	0	ASC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	ASC[7:0]							

Figure 136. Platform XBAR Slave Configuration (PLASC) register

Table 126. ASC field descriptions

Field	Description
DP64	64-bit Datapath 0 Datapath width is 32 bits. 1 Datapath width is 64 bits.
ASC[7:0]	XBAR Slave Configuration 0 Bus slave connection to XBAR output port *n* is not present. 1 Bus slave connection to XBAR output port *n* is present.

15.4.2.5 IPS Module Configuration (IMC) register

The IMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order IPS peripheral modules connected to the primary slave bus controller. The state of this register is defined by a module input signal; it can only be read from the IPS programming model. Any attempted write is ignored.

Address Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MC[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MC[15:0]															
W																
Reset	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 137. IPS Module Configuration (IMC) register

Table 127. IMC field descriptions

Field	Description
0-31 MC[31:0]	IPS Module Configuration 0 IPS module connection to decoded slot *n* not present 1 IPS module connection to decoded slot *n* present

15.4.2.6 Miscellaneous Reset Status Register (MRSR)

The MRSR contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the MRSR, reflecting the cause of the most recent reset as signaled by device reset input signals. The MRSR can only be read from the IPS programming model. Any attempted write is ignored.

Address: Base + 0x000F Access: User read-only

	0	1	2	3	4	5	6	7
R	POR	DIR	0	0	0	0	0	0
W								
Reset	x	x	0	0	0	0	0	0

Figure 138. Miscellaneous Reset Status Register (MRSR)

Table 128. MRSR field descriptions

Field	Description
0 POR	Power-On Reset 0 Last recorded event was not caused by a power-on reset (based on a device input signal). 1 Last recorded event was caused by a power-on reset (based on a device input signal).
1 DIR	Device Input Reset 0 Last recorded event was not caused by a device input reset. 1 Last recorded event was a reset caused by a device input reset.

15.4.2.7 Miscellaneous Interrupt Register (MIR)

All interrupt requests associated with ECSM are collected in the MIR register. This includes the processor core system bus fault interrupt.

During the appropriate interrupt service routine handling these requests, the interrupt source contained in the ECSMIR must be explicitly cleared.

Address: Base + 0x001F Access: User read/write

	0	1	2	3	4	5	6	7
R	FB0AI	FB0SI	FB1AI	FB1SI	0	0	0	0
W	1	1	1	1	x	x	x	x
Reset	0	0	0	0	0	0	0	0

Figure 139. Miscellaneous Interrupt Register (MIR)

Table 129. MIR field descriptions

Field	Description
0 FB0AI	Flash Bank 0 Abort Interrupt 0 A flash bank 0 abort has not occurred. 1 A flash bank 0 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
1 FB0SI	Flash Bank 0 Stall Interrupt 0 A flash bank 0 stall has not occurred. 1 A flash bank 0 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
2 FB1AI	Flash Bank 1 Abort Interrupt 0 A flash bank 1 abort has not occurred. 1 A flash bank 1 abort has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
3 FB1SI	Flash Bank 1 Stall Interrupt 0 A flash bank 1 stall has not occurred. 1 A flash bank 1 stall has occurred. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.

15.4.2.8 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions. It provides configuration control for assorted modules on the device. The contents of this

register is output from the ECSM to other modules where these user-defined control functions are implemented.

Address Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MUDCR[31]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 140. Miscellaneous User-Defined Control register (MUDCR)

Table 130. MUDCR field descriptions

Name	Description
0 MUDCR[31]	XBAR force_round_robin bit This bit is used to drive the force_round_robin bit of the XBAR. This forces the slaves into round robin mode of arbitration rather than fixed mode (unless a master is using priority elevation, which forces the design back into fixed mode regardless of this bit). By setting the hardware definition to ENABLE_ROUND_ROBIN_RESET, this bit resets to 1. 0 XBAR is in fixed priority mode. 1 XBAR is in round robin mode.

15.4.2.9 ECC registers

There are a number of program-visible registers for the sole purpose of reporting and logging of memory failures. These registers include the following:

- ECC Configuration Register (ECR)
- ECC Status Register (ESR)
- ECC Error Generation Register (EEGR)
- Flash ECC Address Register (FEAR)
- Flash ECC Master Number Register (FEMR)
- Flash ECC Attributes Register (FEAT)
- Flash ECC Data Register (FEDR)
- RAM ECC Address Register (REAR)
- RAM ECC Syndrome Register (RESR)
- RAM ECC Master Number Register (REMR)
- RAM ECC Attributes Register (REAT)
- RAM ECC Data Register (REDR)

The details on the ECC registers are provided in the subsequent sections. If the design does not include ECC on the memories, these addresses are reserved locations within the ECSM's programming model.

15.4.2.10 ECC Configuration Register (ECR)

The ECC Configuration Register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches, which are discarded due to a change-of-flow operation, and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

The reporting of single-bit memory corrections can only be enabled via an SoC-configurable module input signal. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. While not directly accessible to a user, this capability is viewed as important for error logging and failure analysis.

Address: Base + 0x0043 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 141. ECC Configuration register (ECR)

Table 131. ECR field descriptions

Field	Description
2 ER1BR	Enable RAM 1-bit Reporting This bit can only be set if the input enable signal is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. The occurrence of a single-bit RAM correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[R1BC]. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0 Reporting of single-bit RAM corrections disabled 1 Reporting of single-bit RAM corrections enabled
3 EF1BR	Enable Flash 1-bit Reporting This bit can only be set if the input enable signal is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled. The occurrence of a single-bit flash correction generates a ECSM ECC interrupt request as signaled by the assertion of ESR[F1BC]. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0 Reporting of single-bit flash corrections disabled 1 Reporting of single-bit flash corrections enabled
6 ERNCR	Enable RAM Non-Correctable Reporting The occurrence of a non-correctable multi-bit RAM error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[RNCE]. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. 0 Reporting of non-correctable RAM errors disabled 1 Reporting of non-correctable RAM errors enabled

Table 131. ECR field descriptions(Continued)

Field	Description
7 EFNCR	Enable Flash Non-Correctable Reporting The occurrence of a non-correctable multi-bit flash error generates a ECSM ECC interrupt request as signaled by the assertion of ESR[FNCE]. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. 0 Reporting of non-correctable flash errors disabled 1 Reporting of non-correctable flash errors enabled

15.4.2.11 ECC Status Register (ESR)

The ECC Status Register is an 8-bit control register for signaling which types of properly enabled ECC events have been detected. The ESR signals the last properly enabled memory event to be detected. ECC interrupt generation is separated into single-bit error detection/correction, uncorrectable error detection, and the combination of the two as defined by the following boolean equations:

```
ECSM_ECC1BIT_IRQ
    =  ECR[ER1BR] &  ESR[R1BC]// ram, 1-bit correction
    |  ECR[EF1BR] &  ESR[F1BC]// flash, 1-bit correction
ECSM_ECCRNCR_IRQ
    =  ECR[ERNCR] &  ESR[RNCE]// ram, noncorrectable error
ECSM_ECCFNCR_IRQ
    =  ECR[EFNCR] &  ESR[FNCE]// flash, noncorrectable error
ECSM_ECC2BIT_IRQ
    =  ECSM_ECCRNCR_IRQ// ram, noncorrectable error
    |  ECSM_ECCFNCR_IRQ// flash, noncorrectable error
ECSM_ECC_IRQ
    =  ECSM_ECC1BIT_IRQ // 1-bit correction
    |  ECSM_ECC2BIT_IRQ// noncorrectable error
```

where the combination of a properly enabled category in the ECR and the detection of the corresponding condition in the ESR produces the interrupt request.

The ECSM allows a maximum of one bit of the ESR to be asserted at any given time. This preserves the association between the ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly enabled ECC event. If there is a pending ECC interrupt and another properly enabled ECC event occurs, the ECSM hardware automatically handles the ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ESR and verify the current contents matches the original contents. If the two values are different, go back to step 1 and repeat.
4. When the values are identical, write a 1 to the asserted ESR flag to negate the interrupt request.

Address: Base + 0x0047 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W								
Reset	0	0	0	0	0	0	0	0

Figure 142. ECC Status register (ESR)

Table 132. ESR field descriptions

Field	Description
2 R1BC	RAM 1-bit Correction This bit can only be set if ECR[ER1BR] is asserted. The occurrence of a properly enabled single-bit RAM correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit RAM correction detected 1 Reportable single-bit RAM correction detected
3 F1BC	Flash 1-bit Correction This bit can only be set if ECR[EF1BR] is asserted. The occurrence of a properly enabled single-bit flash correction generates a ECSM ECC interrupt request. The address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit flash correction detected 1 Reportable single-bit flash correction detected
6 RNCE	RAM Non-Correctable Error The occurrence of a properly enabled non-correctable RAM error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the REAR, RESR, REMR, REAT and REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable RAM error detected 1 Reportable non-correctable RAM error detected
7 FNCE	Flash Non-Correctable Error The occurrence of a properly enabled non-correctable flash error generates a ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the FEAR, FEMR, FEAT and FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. This bit can only be set if ECR[ERNCR] is asserted. 0 No reportable non-correctable flash error detected 1 Reportable non-correctable flash error detected

In the event that multiple status flags are signaled simultaneously, ECSM records the event with the R1BC as highest priority, then F1BC, then RNCE, and finally FNCE.

15.4.2.12 ECC Error Generation Register (EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the RAM, similar capabilities exist for the flash, that is, the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified.

For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response.

The enabling of these error generation modes requires the same input enable signal (as that used to enable single-bit correction reporting) be asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.

Address Base + 0x004A Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	FRC1BI	FR11BI	0	0	FRCNCI	FR1NCI	0	ERRBIT[6:0]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 143. ECC Error Generation register (EEGR)

Table 133. EEGR field descriptions

Field	Description
2 FRC1BI	Force RAM Continuous 1-Bit Data Inversions 0 No RAM continuous 1-bit data inversions generated 1 1-bit data inversions in the RAM continuously generated The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT[6:0], continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.
3 FR11BI	Force RAM One 1-bit Data Inversion 0 No RAM single 1-bit data inversion generated 1 One 1-bit data inversion in the RAM generated The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT[6:0], on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. This bit can only be set if the same input enable signal (as that used to enable single-bit correction reporting) is asserted. This signal is tied to 1 at SoC level and hence reporting of single-bit memory corrections is always enabled.

Table 133. EEGR field descriptions(Continued)

Field	Description
6 FRCNCI	Force RAM Continuous Non-Correctable Data Inversions 0 No RAM continuous 2-bit data inversions generated 1 2-bit data inversions in the RAM continuously generated The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM.
7 FR1NCI	Force RAM One Non-Correctable Data Inversions 0 No RAM single 2-bit data inversions generated 1 One 2-bit data inversion in the RAM generated The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT[6:0] and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic.
9-15 ERRBIT [6:0]	Error Bit Position The vector defines the bit position that is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The RAM controller follows a vector bit ordering scheme where LSB=0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width. For example, consider a 32-bit RAM implementation. The 32-bit ECC approach requires 7 code bits for a 32-bit word. For PRAM data width of 32 bits, the actual SRAM (32 bits data + 7 bits for ECC) = 39 bits. The following association between the ERRBIT field and the corrupted memory bit is defined: if ERRBIT = 0, then RAM[0] of the odd bank is inverted. if ERRBIT = 1, then RAM[1] of the odd bank is inverted. ... if ERRBIT = 31, then RAM[31] of the odd bank is inverted. if ERRBIT = 64, then ECC Parity[0] of the odd bank is inverted. if ERRBIT = 65, then ECC Parity[1] of the odd bank is inverted. ... if ERRBIT = 70, then ECC Parity[6] of the odd bank is inverted. For ERRBIT values of 32 to 63 and greater than 70, no bit position is inverted.

If an attempt to force a non-correctable inversion (by asserting EEGR[FRCNCI] or EEGR[FRC1NCI]) and EEGR[ERRBIT] equals 64, then no data inversion will be generated.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

15.4.2.13 Flash ECC Address Register (FEAR)

The FEAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access

to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address Base + 0x0050 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEAR[31:16]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEAR[15:0]															
W																
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 144. Flash ECC Address register (FEAR)

Table 134. FEAR field descriptions

Field	Description
0-31 FEAR[31:0]	Flash ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled flash ECC event.

15.4.2.14 Flash ECC Master Number Register (FEMR)

The FEMR is a 4-bit register for capturing the XBAR bus master number of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0056 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR[3:0]			
W								
Reset	0	0	0	0	–	–	–	–

Figure 145. Flash ECC Master Number Register (FEMR)

Table 135. FEMR field descriptions

Name	Description
4-7 FEMR[3:0]	Flash ECC Master Number Register This 4-bit register contains the XBAR bus master number of the faulting access of the last properly enabled flash ECC event.

15.4.2.15 Flash ECC Attributes (FEAT) register

The FEAT is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE[2:0]			PROTECTION[3:0]			
W								
Reset	0	0	0	0	–	–	–	–

Figure 146. Flash ECC Attributes (FEAT) Register

Table 136. FEAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

15.4.2.16 Flash ECC Data Register (FEDR)

The FEDR is a 32-bit register for capturing the data associated with the last properly enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the FEAR, FEMR, FEAT and FEDR registers, and the appropriate flag (F1BC or FNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x005C Access: User read-only

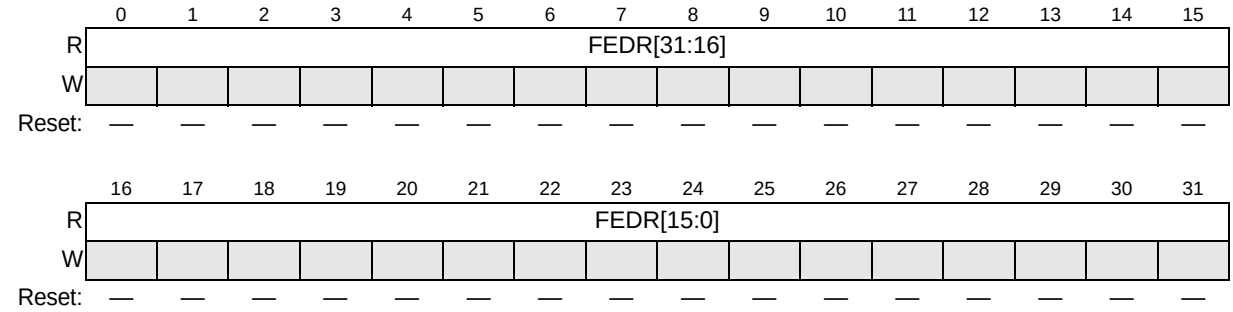


Figure 147. Flash ECC Data register (FEDR)

Table 137. FEDR field descriptions

Name	Description
0-31 FEDR[31:0]	Flash ECC Data Register This 32-bit register contains the data associated with the faulting access of the last properly enabled flash ECC event. The register contains the data value taken directly from the data bus.

15.4.2.17 RAM ECC Address Register (REAR)

The REAR is a 32-bit register for capturing the address of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0060 Access: User read-only

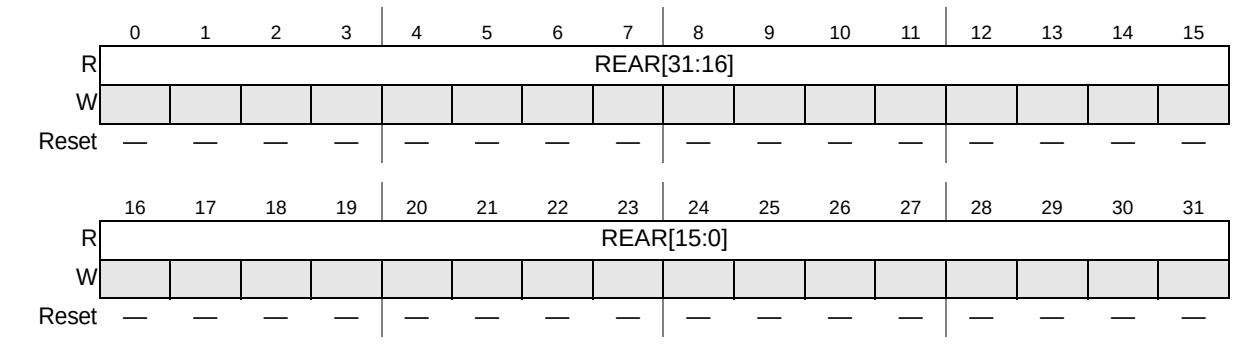


Figure 148. RAM ECC Address register (REAR)

Table 138. REAR field descriptions

Name	Description
0-31 REAR[31:0]	RAM ECC Address Register This 32-bit register contains the faulting access address of the last properly enabled RAM ECC event.

15.4.2.18 RAM ECC Syndrome Register (RESR)

The RESR is an 8-bit register for capturing the error syndrome of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0065 Access: User read-only

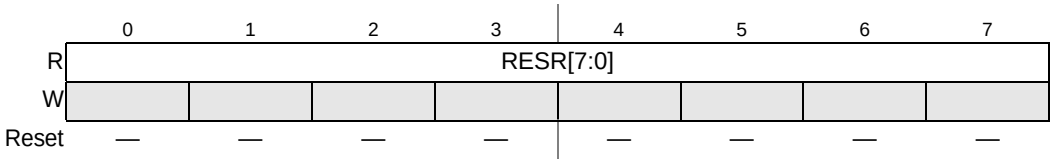


Figure 149. RAM ECC Syndrome Register (RESR)

Table 139. RESR field descriptions

Name	Description
0-7 RESR[7:0]	RAM ECC Syndrome Register This 8-bit syndrome field includes 6 bits of Hamming decoded parity plus an odd-parity bit for the entire 39-bit (32-bit data + 7 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown in *Table 140* associates the upper 7 bits of the syndrome with the data bit in error.

Note: Table 140 associates the upper 7 bits of the ECC syndrome with the exact data bit in error for single-bit correctable codewords. This table follows the bit vectoring notation where the LSB=0. Note that the syndrome value of 0x0001 implies no error condition but this value is not readable when the PRESR is read for the no error case.

Table 140. RAM syndrome mapping for single-bit correctable errors

RESR[7:0]	Data Bit in Error
0x00	ECC ODD[0]
0x01	No Error
0x02	ECC ODD[1]
0x04	ECC ODD[2]
0x06	DATA ODD BANK[31]

Table 140. RAM syndrome mapping for single-bit correctable errors(Continued)

RESR[7:0]	Data Bit in Error
0x08	ECC ODD[3]
0x0A	DATA ODD BANK[30]
0x0C	DATA ODD BANK[29]
0x0E	DATA ODD BANK[28]
0x10	ECC ODD[4]
0x12	DATA ODD BANK[27]
0x14	DATA ODD BANK[26]
0x16	DATA ODD BANK[25]
0x18	DATA ODD BANK[24]
0x1A	DATA ODD BANK[23]
0x1C	DATA ODD BANK[22]
0x50	DATA ODD BANK[21]
0x20	ECC ODD[5]
0x22	DATA ODD BANK[20]
0x24	DATA ODD BANK[19]
0x26	DATA ODD BANK[18]
0x28	DATA ODD BANK[17]
0x2A	DATA ODD BANK[16]
0x2C	DATA ODD BANK[15]
0x58	DATA ODD BANK[14]
0x30	DATA ODD BANK[13]
0x32	DATA ODD BANK[12]
0x34	DATA ODD BANK[11]
0x64	DATA ODD BANK[10]
0x38	DATA ODD BANK[9]
0x62	DATA ODD BANK[8]
0x70	DATA ODD BANK[7]
0x60	DATA ODD BANK[6]
0x40	ECC ODD[6]
0x42	DATA ODD BANK[5]
0x44	DATA ODD BANK[4]
0x46	DATA ODD BANK[3]
0x48	DATA ODD BANK[2]

Table 140. RAM syndrome mapping for single-bit correctable errors(Continued)

RESR[7:0]	Data Bit in Error
0x4A	DATA ODD BANK[1]
0x4C	DATA ODD BANK[0]
0x03,0x05.....0x4D	Multiple bit error
> 0x4D	Multiple bit error

15.4.2.19 RAM ECC Master Number Register (REMR)

The REMR is an 8-bit register in which the 4-bit field REMR[0:3] is used for capturing the XBAR bus master number of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0066 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR[3:0]			
W								
Reset	0	0	0	0	—	—	—	—

Figure 150. RAM ECC Master Number register (REMR)

Table 141. REMR field descriptions

Name	Description
4-7 REMR[3:0]	RAM ECC Master Number Register This 4-bit field contains the XBAR bus master number of the faulting access of the last properly enabled RAM ECC event.

15.4.2.20 RAM ECC Attributes (REAT) register

The REAT is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x0067 Access: User read/write

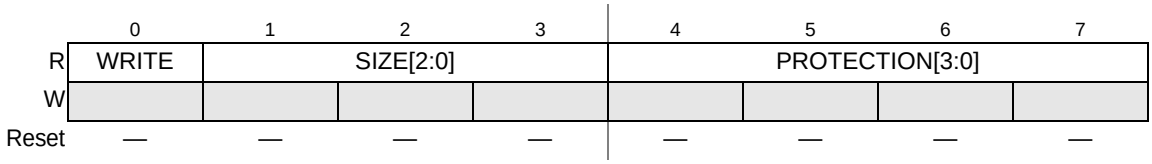


Figure 151. RAM ECC Attributes (REAT) register

Table 142. REAT field descriptions

Name	Description
0 WRITE	AMBA-AHB HWRITE 0 AMBA-AHB read access 1 AMBA-AHB write access
1-3 SIZE[2:0]	AMBA-AHB HSIZE[2:0] 000 8-bit AMBA-AHB access 001 16-bit AMBA-AHB access 010 32-bit AMBA-AHB access 1*xx* Reserved
4 PROTECTION[3]	AMBA-AHB HPROT[3] Protection[0]: Type 0 I-Fetch 1 Data
5 PROTECTION[2]	AMBA-AHB HPROT[2] Protection[1]: Mode 0 User mode 1 Supervisor mode
6 PROTECTION[1]	AMBA-AHB HPROT[1] Protection[2]: Bufferable 0 Non-bufferable 1 Bufferable
7 PROTECTION[0]	AMBA-AHB HPROT[0] Protection[3]: Cacheable 0 Non-cacheable 1 Cacheable

15.4.2.21 RAM ECC Data Register (REDR)

The REDR is a -bit register for capturing the data associated with the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC Configuration Register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the REAR, RESR, REMR, REAT and REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC Status Register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register can only be read from the IPS programming model; any attempted write is ignored.

Address: Base + 0x006C Access: User read-only

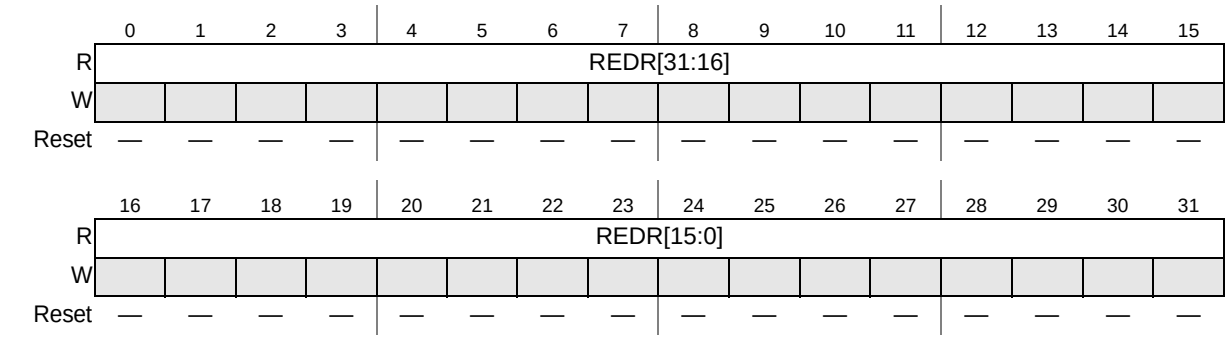


Figure 152. Platform RAM ECC Data register (PREDR)

Table 143. REDR field descriptions

Name	Description
0-31 REDR[31:0]	RAM ECC Data Register This 32-bit register contains the data associated with the faulting access of the last properly enabled RAM ECC event. The register contains the data value taken directly from the data bus.

15.4.3 ECSM_reg_protection

The ECSM_reg_protection logic provides hardware enforcement of supervisor mode access protection for four on-platform IPS modules: INTC, ECSM, STM, and SWT. This logic resides between the on-platform bus sourced by the PBRIDGE bus controller and the individual slave modules. It monitors the bus access type (supervisor or user) and if a user access is attempted, the transfer is terminated with an error and inhibited from reaching the slave module. Identical logic is replicated for each of the five, targeted slave modules. A block diagram of the ECSM_reg_protection module is shown in *Figure 153*.

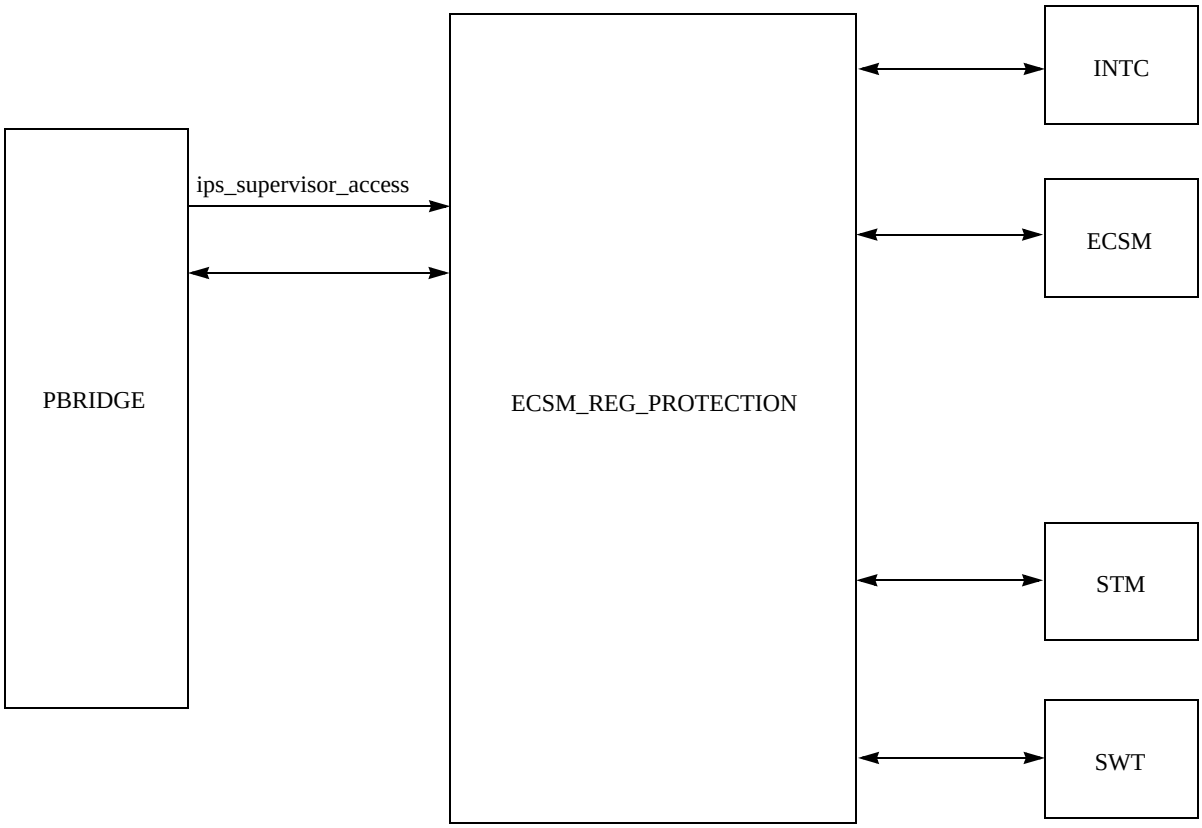


Figure 153. Spp_Ips_Reg_Protection block diagram

Attempted accesses to reserved addresses result in an error termination, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, writes to the programming model must match the size of the register; for example, an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

16 Internal Static RAM (SRAM)

16.1 Introduction

The general-purpose SRAM has a size of 40 KB.

The SRAM provides the following features:

- SRAM can be read/written from any bus master
- Byte, halfword, word and doubleword addressable
- Single-bit correction and double-bit error detection

16.2 SRAM operating mode

The SRAM has only one operating mode. No standby mode is available.

Table 144. SRAM operating modes

Mode	Configuration
Normal (functional)	Allows reads and writes of SRAM

16.3 Module memory map

The SRAM occupies up to 40 KB of address space.

Table 145 shows the SRAM memory map.

Table 145. SRAM memory map

Address	SPC560P44	SPC560P50
0x4000_0000 (Base)	36 KB RAM	40 KB RAM

16.4 Register descriptions

The SRAM has no registers. Registers associated with the ECC are located in the ECSM. See *Section 15.4.2.9: ECC registers*.

16.5 SRAM ECC mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors
- Detects 39-bit reads (32-bit data bus plus the 7-bit ECC) that return all zeros or all ones, asserts an error indicator on the bus cycle, and sets the error flag

SRAM does not detect all errors greater than 2 bits.

Internal SRAM write operations are performed on the following byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32-bit data bus. The 8-bit ECC is appended to the data segment and written to SRAM.

If the write operation is less than the entire 32-bit data width (1 or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32-bit data bus for errors, detecting and either correcting or flagging errors.
2. The write data bytes (1 or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data bus, and the 39-bit value is then written to SRAM.

16.5.1 Access timing

The system bus is a two-stage pipelined bus that makes the timing of any access dependent on the access during the previous clock. *Table 146* lists the various combinations of read and write operations to SRAM and the number of wait states used for the each operation. The table columns contain the following information:

- Current operation—Lists the type of SRAM operation currently executing
- Previous operation—Lists the valid types of SRAM operations that can precede the current SRAM operation (valid operation during the preceding clock)
- Wait states—Lists the number of wait states (bus clocks) the operation requires, which depends on the combination of the current and previous operation

Table 146. Number of wait states required for SRAM operations

Operation type	Current operation	Previous operation	Number of wait states required
Read	Read	Idle	1
		Pipelined read	
		8 , 16 or 32-bit write	0 (read from the same address)
			1 (read from a different address)
	Pipelined read	Read	0

Table 146. Number of wait states required for SRAM operations(Continued)

Operation type	Current operation	Previous operation	Number of wait states required
Write	8 or 16-bit write	Idle	1
		Read	
		Pipelined 8- or 16-bit write	2
		32-bit write	
		8 or 16-bit write	0 (write to the same address)
	Pipelined 8, 16 or 32-bit write	8 , 16 or 32-bit write	0
	32-bit write	Idle	0
		32-bit write	
		Read	

16.5.2 Reset effects on SRAM accesses

Asynchronous reset will possibly corrupt RAM if it asserts during a read or write operation to SRAM. The completion of that access depends on the cycle at which the reset occurs. If no access is occurring when reset occurs, RAM corruption does not happen.

Instead synchronous reset (SW reset) should be used in controlled function (without RAM accesses) in case initialization procedure is needed without RAM initialization.

16.6 Functional description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a R/W operation. Because the ECC bits can contain random data after the device is powered on, the SRAM must be initialized by executing 32-bit write operations prior any read accesses. This is also true for implicit read accesses caused by any write accesses smaller than 32 bits as discussed in *Section 16.5: SRAM ECC mechanism*.

16.7 Initialization and application information

To use the SRAM, the ECC must check all bits that require initialization after power on. All writes must specify an even number of registers performed on 32-bit word-aligned boundaries. If the write is not the entire 32-bits (8 or 16 bits), a read/modify/write operation is generated that checks the ECC value upon the read. Refer to *Section 16.5: SRAM ECC mechanism*.

Note: You must initialize SRAM, even if the application does not use ECC reporting.

17 Flash Memory

17.1 Introduction

The Flash memory comprises a platform Flash controller interface and two Flash memory arrays: one array of 512 KB for code (code Flash) and one array of 64 KB for data (data Flash). The Flash architecture of the SPC560P44Lx, SPC560P50Lx device is illustrated in *Figure 154*.

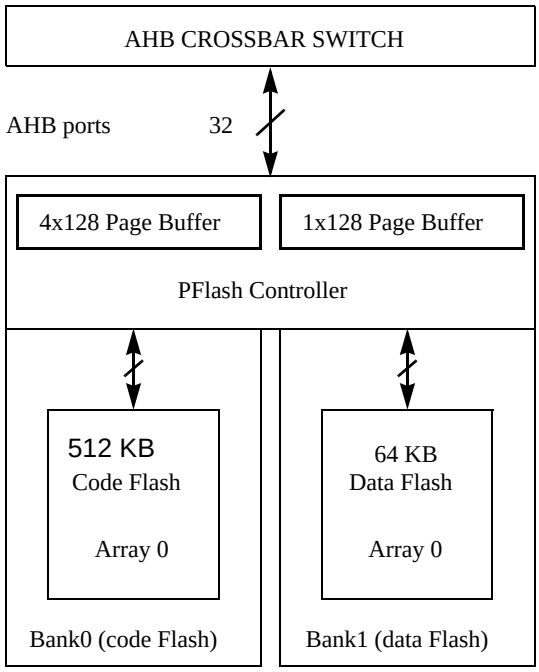


Figure 154. SPC560P44Lx, SPC560P50Lx Flash memory architecture

17.2 Platform Flash controller

17.2.1 Introduction

This section provides an introduction of the platform Flash controller, which acts as the interface between the system bus and as many as two banks of Flash memory arrays (program and data). It intelligently converts the protocols between the system bus and the dedicated Flash array interfaces. Several important terms are used to describe the platform Flash controller module and its connections. These terms are defined here.

- **Port**—This term describes the AMBA-AHB connection(s) into the platform Flash controller. From an architectural and programming model viewpoint, the definition supports as many as two AHB ports, even though this specific controller only supports a single AHB connection.
- **Bank**—This term describes the attached Flash memories. From the platform Flash controller's perspective, there may be one or two attached banks of Flash memory. The code Flash bank is required and always attached to bank0. Additionally, there is a data

Flash attached to bank1. The platform Flash controller interface supports two separate connections, one to each memory bank. On the SPC560P44Lx, SPC560P50Lx device, bank0 and bank1 are internal to the device.

- **Array**—Each memory bank has one Flash array instantiation.
- **Page**—This value defines the number of bits read from the Flash array in a single access. For this controller and memory, the page size is 128 bits (16 bytes).

The nomenclature “page buffers” and “line buffers” are used interchangeably.

17.2.1.1 Overview

The platform Flash controller supports a 32-bit data bus width at the AHB port and connections to 128-bit read data interfaces from two memory banks, where each bank contains one instantiation of the Flash memory array. One Flash bank is connected to the code Flash memory and the other bank is connected to the data Flash memory. The memory controller capabilities vary between the two banks with each bank’s functionality optimized with the typical use cases associated with the attached Flash memory. As an example, the platform Flash controller logic associated with the code Flash bank contains a four-entry “page” buffer, each entry containing 128 bits of data (1 Flash page) plus an associated controller that prefetches sequential lines of data from the Flash array into the buffer, while the controller logic associated with the data Flash bank only supports a 128-bit register that serves as a temporary page holding register and does not support any prefetching. Prefetch buffer hits from the code Flash bank support 0-wait AHB data phase responses. AHB read requests that miss the buffers generate the needed Flash array access and are forwarded to the AHB upon completion, typically incurring two wait states at an operating frequency of 60 to 64 MHz.

This memory controller is optimized for applications where a cacheless processor core, for example the Power e200z0h, is connected through the platform to on-chip memories, for example Flash and RAM, where the processor and platform operate at the same frequency. For these applications, the 2-stage pipeline AMBA-AHB system bus is effectively mapped directly into stages of the processor’s pipeline and 0 wait state responses for most memory accesses are critical for providing the required level of system performance.

17.2.1.2 Features

The following list summarizes the key features of the platform Flash controller:

- Single AHB port interface supports a 32-bit data bus. All AHB aligned and unaligned reads within the 32-bit container are supported. Only aligned word writes are supported.
- Array interfaces support a 128-bit read data bus and a 64-bit write data bus for each bank.
- Interface with code Flash (bank0) provides configurable read buffering and page prefetch support. Four page read buffers (each 128 bits wide) and a prefetch controller support single-cycle read responses (0 AHB data phase wait states) for hits in the buffers. The buffers implement a least-recently-used replacement algorithm to maximize performance.
- Interface with data Flash (bank1) includes a 128-bit register to temporarily hold a single Flash page. This logic supports single-cycle read responses (0 AHB data phase wait

states) for accesses that hit in the holding register. There is no support for prefetching associated with bank1.

- Programmable response for read-while-write sequences including support for stall-while-write, optional stall notification interrupt, optional Flash operation termination, and optional termination notification interrupt
- Separate and independent configurable access timing (on a per bank basis) to support use across a wide range of platforms and frequencies
- Support of address-based read access timing for emulation of other memory types
- Support for reporting of single- and multi-bit Flash ECC events
- Typical operating configuration loaded into programming model by system reset

17.2.2 Modes of operation

The platform Flash controller module does not support any special modes of operation. Its operation is driven from the AMBA-AHB memory references it receives from the platform's bus masters. Its configuration is defined by the setting of the programming model registers, physically located as part of the Flash array modules.

17.2.3 External signal descriptions

The platform Flash controller does not directly interface with any external signals. Its primary internal interfaces include a connection to an AMBA-AHB crossbar (or memory protection unit) slave port and connections with as many as two banks (code and data) of Flash memory, each containing one instantiation of the Flash array. Additionally, the operating configuration for the platform Flash controller is defined by the contents of certain code Flash array0 registers that are inputs to the module.

17.2.4 Memory map and registers description

Two memory maps are associated with the platform Flash controller: one for the Flash memory space and another for the program-visible control and configuration registers. The Flash memory space is accessed via the AMBA-AHB port. The program-visible registers are accessed via the slave peripheral bus. Details on both memory spaces are provided in *Section 17.2.4.1: Memory map*.

There are no program-visible registers that physically reside inside the platform Flash controller. Rather, the platform Flash controller receives control and configuration information from the Flash array controller(s) to determine the operating configuration. These are part of the Flash array's configuration registers mapped into its slave peripheral (IPS) address space but are described here.

Note: Updating the configuration fields that control the platform flash controller behavior should only occur while the flash controller is idle. Changing configuration settings while a flash access is in progress can lead to non-deterministic behavior.

17.2.4.1 Memory map

First, consider the Flash memory space accessed via transactions from the platform Flash controller's AHB port. To support the two separate Flash memory banks, the platform Flash controller uses address bit 23 (haddr[23]) to steer the access to the appropriate memory bank. In addition to the actual Flash memory regions, there are shadow and test sectors included in the system memory map. The program-visible control and configuration registers associated with each memory array are included in the slave peripheral address

region. The system memory map defines one code Flash array and one data Flash array. See *Table 147*.

Caution: Software executing from flash memory must not write to registers that control flash behavior (such as wait state settings or prefetch enable/disable). Doing so can cause data corruption. On this chip, these registers include PFCR0 and PFAPR.

Note: Flash memory configuration registers should be written only with 32-bit write operations to avoid any issues associated with register incoherency caused by bit fields spanning smaller size (8-, 16-bit) boundaries.

Table 147. Flash-related regions in the system memory map

Start address	End address	Size (KB)	Region
0x0000_0000	0x0007_FFFF	512	Code Flash array 0
0x0008_0000	0x001F_FFFF	1536	Reserved
0x0020_0000	0x0020_3FFF	16	Code Flash array 0: shadow sector
0x0020_4000	0x003F_FFFF	2032	Reserved
0x0040_0000	0x0040_3FFF	16	Code Flash array 0: test sector
0x0040_4000	0x007F_FFFF	4080	Reserved
0x0080_0000	0x0080_FFFF	64	Data Flash array 0
0x0081_0000	0x00BF_FFFF	4032	Reserved
0x00C0_0000	0x00C0_3FFF	16	Data Flash array 0: test sector
0x00C0_4000	0x00FF_FFFF	4080	Reserved
0x0100_0000	0x1FFF_FFFF	507904	Emulation Mapping
0xFFE8_8000	0xFFE8_BFFF	16	Code Flash array 0 configuration(1)
0xFFE8_C000	0xFFE8_FFFF	16	Data Flash array 0 configuration(1)
0xFFEB_0000	0xFFEB_BFFF	48	Reserved

1. This region is also aliased to address 0xC3F8_nnnn.

For additional information on the address-based read access timing for emulation of other memory types, see *Section 17.2.17: Wait state emulation*.

Next, consider the memory map associated with the control and configuration registers.

There are multiple registers that control operation of the platform Flash controller. Note the first two Flash array registers (PFCR0, PFCR1) are reset to a device-defined value, while the remaining register (PFAPR) is loaded at reset from specific locations in the array's shadow region.

Regardless of the number of populated banks or the number of Flash arrays included in a given bank, the configuration of the platform Flash controller is wholly specified by the platform Flash controller control registers associated with code Flash array0. The code array0 register settings define the operating behavior of ***both*** Flash banks. It is recommended to set the platform Flash controller control registers for both arrays to the array0 values.

Note: To perform program and erase operations, the control registers in the actual referenced Flash array must both be programmed, but the configuration of the platform Flash controller module is defined by the platform Flash controller control registers of code array0.

The 32-bit memory map for the platform Flash controller control registers is shown in *Table 148*.

Table 148. Platform Flash controller 32-bit memory map

Offset from PFlash_BASE (0xFFE8_8000)	Register	Location
0x001C	Platform Flash Configuration Register 0 (PFCR0)	*on page 380*
0x0020	Platform Flash Configuration Register 1 (PFCR1)	*on page 383*
0x0024	Platform Flash Access Protection Register (PFAPR)	*on page 385*

17.2.5 Functional description

The platform Flash controller interfaces between the AHB-Lite 2.v6 system bus and the Flash memory arrays.

The platform Flash controller generates read and write enables, the Flash array address, write size, and write data as inputs to the Flash array. The platform Flash controller captures read data from the Flash array interface and drives it onto the AHB. As much as four pages of data (128-bit width) from bank0 are buffered by the platform Flash controller. Lines may be prefetched in advance of being requested by the AHB interface, allowing single-cycle (0 AHB wait states) read data responses on buffer hits.

Several prefetch control algorithms are available for controlling page read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis.

Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch.

Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

Throughout this discussion, bk*n*_ is used as a prefix to refer to two signals, each for each bank: bk0_ and bk1_. Also, the nomenclature B*x*_P*y*_RegName is used to reference a program-visible register field associated with bank “*x*” and port “*y*”.

17.2.6 Basic interface protocol

The platform Flash controller interfaces to the Flash array by driving addresses (bk*n*_fl_addr[23:0]) and read or write enable signals (bk*n*_fl_rd_en, bk*n*_fl_wr_en).

The read or write enable signal (bk*n*_fl_rd_en, bk*n*_fl_wr_en) is asserted in conjunction with the reference address for a single rising clock when a new access request is made.

Addresses are driven to the Flash array in a flow-through fashion to minimize array access time. When no outstanding access is in progress, the platform Flash controller drives addresses and asserts bk*n*_fl_rd_en or bk*n*_fl_wr_en and then may change to the next outstanding address in the next cycle.

Accesses are terminated under control of the appropriate read/write wait state control setting. Thus, the access time of the operation is determined by the settings of the wait state control fields. Access timing can be varied to account for the operating conditions of the device (frequency, voltage, temperature) by appropriately setting the fields in the programming model for either bank.

The platform Flash controller also has the capability of extending the normal AHB access time by inserting additional wait states for reads and writes. This capability is provided to allow emulation of other memories that have different access time characteristics. The added wait state specifications are provided by bit 28 to bit 24 of Flash address (haddr[28:24], see *Table 150* and *Table 151*). These wait states are applied in addition to the normal wait states incurred for Flash accesses. Refer to *Section 17.2.17: Wait state emulation* for more details.

Prefetching of next sequential page is blocked when haddr[28:24] is non-zero. Buffer hits are also blocked as well, regardless of whether the access corresponds to valid data in one of the page read buffers. These steps are taken to ensure that timing emulation is correct and that excessive prefetching is avoided. In addition, to prevent erroneous operation in certain rare cases, the buffers are invalidated on any non-sequential AHB access with a non-zero value on haddr[28:24].

17.2.7 Access protections

The platform Flash controller provides programmable configurable access protections for both read and write cycles from masters via the Platform Flash Access Protection Register (PFAPR). It allows restriction of read and write requests on a per-master basis. This functionality is described in *Section 17.3.7.7.3: Platform Flash Access Protection Register (PFAPR)*. Detection of a protection violation results in an error response from the platform Flash controller on the AHB transfer.

17.2.8 Read cycles — buffer miss

Read cycles from the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0] and asserting bk*n*_fl_rd_en for the required setup (and hold) time before (and after) the rising edge of hclk. The platform Flash controller then waits for the programmed number of read wait states before sampling the read data on bk*n*_fl_rdata[127:0]. This data is normally stored in the least-recently updated page read buffer for bank0 in parallel with the requested data being forwarded to the AHB. For bank1, the data is captured in the page-wide temporary holding register as the requested data is forwarded to the AHB bus. Timing diagrams of basic read accesses from the Flash array are shown in *Figure 155* through *Figure 158*.

If the Flash access was the direct result of an AHB transaction, the page buffer is marked as most-recently-used as it is being loaded. If the Flash access was the result of a speculative prefetch to the next sequential line, it is first loaded into the least-recently-used buffer. The status of this buffer is not changed to most-recently-used until a subsequent buffer hit occurs.

17.2.9 Read cycles — buffer hit

Single cycle read responses to the AHB are possible with the platform Flash controller when the requested read access was previously loaded into one of the bank0 page buffers. In these “buffer hit” cases, read data is returned to the AHB data phase with a 0 wait state response.

Likewise, the bank1 logic includes a single 128-bit temporary holding register and sequential accesses that “hit” in this register are also serviced with a 0 wait state response.

17.2.10 Write cycles

In a write cycle, address, write data, and control signals are launched off the same edge of hclk at the completion of the first AHB data phase cycle. Write cycles to the Flash array are initiated by driving a valid access address on bk*n*_fl_addr[23:0], driving write data on bk*n*_fl_wdata[63:0], and asserting bk*n*_fl_wr_en. Again, the controller drives the address and control information for the required setup time before the rising edge of hclk, and provides the required amount of hold time. The platform Flash controller then waits for the appropriate number of write wait states before terminating the write operation. On the cycle following the programmed wait state value, the platform Flash controller asserts hready_out to indicate to the AHB port that the cycle has terminated.

17.2.11 Error termination

The platform Flash controller follows the standard procedure when an AHB bus cycle is terminated with an ERROR response. First, the platform Flash controller asserts hresp[0] and negates hready_out to signal an error has occurred. On the following clock cycle, the platform Flash controller asserts hready_out and holds both hresp[0] and hready_out asserted until hready_in is asserted.

The first case that can cause an error response to the AHB is when an access is attempted by an AHB master whose corresponding Read Access Control or Write Access Control settings do not allow the access, thus causing a protection violation. In this case, the platform Flash controller does not initiate a Flash array access.

The second case that can cause an error response to the AHB is when an access is performed to the Flash array and is terminated with a Flash error response. See *Section 17.2.13: Flash error response operation*. This may occur for either a read or a write operation.

The third case that can cause an error response to the AHB is when a write access is attempted to the Flash array and is disallowed by the state of the bk*n*_fl_ary_access control input. This case is similar to case 1.

A fourth case involves an attempted read access while the Flash array is busy doing a write (program) or erase operation if the appropriate read-while-write control field is programmed for this response. The 3-bit read-while-write control allows for immediate termination of an attempted read, or various stall-while-write/erase operations are occurring.

The platform Flash controller can also terminate the current AHB access if hready_in is asserted before the end of the current bus access. While this circumstance should not occur, this does not result in an error condition being reported, as this behavior is initiated by the AHB. In this circumstance, the platform Flash controller control state machine completes any Flash array access in progress (without signaling the AHB) before handling a new access request.

17.2.12 Access pipelining

The platform Flash controller does not support access pipelining since this capability is not supported by the Flash array. As a result, the APC (Address Pipelining Control) field should typically be the same value as the RWSC (Read Wait State Control) field for best performance, that is, BK*n*_APC = BK*n*_RWSC. It cannot be less than the RWSC.

17.2.13 Flash error response operation

The Flash array may signal an error response by asserting bk*n*_fl_xfr_err to terminate a requested access with an error. This may occur due to an uncorrectable ECC error, or because of improper sequencing during program/erase operations. When an error response is received, the platform Flash controller does not update or validate a bank0 page read buffer nor the bank1 temporary holding register. An error response may be signaled on read or write operations. For more information on the specifics related to signaling of errors, including Flash ECC, refer to subsequent sections in this chapter. For additional information on the system registers that capture the faulting address, attributes, data and ECC information, see *Chapter 15: Error Correction Status Module (ECSM)*.

17.2.14 Bank0 page read buffers and prefetch operation

The logic associated with bank0 of the platform Flash controller contains four 128-bit page read buffers that hold data read from the Flash array. Each buffer operates independently, and is filled using a single array access. The buffers are used for both prefetch and normal demand fetches.

The organization of each page buffer is described as follows in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags.

```
struct {// bk0_page_buffer
   regaddr[23:4];// page address
   regvalid;// valid bit
   regrdata[127:0];// page read data
   regxfr_error;// transfer error indicator from Flash array
   regmulti_ecc_error;// multi-bit ECC error indicator from Flash array
   regsingle_ecc_error;// single-bit correctable ECC indicator from Flash
array
}  bk0_page_buffer[4];
```

For the general case, a page buffer is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the buffer, can be serviced in 0 AHB wait states as the stored read data is routed from the given page buffer back to the requesting bus master.

As noted in *Section 17.2.13: Flash error response operation* a page buffer is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the page buffer and validated. For additional comments on this topic, see *Section 17.2.14.4: Buffer invalidation*.

Prefetch triggering is controllable on a per-master and access-type basis. Bus masters may be enabled or disabled from triggering prefetches, and triggering may be further restricted based on whether a read access is for instruction or data. A read access to the platform Flash controller may trigger a prefetch to the next sequential page of array data on the first idle cycle following the request. The access address is incremented to the next-higher 16-byte boundary, and a Flash array prefetch is initiated if the data is not already resident in a page buffer. Prefetched data is always loaded into the least-recently-used buffer.

Buffers may be in one of six states, listed here in prioritized order:

1. Invalid—the buffer contains no valid data.
2. Used—the buffer contains valid data that has been provided to satisfy an AHB burst type read.
3. Valid—the buffer contains valid data that has been provided to satisfy an AHB single type read.
4. Prefetched—the buffer contains valid data that has been prefetched to satisfy a potential future AHB access.
5. Busy AHB—the buffer is currently being used to satisfy an AHB burst read.
6. Busy Fill—the buffer has been allocated to receive data from the Flash array, and the array access is still in progress.

Selection of a buffer to be loaded on a miss is based on the following replacement algorithm:

1. First, the buffers are examined to determine if there are any invalid buffers. If there are multiple invalid buffers, the one to be used is selected using a simple numeric priority, where buffer 0 is selected first, then buffer 1, etc.
2. If there are no invalid buffers, the least-recently-used buffer is selected for replacement.

Once the candidate page buffer has been selected, the Flash array is accessed and read data loaded into the buffer. If the buffer load was in response to a miss, the just-loaded buffer is immediately marked as most-recently-used. If the buffer load was in response to a speculative fetch to the next-sequential line address after a buffer hit, the recently-used status is not changed. Rather, it is marked as most-recently-used only after a subsequent buffer hit.

This policy maximizes performance based on reference patterns of Flash accesses and allows for prefetched data to remain valid when non-prefetch enabled bus masters are granted Flash access.

Several algorithms are available for prefetch control that trade off performance versus power. They are defined by the B*x*_P*y*_PFLM (prefetch limit) register field. More aggressive prefetching increases power slightly due to the number of wasted (discarded) prefetches, but may increase performance by lowering average read latency.

In order for prefetching to occur, a number of control bits must be enabled. Specifically, the global buffer enable (B*x*_P*y*_BFE) must be set, the prefetch limit (B*x*_P*y*_PFLM) must be non-zero and either instruction prefetching (B*x*_P*y*_IPFE) or data prefetching (B*x*_P*y*_DPFE) enabled. Refer to *Section 17.3.6: Registers description* for a description of these control fields.

17.2.14.1 Instruction/data prefetch triggering

Prefetch triggering may be enabled for instruction reads via the B*x*_P*y*_IPFE control field, while prefetching for data reads is enabled via the B*x*_P*y*_DPFE control field. Additionally, the B*x*_P*y*_PFLIM field must also be set to enable prefetching. Prefetches are never triggered by write cycles.

17.2.14.2 Per-master prefetch triggering

Prefetch triggering may be also controlled for individual bus masters. AHB accesses indicate the requesting master via the hmaster[3:0] inputs. Refer to *Section 17.3.7.7.3: Platform Flash Access Protection Register (PFAPR)* for details on these controls.

17.2.14.3 Buffer allocation

Allocation of the line read buffers is controlled via page buffer configuration (B*x*_P*y*_BCFG) field. This field defines the operating organization of the four page buffers. The buffers can be organized as a “pool” of available resources (with all four buffers in the pool) or with a fixed partition between buffers allocated to instruction or data accesses. For the fixed partition, two configurations are supported. In one configuration, buffers 0 and 1 are allocated for instruction fetches and buffers 2 and 3 for data accesses. In the second configuration, buffers 0, 1, and 2 are allocated for instruction fetches and buffer 3 reserved for data accesses.

17.2.14.4 Buffer invalidation

The page read buffers may be invalidated under hardware or software control.

Any falling edge transition of the array’s bk*n*_fl_done signal causes the page read buffers to be marked as invalid. This input is negated by the Flash array at the beginning of all program/erase operations as well as in certain other cases. Buffer invalidation occurs at the next AHB non-sequential access boundary, but does not affect a burst from a page read buffer in progress.

Software may invalidate the buffers by clearing the B*x*_P*y*_BFE bit, which also disables the buffers. Software may then re-assert the B*x*_P*y*_BFE bit to its previous state, and the buffers will have been invalidated.

One special case needing software invalidation relates to page buffer “hits” on Flash data that was tagged with a single-bit ECC event on the original array access. Recall that the page buffer structure includes an status bit signaling the array access detected and corrected a single-bit ECC error. On all subsequent buffer hits to this type of page data, a single-bit ECC event is signaled by the platform Flash controller. Depending on the specific hardware configuration, this reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

Finally, the buffers are invalidated by hardware on any non-sequential access with a non-zero value on haddr[28:24] to support wait state emulation.

17.2.15 Bank1 temporary holding register

Recall the bank1 logic within the Flash includes a single 128-bit data register, used for capturing read data. Since this bank does not support prefetching, the read data for the referenced address is bypassed directly back to the AHB data bus. The page is also loaded into the temporary data register and subsequent accesses to this page can hit from this register, if it is enabled (B1_Py_BFE).

The organization of the temporary holding register is described as follows, in a pseudo-code representation. The hardware structure includes the buffer address and valid bit, along with 128 bits of page read data and several error flags and is the same as an individual bank0 page buffer.

```
struct {// bk1_page_buffer
   regaddr[23:4];// page address
   regvalid;// valid bit
   regrdata[127:0];// page read data
   regxfr_error;// transfer error indicator from Flash array
   regmulti_ecc_error;// multi-bit ECC error indicator from Flash array
```

```
   regsingle_ecc_error;// single-bit correctable ECC indicator from Flash
array
}  bk1_page_buffer;
```

For the general case, a temporary holding register is written at the completion of an error-free Flash access and the valid bit asserted. Subsequent Flash accesses that “hit” the buffer, that is, the current access address matches the address stored in the temporary holding register, can be serviced in 0 AHB wait states as the stored read data is routed from the temporary register back to the requesting bus master.

The contents of the holding register are invalidated by the falling edge transition of bk1_fl_done and on any non-sequential access with a non-zero value on haddr[28:24] (to support wait state emulation) in the same manner as the bank0 page buffers. Additionally, the B1_Py_BFE register bit can be cleared by software to invalidate the contents of the holding register.

As noted in *Section 17.2.13: Flash error response operation* the temporary holding register is *not* marked as valid if the Flash array access terminated with any type of transfer error. However, the result is that Flash array accesses that are tagged with a single-bit correctable ECC event are loaded into the temporary holding register and validated. Accordingly, one special case needing software invalidation relates to holding register “hits” on Flash data that was tagged with a single-bit ECC event. Depending on the specific hardware configuration, the reporting of a single-bit ECC event may generate an ECC alert interrupt. In order to prevent repeated ECC alert interrupts, the page buffers need to be invalidated by software after the first notification of the single-bit ECC event.

The bank1 temporary holding register effectively operates like a single page buffer.

17.2.16 Read-While-Write functionality

The platform Flash controller supports various programmable responses for read accesses while the Flash is busy performing a write (program) or erase operation. For all situations, the platform Flash controller uses the state of the Flash array’s bk*n*_fl_done output to determine if it is busy performing some type of high-voltage operation, namely, if bk*n*_fl_done = 0, the array is busy.

Specifically, there are two 3-bit read-while-write (BK*n*_RWWC) control register fields that define the platform Flash controller’s response to these types of access sequences. There are five unique responses that are defined by the BK*n*_RWWC setting: one immediately reports an error on an attempted read, and four settings that support various stall-while-write capabilities. Consider the details of these settings.

- BK*n*_RWWC = 0b0*xx*
  - For this mode, any attempted Flash read to a busy array is immediately terminated with an AHB error response and the read is blocked in the controller and not seen by the Flash array.
- BK*n*_RWWC = 0b111
  - This defines the basic stall-while-write capability and represents the default reset setting. For this mode, the platform Flash controller module stalls any read reference until the Flash has completed its program/erase operation. If a read access arrives while the array is busy or if a falling-edge on bk*n*_fl_done occurs while a read is still in progress, the AHB data phase is stalled by negating hready_out and saving the address and attributes into holding registers. Once the array has completed its program/erase operation, the platform Flash controller uses the saved address and attribute information to create a pseudo address

phase cycle to “retry” the read reference and sends the registered information to the array as bk*n*_fl_rd_en is asserted. Once the retried address phase is complete, the read is processed normally and once the data is valid, it is forwarded to the AHB bus and hready_out negated to terminate the system bus transfer.

- BK*n*_RWWC = 0b110
  - This setting is similar to the basic stall-while-write capability provided when BK*n*_RWWC = 0b111 with the added ability to generate a notification interrupt if a read arrives while the array is busy with a program/erase operation. There are two notification interrupts, one for each bank.
- BK*n*_RWWC = 0b101
  - Again, this setting provides the basic stall-while-write capability with the added ability to terminate any program/erase operation if a read access is initiated. For this setting, the read request is captured and retried as described for the basic stall-while-write, plus the program/erase operation is terminated by the platform Flash controller’s assertion of the bkc_fl_abort signal. The bk*n*_fl_abort signal remains asserted until bk*n*_fl_done is driven high. For this setting, there are no notification interrupts generated.
- BK*n*_RWWC = 0b100
  - This setting provides the basic stall-while-write capability with the ability to terminate any program/erase operation if a read access is initiated plus the generation of a termination notification interrupt. For this setting, the read request is captured and retried as described for the basic stall-while-write, the program/erase operation is terminated by the platform Flash controller’s assertion of the bk*n*_fl_abort signal and a termination notification interrupt generated. There are two termination notification interrupts, one for each bank.

As detailed above, there are a total of four interrupt requests associated with the stall-while-write functionality. These interrupt requests are captured as part of ECSM’s Interrupt Register and logically summed together to form a single request to the interrupt controller.

Table 149. Platform Flash controller stall-while-write interrupts

MIR[*n*]	Interrupt description
ECSM.MIR[7]	Platform Flash bank0 termination notification, MIR[FB0AI]
ECSM.MIR[6]	Platform Flash bank0 stall notification, MIR[FB0SI]
ECSM.MIR[5]	Platform Flash bank1 termination notification, MIR[FB1AI]
ECSM.MIR[4]	Platform Flash bank1 stall notification, MIR[FB1S1]

For example timing diagrams of the stall-while-write and terminate-while-write operations, see *Figure 159* and *Figure 160* respectively.

17.2.17 Wait state emulation

Emulation of other memory array timings are supported by the platform Flash controller on read cycles to the Flash. This functionality may be useful to maintain the access timing for blocks of memory that were used to overlay Flash blocks for the purpose of system calibration or tuning during code development.

The platform Flash controller inserts additional wait states according to the values of haddr[28:24],where haddr represents the Flash address. When these inputs are non-zero, additional cycles are added to AHB read cycles. Write cycles are not affected. In addition, no page read buffer prefetches are initiated, and buffer hits are ignored.

Table 150 and *Table 151* show the relationship of haddr[28:24] to the number of additional primary wait states. These wait states are applied to the initial access of a burst fetch or to single-beat read accesses on the AHB system bus.

Note that the wait state specification consists of two components: haddr[28:26] and haddr[25:24] and effectively extends the Flash read by (8 × haddr[25:24] + haddr[28:26]) cycles.

Table 150. Additional wait state encoding

Memory address haddr[28:26]	Additional wait states
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Table 151 shows the relationship of haddr[25:24] to the number of additional wait states. These are applied in addition to those specified by haddr[28:26] and thus extend the total wait state specification capability.

Table 151. Extended additional wait state encoding

Memory address haddr[25:24]	Additional wait states (added to those specified by haddr[28:26])
00	0
01	8
10	16
11	24

17.2.18 Timing diagrams

Since the platform Flash controller is typically used in platform configurations with a cacheless core, the operation of the processor accesses to the platform memories, for example Flash and SRAM, plays a major role in the overall system performance. Given the core/platform pipeline structure, the platform's memory controllers (PFlash, PRAM) are

designed to provide a 0 wait state data phase response to maximize processor performance. The following diagrams illustrate operation of various cycle types and responses referenced earlier in this chapter including stall-while-read (*Figure 159*) and terminate-while-read (*Figure 160*) diagrams.

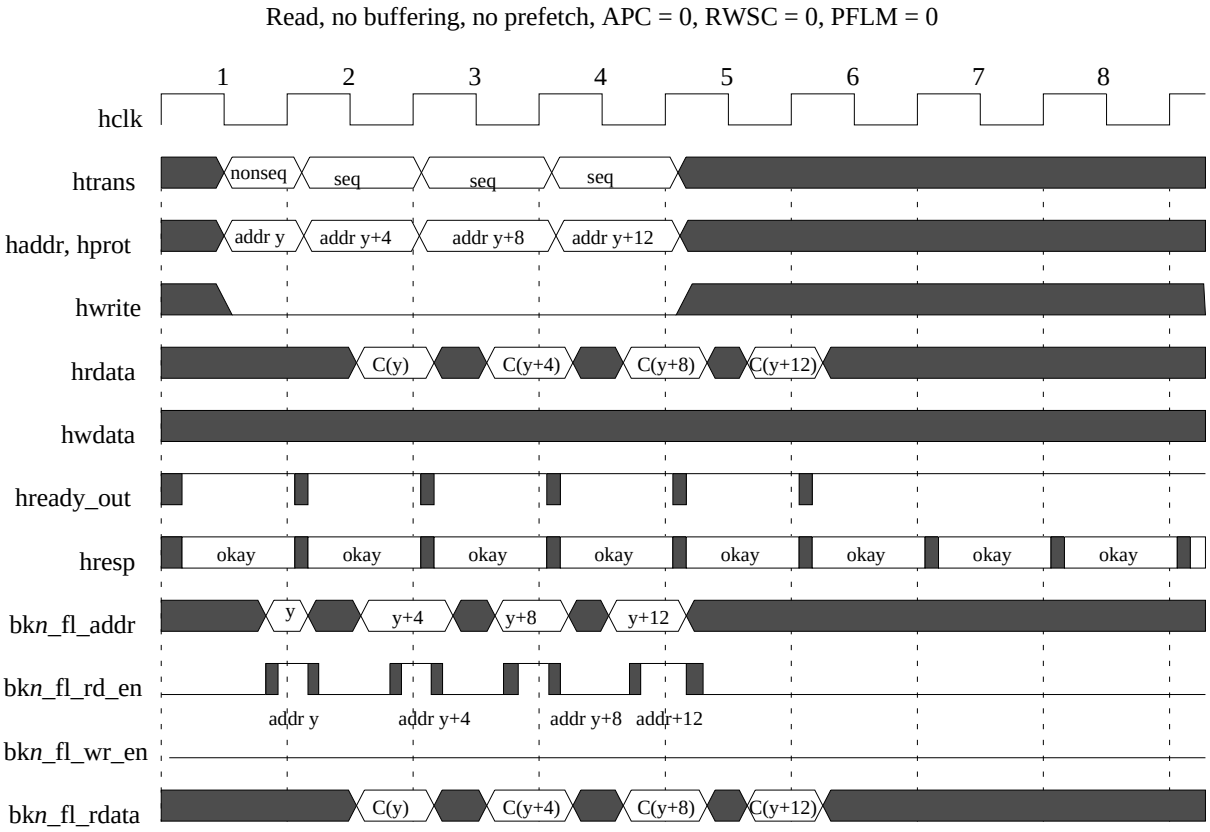


Figure 155. 1-cycle access, no buffering, no prefetch

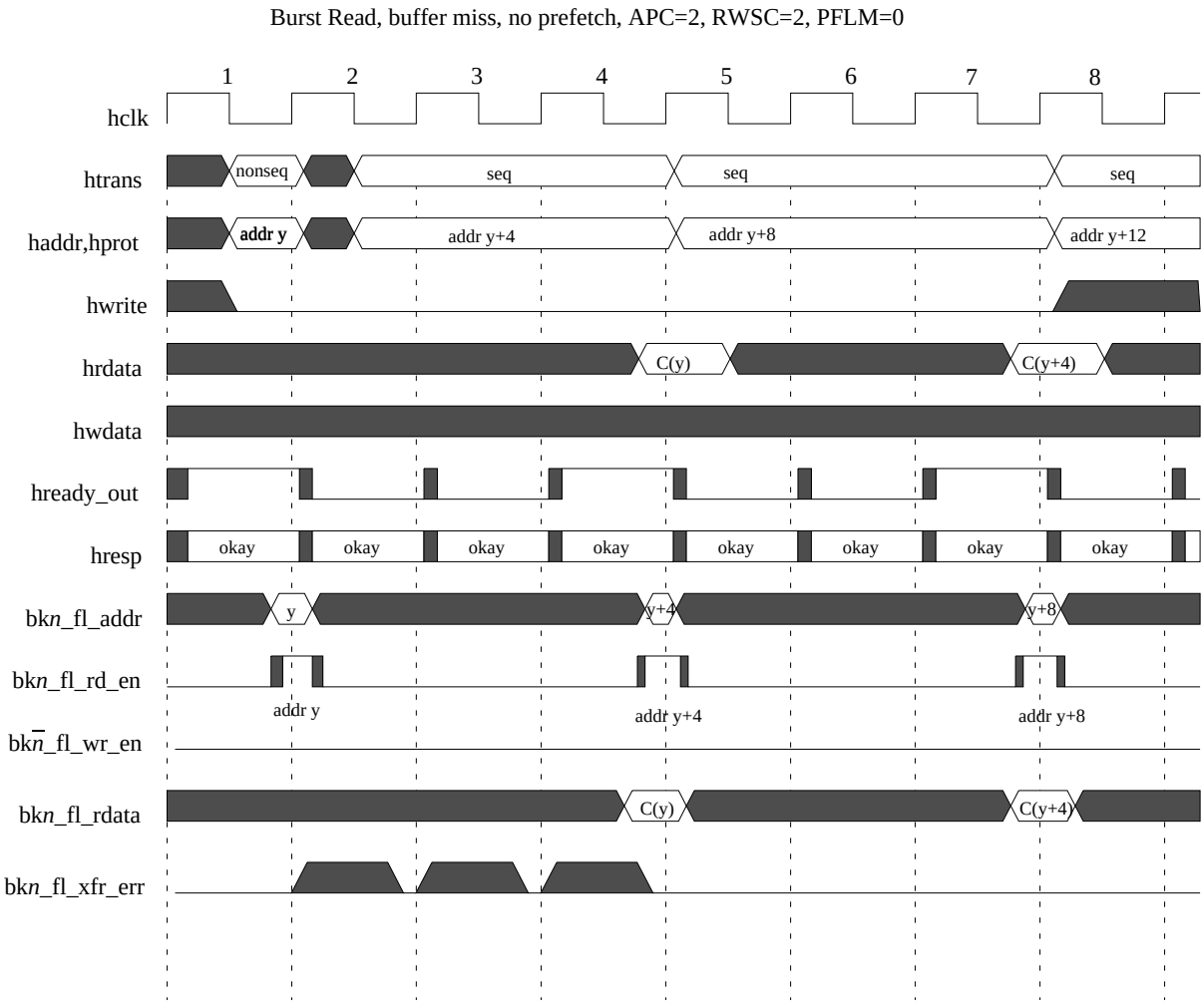


Figure 156. 3-cycle access, no prefetch, buffering disabled

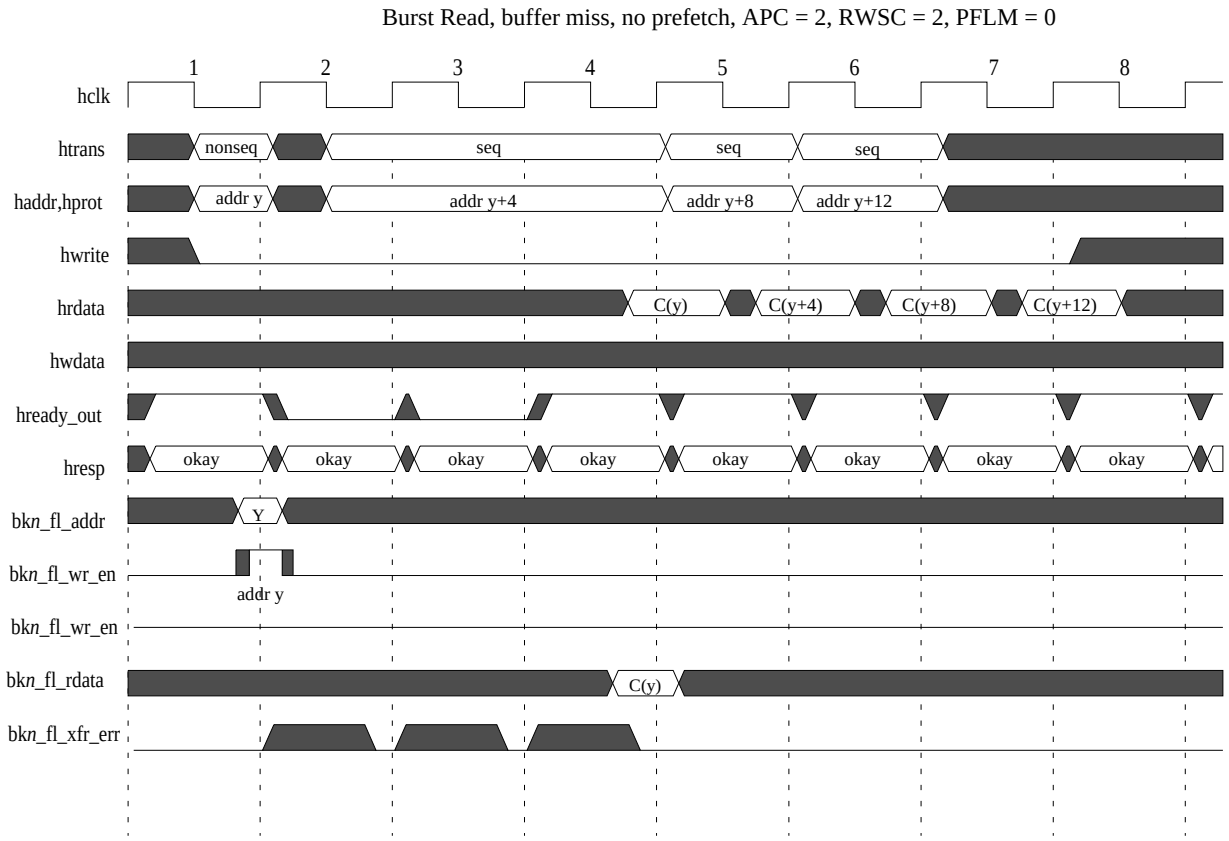


Figure 157. 3-cycle access, no prefetch, buffering enabled

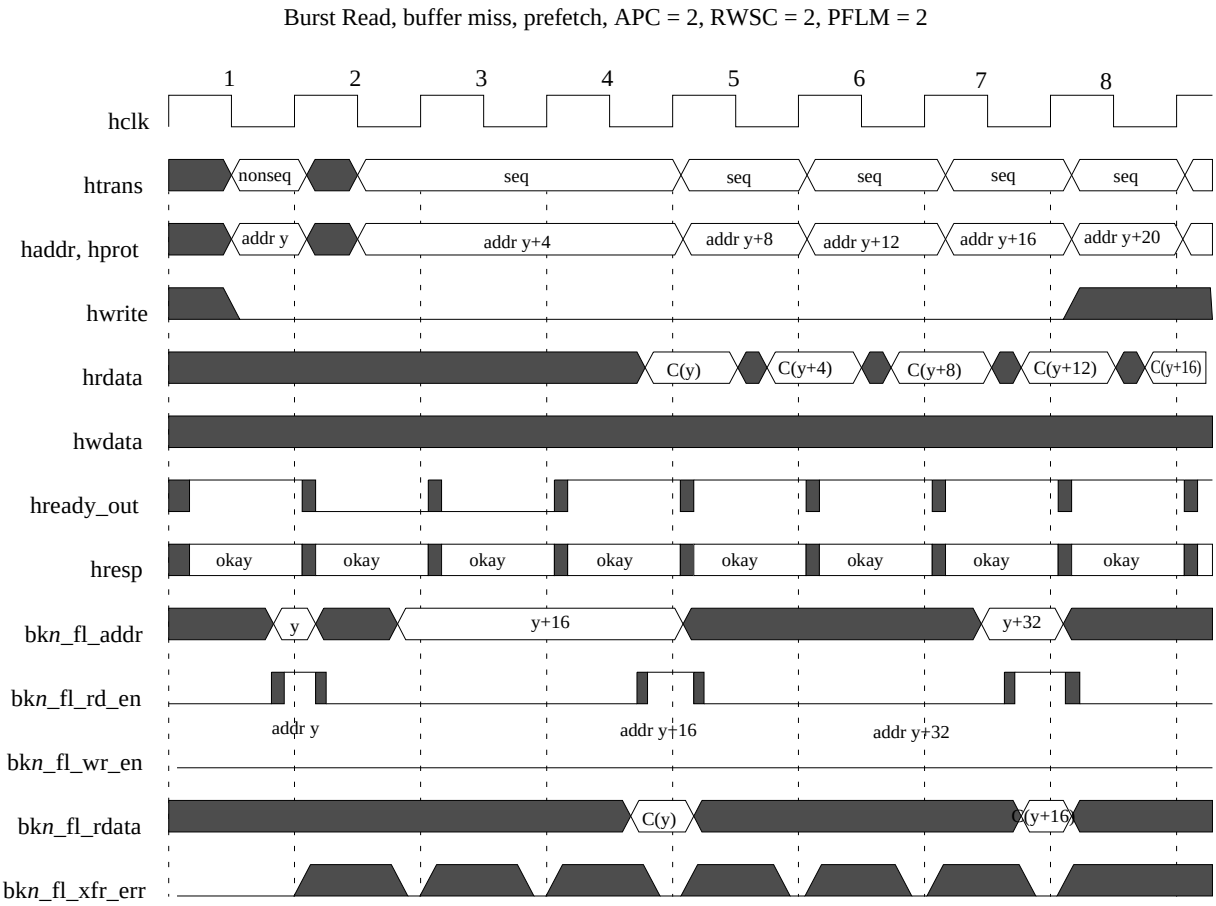


Figure 158. 3-cycle access, prefetch and buffering enabled

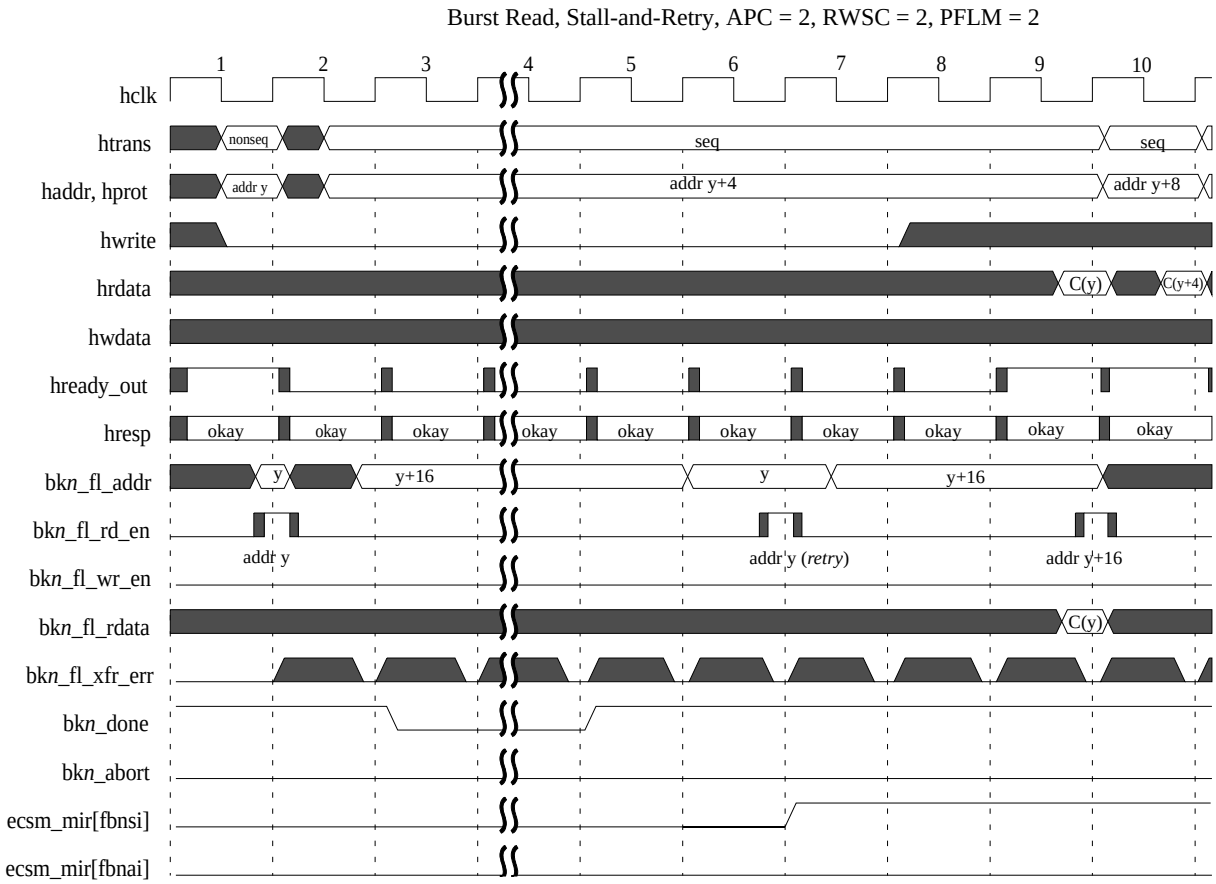


Figure 159. 3-cycle access, stall-and-retry with BK*n*_RWWC = 11*x*

As shown in *Figure 159*, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Eventually, this array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y + 4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal a stall notification interrupt (if BK*n*_RWWC = 110). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*SI] (Flash bank *n* stall interrupt).

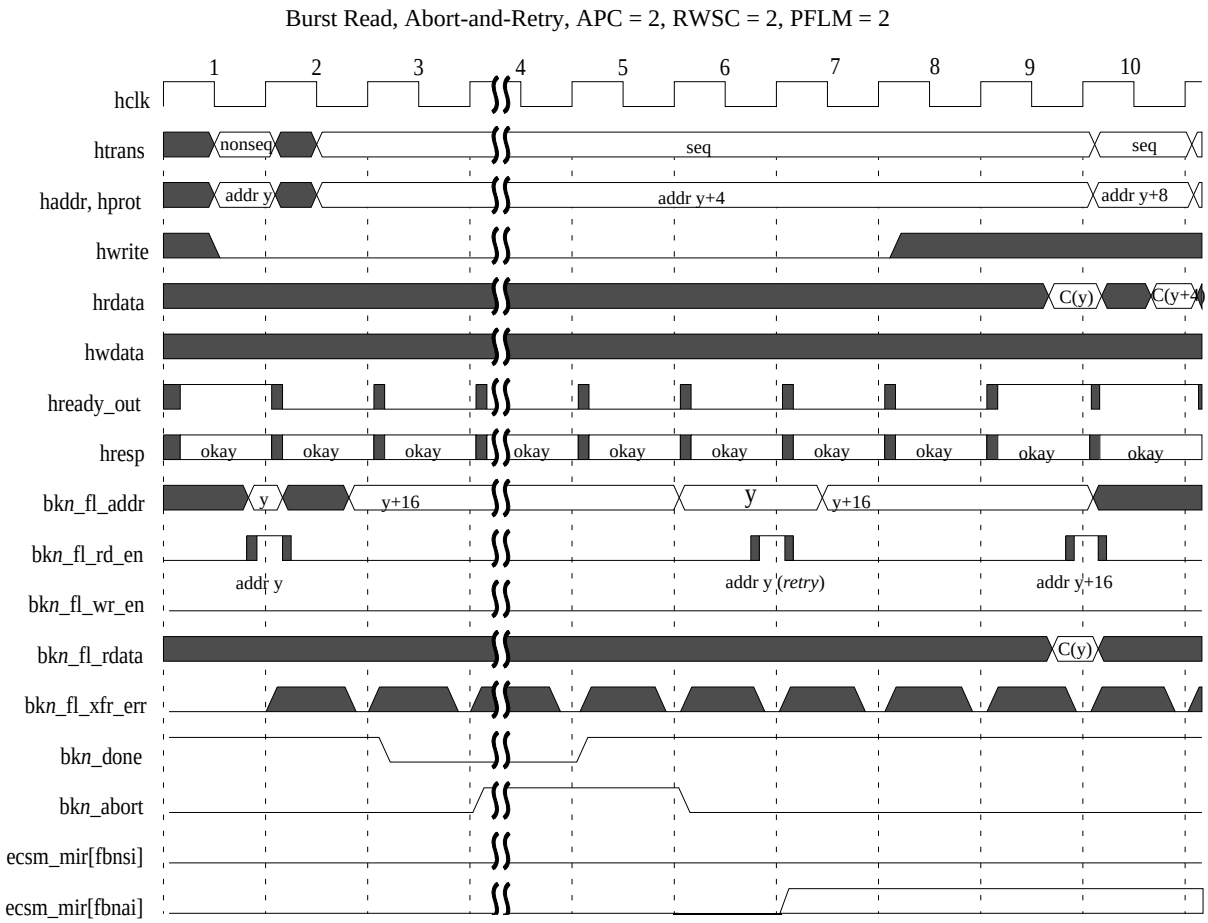


Figure 160. 3-cycle access, terminate-and-retry with BK*n*_RWWC = 10*x*

Figure 160 shows the terminate-while-write timing diagram. In this example, the 3-cycle access to address y is interrupted when an operation causes the bk*n*_done signal to be negated, signaling that the array bank is busy with a high-voltage program or erase event. Based on the setting of BK*n*_RWWC, once the bk*n*_done signal is detected as negated, the platform Flash controller asserts bk*n*_abort, which forces the Flash array to cancel the high-voltage program or erase event. The array operation completes (at the end of cycle 4) and bk*n*_done returns to a logical 1. It should be noted that the time spent in cycle 4 for *Figure 160* is considerably less than the time in the same cycle in *Figure 159* (because of the terminate operation). In cycle 6, the platform Flash controller module retries the read to address y that was interrupted by the negation of bk*n*_done in cycle 3. Note that throughout cycles 2–9, the AHB bus pipeline is stalled with a read to address y in the AHB data phase and a read to address y+4 in the address phase. Depending on the state of the least-significant-bit of the BK*n*_RWWC control field, the hardware may also signal an termination notification interrupt (if BK*n*_RWWC = 100). The stall notification interrupt is shown as the optional assertion of ECSM's MIR[FB*n*AI] (Flash bank *n* termination interrupt).

17.3 Flash memory

17.3.1 Introduction

The Flash module provides electrically programmable and erasable non-volatile memory (NVM), which may be used for instruction and/or data storage.

The Flash module is arranged as two functional units: the Flash core and the memory interface.

The Flash core is composed of arrayed non-volatile storage elements, sense amplifiers, row decoders, column decoders and charge pumps. The arrayed storage elements in the Flash core are sub-divided into physically separate units referred to as blocks (or sectors).

The memory interface contains the registers and logic which control the operation of the Flash core. The memory interface is also the interface between the Flash module and a bus interface unit (BIU), and may contain the ECC logic and redundancy logic. The BIU connects the Flash module to a system bus.

The SPC560P44Lx, SPC560P50Lx provides two Flash modules: one 512 KB code Flash module, and one 64 KB data module.

17.3.2 Main features

- High read parallelism (128 bits)
- Error Correction Code (SEC-DED) to enhance data retention
- Double word program (64 bits)
- Sector erase
- Single bank architecture
  - Read-While-Modify not available within an individual module
  - Read-While-Modify can be performed between the two Flash modules
- Erase suspend available (program suspend not available)
- Software programmable program/erase protection to avoid unwanted writes
- Censored mode against piracy
- Shadow Sector available on code Flash module
- One Time Programmable (OTP) area in TestFlash block

17.3.3 Block diagram

17.3.3.1 Data Flash

The data Flash module contains one module, composed of a Single Bank: Bank 0, normally used for code storage. No Read-While-Modify operations are possible.

The Modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a user registers interface.

The read data bus is 32 bits wide, while the data Flash registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 161* shows the data Flash module structure.

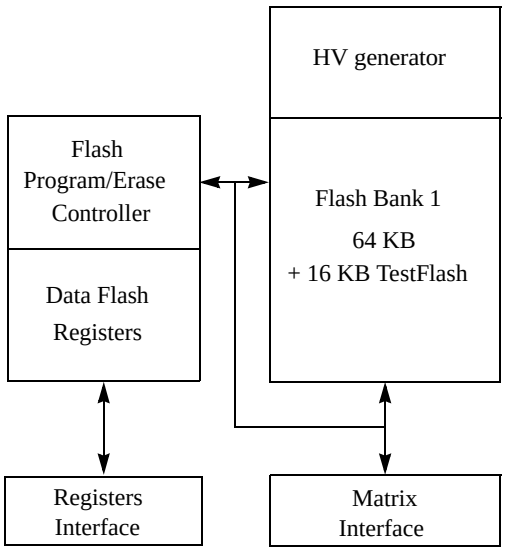


Figure 161. Data Flash module structure

17.3.3.2 Code Flash

The code Flash module contains the matrix modules normally used for Code storage. No Read-While-Modify operations are possible.

The Modify operations are managed by an embedded Flash Program/Erase Controller (FPEC). Commands to the FPEC are given through a User Registers Interface.

The read data bus is 128 bits wide, while the Flash registers are on a separate 32-bit wide bus.

The high voltages needed for Program/Erase operations are internally generated. *Figure 162* shows the code Flash module structure.

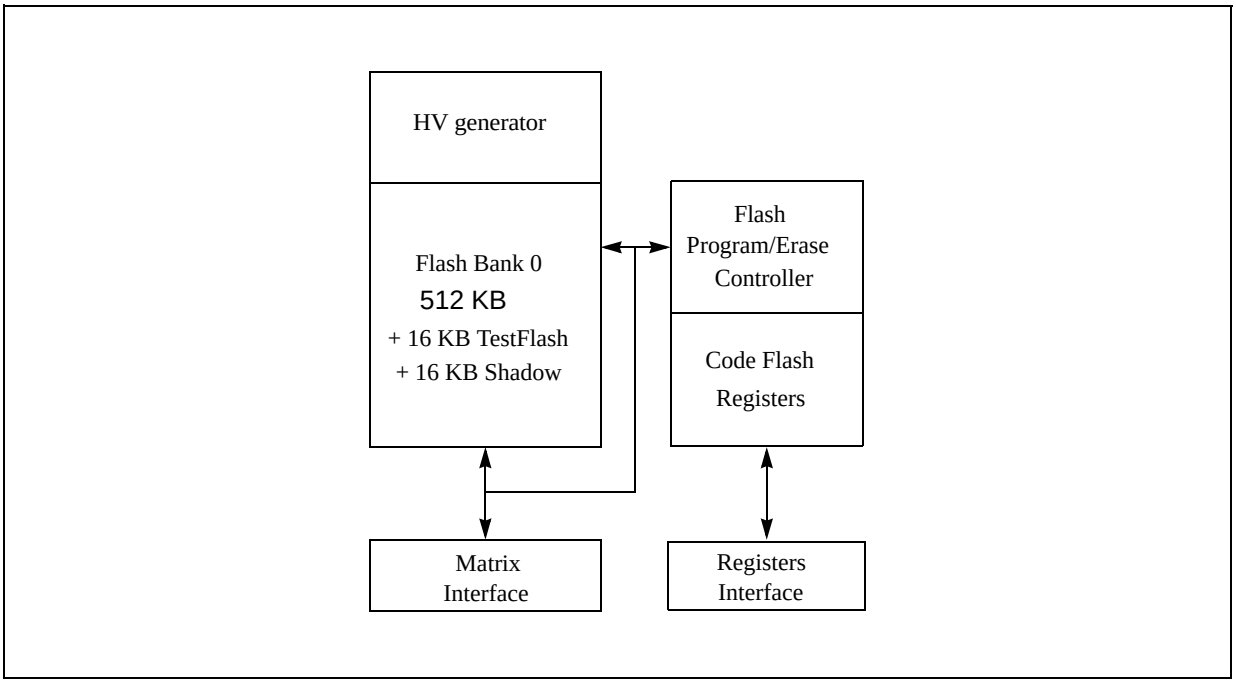


Figure 162. Code Flash module structure

17.3.4 Functional description

17.3.4.1 Macrocell structure

The Flash macrocell provides high density non-volatile memories with high-speed read access.

The Flash module is addressable by word (32 bits) or double-word (64 bits) for programming, and by page (128 bits) for reads. Reads done to the Flash always return 128 bits, although read page buffering may be done in the platform BIU.

Each read of the Flash module retrieves a page, or 4 consecutive words (128 bits) of information. The address for each word retrieved within a page differ from the other addresses in the page only by address bits (3:2).

The Flash page read architecture supports both cache and burst mode at the BIU level for high-speed read application.

The Flash module supports fault tolerance through Error Correction Code (ECC) and/or error detection. The ECC implemented within the Flash module will correct single bit failures and detect double bit failures.

The Flash module uses an embedded hardware algorithm implemented in the memory interface to program and erase the Flash core.

Control logic that works with the software block enables, and software lock mechanisms, is included in the embedded hardware algorithm to guard against accidental program/erase.

The hardware algorithm perform the steps necessary to ensure that the storage elements are programmed and erased with sufficient margin to guarantee data integrity and reliability.

A programmed bit in the Flash module reads as logic level 0 (or low).

An erased bit in the Flash module reads as logic level 1 (or high).

Program and erase of the Flash module requires multiple system clock cycles to complete.

The erase sequence may be suspended.

The program and erase sequences may be terminated.

17.3.4.2 Flash module sectorization

The code Flash module supports 512 KB of user memory, plus 16 KB of test memory (a portion of which is one-time programmable by the user). An extra 16 KB sector is available as Shadow space usable for user option bits or censorship.

The code Flash and data modules are each composed of a single bank: Bank 0 (code Flash) and Bank 1 (data Flash). Read-While-Modify within a module is not supported, but can be performed by reading from one module while writing to another.

The code Flash Bank 0 is divided in 10 sectors including a reserved sector named TestFlash, in which One Time Programmable (OTP) user data are stored, and a Shadow Sector in which user erasable configuration values can be stored (see *Table 152*).

The data Flash Bank 1 is divided in five sectors including a reserved sector named TestFlash (see *Table 153*).

Table 152. 544 KB code Flash module sectorization

Bank	Sector	Addresses	Size	Address space
B0	B0F0	0x0000_0000–0x0000_7FFF	32 KB	Low Address Space
B0	B0F1	0x0000_8000–0x0000_BFFF	16 KB	Low Address Space
B0	B0F2	0x0000_C000–0x0000_FFFF	16 KB	Low Address Space
B0	B0F3	0x0001_0000–0x0001_7FFF	32 KB	Low Address Space
B0	B0F4	0x0001_8000–0x0001_FFFF	32 KB	Low Address Space
B0	B0F5	0x0002_0000–0x0003_FFFF	128 KB	Low Address Space
B0	B0F6	0x0004_0000–0x0005_FFFF	128 KB	Mid Address Space
B0	B0F7(1)	0x0006_0000–0x0007_FFFF	128 KB	Mid Address Space
B0	Reserved	0x0008_0000–0x001F_FFFF	1536 KB	High Address Space
B0	B0SH	0x0020_0000–0x0020_3FFF	16 KB	Shadow Address Space
B0	Reserved	0x0020_4000–0x003F_FFFF	2032 KB	Shadow Address Space
B0	B0TF	0x0040_0000–0x0040_3FFF	16 KB	Test Address Space
B0	Reserved	0x0040_4000–0x007F_FFFF	4080 KB	Test Address Space

1. Not available on SPC560P44

Table 153. 64 KB data Flash module sectorization

Bank	Sector	Addresses	Size (KB)	Address space
B1	B1F0	0x0080_0000 to 0x0080_3FFF	16	Low Address Space

Table 153. 64 KB data Flash module sectorization(Continued)

Bank	Sector	Addresses	Size (KB)	Address space
B1	B1F1	0x0080_4000 to 0x0080_7FFF	16	Low Address Space
B1	B1F2	0x0080_8000 to 0x0080_BFFF	16	Low Address Space
B1	B1F3	0x0080_C000 to 0x0080_FFFF	16	Low Address Space
B1	Reserved	0x0081_0000 to 0x00BF_FFFF	4032	Reserved
B1	B1TF	0x00C0_0000 to 0x00C0_3FFF	16	Test Address Space
B1	Reserved	0x00C0_4000 to 0x00FF_FFFF	4080	Reserved

Each Flash module is divided into blocks to implement independent program/erase protection. A software mechanism is provided to independently lock/unlock each block in address space against program and erase.

17.3.4.2.1 TestFlash block

The TestFlash block exists outside the normal address space and is programmed and read independently of the other blocks. The independent TestFlash block is included also to support systems that require non-volatile memory for security and/or to store system initialization information.

A section of the TestFlash is reserved to store the non-volatile information related to redundancy, configuration, and protection.

ECC is also applied to TestFlash. The usage of reserved TestFlash sectors is detailed in *Table 154*.

Table 154. TestFlash structure

Name	Description	Addresses		Size (bytes)
		Code TestFlash	Data TestFlash	
—	Reserved	0x0040_0000–0x0040_1FFF	0x00C0_0000– 0x00C0_1FFF	8192
—	Reserved	0x0040_2000–0x0040_3CFF	0x00C0_2000– 0x00C0_3CFF	7424
—	User Reserved	0x0040_3D00–0x0040_3DE7	0x00C0_3D00–0x00C0_3DE7	232
NVLML	Non-volatile Low/Mid address space block Locking register	0x0040_3DE8–0x0040_3DEF	0x00C0_3DE8–0x00C0_3DEF	8
—	User Reserved	0x0040_3DF0–0x0040_3DF7	0x00C0_3DF0–0x00C0_3DF7	8
NVSLL	Non-volatile Secondary Low/mid add space block Lock register	0x0040_3DF8–0x0040_3DFF	0x00C0_3DF8–0x00C0_3DFF	8
—	User Reserved	0x0040_3E00–0x0040_3EFF	0x00C0_3E00–0x00C0_3EFF	256
—	Reserved	0x0040_3F00–0x0040_3FFF	0x00C0_3F00–0x00C0_3FFF	256

Erase of the TestFlash block is always locked.

TestFlash block programming restrictions, in terms of how ECC is calculated, are similar to array programming restrictions. Only one program is allowed per 64-bit ECC segment.

Locations of the Code TestFlash block marked as Reserved cannot be programmed by the user application.

The first 8 KB of the Data TestFlash block may be used for user-defined functions (possibly to store serial numbers, other configuration words, or factory process codes). Locations of the Data TestFlash block marked as reserved cannot be programmed by the user application.

17.3.4.2.1.1 Unique Serial Number

For tracking purposes by the customer, a Unique Serial Number is included in the Test Block to allow identification of a particular device. The Unique Serial Number is defined to be 128 bits and is based on the manufacturing lot identifier and additional information specific to each device in a manufacturing lot.

Table 155. Unique Serial Number

Address	Use	Default value	Number of word used (32-bit)
0x403C10	Device Unique Serial Number	x(1)	4 words (128 bits)

1. This value is different on every device.

17.3.4.2.2 Shadow block

A Shadow block is present in each Code flash module, but not in the Data flash module. The Shadow block can be enabled by the BIU.

When the Shadow space is enabled, all the operations are mapped to the Shadow block.

User mode program and erase of the shadow block are enabled only when MCR[PEAS] is set.

The Shadow block may be locked/unlocked against program or erase by using the LML[TSLK] and SLL[STSLK] bitfields.

Program of the Shadow block has similar restriction as the array in terms of how ECC is calculated. Only one program is allowed per 64-bit ECC segment between erases.

Erase of the Shadow block is done similarly as an sectors erase.

The Shadow block contains specified data that are needed for user features.

The user area of Shadow block may be used for user-defined functions (possibly to store boot code, other configuration words, or factory process codes).

The usage of the Shadow sector is detailed in *Table 156*.

Table 156. Shadow sector structure

Name	Description	Addresses	Size (bytes)
—	User Area	0x0020_0000–0x0020_3DCF	15824
—	Reserved	0x0020_3DD0–0x0020_3DD7	8
NVPWD0–1	Non-volatile private censorship password 0–1 registers	0x0020_3DD8–0x0020_3DDF	8
NVSCI0–1	Non-volatile system censorship information 0–1 registers	0x0020_3DE0–0x0020_3DE7	8
—	Reserved	0x0020_3DE8–0x0020_3DFF	24
NVBIU2–3	Non-volatile bus interface unit 2–3 registers	0x0020_3E00–0x0020_3E0F	16
—	Reserved	0x0020_3E10–0x0020_3E17	8
NVUSRO	Non-volatile user options register	0x0020_3E18–0x0020_3E1F	8
—	Reserved	0x0020_3E20–0x0020_3FFF	480

17.3.5 Operating modes

The following operating modes are available in the Flash module:

- Reset
- User mode
- Low-power mode
- Power-down mode

17.3.5.1 Reset

A reset is the highest priority operation for the Flash module and terminates all other operations.

The Flash module uses reset to initialize registers and status bits to their default reset values.

If the Flash module is executing a program or erase operation (MCR[PGM] = 1 or MCR[ERS] = 1) and a reset is issued, the operation is suddenly terminated and the module disables the high voltage logic without damage to the high voltage circuits. Reset terminates all operations and forces the Flash module into User mode ready to receive accesses.

Reset and power-down must not be used as a systematic way to terminate a program or erase operation.

After reset is deasserted, read register access may be done, although it should be noted that registers that require updating from shadow information or other inputs may not read updated values until MCR[DONE] transitions. MCR[DONE] may be polled to determine if the Flash module has transitioned out of reset. Notice that the registers cannot be written until MCR[DONE] is high.

17.3.5.2 User mode

In User mode, the Flash module may be read, written (register writes and interlock writes), programmed, or erased.

The default state of the Flash module is the read state.

The main, shadow, and test address space can be read only in the read state.

The Flash registers are always available for reads. When the module is in power-down mode, most (but not all) registers are available for reads. The exceptions are documented.

The Flash module enters the read state on reset.

The module is in the read state under two sets of conditions:

- The read state is active when the module is enabled (User mode read).
- The read state is active when MCR[ERS] and MCR[ESUS] are set and MCR[PGM] is cleared (Erase Suspend).

No Read-While-Modify is available within an individual module, although one module can be read while the other is being written or otherwise modified.

Flash core reads return 128 bits (1 page = 2 double words).

Register reads return 32 bits (1 word).

Flash core reads are done through the BIU.

Register reads to unmapped register address space return all 0s.

Register writes to unmapped register address space have no effect.

Array reads attempted to invalid locations will result in indeterminate data. Invalid locations occur when addressing is done to blocks that do not exist in non 2^n array sizes.

Interlock writes attempted to invalid locations, will result in an interlock occurring, but attempts to program these blocks will not occur since they are forced to be locked. Erase will occur to selected and unlocked blocks even if the interlock write is to an invalid location.

Simultaneous read cycles on the code Flash block and read/write cycles on the data Flash block are possible. However, simultaneous read/write accesses within a single block are not permitted.

Chip Select, Write Enable, addresses, and data input of registers are not internally latched and must be kept stable by the CPU for all the read/write access that lasts two clock cycles.

17.3.5.3 Low-power mode

The Low-power mode turns off most of the DC current sources within the Flash module.

The module (Flash core and registers) is not accessible for read or write operations once it has entered Low-power mode.

Wake-up time from Low-power mode is faster than wake-up time from Power-down mode.

The user may not read some registers (UMISR0–4, UT1–2 and part of UT0) until the Low-power mode is exited. Write access is locked on all the registers in Low-power mode.

When exiting from Low-power mode, the Flash module returns to its previous state in all cases, unless it was in the process of executing an erase high voltage operation at the time of entering Low-power mode.

If the Flash module is put into Low-power mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits from Low-power mode by clearing MCR[ESUS]. MCR[EHV] must be set to resume the erase operation.

If the Flash module is put in Low-power mode during a program operation, the operation is completed in all cases. Low-power mode is entered only after the programming ends.

Power-down mode cannot be entered when Low-power mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

17.3.5.4 Power-down mode

The Power-down mode allows turning off all Flash DC current sources so that power dissipation is limited only to leakage in this mode.

In Power-down mode, no reads from or writes to the Flash are possible.

When enabled, the Flash module returns to its previous state in all cases, unless in the process of executing an erase high voltage operation at the time of entering Power-down mode. If the Flash module is put into Power-down mode during an erase operation, the MCR[ESUS] bit is set to 1. The user may resume the erase operation when the Flash module exits Power-down mode by clearing the MCR[ESUS] bit. MCR[EHV] must be set to resume the erase operation.

If the Flash module is placed in Power-down mode during a program operation, the operation will be completed in any case and the Power-down mode is entered only after the programming is completed.

If the Flash module is put in Power-down mode and the vector table remains mapped in the Flash address space, the user must observe than the Flash module will require a longer interrupt response time. This should be accomplished by adding several wait states.

Low-power mode cannot be entered when Power-down mode is active. The module must first be set to User mode (or reset) when cycling between Low-power mode and Power-down mode.

17.3.6 Registers description

The Flash user registers mapping is shown in *Table 157*. Except as noted, registers and offsets are identical for the code Flash and data Flash blocks.

Table 157. Flash registers

Offset from xxxx_BASE (0xFFFE_C000)	Register	Location
0x0000	Module Configuration Register (MCR)	*on page 366*
0x0004	Low/mid Address Space Block Locking Register (LML)	*on page 371*
0x0008	Reserved	
0x000C	Secondary Low/mid Address Space Block Lock Register (SLL)	*on page 374*
0x0010	Low/mid Address Space Block Select Register (LMS)	*on page 377*
0x0014	Reserved	
0x0018	Address Register (ADR)	*on page 378*
0x001C	Platform Flash Configuration Register 0 (PFCR0)(1)	*on page 380*
0x0020	Platform Flash Configuration Register 1 (PFCR1)1	*on page 383*

Table 157. Flash registers(Continued)

Offset from xxxx_BASE (0xFFFE_C000)	Register	Location
0x0024	Platform Flash Access Protection Register (PFAPR)[1]	*on page 385*
0x0028	Reserved	
0x003C	User Test Register 0 (UT0)	*on page 386*
0x0040	User Test Register 1 (UT1)	*on page 388*
0x0044	User Test Register 2 (UT2)	*on page 388*
0x0048	User Multiple Input Signature Register 0 (UMISR0)	*on page 389*
0x004C	User Multiple Input Signature Register 1 (UMISR1)	*on page 390*
0x0050	User Multiple Input Signature Register 2 (UMISR2)	*on page 390*
0x0054	User Multiple Input Signature Register 3 (UMISR3)	*on page 391*
0x0058	User Multiple Input Signature Register 4 (UMISR4)	*on page 392*
0x005C–0x3FFF	Reserved	

1. This register is not implemented on the data Flash block.

17.3.7 Register map

Table 158. Flash 512 KB bank0 register map

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	Reserved																
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL1	MSL0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	Reserved																

Table 158. Flash 512 KB bank0 register map(Continued)

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C	PFCR0	BI031	BI030	BI029	BI028	BI027	BI026	BI025	BI024	BI023	BI022	BI021	BI020	BI019	BI018	BI017	BI016
		BI015	BI014	BI013	BI012	BI011	BI010	BI009	BI008	BI007	BI006	BI005	BI004	BI003	BI002	BI001	BI000
0x20	PFCR1	BI131	BI130	BI129	BI128	BI127	BI126	BI125	BI124	BI123	BI122	BI121	BI120	BI119	BI118	BI117	BI116
		BI115	BI114	BI113	BI112	BI111	BI110	BI109	BI108	BI107	BI106	BI105	BI104	BI103	BI102	BI101	BI100
0x24	PFAPR	BI231	BI230	BI229	BI228	BI227	BI226	BI225	BI224	BI223	BI222	BI221	BI220	BI219	BI218	BI217	BI216
		BI215	BI214	BI213	BI212	BI211	BI210	BI209	BI208	BI207	BI206	BI205	BI204	BI203	BI202	BI201	BI200
0x28	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00
0x44	UT2	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
		DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50	UMISR2	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
		MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064
0x54	UMISR3	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
		MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096

Table 158. Flash 512 KB bank0 register map(Continued)

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x58	UMISR4	MS159	MS158	MS157	MS156	MS155	MS154	MS153	MS152	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
		MS143	MS142	MS141	MS140	MS139	MS138	MS137	MS136	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128

Table 159. Flash 64 KB bank1 register map

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x00	MCR	EDC	0	0	0	0	SIZE2	SIZE1	SIZE0	0	LAS2	LAS1	LAS0	0	0	0	MAS
		EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
0x04	LML	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
		LLK15	LLK14	LLK13	LLK12	LLK11	LLK10	LLK9	LLK8	LLK7	LLK6	LLK5	LLK4	LLK3	LLK2	LLK1	LLK0
0x08	Reserved																
0x0C	SLL	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
		SLK15	SLK14	SLK13	SLK12	SLK11	SLK10	SLK9	SLK8	SLK7	SLK6	SLK5	SLK4	SLK3	SLK2	SLK1	SLK0
0x10	LMS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL1	MSL0
		LSL15	LSL14	LSL13	LSL12	LSL11	LSL10	LSL9	LSL8	LSL7	LSL6	LSL5	LSL4	LSL3	LSL2	LSL1	LSL0
0x14	Reserved																
0x18	ADR	0	0	0	0	0	0	0	0	0	AD22	AD21	AD20	AD19	AD18	AD17	AD16
		AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	0	0	0
0x1C–0x3B	Reserved																
0x3C	UT0	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
		0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
0x40	UT1	DAI31	DAI30	DAI29	DAI28	DAI27	DAI26	DAI25	DAI24	DAI23	DAI22	DAI21	DAI20	DAI19	DAI18	DAI17	DAI16
		DAI15	DAI14	DAI13	DAI12	DAI11	DAI10	DAI09	DAI08	DAI07	DAI06	DAI05	DAI04	DAI03	DAI02	DAI01	DAI00

Table 159. Flash 64 KB bank1 register map(Continued)

Address offset	Register name	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x44	UT2	DAI63	DAI62	DAI61	DAI60	DAI59	DAI58	DAI57	DAI56	DAI55	DAI54	DAI53	DAI52	DAI51	DAI50	DAI49	DAI48
		DAI47	DAI46	DAI45	DAI44	DAI43	DAI42	DAI41	DAI40	DAI39	DAI38	DAI37	DAI36	DAI35	DAI34	DAI33	DAI32
0x48	UMISR0	MS031	MS030	MS029	MS028	MS027	MS026	MS025	MS024	MS023	MS022	MS021	MS020	MS019	MS018	MS017	MS016
		MS015	MS014	MS013	MS012	MS011	MS010	MS009	MS008	MS007	MS006	MS005	MS004	MS003	MS002	MS001	MS000
0x4C	UMISR1	MS063	MS062	MS061	MS060	MS059	MS058	MS057	MS056	MS055	MS054	MS053	MS052	MS051	MS050	MS049	MS048
		MS047	MS046	MS045	MS044	MS043	MS042	MS041	MS040	MS039	MS038	MS037	MS036	MS035	MS034	MS033	MS032
0x50	UMISR2	MS095	MS094	MS093	MS092	MS091	MS090	MS089	MS088	MS087	MS086	MS085	MS084	MS083	MS082	MS081	MS080
		MS079	MS078	MS077	MS076	MS075	MS074	MS073	MS072	MS071	MS070	MS069	MS068	MS067	MS066	MS065	MS064
0x54	UMISR3	MS127	MS126	MS125	MS124	MS123	MS122	MS121	MS120	MS119	MS118	MS117	MS116	MS115	MS114	MS113	MS112
		MS111	MS110	MS109	MS108	MS107	MS106	MS105	MS104	MS103	MS102	MS101	MS100	MS099	MS098	MS097	MS096
0x58	UMISR4	MS159	MS158	MS157	MS156	MS155	MS154	MS153	MS152	MS151	MS150	MS149	MS148	MS147	MS146	MS145	MS144
		MS143	MS142	MS141	MS140	MS139	MS138	MS137	MS136	MS135	MS134	MS133	MS132	MS131	MS130	MS129	MS128

In the following sections, some non-volatile registers are described. Please notice that such entities are not Flip-Flops, but locations of TestFlash or Shadow sectors with a special meaning.

During the Flash initialization phase, the FPEC reads these non-volatile registers and updates their related volatile registers. When the FPEC detects ECC double errors in these special locations, it behaves in the following way:

- In case of a failing system locations (configurations, device options, redundancy, EmbAlgo firmware), the initialization phase is interrupted and a Fatal Error is flagged.
- In case of failing user locations (protections, censorship, BIU, ...), the volatile registers are filled with all 1s and the Flash initialization ends by clearing the MCR[PEG] bit.

17.3.7.1 Module Configuration Register (MCR)

The Module Configuration Register enables and monitors all the modify operations of each Flash module. Identical MCRs are provided in the code Flash and the data Flash blocks.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EDC	0	0	0	0	SIZE 2	SIZE 1	SIZE 0	0	LAS2	LAS1	LAS0	0	0	0	MAS
W	r1c															
Reset	0	0	0	0	0	—(1)	—(1)	—(1)	0	—(1)	1	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	0	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	r1c	r1c														
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

Figure 163. Module Configuration Register (MCR)

1. The value for this bit is different between the code and data Flash modules. See the bitfield description.

Table 160. MCR field descriptions

Field	Description
EDC 0	ECC Data Correction EDC provides information on previous reads. If a ECC Single Error detection and correction occurs, the EDC bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Double Error detection, this bit is not set. If EDC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EDC) were not corrected through ECC. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 An ECC Single Error occurred and was corrected during a previous read.
1:4	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
SIZE[2:0] 5:7	Array space SIZE 2–0 The value of SIZE field depends on the size of the Flash module: 000 128 KB 001 256 KB 010 512 KB (the value for the SPC560P44Lx, SPC560P50Lx device in the code Flash module) 011 Reserved (1024 KB) 100 Reserved (1536 KB) 101 Reserved (2048 KB) 110 64 KB (the value for the SPC560P44Lx, SPC560P50Lx device in the data Flash module) 111 Reserved **Note:** The value for this bitfield is different between the code and data Flash modules.
8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 160. MCR field descriptions(Continued)

Field	Description
LAS[2:0] 9:11	Low Address Space 2–0 The value of the LAS field corresponds to the configuration of the Low Address Space: 000 Reserved 001 Reserved 010 32 KB + (2 × 16 KB) + (2 × 32 KB) + 128 KB (the value for the SPC560P44Lx, SPC560P50Lx device in the code Flash module) 011 Reserved 100 Reserved 101 Reserved 110 4 × 16 KB (the value for the SPC560P44Lx, SPC560P50Lx device in the data Flash module) 111 Reserved **Note:** The value for this bitfield is different between the code and data Flash modules.
12:14	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MAS 15	Mid Address Space The value of the MAS field corresponds to the configuration of the Mid Address Space: 0 0 KB or 2 × 128 KB 1 Reserved
EER 16	ECC Event Error EER provides information on previous reads. When an ECC Double Error detection occurs, the EER bit is set to 1. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. In the event of a ECC Single Error detection and correction, this bit will not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 An ECC Double Error occurred during a previous read.
RWE 17	Read-while-Write event Error RWE provides information on previous reads when a Modify operation is on going. If a RWW Error occurs, the RWE bit is set to 1. Read-While-Write Error means that a read access to the Flash module has occurred while the FPEC was performing a program or Erase operation or an Array Integrity Check. This bit must then be cleared, or a reset must occur before this bit will return to a 0 state. This bit may not be set to 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) were correct. Since this bit is an error flag, it must be cleared to 0 by writing 1 to the register location. A write of 0 will have no effect. 0 Reads are occurring normally. 1 A RWW Error occurred during a previous read. **Note:** If stall/terminate-while-write is used, the software should ignore the setting of the RWE flag and should clear this flag after each erase operation. If stall/terminate-while-write is not used, software can handle the RWE error normally.
18:19	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 160. MCR field descriptions(Continued)

Field	Description
PEAS 20	Program/Erase Access Space PEAS indicates which space is valid for program and Erase operations: main array space or shadow/test space. PEAS = 0 indicates that the main address space is active for all Flash module program and erase operations. PEAS = 1 indicates that the test or shadow address space is active for program and erase. The value in PEAS is captured and held with the first interlock write done for Modify operations. The value of PEAS is retained between sampling events (that is, subsequent first interlock writes). 0 Shadow/Test address space is disabled for program/erase and main address space enabled. 1 Shadow/Test address space is enabled for program/erase and main address space disabled.
DONE 21	Modify Operation Done DONE indicates if the Flash module is performing a high voltage operation. DONE is set to 1 on termination of the Flash module reset. DONE is cleared to 0 just after a 0-to-1 transition of EHV, which initiates a high voltage operation, or after resuming a suspended operation. DONE is set to 1 at the end of program and erase high voltage sequences. DONE is set to 1 (within t_{PABT} or t_{EABT}, equal to P/E Abort Latency) after a 1-to-0 transition of EHV, which terminates a high voltage program/erase operation. DONE is set to 1 (within t_{ESUS}, time equal to Erase Suspend Latency) after a 0-to-1 transition of ESUS, which suspends an erase operation. 0 Flash is executing a high voltage operation. 1 Flash is not executing a high voltage operation.
PEG 22	Program/Erase Good The PEG bit indicates the completion status of the last Flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared to 0, indicating the sequence failed. PEG is set to 1 when the Flash module is reset, unless a Flash initialization error has been detected. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to a termination or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1-to-0 transition or EHV makes a 0-to-1 transition. The value in PEG is not valid after a 0-to-1 transition of DONE caused by ESUS being set to logic 1. If program or erase are attempted on blocks that are locked, the response is PEG = 1, indicating that the operation was successful, and the content of the block were properly protected from the program or erase operation. If a program operation tries to program at 1 bits that are at 0, the program operation is correctly executed on the new bits to be programmed at 0, but PEG is cleared, indicating that the requested operation has failed. In Array Integrity Check or Margin Mode, PEG is set to 1 when the operation is completed, regardless the occurrence of any error. The presence of errors can be detected only comparing checksum value stored in UMIRS[0:1]. 0 Program or erase operation failed; or program, erase, Array Integrity Check, or Margin Mode was terminated. 1 Program or erase operation successful; or Array Integrity Check or Margin Mode completed successfully.
23:26	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 160. MCR field descriptions(Continued)

Field	Description
PGM 27	Program PGM sets up the Flash module for a program operation. A 0-to-1 transition of PGM initiates a program sequence. A 1-to-0 transition of PGM ends the program sequence. PGM can be set only under User mode Read (ERS is low and UT0[AIE] is low). PGM can be cleared by the user only when EHV is low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence.
PSUS 28	Program Suspend Writing to this bit has no effect, but the written data can be read back.
ERS 29	Erase ERS sets up the Flash module for an Erase operation. A 0-to-1 transition of ERS initiates an Erase sequence. A 1-to-0 transition of ERS ends the Erase sequence. ERS can be set only under User mode Read (PGM is low and UT0[AIE] is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0 Flash is not executing an Erase sequence. 1 Flash is executing an Erase sequence.
ESUS 30	Erase Suspend ESUS indicates that the Flash module is in Erase Suspend or in the process of entering a Suspend state. The Flash module is in Erase Suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS = 1 and EHV = 1, and PGM = 0. A 0-to-1 transition of ESUS starts the sequence that sets DONE and places the Flash in erase suspend. The Flash module enters Suspend within t_{ESUS} of this transition. ESUS can be cleared only when DONE = 1 and EHV = 1, and PGM = 0. A 1-to-0 transition of ESUS with EHV = 1 starts the sequence that clears DONE and returns the Module to Erase. The Flash module cannot exit Erase Suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.

Table 160. MCR field descriptions(Continued)

Field	Description
EHV 31	Enable High Voltage The EHV bit enables the Flash module for a high voltage program/Erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/Erase sequence. EHV may be set under one of the following conditions: – Erase (ERS = 1, ESUS = 0, UT0[AIE] = 0) – Program (ERS = 0, ESUS = 0, PGM = 1, UT0[AIE] = 0) In normal operation, a 1-to-0 transition of EHV with DONE high and ESUS low terminates the current program/Erase high voltage operation. When an operation is terminated, there is a 1-to-0 transition of EHV with DONE low and the eventual Suspend bit low. A termination causes the value of PEG to be cleared, indicating a failing program/Erase; address locations being operated on by the terminated operation contain indeterminate data after a termination. A suspended operation cannot be terminated. Terminating a high voltage operation leaves the Flash module addresses in an indeterminate data state. This may be recovered by executing an Erase on the affected blocks. EHV may be written during Suspend. EHV must be high to exit Suspend. EHV may not be written after ESUS is set and before DONE transitions high. EHV may not be cleared after ESUS is cleared and before DONE transitions low. 0 Flash is not enabled to perform an high voltage operation. 1 Flash is enabled to perform an high voltage operation.

A number of MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding description, but those locks do not consider the effects of trying to write two or more bits simultaneously.

The Flash module does not allow the user to write bits simultaneously which would put the device into an illegal state. This is implemented through a priority mechanism among the bits. *Table 161* shows the bit changing priorities.

Table 161. MCR bits set/clear priority levels

Priority level	MCR bits
1	ERS
2	PGM
3	EHV
4	ESUS

If the user attempts to write two or more MCR bits simultaneously, only the bit with the lowest priority level is written.

17.3.7.2 Low/Mid Address Space Block Locking register (LML)

The Low/Mid Address Space Block Locking register provides a means to protect blocks from being modified. These bits, along with bits in the SLL register, determine if the block is locked from program or erase. An “OR” of LML and SLL determine the final lock status. Identical LML registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the LML register has a related Non-Volatile Low/Mid Address Space Block Locking register (NVLML) located in TestFlash that contains the default reset value for LML. The NVLML register is read during the reset phase of the Flash module and loaded into the LML. The reset value is 0x00XX_XXXX, initially determined by the NVLML value from test sector.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LME	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK 15	LLK 14	LLK 13	LLK 12	LLK 11	LLK 10	LLK 9	LLK 8	LLK 7	LLK 6	LLK 5	LLK 4	LLK 3	LLK 2	LLK 1	LLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 164. Low/Mid Address Space Block Locking register (LML)

17.3.7.3 Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

Address: Base + 0x40_3DE8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TSLK	0	0	MLK1	MLK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LLK 15	LLK 14	LLK 13	LLK 12	LLK 11	LLK 10	LLK 9	LLK 8	LLK 7	LLK 6	LLK 5	LLK 4	LLK 3	LLK 2	LLK 1	LLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 165. Non-Volatile Low/Mid Address Space Block Locking register (NVLML)

The NVLML register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVLML registers are provided in the code Flash and the data Flash blocks.

Table 162. LML and NVLML field descriptions

Field	Description
LME(1) 0	Low/Mid Address Space Block Enable This bit enables the Lock registers (TSLK, MLK[1:0], and LLK[15:0]) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME the password 0xA1A11111 must be written to the LML register. 0 Low Address Locks are disabled: TSLK, MLK[1:0], and LLK[15:0] cannot be written. 1 Low Address Locks are enabled: TSLK, MLK[1:0], and LLK[15:0] can be written.

Table 162. LML and NVLML field descriptions(Continued)

Field	Description
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
TSLK 11	Test/Shadow Address Space Block Lock This bit locks the block of Test and Shadow Address Space from program and Erase (Erase is any case forbidden for Test block). A value of 1 in the TSLK register signifies that the Test/Shadow block is locked for program and Erase. A value of 0 in the TSLK register signifies that the Test/Shadow block is available to receive program and Erase pulses. The TSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the TSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the TSLK register. The TSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the TSLK bit (assuming erased fuses) would be locked. TSLK is not writable unless LME is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also SLL[STSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MLK[1:0] 14:15	Mid Address Space Block Lock 1-0 These bits lock the blocks of Mid Address Space from program and Erase. MLK[1:0] are related to sectors B0F[7:6], respectively. See *Table 152* for more information. A value of 1 in a bit of the MLK bitfield signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the MLK bitfield signifies that the corresponding block is available to receive program and Erase pulses. The MLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the MLK bitfield is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the MLK bitfields. The MLK bits may be written as a register. Reset causes the bits to revert to their TestFlash block value. The default value of the MLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the MLK bits default to locked, and are not writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. MLK is not writable unless LME is high. 0 Mid Address Space Block is unlocked and can be modified (if also SLL[SMLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.

Table 162. LML and NVLML field descriptions(Continued)

Field	Description
LLK[15:0] 16:31	Low Address Space Block Lock 15-0 These bits lock the blocks of Low Address Space from program and Erase. For code Flash, LLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 152* for more information. For data Flash, LLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 153* for more information. A value of 1 in a bit of the LLK bitfield signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the LLK bitfield signifies that the corresponding block is available to receive program and Erase pulses. The LLK bitfield is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the LLK bitfield is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the LLK bitfields. The LLK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the LLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the LLK bits will default to locked, and will not be writable. The reset value is always 1 (independent of the TestFlash block), and register writes have no effect. In the code Flash macrocell, bits LLK[15:6] are read-only and locked at 1. In the data Flash macrocell, bits LLK[15:4] are read-only and locked at 1. LLK is not writable unless LME is high. 0 Low Address Space Block is unlocked and can be modified (if also SLL[SLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

1. This field is present only in LML

17.3.7.4 Secondary Low/Mid Address Space Block Locking register (SLL)

The Secondary Low/Mid Address Space Block Locking register provides an alternative means to protect blocks from being modified. These bits, along with bits in the LML register, determine if the block is locked from program or Erase. An “OR” of LML and SLL determine the final lock status. Identical SLL registers are provided in the code Flash and the data Flash blocks.

In the code Flash module, the SLL register has a related Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL) located in TestFlash that contains the default reset value for SLL. The reset value is 0x00XX_XXXX, initially determined by NVSLL.

The NVSLL register is read during the reset phase of the Flash module and loaded into the SLL.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLE	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK 15	SLK 14	SLK 13	SLK 12	SLK 11	SLK 10	SLK 9	SLK 8	SLK 7	SLK 6	SLK 5	SLK 4	SLK 3	SLK 2	SLK 1	SLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 166. Secondary Low/mid address space block Locking reg (SLL)

17.3.7.5 Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

The NVSLL register is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes. Identical NVSLL registers are provided in the code Flash and the data Flash blocks.

Address: Base + 0x40_3DF8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	STSLK	0	0	SMK1	SMK0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	x	0	0	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLK 15	SLK 14	SLK 13	SLK 12	SLK 11	SLK 10	SLK 9	SLK 8	SLK 7	SLK 6	SLK 5	SLK 4	SLK 3	SLK 2	SLK 1	SLK 0
W																
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 167. Non-Volatile Secondary Low/Mid Address Space Block Locking register (NVSLL)

Table 163. SLL and NVSLL field descriptions

Field	Description
SLE(1) 0	Secondary Low/Mid Address Space Block Enable This bit enables the Lock registers (STSLK, SMK[1:0], and SLK[15:0]) to be set or cleared by registers writes. This bit is a status bit only. The method to set this bit is to write a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE the password 0xC3C3_3333 must be written to the SLL register. 0 Secondary Low/Mid Address Locks are disabled: STSLK, SMK[1:0], and SLK[15:0] cannot be written. 1 Secondary Low/Mid Address Locks are enabled: STSLK, SMK[1:0], and SLK[15:0] can be written.
1:10	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 163. SLL and NVSLL field descriptions(Continued)

Field	Description
STSLK 11	Secondary Test/Shadow address space block LocK This bit is used as an alternate means to lock the block of Test and Shadow Address Space from program and Erase (Erase is any case forbidden for Test block). A value of 1 in the STSLK bitfield signifies that the Test/Shadow block is locked for program and Erase. A value of 0 in the STSLK register signifies that the Test/Shadow block is available to receive program and Erase pulses. The STSLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the STSLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the STSLK register. The STSLK bit may be written as a register. Reset will cause the bit to go back to its TestFlash block value. The default value of the STSLK bit (assuming erased fuses) would be locked. STSLK is not writable unless SLE is high. 0 Test/Shadow Address Space Block is unlocked and can be modified (if also LML[TSLK] = 0). 1 Test/Shadow Address Space Block is locked and cannot be modified.
12:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
SMK[1:0] 14:15	Secondary Mid Address Space Block Lock 1–0 These bits are used as an alternate means to lock the blocks of Mid Address Space from program and Erase. SMK[1:0] are related to sectors B0F[7:6], respectively. See *Table 152* for more information. A value of 1 in a bit of the SMK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the SMK register signifies that the corresponding block is available to receive program and Erase pulses. The SMK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SMK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SMK registers. The SMK bits may be written as a register. Reset will cause the bits to go back to their TestFlash block value. The default value of the SMK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SMK bits will default to locked, and will not be writable. The reset value will always be 1 (independent of the TestFlash block), and register writes have no effect. SMK is not writable unless SLE is high. 0 Mid Address Space Block is unlocked and can be modified (if also LML[MLK] = 0). 1 Mid Address Space Block is locked and cannot be modified.

Table 163. SLL and NVSLL field descriptions(Continued)

Field	Description
SLK[15:0] 16:31	Secondary Low Address Space Block Lock 15–0 These bits are used as an alternate means to lock the blocks of Low Address Space from program and Erase. For code Flash, SLK[5:0] are related to sectors B0F[5:0], respectively. See *Table 152* for more information. For data Flash, SLK[3:0] are related to sectors B1F[3:0], respectively. See *Table 153* for more information. A value of 1 in a bit of the SLK register signifies that the corresponding block is locked for program and Erase. A value of 0 in a bit of the SLK register signifies that the corresponding block is available to receive program and Erase pulses. The SLK register is not writable once an interlock write is completed until MCR[DONE] is set at the completion of the requested operation. Likewise, the SLK register is not writable if a high voltage operation is suspended. Upon reset, information from the TestFlash block is loaded into the SLK registers. The SLK bits may be written as a register. Reset causes the bits to go back to their TestFlash block value. The default value of the SLK bits (assuming erased fuses) would be locked. In the event that blocks are not present (due to configuration or total memory size), the SLK bits default to locked, and are not writable. The reset value will always be 1 (independent of the TestFlash block), and register writes will have no effect. In the code Flash macrocell, bits SLK[15:6] are read-only and locked at 1. . In the data Flash macrocell, bits SLK[15:4] are read-only and locked at 1. SLK is not writable unless SLE is high. 0 Low Address Space Block is unlocked and can be modified (if also LML[LLK] = 0). 1 Low Address Space Block is locked and cannot be modified.

1. This field is present only in SLL

17.3.7.6 Low/Mid Address Space Block Select register (LMS)

The Low/Mid Address Space Block Select register provides a means to select blocks to be operated on during erase. Identical LMS registers are provided in the code Flash and the data Flash blocks.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSL1	MSL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LSL 15	LSL 14	LSL 13	LSL 12	LSL 11	LSL 10	LSL 9	LSL 8	LSL 7	LSL 6	LSL 5	LSL 4	LSL 3	LSL 2	LSL 1	LSL 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 168. Low/Mid Address Space Block Select register (LMS)

Table 164. LMS field descriptions

Field	Description
0:13	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
MSL[1:0] 14:15	Mid Address Space Block Select 1–0 A value of 1 in the select register signifies that the block is selected for erase. See *Table 152* for more information. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. MSL[1:0] are related to sectors B0F[7:6], respectively. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding MSL bits default to unselected, and are not writable. The reset value will always be 0, and register writes have no effect. 0 Mid Address Space Block is unselected for Erase. 1 Mid Address Space Block is selected for Erase.
LSL[15:0] 16:31	Low Address Space Block Select 15–0 A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected for erase. The reset value for the select register is 0, or unselected. For code Flash, LSL[5:0] are related to sectors B0F[5:0], respectively. See *Table 152* for more information. For data Flash, LSL[3:0] are related to sectors B1F[3:0], respectively. See *Table 153* for more information. The blocks must be selected (or unselected) before doing an erase interlock write as part of the Erase sequence. The select register is not writable once an interlock write is completed or if a high voltage operation is suspended. In the event that blocks are not present (due to configuration or total memory size), the corresponding LSL bits will default to unselected, and will not be writable. The reset value will always be 0, and register writes will have no effect. In the code Flash macrocell, bits LSL[15:6] are read-only and locked at 0. In the data Flash macrocell, bits LSL[15:4] are read-only and locked at 0. 0 Low Address Space Block is unselected for Erase. 1 Low Address Space Block is selected for Erase.

17.3.7.7 Address Register (ADR)

The Address Register provides the first failing address in the event module failures (ECC, RWW, or FPEC) or the first address at which a ECC single error correction occurs.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	AD 22	AD 21	AD 20	AD 19	AD 18	AD 17	AD 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AD 15	AD 14	AD 13	AD 12	AD 11	AD 10	AD 9	AD 8	AD 7	AD 6	AD 5	AD 4	AD 3	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 169. Address Register (ADR)

Table 165. ADR field descriptions

Field	Description
0:8	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
AD[22:3] 9:28	Address 22–3 ADR provides the first failing address in the event of ECC error (MCR[EER] set) or the first failing address in the event of RWW error (MCR[RWE] set), or the address of a failure that may have occurred in a FPEC operation (MCR[PEG] cleared). ADR also provides the first address at which a ECC single error correction occurs (MCR[EDC] set). The ECC double error detection takes the highest priority, followed by the RWW error, the FPEC error, and the ECC single error correction. When accessed ADR will provide the address related to the first event occurred with the highest priority. The priorities between these four possible events is summarized in *Table 166*. This address is always a double word address that selects 64 bits. In case of a simultaneous ECC double error detection on both double words of the same page, bit AD3 will output 0. The same is valid for a simultaneous ECC single error correction on both double words of the same page. In User mode, ADR is read only.
29:31	*Reserved* Write these bits has no effect and read these bits always outputs 0.

Table 166. ADR content: priority list

Priority level	Error flag	ADR content
1	MCR[EER] = 1	Address of first ECC Double Error
2	MCR[RWE] = 1	Address of first RWW Error
3	MCR[PEG] = 0	Address of first FPEC Error
4	MCR[EDC] = 1	Address of first ECC Single Error Correction

17.3.7.7.1 Platform Flash Configuration Register 0 (PFCR0)

The Platform Flash Configuration Register 0 (PFCR0) defines the configuration associated with Flash memory bank0, which corresponds to the code Flash. The register is described in *Figure 170* and *Table 167*.

Note: This register is not implemented on the data Flash block.

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BK0_APC				BK0_WWSC				BK0_RWSC				BK0_RWWC			
Reset	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BK0_RWWC	0	0	0	0	0	0	0	BK0_RWWC	B0_P0_BCFG		B0_P0_DPFE	B0_P0_IPFE	B0_P0_PFLM		B0_P0_BFE
Reset	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	1

Figure 170. Platform Flash Configuration Register 0 (PFCR0)

Table 167. PFCR0 field descriptions

Field	Description
0-4 BK0_APC	Bank0 Address Pipelining Control This field controls the number of cycles between Flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to 0b00011 by hardware reset. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. ... 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.
5-9 BK0_WWSC	Bank0 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to an appropriate value by hardware reset. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.

Table 167. PFCR0 field descriptions(Continued)

Field	Description
10-14 BK0_RWSC	Bank0 Read Wait State Control This field controls the number of wait states to be added to the Flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 22 MHz APC = RWSC = 0. 22 MHz, < 44 MHz APC = RWSC = 1. 44 MHz, < 66 MHz APC = RWSC = 2. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.
15-16,24 BK0_RWWC	Bank0 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation termination and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt. 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.
17-23	Reserved

Table 167. PFCR0 field descriptions(Continued)

Field	Description
25-26 B0_P0_BCFG	Bank0, Port 0 Page Buffer Configuration This field controls the configuration of the four page buffers in the PFlash controller. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. If enabled, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the Flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. 00 All four buffers are available for any Flash access, that is, there is no partitioning of the buffers based on the access type. 01 Reserved. 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses. This field is set to 2b11 by hardware reset.
27 B0_P0_DPFE	Bank0, Port 0 Data Prefetch Enable This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any data read access.
28 B0_P0_IPFE	Bank0, Port 0 Instruction Prefetch Enable This field enables or disables prefetching initiated by an instruction fetch read access. This field is set by hardware reset. 0 No prefetching is triggered by an instruction fetch read access. 1 If page buffers are enabled (B0_P0_BFE = 1), prefetching is triggered by any instruction fetch read access.
29-30 B0_P0_PFLM	Bank0, Port 0 Prefetch Limit This field controls the prefetch algorithm used by the PFlash controller. This field defines the prefetch behavior. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is set to 2b10 by hardware reset. 00 No prefetching is performed. 01 The referenced line is prefetched on a buffer miss, that is, *prefetch on miss*. 1*x* The referenced line is prefetched on a buffer miss, or the next sequential page is prefetched on a buffer hit (if not already present), that is, *prefetch on miss or hit*.
31 B0_P0_BFE	Bank0, Port 0 Buffer Enable This bit enables or disables page buffer read hits. It is also used to invalidate the buffers. This bit is set by hardware reset. 0 The page buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The page buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

17.3.7.7.2 Platform Flash Configuration Register 1 (PFCR1)

The Platform Flash Configuration Register 1 (PFCR1) defines the configuration associated with Flash memory bank1. This corresponds to the data Flash. The register is described in *Figure 171* and *Table 168*.

Note: This register is not implemented on the data Flash block.

Address: Base + 0x0020 Access: User read/write

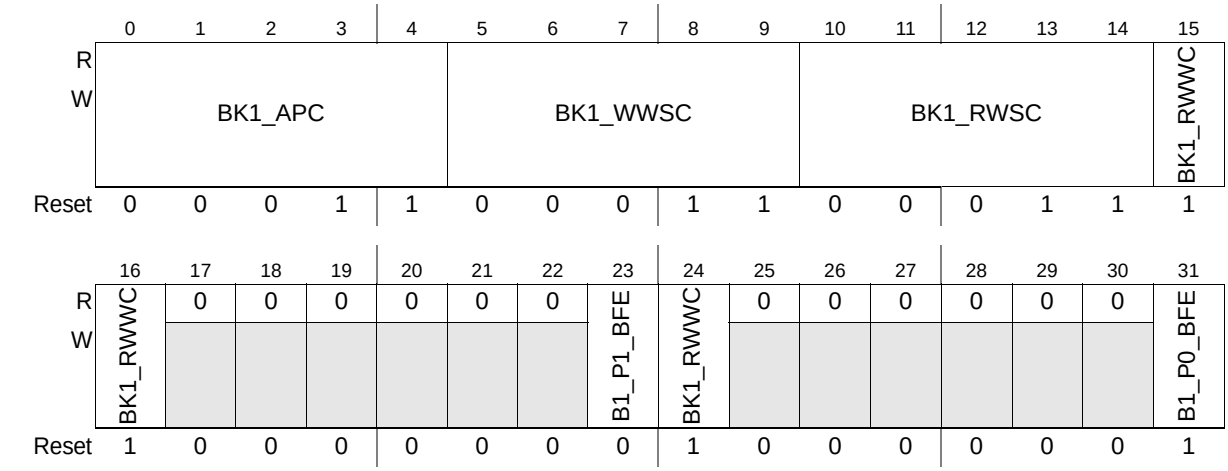


Figure 171. Platform Flash Configuration Register 1 (PFCR1)

Table 168. PFCR1 field descriptions

Field	Description
0-4 BK1_APC	Bank1 Address Pipelining Control This field controls the number of cycles between Flash array access requests. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to 0b00011 by hardware reset. 00000 Accesses may be initiated on consecutive (back-to-back) cycles. 00001 Access requests require one additional hold cycle. 00010 Access requests require two additional hold cycles. ... 11110 Access requests require 30 additional hold cycles. 11111 Access requests require 31 additional hold cycles.
5-9 BK1_WWSC	Bank1 Write Wait State Control This field controls the number of wait states to be added to the Flash array access time for writes. This field must be set to a value appropriate to the operating frequency of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. This field is set to an appropriate value by hardware reset. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.

Table 168. PFCR1 field descriptions(Continued)

Field	Description
10-14 BK1_RWSC	Bank1 Read Wait State Control This field controls the number of wait states to be added to the Flash array access time for reads. This field must be set to a value corresponding to the operating frequency of the PFlash and the actual read access time of the PFlash. Higher operating frequencies require non-zero settings for this field for proper Flash operation. 0 MHz, < 22 MHz APC = RWSC = 0. 22 MHz, < 44 MHz APC = RWSC = 1. 44 MHz, < 66 MHz APC = RWSC = 2. This field is set to 0b00011 by hardware reset. 00000 No additional wait states are added. 00001 1 additional wait state is added. 00010 2 additional wait states are added. ... 111111 31 additional wait states are added.
15-16,24 BK1_RWWC	Bank1 Read-While-Write Control This 3-bit field defines the controller response to Flash reads while the array is busy with a program (write) or erase operation. 0*xx* Terminate any attempted read while write/erase with an error response. 100 Generate a bus stall for a read while write/erase, enable the operation abort and the abort notification interrupt. 101 Generate a bus stall for a read while write/erase, enable the operation abort, disable the abort notification interrupt. 110 Generate a bus stall for a read while write/erase, enable the stall notification interrupt, disable the abort + abort notification interrupt. 111 Generate a bus stall for a read while write/erase, disable the stall notification interrupt, disable the abort + abort notification interrupt. This field is set to 0b111 by hardware reset enabling the stall-while-write/erase and disabling the abort and notification interrupts.
17-22	Reserved, should be cleared.
23 B1_P1_BFE	Bank1, Port 1 Buffer Enable This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is cleared by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits. **Note:** This bit can be written (after reset, its value is ‘0’).
25-30	Reserved, should be cleared.
31 B1_P0_PFE	Bank1, Port 0 Buffer Enable This bit enables or disables read hits from the 128-bit holding register. It is also used to invalidate the contents of the holding register. This bit is set by hardware reset, enabling the use of the holding register. 0 The holding register is disabled from satisfying read requests. 1 The holding register is enabled to satisfy read requests on hits.

17.3.7.7.3 Platform Flash Access Protection Register (PFAPR)

The Platform Flash Access Protection Register (PFAPR) controls read and write accesses to the Flash based on system master number. Prefetching capabilities are defined on a per master basis. This register also defines the arbitration mode for controllers supporting two AHB ports. The register is described in *Figure 172* and *Table 169*.

The contents of the register are loaded from location 0x20_3E00 of the shadow region in the code Flash (bank0) array at reset. To temporarily change the values of any of the fields in the PFAPR, a write to the IPS-mapped register is performed. To change the values loaded into the PFAPR *at reset*, the word location at address 0x20_3E00 of the shadow region in the Flash array must be programmed using the normal sequence of operations. The reset value shown in *Table 172* reflects an erased or unprogrammed value from the shadow region.

Note: This register is not implemented on the data Flash block.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	ARBM		M7 PFD	M6 PFD	M5 PFD	M4 PFD	M3 PFD	M2 PFD	M1 PFD	M0 PFD
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	M7AP		M6AP		M5AP		M4AP		M3AP		M2AP		M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 172. Platform Flash Access Protection Register (PFAPR)

Table 169. PFAPR field descriptions

Field	Description
0-5	Reserved, should be cleared.
6-7 ARBM	Arbitration Mode This 2-bit field controls the arbitration for PFlash controllers supporting 2 AHB ports. 00 Fixed priority arbitration with AHB p0 > p1. 01 Fixed priority arbitration with AHB p1 > p0. 1*x* Round-robin arbitration.
8-15 M*x*PFD	Master *x* Prefetch Disable (*x* = 0,1,2,...,7) These bits control whether prefetching may be triggered based on the master number of the requesting AHB master. This field is further qualified by the PFCR0[B0_P*x*_DPFE, B0_P*x*_IPFE, B*x*_P*y*_BFE] bits. 0 Prefetching may be triggered by this master. 1 No prefetching may be triggered by this master.

Table 169. PFAPR field descriptions(Continued)

Field	Description
16-31 M*x*AP	Master *x* Access Protection (*x* = 0,1,2,...,7) These fields control whether read and write accesses to the Flash are allowed based on the master number of the initiating module. 00 No accesses may be performed by this master. 01 Only read accesses may be performed by this master. 10 Only write accesses may be performed by this master. 11 Both read and write accesses may be performed by this master.

17.3.7.8 User Test 0 register (UT0)

The User Test feature gives the user of the Flash module the ability to perform test features on the Flash. The User Test 0 register allows controlling the way in which the Flash content check is done.

The UT0[MRE], UT0[MRV], UT0[AIS], UT0[EIE], and DSI[7:0] bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	0	0	0	0	0	0	0	DSI7	DSI6	DSI5	DSI4	DSI3	DSI2	DSI1	DSI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	X	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 173. User Test 0 register (UT0)

Table 170. UT0 field descriptions

Field	Description
UTE 0	User Test Enable This status bit indicates when User Test is enabled. All bits in UT0–2 and UMISR0–4 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. UTE can only be cleared if UTE[AID] = 1, UTE[AIE] and UTE[EIE] = 0 For UTE the password 0xF9F9_9999 must be written to the UT0 register.
1:7	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.

Table 170. UT0 field descriptions(Continued)

Field	Description
DSI7-0 8:15	Data Syndrome Input 7–0 These bits represent the input of Syndrome bits of ECC logic used in the ECC Logic Check. The DSI7–0 bits correspond to the 8 syndrome bits on a double word. These bits are not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 The syndrome bit is forced at 0. 1 The syndrome bit is forced at 1.
16:24	*Reserved* (Read Only) A write to these bits has no effect. A read of these bits always outputs 0.
25	*Reserved* (Read/Write) This bit can be written and its value can be read back, but there is no function associated. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect.
MRE 26	Margin Read Enable MRE enables margin reads to be done. This bit, combined with MRV, enables regular user mode reads to be replaced by margin reads. Margin reads are only active during Array Integrity Checks; Normal user reads are not affected by MRE. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Margin reads are disabled. All reads are User mode reads. 1 Margin reads are enabled.
MRV 27	Margin Read Value If MRE is high, MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 Zero's (programmed) margin reads are requested (if MRE = 1). 1 One's (erased) margin reads are requested (if MRE = 1).
EIE 28	ECC data Input Enable EIE enables the ECC Logic Check operation to be done. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. 0 ECC Logic Check is disabled. 1 ECC Logic Check is enabled.
AIS 29	Array Integrity Sequence AIS determines the address sequence to be used during array integrity checks or Margin Mode. The default sequence (AIS = 0) is meant to replicate sequences normal user code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. This bit is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data, and writes have no effect. In Margin Mode only the linear sequence (AIS = 1) is allowed, while the proprietary sequence (AIS = 0) is forbidden. 0 Array Integrity sequence is a proprietary sequence. 1 Array Integrity or Margin Mode sequence is sequential.

Table 170. UT0 field descriptions(Continued)

Field	Description
AIE 31	Array Integrity Enable AIE set to 1 starts the Array Integrity Check done on all selected and unlocked blocks. The pattern is selected by AIS, and the MISR (UMISR0–4) can be checked after the operation is complete, to determine if a correct signature is obtained. AIE can be set only if MCR[ERS], MCR[PGM], and MCR[EHV] are all low. 0 Array Integrity Checks are disabled. 1 Array Integrity Checks are enabled.
AID 31	Array Integrity Done AID is cleared upon an Array Integrity Check being enabled (to signify the operation is on-going). Once completed, AID is set to indicate that the Array Integrity Check is complete. At this time, the MISR (UMISR0–4) can be checked. 0 Array Integrity Check is on-going. 1 Array Integrity Check is done.

17.3.7.9 User Test 1 register (UT1)

The User Test 1 register allows to enable the checks on the ECC logic related to the 32 LSB of the Double Word.

The User Test 1 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI 31	DAI 30	DAI 29	DAI 28	DAI 27	DAI 26	DAI 25	DAI 24	DAI 23	DAI 22	DAI 21	DAI 20	DAI 19	DAI 18	DAI 17	DAI 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI 15	DAI 14	DAI 13	DAI 12	DAI 11	DAI 10	DAI 9	DAI 8	DAI 7	DAI 6	DAI 5	DAI 4	DAI 3	DAI 2	DAI 1	DAI 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 174. User Test 1 register (UT1)

Table 171. UT1 field descriptions

Field	Description
DAI[31:0] 0:31	Data Array Input 31–0 These bits represent the input of the even word of ECC logic used in the ECC Logic Check. The DAI[31:0] bits correspond to the 32 array bits representing Word 0 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

17.3.7.10 User Test 2 register (UT2)

The User Test 2 register allows to enable the checks on the ECC logic related to the 32 MSB of the Double Word.

The User Test 2 register is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI 63	DAI 62	DAI 61	DAI 60	DAI 59	DAI 58	DAI 57	DAI 56	DAI 55	DAI 54	DAI 53	DAI 52	DAI 51	DAI 50	DAI 49	DAI 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI 47	DAI 46	DAI 45	DAI 44	DAI 43	DAI 42	DAI 41	DAI 40	DAI 39	DAI 38	DAI 37	DAI 36	DAI 35	DAI 34	DAI 33	DAI 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 175. User Test 2 register (UT2)

Table 172. UT2 field descriptions

Field	Description
DAI[63:32] 0:31	Data Array Input [63:32] These bits represent the input of the odd word of ECC logic used in the ECC Logic Check. The DAI[63:32] bits correspond to the 32 array bits representing Word 1 within the double word. 0 The array bit is forced at 0. 1 The array bit is forced at 1.

17.3.7.11 User Multiple Input Signature Register 0 (UMISR0)

The Multiple Input Signature Register 0 (UMISR0) provides a mean to evaluate the array integrity. UMISR0 represents the bits 31:0 of the whole 144-bit word (2 double words including ECC).

UMISR0 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 031	MS 030	MS 029	MS 028	MS 027	MS 026	MS 025	MS 024	MS 023	MS 022	MS 021	MS 020	MS 019	MS 018	MS 017	MS 016
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 015	MS 014	MS 013	MS 012	MS 011	MS 010	MS 009	MS 008	MS 007	MS 006	MS 005	MS 004	MS 003	MS 002	MS 001	MS 000
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 176. User Multiple Input Signature Register 0 (UMISR0)

Table 173. UMSIR0 field descriptions

Field	Description
MS[031:000] 0:31	Multiple input Signature 031–000 These bits represent the MISR value obtained by accumulating the bits 31:0 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR0 register.

17.3.7.12 User Multiple Input Signature Register 1 (UMISR1)

The Multiple Input Signature Register 1 (UMISR1) provides a means to evaluate the array integrity. UMISR1 represents bits 63:32 of the whole 144-bit word (2 double words including ECC).

UMISR1 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x004C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 063	MS 062	MS 061	MS 060	MS 059	MS 058	MS 057	MS 056	MS 055	MS 054	MS 053	MS 052	MS 051	MS 050	MS 049	MS 048
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 047	MS 046	MS 045	MS 044	MS 043	MS 042	MS 041	MS 040	MS 039	MS 038	MS 037	MS 036	MS 035	MS 034	MS 033	MS 032
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 177. User Multiple Input Signature Register 1 (UMISR1)

Table 174. UMISR1 field descriptions

Field	Description
MS[063:032] 0:31	Multiple input Signature 063–032 These bits represent the MISR value obtained accumulating the bits 63:32 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR1 register.

17.3.7.13 User Multiple Input Signature Register 2 (UMISR2)

The Multiple Input Signature Register (UMISR2) provides a mean to evaluate the array integrity. UMISR2 represents the bits 95-64 of the whole 144-bit word (2 double words including ECC).

UMISR2 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0050 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 095	MS 094	MS 093	MS 092	MS 091	MS 090	MS 089	MS 088	MS 087	MS 086	MS 085	MS 084	MS 083	MS 082	MS 081	MS 080
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 079	MS 078	MS 077	MS 076	MS 075	MS 074	MS 073	MS 072	MS 071	MS 070	MS 069	MS 068	MS 067	MS 066	MS 065	MS 064
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 178. User Multiple Input Signature Register 2 (UMISR2)

Table 175. UMISR2 field descriptions

Field	Description
MS[095:064] 0:31	Multiple input Signature 095–064 These bits represent the MISR value obtained by accumulating the bits 95:64 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR2 register.

17.3.7.14 User Multiple Input Signature Register 3 (UMISR3)

The Multiple Input Signature Register 3 (UMISR3) provides a means to evaluate the array integrity. UMISR3 represents bits 127:96 of the whole 144-bit word (2 double words including ECC).

UMISR3 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0054 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 127	MS 126	MS 125	MS 124	MS 123	MS 122	MS 121	MS 120	MS 119	MS 118	MS 117	MS 116	MS 115	MS 114	MS 113	MS 112
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 111	MS 110	MS 109	MS 108	MS 107	MS 106	MS 105	MS 104	MS 103	MS 102	MS 101	MS 100	MS 099	MS 098	MS 097	MS 096
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 179. User Multiple Input Signature Register 3 (UMISR3)

Table 176. UMISR3 field descriptions

Field	Description
MS[127:096] 0:31	Multiple Input Signature 127–096 These bits represent the MISR value obtained accumulating bits 127:96 of all the pages read from the Flash memory. The MS can be seeded to any value by writing the UMISR3 register.

17.3.7.15 User Multiple Input Signature Register 4 (UMISR4)

The Multiple Input Signature Register 4 (UMISR4) provides a means to evaluate the array integrity. The UMISR4 represents the ECC bits of the whole 144-bit word (2 double words including ECC). Bits 8:15 are ECC bits for the odd double word and bits 24:31 are the ECC bits for the even double word. Bits 4:5 and 20:21 of UMISR4 are the double and single ECC error detection for odd and even double words, respectively.

UMISR4 is not accessible whenever MCR[DONE] or UT0[AID] are low. Reads return indeterminate data. Writes have no effect.

Address: Base + 0x0058 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MS 159	MS 158	MS 157	MS 156	MS 155	MS 154	MS 153	MS 152	MS 151	MS 150	MS 149	MS 148	MS 147	MS 146	MS 145	MS 144
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MS 143	MS 142	MS 141	MS 140	MS 139	MS 138	MS 137	MS 136	MS 135	MS 134	MS 133	MS 132	MS 131	MS 130	MS 129	MS 128
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 180. User Multiple Input Signature Register 4 (UMISR4)

Table 177. UMISR4 field descriptions

Field	Description
MS[159:128] 0:31	Multiple Input Signature 159:128 These bits represent the MISR value obtained accumulating: – MS[135:128]—8 ECC bits for the even double word – MS138—Single ECC error detection for even double word – MS139—Double ECC error detection for even double word – MS[151:144]—8 ECC bits for the odd double word – MS154—Single ECC error detection for odd double word – MS155—Double ECC error detection for odd double word The MS can be seeded to any value by writing the UMISR4 register.

17.3.7.16 Non-Volatile Private Censorship Password 0 register (NVPWD0)

The Non-Volatile Private Censorship Password 0 register (NVPWD0) contains the 32 LSB of the password used to validate the Censorship information contained in NVSCI0–1 registers.

Note: This register is not implemented on the data Flash block.

Address: 0x20_3DD8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD 31	PWD 30	PWD 29	PWD 28	PWD 27	PWD 26	PWD 25	PWD 24	PWD 23	PWD 22	PWD 21	PWD 20	PWD 19	PWD 18	PWD 17	PWD 16
Reset	1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD 15	PWD 14	PWD 13	PWD 12	PWD 11	PWD 10	PWD 9	PWD 8	PWD 7	PWD 6	PWD 5	PWD 4	PWD 3	PWD 2	PWD 1	PWD 0
Reset	1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0

Figure 181. Non-Volatile private Censorship Password 0 register (NVPWD0)

Table 178. NVPWD0 field descriptions

Field	Description
PWD[31:0] 0:31	Password 31–0 The PWD[31:0] bits represent the 32 LSB of the private censorship password.

17.3.7.17 Non-Volatile Private Censorship Password 1 register (NVPWD1)

The Non-Volatile Private Censorship Password 1 Register (NVPWD1) contains the 32 MSB of the password used to validate the Censorship information contained in NVSCI0–1 registers.

Note: This register is not implemented on the data Flash block.

Address: 0x20_3DDC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PWD 63	PWD 62	PWD 61	PWD 60	PWD 59	PWD 58	PWD 57	PWD 56	PWD 55	PWD 54	PWD 53	PWD 52	PWD 51	PWD 50	PWD 49	PWD 48
Reset	1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWD 47	PWD 46	PWD 45	PWD 44	PWD 43	PWD 42	PWD 41	PWD 40	PWD 39	PWD 38	PWD 37	PWD 36	PWD 35	PWD 34	PWD 33	PWD 32
Reset	1	0	1	1	1	1	1	0	1	1	1		1	1	1	1

Figure 182. Non-Volatile Private Censorship Password 1 register (NVPWD1)

Table 179. NVPWD1 field descriptions

Field	Description
0:31	**PWD63–32**: *PassWorD 63–32* The PWD63–32 registers represent the 32 MSB of the Private Censorship Password.

17.3.7.18 Non-Volatile System Censoring Information 0 register (NVSCI0)

The Non-Volatile System Censoring Information 0 register (NVSCI0) stores the 32 LSB of the Censorship Control Word of the device.

NVSCI0 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Address: 0x20_3DE0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC	SC
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW	CW
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Figure 183. Non-Volatile System Censoring Information 0 register (NVSCI0)

Table 180. NVSCI0 field descriptions

Field	Description
SC[15:0] 0:15	Serial Censorship control word 15–0 These bits represent the 16 LSB of the Serial Censorship Control Word (SCCW). If SC[15:0] = 0x55AA and NVSCI1 = NVSCI0, the Public Access is disabled. If SC[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0, the Public Access is enabled.
CW[15:0] 16:31	Censorship control Word 15–0 These bits represent the 16 LSB of the Censorship Control Word (CCW). If CW[15:0] = 0x55AA and NVSCI1 = NVSCI0, the Censored mode is disabled. If CW[15:0] ≠ 0x55AA or NVSCI1 ≠ NVSCI0, the Censored mode is enabled.

17.3.7.19 Non-Volatile System Censoring Information 1 register (NVSCI1)

The Non-Volatile System Censoring Information 1 register (NVSCI1) stores the 32 MSB of the Censorship Control Word of the device.

NVSCI1 is a non-volatile register located in Shadow sector. It is read during the reset phase of the Flash module and the protection mechanisms are activated consequently.

The parts are delivered uncensored to the user. Delivery value: 0x55AA_55AA.

Note: This register is not implemented on the data Flash block.

Address: 0x20_3DE4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SC 31	SC 30	SC 29	SC 28	SC 27	SC 26	SC 25	SC 24	SC 23	SC 22	SC 21	SC 20	SC 19	SC 18	SC 17	SC 16
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CW 31	CW 30	CW 29	CW 28	CW 27	CW 26	CW 25	CW 24	CW 23	CW 22	CW 21	CW 20	CW 19	CW 18	CW 17	CW 16
Reset	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

Figure 184. Non-Volatile System Censoring Information 1 register (NVSCI1)

Table 181. NVSCI1 field descriptions

Field	Description
SC[32:16] 0:15	Serial Censorship control word 32–16 These bits represent the 16 MSB of the Serial Censorship Control Word (SCCW). If SC[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Public Access is disabled. If SC[32:16] ≠ 0x55AA or NVSCI1≠ NVSCI0, the Public Access is enabled.
CW[32:16] 16:31	Censorship control Word 32–16 These bits represent the 16 MSB of the Censorship Control Word (CCW). CW[32:16] = 0x55AA and NVSCI1 = NVSCI0, the Censored mode is disabled. CW[32:16]≠ 0x55AA or NVSCI1≠ NVSCI0, the Censored mode is enabled.

17.3.7.20 Non-Volatile User Options register (NVUSRO)

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Note: This register is not implemented on the data Flash block.

Address: 0x20_3E18 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WATCH DOG_EN	UO30	PAD3V5V	UO28	UO27	UO26	UO25	UO24	UO23	UO22	UO21	UO20	UO19	UO18	UO17	UO16
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	UO15	UO14	UO13	UO12	UO11	UO10	UO9	UO8	UO7	UO6	UO5	UO4	UO3	UO2	UO1	UO0
Reset	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Figure 185. Non-Volatile User Options register (NVUSRO)

The Non-Volatile User Options Register (NVUSRO) contains configuration information for the user application.

NVUSRO is a 64-bit register, the 32 most significant bits of which (bits 63:32) are “don’t care” bits that are eventually used to manage ECC codes.

Table 182. NVUSRO field descriptions

Field	Description
UO[28:0] 3:31	User Options 28–0 The UO[28:0] bits are reset based on the information stored in NVUSRO.
PAD3V5V 2	PAD3V5V 0 High Voltage supply is 5.0 V. 1 High Voltage supply is 3.3 V. Default manufacturing value before Flash initialization is '1' (3.3 V), which should ensure correct minimum slope for boundary scan.
UO[30] 1	User Option 30 The UO[30] bit is reset based on the information stored in NVUSRO.
WATCHDOG_EN 0	Watchdog Enable 0 Disable after reset. 1 Enable after reset. Default manufacturing value before Flash initialization is '1'

17.3.8 Programming considerations

17.3.8.1 Modify operation

All the modify operations of the Flash module are managed through the Flash user registers interface.

All the sectors of the Flash module belong to the same partition (Bank), therefore when a Modify operation is active on some sectors, no read access is possible on any other sector (Read-While-Modify is not supported).

During a Flash modify operation, any attempt to read any Flash location will output invalid data and the MCR[RWE] bit will be automatically set. This means that the Flash block is not fetchable when a modify operation is active and these commands must be executed from another memory (internal RAM or another Flash block).

If a reset occurs during a modify operation, the operation is interrupted and the block is reset to read mode. The data integrity of the Flash section where the modify operation has been terminated is not guaranteed. The interrupted Flash modify operation must be repeated.

In general, each modify operation is started through a sequence of three steps:

1. The first instruction selects the desired operation by setting its corresponding selection bit in MCR (MCR[PGM] or MCR[ERS]) or UT0 (UT0[MRE] or UT0[EIE]).
2. The second step defines the operands: the address and the data for programming or the sectors for erase or margin read.
3. The third instruction starts the modify operation by setting MCR[EHV] or UT0[AIE].

Once selected, but not yet started, one operation can be canceled by resetting the operation selection bit.

A summary of the available Flash modify operations are shown in *Table 183*.

Table 183. Flash modify operations

Operation	Select bit	Operands	Start bit
Double word program	MCR[PGM]	Address and data by interlock writes	MCR[EHV]
Sector erase	MCR[ERS]	LMS	MCR[EHV]
Array integrity check	None	LMS	UT0[AIE]
Margin read	UT0[MRE]	UT0[MRV] + LMS	UT0[AIE]
ECC logic check	UT0[EIE]	UT0[DSI], UT1, UT2	UT0[AIE]

Once MCR[EHV] (or UT0[AIE]) is set, the operands cannot be modified until MCR[DONE] (or UT0[AID]) is high.

In general, each modify operation is completed through a sequence of four steps:

1. Wait for operation completion: wait for bit MCR[DONE] (or UT0[AID]) to go high.
2. Check operation result: check bit MCR[PEG] (or compare UMISR0–4 with expected value).
3. Switch off FPEC by resetting MCR[EHV] (or UT0[AIE]).
4. Deselect current operation by clearing MCR[PGM] and MCR[ERS] (or UT0[MRE] and UT0[EIE]).

If a modify operation is on-going in one of the Flash blocks, it is forbidden to start any other modify operation on the other Flash block.

In the following sections, all the possible modify operations are described and some examples of the sequences needed to activate them are presented.

17.3.8.1.1 Double word program

A Flash program sequence operates on any double word within the Flash core.

One or both words within a double word may be altered in a single program operation.

Whenever the Flash is programmed, ECC bits are also programmed (unless the selected address belongs to a sector in which the ECC has been disabled in order to allow bit manipulation). ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed since ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be of 64 bits. The programming operation should completely fill selected ECC segments within the double word.

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1.

Addresses in locked/disabled blocks cannot be programmed.

The user may program the values in any or all of 2 words of a double word with a single program sequence.

Double word-bound words have addresses that differ only in address bit 2.

The program operation consists of the following sequence of events:

1. Change the value in the MCR[PGM] bit from 0 to 1.
2. Ensure the block that contains the address to be programmed is unlocked.
   a) Write the first address to be programmed with the program data.
   b) The Flash module latches address bits (22:3) at this time.
   c) The Flash module latches data written as well.
   d) This write is referred to as a program data interlock write. An interlock write may be as large as 64 bits, and as small as 32 bits (depending on the CPU bus).
3. If more than 1 word is to be programmed, write the additional address in the double word with data to be programmed. This is referred to as a program data write.
   The Flash module ignores address bits (22:3) for program data writes.
   The eventual unwritten data word defaults to 0xFFFF_FFFF.
4. Set MCR[EHV] to 1 to start the internal program sequence, or skip to step 9 to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Write a logic 0 to the MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the MCR[PGM] bit to terminate the program operation.

A program operation may be initiated with the 0-to-1 transition of the MCR[PGM] bit or by clearing the MCR[EHV] bit at the end of a previous program.

The first write after a program is initiated determines the page address to be programmed. This first write is referred to as an interlock write. The interlock write determines whether the shadow, test, or normal array space will be programmed by causing MCR[PEAS] to be set/cleared.

An interlock write must be performed before setting MCR[EHV]. The user may terminate a program sequence by clearing MCR[PGM] prior to setting MCR[EHV].

After the interlock write, additional writes only affect the data to be programmed at the word location determined by address bit 2. Unwritten locations default to a data value of 0xFFFF_FFFF. If multiple writes are done to the same location, the data for the last write is used in programming.

While MCR[DONE] is low and MCR[EHV] is high, the user may clear MCR[EHV], resulting in a program terminate. A program termination forces the module to step 8 of the program sequence.

A terminated program results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The data space being operated on before the termination will contain indeterminate data. This may be recovered by repeating the same program instruction or executing an erase of the affected blocks.

Example 1 Double word program of data 0x55AA_55AA at address 0x00_AAA8 and data 0xAA55_AA55 at address 0x00_AAAC

```
MCR   = 0x00000010;/* Set PGM in MCR: Select Operation */
(0x00AAA8)= 0x55AA55AA;/* Latch Address and 32 LSB data */
(0x00AAAC)= 0xAA55AA55;/* Latch 32 MSB data */
MCR   = 0x00000011;/* Set EHV in MCR: Operation Start */
```

```
do              /* Loop to wait for DONE=1 */
{ tmp= MCR;     /* Read MCR */
} while ( !(tmp & 0x00000400) );
status= MCR & 0x00000200;/* Check PEG flag */
MCR   = 0x00000010;/* Reset EHV in MCR: Operation End */
MCR              = 0x00000000;                    /* Reset PGM in MCR:
Deselect Operation */
```

17.3.8.1.2 Sector erase

Erase changes the value stored in all bits of the selected block(s) to logic 1.

An erase sequence operates on any combination of blocks (sectors) in the low, mid or high address space, or the shadow block (if available). The test block cannot be erased.

The erase sequence is fully automated within the Flash. The user only needs to select the blocks to be erased and initiate the erase sequence.

Locked/disabled blocks cannot be erased.

If multiple blocks are selected for erase during an erase sequence, no specific operation order must be assumed.

The Erase operation consists of the following sequence of events:

1. Change the value in the MCR[ERS] bit from 0 to 1.
2. Select the block(s) to be erased by writing 1s to the LMS register. If the shadow block is to be erased, this step may be skipped, and LMS is ignored.

Note: Lock and Select are independent. If a block is selected and locked, no erase will occur.

3. Write to any address in Flash. This is referred to as an erase interlock write.
4. Set MCR[EHV] to start the internal erase sequence, or skip to step *9.* to terminate.
5. Wait until the MCR[DONE] bit goes high.
6. Confirm MCR[PEG] = 1.
7. Clear the MCR[EHV] bit.
8. If more blocks are to be erased, return to step *2.*.
9. Write a logic 0 to the MCR[ERS] bit to terminate the erase operation.

After setting MCR[ERS], one write, referred to as an interlock write, must be performed before MCR[EHV] can be set to 1. Data words written during erase sequence interlock writes are ignored.

The user may terminate the erase sequence by clearing MCR[ERS] before setting MCR[EHV].

An erase operation may be terminated by clearing MCR[EHV], assuming MCR[DONE] is low, MCR[EHV] is high, and MCR[ESUS] is low.

An erase termination forces the Module to step *8.* of the erase sequence.

A terminated erase results in MCR[PEG] being cleared, indicating a failed operation. MCR[DONE] must be checked to know when the terminating command has completed.

The block(s) being operated on before the termination contain indeterminate data. This may be recovered by executing an erase on the affected blocks.

The user may not terminate an erase sequence while in erase suspend.

Example 2 Erase of sectors B0F1 and B0F2

```
MCR   = 0x00000004;/* Set ERS in MCR: Select Operation */
LMS   = 0x00000006;/* Set LSL2-1 in LMS: Select Sectors to erase */
(0x000000)= 0xFFFFFFFF;/* Latch a Flash Address with any data */
MCR   = 0x00000005;/* Set EHV in MCR: Operation Start */
do            /* Loop to wait for DONE=1 */
{ tmp= MCR;  /* Read MCR */
} while ( !(tmp & 0x00000400) );
status= MCR & 0x00000200;/* Check PEG flag */
MCR   = 0x00000004;/* Reset EHV in MCR: Operation End */
MCR               = 0x00000000;              /* Reset ERS in MCR: Deselect
Operation */
```

The erase sequence may be suspended to allow read access to the Flash core.

It is not possible to program or to erase during an erase suspend.

During erase suspend, all reads to blocks targeted for erase return indeterminate data.

An erase suspend can be initiated by changing the value of the MCR[ESUS] bit from 0 to 1. MCR[ESUS] can be set to 1 at any time when MCR[ERS] and MCR[EHV] are high and MCR[PGM] is low. A 0-to-1 transition of MCR[ESUS] causes the module to start the sequence that places it in erase suspend.

The user must wait until MCR[DONE] = 1 before the Module is suspended and further actions are attempted. MCR[DONE] will go high no more than t_{ESUS} after MCR[ESUS] is set to 1.

Once suspended, the array may be read. Flash core reads while MCR[ESUS] = 1 from the block(s) being erased return indeterminate data.

Example 3 Sector Erase Suspend

```
MCR   = 0x00000007;/* Set ESUS in MCR: Erase Suspend */
do              /* Loop to wait for DONE=1 */
{ tmp= MCR;     /* Read MCR */
} while ( !(tmp & 0x00000400) );
```

Notice that there is no need to clear MCR[EHV] and MCR[ERS] in order to perform reads during erase suspend.

The Erase sequence is resumed by writing a logic 0 to MCR[ESUS].

MCR[EHV] must be set to 1 before MCR[ESUS] can be cleared to resume the operation.

The Module continues the erase sequence from one of a set of predefined points. This may extend the time required for the erase operation.

Example 4 Sector Erase Resume

```
MCR               = 0x00000005;                    /* Reset ESUS in MCR:
Erase Resume */
```

17.3.8.1.3 User Test mode

User Test mode is a customer-accessible mode that can be used to perform specific tests to check the integrity of the Flash module. Three kinds of test can be performed:

- Array integrity self-check
- Margin mode read
- ECC logic check

The User Test mode is equivalent to a Modify operation. Read accesses attempted by the user during User Test mode generate a Read-While-Write Error (the MCR[RWE] bit is set).

User Test operations are not allowed on the Test and Shadow blocks.

17.3.8.1.3.1 Array integrity self check

Array integrity is checked using a predefined address sequence (proprietary), and this operation is executed on selected and unlocked blocks. Once the operation is completed, the results of the reads can be checked by reading the MISR value (stored in UMISR0–4), to determine if an incorrect read, or ECC detection was noted.

The internal MISR calculator is a 32-bit register.

The 128-bit data, the 16-bit ECC data, and the single and double ECC errors of the two double words are therefore captured by the MISR through five different read accesses at the same location.

The whole check is done through five complete scans of the memory address space:

1. The first pass scans only bits 31:0 of each page.
2. The second pass scans only bits 63:32 of each page.
3. The third pass scans only bits 95:64 of each page.
4. The fourth pass scans only bits 127:96 of each page.
5. The fifth pass scans only the ECC bits (8 + 8) and the single and double ECC errors (2 + 2) of both double words of each page.

The 128-bit data and the 16-bit ECC data are sampled before the eventual ECC correction, while the single and double error flags are sampled after the ECC evaluation.

Only data from existing and unlocked locations are captured by the MISR.

The MISR can be seeded to any value by writing the UMISR0–4 registers.

The Array Integrity Self Check consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the LMS register.

Note: Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set eventually UT0[AIS] bit for a sequential addressing only.
4. Write a logic 1 to the UT0[AIE] bit to start the Array Integrity Check.
5. Wait until the UT0[AID] bit is set.
6. Compare the contents of the UMISR0–4 registers with the expected results.
7. Write a logic 0 to the UT0[AIE] bit.
8. If more blocks are to be checked, return to step 2.

It is recommended to leave UT0[AIS] at 0 and use the proprietary address sequence that checks the read path more fully, although this sequence takes more time.

Note: During the execution of the Array Integrity Check operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

While UT0[AID] is low and UT0[AIE] is high, the user may clear UT0[AIE], resulting in an Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 5 Array Integrity Check of sectors B0F1 and B0F2

```
UT0   = 0xF9F99999;/* Set UTE in UT0: Enable User Test */
LMS   = 0x00000006;/* Set LSL2-1 in LMS: Select Sectors */
UT0   = 0x80000002;/* Set AIE in UT0: Operation Start */
do            /* Loop to wait for AID=1 */
{ tmp= UT0;  /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0= UMISR0;/* Read UMISR0 content*/
data1= UMISR1;/* Read UMISR1 content*/
data2= UMISR2;/* Read UMISR2 content*/
data3= UMISR3;/* Read UMISR3 content*/
data4= UMISR4;/* Read UMISR4 content*/
UT0               = 0x00000000;              /* Reset UTE and AIE in UT0:
Operation End */
```

17.3.8.1.3.2 Margin read

The Margin read procedure (either Margin 0 or Margin 1) can be run on unlocked blocks in order to unbalance the Sense Amplifiers with respect to standard read conditions so that all the read accesses reduce the margin vs. 0 (UT0[MRV] = 0) or vs. 1 (UT0[MRV] = 1). Locked sectors are ignored by MISR calculation and ECC flagging.

The results of the margin reads can be checked by comparing the checksum value in UMISR[0:4].

Since Margin reads are done at voltages that differ than the normal read voltage, the lifetime expectancy of the Flash macrocell is impacted by the execution of Margin reads.

Repeated Margin reads will result in degradation of the Flash array, and will shorten the expected lifetime experienced at normal read levels.

For these reasons, Margin reads are allowed only at the factory. Margin reads are forbidden for use by user applications.

In any case the charge losses detected through the Margin mode cannot be considered failures of the device and no failure analysis will be opened on them.

The Margin Read Setup operation consists of the following sequence of events:

1. Set UTE in UT0 by writing the related password in UT0.
2. Select the block(s) to be checked by writing 1s to the LMS register.

Note: Note that Lock and Select are independent. If a block is selected and locked, no Array Integrity Check will occur.

3. Set UT0[AIS] bit for a sequential addressing only.
4. Change the value in the UT0[MRE] bit from 0 to 1.
5. Select the Margin level: UT0[MRV] = 0 for 0s margin, UT0[MRV] = 1 for 1s margin.
6. Write a logic 1 to the UT0[AIE] bit to start the Margin Read.
7. Wait until the UT0[AID] bit goes high.
8. Compare UMISR[0:4] content with the expected result.
9. Write a logic 0 to the UT0[AIE], UT0[MRE] and UT0[MRV] bits.
10. If more blocks are to be checked, return to step *2.*.

It is mandatory to leave UT0[AIS] at 1 and use the linear address sequence (that also takes less time).

During the execution of the Margin Mode operation it is forbidden to modify the content of Block Select (LMS) and Lock (LML, SLL) registers, otherwise the MISR value can vary in an unpredictable way.

The read accesses will be done with the addition of a proper number of wait states to guarantee the correctness of the result.

While UT0[AID] is low and UT0[AIE] is high, the user may clear AIE, resulting in a Array Integrity Check termination.

UT0[AID] must be checked to know when the terminating command has completed.

Example 6 Margin Read Check versus 1s

```
UT0 = 0xF9F99999; /* Set UTE in UT0: Enable User Test */
LMS = 0x00000006; /* Set LSL2-1 in LMS: Select Sectors */
UT0 = 0x80000004; /* Set AIS in UT0: Select Operation */
UT0 = 0x80000024; /* Set MRE in UT0: Select Operation */
UT0 = 0x80000034; /* Set MRV in UT0: Select Margin versus 1’s */
UT0 = 0x80000036; /* Set AIE in UT0: Operation Start */
do /* Loop to wait for AID=1 */
{ tmp = UT0; /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0 = UMISR0; /* Read UMISR0 content*/
data1 = UMISR1; /* Read UMISR1 content*/
data2 = UMISR2; /* Read UMISR2 content*/
data3 = UMISR3; /* Read UMISR3 content*/
data4 = UMISR4; /* Read UMISR4 content*/
UT0 = 0x80000034; /* Reset AIE in UT0: Operation End */
UT0 = 0x00000000; /* Reset UTE, MRE, MRV, AIS in UT0: Deselect Op. */
```

17.3.8.1.3.3 ECC logic check

ECC logic can be checked by forcing the input of ECC logic: The 64 bits of data and the 8 bits of ECC syndrome can be individually forced and they will drive simultaneously at the same value the ECC logic of the whole page (2 double words).

The results of the ECC Logic Check can be verified by reading the MISR value.

The ECC Logic Check operation consists of the following sequence of events:

1. Set UT0[UTE] by writing the related password in UT0.
2. Write the double word input value to UT1[DAI31–0] and UT2[DAI[63–32].
3. Write the Syndrome Input value to UT0[DSI7–0].
4. Select the ECC Logic Check: write a logic 1 to the UT0[EIE] bit.
5. Write a logic 1 to the UT0[AIE] bit to start the ECC Logic Check.
6. Wait until the UT0[AID] bit goes high.
7. Compare the contents of the UMISR0–4 registers with the expected results.
8. Write a logic 0 to the UT0[AIE] bit.

Notice that when UT0[AID] = 0, the UMISR0–4 and UT1–2 registers and the UT0[MRE], UT0[MRV], UT0[EIE], UT0[AIS], and UT0[DSI7–0] bits are not accessible. Reads return indeterminate data; writes have no effect.

Example 7 ECC logic check

```
UT0   = 0xF9F99999;/* Set UTE in UT0: Enable User Test */
UT1   = 0x55555555;/* Set DAI31-0 in UT1: Even Word Input Data */
UT2   = 0xAAAAAAAA;/* Set DAI63-32 in UT2: Odd Word Input Data */
UT0   = 0x80FF0000;/* Set DSI7-0 in UT0: Syndrome Input Data */
UT0   = 0x80FF0008;/* Set EIE in UT0: Select ECC Logic Check */
UT0   = 0x80FF000A;/* Set AIE in UT0: Operation Start */
do           /* Loop to wait for AID=1 */
{ tmp= UT0;  /* Read UT0 */
} while ( !(tmp & 0x00000001) );
data0= UMISR0;/* Read UMISR0 content (expected 0x55555555) */
data1= UMISR1;/* Read UMISR1 content (expected 0xAAAAAAAA) */
data2= UMISR2;/* Read UMISR2 content (expected 0x55555555) */
data3= UMISR3;/* Read UMISR3 content (expected 0xAAAAAAAA) */
data4= UMISR4;/* Read UMISR4 content (expected 0x00FF00FF) */
UT0              = 0x00000000;              /* Reset UTE, AIE and EIE in
UT0: Operation End */
```

17.3.8.2 Error Correction Code (ECC)

The Flash macrocell provides a method to improve the reliability of the data stored in Flash: the usage of an Error Correction Code. The word size is fixed at 64 bits.

Each double word of 64 bits has an associated 8 ECC bits that are programmed in such a way to guarantee a Single Error Correction and a Double Error Detection (SEC-DED).

ECC circuitry provides correction of single bit faults and is used to achieve automotive reliability targets. Some units will experience single bit corrections throughout the life of the product with no impact on product reliability.

17.3.8.2.1 ECC algorithms

The Flash macrocell supports the ECC algorithm “All 1s No Error”.

17.3.8.2.2 All 1s No Error

The All 1s No Error algorithm detects as valid any double word read on a just erased sector (all the 72 bits are 1s).

This option allows performing a Blank Check after a Sector Erase operation.

17.3.8.3 EEPROM emulation

The chosen ECC algorithm allows some bit manipulations so that a Double Word can be rewritten several times without needing an erase of the sector. This allows to use a Double Word to store flags useful for the EEPROM emulation.

As an example the chosen ECC algorithm allows to start from an All ‘1’s Double Word value and rewrite whichever of its four 16-bit Half-Words to an All ‘0’s content by keeping the same ECC value.

Table 184 shows a set of Double Words sharing the same ECC value.

Table 184. Bits manipulation: double words with the same ECC value

Double word	ECC value – All 1s No Error
0xFFFF_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_FFFF_0000	0xFF
0xFFFF_FFFF_0000_FFFF	0xFF
0xFFFF_0000_FFFF_FFFF	0xFF
0x0000_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_0000_0000	0xFF
0xFFFF_0000_FFFF_0000	0xFF
0x0000_FFFF_FFFF_0000	0xFF
0xFFFF_0000_0000_FFFF	0xFF
0x0000_FFFF_0000_FFFF	0xFF
0x0000_0000_FFFF_FFFF	0xFF
0xFFFF_0000_0000_0000	0xFF
0x0000_FFFF_0000_0000	0xFF
0x0000_0000_0000_0000	0xFF

When some Flash sectors are used to perform an EEPROM emulation, it is recomended for safety reasons to reserve at least three sectors for this purpose.

17.3.8.4 Protection strategy

Two kinds of protection are available: Modify Protection to avoid unwanted program/erase in Flash sectors, and Censored mode to avoid piracy.

17.3.8.4.1 Modify protection

The Flash modify protection information is stored in non-volatile Flash cells located in the TestFlash. This information is read once during the Flash initialization phase following the exit from reset and they are stored in Volatile registers that act as actuators.

The reset state of all the volatile modify protection registers is the protected state.

All the non-volatile modify protection registers can be programmed through a normal double word program operation at the related locations in TestFlash.

The non-volatile modify protection registers cannot be erased.

- The non-volatile modify protection registers are physically located in TestFlash. Their bits can be programmed to 0 only once, after which they cannot be restored to 1.
- The volatile modify protection registers are read/write registers containing bits that can be written at 0 or 1 by the user application.

A software mechanism is provided to independently lock/unlock each Low, Mid, and High Address Space block against program and erase.

Software locking is done through the LML (Low/Mid Address Space Block Lock Register) register.

An alternate means to enable software locking for blocks of Low Address Space only is through the SLL (Secondary Low/Mid Address Space Block Lock Register).

All these registers have a non-volatile image stored in TestFlash (NVLML, NVSLL), so that the locking information is kept on reset.

On delivery the TestFlash non-volatile image is at all 1s, meaning all sectors are locked.

By programming the non-volatile locations in TestFlash, the selected sectors can be unlocked.

Because the TestFlash is one-time programmable (that is, not erasable), once the sectors have been unlocked, they cannot be locked again.

Of course, on the contrary, all the volatile registers can be written at 0 or 1 at any time, therefore the user application can lock and unlock sectors when desired.

17.3.8.4.2 Censored mode

The Censored mode information is stored in non-volatile Flash cells located in the Shadow Sector. This information is read once during the Flash initialization phase following the exit from Reset and they are stored in Volatile registers that act as actuators.

The reset state of all the volatile censored mode registers is the protected state.

All the non-volatile censored mode registers can be programmed through a normal double word program operation at the related locations in the Shadow Sector.

The non-volatile censored mode registers can be erased by erasing the Shadow Sector.

- The non-volatile censored mode registers are physically located in the Shadow Sector their bits can be programmed to 0 and eventually restored to 1 by erasing the Shadow Sector.
- The volatile censored mode registers are registers not accessible by the user application.

The Flash block provides two levels of protection against piracy:

- If bits NVSCI0[CW[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Censored mode is disabled. All other possible values enable Censored mode.
- If bits NVSCI0[SC[15:0]] are programmed to 0x55AA and NVSC1 = NVSCI0, Public Access is disabled. All other possible values enable Public Access.

The parts are delivered to the user with Censored mode and public access disabled.

The chosen Flash ECC algorithm allows to modify the censorship status without erasing the Shadow sector, as shown in *Table 185*.

Table 185. Bits manipulation: censorship management

Censored mode	Public access	NVSCI0	NVSCI1
Enabled	Enabled	0xFFFF_FFFF	0xFFFF_FFFF
Disabled	Enabled	0xFFFF_55AA	0xFFFF_55AA
Enabled	Disabled	0x55AA_FFFF	0x55AA_FFFF
Disabled	Disabled	0x55AA_55AA	0x55AA_55AA
Enabled	Disabled	0x55AA_0000	0x55AA_0000
Disabled	Enabled	0x0000_55AA	0x0000_55AA
Enabled	Enabled	0x0000_0000	0x0000_0000

18 Enhanced Direct Memory Access (eDMA)

18.1 Introduction

This chapter describes the enhanced Direct Memory Access (eDMA) Controller, a second-generation module capable of performing complex data transfers with minimal intervention from a host processor.

18.2 Overview

The enhanced direct memory access (eDMA) controller hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with SRAM-based local memory containing the transfer control descriptors (TCD) for the channels.

Figure 186 is a block diagram of the eDMA module.

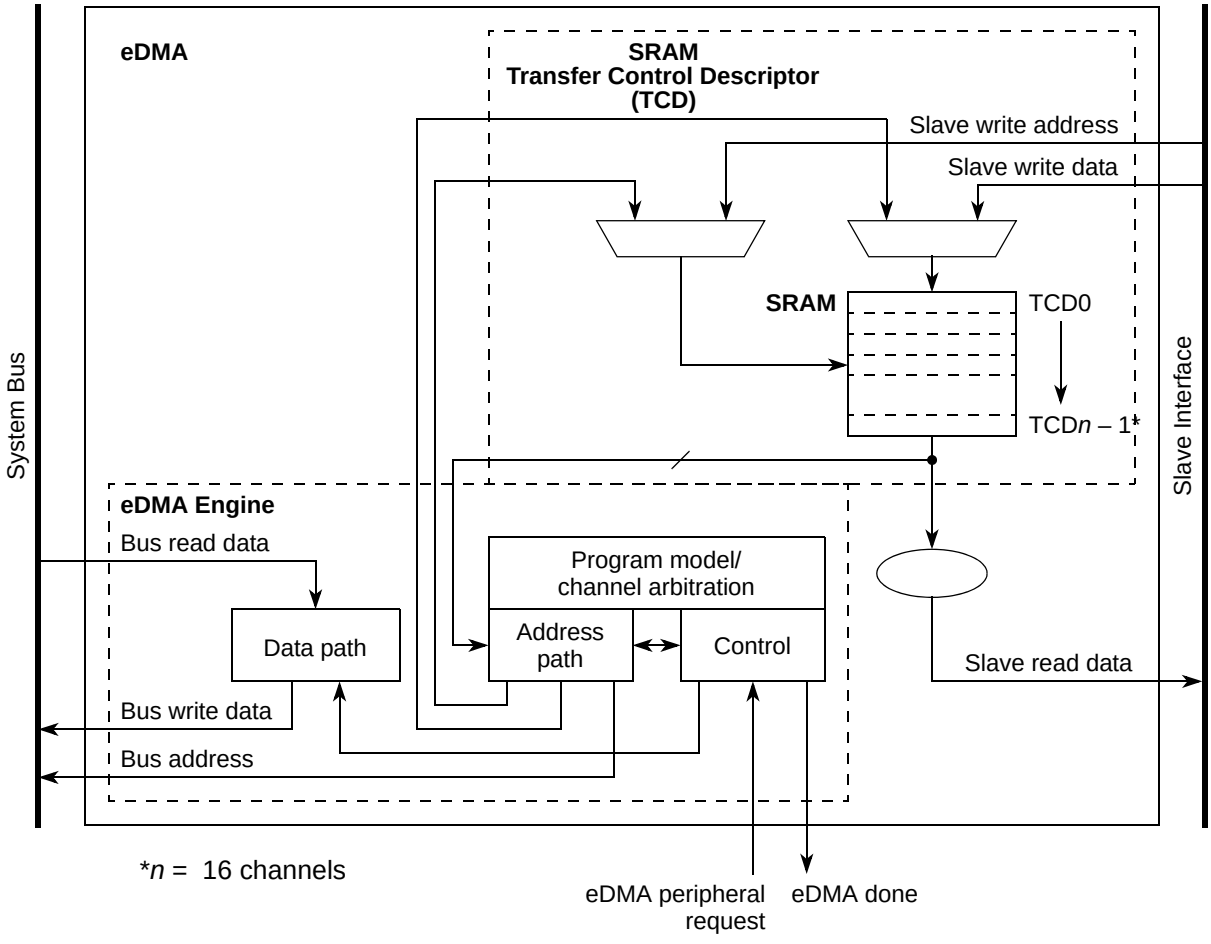


Figure 186. eDMA block diagram

18.3 Features

The eDMA is a highly programmable data transfer engine, which has been optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known, and is *not* defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination
- Programmable source, destination addresses, transfer size, plus support for enhanced addressing modes
- 16-channel implementation performs complex data transfers with minimal intervention from a host processor
  - 32 bytes of data registers, used as temporary storage to support burst transfers (refer to SSIZE bit)
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD per channel stored in local memory
  - An inner data transfer loop defined by a minor byte transfer count
  - An outer data transfer loop defined by a major iteration count
- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continual transfers
  - Peripheral-paced hardware requests (one per channel)

Note: For all three methods, one activation per execution of the minor loop is required.

- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - 1 interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are enabled per channel, and logically summed together to form a single error interrupt.
- Support for scatter/gather DMA processing
- Any channel can be programmed so that it can be suspended by a higher priority channel's activation, before completion of a minor loop.

Throughout this chapter, *n* is used to reference the channel number. Additionally, data sizes are defined as byte (8-bit), halfword (16-bit), word (32-bit) and doubleword (64-bit).

18.4 Modes of operation

18.4.1 Normal mode

In normal mode, the eDMA transfers data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

18.4.2 Debug mode

If enabled by EDMA_CR[EDBG] and the CPU enters debug mode, the eDMA does not grant a service request when the debug input signal is asserted. If the signal asserts during a data block transfer as described by a minor loop in the current active channel's TCD, the eDMA continues the operation until the minor loop completes.

18.5 Memory map and register definition

18.5.1 Memory map

The eDMA programming model is partitioned into two regions:

Region 1 defines control registers; Region 2 defines the local transfer control for the descriptor memory.

Table 186 is a 32-bit view of the eDMA memory map.

Table 186. eDMA memory map

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0000	EDMA_CR—Control Register	*on page 412*
0x0004	EDMA_ESR—eDMA Error Status Register	*on page 413*
0x0008	Reserved	
0x000C	EDMA_ERQL—eDMA Enable Request Register	*on page 415*
0x0010	Reserved	
0x0014	EDMA_EEIRL—eDMA Enable Error Interrupt Register	*on page 416*
0x0018	EDMA_SERQR—eDMA Set Enable Request Register	*on page 417*
0x0019	EDMA_CERQR—eDMA Clear Enable Request Register	*on page 418*
0x001A	EDMA_SEEI—eDMA Set Enable Error Interrupt Register	*on page 418*
0x001B	EDMA_CEEI—eDMA Clear Enable Error Interrupt Register	*on page 419*
0x001C	EDMA_CIRQR—eDMA Clear Interrupt Request Register	*on page 419*
0x001D	EDMA_CER—eDMA Clear Error Register	*on page 420*
0x001E	EDMA_SSBR—eDMA Set START Bit Register	*on page 420*
0x001F	EDMA_CDSBR—eDMA Clear DONE Status Register	*on page 421*
0x0020	Reserved	
0x0024	EDMA_IRQRL—eDMA Interrupt Request Register	*on page 421*
0x0028	Reserved	
0x002C	EDMA_ERL—eDMA Error Register	*on page 422*
0x0030	Reserved	

Table 186. eDMA memory map(Continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x0034	EDMA_HRSL—eDMA Hardware Request Status Register	*on page 423*
0x0038–0x00FF	Reserved	
0x0100	EDMA_CPR0—eDMA Channel 0 Priority Register	*on page 424*
0x0101	EDMA_CPR1—eDMA Channel 1 Priority Register	*on page 424*
0x0102	EDMA_CPR2—eDMA Channel 2 Priority Register	*on page 424*
0x0103	EDMA_CPR3—eDMA Channel 3 Priority Register	*on page 424*
0x0104	EDMA_CPR4—eDMA Channel 4 Priority Register	*on page 424*
0x0105	EDMA_CPR5—eDMA Channel 5 Priority Register	*on page 424*
0x0106	EDMA_CPR6—eDMA Channel 6 Priority Register	*on page 424*
0x0107	EDMA_CPR7—eDMA Channel 7 Priority Register	*on page 424*
0x0108	EDMA_CPR8—eDMA Channel 8 Priority Register	*on page 424*
0x0109	EDMA_CPR9—eDMA Channel 9 Priority Register	*on page 424*
0x010A	EDMA_CPR10—eDMA Channel 10 Priority Register	*on page 424*
0x010B	EDMA_CPR11—eDMA Channel 11 Priority Register	*on page 424*
0x010C	EDMA_CPR12—eDMA Channel 12 Priority Register	*on page 424*
0x010D	EDMA_CPR13—eDMA Channel 13 Priority Register	*on page 424*
0x010E	EDMA_CPR14—eDMA Channel 14 Priority Register	*on page 424*
0x010F	EDMA_CPR15—eDMA Channel 15 Priority Register	*on page 424*
0x0110–0x0FFF	Reserved	
0x1000	TCD00—Transfer Control Descriptor 0	*on page 425*
0x1020	TCD01—Transfer Control Descriptor 1	*on page 425*
0x1040	TCD02—Transfer Control Descriptor 2	*on page 425*
0x1060	TCD03—Transfer Control Descriptor 3	*on page 425*
0x1080	TCD04—Transfer Control Descriptor 4	*on page 425*
0x10A0	TCD05—Transfer Control Descriptor 5	*on page 425*
0x10C0	TCD06—Transfer Control Descriptor 6	*on page 425*
0x10E0	TCD07—Transfer Control Descriptor 7	*on page 425*
0x1100	TCD08—Transfer Control Descriptor 8	*on page 425*
0x1120	TCD09—Transfer Control Descriptor 9	*on page 425*
0x1140	TCD10—Transfer Control Descriptor 10	*on page 425*
0x1160	TCD11—Transfer Control Descriptor 11	*on page 425*

Table 186. eDMA memory map(Continued)

Offset from EDMA_BASE (0xFFF4_4000)	Register	Location
0x1180	TCD12—Transfer Control Descriptor 12	*on page 425*
0x11A0	TCD13—Transfer Control Descriptor 13	*on page 425*
0x11C0	TCD14—Transfer Control Descriptor 14	*on page 425*
0x11E0	TCD15—Transfer Control Descriptor 15	*on page 425*
0x1200–0x3FFF	Reserved	

18.5.2 Register descriptions

Read operations on reserved bits in a register return undefined data. Do not write operations to reserved bits. Writing to reserved bits in a register can generate errors. The maximum register bit-width for this device is 16 bits wide.

18.5.2.1 eDMA Control Register (EDMA_CR)

The 32-bit EDMA_CR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in one group of 16 channels.

Arbitration can be configured to use either fixed-priority or round-robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. In round-robin arbitration mode, the channel priorities are ignored and the channels are cycled through, from channel 15 down to channel 0, without regard to priority.

Refer to *Section 18.5.2.16: eDMA Channel n Priority Registers (EDMA_CPRn)*.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	ERCA	EDBG	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 187. eDMA Control Register (EDMA_CR)

Table 187. EDMA_CR field descriptions

Field	Description
0-28	Reserved.
29 ERCA	Enable round-robin channel arbitration. 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
30 EDBG	Enable debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.
31	Reserved.

18.5.2.2 eDMA Error Status Register (EDMA_ESR)

The EDMA_ESR provides information concerning the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed arbitration mode, a configuration error is caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. For either type of priority configuration error, the ERRCHN field is undefined. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled.

If a scatter/gather operation is enabled upon channel completion, a configuration error is reported if the scatter/gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled upon channel completion, a configuration error is reported when the link is attempted if the TCD.CITER.E_LINK bit does not equal the TCD.BITER.E_LINK bit. All configuration error conditions except scatter/gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write executes using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence executes before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the eDMA engine to stop the active channel, and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected. After the error status has been updated, the eDMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel executes and terminates with the same error condition.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 188. eDMA Error Status Register (EDMA_ESR)

Table 188. EDMA_ESR field descriptions

Field	Description
0 VLD	Logical OR of all EDMA_ERH and EDMA_ERL status bits. 0 No EDMA_ER bits are set. 1 At least one EDMA_ER bit is set indicating a valid error exists that has not been cleared.
1–16	Reserved.
17 CPE	Channel priority error. 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
18–23 ERRCHN[0:5]	Error channel number. Channel number of the last recorded error (excluding CPE error). **Note:** Do not rely on the number in the ERRCHN field for group and channel priority errors. Group and channel priority errors need to be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
24 SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCD.SADDR field, indicating TCD.SADDR is inconsistent with TCD.SSIZE.
25 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.SOFF field, indicating TCD.SOFF is inconsistent with TCD.SSIZE.

Table 188. EDMA_ESR field descriptions(Continued)

Field	Description
26 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCD.DADDR field, indicating TCD.DADDR is inconsistent with TCD.DSIZE.
27 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCD.DOFF field, indicating TCD.DOFF is inconsistent with TCD.DSIZE.
28 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD.NBYTES or TCD.CITER fields, indicating the following conditions exist: – TCD.NBYTES is not a multiple of TCD.SSIZE and TCD.DSIZE, or – TCD.CITER is equal to zero, or – TCD.CITER.E_LINK is not equal to TCD.BITER.E_LINK.
29 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD.DLAST_SGA field, indicating TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD.E_SG is enabled.
30 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
31 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

18.5.2.3 eDMA Enable Request Register (EDMA_ERQRL)

The EDMA_ERQRL provides a bit map for the 16 implemented channels to enable the request signal for each channel. EDMA_ERQRL maps to channels 15–0.

The state of any given channel enable is directly affected by writes to this register; the state is also affected by writes to the EDMA_SERQR and EDMA_CERQR. The EDMA_CERQR and EDMA_SERQR are provided so that the request enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_ERQRL.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does *not* affect a channel service request made explicitly through software or a linked channel request.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 189. eDMA Enable Request Low Register (EDMA_ERQRL)

Table 189. EDMA_ERQRL field descriptions

Field	Description
16–31 ERQ*n*	Enable DMA hardware service request *n.* 0 The DMA request signal for channel n is disabled. 1 The DMA request signal for channel n is enabled.

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that can affect the ending state of the EDMA_ERQR bit for that channel. If the TCD.D_REQ bit is set, then the corresponding EDMA_ERQR bit is cleared after the major loop is complete, disabling the DMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_ERQR bit is unaffected.

18.5.2.4 eDMA Enable Error Interrupt Register (EDMA_EEIRL)

The EDMA_EEIRL provides a bit map for the 16 channels to enable the error interrupt signal for each channel. EDMA_EEIRL maps to channels 15-0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_SEEIR and EDMA_CEEIR. The EDMA_SEEIR and EDMA_CEEIR are provided so that the error interrupt enable for a *single* channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_EEIRL.

Both the DMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI09	EEI08	EEI07	EEI06	EEI05	EEI04	EEI03	EEI02	EEI01	EEI00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 190. eDMA Enable Error Interrupt Low Register (EDMA_EEIRL)

Table 190. EDMA_EEIRL field descriptions

Field	Description
16-31 EEI*n*	Enable error interrupt *n.* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

18.5.2.5 eDMA Set Enable Request Register (EDMA_SERQR)

The EDMA_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_ERQRL to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be set. Setting bit 1 (SERQ*n*) provides a global set function, forcing the entire contents of EDMA_ERQRL to be asserted. Reads of this register return all zeroes.

Address: Base + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 191. eDMA Set Enable Request Register (EDMA_SERQR)

Table 191. EDMA_SERQR field descriptions

Field	Descriptions
0	Reserved.
1–7 SERQ[0:6]	Set enable request. 0–15 Set corresponding bit in EDMA_ERQRL 16–63 Reserved 64–127 Set all bits in EDMA_ERQRL **Note:** Bit 2 (SERQ1) is not used.

18.5.2.6 eDMA Clear Enable Request Register (EDMA_CERQR)

The EDMA_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQRL to be cleared. Setting bit 1 (CERQ*n*) provides a global clear function, forcing the entire contents of the EDMA_ERQRL to be zeroed, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: Base + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 192. eDMA Clear Enable Request Register (EDMA_CERQR)

Table 192. EDMA_CERQR field descriptions

Field	Description
0	Reserved.
1–7 CERQ[0:6]	Clear enable request. 0–15 Clear corresponding bit in EDMA_ERQRL 16–63Reserved 64–127Clear all bits in EDMA_ERQRL **Note:** Bit 2 (CERQ1) is not used.

18.5.2.7 eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

The EDMA_SEEIR provides a simple memory-mapped mechanism to set a given bit in the EDMA_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be set. Setting bit 1 (SEEI*n*) provides a global set function, forcing the entire contents of EDMA_EEIRL to be asserted. Reads of this register return all zeroes.

Address: Base + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 193. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Table 193. EDMA_SEEIR field descriptions

Field	Description
0	Reserved.
1–7 SEEI[0:6]	Set enable error interrupt. 0–15 Set corresponding bit in EDMA_EIRRL 16–63 Reserved 64–127 Set all bits in EDMA_EEIRL **Note:** Bit 2 (SEEI1) is not used.

18.5.2.8 eDMA Clear Enable Error Interrupt Register (EDMA_CEEIR)

The EDMA_CEEIR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEIRL to be cleared. Setting bit 1 (CEEI*n*) provides a global clear function, forcing the entire contents of the EDMA_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

Address: Base + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 194. eDMA Set Enable Error Interrupt Register (EDMA_SEEIR)

Table 194. EDMA_CEEIR field descriptions

Field	Description
0	Reserved.
1–7 CEEI[0:6]	Clear enable error interrupt. 0–15 Clear corresponding bit in EDMA_EEIRL 16–63 Reserved 64–127 Clear all bits in EDMA_EEIRL **Note:** Bit 2 (CEEI1) is not used.

18.5.2.9 eDMA Clear Interrupt Request Register (EDMA_CIRQR)

The EDMA_CIRQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_IRQRL to be cleared. Setting bit 1 (CINT*n*) provides a global clear function, forcing the entire contents of the EDMA_IRQRL to be zeroed, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Address: Base + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 195. eDMA Clear Interrupt Request (EDMA_CIRQR)

Table 195. EDMA_CIRQR field descriptions

Field	Description
0	Reserved.
1–7 CINT[0:6]	Clear interrupt request. 0–15 Clear corresponding bit in EDMA_IRQRL 16–63 Reserved 64–127 Clear all bits in EDMA_IRQRL **Note:** Bit 2 (CINT1) is not used.

18.5.2.10 eDMA Clear Error Register (EDMA_CER)

The EDMA_CER provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_ERL to be cleared. Setting bit 1 (CER*n*) provides a global clear function, forcing the entire contents of the EDMA_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

Address: Base + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CER[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 196. eDMA Clear Error Register (EDMA_CER)

Table 196. EDMA_CER field descriptions

Field	Description
0	Reserved.
1–7 CER[0:6]	Clear error indicator. 0–15 Clear corresponding bit in EDMA_ERL 16–63 Reserved 64–127 Clear all bits in EDMA_ERL

18.5.2.11 eDMA Set START Bit Register (EDMA_SSBR)

The EDMA_SSBR provides a simple memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in

the corresponding transfer control descriptor to be set. Setting bit 1 (SSB*n*) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

Address: Base + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 197. eDMA Set START Bit Register (EDMA_SSBR)

Table 197. EDMA_SSBR field descriptions

Field	Description
0	Reserved.
1–7 SSB[0:6]	Set START bit (channel service request). 0–15 Set the corresponding channel's TCD START bit 16–63 Reserved 64–127 Set all TCD START bits **Note:** Bit 2 (SSB1) is not used.

18.5.2.12 eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

The EDMA_CDSBR provides a simple memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB*n*) provides a global clear function, forcing all DONE bits to be cleared.

Address: Base + 0x001F Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W		CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 198. eDMA Clear DONE Status Bit Register (EDMA_CDSBR)

Table 198. EDMA_CDSBR field descriptions

Field	Description
0	Reserved.
1–7 CDSB[0:6]	Clear DONE status bit. 0–15 Clear the corresponding channel's DONE bit 16–63 Reserved 64–127 Clear all TCD DONE bits **Note:** Bit 2 (CDSB1) is not used.

18.5.2.13 eDMA Interrupt Request Register (EDMA_IRQRL)

The EDMA_IRQRL provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. EDMA_IRQRL maps to channels 15–0.

The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, it is software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_CIRQR. On writes to the EDMA_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA_CIRQR is provided so the interrupt request for a *single* channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA_IRQRL.

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 199. eDMA Interrupt Request Low Register (EDMA_IRQRL)

Table 199. EDMA_IRQRL field descriptions

Field	Description
16–31 INT*n*	eDMA interrupt request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

18.5.2.14 eDMA Error Register (EDMA_ERL)

The EDMA_ERL provides a bit map for the 16 channels signaling the presence of an error for each channel. EDMA_ERL maps to channels 15-0.

The eDMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_EEIRL, then logically summed across groups of 16 and 32 channels to form several group error interrupt requests that are then routed to the interrupt controller. During the execution of the interrupt service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_CER in the interrupt service routine is used for this purpose. Recall the normal DMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are *not* affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_EEIRL. The EDMA_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also

affected by writes to the EDMA_CER. On writes to EDMA_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no affect on the corresponding channel's current error status. The EDMA_CER is provided so the error indicator for a *single* channel can easily be cleared.

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 200. eDMA Error Low Register (EDMA_ERL)

Table 200. EDMA_ERL field descriptions

Field	Description
16–31 ERR*n*	eDMA Error *n.* 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

18.5.2.15 eDMA Hardware Request Status (EDMA_HRSL)

The EDMA_HRSL registers provide a bit map for the implemented channels 16 to show the current hardware request status for each channel. Hardware request status reflects the current state of the enabled hardware requests as seen by the eDMA's arbitration logic. This view into the hardware request signals may be used for debug purposes.

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HRS 15	HRS 14	HRS 13	HRS 12	HRS 11	HRS 10	HRS 09	HRS 08	HRS 07	HRS 06	HRS 05	HRS 04	HRS 03	HRS 02	HRS 01	HRS 00
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 201. EDMA Hardware Request Status Register Low (EDMA_HRSL)

Table 201. EDMA_HRSL field descriptions

Field	Description
16–31 HRSn	DMA Hardware Request Status 0 A hardware service request for channel n is not present. 1 A hardware service request for channel n is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

18.5.2.16 eDMA Channel n Priority Registers (EDMA_CPRn)

When the fixed-priority channel arbitration mode is enabled (EDMA_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software chooses to modify channel priority values, then the software must ensure that the channel priorities contain unique values, otherwise a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_CPR*n* registers. The group priority is assigned in the EDMA_CR.

Refer to *Figure 187* and *Table 187* for the EDMA_CR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_CPR*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all of its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel is requesting service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes.

Address: Base + 0x100 + *n* Access: User read/write

	0	1	2	3	4	5	6	7
R	ECP	0	0	0	CHPRI			
W								
Reset	0	0	0	0	— [1]			

1 The reset value for the channel priority fields, GRPPRI[0–1] and CHPRI[0–3] is the channel number for the priority register;
EDMA_CPR15[CHPRI] = 0b1111.

Figure 202. eDMA Channel *n* Priority Register (EDMA_CPR*n*)

The following table describes the fields in the eDMA channel *n* priority register:

Table 202. EDMA_CPR*n* field descriptions

Field	Description
0 ECP	Enable channel preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
1-3	Reserved.
4–7 CHPRI[0:3]	Channel *n* arbitration priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; that is, EDMA_CPR31[CHPRI] = 0b1111.

18.5.2.17 Transfer Control Descriptor (TCD)

Each channel requires a 256-bit transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel 1,... channel 15. The definitions of the TCD are presented as 23 variable-length fields.

Table 203 defines the fields of the basic TCD structure.

Table 203. TCD*n* 32-bit memory structure

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 0	32	Source address	SADDR	Word 0
0x1000 + (32 × *n*) + 32	5	Source address modulo	SMOD	Word 1
0x1000 + (32 × *n*) + 37	3	Source data transfer size	SSIZE	
0x1000 + (32 × *n*) + 40	5	Destination address modulo	DMOD	
0x1000 + (32 × *n*) + 45	3	Destination data transfer size	DSIZE	
0x1000 + (32 × *n*) + 48	16	Signed Source Address Offset	SOFF	
0x1000 + (32 × *n*) + 64	32	Inner minor byte count	NBYTES	Word 2
0x1000 + (32 × *n*) + 96	32	Last Source Address Adjustment	SLAST	Word 3
0x1000 + (32 × *n*) + 128	32	Destination Address	DADDR	Word 4
0x1000 + (32 × *n*) + 160	1	Channel-to-channel Linking on Minor Loop Complete	CITER.E_LINK	Word 5
0x1000 + (32 × *n*) + 161	6	Current Major Iteration Count or Link Channel Number	CITER or CITER.LINKCH	
0x1000 + (32 × *n*) + 167	9	Current Major Iteration Count	CITER	
0x1000 + (32 × *n*) + 176	16	Destination Address Offset (Signed)	DOFF	
0x1000 + (32 × *n*) + 192	32	Last Destination Address Adjustment / Scatter Gather Address	DLAST_SGA	Word 6

Table 203. TCD*n* 32-bit memory structure(Continued)

eDMA Bit Offset	Bit Length	TCD*n* Field Name	TCD*n* Abbreviation	Word #
0x1000 + (32 × *n*) + 224	1	Channel-to-channel Linking on Minor Loop Complete	BITER.E_LINK	Word 7
0x1000 + (32 × *n*) + 225	6	Starting Major Iteration Count or Link Channel Number	BITER or BITER.LINKCH	
0x1000 + (32 × *n*) + 231	9	Starting Major Iteration Count	BITER	
0x1000 + (32 × *n*) +240	2	Bandwidth Control	BWC	
0x1000 + (32 × *n*) + 242	6	Link Channel Number	MAJOR.LINKCH	
0x1000 + (32 × *n*) + 248	1	Channel Done	DONE	
0x1000 + (32 × *n*) + 249	1	Channel Active	ACTIVE	
0x1000 + (32 × *n*) + 250	1	Channel-to-channel Linking on Major Loop Complete	MAJOR.E_LINK	
0x1000 + (32 × *n*) + 251	1	Enable Scatter/Gather Processing	E_SG	
0x1000 + (32 × *n*) + 252	1	Disable Request	D_REQ	
0x1000 + (32 × *n*) + 253	1	Channel Interrupt Enable When Current Major Iteration Count is Half Complete	INT_HALF	
0x1000 + (32 × *n*) + 254	1	Channel Interrupt Enable When Current Major Iteration Count Complete	INT_MAJ	
0x1000 + (32 × *n*) + 255	1	Channel Start	START	

Figure 203 defines the fields of the TCD*n* structure.

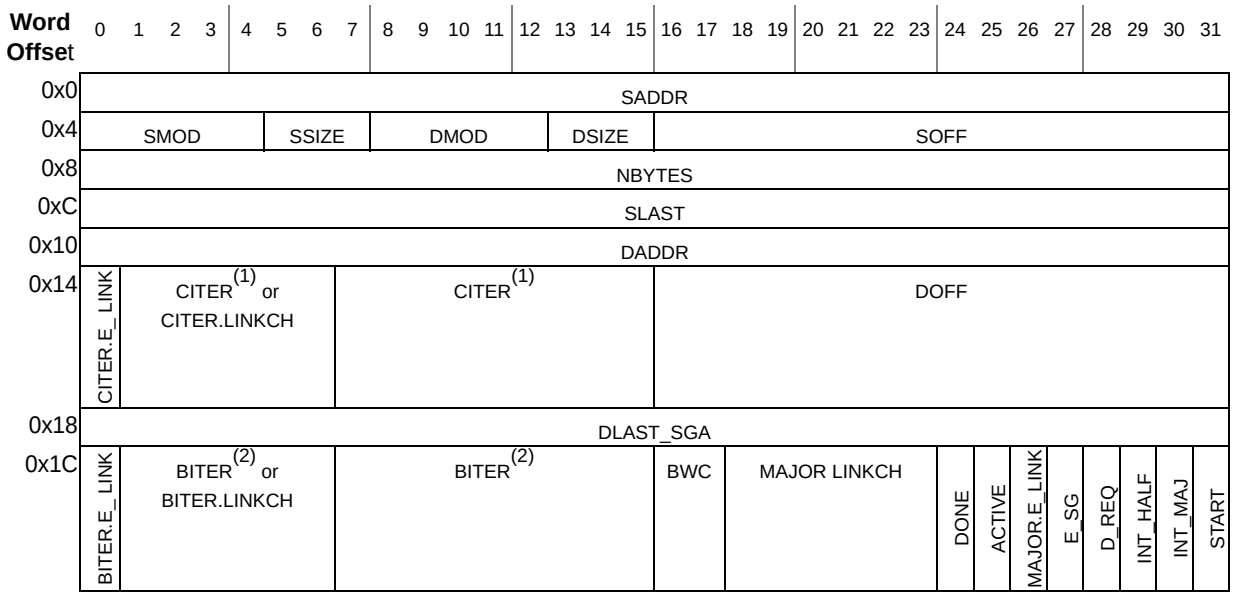


Figure 203. TCD structure

1. If channel linking on minor link completion is disabled, TCD bits [161:175] form a 15-bit CITER field; if channel-to-channel linking is enabled, CITER becomes a 9-bit field.
2. If channel linking on minor link completion is disabled, TCD bits [225:239] form a 15-bit BITER field; if channel-to-channel linking is enabled, BITER becomes a 9-bit field.

Note: The TCD structures for the eDMA channels shown in Figure 203 are implemented in internal SRAM. These structures are not initialized at reset. Therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 204 gives a detailed description of the TCN*n* fields.

Table 204. TCD*n* field descriptions

Bits Word Offset [n:n]	Field Name	Description
0–31 0x0 [0:31]	SADDR [0:31]	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
32–36 0x4 [0:4]	SMOD [0:4]	Source address modulo. 0 Source address modulo feature is disabled. not 0 This value defines a specific address range that is specified to be either the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 “size” bytes, start the queue at a 0-modulo-size address and set the SMOD field to the value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.

Table 204. TCD*n* field descriptions(Continued)

Bits Word Offset [n:n]	Field Name	Description
37–39 0x4 [5:7]	SSIZE [0:2]	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 32-bit 101 32-byte burst (64-bit x 4) 110 Reserved 111 Reserved The attempted specification of a ‘reserved’ encoding causes a configuration error.
40–44 0x4 [8:12]	DMOD [0:4]	Destination address modulo. Refer to the SMOD[0:5] definition.
45–47 0x4 [13:15]	DSIZE [0:2]	Destination data transfer size. Refer to the SSIZE[0:2] definition.
48–63 0x4 [16:31]	SOFF [0:15]	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
64–95 0x8 [0:31]	NBYTES [0:31]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a four GB transfer.
96–127 0xC [0:31]	SLAST [0:31]	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
128–159 0x10 [0:31]	DADDR [0:31]	Destination address. Memory address pointing to the destination data.
160 0x14 [0]	CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit otherwise a configuration error is reported.

Table 204. TCD*n* field descriptions(Continued)

Bits Word Offset [n:n]	Field Name	Description
161–166 0x14 [1:6]	CITER [0:5] or CITER.LINKCH [0:5]	Current “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.CITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] form a 15-bit CITER field. otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s TCD.START bit.
167–175 0x14 [7:15]	CITER [6:14]	Current “major” iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
176–191 0x14 [16:31]	DOFF [0:15]	Destination address signed offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
192–223 0x18 [0:31]	DLAST_SGA [0:31]	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If scatter/gather processing for the channel is disabled (TCD.E_SG = 0) then – Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to “restore” the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise – This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter/gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.

Table 204. TCD*n* field descriptions(Continued)

Bits Word Offset [n:n]	Field Name	Description
224 0x1C [0]	BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
225–230 0x1C [1:6]	BITER [0:5] or BITER.LINKCH [0:5]	Beginning or starting “major” iteration count or link channel number. If channel-to-channel linking is disabled (TCD.BITER.E_LINK = 0), then – No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] form a 15-bit BITER field. Otherwise – After the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel’s TCD.START bit. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
231–239 0x1C [7:15]	BITER [6:14]	Beginning or starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER must be 0x0001.
240–241 0x1C [16:17]	BWC [0:1]	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). To minimize start-up latency, bandwidth control stalls are suppressed for the first two system bus cycles and after the last write of each minor loop. 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for four cycles after each r/w 11 eDMA engine stalls for eight cycles after each r/w

Table 204. TCD*n* field descriptions(Continued)

Bits Word Offset [n:n]	Field Name	Description
242–247 0x1C [18:23]	MAJOR.LINKCH [0:5]	Link channel number. If channel-to-channel linking on major loop complete is disabled (TCD.MAJOR.E_LINK = 0) then: – No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise – After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's TCD.START bit.
248 0x1C [24]	DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the eDMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the channel has begun to be processed by the eDMA engine, not when the first data transfer occurs). **Note:** This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
249 0x1C [25]	ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the inner minor loop completes or if any error condition is detected.
250 0x1C [26]	MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the TCD.START bit of the specified channel. NOTE: To support the dynamic linking coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
251 0x1C [27]	E_SG	Enable scatter/gather processing. As the channel completes the outer major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure that is loaded as the transfer control descriptor into the local memory. NOTE: To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCD.DONE bit is set. 0 The current channel's TCD is "normal" format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.
252 0x1C [28]	D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_ERQL bit is not affected. 1 The channel's EDMA_ERQL bit is cleared when the outer major loop is complete.

Table 204. TCD*n* field descriptions(Continued)

Bits Word Offset [n:n]	Field Name	Description
253 0x1C [29]	INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the bit in the EDMA_ERQL when the current major iteration count reaches the halfway point. The eDMA engine performs the compare (CITER == (BITER >> 1)). This halfway point interrupt request supports double-buffered (aka ping-pong) schemes, or where the processor needs an early indication of the data transfer's progress during data movement. CITER = BITER = 1 with INT_HALF enabled generates an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
254 0x1C [30]	INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
255 0x1C [31]	START	Channel start. If this flag is set, the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

18.6 Functional description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

18.6.1 eDMA microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules, as shown in the following list:

- eDMA engine
  - Address path: This module implements registered versions of two channel transfer control descriptors: channel 'x' and channel 'y,' and is responsible for all the master bus address calculations. All the implemented channels provide the exact same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

    When any other channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{x,y}. After the inner minor loop completes execution, the address

path hardware writes the new values for the TCDn.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter/gather operation.

– Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.

The address and data path modules directly support the 2-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the 2nd stage of the pipeline (the data phase).

– Program model/channel arbitration: This module implements the first section of eDMA's programming model as well as the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the Control logic).

– Control: This module provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner 'minor loop' byte count has been moved.

A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as the following:

if (SSIZE < DSIZE)

transfer size = destination transfer size (# of bytes)

else

transfer size = source transfer size (# of bytes)

Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. As an example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  – Memory controller: This logic implements the required dual-ported controller, handling accesses from both the eDMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the slave transaction is stalled. The hooks to a BIST controller for the local TCD memory are included in this module.
  – Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

18.6.2 eDMA basic data flow

The basic flow of a data transfer can be partitioned into three segments. As shown in *Figure 204*, the first segment involves the channel service request. In the diagram, this

example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed through the eDMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the eDMA engine address path channel{x,y} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel{x,y} registers.

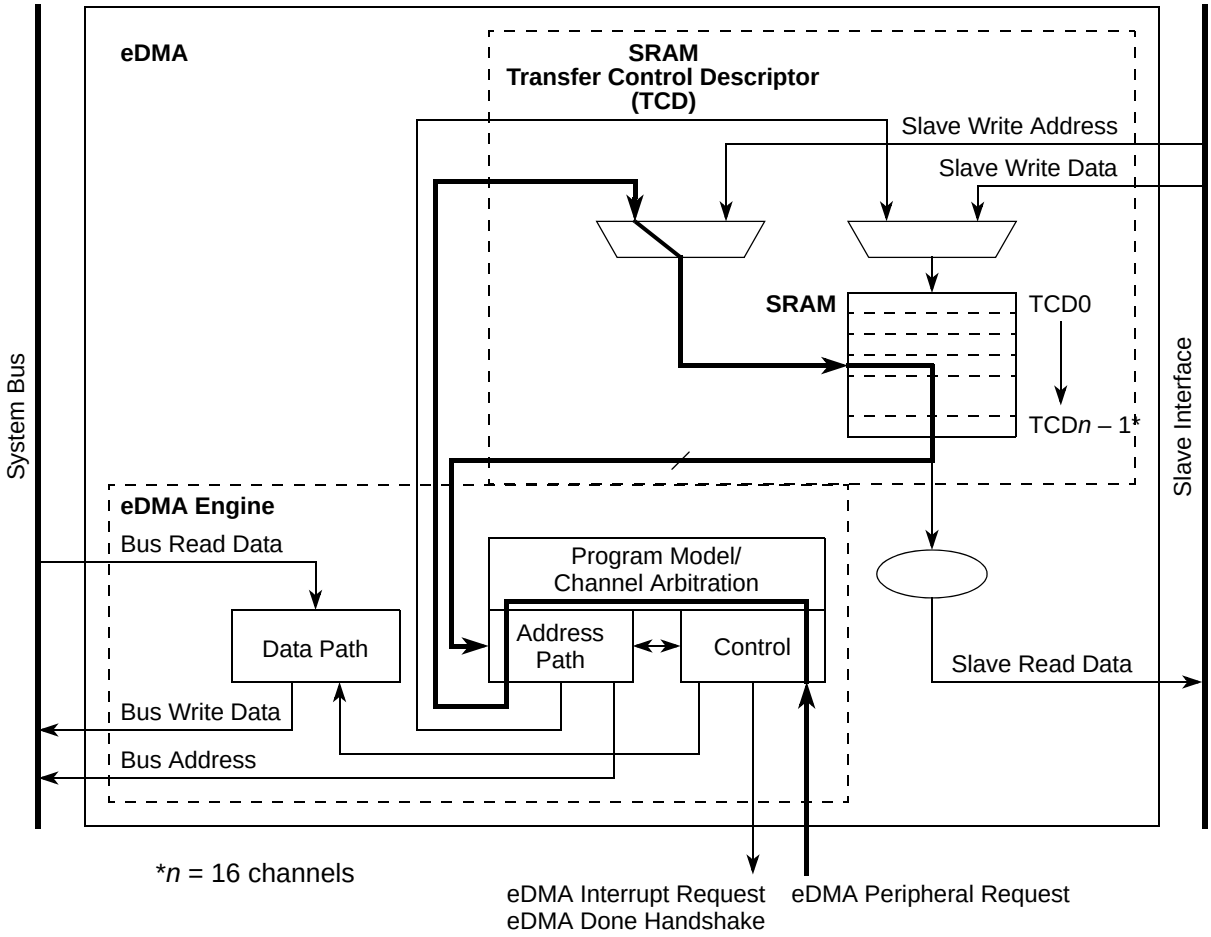


Figure 204. eDMA operation, part 1

In the second part of the basic data flow as shown in *Figure 205*, the modules associated with the data transfer (address path, data path and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA Done Handshake signal is asserted at the end of the minor byte count transfer.

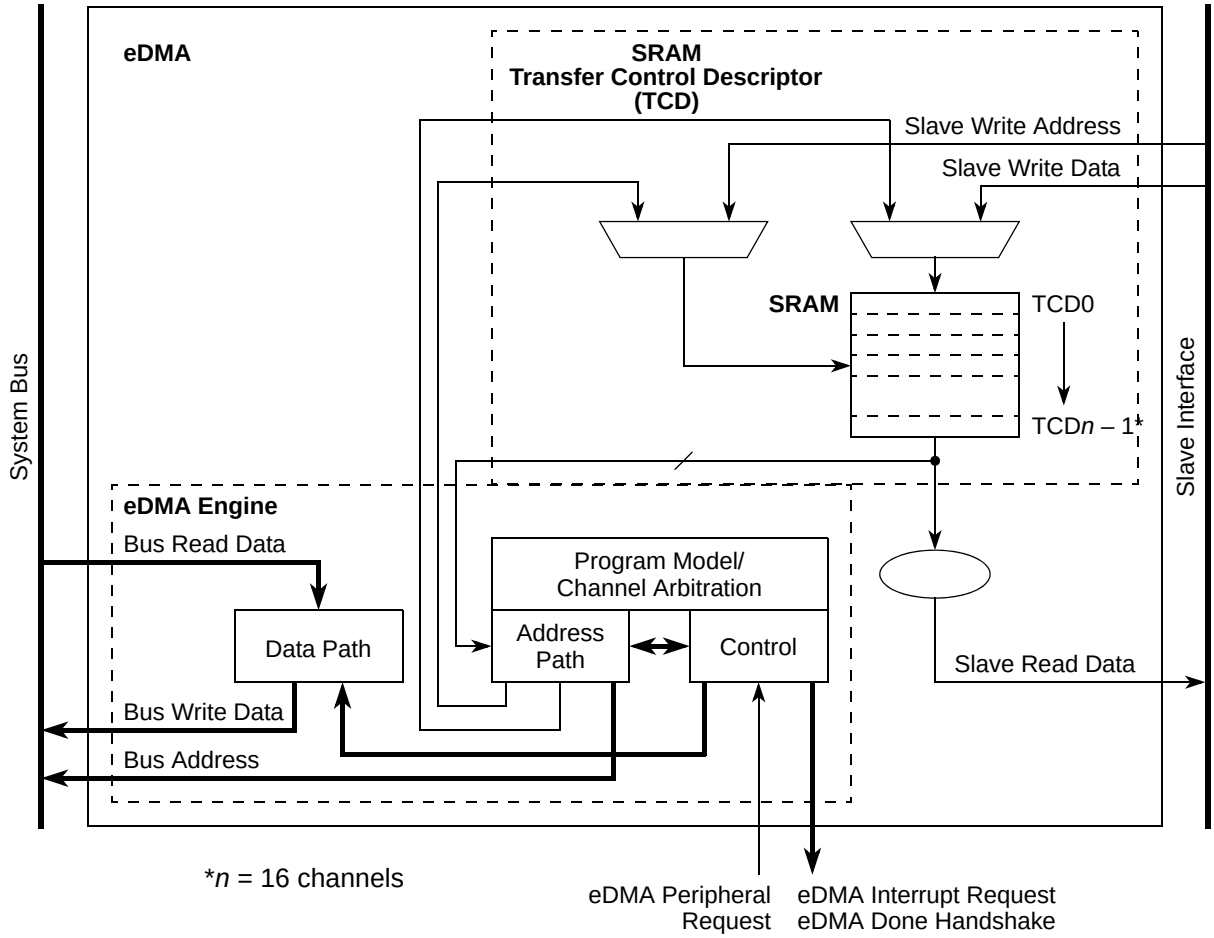


Figure 205. eDMA operation, part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel's TCD: for example., SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in *Figure 206*.

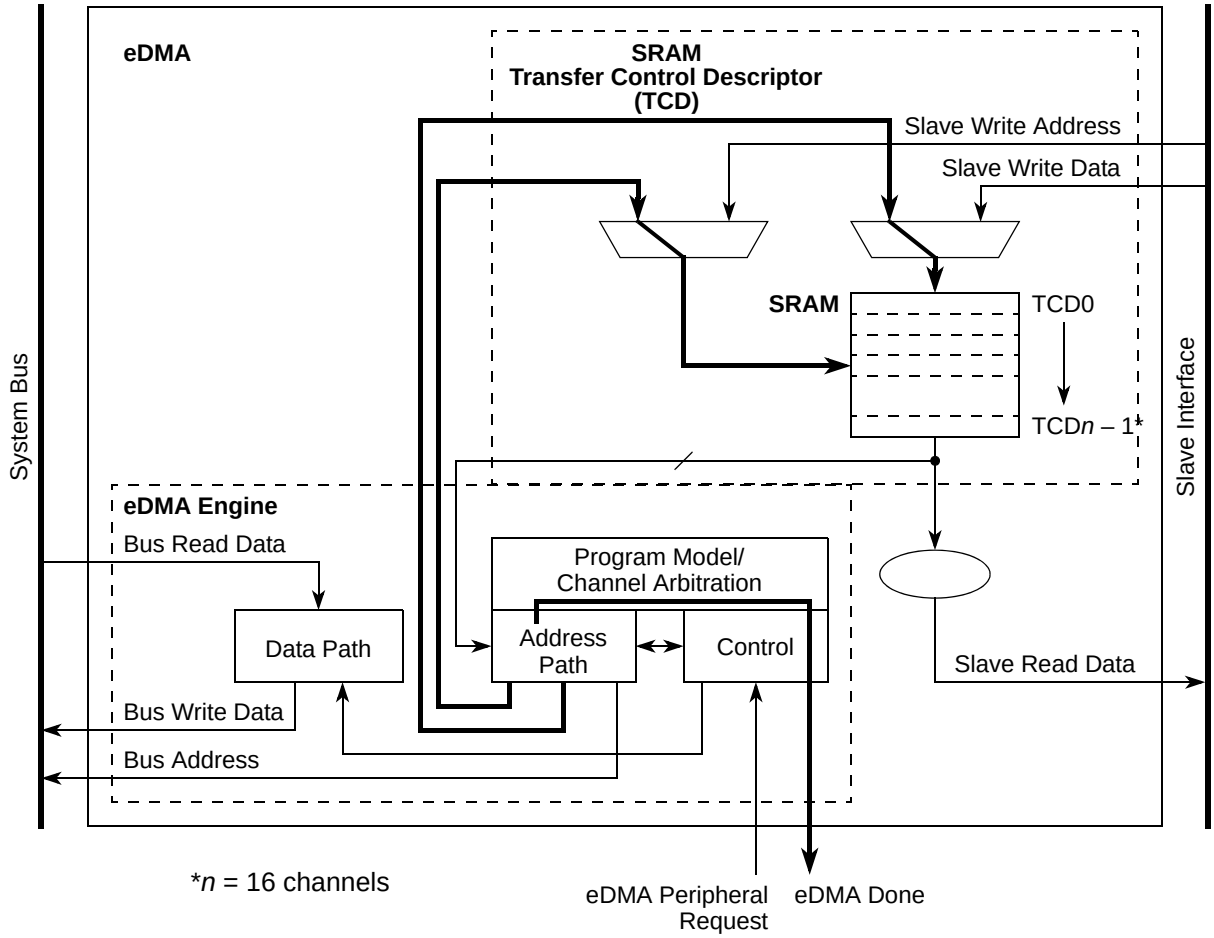


Figure 206. eDMA operation, part 3

18.6.3 eDMA performance

This section addresses the performance of the eDMA module, focusing on two separate metrics. In the traditional data movement context, performance is best expressed as the peak data transfer rates achieved using the eDMA. In most implementations, this transfer rate is limited by the speed of the source and destination address spaces. In a second context where device-paced movement of single data values to/from peripherals is dominant, a measure of the requests that can be serviced in a fixed time is a more useful metric. In this environment, the speed of the source and destination address spaces remains important, but the microarchitecture of the eDMA also factors significantly into the resulting metric.

The peak transfer rates for several different source and destination transfers are shown in *Table 205*. The following assumptions apply to *Table 205* and *Table 206*:

- Internal SRAM can be accessed with zero wait-states when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- All slave accesses are 32-bits in size.

Table 205. eDMA peak transfer rates (MB/Sec)

System Speed, Transfer Size	Internal SRAM-to-Internal SRAM	32-bit Slave-to-Internal SRAM	Internal SRAM-to-32-bit Slave (buffering disabled)	Internal SRAM-to-32-bit Slave (buffering enabled)
66.7 MHz, 32-bit	66.7	66.7	53.3	88.7
66.7 MHz, 64-bit	133.3	66.7	53.3	88.7
66.7 MHz, 256-bit(1)	213.4	N/A(2)	N/A(2)	N/A(2)
83.3 MHz, 32-bit	83.3	83.3	66.7	110.8
83.3 MHz, 64-bit	166.7	83.3	66.7	110.8
83.3 MHz, 256-bit(1)	266.6	N/A(2)	N/A(2)	N/A(2)
100.0 MHz, 32-bit	100.0	100.0	80.0	133.0
100.0 MHz, 64-bit	200.0	100.0	80.0	133.0
100.0 MHz, 256-bit(1)	320.0	N/A(2)	N/A(2)	N/A(2)
132.0 MHz, 32-bit	132.0	132.0	105.6	175.6
132.0 MHz, 64-bit	264.0	132.0	105.6	175.6
132.0 MHz, 256-bit(1)	422.4	N/A(2)	N/A(2)	N/A(2)

1. A 256-bit transfer occurs as a burst of four 64-bit beats.
2. Not applicable: burst access to a slave port is not supported.

Table 205 presents a peak transfer rate comparison, measured in MBs per second where the internal-SRAM-to-internal-SRAM transfers occur at the core's datapath width; that is, either 32- or 64-bits per access. For all transfers involving the slave bus, 32-bit transfer sizes are used. In all cases, the transfer rate includes the time to read the source plus the time to write the destination.

The second performance metric is a measure of the number of DMA requests that can be serviced in a given amount of time. For this metric, it is assumed the peripheral request causes the channel to move a single slave-mapped operand to/from internal SRAM. The same timing assumptions used in the previous example apply to this calculation. In particular, this metric also reflects the time required to activate the channel. The eDMA design supports the following hardware service request sequence:

- Cycle 1: eDMA peripheral request is asserted.
- Cycle 2: The eDMA peripheral request is registered locally in the eDMA module and qualified. (TCD.START bit initiated requests start at this point with the registering of the slave write to TCD bit 255).
- Cycle 3: Channel arbitration begins.
- Cycle 4: Channel arbitration completes. The transfer control descriptor local memory read is initiated.
- Cycle 5–6: The first two parts of the activated channel's TCD is read from the local memory. The memory width to the eDMA engine is 64 bits, so the entire descriptor can be accessed in four cycles.

- Cycle 7: The first system bus read cycle is initiated, as the third part of the channel's TCD is read from the local memory. Depending on the state of the crossbar switch, arbitration at the system bus can insert an additional cycle of delay here.
- Cycle 8 – *n*: The last part of the TCD is read in. This cycle represents the 1st data phase for the read, and the address phase for the destination write.

The exact timing from this point is a function of the response times for the channel's read and write accesses. In this case of an slave read and internal SRAM write, the combined data phase time is 4 cycles. For an SRAM read and slave write, it is 5 cycles.

- Cycle *n* + 1: This cycle represents the data phase of the last destination write.
- Cycle *n* + 2: The eDMA engine completes the execution of the inner minor loop and prepares to write back the required TCD*n* fields into the local memory. The control/status fields at word offset 0x1C in TCD*n* are read. If the major loop is complete, the MAJOR.E_LINK and E_SG bits are checked and processed if enabled.
- Cycle *n* + 3: The appropriate fields in the first part of the TCD*n* are written back into the local memory.
- Cycle *n* + 4: The fields in the second part of the TCD*n* are written back into the local memory. This cycle coincides with the next channel arbitration cycle start.
- Cycle *n* + 5: The next channel to be activated performs the read of the first part of its TCD from the local memory. This is equivalent to Cycle 4 for the first channel's service request.

Assuming zero wait states on the system bus, DMA requests can be processed every 9 cycles. Assuming an average of the access times associated with slave-to-SRAM (4 cycles) and SRAM-to-slave (5 cycles), DMA requests can be processed every 11.5 cycles (4 + (4 + 5)/2 + 3). This is the time from Cycle 4 to Cycle "*n* 5." The resulting peak request rate, as a function of the system frequency, is shown in *Table 206*. This metric represents millions of requests per second.

Table 206. eDMA peak request Rate (MReq/sec)

System Frequency (MHz)	Request Rate (Zero Wait States)	Request Rate (with Wait States)
66.6	7.4	5.8
83.3	9.2	7.2
100.0	11.1	8.7
133.3	14.8	11.6
150.0	16.6	13.0

A general formula to compute the peak request rate (with overlapping requests) is:

Equation 13 PEAKreq = freq / [entry + (1 + read_ws) + (1 + write_ws) + exit]

where:

PEAKreq — peak request rate
freq — system frequency
entry — channel startup (four cycles)
read_ws — wait states seen during the system bus read data phase
write_ws — wait states seen during the system bus write data phase

exit — channel shutdown (three cycles)

For example: consider a system with the following characteristics:

- Internal SRAM can be accessed with one wait-state when viewed from the system bus data phase.
- All slave reads require two wait-states, and slave writes three wait-states, again viewed from the system bus data phase.
- System operates at 150 MHz.

For an SRAM to slave transfer,

Equation 14 PEAKreq = 150 MHz / [4 + (1 + 1) + (1 + 3) + 3] cycles = 11.5 Mreq/sec

For an slave to SRAM transfer,

Equation 15 PEAKreq = 150 MHz / [4 + (1 + 2) + (1 + 1) + 3] cycles = 12.5 Mreq/sec

Assuming an even distribution of the two transfer types, the average peak request rate is:

Equation 16 PEAKreq = (11.5 Mreq/sec + 12.5 Mreq/sec) / 2 = 12.0 Mreq/sec

The minimum number of cycles to perform a single read/write, zero wait states on the system bus, from a cold start (no channel is executing, eDMA is idle) are the following:

- 11 cycles for a software (TCD.START bit) request
- 12 cycles for a hardware (eDMA peripheral request signal) request

Two cycles account for the arbitration pipeline and one extra cycle on the hardware request resulting from the internal registering of the eDMA peripheral request signals. For the peak request rate calculations above, the arbitration and request registering is absorbed in or overlap the previous executing channel.

Note: When channel linking or scatter/gather is enabled, a two-cycle delay is imposed on the next channel selection and startup. This allows the link channel or the scatter/gather channel to be eligible and considered in the arbitration pool for next channel selection.

18.7 Initialization / application information

18.7.1 eDMA initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_EEIRL and/or EDMA_EEIRH registers (optional).
4. Write the 32-byte TCD for each channel that can request service.
5. Enable any hardware service requests via the EDMA_ERQRH and/or EDMA_ERQRL registers.
6. Request channel service by either software (setting the TCD.START bit) or by hardware (slave device asserting its eDMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The eDMA engine reads

the entire TCD, including the primary transfer control parameter shown in *Table 207*, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, TCD.SADDR) to the destination (as defined by the destination address, TCD.DADDR) continue until the specified number of bytes (TCD.NBYTES) have been transferred. When the transfer is complete, the eDMA engine's local TCD.SADDR, TCD.DADDR, and TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed: for example, interrupts, major loop channel linking, and scatter/gather operations, if enabled.

Table 207. TCD primary control and status fields

TCD Field Name	Description
START	Control bit to explicitly start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (Cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for “throttling” bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 207 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

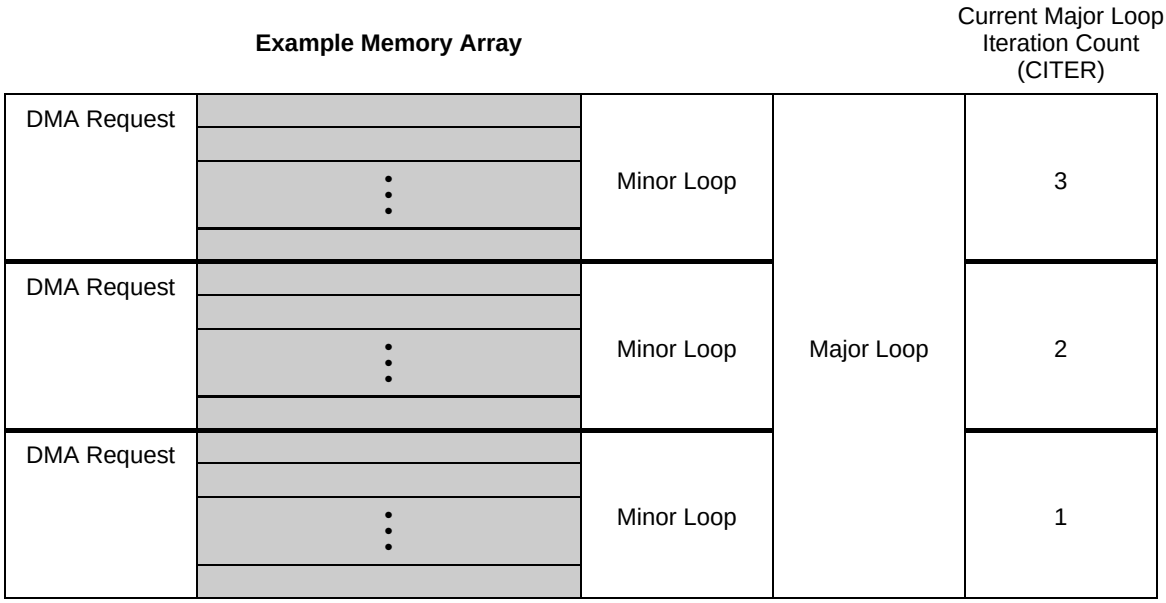


Figure 207. Example of multiple loop iterations

Figure 208 lists the memory array terms and how the TCD settings interrelate.

xADDR: (Starting Address)	xSIZE: (Size of one data transfer)	Minor Loop (NBYTES in Minor Loop, often the same value as xSIZE)	Offset (xOFF): Number of bytes added to current address after each transfer (Often the same value as xSIZE)
			Each DMA Source (S) and Destination (D) has its own: • Address (xADDR) • Size (xSIZE) • Offset (xOFF) • Modulo (xMOD) • Last Address Adjustment (xLAST) where x = S or D Peripheral queues typically have size and offset equal to NBYTES
		Minor Loop	
xLAST: Number of bytes added to current address after Major Loop (Typically used to loop back)		Last Minor Loop	

Figure 208. Memory array terms

18.7.2 DMA programming errors

The eDMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per channel basis with the exception of channel priority error (EDMA_ESR[CPE]).

For all error types other than channel priority errors, the channel number causing the error is recorded in the EDMA_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

If priority levels are not unique, the highest priority that has an active request is selected, but the lowest numbered with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts and error reporting are associated with the selected channel.

18.7.3 DMA request assignments

The assignments between the DMA requests from the modules to the channels of the eDMA are shown in *Table 208*. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 208. DMA request summary for eDMA

DMA Request	Ch.	Source	Description
DMA_MUX_CHCONFIG0_SOURCE	0	DMA_MUX.CHCONFIG0[SOURCE]	DMA MUX channel 0 source
DMA_MUX_CHCONFIG1_SOURCE	1	DMA_MUX.CHCONFIG1[SOURCE]	DMA MUX channel 1 source
DMA_MUX_CHCONFIG2_SOURCE	2	DMA_MUX.CHCONFIG2[SOURCE]	DMA MUX channel 2 source
DMA_MUX_CHCONFIG3_SOURCE	3	DMA_MUX.CHCONFIG3[SOURCE]	DMA MUX channel 3 source
DMA_MUX_CHCONFIG4_SOURCE	4	DMA_MUX.CHCONFIG4[SOURCE]	DMA MUX channel 4 source
DMA_MUX_CHCONFIG5_SOURCE	5	DMA_MUX.CHCONFIG5[SOURCE]	DMA MUX channel 5 source
DMA_MUX_CHCONFIG6_SOURCE	6	DMA_MUX.CHCONFIG6[SOURCE]	DMA MUX channel 6 source
DMA_MUX_CHCONFIG7_SOURCE	7	DMA_MUX.CHCONFIG7[SOURCE]	DMA MUX channel 7 source
DMA_MUX_CHCONFIG8_SOURCE	8	DMA_MUX.CHCONFIG8[SOURCE]	DMA MUX channel 8 source
DMA_MUX_CHCONFIG9_SOURCE	9	DMA_MUX.CHCONFIG9[SOURCE]	DMA MUX channel 9 source
DMA_MUX_CHCONFIG10_SOURCE	10	DMA_MUX.CHCONFIG10[SOURCE]	DMA MUX channel 10 source
DMA_MUX_CHCONFIG11_SOURCE	11	DMA_MUX.CHCONFIG11[SOURCE]	DMA MUX channel 11 source

18.7.4 DMA arbitration mode considerations

18.7.4.1 Fixed-channel arbitration

In this mode, the channel service request from the highest priority channel is selected to execute. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

18.7.4.2 Fixed-group arbitration, round-robin channel arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

18.7.5 DMA transfer

18.7.5.1 Single request

To perform a simple transfer of '*n*' bytes of data with one activation, set the major loop to 1 (TCD.CITER = TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer completes, the TCD.DONE bit is set and an interrupt is generated if correctly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte-wide memory port located at 0x1000. The destination memory has a word-wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

TCD.CITER = TCD.BITER = 1
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –16
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA= –16
TCD.INT_MAJ = 1
TCD.START = 1 (Initialize all other fields before writing to this bit)
All other TCD fields = 0

This generates the following sequence of events:

1. Slave write to the TCD.START bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte (0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word (0x2000) → first iteration of the minor loop
   c) read_byte (0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word (0x2004) → second iteration of the minor loop
   e) read_byte (0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word (0x2008) → third iteration of the minor loop
   g) read_byte (0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
   h) write_word (0x200c) → last iteration of the minor loop → major loop complete
6. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 1 (TCD.BITER).
7. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

18.7.5.2 Multiple requests

The next example is the same as previous with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_ERQR, channel service requests are initiated by the slave device (set ERQR after TCD; TCD.START = 0).

TCD.CITER = TCD.BITER = 2
TCD.NBYTES = 16
TCD.SADDR = 0x1000
TCD.SOFF = 1
TCD.SSIZE = 0
TCD.SLAST = –32
TCD.DADDR = 0x2000
TCD.DOFF = 4
TCD.DSIZE = 2
TCD.DLAST_SGA = –32
TCD.INT_MAJ = 1
TCD.START = 0 (Initialize all other fields before writing this bit.)
All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers execute as follows:
   a) read_byte (0x1000), read_byte (0x1001), read_byte (0x1002), read_byte (0x1003)
   b) write_word (0x2000) → first iteration of the minor loop
   c) read_byte (0x1004), read_byte (0x1005), read_byte (0x1006), read_byte (0x1007)
   d) write_word (0x2004) → second iteration of the minor loop
   e) read_byte (0x1008), read_byte (0x1009), read_byte (0x100a), read_byte (0x100b)
   f) write_word (0x2008) → third iteration of the minor loop
   g) read_byte (0x100c), read_byte (0x100d), read_byte (0x100e), read_byte (0x100f)
   h) write_word (0x200c) → last iteration of the minor loop
6. eDMA engine writes: TCD.SADDR = 0x1010, TCD.DADDR = 0x2010, TCD.CITER = 1.
7. eDMA engine writes: TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: TCD.DONE = 0, TCD.START = 0, TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers execute as follows:
    a) read_byte (0x1010), read_byte (0x1011), read_byte (0x1012), read_byte (0x1013)
    b) write_word (0x2010) → first iteration of the minor loop
    c) read_byte (0x1014), read_byte (0x1015), read_byte (0x1016), read_byte (0x1017)
    d) write_word (0x2014) → second iteration of the minor loop
    e) read_byte (0x1018), read_byte (0x1019), read_byte (0x101a), read_byte (0x101b)
    f) write_word (0x2018) → third iteration of the minor loop
    g) read_byte (0x101c), read_byte (0x101d), read_byte (0x101e), read_byte (0x101f)
    h) write_word (0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: TCD.SADDR = 0x1000, TCD.DADDR = 0x2000, TCD.CITER = 2 (TCD.BITER).
15. eDMA engine writes: TCD.ACTIVE = 0, TCD.DONE = 1, EDMA_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

18.7.5.3 Modulo feature

The modulo feature of the eDMA provides the ability to easily implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for the source and destination in the TCD, and specifies which lower address bits are incremented from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. Clearing this field to 0 disables the modulo feature.

Table 209 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 209. Modulo feature example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

18.7.6 TCD status

18.7.6.1 Minor loop complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD.CITER field and test for a change. Another method can be extracted from the following sequence. The second method is to test the TCD.START bit AND the TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the TCD.START was written to a one. Polling the TCD.ACTIVE bit can be inconclusive because the active status can be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. TCD.START = 1, TCD.ACTIVE = 0, TCD.DONE = 0 (issued service request via software)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

The best method to test for minor loop completion when using hardware initiated service requests is to read the TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (issued service request via hardware)
2. TCD.START = 0, TCD.ACTIVE = 1, TCD.DONE = 0 (executing)
3. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 0 (completed minor loop and is idle) or
4. TCD.START = 0, TCD.ACTIVE = 0, TCD.DONE = 1 (completed major loop and is idle)

For both activation types, the major loop complete status is explicitly indicated via the TCD.DONE bit.

The TCD.START bit is cleared automatically when the channel begins execution regardless of how the channel was activated.

18.7.6.2 Active channel TCD reads

the eDMA reads the true TCD.SADDR, TCD.DADDR, and TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

18.7.6.3 Preemption status

Preemption is only available when fixed arbitration is selected for both group and channel arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel

and/or group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

18.7.7 Channel linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD.START bit of another channel (or itself) thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD.CITER.E_LINK field determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last.

When the major loop is exhausted, only the major loop channel link fields are used to determine whether to make a channel link. For example, with the initial fields of:

TCD.CITER.E_LINK = 1
TCD.CITER.LINKCH = 0xC
TCD.CITER value = 0x4
TCD.MAJOR.E_LINK = 1
TCD.MAJOR.LINKCH = 0x7

channel linking executes as:

1. Minor loop done → set channel 12 TCD.START bit
2. Minor loop done → set channel 12 TCD.START bit
3. Minor loop done → set channel 12 TCD.START bit
4. Minor loop done, major loop done → set channel 7 TCD.START bit

When minor loop linking is enabled (TCD.CITER.E_LINK = 1), the TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (TCD.CITER.E_LINK = 0), the TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

Note: After configuration, the TCD.CITER.E_LINK bit and the TCD.BITER.E_LINK bit must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

Table 210 summarizes how a DMA channel can “link” to another DMA channel, i.e, use another channel’s TCD, at the end of a loop.

Table 210. Channel linking parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of Minor Loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration)
	citer.linkch	Link channel number when linking at end of minor loop (current iteration)
Link at end of Major Loop	major.e_link	Enable channel-to-channel linking on major loop completion
	major.linkch	Link channel number when linking at end of major loop

18.7.8 Dynamic programming

This section provides recommended methods to change the programming model during channel execution.

18.7.8.1 Dynamic channel linking and dynamic scatter/gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD.MAJOR.E_LINK or TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the *end* of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The TCD.MAJOR.E_LINK is set in the programmer's model, but it is unclear whether the link completed before the channel retired.

Use the following coherency model when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD.MAJOR.E_LINK bit
2. Read the TCD.MAJOR.E_LINK bit
3. Test the TCD.MAJOR.E_LINK request status:
   a) If the bit is set, the dynamic link attempt was successful.D
   b) If the bit is cleared, the channel had already retired before the dynamic link completed.

This same coherency model is true for dynamic scatter/gather operations. For both dynamic requests, the TCD local memory controller forces the TCD.MAJOR.E_LINK and TCD.E_SG bits to zero on any writes to a channel's TCD after that channel's TCD.DONE bit is set indicating the major loop is complete.

Note: The user must clear the TCD.DONE bit before writing the TCD.MAJOR.E_LINK or TCD.E_SG bits. The TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

19 DMA Channel Mux (DMA_MUX)

19.1 Introduction

19.1.1 Overview

The DMA Mux allows to route a configurable amount of DMA sources (slots) to a configurable amount of DMA channels. This is illustrated in *Figure 209*.

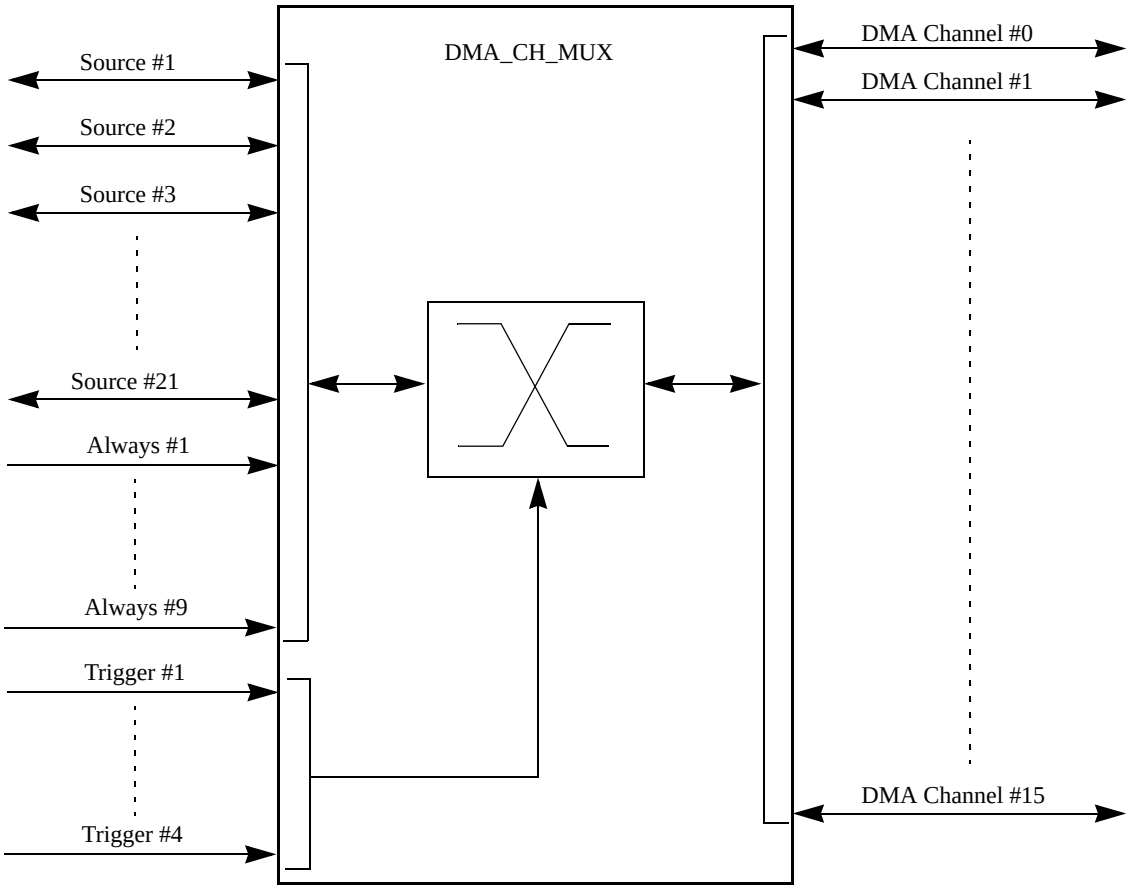


Figure 209. DMA Mux block diagram

19.1.2 Features

The DMA Mux has these major features:

- 16 independently selectable DMA channel routers
  - 4 channels with normal or periodic triggering capability
  - 12 channels with normal capability
- Each channel router can be assigned to 1 of 21 possible peripheral DMA sources

19.1.3 Modes of operation

The following operation modes are available:

- Disabled Mode

  In this mode, the DMA channel is disabled. Since disabling and enabling of DMA channels is done primarily via the DMA configuration registers, this mode is used mainly as the reset state for a DMA channel in the DMA Channel Mux. It may also be used to temporarily suspend a DMA channel while reconfiguration of the system takes place (for example, changing the period of a DMA trigger).

- Normal Mode

  In this mode, a DMA source (such as DSPI_0_TX or DSPI_0_RX example) is routed directly to the specified DMA channel. The operation of the DMA Mux in this mode is completely transparent to the system.

- Periodic Trigger Mode

  In this mode, a DMA source may only request a DMA transfer (such as when a transmit buffer becomes empty or a receive buffer becomes full) periodically. Configuration of the period is done in the registers of the Periodic Interrupt Timer (PIT).

DMA channels 0–3 may be used in all the modes listed above but channels 4–15 may be configured only to disabled or normal mode.

19.2 External signal description

19.2.1 Overview

The DMA Mux has no external pins.

19.3 Memory map and register definition

This section provides a detailed description of all memory-mapped registers in the DMA Mux.

19.3.1 Memory map

Table 211 shows the memory map for the DMA Mux. Note that all addresses are offsets; the absolute address may be computed by adding the specified offset to the base address of the DMA Mux.

Table 211. DMA_MUX memory map

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0000	Channel #0 Configuration (CHCONFIG0)	*on page 451*
0x0001	Channel #1 Configuration (CHCONFIG1)	*on page 451*
0x0002	Channel #2 Configuration (CHCONFIG2)	*on page 451*
0x0003	Channel #3 Configuration (CHCONFIG3)	*on page 451*

Table 211. DMA_MUX memory map(Continued)

Offset from DMA_MUX_BASE (0xFFFD_C000)	Register	Location
0x0004	Channel #4 Configuration (CHCONFIG4)	*on page 451*
0x0005	Channel #5 Configuration (CHCONFIG5)	*on page 451*
0x0006	Channel #6 Configuration (CHCONFIG6)	*on page 451*
0x0007	Channel #7 Configuration (CHCONFIG7)	*on page 451*
0x0008	Channel #8 Configuration (CHCONFIG8)	*on page 451*
0x0009	Channel #9 Configuration (CHCONFIG9)	*on page 451*
0x000A	Channel #10 Configuration (CHCONFIG10)	*on page 451*
0x000B	Channel #11 Configuration (CHCONFIG11)	*on page 451*
0x000C	Channel #12 Configuration (CHCONFIG12)	*on page 451*
0x000D	Channel #13 Configuration (CHCONFIG13)	*on page 451*
0x000E	Channel #14 Configuration (CHCONFIG14)	*on page 451*
0x000F	Channel #15 Configuration (CHCONFIG15)	*on page 451*
0x0010–0x3FFF	Reserved	

All registers are accessible via 8-bit, 16-bit or 32-bit accesses. However, 16-bit accesses must be aligned to 16-bit boundaries, and 32-bit accesses must be aligned to 32-bit boundaries. As an example, CHCONFIG0 through CHCONFIG3 are accessible by a 32-bit READ/WRITE to address ‘Base + 0x0000’, but performing a 32-bit access to address ‘Base + 0x0001’ is illegal.

19.3.2 Register descriptions

19.3.2.1 Channel Configuration Registers

Each of the DMA channels can be independently enabled/disabled and associated with one of the #SRC + #ALE total DMA sources in the system.

Address: Base + *#n* Access: User read/write

	7	6	5	4	3	2	1	0
R W	ENBL	TRIG	SOURCE					
Reset	0	0	0	0	0	0	0	0

Figure 210. Channel Configuration Registers (CHCONFIG#*n*)

Table 212. CHCONFIG#*x* field descriptions

Field	Description
7 ENBL	DMA Channel Enable ENBL enables the DMA Channel. 0 DMA channel is disabled. This mode is primarily used during configuration of the DMA Mux. The DMA has separate channel enables/disables that should be used to disable or reconfigure a DMA channel. 1 DMA channel is enabled.
6 TRIG	DMA Channel Trigger Enable (for triggered channels only) TRIG enables the periodic trigger capability for the DMA Channel. 0 Triggering is disabled. If triggering is disabled, and the ENBL bit is set, the DMA Channel routes the specified source to the DMA channel. 1 Triggering is enabled.
5–0 SOURCE	DMA Channel Source (slot) SOURCE specifies which DMA source, if any, is routed to a particular DMA channel. See *Table 214*.

Table 213. Channel and trigger enabling

ENBL	TRIG	Function	Mode
0	X	DMA Channel is disabled	Disabled Mode
1	0	DMA Channel is enabled with no triggering (transparent)	Normal Mode
1	1	DMA Channel is enabled with triggering	Periodic Trigger Mode

Note: Setting multiple CHCONFIG registers with the same Source value will result in unpredictable behavior.

Note: Before changing the trigger or source settings a DMA channel must be disabled via the CHCONFIG[#n].ENBL bit.

19.4 DMA request mapping

Table 214. DMA channel mapping

DMA_CH_MUX channel	Module	DMA requesting module	DMA Mux input #
1	DSPI_0	DSPI_0 TX	DMA MUX Source #1
2	DSPI_0	DSPI_0 RX	DMA MUX Source #2
3	DSPI_1	DSPI_1 TX	DMA MUX Source #3
4	DSPI_1	DSPI_1 RX	DMA MUX Source #4
5	DSPI_2	DSPI_2 TX	DMA MUX Source #5
6	DSPI_2	DSPI_2 RX	DMA MUX Source #6
9	CTU_0	CTU	DMA MUX Source #9
10	CTU_0	CTU FIFO 0	DMA MUX Source #10
11	CTU_0	CTU FIFO 1	DMA MUX Source #11

Table 214. DMA channel mapping(Continued)

DMA_CH_MUX channel	Module	DMA requesting module	DMA Mux input #
12	CTU_0	CTU FIFO 2	DMA MUX Source #12
13	CTU_0	CTU FIFO 3	DMA MUX Source #13
16	etimer_0	eTimer_0 CH0	DMA MUX Source #16
17	etimer_0	eTimer_0 CH1	DMA MUX Source #17
20	adc_0	ADC_0	DMA MUX Source #20
22	—	ALWAYS requestors	DMA MUX Source #22
23	—	ALWAYS requestors	DMA MUX Source #23
24	—	ALWAYS requestors	DMA MUX Source #24
25	—	ALWAYS requestors	DMA MUX Source #25
26	—	ALWAYS requestors	DMA MUX Source #26
27	—	ALWAYS requestors	DMA MUX Source #27
28	—	ALWAYS requestors	DMA MUX Source #28
29	—	ALWAYS requestors	DMA MUX Source #29
30	—	ALWAYS requestors	DMA MUX Source #30

19.5 Functional description

This section provides a complete functional description of the DMA Mux. The primary purpose of the DMA Mux is to provide flexibility in the system's use of the available DMA channels. As such, configuration of the DMA Mux is intended to be a static procedure done during execution of the system boot code. However, if the procedure outlined in *Section 19.6.2: Enabling and configuring sources* is followed, the configuration of the DMA Mux may be changed during the normal operation of the system.

Functionally, the DMA Mux channels may be divided into two classes: Channels that implement the normal routing functionality plus periodic triggering capability, and channels that implement only the normal routing functionality.

19.5.1 DMA channels with periodic triggering capability

Besides the normal routing functionality, the first four channels of the DMA Mux provide a special periodic triggering capability that can be used to provide an automatic mechanism to transmit bytes, frames or packets at fixed intervals without the need for processor intervention. The trigger is generated by the Periodic Interrupt Timer (PIT); as such, the configuration of the periodic triggering interval is done via configuration registers in the PIT. Please refer to *Chapter 31: Periodic Interrupt Timer (PIT)* for more information on this topic.

Note: Because of the dynamic nature of the system (i.e., DMA channel priorities, bus arbitration, interrupt service routine lengths, etc.), the number of clock cycles between a trigger and the actual DMA transfer cannot be guaranteed.

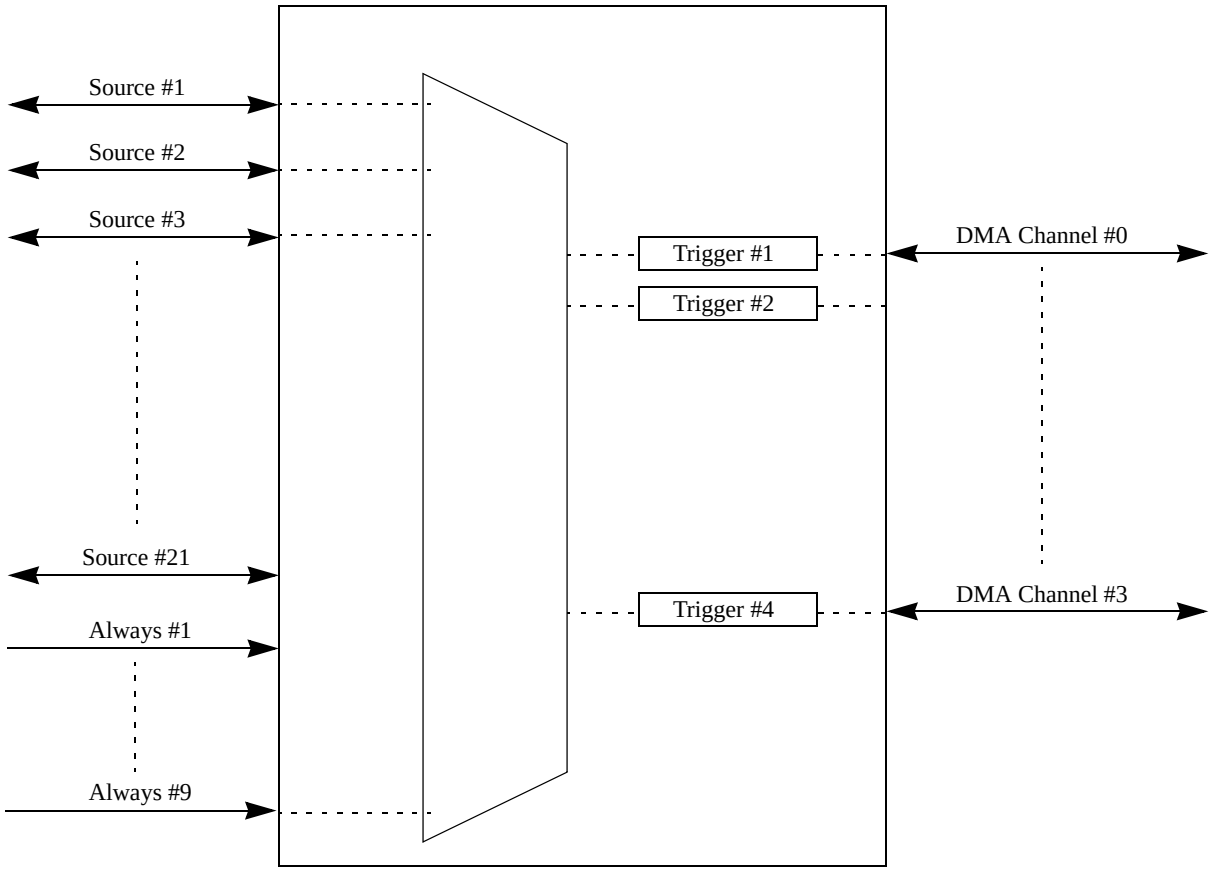


Figure 211. DMA mux triggered channels diagram

The DMA channel triggering capability allows the system to “schedule” regular DMA transfers, usually on the transmit side of certain peripherals, without the intervention of the processor. This trigger works by gating the request from the peripheral to the DMA until a trigger event has been seen. This is illustrated in *Figure 212*.

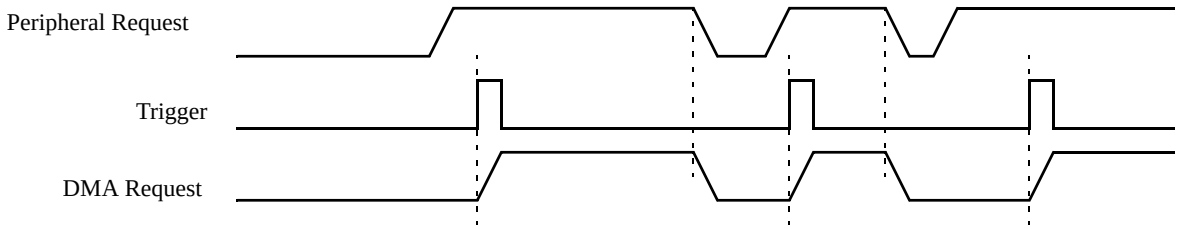


Figure 212. DMA mux channel triggering: normal operation

Once the DMA request has been serviced, the peripheral will negate its request, effectively resetting the gating mechanism until the peripheral re-asserts its request AND the next trigger event is seen. This means that if a trigger is seen, but the peripheral is not requesting a transfer, that triggered will be ignored. This situation is illustrated in *Figure 213*.

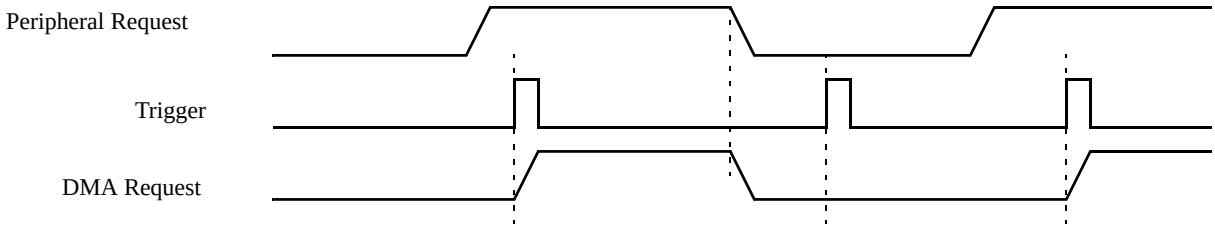


Figure 213. DMA mux channel triggering: ignored trigger

This triggering capability may be used with any peripheral that supports DMA transfers, and is most useful for two types of situations:

- Periodically polling external devices on a particular bus. As an example, the transmit side of an SPI is assigned to a DMA channel with a trigger, as described above. Once setup, the SPI will request DMA transfers (presumably from memory) as long as its transmit buffer is empty. By using a trigger on this channel, the SPI transfers can be automatically performed every 5 µs (as an example). On the receive side of the SPI, the SPI and DMA can be configured to transfer receive data into memory, effectively implementing a method to periodically read data from external devices and transfer the results into memory without processor intervention.
- Using the GPIO Ports to drive or sample waveforms. By configuring the DMA to transfer data to one or more GPIO ports, it is possible to create complex waveforms using tabular data stored in on-chip memory. Conversely, using the DMA to periodically transfer data from one or more GPIO ports, it is possible to sample complex waveforms and store the results in tabular form in on-chip memory.

A more detailed description of the capability of each trigger (for example, resolution, range of values, etc.) may be found in *Chapter 31: Periodic Interrupt Timer (PIT)*.

19.5.2 DMA channels with no triggering capability

Channels 4–15 of the DMA Mux provide the normal routing functionality as described in *Section 19.1.3: Modes of operation*.

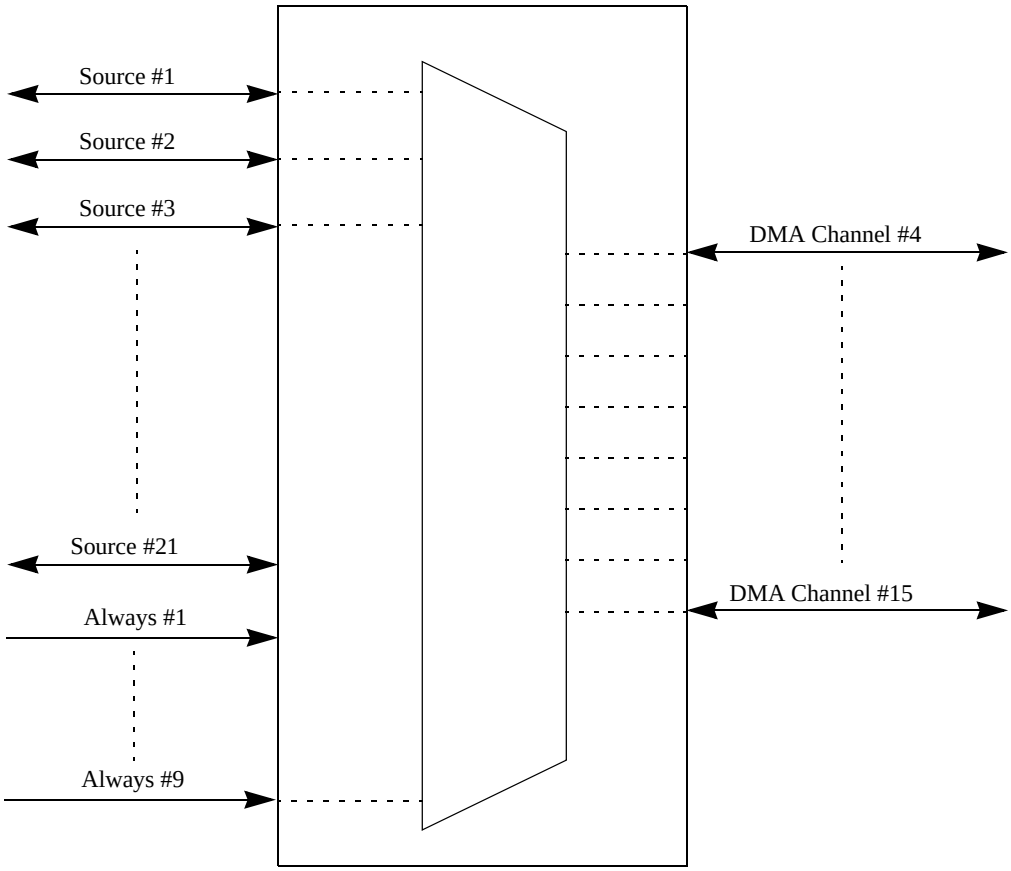


Figure 214. DMA mux channel 4–15 block diagram

19.6 Initialization/application information

19.6.1 Reset

The reset state of each individual bit is shown within the register description section (see *Section 19.3.2: Register descriptions*). In summary, after reset, all channels are disabled and must be explicitly enabled before use.

19.6.2 Enabling and configuring sources

Enabling a source with periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember that only the first four DMA channels have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. In the PIT, configure the corresponding timer.
5. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 1 Configure source #5 Transmit for use with DMA channel 2, with periodic triggering capability

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Configure Timer 3 in the Periodic Interrupt Timer (PIT) for the desired trigger interval.
4. Write 0xC5 to CHCONFIG2 (Base Address + 0x02).

The following code example illustrates steps *1.* and *4.* above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
```

```
#include "registers.h"
        :
        :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0xC5;
```

Enabling a source without periodic triggering:

1. Determine with which DMA channel the source will be associated. Remember that only DMA channels 0–7 have periodic triggering capability.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Ensure that the DMA channel is properly configured in the DMA. The DMA channel may be enabled at this point.
4. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL is set and the TRIG bit is cleared.

Example 2 Configure source #5 Transmit for use with DMA Channel 2, with no periodic triggering capability.

1. Write 0x00 to CHCONFIG2 (Base Address + 0x02).
2. Configure Channel 2 in the DMA, including enabling the channel.
3. Write 0x85 to CHCONFIG2 (Base Address + 0x02).

The following code example illustrates steps *1.* and *3.* above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
```

```
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
        :
        :
*CHCONFIG2 = 0x00;
*CHCONFIG2 = 0x85;
```

Disabling a source:

A particular DMA source may be disabled by not writing the corresponding source value into any of the CHCONFIG registers. Additionally, some module specific configuration may be necessary. Please refer to the appropriate section for more details.

Switching the source of a DMA channel:

1. Disable the DMA channel in the DMA and reconfigure the channel for the new source.
2. Clear the ENBL and TRIG bits of the DMA channel.
3. Select the source to be routed to the DMA channel. Write to the corresponding CHCONFIG register, ensuring that the ENBL and TRIG bits are set.

Example 3 Switch DMA Channel 8 from source #5 transmit to source #7 transmit

1. In the DMA configuration registers, disable DMA channel 8 and reconfigure it to handle the DSPI_0 transmits. This example assumes channel 0..7 have triggering capability (#TRG = 8).
2. Write 0x00 to CHCONFIG8 (Base Address + 0x08).
3. Write 0x87 to CHCONFIG8 (Base Address + 0x08). In this case, setting the TRIG bit would have no effect, because channels 8 and above do not support the periodic triggering functionality.

The following code example illustrates steps *2.* and *3.* above:

```
In File registers.h:
#define DMAMUX_BASE_ADDR     0xFC084000/* Example only ! */
/* Following example assumes char is 8-bits */
volatile unsigned char *CHCONFIG0 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0000);
volatile unsigned char *CHCONFIG1 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0001);
volatile unsigned char *CHCONFIG2 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0002);
volatile unsigned char *CHCONFIG3 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0003);
volatile unsigned char *CHCONFIG4 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0004);
volatile unsigned char *CHCONFIG5 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0005);
volatile unsigned char *CHCONFIG6 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0006);
volatile unsigned char *CHCONFIG7 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0007);
volatile unsigned char *CHCONFIG8 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0008);
volatile unsigned char *CHCONFIG9 = (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x0009);
```

```
volatile unsigned char *CHCONFIG10= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000A);
volatile unsigned char *CHCONFIG11= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000B);
volatile unsigned char *CHCONFIG12= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000C);
volatile unsigned char *CHCONFIG13= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000D);
volatile unsigned char *CHCONFIG14= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000E);
volatile unsigned char *CHCONFIG15= (volatile unsigned char *)
(DMAMUX_BASE_ADDR+0x000F);

In File main.c:
#include "registers.h"
         :
         :
*CHCONFIG8 = 0x00;
*CHCONFIG8 = 0x87;
```

20 FlexRay Communication Controller (FlexRay)

20.1 Introduction

20.1.1 Color coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

20.1.2 Overview

The controller is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The controller has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the controller with its surrounding modules is given in *Figure 215*.

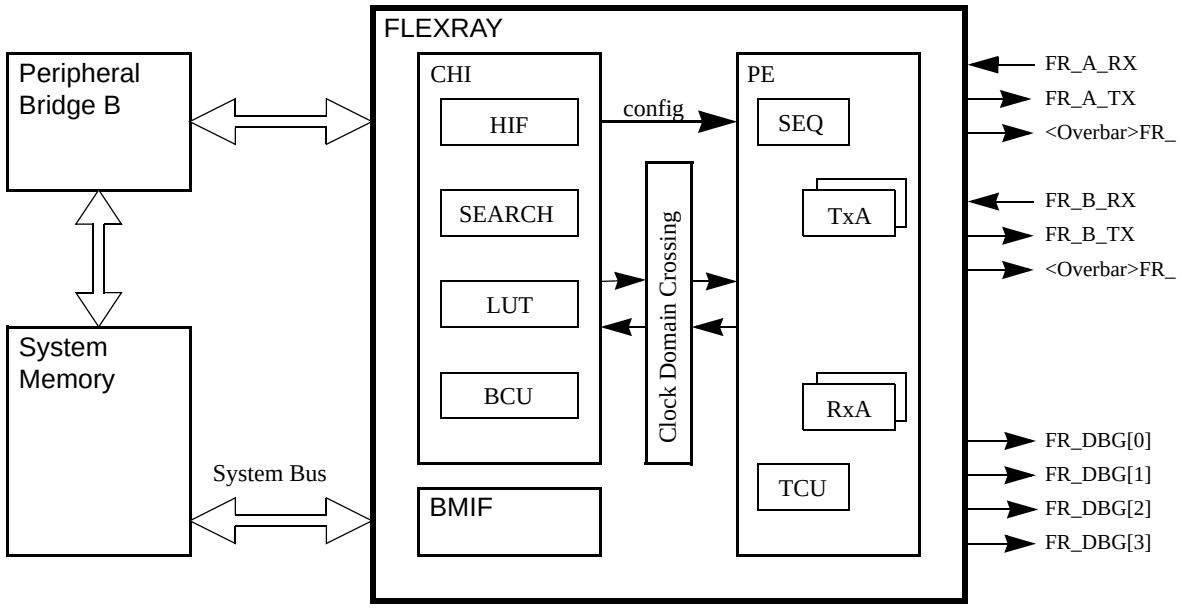


Figure 215. FlexRay block diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control

unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the FlexRay memory.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The controller stores the frame header and payload data of frames received or of frames to be transmitted in the FlexRay memory. The application accesses the FlexRay memory to retrieve and provide the frames to be processed by the controller. In addition to the frame header and payload data, the controller stores the synchronization frame related tables in the FlexRay memory for application processing.

The FlexRay memory is located in the system memory of the MCU. The controller has access to the FlexRay memory via its bus master interface (BMIF). The host provides the start address of the FlexRay memory window within the system memory by programming the *Section 20.5.2.5: System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*. All FlexRay memory related offsets are stored in offset registers. The physical address pointer into the FlexRay memory window of the MCU system memory is calculated using the offset values the FlexRay memory base address.

Note: The controller does not provide a memory protection scheme for the FlexRay memory.

20.1.3 Features

The controller provides the following features:

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A compliant protocol implementation
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A compliant bus driver interface
- Single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 32 configurable message buffers with
  - Individual frame ID filtering
  - Individual channel ID filtering
  - Individual cycle counter filtering
- Message buffer header, status and payload data stored in dedicated FlexRay memory
  - Allows for flexible and efficient message buffer implementation
  - Consistent data access ensured by means of buffer locking scheme
  - Application can lock multiple buffers at the same time
- Size of message buffer payload data section configurable from 0 to 254 bytes
- 2 independent message buffer segments with configurable size of payload data section
  - Each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- Zero padding for transmit message buffers in static segment
  - Applied when the frame payload length exceeds the size of the message buffer data section
- Transmit message buffers configurable with state/event semantics
- Message buffers can be configured as
  - Receive message buffer
  - Single buffered transmit message buffer
  - Double buffered transmit message buffer (combines two single buffered message buffer)
- Individual message buffer reconfiguration supported
  - Means provided to safely disable individual message buffers
  - Disabled message buffers can be reconfigured
- 2 independent receive FIFOs
  - 1 receive FIFO per channel
  - As many as 255 entries for each FIFO
  - Global frame ID filtering, based on both value/mask filters and range filters
  - Global channel ID filtering
  - Global message ID filtering for the dynamic segment
- 4 configurable slot error counters
- 4 dedicated slot status indicators
  - Used to observe slots without using receive message buffers

- Measured value indicators for the clock synchronization
  - Internal synchronization frame ID and synchronization frame measurement tables can be copied into the FlexRay memory
- Fractional macroticks are supported for clock correction
- Maskable interrupt sources provided via individual and combined interrupt lines
- 1 absolute timer
- 1 timer that can be configured to absolute or relative

20.1.4 Modes of operation

This section describes the basic operational power modes of the controller.

20.1.4.1 Disabled mode

The controller enters the Disabled Mode during hard reset. The controller indicates that it is in the Disabled Mode by negating the module enable bit MEN in the *Section 20.5.2.4: Module Configuration Register (MCR)*.

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in *Section 20.5.2: Register descriptions*.

The application configures the controller by accessing the configuration bits and fields in the *Section 20.5.2.4: Module Configuration Register (MCR)*

20.1.4.1.1 Leave disabled mode

The controller leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the *Section 20.5.2.4: Module Configuration Register (MCR)*.

Note: When the controller was enabled, it cannot be disabled the later on.

20.1.4.2 Normal mode

In this mode the controller is fully functional. The controller indicates that it is in Normal Mode by asserting the module enable bit MEN in the *Section 20.5.2.4: Module Configuration Register (MCR)*.

20.1.4.2.1 Enter normal mode

This mode is entered when the application requests the controller to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in *Section 20.7.1.2: Protocol initialization*. to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the *Section 20.5.2.4: Module Configuration Register (MCR)*, the corresponding FlexRay bus driver ports are enabled and driven.

20.2 External signal description

This section lists and describes the controller signals, connected to external pins. These signals are summarized in *Table 215* and described in detail in *Section 20.2.1: Detailed signal descriptions*.

Note: The off-chip signals FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN are available on each package option. The availability of the other off-chip signals depends on the package option.

Table 215. External signal properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
<Overbar>FR_A_TX_EN	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
<Overbar>FR_B_TX_EN	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

20.2.1 Detailed signal descriptions

This section provides a detailed description of the controller signals, connected to external pins.

20.2.1.1 FR_A_RX — receive data channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

20.2.1.2 FR_A_TX — transmit data channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

20.2.1.3 <Overbar>FR_A_TX_EN — transmit enable channel A

The <Overbar>FR_A_TX_EN signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel A.

20.2.1.4 FR_B_RX — receive data channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

20.2.1.5 FR_B_TX — transmit data channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

20.2.1.6 <Overbar>FR_B_TX_EN — transmit enable channel B

The <Overbar>FR_B_TX_EN signal indicates to the FlexRay bus driver that the controller is attempting to transmit data on channel B.

20.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — strobe signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to *Section 20.6.16: Strobe signal support*.

20.3 Controller host interface clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. Since the FlexRay protocol requires data delivery at fixed points in time, the memory read cycles from the FlexRay memory must be finished after a fixed amount of time. To ensure this, a minimum frequency f_{chi} of the CHI clock is required, which is given in *Equation 17*.

Equation 17

$$f_{chi} \geq 32MHz$$

Additional requirements for the minimum frequency of the CHI clock result from the number of message buffers. These requirements are provided in *Section 20.7.3: Number of usable message buffers*.

20.4 Protocol engine clocking

The FlexRay protocol engine does not support operation with the IRC clock. Refer to The clock for the protocol engine is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the *Section 20.5.2.4: Module Configuration Register (MCR)*.

20.4.1 PLL Clocking

If the protocol engine is clocked by the internal FMPLL_1, the output frequency of the FMPLL_1 has to configured to be 80 MHz.

20.5 Memory map and register description

The controller occupies 512 bytes of address space starting at the controller's base address defined by the memory map of the MCU.

20.5.1 Memory map

The complete memory map of the controller is shown in *Table 216*.

Table 216. FlexRay memory map

Offset from FlexRay_BASE 0xFFFE_0000	Register	Location
Module Configuration and Control		
0x0000	Module Version Register (MVR)	on page 472
0x0002	Module Configuration Register (MCR)	on page 472
0x0004	System Memory Base Address High Register (SYMBADHR)	on page 474
0x0006	System Memory Base Address Low Register (SYMBADLR)	on page 474
0x0008	Strobe Signal Control Register (STBSCR)	on page 475
0x000A	Reserved	
0x000C	Message Buffer Data Size Register (MBDSR)	on page 478
0x000E	Message Buffer Segment Size and Utilization Register (MBSSUTR)	on page 478
0x0010–0x0013	Reserved	
Interrupt and Error Handling		
0x0014	Protocol Operation Control Register (POCR)	on page 479
0x0016	Global Interrupt Flag and Enable Register (GIFER)	on page 481
0x0018	Protocol Interrupt Flag Register 0 (PIFR0)	on page 483
0x001A	Protocol Interrupt Flag Register 1 (PIFR1)	on page 486
0x001C	Protocol Interrupt Enable Register 0 (PIER0)	on page 487
0x001E	Protocol Interrupt Enable Register 1 (PIER1)	on page 488
0x0020	CHI Error Flag Register (CHIERFR)	on page 489
0x0022	Message Buffer Interrupt Vector Register (MBIVEC)	on page 492
0x0024	Channel A Status Error Counter Register (CASERCR)	on page 492
0x0026	Channel B Status Error Counter Register (CBSERCR)	on page 493
Protocol Status		
0x0028	Protocol Status Register 0 (PSR0)	on page 493
0x002A	Protocol Status Register 1 (PSR1)	on page 495
0x002C	Protocol Status Register 2 (PSR2)	on page 496
0x002E	Protocol Status Register 3 (PSR3)	on page 498
0x0030	Macrotick Counter Register (MTCTR)	on page 499
0x0032	Cycle Counter Register (CYCTR)	on page 500
0x0034	Slot Counter Channel A Register (SLTCTAR)	on page 500
0x0036	Slot Counter Channel B Register (SLTCTBR)	on page 500
0x0038	Rate Correction Value Register (RTCORVR)	on page 501
0x003A	Offset Correction Value Register (OFCORVR)	on page 501

Table 216. FlexRay memory map (Continued)

Offset from FlexRay_BASE 0xFFFE_0000	Register	Location
0x003C	Combined Interrupt Flag Register (CIFRR)	on page 502
0x003E	System Memory Access Time-Out Register (SYMATOR)	on page 503
Sync Frame Counter and Tables		
0x0040	Sync Frame Counter Register (SFCNTR)	on page 504
0x0042	Sync Frame Table Offset Register (SFTOR)	on page 504
0x0044	Sync Frame Table Configuration, Control, Status Register (SFTCCSR)	on page 505
Sync Frame Filter		
0x0046	Sync Frame ID Rejection Filter Register (SFIDRFR)	on page 506
0x0048	Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)	on page 507
0x004A	Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)	on page 507
Network Management Vector		
0x004C	Network Management Vector Register 0 (NMVR0)	on page 507
0x004E	Network Management Vector Register 1 (NMVR1)	on page 507
0x0050	Network Management Vector Register 2 (NMVR2)	on page 507
0x0052	Network Management Vector Register 3 (NMVR3)	on page 507
0x0054	Network Management Vector Register 4 (NMVR4)	on page 507
0x0056	Network Management Vector Register 5 (NMVR5)	on page 507
0x0058	Network Management Vector Length Register (NMVLR)	on page 508
Timer Configuration		
0x005A	Timer Configuration and Control Register (TICCR)	on page 509
0x005C	Timer 1 Cycle Set Register (TI1CYSR)	on page 510
0x005E	Timer 1 Macrotick Offset Register (TI1MTOR)	on page 510
0x0060	Timer 2 Configuration Register 0 (TI2CR0)	on page 511
0x0062	Timer 2 Configuration Register 1 (TI2CR1)	on page 511
Slot Status Configuration		
0x0064	Slot Status Selection Register (SSSR)	on page 512
0x0066	Slot Status Counter Condition Register (SSCCR)	on page 513
Slot Status		
0x0068	Slot Status Register 0 (SSR0)	on page 515
0x006A	Slot Status Register 1 (SSR1)	on page 515
0x006C	Slot Status Register 2 (SSR2)	on page 515
0x006E	Slot Status Register 3 (SSR3)	on page 515

Table 216. FlexRay memory map (Continued)

Offset from FlexRay_BASE 0xFFFE_0000	Register	Location
0x0070	Slot Status Register 4 (SSR4)	on page 515
0x0072	Slot Status Register 5 (SSR5)	on page 515
0x0074	Slot Status Register 6 (SSR6)	on page 515
0x0076	Slot Status Register 7 (SSR7)	on page 515
0x0078	Slot Status Counter Register 0 (SSCR0)	on page 516
0x007A	Slot Status Counter Register 1 (SSCR1)	on page 516
0x007C	Slot Status Counter Register 2 (SSCR2)	on page 516
0x007E	Slot Status Counter Register 3 (SSCR3)	on page 516
MTS Generation		
0x0080	MTS A Configuration Register (MTSACFR)	on page 517
0x0082	MTS B Configuration Register (MTSBCFR)	on page 517
Shadow Buffer Configuration		
0x0084	Receive Shadow Buffer Index Register (RSBIR)	on page 518
Receive FIFO — Configuration		
0x0086	Receive FIFO Selection Register (RFSR)	on page 519
0x0088	Receive FIFO Start Index Register (RFSIR)	on page 519
0x008A	Receive FIFO Depth and Size Register (RFDSR)	on page 520
Receive FIFO - Status		
0x008C	Receive FIFO A Read Index Register (RFARIR)	on page 520
0x008E	Receive FIFO B Read Index Register (RFBRIR)	on page 521
Receive FIFO - Filter		
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)	on page 521
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)	on page 522
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)	on page 522
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)	on page 522
0x0098	Receive FIFO Range Filter Configuration Register (RFRFCFR)	on page 523
0x009A	Receive FIFO Range Filter Control Register (RFRFCTR)	on page 523
Dynamic Segment Status		
0x009C	Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)	on page 524
0x009E	Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)	on page 525
Protocol Configuration		

Table 216. FlexRay memory map (Continued)

Offset from FlexRay_BASE 0xFFFE_0000	Register	Location
0x00A0 ... 0x00DC	Protocol Configuration Register 0 (PCR0) ... Protocol Configuration Register 30 (PCR30)	*on page 527* ... *on page 534*
0x00DE–0x00FE	Reserved	
Message Buffers Configuration, Control, Status		
0x0100	Message Buffer Configuration, Control, Status Register 0 (MBCCSR0)	*on page 534*
0x0102	Message Buffer Cycle Counter Filter Register 0 (MBCCFR0)	*on page 536*
0x0104	Message Buffer Frame ID Register 0 (MBFIDR0)	*on page 537*
0x0106	Message Buffer Index Register 0 (MBIDXR0)	*on page 537*
...	...	...
0x01F8	Message Buffer Configuration, Control, Status Register 31 (MBCCSR31)	*on page 534*
0x01FA	Message Buffer Cycle Counter Filter Register 31 (MBCCFR31)	*on page 536*
0x01FC	Message Buffer Frame ID Register 31 (MBFIDR31)	*on page 537*
0x01FE	Message Buffer Index Register 31 (MBIDXR31)	*on page 537*

20.5.2 Register descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 217 provides a key for the register figures and register tables.

Table 217. Register access conventions

Convention	Description
	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

20.5.2.1 Register reset

All registers except the Message Buffer Cycle Counter Filter Registers (MBCCFRn), Message Buffer Frame ID Registers (MBFIDRn), and Message Buffer Index Registers (MBIDXRn) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in *Table 218*.

Table 218. Additional register reset conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*.
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit MBCCSRn[EDT] while the message buffer is enabled (MBCCSn[EDS] = 1) and the controller grants the disable to the application by clearing the MBCCSRn[EDS] bit.

20.5.2.2 Register write access

This section describes the write access restriction terms that apply to all registers.

20.5.2.2.1 Register write access restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in *Table 219*. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.

Table 219. Register write access restrictions

Condition	Indication	Description
Any Time	—	No write access restriction.
Disabled Mode	MCR[MEN] = 0	Write access only when the controller is in Disabled Mode.
Normal Mode	MCR[MEN] = 1	Write access only when the controller is in Normal Mode.
POC:config	PSR0[PROTSTATE] = *POC:config*	Write access only when the Protocol is in the *POC:config* state.
MB_DIS	MBCCSR[EDS] = 0	Write access only when the related Message Buffer is disabled.
MB_LCK	MBCCSRn[LCKS] = 1	Write access only when the related Message Buffer is locked.

20.5.2.2.2 Register write access requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations. For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected

20.5.2.2.3 Internal register access

The following memory mapped registers are used to access multiple internal registers.

- *Section 20.5.2.6: Strobe Signal Control Register (STBSCR)*
- *Section 20.5.2.44: Slot Status Selection Register (SSSR)*
- *Section 20.5.2.45: Slot Status Counter Condition Register (SSCCR)*
- *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field selects the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

20.5.2.3 Module Version Register (MVR)

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHIVER								PEVER							
W																
Reset	1	0	0	1	1	1	0	0	0	1	1	0	0	1	1	0

Figure 216. Module Version Register (MVR)

This register provides the controller version number. The module version number is derived from the CHI version number and the PE version number.

Table 220. MVR field descriptions

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the controller host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

20.5.2.4 Module Configuration Register (MCR)

Base + 0x0002

Write: MEN, SBFF, SCM, CHB, CHA, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	0	R*	0	0	0	CLK SEL	BITRATE			0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 217. Module Configuration Register (MCR)

This register defines the global configuration of the controller.

Table 221. MCR field descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the controller is in the Disabled Mode. The application requests the controller to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see *Section 20.1.4: Modes of operation*. 0 Write: ignored, controller disable not possible. Read: controller disabled. 1 Write: enable controller. Read: controller enabled. **Note:** If the controller is enabled it cannot be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the controller in case of a system bus failure. 0 Continue normal operation. 1 Transition to freeze mode.
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the controller as described in *Section 20.6.10: Channel device modes*. 0 controller works in dual channel device mode. 1 controller works in single channel device mode.
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given *Table 222*.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see *Section 20.6.15: Sync frame filtering*. 0 Synchronization frame filtering disabled. 1 Synchronization frame filtering enabled.
R*	**Reserved** — This bit is reserved. It is read as 0. Application must not write 1 to this bit.
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the bit rate of the FlexRay channels according to *Table 223*.

Table 222. FlexRay channel selection

SCM	CHB	CHA	Description
Dual channel device modes			
0	0	0	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel B

Table 222. FlexRay channel selection(Continued)

SCM	CHB	CHA	Description
Single channel device mode			
1	0	0	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN not driven by controller ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN not driven by controller
	0	1	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN not driven by controller
	1	0	ports FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN driven by controller - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and <Overbar>FR_A_TX_EN not driven by controller
	1	1	reserved

Table 223. FlexRay channel bit rate selection

MCR[BITRATE]	FlexRay channel bit rate [Mbit/s]
000	10.0
001	5.0
010	2.5
011	8.0
100	reserved
101	reserved
110	reserved
111	reserved

20.5.2.5 System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)

Base + 0x0004 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SYS_MEM_BASE_ADDR[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 218. System Memory Base Address High Register (SYMBADHR)

Base + 0x0006 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SYS_MEM_BASE_ADDR[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 219. System Memory Base Address Low Register (SYMBADLR)

Note: The system memory base address must be set before the controller is enabled.

The system memory base address registers define the base address of the FlexRay memory within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 224. SYMBADHR and SYMBADLR field descriptions

Field	Description
SYS_MEM_ BASE_ADDR	This base address will be added to all system memory offset values stored in registers or calculated in the controller before the controller accesses the system memory via its bus master interface. The system memory base address must be aligned to an 16-byte boundary.

20.5.2.6 Strobe Signal Control Register (STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SEL							0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 220. Strobe Signal Control Register (STBSCR)

This register assigns the individual protocol timing related strobe signals given in *Table 226* to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to *Section 20.6.16: Strobe signal support*.

Note: In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 225. STBSCR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in *Table 226* to be enabled or disabled and assigned to one of the four strobe ports given in *Table 226*.
ENB	**Strobe Signal Enable** — This control bit enables and disables the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 Assign selected signal to FR_DBG[0]. 01 Assign selected signal to FR_DBG[1]. 10 Assign selected signal to FR_DBG[2]. 11 Assign selected signal to FR_DBG[3].

Table 226. Strobe signal mapping

SEL		Description	Channel	Type	Offset(1)	Reference
dec	hex					
0	0x00	poc_startup_state[0] (for coding see PSR0[4])	—	value	0	MT start
1	0x01	poc_startup_state[1] (for coding see PSR0[5])				
2	0x02	poc_startup_state[2] (for coding see PSR0[6])				
3	0x03	poc_startup_state[3] (for coding see PSR0[7])				
4	0x04	poc_state[0] (for coding see PSR0[8])				
5	0x05	poc_state[1] (for coding see PSR0[9])				
6	0x06	poc_state[2] (for coding see PSR0[10])				
7	0x07	channel idle indicator	A	level	+5	FR_A_RX
8	0x08		B			FR_B_RX
9	0x09	receive data after glitch filtering	A	value	+4	FR_A_RX
10	0x0A		B			FR_B_RX
11	0x0B	synchronization edge strobe	A	pulse	+4	FR_A_RX
12	0x0C		B			FR_B_RX
13	0x0D	header received	A	pulse	+4	FR_A_RX
14	0x0E		B			FR_B_RX
15	0x0F	wakeup symbol decoded	A	pulse	+5	FR_A_RX
16	0x10		B			FR_B_RX
17	0x11	MTS or CAS symbol decoded	A	pulse	+4	FR_A_RX
18	0x12		B			FR_B_RX
19	0x13	frame decoded	A	pulse	+4	FR_A_RX
20	0x14		B			FR_B_RX
21	0x15	channel idle detected	A	pulse	+4	FR_A_RX
22	0x16		B			FR_B_RX
23	0x17	start of communication element detected	A	pulse	+4	FR_A_RX
24	0x18		B			FR_B_RX
25	0x19	potential frame start channel	A	pulse	+4	FR_A_RX
26	0x1A		B			FR_B_RX
27	0x1B	wakeup collision detected	A	pulse	+5	FR_A_RX
28	0x1C		B			FR_B_RX
29	0x1D	content error detected	A	level	+4	FR_A_RX
30	0x1E		B			FR_B_RX
31	0x1F	syntax error detected	A	pulse	+4	FR_A_RX
32	0x20		B			FR_B_RX
33	0x21	start transmission of wakeup pattern	A	pulse	-1	FR_A_TX
34	0x22		B			FR_B_TX
35	0x23	start transmission of MTS or CAS symbol	A	pulse	-1	FR_A_TX
36	0x24		B			FR_B_TX
37	0x25	start of transmission	A	pulse	-1	FR_A_TX
38	0x26		B			FR_B_TX
39	0x27	end of transmission	A	pulse	-1	FR_A_TX
40	0x28		B			FR_B_TX

Table 226. Strobe signal mapping(Continued)

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
41	0x29	static segment indicator	—	level	0	MT start
42	0x2A	dynamic segment indicator	—	level	0	MT start
43	0x2B	symbol window indicator	—	level	0	MT start
44	0x2C	NIT indicator	—	level	0	MT start
45	0x2D	action point	—	pulse	-1	FR_A_TX
46	0x2E	sync calculation complete[2]	—	pulse	—	—
47	0x2F	start of offset correction	—	pulse	-2	MT start
48	0x30	cycle count[0]	—	value	-2	MT start
49	0x31	cycle count[1]				
50	0x32	cycle count[2]				
51	0x33	cycle count[3]				
52	0x34	cycle count[4]				
53	0x35	cycle count[5]				
54	0x36	slot count[0]	A	value	0	MT start
55	0x37	slot count[1]				
56	0x38	slot count[2]				
57	0x39	slot count[3]				
58	0x3A	slot count[4]				
59	0x3B	slot count[5]				
60	0x3C	slot count[6]				
61	0x3D	slot count[7]				
62	0x3E	slot count[8]				
63	0x3F	slot count[9]				
64	0x40	slot count[10]				
65	0x41	slot count[0]	B	value	0	MT start
66	0x42	slot count[1]				
67	0x43	slot count[2]				
68	0x44	slot count[3]				
69	0x45	slot count[4]				
70	0x46	slot count[5]				
71	0x47	slot count[6]				
72	0x48	slot count[7]				
73	0x49	slot count[8]				
74	0x4A	slot count[9]				
75	0x4B	slot count[10]				
76	0x4C	cycle start	—	pulse	0	MT start
77	0x4D	slot start	A	pulse	0	MT start
78	0x4E		B			
79	0x4F	minislot start	—	pulse	0	MT start
80	0x50	arm	—	value	+1	MT start
81	0x51	mt	—	value	+1	MT start

1. Given in PE clock cycles
2. Indicates internal PE event not directly related to FlexRay bus timing

20.5.2.7 Message Buffer Data Size Register (MBDSR)

Base + 0x000C Write: *POC:config*

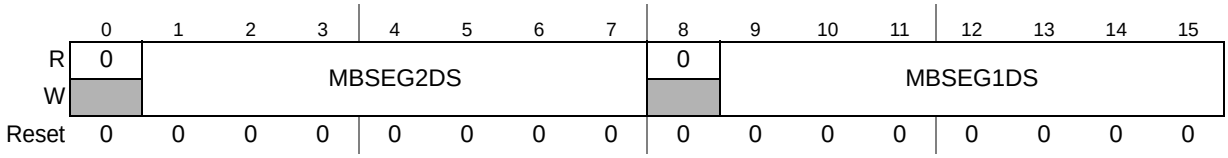


Figure 221. Message Buffer Data Size Register (MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The controller provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 227. MBDSR field descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

20.5.2.8 Message Buffer Segment Size and Utilization Register (MBSSUTR)

Base + 0x000E Write: *POC:config*

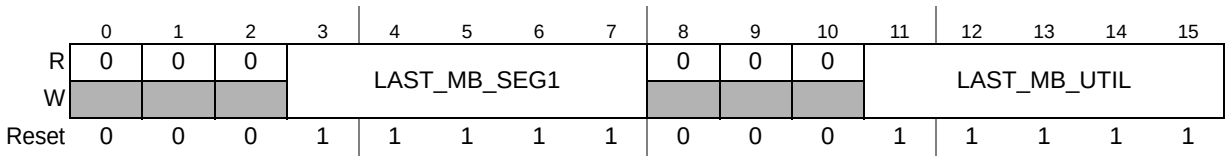


Figure 222. Message Buffer Segment Size and Utilization Register (MBSSUTR)

This register defines the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 228. MBSSUTR field descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with n <= LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1+1 individual message buffers. **Note:** The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers MBCCSRn, MBCCFRn, MBFIDRn, MBIDXRn with LAST_MB_SEG1 < n < 32. **Note:** If LAST_MB_SEG1 = 31 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number n <= LAST_MB_UTIL. **Note:** If LAST_MB_UTIL=LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

20.5.2.9 Protocol Operation Control Register (POCR)

Base + 0x0014 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 223. Protocol Operation Control Register (POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see *Section 20.7.4: Protocol control command execution*.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to *Section 20.6.11: External clock synchronization*.

Table 229. POCR field descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field triggers the application of the external offset correction value defined in the *Section 20.5.2.64.30: Protocol Configuration Register 29 (PCR29)*. 00 Do not apply external offset correction value. 01 Reserved. 10 Subtract external offset correction value. 11 Add external offset correction value.
ERC_AP	**External Rate Correction Application** — This field triggers application of the external rate correction value defined in the *Section 20.5.2.64.22: Protocol Configuration Register 21 (PCR21)* 00 Do not apply external rate correction value. 01 Reserved. 10 Subtract external rate correction value. 11 Add external rate correction value.
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The controller sets this status bit when the application has issued a protocol control command via the POCCMD field. The controller clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the controller ignores this command, sets the protocol command ignored error flag PCMI_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*, and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The controller sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed(1) transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 Reserved. 1010 Reserved. 1011 Reserved. 1100 RESET(2) — Immediately reset the Protocol Engine (see *Section 20.7.5: Protocol reset command*). 1101 Reserved. 1110 Reserved. 1111 Reserved.

1. Delayed means on completion of current communication cycle.
2. Additional to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*

20.5.2.10 Global Interrupt Flag and Enable Register (GIFER)

Base + 0x0016 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUPIF	FNEBIF	FNEAIF	RBIF	TBIF	MIE	PRIE	CHIE	WUPIE	FNEBIE	FNEAIE	RBIE	TBIE
W				w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 224. Global Interrupt Flag and Enable Register (GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is shown in *Figure 359*. For more details on interrupt generation, see *Section 20.6.20: Interrupt support*. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 230. GIFER field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if at least one of the other interrupt flags is in this register is asserted and the related interrupt enable is asserted, too. The controller generates the module interrupt request if MIE is asserted. 0 No interrupt flag is asserted or no interrupt enable is set. 1 At least one of the other interrupt flags in this register is asserted and the related interrupt bit is asserted, too.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* and *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)* is asserted and the related interrupt enable flag is asserted, too. The controller generates the combined protocol interrupt request if the PRIE flag is asserted. 0 All individual protocol interrupt flags are equal to 0 or no interrupt enable bit is set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is asserted and the chi error interrupt enable GIFER[CHIE] is asserted. The controller generates the combined CHI error interrupt if the CHIE flag is asserted, too. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled. 1 At least one CHI error flag is asserted and chi error interrupt is enabled.
WUPIF	**Wakeup Interrupt Flag** — This flag is set when the controller has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the *Section 20.5.2.22: Protocol Status Register 3 (PSR3)*. The controller generates the wakeup interrupt request if the WUPIE flag is asserted. 0 No wakeup condition or interrupt disabled. 1 Wakeup symbol received on FlexRay bus and interrupt enabled.

Table 230. GIFER field descriptions(Continued)

Field	Description
FNEBIF	**Receive FIFO channel B Not Empty Interrupt Flag** — This flag is set when the receive FIFO for channel B is not empty. If the application writes 1 to this bit, the controller updates the FIFO status, increments or wraps the FIFO read index in the *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)* and clears the interrupt flag if the FIFO B is now empty. If the FIFO is still not empty, the controller sets this flag again. The controller generates the Receive FIFO B Not empty interrupt if the FNEBIE flag is asserted. 0 Receive FIFO B is empty or interrupt is disabled. 1 Receive FIFO B is not empty and interrupt enabled.
FNEAIF	**Receive FIFO channel A Not Empty Interrupt Flag** — This flag is set when the receive FIFO for channel A is not empty. If the application writes 1 to this bit, the controller updates the FIFO status, increments or wraps the FIFO read index in the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* and clears the interrupt flag if the FIFO A is now empty. If the FIFO is still not empty, the controller sets this flag again. The controller generates the Receive FIFO A Not empty interrupt if the FNEAIE flag is asserted. 0 Receive FIFO A is empty or interrupt is disabled. 1 Receive FIFO A is not empty and interrupt enabled.
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)* are asserted. The application cannot clear this RBIF flag directly. This flag is cleared by the controller when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag asserted. 1 At least one individual receive message buffer has the MBIF and MBIE flag asserted.
TBIF	**Transmit Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)* are equal to 1. The application cannot clear this TBIF flag directly. This flag is cleared by the controller when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the host has cleared the interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag asserted. 1 At least one individual transmit message buffer has the MBIF and MBIE flag asserted.
MIE	**Module Interrupt Enable** — This flag controls if the module interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This flag controls if the protocol interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This flag controls if the CHI interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	**Wakeup Interrupt Enable** — This flag controls if the wakeup interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 230. GIFER field descriptions(Continued)

Field	Description
FNEBIE	**Receive FIFO channel B Not Empty Interrupt Enable** — This flag controls if the receive FIFO B interrupt line is asserted when the FNEBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FNEAIE	**Receive FIFO channel A Not Empty Interrupt Enable** — This flag controls if the receive FIFO A interrupt line is asserted when the FNEAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
RBIE	**Receive Buffer Interrupt Enable** — This flag controls if the receive buffer interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	**Transmit Interrupt Enable** — This flag controls if the transmit buffer interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

20.5.2.11 Protocol Interrupt Flag Register 0 (PIFR0)

Base + 0x0018 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 225. Protocol Interrupt Flag Register 0 (PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 231. PIFR0 field description

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the *Section 20.5.2.64.15: Protocol Configuration Register 14 (PCR14)* and *Section 20.5.2.64.16: Protocol Configuration Register 15 (PCR15)* when the CONFIG_COMPLETE command was sent by the application via the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*. If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the *Section 20.5.2.64.4: Protocol Configuration Register 3 (PCR3)*. 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the *Section 20.5.2.64.10: Protocol Configuration Register 9 (PCR9)* and the *rate_correction_out* field in the *Section 20.5.2.64.15: Protocol Configuration Register 14 (PCR14)*. 0 No such event. 1 Offset or rate correction limit reached.

Table 231. PIFR0 field description(Continued)

Field	Description
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the *Section 20.5.2.64.31: Protocol Configuration Register 30 (PCR30)*. 0 No such event. 1 More than *node_sync_max* sync frames detected. **Note:** Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

20.5.2.12 Protocol Interrupt Flag Register 1 (PIFR1)

Base + 0x001A Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 226. Protocol Interrupt Flag Register 1 (PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 232. PIFR1 field descriptions

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the *Section 20.5.2.19: Protocol Status Register 0 (PSR0)* is changed by the controller. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*, and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see *Section 20.7.4: Protocol control command execution*. 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the controller has detected a communication failure between the protocol engine and the controller host interface 0 No such event. 1 Protocol Engine Communication Failure detected.
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the *Section 20.5.2.19: Protocol Status Register 0 (PSR0)* has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding *Section 20.5.2.47: Slot Status Counter Registers (SSCR0–SSCR3)* is incremented. 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the even cycle. 0 No such event. 1 Sync frame measurement table written
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the controller has written the sync frame measurement / ID tables into the FlexRay memory for the odd cycle. 0 No such event. 1 Sync frame measurement table written

20.5.2.13 Protocol Interrupt Enable Register 0 (PIER0)

Base + 0x001C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 227. Protocol Interrupt Enable Register 0 (PIER0)

This register defines whether or not the individual interrupt flags defined in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* can generate a protocol interrupt request.

Table 233. PIER0 field descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

Table 233. PIER0 field descriptions

Field	Description
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

20.5.2.14 Protocol Interrupt Enable Register 1 (PIER1)

Base + 0x001E — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PEC F_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 228. Protocol Interrupt Enable Register 1 (PIER1)

This register defines whether or not the individual interrupt flags defined in *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)* can generate a protocol interrupt request.

Table 234. PIER1 field descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled. 1 interrupt request generation enabled.

20.5.2.15 CHI Error Flag Register (CHIERFR)

Base + 0x0020 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	DBL_EF	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 229. CHI Error Flag Register (CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*.

Table 235. CHIERFR field descriptions

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the *Section 20.5.2.9: Protocol Operation Control Register (POCR)* while the BSY flag is equal to 1. In this case the command is ignored by the controller and is lost. 0 No such error 1 POC command ignored
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the Receive FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the Receive FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 Receive FIFO overrun on channel B has been detected
MSB_EF	**Message Buffer Search Error Flag** — This flag is set if the message buffer search engine is still running while the next search cycle must be started due to the FlexRay protocol timing. In this case, not all message buffers are considered while searching. 0 No such event 1 Search engine active while search start appears
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*. If the application writes to a MBCCSRn register with n > LAST_MB_UTIL, the controller ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*. 0 No such event 1 Non-utilized message buffer enabled.
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the controller due to internal operations. In that case, the controller does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected

Table 235. CHIERFR field descriptions(Continued)

Field	Description
DBL_EF	**Double Transmit Message Buffer Lock Error Flag** — This flag is set if the application tries to lock the transmit side of a double transmit message buffer. In this case, the controller does not grant the lock to the transmit side of a double transmit message buffer. 0 No such event 1 Double transmit buffer lock error occurred
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see *Section 20.6.19.2: System bus access timeout*). 0 No such event 1 System bus access not finished in time
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the *Section 20.5.2.64.25: Protocol Configuration Register 24 (PCR24)*. 0 No such error occurred 1 Dynamic payload length error occurred
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a single or double transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the *Section 20.5.2.64.20: Protocol Configuration Register 19 (PCR19)*. 0 No such error occurred 1 Static payload length error occurred
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the *Section 20.5.2.38: Network Management Vector Length Register (NMVLR)*. In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see *Section 20.5.2.37: Network Management Vector Registers (NMVR0–NMVR5)*) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the controller (see *Section 20.6.19.1: System bus illegal address access*). 0 No such event 1 Illegal system bus address accessed

20.5.2.16 Message Buffer Interrupt Vector Register (MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TBIVEC					0	0	0	RBIVEC				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 230. Message Buffer Interrupt Vector Register (MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 236. MBIVEC field descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

20.5.2.17 Channel A Status Error Counter Register (CASERCR)

Base + 0x0024 **A**dditional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 231. Channel A Status Error Counter Register (CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see *Section 20.6.18: Slot status monitoring*.

Table 237. CASERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

20.5.2.18 Channel B Status Error Counter Register (CBSERCR)

Base + 0x0026 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 232. Channel B Status Error Counter Register (CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The controller increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see *Section 20.6.18: Slot status monitoring*.

Table 238. CBSERCR field descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

20.5.2.19 Protocol Status Register 0 (PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 233. Protocol Status Register 0 (PSR0)

This register provides information about the current protocol status.

Table 239. PSR0 field descriptions

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integration listen* 0110 reserved 0111 *POC:initialize schedule* 1000 reserved 1001 reserved 1010 *POC:coldstart consistency check* 1011 reserved 1100 reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

20.5.2.20 Protocol Status Register 1 (PSR1)

Base + 0x002A Additional Reset: CSAA, CSP, CPN: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 234. Protocol Status Register 1 (PSR1)

Table 240. PSR1 field descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: 'set coldstart abort indicator in CHI' This flag is set when the controller has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	**Leading Cold Start Path** — This status bit is set when the controller has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 *POC:normal active* reached from *POC:startup* state via leading cold start path
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the controller will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the controller has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the controller was starting up the cluster. 0 No such event 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when controller receives the HALT command from the application via the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*. The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 HALT command received
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the controller has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The controller clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the *Section 20.5.2.64.13: Protocol Configuration Register 12 (PCR12)*.

20.5.2.21 Protocol Status Register 2 (PSR2)

Base + 0x002C Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 235. Protocol Status Register 2 (PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the controller after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the controller after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the controller after the end of the static segment and before the end of the current communication cycle.

Table 241. PSR2 field descriptions

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected

Table 241. PSR2 field descriptions(Continued)

Field	Description
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the *Section 20.5.2.64.9: Protocol Configuration Register 8 (PCR8)*. The controller resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

20.5.2.22 Protocol Status Register 3 (PSR3)

Base + 0x002E — Additional Reset: RUN Command — Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 236. Protocol Status Register 3 (PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 242. PSR3 field descriptions

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received

Table 242. PSR3 field descriptions (Continued)

Field	Description
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

20.5.2.23 Macrotick Counter Register (MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 237. Macrotick Counter Register (MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 243. MTCTR field descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

20.5.2.24 Cycle Counter Register (CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 238. Cycle Counter Register (CYCTR)

This register provides the number of the current communication cycle.

Table 244. CYCTR field descriptions

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

20.5.2.25 Slot Counter Channel A Register (SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 239. Slot Counter Channel A Register (SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 245. SLTCTAR field descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

20.5.2.26 Slot Counter Channel B Register (SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 240. Slot Counter Channel B Register (SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 246. SLTCTBR field descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

20.5.2.27 Rate Correction Value Register (RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

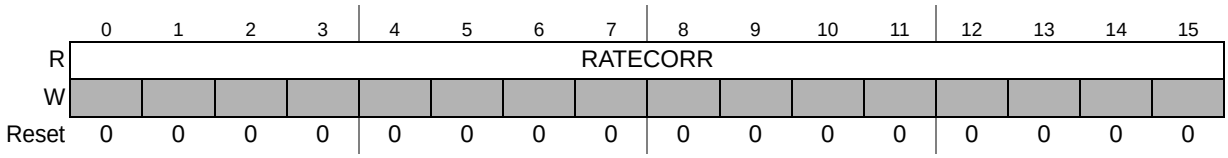


Figure 241. Rate Correction Value Register (RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT of each odd numbered communication cycle.

Table 247. RTCORVR field descriptions

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the *Section 20.5.2.64.14: Protocol Configuration Register 13 (PCR13)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)*. **Note:** If the controller was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

20.5.2.28 Offset Correction Value Register (OFCORVR)

Base + 0x003A Additional Reset: RUN Command

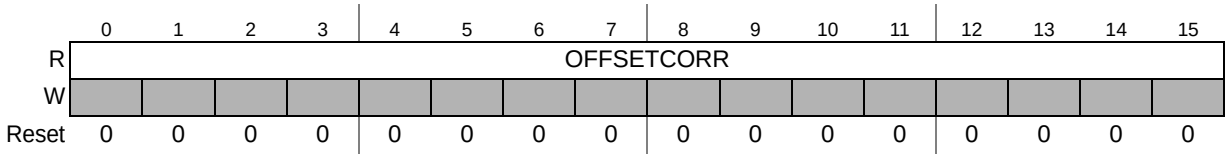


Figure 242. Offset Correction Value Register (OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The controller updates this register during the NIT.

Table 248. OFCORVR field descriptions

Field	Description
OFFSET-CORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the *Section 20.5.2.64.30: Protocol Configuration Register 29 (PCR29)*, the clock correction reached limit interrupt flag CCL_IF is set in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)*. **Note:** If the controller was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

20.5.2.29 Combined Interrupt Flag Register (CIFRR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUPIF	FNEBIF	FNEAIF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 243. Combined Interrupt Flag Register (CIFRR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is shown in *Figure 360*. The individual interrupt flags WUPIF, FNEBIF, and FNEAIF are copies of corresponding flags in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)* and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*.

Note: The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER).

Table 249. CIFRR field descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted. 1 At least one interrupt source has its interrupt flag asserted.
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* or *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)* is equal to 1. 0 All individual protocol interrupt flags are equal to 0. 1 At least one of the individual protocol interrupt flags is equal to 1.
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is equal to 1. 0 All CHI error flags are equal to 0. 1 At least one CHI error flag is equal to 1.
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as GIFER[WUPIF]
FNEBIF	**Receive FIFO channel B Not Empty Interrupt Flag** — Provides the same value as GIFER[FNEBI]
FNEAIF	**Receive FIFO channel A Not Empty Interrupt Flag** — Provides the same value as GIFER[FNEAIF]
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)* is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual single or double transmit message buffers (MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)* is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

20.5.2.30 System Memory Access Time-Out Register (SYMATOR)

Base + 0x003E — Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	TIMEOUT				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 244. System Memory Access Time-Out Register (SYMATOR)

Table 250. SYMATOR field descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines the maximum amount of time to finish a system bus access in order to ensure correct frame transmission and reception (see *Section 20.6.19.2: System bus access timeout*).

20.5.2.31 Sync Frame Counter Register (SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 245. Sync Frame Counter Register (SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

Note: If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the controller will not update the fields SFEVB and SFEVA.

Note: If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the controller will not update the values SFODB and SFODA.

Table 251. SFCNTR field descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

20.5.2.32 Sync Frame Table Offset Register (SFTOR)

Base + 0x0042 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFT_OFFSET[15:1]															0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 246. Sync Frame Table Offset Register (SFTOR)

This register defines the FlexRay Memory related offset for sync frame tables. For more details, see *Section 20.6.12: Sync frame ID and sync frame deviation tables*.

Table 252. SFTOR field description

Field	Description
SFTOR	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the FlexRay Memory. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

20.5.2.33 Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

Base + 0x0044 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDVEN	SIDEN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 247. Sync Frame Table Configuration, Control, Status Register (SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see *Section 20.6.12: Sync frame ID and sync frame deviation tables*.

Table 253. SFTCCSR field descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit locks and unlocks the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit locks and unlocks the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).

Table 253. SFTCCSR field descriptions

Field	Description
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The controller clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the controller writes continuously or only one pair of Sync Frame Tables into the FlexRay memory. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the controller writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the FlexRay memory. In this case, the controller clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the controller writes continuously the enabled Sync Frame Tables into the FlexRay memory. 0 Write continuously pairs of enabled Sync Frame Tables into FlexRay memory. 1 Write only one pair of enabled Sync Frame Tables into FlexRay memory.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the controller to write the Sync Frame Deviation Tables into the FlexRay memory. 0 Do not write Sync Frame Deviation Tables. 1 Write Sync Frame Deviation Tables into FlexRay memory. **Note:** If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the controller to write the Sync Frame ID Tables into the FlexRay memory. 0 Do not write Sync Frame ID Tables. 1 Write Sync Frame ID Tables into FlexRay memory.

20.5.2.34 Sync Frame ID Rejection Filter Register (SFIDRFR)

Base + 0x0046 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 248. Sync Frame ID Rejection Filter Register (SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the controller accepts the sync frame in the current cycle.

Table 254. SFIDRFR field descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see *Section 20.6.15.2: Sync frame rejection filtering*.

20.5.2.35 Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

Base + 0x0048 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FVAL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 249. Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see *Section 20.6.15: Sync frame filtering*.

Table 255. SFIDAFVR field descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

20.5.2.36 Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

Base + 0x004A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FMSK									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 250. Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see *Section 20.6.15.1: Sync frame acceptance filtering*.

Table 256. SFIDAFMR field descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

20.5.2.37 Network Management Vector Registers (NMVR0–NMVR5)

Base + 0x004C (NMVR0)
Base + 0x004E (NMVR1)
Base + 0x0050 (NMVR2)
Base + 0x0052 (NMVR3)
Base + 0x0054 (NMVR4)
Base + 0x0056 (NMVR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMVP[15:8]								NMVP[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 251. Network Management Vector Registers (NMVR0–NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the *Section 20.5.2.38: Network Management Vector Length Register (NMVLR)*. If NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the *Section 20.5.2.37: Network Management Vector Registers (NMVR0–NMVR5)*, which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 257. NMVR[0:5] field descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the *Section 20.5.2.37: Network Management Vector Registers (NMVR0–NMVR5)* and the receive message buffer payload bytes in NMV[0:11] is shown in *Table 258*.

Table 258. Mapping of NMVRn to received payload bytes NMVn

NMVRn Register	NMVn Received Payload
NMVR0[NMVP[15:8]]	NMV0
NMVR0[NMVP[7:0]]	NMV1
NMVR1[NMVP[15:8]]	NMV2
NMVR1[NMVP[7:0]]	NMV3
...	
NMVR5[NMVP[15:8]]	NMV10
NMVR5[NMVP[7:0]]	NMV11

20.5.2.38 Network Management Vector Length Register (NMVLR)

Base + 0x0058 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 252. Network Management Vector Length Register (NMVLR)

This register defines the length of the network management vector in bytes.

Table 259. NMVLR field descriptions

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

20.5.2.39 Timer Configuration and Control Register (TICCR)

Base + 0x005A

Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_ CFG	T2_ REP	0	0	0	T2ST	0	0	0	T1_ REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 253. Timer Configuration and Control Register (TICCR)

This register configures and controls the two timers T1 and T2. For timer details, see *Section 20.6.17: Timer support*. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 260. TICCR field descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the time base mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	**Timer T2 Stop** — This trigger bit stops timer T2. 0 no effect 1 stop timer T2
T2TR	**Timer T2 Trigger** — This trigger bit starts timer T2. 0 no effect 1 start timer T2
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	**Timer T1 Stop** — This trigger bit stops timer T1. 0 no effect 1 stop timer T1
T1TR	**Timer T1 Trigger** — This trigger bit starts timer T1. 0 no effect 1 start timer T1
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

Note: Both timers are deactivated immediately when the protocol enters a state different from POC:normal active or POC:normal passive.

20.5.2.40 Timer 1 Cycle Set Register (TI1CYSR)

Base + 0x005C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 254. Timer 1 Cycle Set Register (TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to *Section 20.6.17.1: Absolute timer T1*.

Table 261. TI1CYSR field descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

20.5.2.41 Timer 1 Macrotick Offset Register (TI1MTOR)

Base + 0x005E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 255. Timer 1 Macrotick Offset Register (TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to *Section 20.6.17.1: Absolute timer T1*.

Table 262. TI1MTOR field descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

Note: If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

20.5.2.42 Timer 2 Configuration Register 0 (TI2CR0)

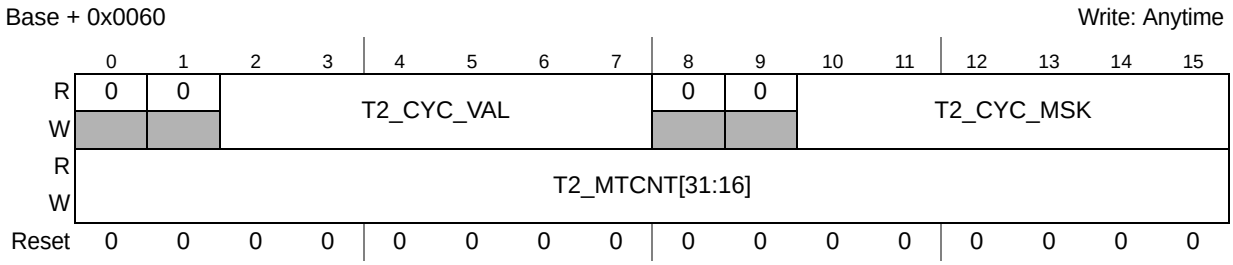


Figure 256. Timer 2 Configuration Register 0 (TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the *Section 20.5.2.39: Timer Configuration and Control Register (TICCR)*. For a detailed description of timer T2, refer to *Section 20.6.17.2: Absolute / relative timer T2*.

Table 263. TI2CR0 field descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

Note: If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

20.5.2.43 Timer 2 Configuration Register 1 (TI2CR1)

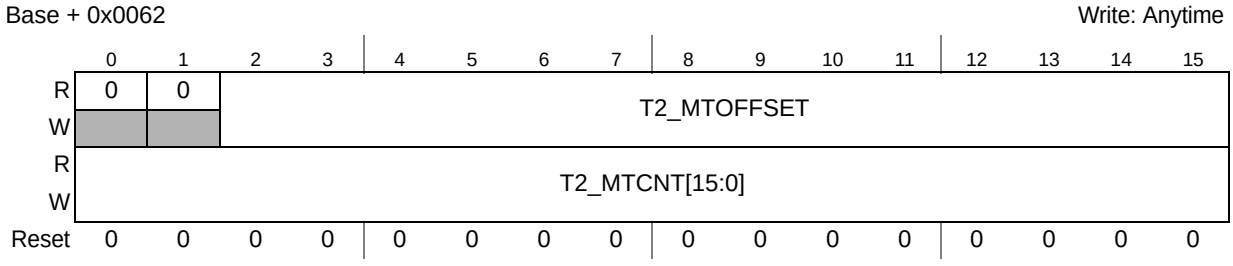


Figure 257. Timer 2 Configuration Register 1 (TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the *Section 20.5.2.39: Timer Configuration and Control Register (TICCR)*. For a detailed description of timer T2, refer to *Section 20.6.17.2: Absolute / relative timer T2*.

Table 264. TI2CR1 field descriptions

Field	Description
Fields for absolute timer T2 (TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

Note: If timer T2 is configured as an absolute timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.

Note: If timer T2 is configured as a relative timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

20.5.2.44 Slot Status Selection Register (SSSR)

Base + 0x0064 — 16-bit write access required — Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	SLOTNUMBER										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 258. Slot Status Selection Register (SSSR)

This register accesses the four internal non memory-mapped slot status selection registers SSSR0 to SSSR3. Each internal registers selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding *Section 20.5.2.46: Slot Status Registers (SSR0–SSR7)* according to *Table 266*. For a detailed description of slot status monitoring, refer to *Section 20.6.18: Slot status monitoring*.

Table 265. SSSR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 select SSSR0. 01 select SSSR1. 10 select SSSR2. 11 select SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. **Note:** If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 266. Mapping between SSSRn and SSRn

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by SSSRn for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
SSSR0	SSR0[15:8]	SSR0[7:0]	SSR1[15:8]	SSR1[7:0]
SSSR1	SSR2[15:8]	SSR2[7:0]	SSR3[15:8]	SSR3[7:0]
SSSR2	SSR4[15:8]	SSR4[7:0]	SSR5[15:8]	SSR5[7:0]
SSSR3	SSR6[15:8]	SSR6[7:0]	SSR7[15:8]	SSR7[7:0]

20.5.2.45 Slot Status Counter Condition Register (SSCCR)

Base + 0x0066 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 259. Slot Status Counter Condition Register (SSCCR)

This register accesses and programs the four internal non-memory mapped Slot Status Counter Condition Registers SSCCR0 to SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding *Section 20.5.2.47: Slot Status Counter Registers (SSCR0–SSCR3)*. The correspondence is given in *Table 268*. For a detailed description of slot status counters, refer to *Section 20.6.18.4: Slot status counter registers*.

Table 267. SSCCR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 select SSCCR0. 01 select SSCCR1. 10 select SSCCR2. 11 select SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.

Table 267. SSCCR field descriptions

Field	Description
VFR	**Valid Frame Restriction** — This bit restricts the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit restricts the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit restricts the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit restricts the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUS MASK[3:0]	**Slot Status Mask** — This bit field enables the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 268. Mapping between internal SSCCRn and SSCRn

Condition Register	Condition Defined for Register
SSCCR0	SSCR0
SSCCR1	SSCR1
SSCCR2	SSCR2
SSCCR3	SSCR3

20.5.2.46 Slot Status Registers (SSR0–SSR7)

Base + 0x0068 (SSR0)
Base + 0x006A (SSR1)
Base + 0x006C (SSR2)
Base + 0x006E (SSR3)
Base + 0x0070 (SSR4)
Base + 0x0072 (SSR5)
Base + 0x0074 (SSR6)
Base + 0x0076 (SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 260. Slot Status Registers (SSR0–SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the *Section 20.5.2.44: Slot Status Selection Register (SSSR)*. Each register is updated after the end of the corresponding slot as shown in *Figure 357*. The register bits are directly related to the protocol variables and described in more detail in *Section 20.6.18: Slot status monitoring*.

Table 269. SSR0–SSR7 field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

Table 269. SSR0–SSR7 field descriptions

Field	Description
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

20.5.2.47 Slot Status Counter Registers (SSCR0–SSCR3)

Base + 0x0078 (SSCR0)
Base + 0x007A (SSCR1)
Base + 0x007C (SSCR2)
Base + 0x007E (SSCR3)

Additional Reset: RUN Command

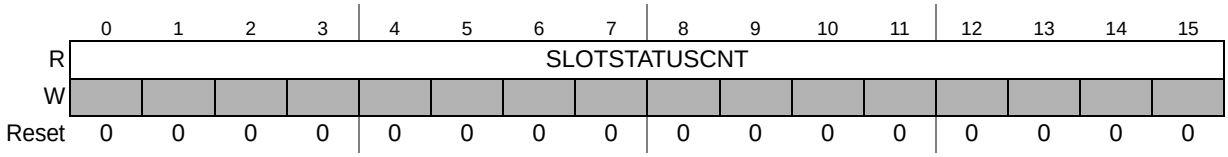


Figure 261. Slow Status Counter Registers (SSCR0–SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register SSCCRn, which can be programmed by using the *Section 20.5.2.45: Slot Status Counter Condition Register (SSCCR)*. For more details, see *Section 20.6.18.4: Slot status counter registers*.

Note: If the counter has reached its maximum value 0xFFFF and is in the multicycle mode, i.e., SSCCRn[MCY] = 1, the counter is not reset to 0x0000. The application can reset the

counter by clearing the SSCCRn[MCY] bit and waiting for the next cycle start, when the controller clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 270. SSCR0–SSCR3 field descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

20.5.2.48 MTS A Configuration Register (MTSACFR)

Base + 0x0080 Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

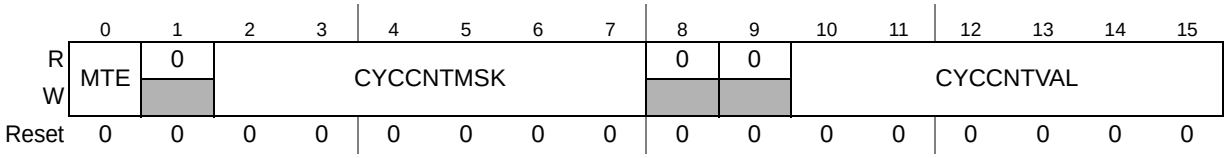


Figure 262. MTS A Configuration Register (MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see *Section 20.6.13: MTS generation*.

Table 271. MTSACFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit enables and disables the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

20.5.2.49 MTS B Configuration Register (MTSBCFR)

Base + 0x0082 Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

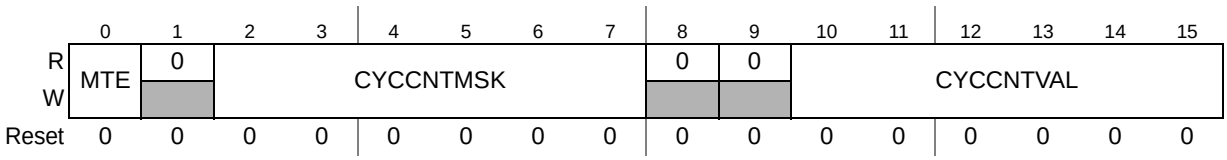


Figure 263. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see *Section 20.6.13: MTS generation*.

Table 272. MTSBCFR field descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit enables and disables the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled. 1 MTS transmission enabled.
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

20.5.2.50 Receive Shadow Buffer Index Register (RSBIR)

Base + 0x0084 16-bit write access required Write: WMD, SEL: Any Time
RSBIDX: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	0	0	RSBIDX					
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 264. Receive Shadow Buffer Index Register (RSBIR)

This register provides and retrieves the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to *Section 20.6.6.3.7: Receive Shadow Buffers Concept*.

Table 273. RSBIR field descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field selects the internal receive shadow buffer index register for access. 00 RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
RSBIDX	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The controller uses this index to determine the physical location of the shadow buffer header field in the FlexRay memory. The controller will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. controller: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index.

20.5.2.51 Receive FIFO Selection Register (RFSR)

Base + 0x0086 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SEL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 265. Receive FIFO Selection Register (RFSR)

This register selects a receiver FIFO for subsequent access through the receiver FIFO configuration registers summarized in *Table 274*.

Table 274. SEL controlled receiver FIFO registers

Register
Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)
Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)
Section 20.5.2.56: Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)
Section 20.5.2.57: Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)
Section 20.5.2.58: Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)
Section 20.5.2.59: Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)
Section 20.5.2.60: Receive FIFO Range Filter Configuration Register (RFRFCFR)
Section 20.5.2.61: Receive FIFO Range Filter Control Register (RFRFCTR)

Table 275. RFSR field descriptions

Field	Description
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

20.5.2.52 Receive FIFO Start Index Register (RFSIR)

Base + 0x0088 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 266. Receive FIFO Start Index Register (RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 276. RFSIR field descriptions

Field	Description
SIDX	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected receive FIFO. The controller uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

20.5.2.53 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FIFO_DEPTH								0	ENTRY_SIZE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 267. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e., the number of entries and the size of each entry.

Table 277. RFDSR field descriptions

Field	Description
FIFO_DEPTH	**FIFO Depth** — This field defines the depth of the selected receive FIFO, i.e., the number of entries.
ENTRY_SIZE	**Entry Size** — This field defines the size of the frame data sections for the selected receive FIFO in 2 byte entities.

20.5.2.54 Receive FIFO A Read Index Register (RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 268. Receive FIFO A Read Index Register (RFARIR)

This register provides the message buffer header index of the next available receive FIFO A entry that the application can read.

Table 278. RFARIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available receive FIFO message buffer that the application can read. The controller increments this index when the application writes to the FNEAIF flag in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*. The index wraps back to the first message buffer header index if the end of the FIFO was reached.

Note: If the receive FIFO not empty flag FNEAIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEAIF is set, the message buffer indicated by RDIDX contains valid data.

20.5.2.55 Receive FIFO B Read Index Register (RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 269. Receive FIFO B Read Index Register (RFBRIR)

This register provides the message buffer header index of the next available receive FIFO B entry that the application can read.

Table 279. RFBRIR field descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available receive FIFO entry that the application can read. The controller increments this index when the application writes to the FNEBIF flag in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*.The index wraps back to the first message buffer header index if the end of the FIFO was reached.

Note: If the receive FIFO not empty flag FNEBIF is not set, the RDIDX field points to an physical message buffer which content is not valid. Only when FNEBIF is set, the message buffer indicated by RDIDX contains valid data.

20.5.2.56 Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

Base + 0x0090 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIDAFVAL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 270. Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see *Section 20.6.9.5: Receive FIFO filtering*.

Table 280. RFMIDAFVR field descriptions

Field	Description
MIDAFVAL	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

20.5.2.57 Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

Base + 0x0092 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIDAFMSK															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 271. Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected receive FIFO. For details on message ID filtering see *Section 20.6.9.5: Receive FIFO filtering*.

Table 281. RFMIAFMR field descriptions

Field	Description
MIDAFMSK	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

20.5.2.58 Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

Base + 0x0094 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFVAL										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 272. Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see *Section 20.6.9.5: Receive FIFO filtering*.

Table 282. RFFIDRFVR field descriptions

Field	Description
FIDRFVAL	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

20.5.2.59 Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FIDRFMSK										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 273. Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected receive FIFO. For details on frame ID filtering see *Section 20.6.9.5: Receive FIFO filtering*.

Table 283. RFFIDRFMR field descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

20.5.2.60 Receive FIFO Range Filter Configuration Register (RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time
SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	SID										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 274. Receive FIFO Range Filter Configuration Register (RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected receive FIFO. For details on frame ID range filter see *Section 20.6.9.5: Receive FIFO filtering*.

Table 284. RFRFCFR field descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 Program lower interval boundary. 1 Program upper interval boundary.
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 Select frame ID range filter 0. 01 Select frame ID range filter 1. 10 Select frame ID range filter 2. 11 Select frame ID range filter 3.
SID	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

20.5.2.61 Receive FIFO Range Filter Control Register (RFRFCTR)

Base + 0x009A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 275. Receive FIFO Range Filter Control Register (RFRFCTR)

This register enables and disables each frame ID range filter and defines whether it is running as acceptance or rejection filter.

Table 285. RFRFCTR field descriptions

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter. 1 range filter 3 runs as rejection filter.
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter . 1 range filter 2 runs as rejection filter.
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter. 1 range filter 1 runs as rejection filter.
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter. 1 range filter 0 runs as rejection filter.
F3EN	**Range Filter 3 Enable** — This control bit enables and disables the frame ID range filter 3. 0 range filter 3 disabled. 1 range filter 3 enabled.
F2EN	**Range Filter 2 Enable** — This control bit enables and disables the frame ID range filter 2. 0 range filter 2 disabled. 1 range filter 2 enabled.
F1EN	**Range Filter 1 Enable** — This control bit enables and disables the frame ID range filter 1. 0 range filter 1 disabled. 1 range filter 1 enabled.
F0EN	**Range Filter 0 Enable** — This control bit enables and disables the frame ID range filter 0. 0 range filter 0 disabled. 1 range filter 0 enabled.

20.5.2.62 Last Dynamic Transmit Slot Channel A Register (LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 276. Last Dynamic Slot Channel A Register (LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 286. LDTXSLAR field descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

20.5.2.63 Last Dynamic Transmit Slot Channel B Register (LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 277. Last Dynamic Slot Channel B Register (LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 287. LDTXSLBR field descriptions

Field	Description
LASTDYNTXSLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

20.5.2.64 Protocol configuration registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in *Table 288*. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 288. Protocol configuration register fields

Name	Description[1]	Min	Max	Unit	PCR
coldstart_attempts	gColdstartAttempts			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			gdBit	4
dynamic_slot_idle_phase	gdDynamicSlotIdlePhase			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	gdStaticSlot			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	gdTSSTransmitter			gdBit	5
wakeup_symbol_rx_idle	gdWakeupSymbolRxIdle			gdBit	5
wakeup_symbol_rx_low	gdWakeupSymbolRxLow			gdBit	3
wakeup_symbol_rx_window	gdWakeupSymbolRxWindow			gdBit	4

Table 288. Protocol configuration register fields(Continued)

Name	Description(1)	Min	Max	Unit	PCR
wakeup_symbol_tx_idle	gdWakeupSymbolTxIdle			gdBit	8
wakeup_symbol_tx_low	gdWakeupSymbolTxLow			gdBit	5
noise_listen_timeout	(*gListenNoise* × *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	pMacroInitialOffset[A]			MT	6
macro_initial_offset_b	pMacroInitialOffset[B]			MT	16
macro_per_cycle	gMacroPerCycle			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	gMaxWithoutClockCorrectionFatal			cyclepairs	8
max_without_clock_correction_passive	gMaxWithoutClockCorrectionPassive			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	gNumberOfStaticSlots			static slot	2
offset_correction_start	gOffsetCorrectionStart			MT	11
payload_length_static	gPayloadLengthStatic			2-bytes	19
max_payload_length_dynamic	pPayloadLengthDynMax			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	pAllowHaltDueToClock			bool	26
allow_passive_to_active	pAllowPassiveToActive			cyclepairs	12
cluster_drift_damping	pClusterDriftDamping			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	pKeySlotId			number	18
key_slot_used_for_startup	pKeySlotUsedForStartup			bool	11
key_slot_used_for_sync	pKeySlotUsedForSync			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	gSyncNodeMax			number	30
micro_initial_offset_a	pMicroInitialOffset[A]			µT	20
micro_initial_offset_b	pMicroInitialOffset[B]			µT	20
micro_per_cycle	pMicroPerCycle			µT	22/23
micro_per_cycle_min	pMicroPerCycle - pdMaxDrift			µT	24/25
micro_per_cycle_max	pMicroPerCycle + pdMaxDrift			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	pOffsetCorrectionOut			µT	9
rate_correction_out	pRateCorrectionOut			µT	14

Table 288. Protocol configuration register fields(Continued)

Name	Description[1]	Min	Max	Unit	PCR
single_slot_enabled	pSingleSlotEnabled			bool	10
wakeup_channel	pWakeupChannel	see *Table 289*			10
wakeup_pattern	pWakeupPattern			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	pExternOffsetCorrection			µT	29
extern_rate_correction	pExternRateCorrection			µT	21

1. See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 289. Wakeup channel selection

wakeup_channel	Wakeup Channel
0	A
1	B

20.5.2.64.1 Protocol Configuration Register 0 (PCR0)

Base + 0x00A0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	action_point_offset								static_slot_length							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 278. Protocol Configuration Register 0 (PCR0)

20.5.2.64.2 Protocol Configuration Register 1 (PCR1)

Base + 0x00A2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	macro_after_first_static_slot													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 279. Protocol Configuration Register 1 (PCR1)

20.5.2.64.3 Protocol Configuration Register 2 (PCR2)

Base + 0x00A4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	minislot_after_action_point								number_of_static_slots							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 280. Protocol Configuration Register 2 (PCR2)

20.5.2.64.4 Protocol Configuration Register 3 (PCR3)

Base + 0x00A6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	wakeup_symbol_rx_low						minislot_action_point_offset[4:0]						coldstart_attempts			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 281. Protocol Configuration Register 3 (PCR3)

20.5.2.64.5 Protocol Configuration Register 4 (PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	cas_rx_low_max								wakeup_symbol_rx_window							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 282. Protocol Configuration Register 4 (PCR4)

20.5.2.64.6 Protocol Configuration Register 5 (PCR5)

Base + 0x00AA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	tss_transmitter				wakeup_symbol_tx_low						wakeup_symbol_rx_idle					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 283. Protocol Configuration Register 5 (PCR5)

20.5.2.64.7 Protocol Configuration Register 6 (PCR6)

Base + 0x00AC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point								macro_initial_offset_a						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 284. Protocol Configuration Register 6 (PCR6)

20.5.2.64.8 Protocol Configuration Register 7 (PCR7)

Base + 0x00AE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_b								micro_per_macro_nom_half							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 285. Protocol Configuration Register 7 (PCR7)

20.5.2.64.9 Protocol Configuration Register 8 (PCR8)

Base + 0x00B0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	max_without_clock_correction_fatal				max_without_clock_correction_passive				wakeup_symbol_tx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 286. Protocol Configuration Register 8 (PCR8)

20.5.2.64.10 Protocol Configuration Register 9 (PCR9)

Base + 0x00B2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	mini slot_ exists	sym bol_ win dow_ exists	offset_correction_out													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 287. Protocol Configuration Register 9 (PCR9)

20.5.2.64.11 Protocol Configuration Register 10 (PCR10)

Base + 0x00B4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	single_slot _enabled	wake up_ chan nel	macro_per_cycle													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 288. Protocol Configuration Register 10 (PCR10)

20.5.2.64.12 Protocol Configuration Register 11 (PCR11)

Base + 0x00B6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	key_slot_used_for_startup				key_slot_used_for_sync				offset_correction_start							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 289. Protocol Configuration Register 11 (PCR11)

20.5.2.64.13 Protocol Configuration Register 12 (PCR12)

Base + 0x00B8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	allow_passive_to_active								key_slot_header_crc							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 290. Protocol Configuration Register 12 (PCR12)

20.5.2.64.14 Protocol Configuration Register 13 (PCR13)

Base + 0x00BA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	first_minislot_action_point_offset								static_slot_after_action_point							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 291. Protocol Configuration Register 13 (PCR13)

20.5.2.64.15 Protocol Configuration Register 14 (PCR14)

Base + 0x00BC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	rate_correction_out												listen_timeout[20:16]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 292. Protocol Configuration Register 14 (PCR14)

20.5.2.64.16 Protocol Configuration Register 15 (PCR15)

Base + 0x00BE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 293. Protocol Configuration Register 15 (PCR15)

20.5.2.64.17 Protocol Configuration Register 16 (PCR16)

Base + 0x00C0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b								noise_listen_timeout[24:16]							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 294. Protocol Configuration Register 16 (PCR16)

20.5.2.64.18 Protocol Configuration Register 17 (PCR17)

Base + 0x00C2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 295. Protocol Configuration Register 17 (PCR17)

20.5.2.64.19 Protocol Configuration Register 18 (PCR18)

Base + 0x00C4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_pattern								key_slot_id							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 296. Protocol Configuration Register 18 (PCR18)

20.5.2.64.20 Protocol Configuration Register 19 (PCR19)

Base + 0x00C6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_a								payload_length_static							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 297. Protocol Configuration Register 19 (PCR19)

20.5.2.64.21 Protocol Configuration Register 20 (PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	micro_initial_offset_b								micro_initial_offset_a							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 298. Protocol Configuration Register 20 (PCR20)

20.5.2.64.22 Protocol Configuration Register 21 (PCR21)

Base + 0x00CA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	extern_rate_correction				latest_tx											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 299. Protocol Configuration Register 21 (PCR21)

20.5.2.64.23 Protocol Configuration Register 22 (PCR22)

Base + 0x00CC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	R*	comp_accepted_startup_range_a											micro_per_cycle[19:16			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 300. Protocol Configuration Register 22 (PCR22)

20.5.2.64.24 Protocol Configuration Register 23 (PCR23)

Base + 0x00CE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 301. Protocol Configuration Register 23 (PCR23)

20.5.2.64.25 Protocol Configuration Register 24 (PCR24)

Base + 0x00D0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W	cluster_drift_damping				max_payload_length_dynamic								micro_per_cycle_min [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 302. Protocol Configuration Register 24 (PCR24)

20.5.2.64.26 Protocol Configuration Register 25 (PCR25)

Base + 0x00D2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 303. Protocol Configuration Register 25 (PCR25)

20.5.2.64.27 Protocol Configuration Register 26 (PCR26)

Base + 0x00D4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_halt_due_to_clock				comp_accepted_startup_range_b								micro_per_cycle_max[19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 304. Protocol Configuration Register 26 (PCR26)

20.5.2.64.28 Protocol Configuration Register 27 (PCR27)

Base + 0x00D6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_max[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 305. Protocol Configuration Register 27 (PCR27)

20.5.2.64.29 Protocol Configuration Register 28 (PCR28)

Base + 0x00D8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	dynamic_slot_idle_phase				macro_after_offset_correction											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 306. Protocol Configuration Register 28 (PCR28)

20.5.2.64.30 Protocol Configuration Register 29 (PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_offset_correction				minislots_max											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 307. Protocol Configuration Register 29 (PCR29)

20.5.2.64.31 Protocol Configuration Register 30 (PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 308. Protocol Configuration Register 30 (PCR30)

20.5.2.65 Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

Base + 0x0100 (MBCCSR0)
Base + 0x0108 (MBCCSR1)
...
Base + 0x01F8 (MBCCSR31)

Write: MCM, MBT, MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MCM	MBT	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 309. Message Buffer Configuration, Control, Status Registers (MBCCSR*n*)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in *Section 20.6.6: Individual message buffer functional description*.

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 290. MBCCSR*n* field descriptions

Field	Description
Message buffer configuration	
MCM	**Message Buffer Commit Mode** — This bit configures the commit mode of a double buffered message buffer. 0 Streaming commit mode 1 Immediate commit mode
MBT	**Message Buffer Type** — This bit configures the buffering type of a transmit message buffer. 0 Single buffered message buffer 1 Double buffered message buffer
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer
Message buffer control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission. 1 Message buffer data ready for transmission.
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value EDS status bit is 0. 0 No effect. 1 Message buffer enable or disable is triggered.
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect. 1 Message buffer lock or unlock is triggered.
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled. 1 Interrupt request generation enabled.

Table 290. MBCCSR*n* field descriptions(Continued)

Field	Description
Message buffer status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated 1 Frame Header and Message buffer data field updated
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time 1 receive message buffer contains valid frame data / message will be transferred again
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event 1 Slot status field updated or transmit message buffer just enabled.

20.5.2.66 Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

Base + 0x0102 (MBCCFR0)
Base + 0x010A (MBCCFR1)
...
Base + 0x011FA (MBCCFR31)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MTM	CHA	CHB	CCFE	CCFMSK								CCFVAL			
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Figure 310. Message Buffer Cycle Counter Filter Registers (MBCCFR*n*)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to *Section 20.6.7.1: Message buffer cycle counter filtering*.

Table 291. MBCCFR*n* field descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to *Table 292*.
CCFE	**Cycle Counter Filtering Enable** — This control bit enables and disables the cycle counter filtering. 0 Cycle counter filtering disabled. 1 Cycle counter filtering enabled.
CCFMSK	**Cycle Counter Filtering Mask** — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 292. Channel assignment description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	Reserved (function not available)	store first valid frame received on either channel A or channel B	Reserved (function not available)
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

Note: If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

20.5.2.67 Message Buffer Frame ID Registers (MBFIDR*n*)

Base + 0x0104 (MBFIDR0)
Base + 0x010C (MBFIDR1)
...
Base + 0x01FC (MBFIDR31)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-

Figure 311. Message Buffer Frame ID Registers (MBFIDR*n*)

Table 293. MBFIDRn field descriptions

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. – *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. – *Transmit Message Buffer:* This field determines the slot in which the message in this message buffer should be transmitted.

20.5.2.68 Message Buffer Index Registers (MBIDXR*n*)

Base + 0x0106 (MBIDXR0)
Base + 0x010E (MBIDXR1)
...
Base + 0x01FE (MBIDXR31)

Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	MBIDX					
W																
Reset	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-

Figure 312. Message Buffer Index Registers (MBIDXR*n*)

Table 294. MBIDXR*n* field descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The controller updates this register after frame reception or transmission.

20.6 Functional description

This section provides a detailed description of the functionality implemented in the controller.

20.6.1 Message buffer concept

The controller uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in *Section 20.6.3: Message buffer types*. The physical message buffer is located in the FlexRay memory and is described in *Section 20.6.2: Physical message buffer*.

20.6.2 Physical message buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical*

message buffers. The physical message buffers are located in the FlexRay memory.The structure of a physical message buffer is shown in *Figure 313*.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header*, the *data field offset*, and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

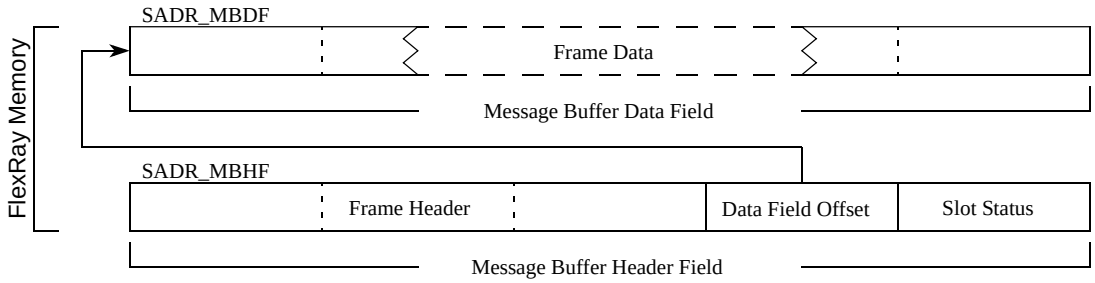


Figure 313. Physical message buffer structure

20.6.2.1 Message buffer header field

The message buffer header field is a contiguous region in the FlexRay memory and occupies ten bytes. It contains the frame header, the data field offset, and the slot status. Its structure is shown in *Figure 313*. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

20.6.2.1.1 Frame header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the usage and the content of the frame header is provided in *Section 20.6.5.2.1: Frame header description*.

20.6.2.1.2 Data field offset

The data field offset follows the frame header in the message buffer data field and occupies two bytes. It contains the offset of the corresponding message buffer data field with respect to the controller FlexRay memory base address as provided by SYS_MEM_BASE_ADDR field in the *Section 20.5.2.5: System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*. The data field offset determines the start address *SADR_MBDF* of the corresponding message buffer data field in the FlexRay memory according to *Equation 18*.

Equation 18 SADR_MBDF = [Data Field Offset] + SYS_MEM_BASE_ADDR

20.6.2.1.3 Slot status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications*

System Protocol Specification, Version 2.1 Rev A. A detailed description of the content and usage of the slot status is provided in *Section 20.6.5.2.3: Slot status description*.

20.6.2.2 Message buffer data field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in *Section 20.6.3: Message buffer types*.

20.6.3 Message buffer types

The controller provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

20.6.3.1 Individual message buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The controller supports three types of individual message buffers, which are described in *Section 20.6.6: Individual message buffer functional description*.

Each individual message buffer consists of two parts, the physical message buffer, which is located in the FlexRay memory, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in *Figure 314*.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers MBCCSRn, MBCCFRn, MBFIDRn, and MBIDXRn.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to *Equation 19*.

Equation 19 SADR_MBHF = (MBIDXRn[MBIDX] × 10) + SYS_MEM_BASE_ADDR

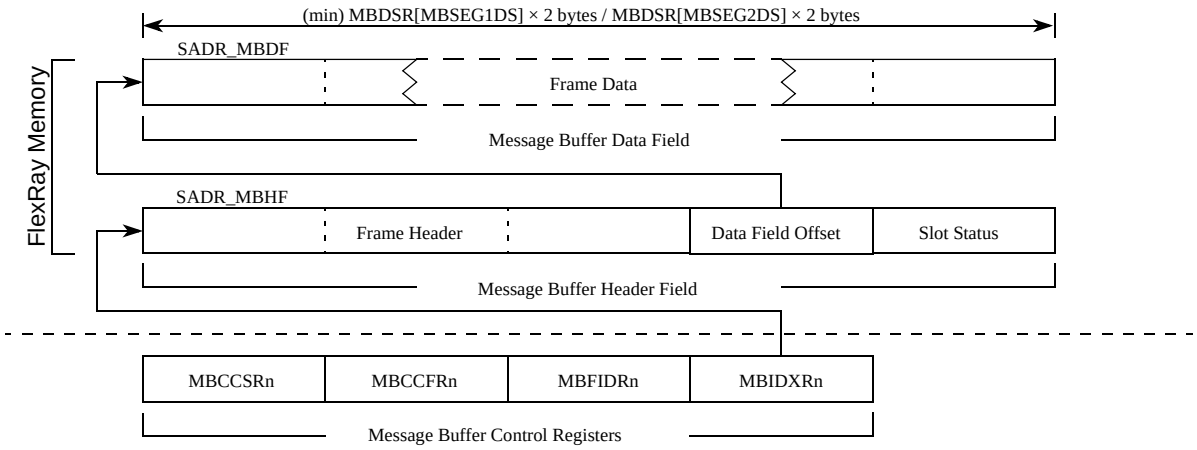


Figure 314. Individual message buffer structure

20.6.3.1.1 Individual message buffer segments

The set of the individual message buffers can be split up into two message buffer segments using the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*. All individual message buffers with a message buffer number $n \leq$ MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number $n >$ MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- All physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length.
- The minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 × MBDSR[MBSEG1DS] bytes.
- The minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 × MBDSR[MBSEG2DS] bytes.

20.6.3.2 Receive shadow buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The controller provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the FlexRay memory and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in *Figure 315*. The four internal shadow buffer control registers can be accessed by the *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*.

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*. The start address SADR_MBHF of the related message buffer header field in the FlexRay memory is determined according to *Equation 20*.

Equation 20 SADR_MBHF = (RSBIR[RSBIDX] × 10) + SYS_MEM_BASE_ADDR

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*.

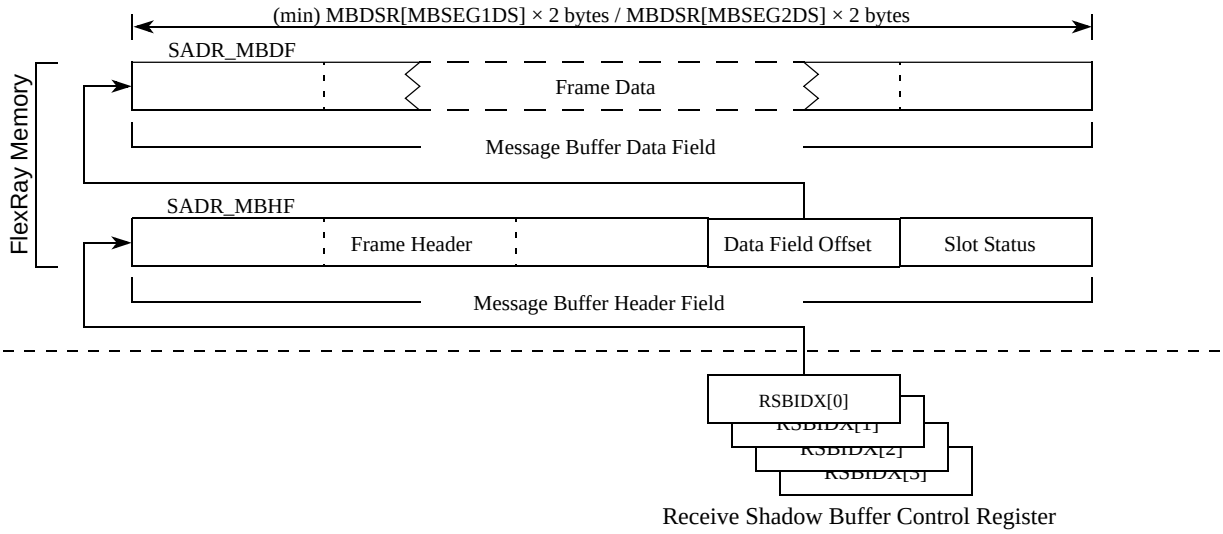


Figure 315. Receive shadow buffer structure

20.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The controller provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the FlexRay memory and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in *Figure 316*.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the start index field SIDX in the *Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)*, the FIFO depth field FIFO_DEPTH in the *Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)*, and the read index field RDIDX *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* / *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)*. The start address SADR_MBHF_1 of the first message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to *Equation 21*.

Equation 21 SADR_MBHF[1] = (RFSIR[SIDX] × 10) + SYS_MEM_BASE_ADDR

The start address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the FlexRay memory is determined according to *Equation 22*.

Equation 22 SADR_MBHF[n] = ((RFSIR[SIDX] + RFDSR[FIFO_DEPTH]) × 10) + SYS_MEM_BASE_ADDR

Note: All message buffer header fields assigned to a receive FIFO must be a contiguous region.

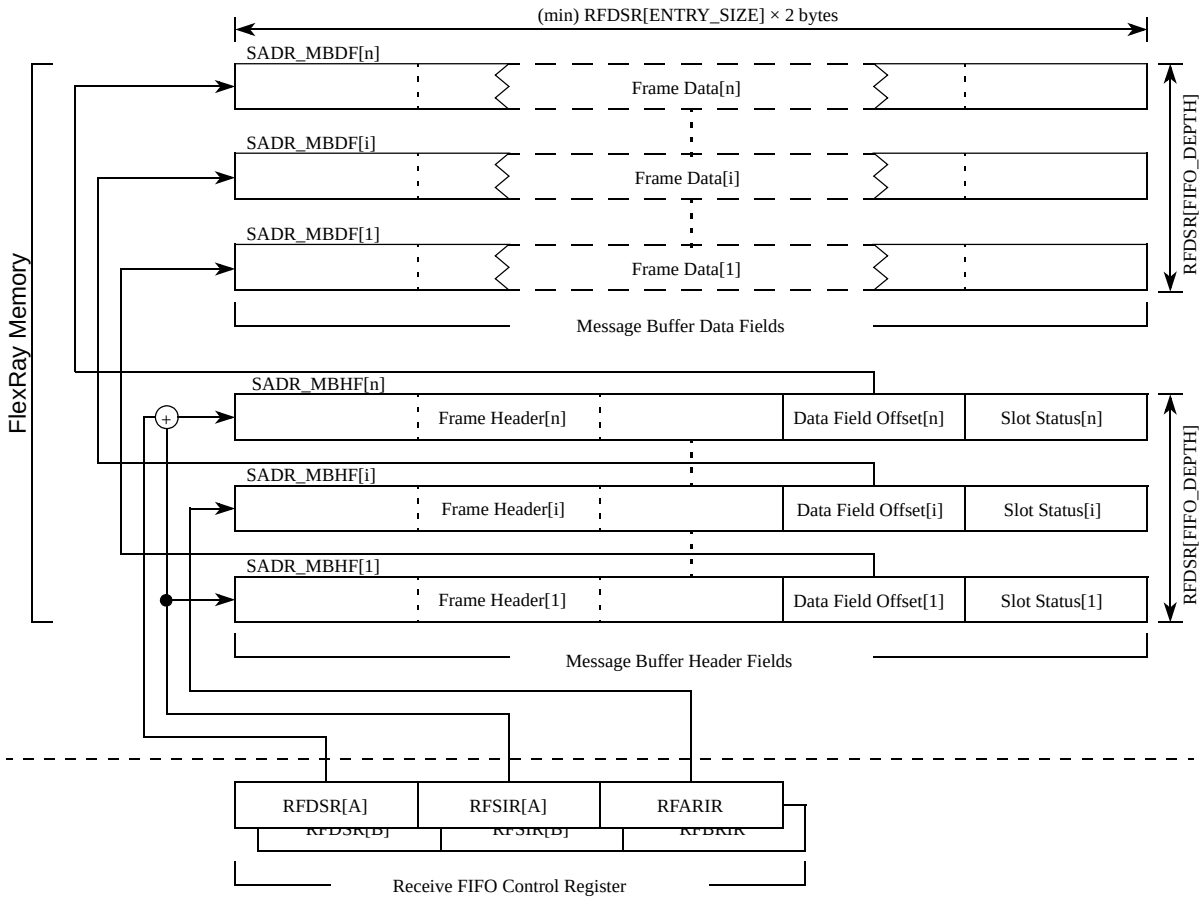


Figure 316. Receive FIFO structure

20.6.3.4 Message buffer configuration and control data

This section describes the configuration and control data for each message buffer type.

20.6.3.4.1 Individual message buffer configuration data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

20.6.3.4.1.1 Common configuration data

The set of common configuration data for individual message buffers is located in the following registers.

- *Section 20.5.2.7: Message Buffer Data Size Register (MBDSR)*
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see *Section 20.6.3.1.1: Individual message buffer segments*.

20.6.3.4.1.2 Specific configuration data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*
  The MCM, MBT, MTD bits configure the message buffer type.
- *Section 20.5.2.66: Message Buffer Cycle Counter Filter Registers (MBCCFRn)*
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*
  For a transmit message buffer, the FID field determines the slot in which the message in this message buffer will be transmitted.
- *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

20.6.3.5 Individual message buffer control data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

20.6.3.6 Receive shadow buffer configuration data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*. For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full controller control.

20.6.3.7 Receive FIFO control and configuration data

This section describes the configuration and control data for the two receive FIFOs.

20.6.3.7.1 Receive FIFO configuration data

The controller provides two completely independent receive FIFOs, one per channel. Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- *Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)*
- *Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)*
- *Section 20.5.2.56: Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)*
- *Section 20.5.2.57: Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)*
- *Section 20.5.2.58: Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)*
- *Section 20.5.2.59: Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)*
- *Section 20.5.2.60: Receive FIFO Range Filter Configuration Register (RFRFCFR)*

20.6.3.7.2 Receive FIFO control data

The application can access the receive FIFO at any time using the values provided in the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* and *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)*. To update the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)*, the application must write 1 to the FIFO A Not Empty Interrupt Flag FNEAIF in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*. To update the *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)* the application must write 1 to the FIFO B Not Empty Interrupt Flag FNEBIF in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*. As long as the FIFO is not empty, each update increments the related read index. If the read index has reached the last FIFO entry, it wraps back to the FIFO start index.

20.6.4 FlexRay memory layout

The controller supports a wide range of possible layouts for the FlexRay memory. *Figure 317* shows an example layout. The following set of rules applies to the layout of the FlexRay memory:

- The FlexRay memory is a contiguous region.
- The FlexRay memory size is maximum 64 KB.
- The FlexRay memory starts at a 16 byte boundary.

The FlexRay memory contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*. The areas are described in this section.

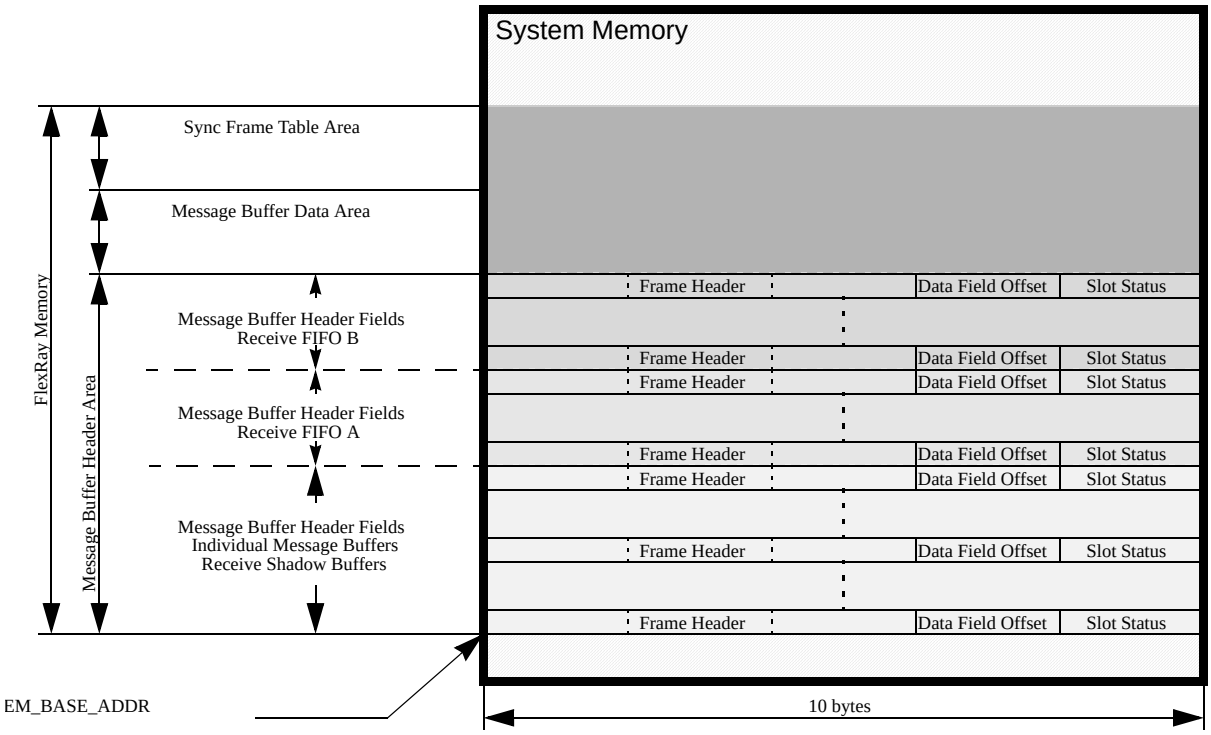


Figure 317. Example of FlexRay memory layout

20.6.4.1 Message buffer header area

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill *Equation 23*.

Equation 23 SADR_MBHF = (i × 10) + SYS_MEM_BASE_ADDR; (0 <= i < 64)

2. The start address SADR_MBHF of each message buffer header field for the *receive FIFO* must fulfill *Equation 24*.

Equation 24 SADR_MBHF = (i × 10) + SYS_MEM_BASE_ADDR; (0 <= i < 1024)

3. The message buffer header fields for a receive FIFO have to be a contiguous area.

20.6.4.2 Message buffer data area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

20.6.4.3 Sync frame table area

The sync frame table area provides a copy of the internal sync frame tables for application access. Refer to *Section 20.6.12: Sync frame ID and sync frame deviation tables* for the description of the sync frame table area.

20.6.5 Physical message buffer description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

20.6.5.1 Message buffer protection and data consistency

The physical message buffers are located in the FlexRay memory. The controller provides no means to protect the FlexRay memory from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

20.6.5.2 Message buffer header field description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in *Section 20.6.2.1: Message buffer header field*. Each message buffer header field consists of three sections: the frame header section, the data field offset, and the slot status section. For a detailed description of the Data Field Offset, see *Section 20.6.2.1.2: Data field offset*.

20.6.5.2.1 Frame header description

20.6.5.2.1.1 Frame header content

The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in *Figure 318*. A detailed description is given in *Table 296*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2	0	0	CYCCNT						0	PLDLEN						
0x4	0	0	0	0	0	HDCRC										

Figure 318. Frame header structure (Receive message buffer and receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in *Figure 319*. A detailed description is given in *Table 297*. The checks that will be performed are described in *Section 20.6.5.2.1.3: Frame header checks*.

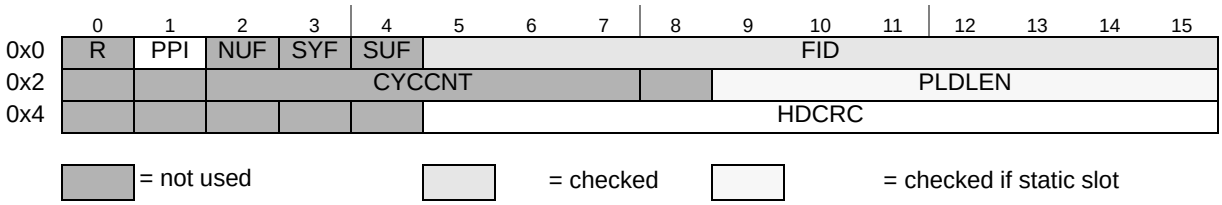


Figure 319. Frame header structure (Transmit message buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in *Figure 320*.

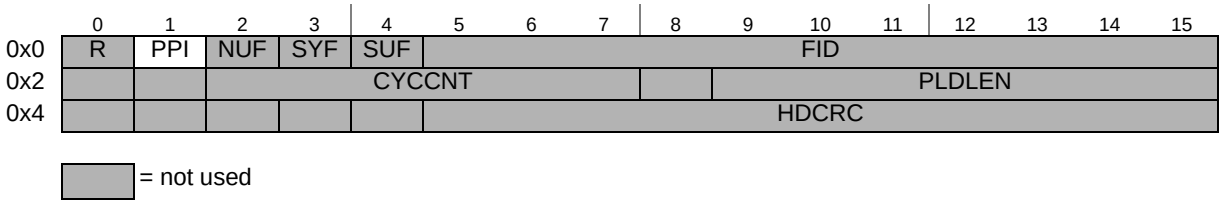


Figure 320. Frame header structure (Transmit message buffer for key slot)

20.6.5.2.1.2 Frame header access

The frame header is located in the FlexRay memory. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 295*. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 295. Frame header write access constraints (Transmit message buffer)

Field	Single Buffered		Double Buffered			
	Static Segment	Dynamic Segment	Static Segment		Dynamic Segment	
			Commit Side	Transmit Side	Commit Side	Transmit Side
FID	*POC:config* or MB_DIS					
PPI, PLDLEN, HDCRC	*POC:config* or MB_DIS or					
		MB_LCK			MB_LCK	

20.6.5.2.1.3 Frame header checks

As shown in *Figure 319* and *Figure 320* not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*. If the controller detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*. The value of the FID field will be ignored and replaced by the value provided in the *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*.

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the *Section 20.5.2.64.20: Protocol Configuration Register 19 (PCR19)*. If this is not fulfilled, the static payload length error flag SPL_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the *Section 20.5.2.64.25: Protocol Configuration Register 24 (PCR24)*. If this is not fulfilled, the dynamic payload length error flag DPL_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 296. Frame header field descriptions (Receive message buffer and receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer.
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 297. Frame header field descriptions (Transmit message buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
FID	**Frame ID** — This field is checked as described in *Section 20.6.5.2.1.3: Frame header checks*.
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.
PLDLEN	**Payload Length** — This field is checked and used as described in *Section 20.6.5.2.1.3: Frame header checks*.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

20.6.5.2.2 Data field offset description

20.6.5.2.2.1 Data field offset content

For a detailed description of the Data Field Offset, see *Section 20.6.2.1.2: Data field offset*.

20.6.5.2.2.2 Data field offset access

The application shall program the Data Field Offset when configuring the message buffers either in the *POC:config* state or when the message buffer is disabled.

20.6.5.2.3 Slot status description

The slot status is a read-only structure for the application and a write-only structure for the controller. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

20.6.5.2.3.1 Receive message buffer and receive FIFO slot status description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by *Table 298*.

Table 298. Receive message buffer slot status content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels MBCCSRn[CHA]=1 and MBCCSRn[CHB]=1	see *Figure 321*
Individual Receive Message Buffer assigned to channel A MBCCSRn[CHA]=1 and MBCCSRn[CHB[=0	see *Figure 322*
Individual Receive Message Buffer assigned to channel B MBCCSRn[CHA]=0 and MBCCSRn[CHB]=1	see *Figure 323*
Receive FIFO Channel A Message Buffer	see *Figure 322*
Receive FIFO Channel B Message Buffer	see *Figure 323*

The meaning of the bits in the slot status structure is explained in *Table 299*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 321. Receive message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 322. Receive message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 323. Receive message buffer slot status structure (ChB)

Table 299. Receive message buffer slot status field descriptions

Field	Description
Common Message Buffer Status Bits	
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.

Table 299. Receive message buffer slot status field descriptions(Continued)

Field	Description
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all. 0 first valid frame received on channel B.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!Vali. dFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.

20.6.5.2.3.2 Transmit message buffer slot status description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by *Table 300*.

Table 300. Transmit message buffer slot status content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels MBCCSRn[CHA]=1 and MBCCSRn[CHB]=1	see *Figure 324*
Individual Transmit Message Buffer assigned to channel A MBCCSRn[CHA]=1 and MBCCSRn[CHB]=0	see *Figure 325*
Individual Transmit Message Buffer assigned to channel B MBCCSRn[CHA]=0 and MBCCSRn[CHB]=1	see *Figure 326*

The meaning of the bits in the slot status structure is described in *Table 299*.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 324. Transmit message buffer slot status structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 325. Transmit message buffer slot status structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 326. Transmit message buffer slot status structure (ChB)

Table 301. Transmit message buffer slot status structure field descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.

Table 301. Transmit message buffer slot status structure field descriptions(Continued)

Field	Description
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0. 1 *vSS!ValidFrame* = 1.
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0. 1 *vRF!Header!SyFIndicator* = 1.
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0. 1 *vRF!Header!NFIndicator* = 1.
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0. 1 *vRF!Header!SuFIndicator* = 1.
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0. 1 *vSS!SyntaxError* = 1.
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0. 1 *vSS!ContentError* = 1.
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0. 1 *vSS!BViolation* = 1.
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0. 1 *vSS!TxConflict* = 1.

20.6.5.3 Message buffer data field description

The message buffer data field stores the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in *Table 302*. The structure of the message buffer data field is given in *Figure 327*.

Table 302. Message buffer data field minimum length

physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	MBDSR[MBSEG2DS]
Receive FIFO for channel A	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 0)
Receive FIFO for channel B	RFDSR[ENTRY_SIZE] (RFSR[SEL] = 1)

Note: The controller will not access any locations outside the message buffer data field boundaries given by Table 302.

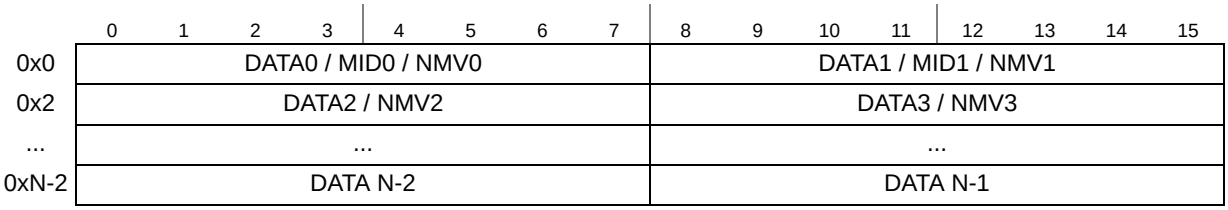


Figure 327. Message buffer data field structure

The message buffer data field is located in the FlexRay memory; thus, the controller has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

20.6.5.3.1 Message buffer data field read access

For transmit message buffers, the controller will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*. While the message buffer is locked, the controller will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* or the *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)* when the related receive FIFO non-empty interrupt flag FNEAIF or FNEBIF is set in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)*. While the non-empty interrupt flag is set, the controller will not update the Message Buffer Data Field related to message buffer indicated by *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* or the *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)*.

20.6.5.3.2 Message buffer data field write access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in *Table 303*.

Table 303. Frame data write access constraints

Field	Single buffered	Double buffered	
		Commit side	Transmit side
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS or MB_LCK	*POC:config* or MB_DIS

Table 304. Frame data field descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. **Note:** The MID and NMV bytes replace the corresponding DATA bytes.

20.6.6 Individual message buffer functional description

The controller provides three basic types of individual message buffers:

1. Single Transmit Message Buffers
2. Double Transmit Message Buffers
3. Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in *Section 20.6.3.4.1: Individual message buffer configuration data*.

20.6.6.1 Individual message buffer configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the FlexRay memory. The second step is the programming of the message buffer configuration registers, which is described in this section.

20.6.6.1.1 Common configuration data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*.

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the *Section 20.5.2.7: Message Buffer Data Size Register (MBDSR)*.

Depending on the current receive functionality of the controller, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the *Section 20.5.2.50: Receive Shadow Buffer Index Register (RSBIR)*.

20.6.6.1.2 Specific configuration data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled, i.e., MBCCSR*n*[EDS] = 0

The individual message buffer type is defined by the MTD and MBT bits in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)* as given in *Table 305*.

Table 305. Individual message buffer types

MBCCSRn[MTD]	MBCCSRn[MBT]	Individual Message Buffer Description
0	0	Receive Message Buffer
0	1	Reserved
1	0	Single Transmit Message Buffer
1	1	Double Transmit Message Buffer

The message buffer specific configuration data are

1. MCM, MBT, MTD bits in *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*
2. all fields and bits in *Section 20.5.2.66: Message Buffer Cycle Counter Filter Registers (MBCCFRn)*
3. all fields and bits in *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*
4. all fields and bits in *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions *Section 20.6.6.2: Single transmit message buffers*, *Section 20.6.6.3: Receive message buffers* and *Section 20.6.6.4: Double transmit message buffer*.

20.6.6.2 Single transmit message buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A single transmit message buffer is used by the application to provide message data to the controller that will be transmitted over the FlexRay Bus. The controller uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a single transmit message buffer by the following settings:

- MBCCSRn[MBT] = 0 (single buffered message buffer)
- MBCCSRn[MTD] = 1 (transmit message buffer)

20.6.6.2.1 Access regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the access to the data, control, and status bits of a message buffer. The access regions for single transmit message buffers are shown in *Figure 328*. A description of the regions is given in *Table 306*. If an region is active as indicated in *Table 307*, the access scheme given for that region applies to the message buffer.

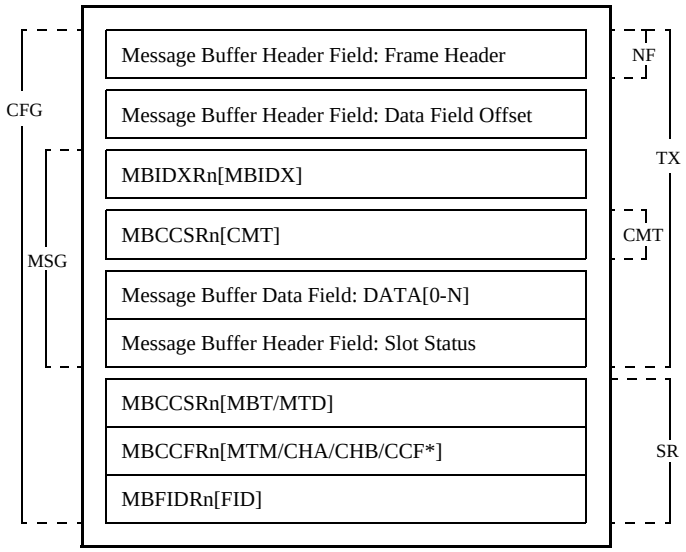


Figure 328. Single transmit message buffer access regions

Table 306. Single transmit message buffer access regions description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Data and Slot Status Access
NF	—	read-only	Message Header Access for Null Frame Transmission
TX	—	read/write	Message Transmission and Slot Status Update
CM	—	read-only	Message Buffer Validation
SR	—	read-only	Message Buffer Search

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT], and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSnR[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller clear access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in *Figure 329*. A description of the states is given in *Table 307*, which also provides the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

20.6.6.2.2 Message buffer states

This section describes the transmit message buffer states and provides a state diagram.

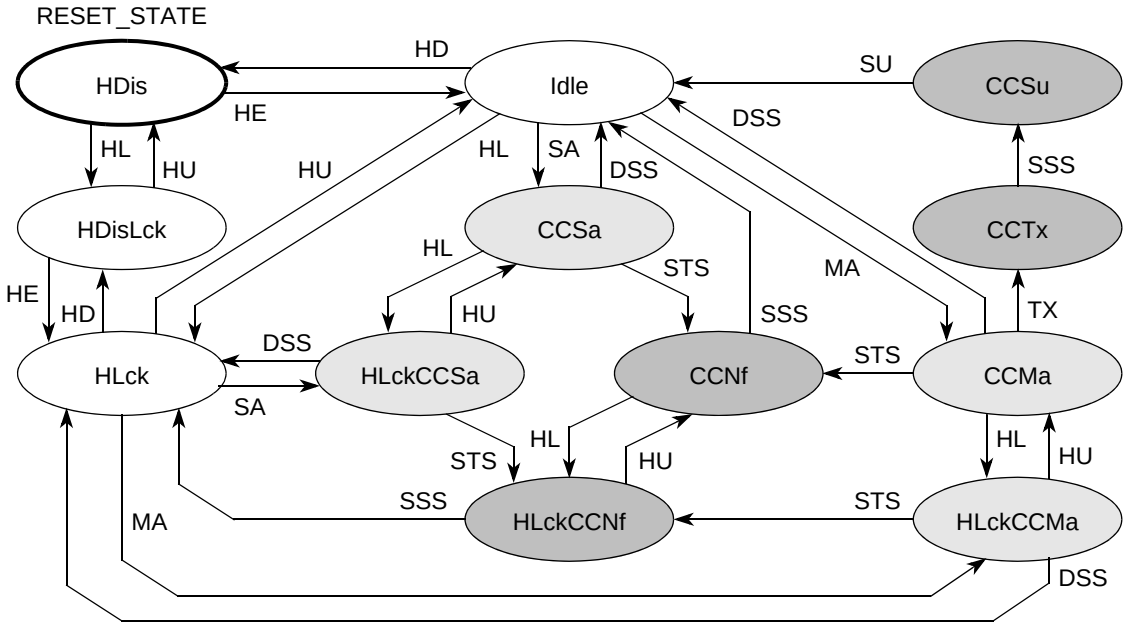


Figure 329. Single transmit message buffer states

Table 307. Single transmit message buffer state description

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	CM, SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked — Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	—	—	Slot Assigned — Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	—	Locked and Slot Assigned — Applications access to data, control, and status. Message buffer assigned to next static slot
CCNf	1	0	—	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission — Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	—	CM	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	—	Locked and Message Available — Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.

Table 307. Single transmit message buffer state description(Continued)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCTx	1	0	—	TX	Message Transmission — Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	TX	Status Update — Message buffer status update. Update of status flags, the slot status field, and the header index.

20.6.6.2.3 Message buffer transitions

20.6.6.2.3.1 Application transitions

The application transitions can be triggered by the application using the commands described in *Table 308*. The application issues the commands by writing to the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

20.6.6.2.4 Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

20.6.6.2.5 Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is set.

Table 308. Single transmit message buffer application transitions

Transition	Command	Condition	Description
HE	MBCCSRn[EDT]:= 1	MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSRn[LCKT]:= 1	MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

20.6.6.2.5.1 Module transitions

The module transitions that can be triggered by the controller are described in *Table 309*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 309. Single transmit message buffer module transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned — Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSRn[CMT] = 1	Transmission Slot Start — Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated — Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start — Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start — Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of static slot or dynamic slot or symbol window or NIT.

20.6.6.2.5.2 Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 310*, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of *Table 310*.

Table 310. Single transmit message buffer transition priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

20.6.6.2.6 Transmit message setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. The physical access to the message buffer data field is described in *Section 20.6.3.1: Individual message buffers*.

As indicated by *Table 307*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 308*. The state change is indicated through the MBCCSRn[EDS] and MBCCSRn[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the MBCCSRn[DVAL] flag is negated.

20.6.6.2.7 Message transmission

As a result of the message buffer search described in *Section 20.6.7: Individual message buffer search* the controller triggers the message available transition MA for as many as two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The controller transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid, i.e., MBCCSR*n*[CMT] = 1

In this case, the controller triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in *Figure 330*. In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid, i.e., MBCCSR*n*[CMT] = 1.

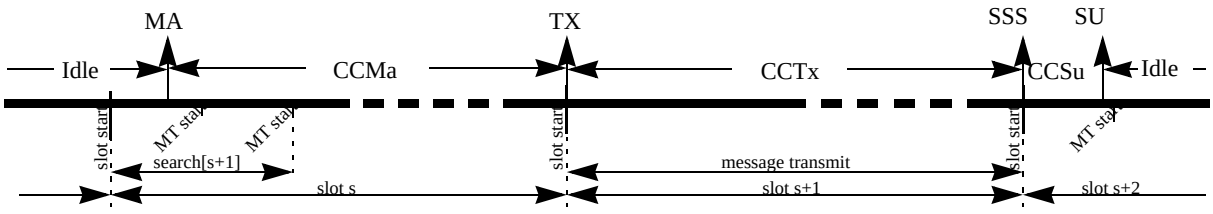


Figure 330. Message transmission timing

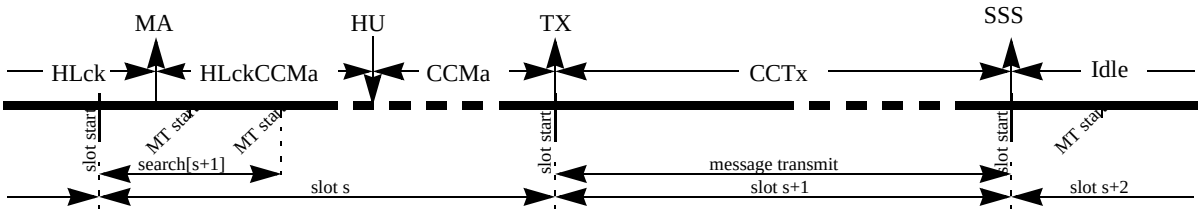


Figure 331. Message transmission from HLck state with unlock

The amount of message data read from the FlexRay memory and transferred to the FlexRay bus is determined by the following three items:

1. the message buffer segment that the message buffer is assigned to, as defined by MBSSUTR (see *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*")
2. the message buffer data field size, as defined by the related field of MBDSR (see *Section 20.5.2.7: Message Buffer Data Size Register (MBDSR)*")
3. the value of the PLDLEN field in the message buffer header field, as described in *Section 20.6.5.2.1: Frame header description*.

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 × MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN <= MBSEG1DS, the controller reads and transfers 2 × PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

20.6.6.2.8 Null frame transmission

A static slot with slot number S is assigned to the controller for channel A, if at least one transmit message buffer is configured with the MBFIDR*n*[FID] set to S and MBCCFRn[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the controller for channel A, and all transmit message buffers with MBFIDR*n*[FID] = s and MBCCFR*n*[CHA] = 1 are either not committed, i.e., MBCCSR*n*[CMT] = 0, or locked by the application, i.e., MBCCSR*n*[LCKS] = 1, or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in *Section 20.6.7: Individual message buffer search* the controller triggers the slot assigned transition SA for as many as two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in *Figure 332*.

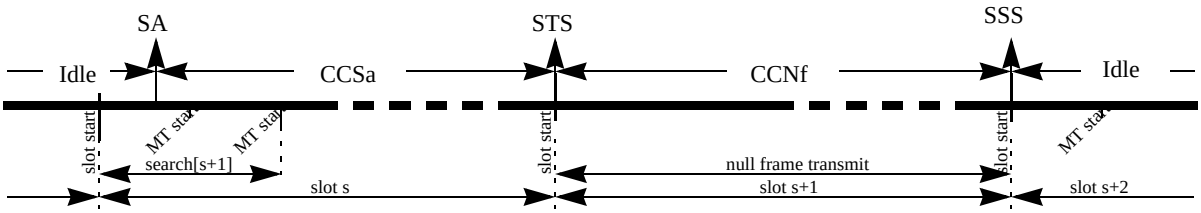


Figure 332. Null frame transmission from idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in *Figure 333*.

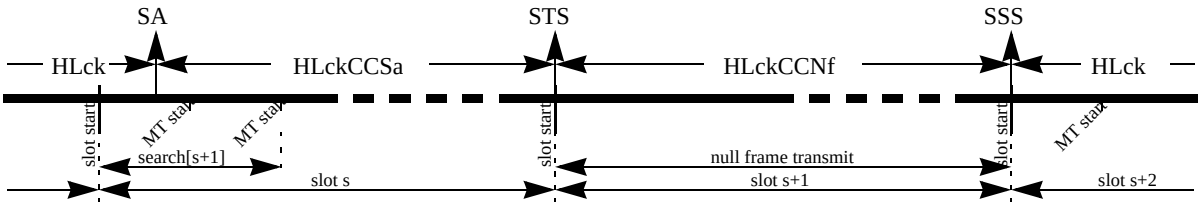


Figure 333. Null frame transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 334*.

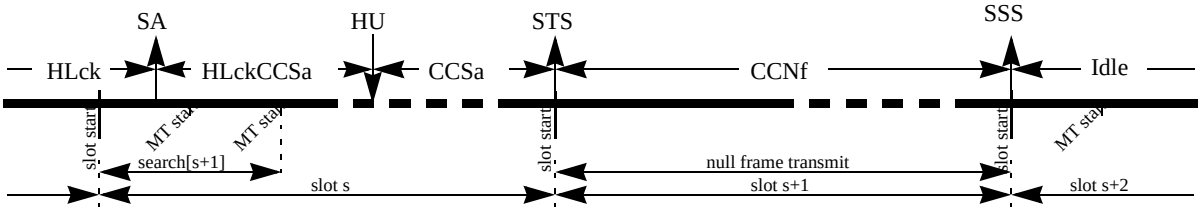


Figure 334. Null frame transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in *Figure 335*.

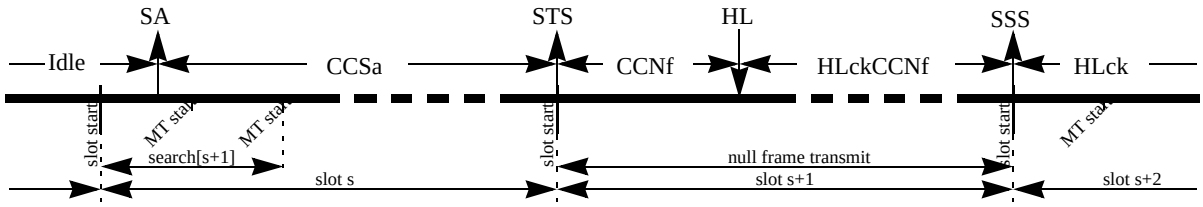


Figure 335. Null frame transmission from idle state with locking

20.6.6.2.9 Message buffer status update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

20.6.6.2.9.1 Message buffer status update after complete message transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the controller updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the controller sets the message buffer interrupt flag MBCCSn[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag MBCCFRn[MTM], the controller changes the commit flag MBCCSRn[CMT] and the valid flag MBCCSRn[DVAL]. If the MBCCFRn[MTM] flag is negated, the message buffer is in the *event transmission mode.* In this case, each committed message is transmitted only once. The commit flag MBCCSRn[CMT] is cleared with the SU transition. If the MBCCFRn[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The controller will not clear the MBCCSRn[CMT] flag at the end of transmission and will set the valid flag MBCCSRn[DVAL] to indicate that the message will be transmitted again.

20.6.6.2.9.2 Message buffer status update after incomplete message transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)*.

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

20.6.6.2.9.3 Message buffer status update after null frame transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

20.6.6.3 Receive message buffers

The section provides a detailed description of the functionality of the receive message buffers.

A receive message buffer receives a message from the FlexRay Bus based on individual filter criteria. The controller uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- MBCCSRn[MBT] = 0 (single buffered message buffer)
- MBCCSRn[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented that controls the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are shown in *Figure 336*. A description of the regions is given in *Table 311*. If an region is active as indicated in *Table 312*, the access scheme given for that region applies to the message buffer.

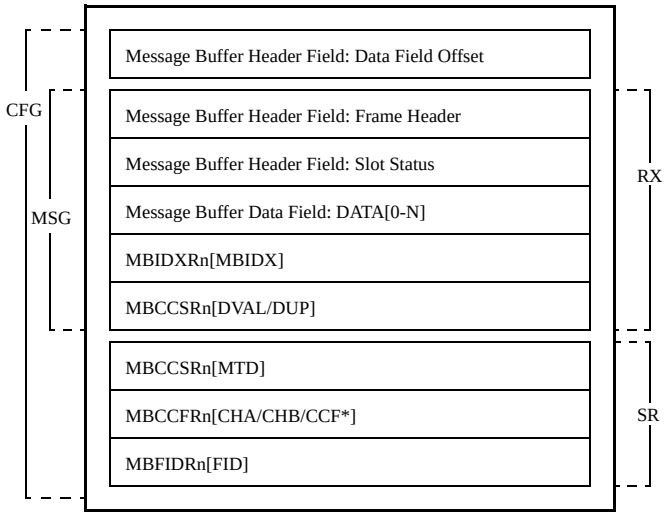


Figure 336. Receive message buffer access regions

Table 311. Receive message buffer access region description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	—	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	—	Message Data, Header, and Status Access
RX	—	write-only	Message Reception and Status Update
SR	—	read-only	Message Buffer Search Data

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT] and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSRn[MBIF] is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in *Figure 337*. A description of the message buffer states is given in *Table 307*, which also provides the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

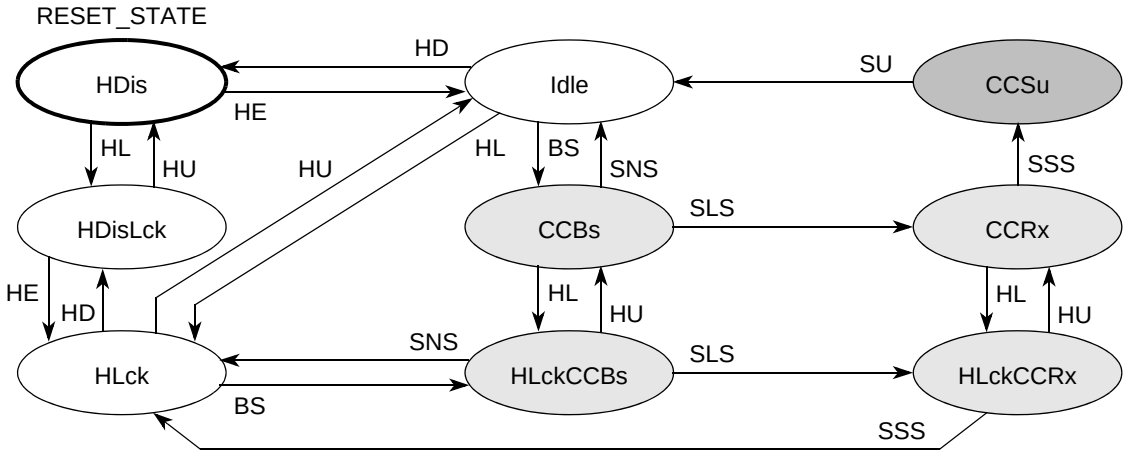


Figure 337. Receive message buffer states

Table 312. Receive message buffer states and access

State	MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	—	SR	**Idle** — Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	—	**Disabled and Locked** — Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	—	**Locked** — Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	—	—	**Buffer Subscribed** — Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	—	**Locked and Buffer Subscribed** — Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	—	—	**Message Receive** — Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	—	Locked and Message Receive — Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	—	RX	**Status Update** — Message buffer status update. Update of status flags, the slot status field, and the header index.

20.6.6.3.1 Message buffer transitions

20.6.6.3.1.1 Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 313*. The application issues the commands by writing to the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

20.6.6.3.2 Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

20.6.6.3.3 Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit MBCCSRn[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is set.

Table 313. Receive message buffer application transitions

Transition	Host Command	Condition	Description
HE	MBCCSRn[EDT]:= 1	MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSRn[LCKT]:= 1	MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

20.6.6.3.3.1 Module transitions

The module transitions that can be triggered by the controller are described in *Table 314*. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 314. Receive message buffer module transitions

Transition	Condition	Description
BS	slot match and CycleCounter match	Buffer Subscribed — The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start — Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start — Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated — Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

20.6.6.3.3.2 Transition priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in *Table 315*, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 315. Receive message buffer transition priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

20.6.6.3.4 Message reception

As a result of the message buffer search, the controller changes the state of as many as two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see *Section 20.6.6.3.7: Receive Shadow Buffers Concept*. The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

20.6.6.3.5 Message buffer update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*.

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in *Table 316*.

Table 316. Receive message buffer update

vSS!ValidFrame	vRF!Header!NFIndicator	Update description
1	1	Valid non-null frame received. - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1
1	0	Valid null frame received. - Message Buffer Data Field *not* updated. - Frame Header Field *not* updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed - MBIF:= 1
0	x	No valid frame received. - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed. - MBIF:= 1, if the slot was not an empty dynamic slot. **Note:** An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

Note: If the number of the last slot in the current communication cycle on a given channel is n, then all receive message buffers assigned to this channel with MBFIDRn[FID] > n will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in *Figure 338*.

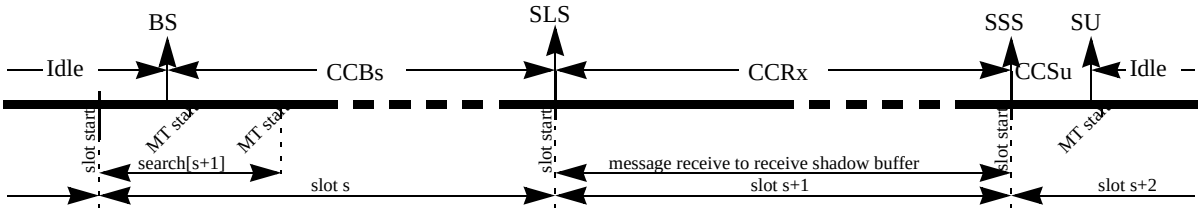


Figure 338. Message reception timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following three items:

1. the message buffer segment that the message buffer is assigned to, as defined by the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*
2. the message buffer data field size, as defined by the related field of the *Section 20.5.2.7: Message Buffer Data Size Register (MBDSR)*
3. the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*MBDSR.MBSEG1DS, the controller writes only 2*MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*MBDSR.MBSEG1DS, the controller writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with MBDSR.MBSEG2DS.

20.6.6.3.6 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in *Section 20.6.3.1: Individual message buffers*.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the controller will not change the content of the message buffer.

20.6.6.3.7 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in *Section 20.6.3.2: Receive shadow buffers*.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see *Table 316*), the controller writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)* with the content of the corresponding internal shadow buffer index register. In all other cases, the controller writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the FlexRay memory located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the FlexRay memory accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*. Instead, the index of the message buffer header field must be fetched from the *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*.

20.6.6.4 Double transmit message buffer

The section provides a detailed description of the functionality of the double transmit message buffers.

Double transmit message buffers are used by the application to provide the controller with the message data to be transmitted over the FlexRay Bus. The controller uses this message buffer to provide information to the application about the transmission process, and status information about the slot in which message data was transmitted.

In contrast to the single transmit message buffers, the application can provide new transmission data while the transmission of the previously provided message data is running. This scheme is called double buffering and can be considered as a FIFO of depth 2.

Double transmit message buffers are implemented by combining two individual message buffers that form the two sides of an double transmit message buffer. One side is called the *commit side* and will be accessed by the application to provide the message data. The other side is called the *transmit side* and is used by the controller to transmit the message data to the FlexRay bus. The two sides are located in adjacent individual message buffers. The message buffer that implements the commit side has an even message buffer number 2n. The transmit side message buffer follows the commit side message buffer and has the message buffer number 2n+1. The basic structure and data flow of a double transmit message buffer is given in *Figure 339*.

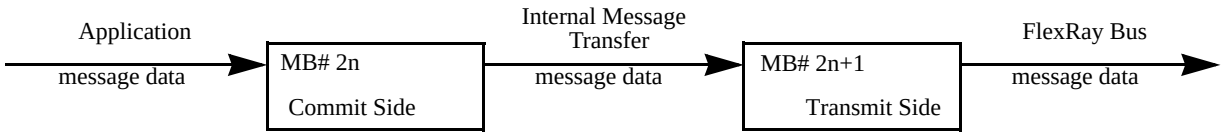


Figure 339. Double transmit buffer structure and data flow

Note: Both the commit and the transmit side must be configured with identical values except for the Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn).

20.6.6.4.1 Access Regions

To certain message buffer fields, both the application and the controller have access. To ensure data consistency, a message buffer locking scheme is implemented, which controls the exclusive access to the data, control, and status bits of the message buffer.

The access scheme for double transmit message buffers is shown in *Figure 340*. The given regions represent fields that can be accessed from both the application and the controller and, thus, require access restrictions. A description of the regions is given in *Table 317*.

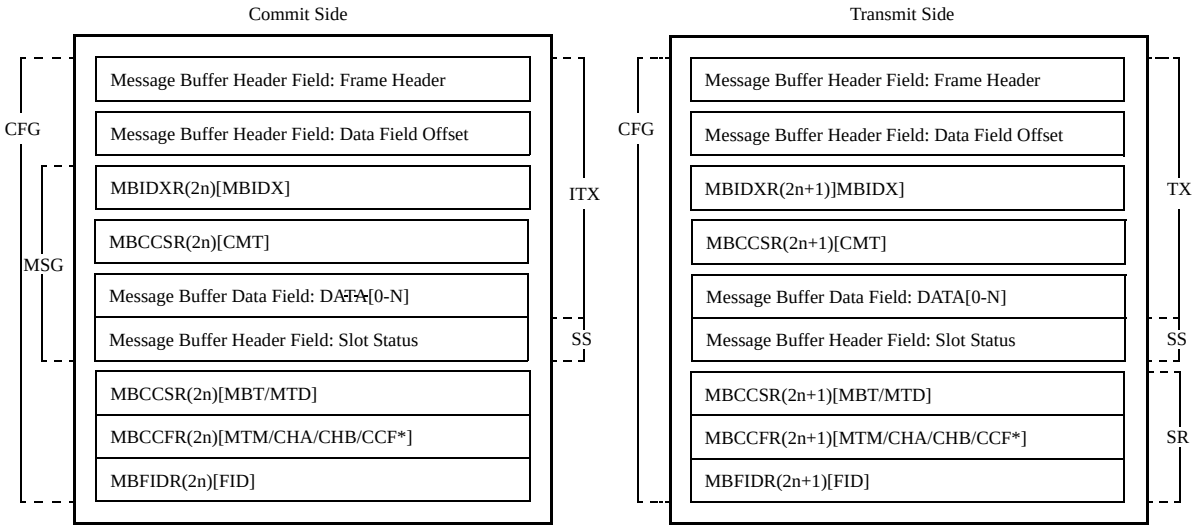


Figure 340. Double transmit message buffer access regions layout

Table 317. Double transmit message buffer access regions description

Access			Description
Region	Type		
	Application	Module	
Commit Side			
CFG	read/write	—	Message Buffer Configuration
MSG	read/write	—	Message Buffer Data and Control access
ITX	—	read/write	Internal Message Transfer.
SS	—	write-only	Slot Status Update
Transmit Side			
CFG	read/write	—	Message Buffer Configuration
SR	—	read-only	Message Buffer Search
TX	—	read-only	Internal Message Transfer, Message Transmission
SS	—	write-only	Slot Status Update

The trigger bits MBCCSRn[EDT] and MBCCSRn[LCKT], and the interrupt enable bit MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] are not under access control and can be accessed from the controller at any time.

The interrupt flag MBCCSnR.MBIF is not under access control and can be accessed from the application and the controller at any time. controller set access has higher priority.

The controller restricts its access to the regions, depending on the current state of the corresponding part of the double transmit message buffer. The application must adhere to these restrictions in order to ensure data consistency. The states for the commit side of a double transmit message buffer are given in *Figure 341*. A description of the states is given in *Table 319*. The states for the transmit side of a double transmit message buffer are given

in *Figure 342*. A description of the states is given in *Table 319*. The description tables also provide the access scheme for the access regions.

The status bits MBCCSRn[EDS] and MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

20.6.6.4.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

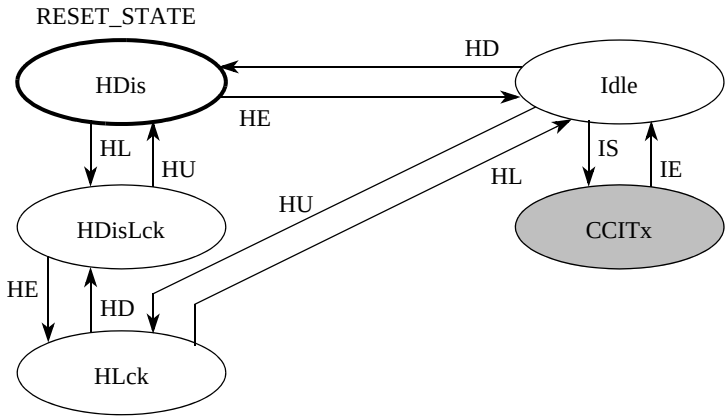


Figure 341. Double transmit message buffer state diagram (commit side)

A description of the states of the commit side of a double transmit message buffer is given in *Table 318*.

Table 318. Double transmit message buffer state description (commit side)

State	MBCCSR(2n)		Access Region		Description
	EDS	LCKS	Appl.	Module	
common states					
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
CCITx	1	0	—	ITX	**Internal Message Transfer** — Message Buffer Data transferred from commit side to transmit side.
commit side specific states					
Idle	1	0	—	ITX, SS	**Idle** — Message Buffer Commit Side is idle. Commit Side can be used for internal message transfer.
HDisLck	0	1	CFG	SS	**Disabled and Locked** — Message Buffer under configuration. Commit Side can *not* be used for internal message transfer.
HLck	1	1	MSG	SS	**Locked** — Applications access to data, control, and status. Commit Side can *not* be used for internal message transfer.

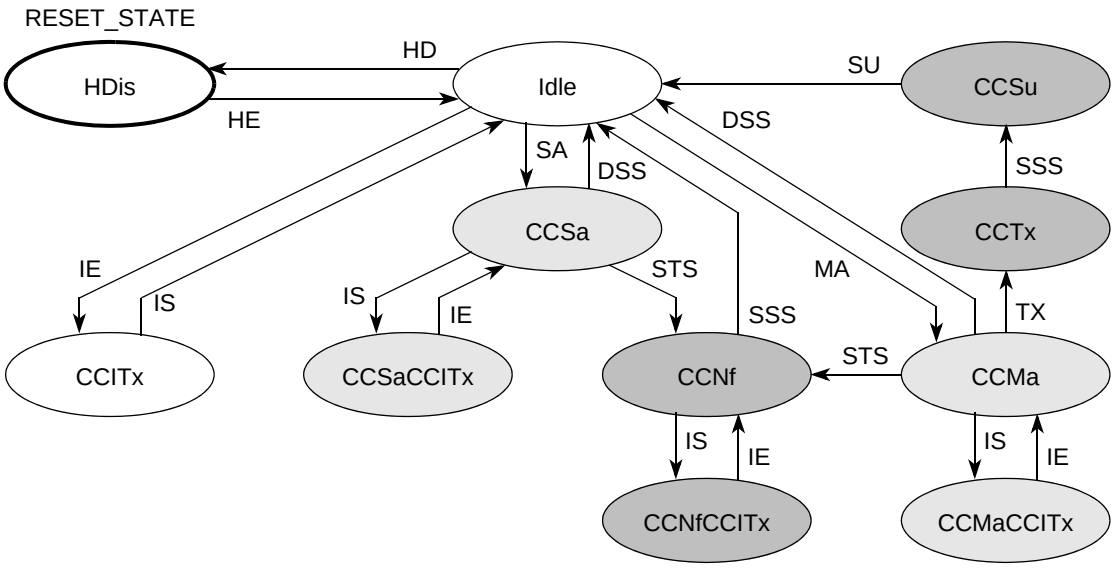


Figure 342. Double transmit message buffer state diagram (transmit side)

A description of the states of the transmit side of a double transmit message buffer is given in *Table 319*.

Table 319. Double transmit message buffer state description (transmit side)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Common states					
HDis	0	0	CFG	—	**Disabled** — Message Buffer under configuration. Excluded from message buffer search.
CCITx	1	0	—	TX	**Internal Message Transfer** — Message Buffer Data transferred from commit side to transmit side.
Transmit side specific states					
Idle	1	0	—	SR	**Idle** — Message Buffer Transmit Side is idle. Transmit Side is included in message buffer search.
CCSa	1	0	—	—	Slot Assigned — Message buffer assigned to next static slot. Ready for Null Frame transmission.
CCSaCCITx	1	0	—	TX	Slot Assigned and Internal Message Transfer — Message buffer assigned to next static slot and Message Buffer Data transferred from commit side to transmit side.
CCNf	1	0	—	TX	Null Frame Transmission Header is used for null frame transmission.
CCNfCCITx	1	0	—	TX	Null Frame Transmission and Internal Message Transfer — Header is used for null frame transmission and Message Buffer Data transferred from commit side to transmit side.

Table 319. Double transmit message buffer state description (transmit side)(Continued)

State	MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCMa	1	0	—	—	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
CCMaCCITx	1	0	—	—	Message Available and Internal Message Transfer — Message buffer is assigned to next slot and cycle counter filter matches and Message Buffer Data transferred from commit side to transmit side.
CCTx	1	0	—	TX	Message Transmission — Message buffer data transmit. Payload data from buffer transmitted
CCSu	1	0	—	SS	Status Update — Message buffer status update. Update of status flags, the slot status field, and the header index. Note: The slot status field of the commit side is updated too, even if the application has locked the commit side.

20.6.6.4.3 Message buffer transitions

20.6.6.4.3.1 Application transitions

The application transitions that can be triggered by the application using the commands described in *Table 320*. The application issues the commands by writing to the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

20.6.6.4.4 Message Buffer Enable and Disable

The enable and disable commands can be issued on the transmit side only. Any enable or disable command issued on the commit side will be ignored without notification. The transitions that will be triggered depends on the value of the EDS bit. The enable and disable commands will affect both the commit side and the transmit side at the same time. If the application triggers the disable transition HD while the transmit side is in one of the states CCSa, CCSaCCITx, CCNf, CCNfCCITx, CCMa, CCMaCCITx, CCTx, or CCSu, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

20.6.6.4.5 Message Buffer Lock and Unlock

The lock and unlock commands can be issued on the commit side only. Any lock or unlock command issued on the transmit side will be ignored and the double transmit buffer lock error flag DBL_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* will be set. The transitions that will be triggered depends on the current value of the LCKS bit. The lock and unlock commands will only affect the commit side. If the application triggers the lock transition HL while the commit side is in the state CCITx, the message buffer state will not be changed and the message buffer lock error flag LCK_EF in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* will be set.

Table 320. Double Transmit Message Buffer Host Transitions

Transition	Host Command	Condition	Description
HE	MBCCSR(2n+1)[EDT]:=1	MBCCSR(2n+1)[EDS] = 0	Application triggers message buffer enable.
HD		MBCCSR(2n+1)[EDS] = 1	Application triggers message buffer disable.
HL	MBCCSR(2n)[LCKT]:=1	MBCCSR(2n)[LCKS] = 0	Application triggers message buffer lock.
HU		MBCCSR(2n)[LCKS] = 1	Application triggers message buffer unlock.

20.6.6.4.5.1 Module Transitions

The module transitions that can be triggered by the controller are described in *Table 321*. The transitions C1 and C2 apply to both sides of the message buffer and are applied at the same time. All other controller transitions apply to the transmit side only.

Table 321. Double Transmit Message Buffer Module Transitions

Transition	Condition	Description
common transitions		
IS	see *Section 20.6.6.4.7: Internal message transfer*"	Internal Message Transfer Start — Start transfer of message data from commit side to transmit side.
IE		Internal Message Transfer End — Stop transfer of message data from commit side to transmit side. Note: The internal message transfer is stopped before the slot or segment start.
transmit side specific transitions		
SA	slot match and static slot	Slot Assigned — Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available — Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and MBCCSR(2n+1)[CMT]=1	Transmission Slot Start — Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated — Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start — Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start — Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start — Start of static slot or dynamic slot or symbol window or NIT.

20.6.6.4.5.2 Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of *Table 322*, the module transitions have a higher priority than the application transitions. The priorities among the controller transitions and the related states are given in the second part of *Table 322*. These priorities apply only to the transmit side. The internal message transmit start transition IS has tho lowest priority.

Table 322. Double transmit message buffer transition priorities

State	Priority	Description
Module vs. Application		
Idle	IS > HD IS > HL	Internal Message Transfer Start > Message Buffer Disable Internal Message Transfer Start > Message Buffer Lock
Module internal		
Idle	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

20.6.6.4.6 Message preparation

The application provides the message data through the commit side. The transmission itself is executed from the transmit side. The transfer of the message data from the commit side to the transmit side is done by the *Internal Message Transfer*, which is described in *Section 20.6.6.4.7: Internal message transfer*.

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field of the commit side and sets the commit bit CMT in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. The physical access to the message buffer data field is described in *Section 20.6.3.1: Individual message buffers*.

As indicated by *Table 318*, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, or HLck. The application can change the state of a message buffer if it issues the appropriate commands shown in *Table 320*. The state change is indicated through the MBCCSRn[EDS] and MBCCSRn[LCKS] status bits.

20.6.6.4.7 Internal message transfer

The internal message transfer transfers the message data from the commit side to the transmit side. The internal message transfer is implemented as the swapping of the content of the *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)* of the commit side and the transmit side. After the swapping, the commit side CMT bit is cleared, the commit side interrupt flag MBIF is set, the transmit side CMT bit is set, and the transmit side DVAL bit is cleared.

The conditions and the point in time when the internal message transfer is started are controlled by the message buffer commit mode bit MCM in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*. The MCM bit configures the message buffer for either the streaming commit mode or the immediate commit mode. A detailed description is given in *Section 20.6.6.4.8: Streaming commit mode* and *Section 20.6.6.4.8.1: Immediate commit mode*. The Internal Message Transfer is triggered with the transition IS. Both sides of the message buffer enter one of the CCITx states. The internal message transfer is finished with the transition IE.

20.6.6.4.8 Streaming commit mode

The intention of the streaming commit mode is to ensure that each committed message is transmitted *at least once*. The controller will not start the Internal Message Transfer for a message buffer as long as the message data on the transmit side is not transmitted at least once.

The streaming commit mode is configured by clearing the message buffer commit mode bit MCM in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for a double transmit message buffer when all of the following conditions are fulfilled:

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e., MBCCSR(2*n*)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa
4. the transmit side contains either no valid message data, i.e., MBCCSR(2*n* + 1)[CMT] = 0 or the message data were transmitted at least once, i.e., MBCCSR(2*n* + 1)[DVAL] = 1

An example of a streaming commit mode state change diagram is given in *Figure 343*. In this example, both the commit and the transmit side do not contain valid message data and the application provides two messages. The message buffer does not match the next slot.

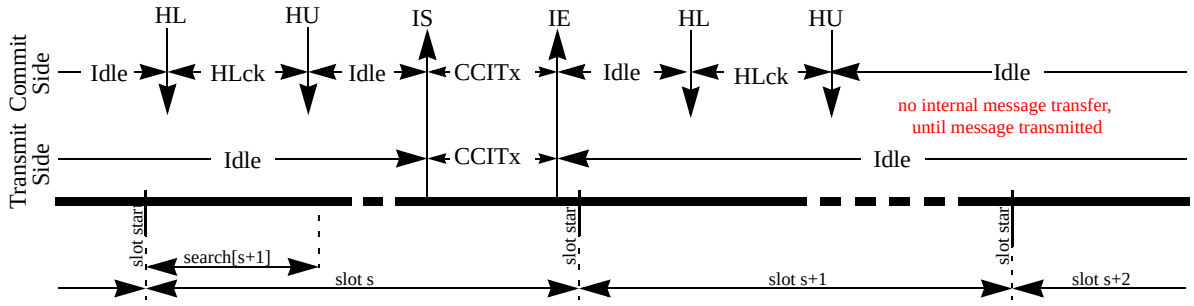


Figure 343. Internal message transfer in streaming commit mode

20.6.6.4.8.1 Immediate commit mode

The intention of the immediate commit mode is to transmit the *latest* data provided by the application. This implies that it is not guaranteed that each provided message will be transmitted at least once.

The immediate commit mode is configured by setting the message buffer commit mode bit MCM in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*.

In this mode, the internal message transfer from the commit side to the transmit side is started for one double transmit message buffer when all of the following conditions are fulfilled:

1. the commit side is in the Idle state
2. the commit site message data are valid, i.e., MBCCSR(2*n*)[CMT] = 1
3. the transmit side is in one of the states Idle, CCSa, or CCMa

It is not checked whether the transmit side contains no valid message data or valid message data were transmitted at least once. If message data are valid and not transmitted, they may be overwritten.

An example of a streaming commit mode state change diagram is given in *Figure 344*. In this example, both the commit and the transmit side do not contain valid message data, and

the application provides two messages and the first message is gets overwritten. The message buffer does not match the next slot.

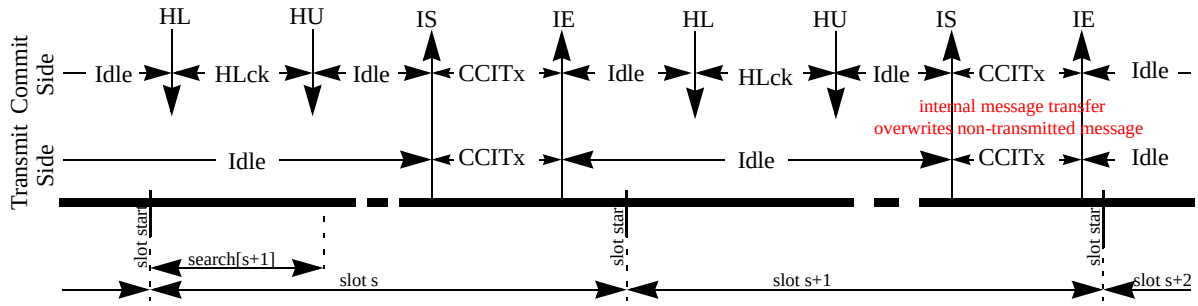


Figure 344. Internal message transfer in immediate commit mode

20.6.6.4.9 Message transmission

For double transmit message buffers, the message buffer search checks only the transmit side part. The internal scheduling ensures that the internal message transfer is stopped on the message buffer search start. Thus, the transmit side of message buffer, that is not in its transmission or status update slot, is always in the Idle state.

The message transmit behavior and transmission state changes of the transmit side of a double transmit message buffer are the same as for single buffered transmit buffers, except that the transmit side of double buffers cannot be locked by the application, i.e., the HU and HL transition do not exist. Therefore, refer to *Section 20.6.6.2.7: Message transmission*.

20.6.6.4.10 Message buffer status update

The message buffer status update behavior of the transmit side of a double transmit message buffer is the same as for single transmit message buffers which is described in *Section 20.6.6.2.9: Message buffer status update*.

Additionally, the slot status field of the commit side is update after the update of the slot status field of the transmit side, even if the commit side is locked by the application. This is implemented to provide the slot status of the most recent transmission slot.

20.6.7 Individual message buffer search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers that are

1. Enabled, i.e., MBCCSR*n*[EDS] = 1, and
2. Matches the next slot *n*, i.e., MBFIDR*n*[FID] = *n*, and
3. Are the transmit side buffer in case of a double transmit message buffer.

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of *Table 323*. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of *Table 324*. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to *Table 323* for the static segment and according to *Table 324* for the dynamic segment are selected.

Table 323. Message buffer search priority (static segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	—	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	—	1	transmit buffer, matches cycle count, locked	SA
2	1	—	—	—	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section 20.6.7.1: Message buffer cycle counter filtering*.

Table 324. Message buffer search priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM (1)	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

1. Cycle Counter Filter Match, see *Section 20.6.7.1: Message buffer cycle counter filtering*.

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in *Section 20.6.7.2: Message buffer channel assignment consistency*.

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in *Section 20.6.3.1: Individual message buffers*.

20.6.7.1 Message buffer cycle counter filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the *Section 20.5.2.66: Message Buffer Cycle Counter Filter Registers (MBCCFRn)*. This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled, i.e., CCFE = 0, this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

Equation 25

$$\text{MBCCFRn[CCFE]} = 0$$

Equation 26

$$\text{CYCCNT} \,\&\, \text{MBCCFRn[CCFMSK]} = \text{MBCCFRn[CCFVAL]} \,\&\, \text{MBCCFRn[CCFMSK]}$$

20.6.7.2 Message buffer channel assignment consistency

The message buffer channel assignment given by the CHA and CHB bits in the *Section 20.5.2.66: Message Buffer Cycle Counter Filter Registers (MBCCFRn)* defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if MBCCFR*n*[CHA] = 1 and transmits or receives on channel B if MBCCFR*n*[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is shown in *Figure 345*.

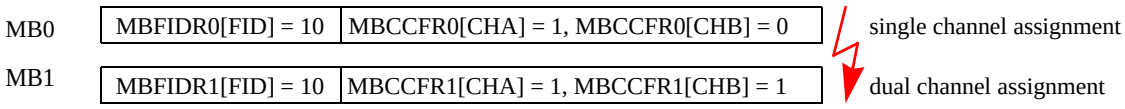


Figure 345. Inconsistent channel assignment

20.6.7.3 Node related slot multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to *Table 324* the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

20.6.7.4 Message buffer search error

If the message buffer search is running while the next message buffer search start event appears, the message buffer search is stopped and the Message Buffer Search Error Flag MSB_EF is set in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*. This appears only if the CHI frequency is too low to search through all message buffers within the NIT or a minislot. The message buffer result is not defined in this case. For more details see *Section 20.7.3: Number of usable message buffers*.

20.6.8 Individual message buffer reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on *Section 20.6.3.4.1.2: Specific configuration data*. The common configuration data given in the section on *Section 20.6.3.4.1.2: Specific configuration data* cannot be reconfigured when the protocol is out of the *POC:config* state.

20.6.8.1 Reconfiguration schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

20.6.8.1.1 Basic type not changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if both the message buffer transfer direction bit MBCCSn[MTD] and the message buffer type bit MBCCSn[MBT] are not changed. This type of reconfiguration is denoted by RC1 in *Figure 346*. Single transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state. Double transmit message buffers can be RC1-reconfigured if both the transmit side and the commit side are in the HDis state.

20.6.8.1.2 Buffer type not changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer if the message buffer buffer type bit MBCCSRn[MBT] is not changed. This type of reconfiguration is denoted by RC2 in *Figure 346*. It applies only to single transmit and receive message buffers. Single transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

20.6.8.1.3 Buffer type changed (RC3)

A reconfiguration will change the buffer type of the individual message buffer if the message buffer type bit MBCCSRn[MBT] is changed. This type of reconfiguration is denoted by RC3

in *Figure 346*. The RC3 reconfiguration splits one double buffer into two single buffers or combines two single buffer into one double buffer. In the later case, the two single message buffers must have consecutive message buffer numbers and the smaller one must be even. Message Buffers can be RC3 reconfigured if they are in the HDis state.

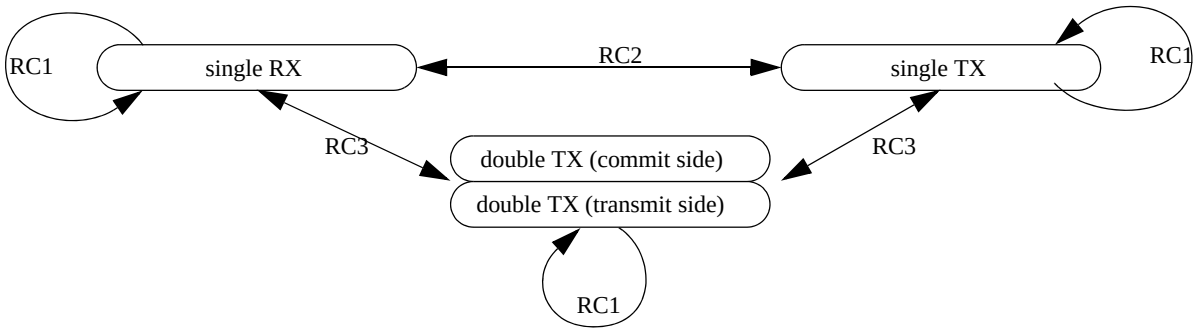


Figure 346. Message buffer reconfiguration scheme

20.6.9 Receive FIFO

This section provides a detailed description of the two receive FIFOs.

20.6.9.1 Overview

The receive FIFOs implement the queued receive buffer defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.* One receive FIFO is assigned to channel A, the other receive FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in *Section 20.6.3.3: Receive FIFO*. The area in the FlexRay memory for each of the two receive FIFOs is characterized by:

- The index of the first FIFO entry given by *Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)*
- The number of FIFO entries and the length of each FIFO entry as given by *Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)*

20.6.9.2 Receive FIFO configuration

The receive FIFO control and configuration data are given in *Section 20.6.3.7: Receive FIFO control and configuration data*. The configuration of the receive FIFOs consists of two steps.

The first step is the allocation of the required amount of FlexRay memory for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see *Section 20.6.4: FlexRay memory layout*.

The second step is the programming of the configuration data register while the PE is in *POC:config.*

The following steps configure the layout of the FIFO.

- The number of the first message buffer header index that belongs to the FIFO is written into the *Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)*.
- The depth of the FIFO is written into the FIFO_DEPTH field in the *Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)*.
- The length of the message buffer data field for the FIFO is written into the ENTRY_SIZE field in the *Section 20.5.2.53: Receive FIFO Depth and Size Register (RFDSR)*.

Note: To ensure that the read index RDIDX always points to a message buffer that contains valid data, the receive FIFO must have at least two entries.

The FIFO filters are configured through the FIFO filter registers.

20.6.9.3 Receive FIFO reception

The frame reception to the receive FIFO is enabled if, for a certain slot, no message buffer is assigned or subscribed. In this case the FIFO filter path shown in *Figure 347* is activated.

When the receive FIFO filter path indicates that the received frame must be appended to the FIFO, the controller writes the received frame header and slot status into the message buffer header field indicated by the internal FIFO header write index. The payload data are written in the message buffer data field. If the status of the received frame indicates a valid frame, the internal FIFO header write index is updated and the FIFO not-empty interrupt flag FNEAIF/FNEBIF in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)* is set.

20.6.9.4 Receive FIFO message access

If the FIFO not-empty interrupt flag FNEAIF/FNEBIF in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)* is set, the receive FIFO contains valid received messages, which can be accessed by the application.

The receive FIFO does not require locking to access the message buffers. To access the message the application first reads the receive FIFO read index RDIDX from the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* or *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)*, respectively. This index points to the message buffer header field of the next message buffer that contains valid data. The application can access the message data as described in *Section 20.6.3.3: Receive FIFO*. When the application has read all message buffer data and status information, it writes 1 to the FIFO not-empty interrupt flags FNEAIF or FNEBIF. This clears the interrupt flag and updates the RDIDX field in the *Section 20.5.2.54: Receive FIFO A Read Index Register (RFARIR)* or *Section 20.5.2.55: Receive FIFO B Read Index Register (RFBRIR)*, respectively.When the RDIDX value has reached the last message buffer header field that belongs to the FIFO, it wraps around to the index of the first message buffer header field that belongs to the FIFO. This value is provided by the SIDX field in the *Section 20.5.2.52: Receive FIFO Start Index Register (RFSIR)*.

20.6.9.5 Receive FIFO filtering

The receive FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The controller provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the

pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is shown in *Figure 347*.

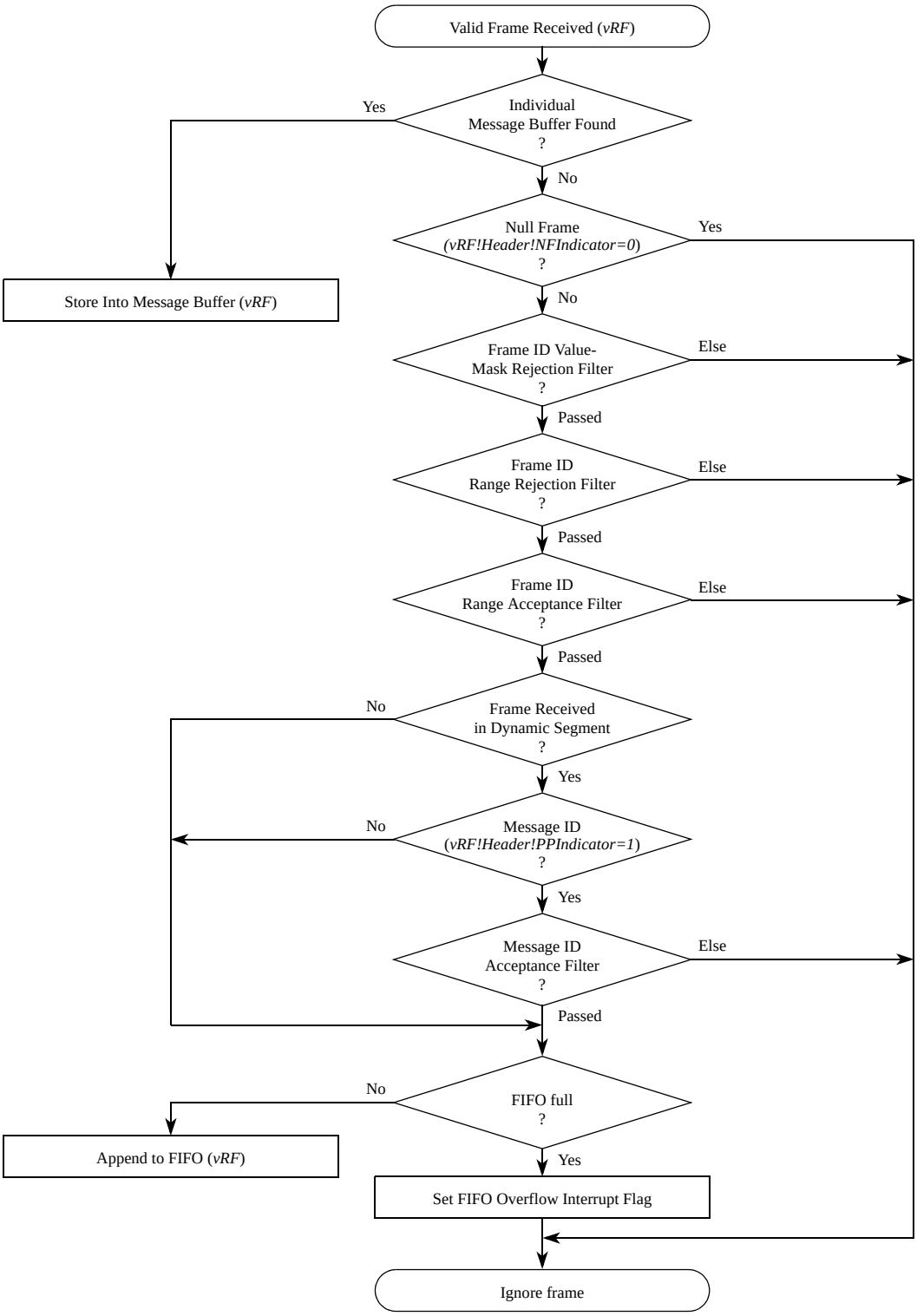


Figure 347. Received frame FIFO filter path

A received frame passes the FIFO filtering if it has passed all three types of filter.

20.6.9.5.1 RX FIFO frame ID value-mask rejection filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the *Section 20.5.2.58: Receive FIFO Frame ID Rejection Filter Value Register (RFFIDRFVR)* and the *Section 20.5.2.59: Receive FIFO Frame ID Rejection Filter Mask Register (RFFIDRFMR)*. Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e., is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if *Equation 27* is fulfilled.

Equation 27

$$FID \,\&\, RFFIDRFMR[FIDRFMSK] \neq RFFIDRFVR[FIDRFVAL] \,\&\, RFFIDRFMR[FIDRFMSK]$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- RFFIDRFVR[FIDRFVAL] := 0x000 and RFFIDRFMR[FIDRFMSK] := 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- RFFIDRFMR[FIDRFMSK] := 0x000

Using the settings above, *Equation 27* can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

20.6.9.5.2 RX FIFO frame ID range rejection filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the *Section 20.5.2.60: Receive FIFO Range Filter Configuration Register (RFRFCFR)* and controlled by the *Section 20.5.2.61: Receive FIFO Range Filter Control Register (RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e., RFRFCTR.FiMD = 1 and RFRFCTR.FiEN = 1, *Equation 28* is fulfilled.

Equation 28

$$(FID < RFRFCFR_{SEL}[SID_{IBD=0}]) \text{ or } (RFRFCFR_{SEL}[SID_{IBD=1}] < FID)$$

Consequently, all frames with a frame ID that fulfills *Equation 29* for at least one of the enabled rejection filters will be rejected and thus not pass.

Equation 29

$$RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq RFRFCFR_{SEL}[SID_{IBD=1}]$$

20.6.9.5.3 RX FIFO frame ID range acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the *Section 20.5.2.60: Receive FIFO Range Filter Configuration Register (RFRFCFR)* and controlled by the *Section 20.5.2.61: Receive FIFO Range Filter Control Register (RFRFCTR)*. The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e., RFRFCTR.FiMD = 0 and RFRFCTR.FiEN = 1, *Equation 30* is fulfilled.

Equation 30

$$RFRFCFR_{SEL}[SID_{IBD=0}] \leq FID \leq RFRFCFR_{SEL}[SID_{IBD=1}]$$

20.6.9.5.4 RX FIFO message ID acceptance filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the *Section 20.5.2.56: Receive FIFO Message ID Acceptance Filter Value Register (RFMIDAFVR)* and the *Section 20.5.2.57: Receive FIFO Message ID Acceptance Filter Mask Register (RFMIAFMR)*. This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if *Equation 31* is fulfilled.

Equation 31

$$MID \;\&\; RFMIDAFMR[MIDAFMSK] =$$

$$= RFMIDAFMR[MIDAFVAL] \;\&\; RFMIDAFMR[MIDAFMSK]$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- RFMIDAFMR[MIDAFMSK] := 0x000

Using the settings above, *Equation 31* is always fulfilled and all frames will pass.

20.6.10 Channel device modes

This section describes the two FlexRay channel device modes that are supported by the controller.

20.6.10.1 Dual channel device mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and <Overbar>FR_B_TX_EN is connected to the physical bus channel B. The dual channel system is shown in *Figure 348*.

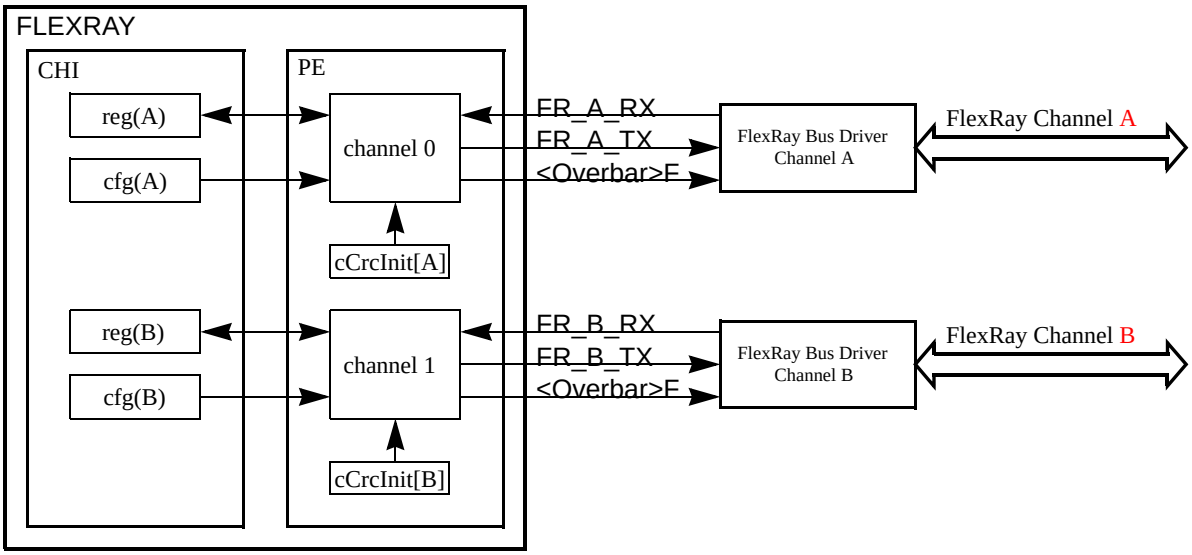


Figure 348. Dual channel device mode

20.6.10.2 Single channel device mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and <Overbar>FR_A_TX_EN and can be connected to either the physical bus channel A (shown in *Figure 349*) or the physical bus channel B (shown in *Figure 350*).

If the device is configured as a single channel device by setting MCR.SCD to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of MCR.CHA and MCR.CHB, the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit MCR.CHA must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit MCR.CHB must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

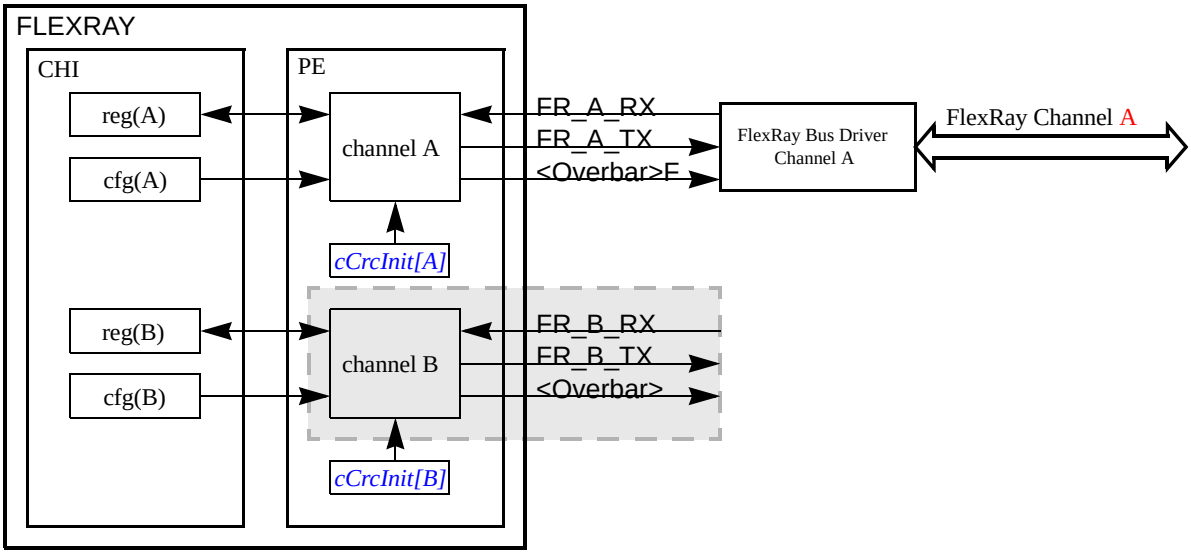


Figure 349. Single channel device mode (Channel A)

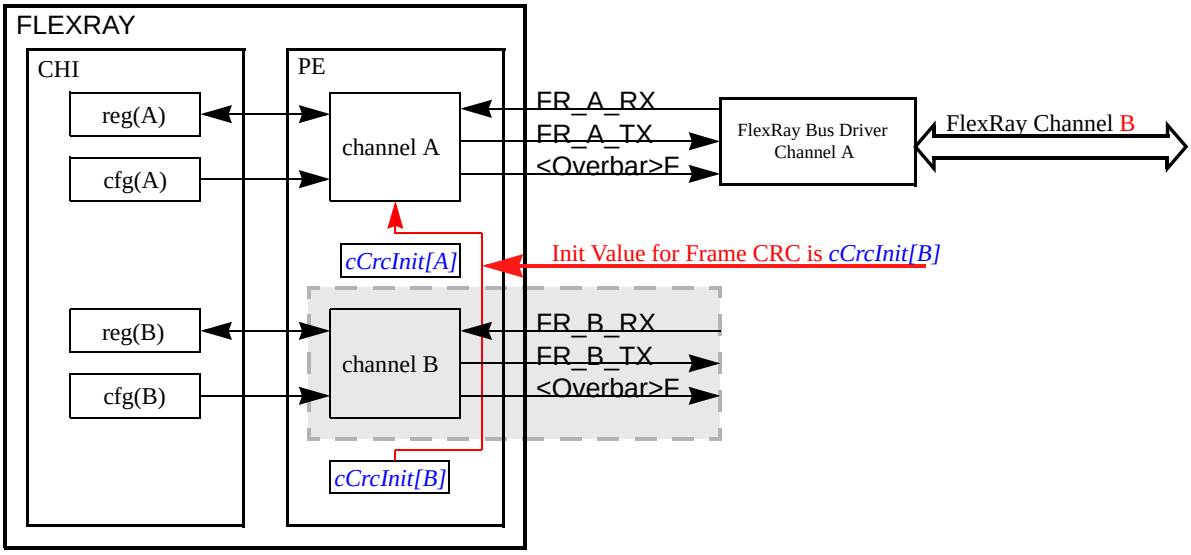


Figure 350. Single channel device mode (Channel B)

20.6.11 External clock synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*. The PE applies the external correction values in the next even-odd cycle pair as shown in *Figure 351* and *Figure 352*.

Note: The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to *Figure 351* for timing details.

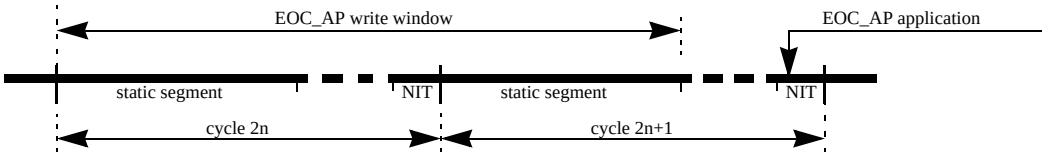


Figure 351. External offset correction write and application timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affected by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to *Figure 352* for details.

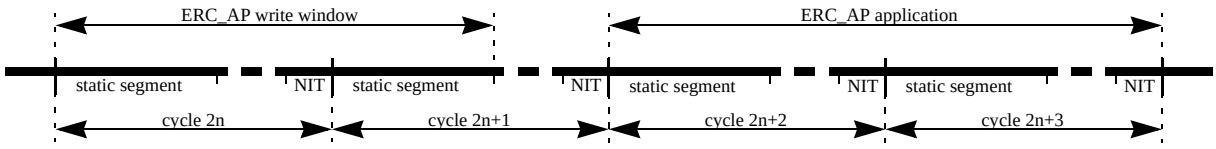


Figure 352. External rate correction write and application timing

20.6.12 Sync frame ID and sync frame deviation tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The controller provides the means to write a copy of these internal tables into the FlexRay memory and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the controller will not overwrite these tables and the application can read a consistent snapshot.

Note: Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

20.6.12.1 Sync frame ID table content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

20.6.12.2 Sync frame deviation table content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

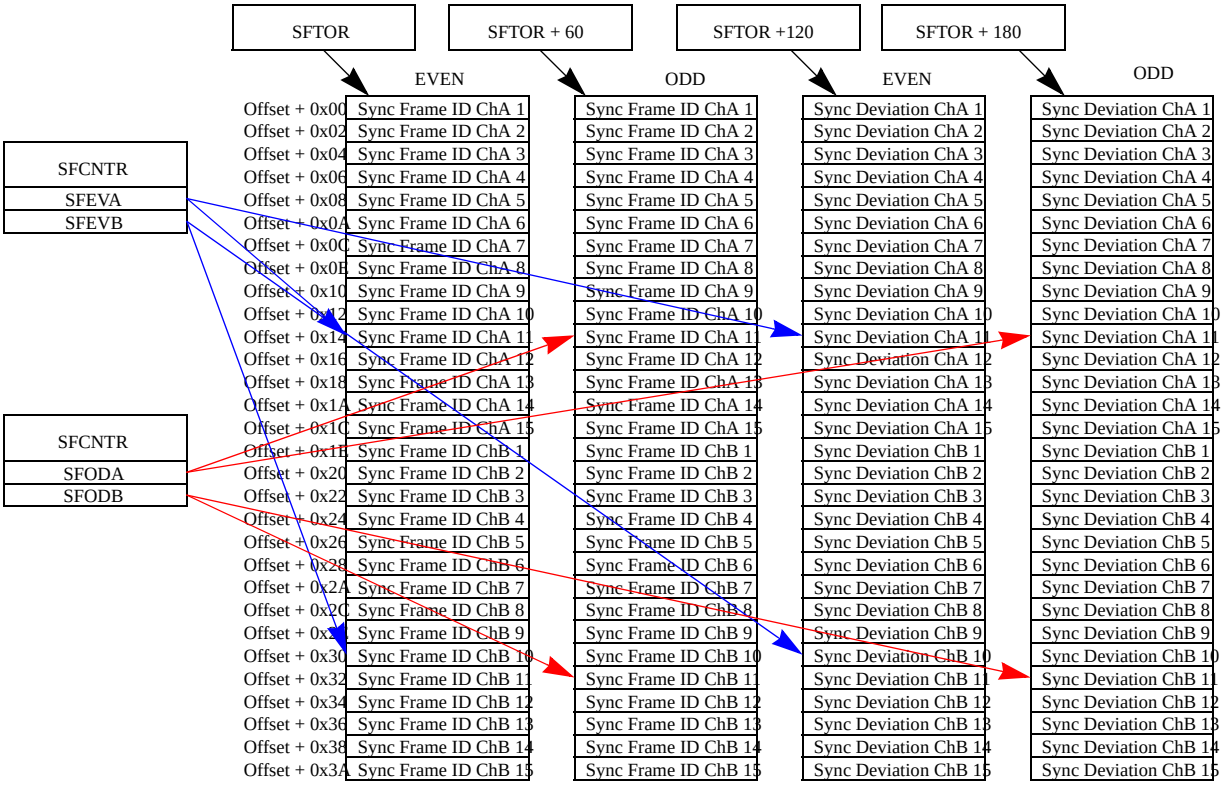


Figure 353. Sync table memory layout

20.6.12.3 Sync frame ID and sync frame deviation table setup

The controller writes a copy of the internal synchronization frame ID and deviation tables into the FlexRay memory if requested by the application. The application must provide the appropriate amount of FlexRay memory for the tables. The memory layout of the tables is given in *Figure 353*. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the *Section 20.5.2.32: Sync Frame Table Offset Register (SFTOR)*.

20.6.12.4 Sync frame ID and sync frame deviation table generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the FlexRay memory using the *Section 20.5.2.33: Sync Frame Table Configuration, Control, Status Register (SFTCCSR)*. A summary of the copy modes is given in *Table 325*.

Table 325. Sync frame table generation modes

SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
0	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting SFTCCSR.SIDEN to 1, the controller starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The controller checks if the application has locked the tables by reading the SFTCCSR.ELKS lock status bit. If this bit is set, the controller will not update the table in this cycle. If this bit is cleared, the controller locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the controller clears the even table valid status bit SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the FlexRay memory, the controller sets the even table valid bit SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)*. If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the controller from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger SFTCCSR.ELKT. When the even table is not currently updated by the controller, the lock is granted and the even table lock status bit SFTCCSR.ELKS is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the Sync Frame Counter Register (SFCNTR). The value in the SFTCCSR.CYCNUM field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the SFCNTR.SFEVA and SFCNTR.SFEVB fields. The application can now start to read the sync table data from the locations given in *Figure 353*.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger SFTCCSR.ELKT again. The even table lock status bit SFTCCSR.ELKS is reset immediately.

If the sync frame table generation is disabled, the table valid bits SFTCCSR[EVAL] and SFTCCSR[EVAL] are reset when the counter values in the Sync Frame Counter Register (SFCNTR) are updated. This is done because the tables stored in the FlexRay memory are no longer related to the values in the Sync Frame Counter Register (SFCNTR).

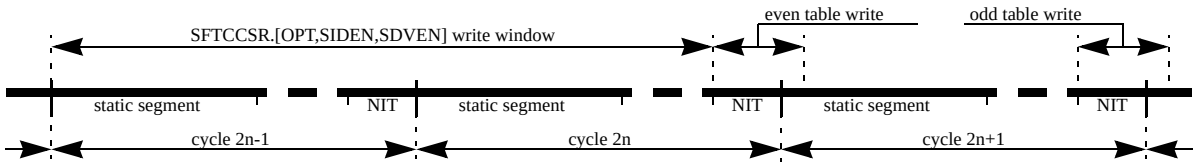


Figure 354. Sync frame table trigger and generation timing

20.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the FlexRay memory during the table write windows shown in *Figure 354*. During the table write, the application cannot lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

20.6.12.5.1 Sync frame table locking and unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the *Section 20.5.2.33: Sync Frame Table Configuration, Control, Status Register (SFTCCSR)*. If the affected table is not currently written to the FlexRay memory, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the FlexRay memory, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

20.6.13 MTS generation

The controller provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the *Section 20.5.2.48: MTS A Configuration Register (MTSACFR)* and *Section 20.5.2.49: MTS B Configuration Register (MTSBCFR)*.

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the *Section 20.5.2.48: MTS A Configuration Register (MTSACFR)* or *Section 20.5.2.49: MTS B Configuration Register (MTSBCFR)*. If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if *Equation 33*, *Equation 34*, and *Equation 35* are fulfilled.

Equation 32

$$PSR0[PROTSTATE] = \textit{POC:normal active}$$

Equation 33

$$MTSACRF[MTE] = 1$$

Equation 34

$$CYCCNT \& MTSACFR[CYCCNTMSK] =$$

$$= MTSACFR[CYCCNTVAL] \& MTSACFR[CYCCNTMSK]$$

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if *Equation 32*, *Equation 35*, and *Equation 36* are fulfilled.

Equation 35

$$MTSBCRF[MTE] = 1$$

Equation 36

$$CYCCNT \& MTSBCFR[CYCCNTMSK] =$$

$$= MTSBCFR[CYCCNTVAL] \& MTSBCFR[CYCCNTMSK]$$

20.6.14 Key slot transmission

20.6.14.1 Key slot assignment

A key slot is assigned to the controller if the key_slot_id field in the *Section 20.5.2.64.19: Protocol Configuration Register 18 (PCR18)* is configured with a value greater than 0 and less or equal to number_of_static_slots in *Section 20.5.2.64.3: Protocol Configuration Register 2 (PCR2)*, otherwise no key slot is assigned.

20.6.14.2 Key slot transmission in *POC:startup*

If a key slot is assigned and the controller is in the *POC:startup* state, startup null frames will be transmitted as specified by FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

20.6.14.3 Key slot transmission in *POC:normal active*

If a key slot is assigned and the controller is in *POC:normal active*, a frame of the type as shown in *Table 326* is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see *Section 20.6.6.2.7:*

Message transmission"). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see *Section 20.6.6.2.8: Null frame transmission*).

Table 326. Key slot frame type

PCR11[key_slot_used_for_sync]	PCR11[key_slot_used_for_startup]	Key slot frame type
0	0	normal frame
0	1	normal frame(1)
1	0	sync frame
1	1	startup frame

1. The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

20.6.15 Sync frame filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e., the SFFE control bit in the *Section 20.5.2.4: Module Configuration Register (MCR)* is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

20.6.15.1 Sync frame acceptance filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the *Section 20.5.2.35: Sync Frame ID Acceptance Filter Value Register (SFIDAFVR)* and the mask is configured in the *Section 20.5.2.36: Sync Frame ID Acceptance Filter Mask Register (SFIDAFMR)*. A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if *Equation 37* or *Equation 38*evaluates to true.

Equation 37

$$MCR[SFFE] = 0$$

Equation 38

$$FID \,\&\, SFIDAFMR[FMSK] = SFIDAFVR[FVAL] \,\&\, SFIDAFMR[FMSK]$$

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

20.6.15.2 Sync frame rejection filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the *Section 20.5.2.34: Sync Frame ID Rejection Filter Register (SFIDRFR)*. A received synchronization frame with the frame ID FID passes the sync frame rejection filter if *Equation 39* or *Equation 40* evaluates to true.

Equation 39

$$MCR[SFFE] = 0$$

Equation 40

$$FID \neq SFIDRFR[SYNFRID]$$

Note: Sync frames are transmitted in the static segment only. Thus FID <= 1023.

20.6.16 Strobe signal support

The controller provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in *Table 226*.

20.6.16.1 Strobe signal assignment

Each of the strobe signals listed in *Table 226* can be assigned to one of the four strobe ports using the *Section 20.5.2.6: Strobe Signal Control Register (STBSCR)*. To assign multiple strobe signals, the application must write multiple times to the *Section 20.5.2.6: Strobe Signal Control Register (STBSCR)* with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence:

1. Write to STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
2. Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

20.6.16.2 Strobe signal timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in *Table 226* with a negative clock offset. An example waveform is given in *Figure 355*.

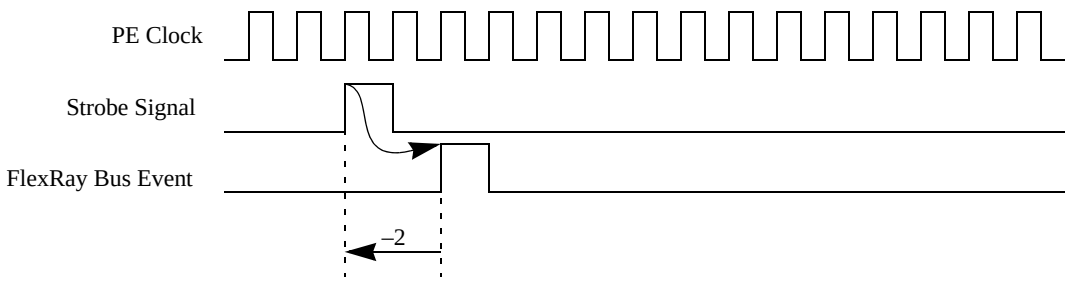


Figure 355. Strobe signal timing (type = pulse, clk_offset = –2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in *Table 226* with a positive clock offset. An example waveform is given in *Figure 356*.

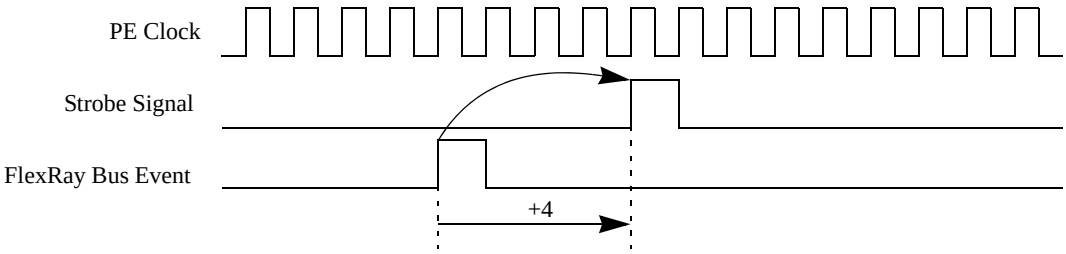


Figure 356. Strobe signal timing (type = pulse, clk_offset = +4)

20.6.17 Timer support

The controller provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

20.6.17.1 Absolute timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* is set at the macrotick start event, if and *Equation 42* are fulfilled

Equation 41

$$CYCTR[CTCCNT] \,\&\, TI1CYSR[T1\_CYC\_MSK] =$$

$$= TI1CYSR[T1\_CYC\_VAL] \,\&\, TI1CYSR[T1\_CYC\_MSK]$$

Equation 42

$$MTCTR[MTCT] = TI1MTOR[T1\_MTOFFSET]$$

If the timer 1 interrupt enable bit TI1_IE in the *Section 20.5.2.13: Protocol Interrupt Enable Register 0 (PIER0)* is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

20.6.17.2 Absolute / relative timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the *Section 20.5.2.39: Timer Configuration and Control Register (TICCR)*. The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

20.6.17.2.1 Absolute timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from *Section 20.5.2.42: Timer 2 Configuration Register 0 (TI2CR0)* and *Section 20.5.2.43: Timer 2 Configuration Register 1 (TI2CR1)* is used. On expiration of timer T2, the interrupt flag TI2_IF in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* is set. If the timer 1 interrupt enable bit TI1_IE in the *Section 20.5.2.13: Protocol Interrupt Enable Register 0 (PIER0)* is asserted, an interrupt request is generated.

20.6.17.2.2 Relative timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* is set, when the programmed amount of macroticks MT[31:0], defined by *Section 20.5.2.42: Timer 2 Configuration Register 0 (TI2CR0)* and *Section 20.5.2.43: Timer 2 Configuration Register 1 (TI2CR1)*, has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

20.6.18 Slot status monitoring

The controller provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in *Table 327*. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in *Figure 357*.

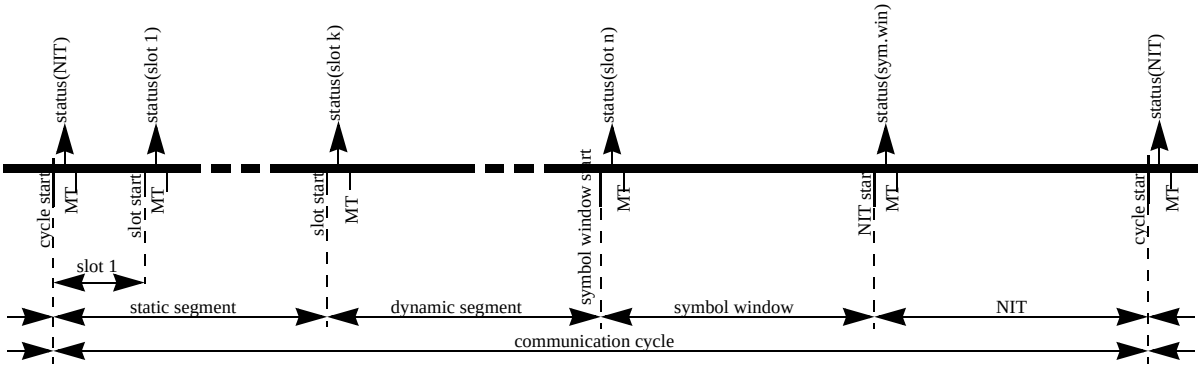


Figure 357. Slot status vector update

Note: The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 327. Slot status content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if vSS!ValidFrame = 1
	b) last received header, if vSS!ValidFrame = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Table 327. Slot status content(Continued)

	Status Content
Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

20.6.18.1 Channel status error counter registers

The two channel status error counter registers, *Section 20.5.2.17: Channel A Status Error Counter Register (CASERCR)* and *Section 20.5.2.18: Channel B Status Error Counter Register (CBSERCR)*, incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

20.6.18.2 Protocol status registers

The *Section 20.5.2.21: Protocol Status Register 2 (PSR2)* provides slot status information about the Network Idle Time NIT and the Symbol Window. The *Section 20.5.2.22: Protocol Status Register 3 (PSR3)* provides aggregated slot status information.

20.6.18.3 Slot status registers

The eight slot status registers, *Section 20.5.2.46: Slot Status Registers (SSR0–SSR7)*, can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in *Table 327*, except of the *first valid* indicator for non-transmission slots.

20.6.18.4 Slot status counter registers

The controller provides four slot status error counter registers, *Section 20.5.2.47: Slot Status Counter Registers (SSCR0–SSCR3)*. Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the *Section 20.5.2.45: Slot Status Counter Condition Register (SSCCR)*, matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken

into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in *Figure 358*.

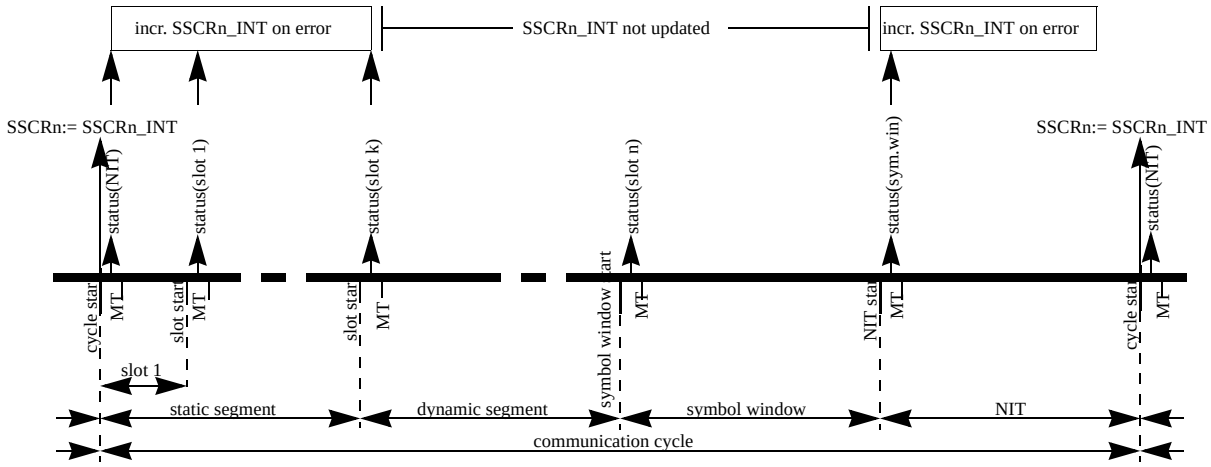


Figure 358. Slot status counting and SSCR*n* update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (SSCCR*n*.VFR | SSCCR*n*.SYF | SSCCR*n*.NUF | SSCCR*n*.SUF) // count on frame condition = 1;

and

- ((~SSCCR*n*.VFR | *vSS!ValidFrame*) & // valid frame restriction
  (~SSCCR*n*.SYF | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~SSCCR*n*.NUF | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~SSCCR*n*.SUF | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction
  = 1;

Note: The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((SSCCR*n*.STATUSMASK[3] & *vSS!ContentError*) | // increment on content error
  (SSCCR*n*.STATUSMASK[2] & *vSS!SyntaxError*) | // increment on syntax error
  (SSCCR*n*.STATUSMASK[1] & *vSS!BViolation*) | // increment on boundary violation
  (SSCCR*n*.STATUSMASK[0] & *vSS!TxConflict*)) // increment on transmission conflict
  = 1;

If the slot status counter is in single cycle mode, i.e., SSCCR*n*[MCY] = 0, the internal slot status counter SSCR*n*_INT is reset at each cycle start. If the slot status counter is in the multicycle mode, i.e., SSCCR*n*[MCY] = 1, the counter is not reset and incremented, until the maximum value is reached.

20.6.18.5 Message buffer slot status field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in *Table 327* for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in *Section 20.6.6: Individual message buffer functional description*.

20.6.19 System bus access

This section provides a description of the system bus accesses performed by the controller.

All FlexRay memory data located in the system memory are accessed via the system bus. There are two types of failures that can occur during the system bus access, the system bus illegal address access and the system bus access timeout.

The behavior of the controller after the occurrence of a system bus failure is defined by the SBFF bit in the *Section 20.5.2.4: Module Configuration Register (MCR)*.

20.6.19.1 System bus illegal address access

If the system bus detects an controller access to an illegal address, the controller receives a notification from the system bus about this event and sets the ILSA_EF flag in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*.

20.6.19.2 System bus access timeout

The controller starts a timer when it has send an access request to the system bus. This timer expires after 2 × SYMATOR.TIMEOUT + 2 system bus clock cycles. If the access is not finished within this amount of time, the SBCF_EF flag in the *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)* is set.

20.6.19.3 Continue after system bus failure

If the SBFF bit in the *Section 20.5.2.4: Module Configuration Register (MCR)* is 0, the controller will continue its operation after the occurrence of the system bus access failure but will not generate any system bus accesses until the start of the next communication cycle.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

20.6.19.4 Freeze after system bus failure

If the SBFF bit in the *Section 20.5.2.4: Module Configuration Register (MCR)* is set to 1, the controller will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

20.6.20 Interrupt support

The controller provides 8 individual interrupt sources and five combined interrupt sources.

20.6.20.1 Individual interrupt sources

20.6.20.1.1 Receive FIFO interrupts

The controller provides 2 Receive FIFO interrupt sources.

Each of the 2 Receive FIFO provides a Receive FIFO Not Empty Interrupt Flag. The controller sets the Receive FIFO Not Empty Interrupt Flags (GIFER.FNEBIF, GIFER.FNEAIF) in the *Section 20.5.2.10: Global Interrupt Flag and Enable Register (GIFER)* if the corresponding Receive FIFO is not empty.

20.6.20.1.2 Wakeup interrupt

The controller provides one interrupt source related to the wakeup.

The controller sets the Wakeup Interrupt Flag GIFER.WUPIF when it has received a wakeup symbol on the FlexRay bus. The controller generates an interrupt request if the interrupt enable bit GIFER.WUPIE is asserted.

20.6.20.2 Combined interrupt sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

20.6.20.2.1 Receive message buffer interrupt

The combined receive message buffer interrupt request RBIRQ is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.RBIE is set.

20.6.20.2.2 Transmit message buffer interrupt

The combined transmit message buffer interrupt request TBIRQ is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit GIFER.TBIE is asserted.

20.6.20.2.3 Protocol interrupt

The controller provides 25 interrupt sources for protocol related events. For details, see *Section 20.5.2.11: Protocol Interrupt Flag Register 0 (PIFR0)* and *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)*. Each interrupt source has its own interrupt enable bit.

The combined protocol interrupt request PRTIRQ is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit GIFER.PRIE is set.

20.6.20.2.4 Module interrupt

The combined module interrupt request MIRQ is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit GIFER.MIE is set.

20.6.20.2.5 CHI error interrupts

The controller provides 16 interrupt sources for CHI related error events. For details, see *Section 20.5.2.15: CHI Error Flag Register (CHIERFR)*. There is one common interrupt enable bit GIFER.CHIIE for all CHI error interrupt sources.

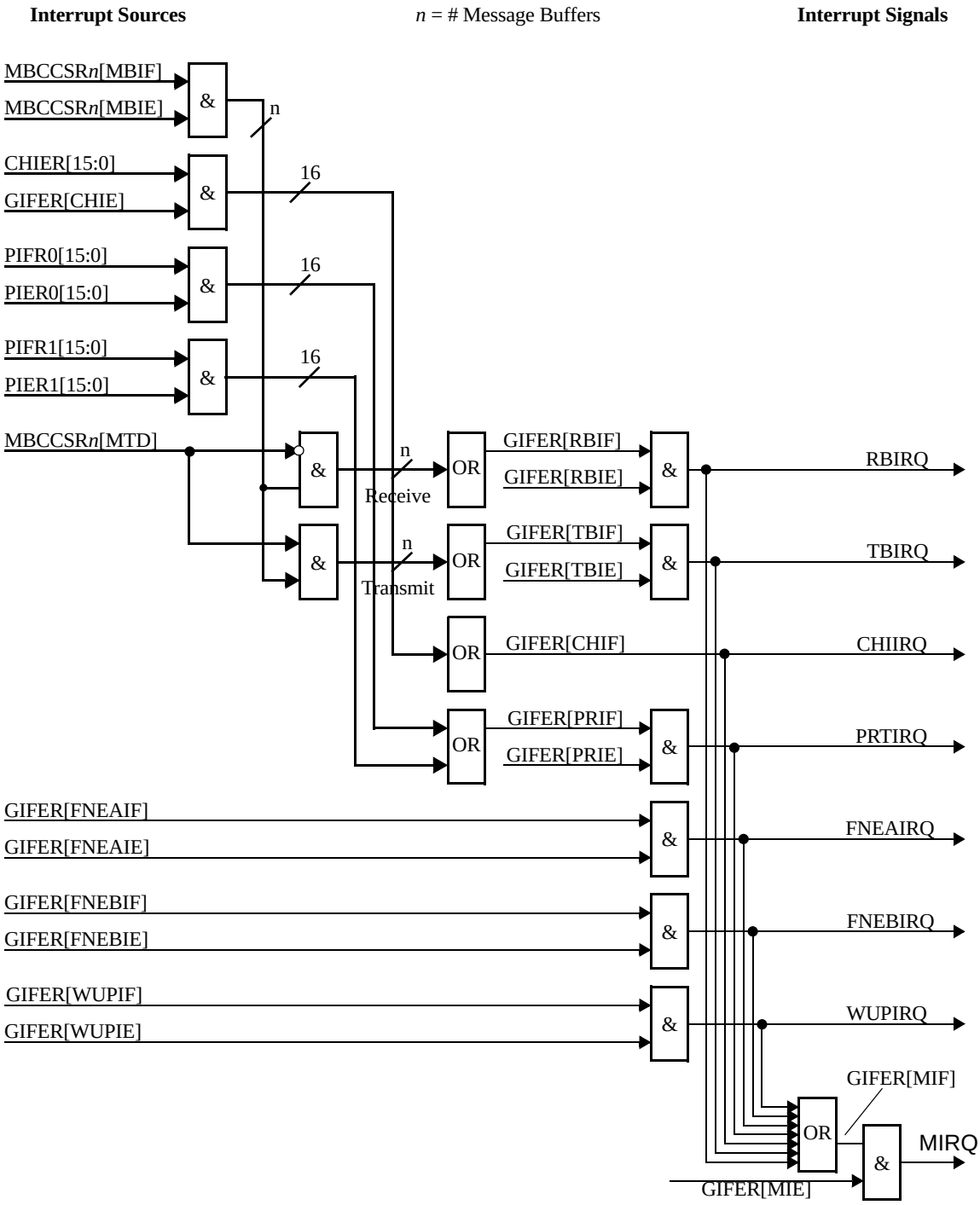


Figure 359. Scheme of cascaded interrupt request

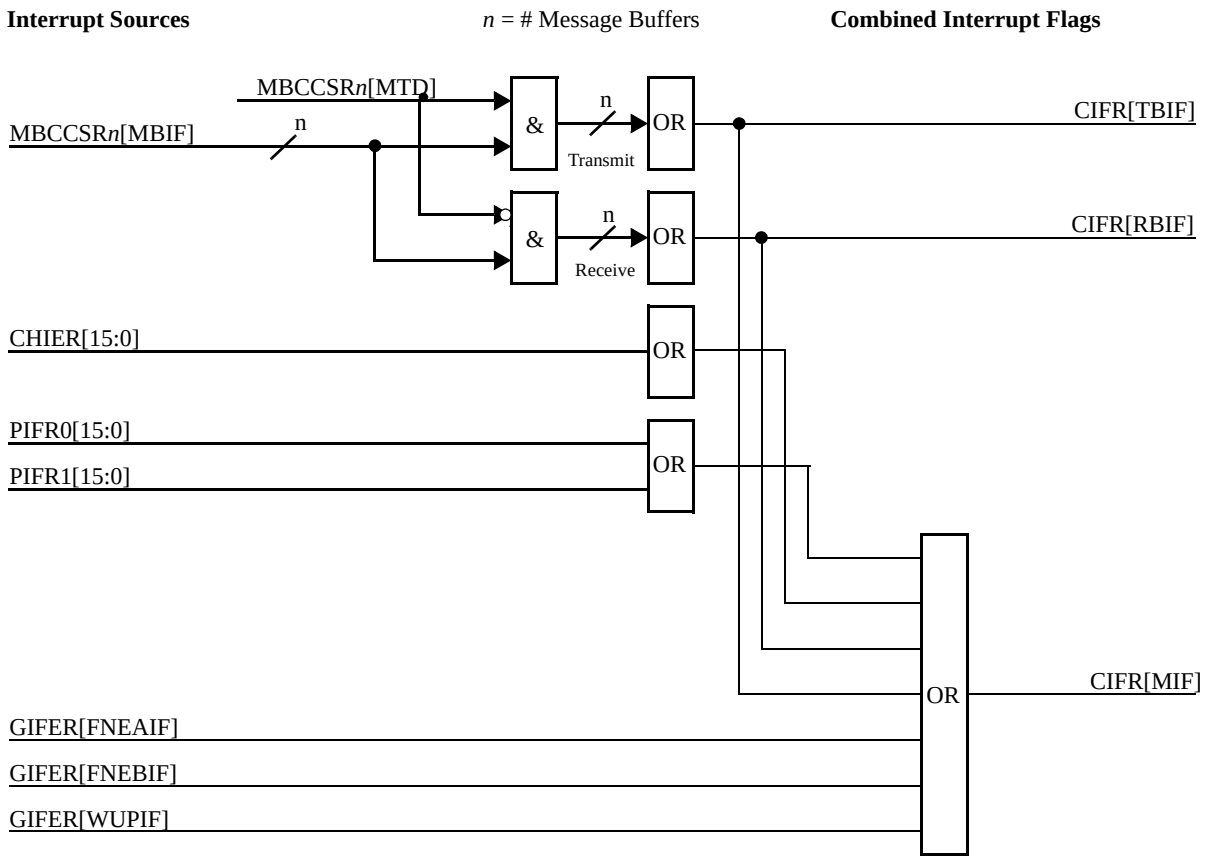


Figure 360. Scheme of combined interrupt flags

20.6.21 Lower bit rate support

The controller supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset.* The application configures the FlexRay channel bit rate by setting the BITRATE field in the Module Configuration Register (MCR). The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in *Table 328*.

Table 328. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	MCR.BITRATE	pdMicrotick [ns]	gdSampleClockPeriod [ns]	pSamplesPerMicrotick	cSamplesPerBit	cStrobeOffset
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

Note: The bit rate of 8 Mbit/s is not defined by the FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

20.7 Application information

20.7.1 Initialization sequence

This section describes the required steps to initialize the controller. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the controller.

20.7.1.1 Module initialization

This section describes the module related initialization steps after a system reset.

1. Configure controller.
   a) configure the control bits in the *Section 20.5.2.4: Module Configuration Register (MCR)*
   b) configure the system memory base address in *Section 20.5.2.5: System Memory Base Address High Register (SYMBADHR) and System Memory Base Address Low Register (SYMBADLR)*
2. Enable the controller.
   a) write 1 to the module enable bit MEN in the *Section 20.5.2.4: Module Configuration Register (MCR)*

The controller now enters the Normal Mode. The application can commence with the protocol initialization described in *Section 20.7.1.2: Protocol initialization*.

20.7.1.2 Protocol initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
   a) issue CONFIG command via *Section 20.5.2.9: Protocol Operation Control Register (POCR)*
   b) wait for *POC:config* in *Section 20.5.2.19: Protocol Status Register 0 (PSR0)*
   c) configure the PCR0,..., PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
   a) set the number of message buffers used and the message buffer segmentation in the *Section 20.5.2.8: Message Buffer Segment Size and Utilization Register (MBSSUTR)*
   b) define the message buffer data size in the *Section 20.5.2.7: Message Buffer Data Size Register (MBDSR)*
   c) configure each message buffer by setting the configuration values in the *Section 20.5.2.65: Message Buffer Configuration, Control, Status Registers (MBCCSRn)*, *Section 20.5.2.66: Message Buffer Cycle Counter Filter Registers (MBCCFRn)*, *Section 20.5.2.67: Message Buffer Frame ID Registers (MBFIDRn)*, *Section 20.5.2.68: Message Buffer Index Registers (MBIDXRn)*
   d) configure the receive FIFOs
   e) issue CONFIG_COMPLETE command via *Section 20.5.2.9: Protocol Operation Control Register (POCR)*
   f) wait for *POC:ready* in *Section 20.5.2.19: Protocol Status Register 0 (PSR0)*

After this sequence, the controller is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

20.7.2 Shut down sequence

This section describes a secure shut down sequence to stop the controller gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) repeatedly write 1 to MBCCSRn[EDT] until MBCCSRn[EDS] == 0.
2. Stop Protocol Engine.
   a) issue HALT command via *Section 20.5.2.9: Protocol Operation Control Register (POCR)*
   b) wait for *POC:halt* in *Section 20.5.2.19: Protocol Status Register 0 (PSR0)*

20.7.3 Number of usable message buffers

This section describes the relationship between the number of message buffers that can be utilized and the required minimum CHI clock frequency. Additional constraints for the minimum CHI clock frequency are given in *Section 20.3: Controller host interface clocking*.

The controller uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search must be finished within one FlexRay slot. The shortest FlexRay slot is an empty dynamic slot. An empty dynamic slot is a minislot and consists of *gdMinislot* macroticks with a nominal duration of *gdMacrotick*. The

minimum duration of a corrected macrotick is $gdMacrotick_{min}$ = 39 µT. This results in a minimum slot length of

Equation 43

$$\Delta_{slotmin} = 39 \cdot pdMicrotick \cdot gdMinislot$$

The search engine is located in the CHI and runs on the CHI clock. It evaluates one individual message buffer per CHI clock cycle. For internal status update and double buffer commit operations, and as a result of the clock domain crossing jitter, an additional amount of 10 CHI clock cycles is required to ensure correct operation. For a given number of message buffers and for a given CHI clock frequency f_{chi}, this results in a search duration of

Equation 44

$$\Delta_{search} = \frac{1}{f_{chi}} \cdot (\#\ \text{MessageBuffers} + 10)$$

The message buffer search must be finished within one slot which requires that *Equation 45* must be fulfilled

Equation 45

$$\Delta_{search} \leq \Delta_{slotmin}$$

This results in the formula given in *Equation 46* which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

Equation 46

$$f_{chi} \geq \frac{\#\ \text{MessageBuffers} + 10}{39 \cdot pdMicrotick \cdot gdMinislot}$$

The minimum CHI frequency for a selected set of relevant protocol parameters is given in *Table 329*.

Table 329. Minimum f_{chi} [MHz] examples (32 message buffers)

pdMicrotick [ns]	gdMinislot					
	2	3	4	5	6	7
25.0	21.54	14.35	10.77	8.61	7.18	6.16
50.0	10.73	7.18	5.39	4.31	3.59	3.08

20.7.4 Protocol control command execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of *Table 229* by writing the command to the POCCMD field of the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*. As a result the controller sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in *Table 330*. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the controller asserts the illegal protocol command interrupt flag IPC_IF in the *Section 20.5.2.12: Protocol Interrupt Flag Register 1 (PIFR1)*. The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by *Table 330*. If the application issues the RESET, FREEZE, or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 330. Protocol control command priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
RESET	(highest) 1	none	
FREEZE	2		RESET
READY	3		RESET
CONFIG_COMPLETE	3		RESET
ALL_SLOTS	4	RESET, FREEZE, READY, CONFIG_COMPLET, fatal protocol error	RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		RESET
RUN	6		RESET, FREEZE, fatal protocol error
WAKEUP	7		RESET, FREEZE, fatal protocol error
DEFAULT_CONFIG	8		RESET, FREEZE, fatal protocol error
CONFIG	9		RESET
HALT	(lowest) 10		RESET, FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

20.7.5 Protocol reset command

The application can use the RESET command to initiate a reset of the protocol engine

The application issues the protocol reset command by writing the RESET command to the POCCMD field of the *Section 20.5.2.9: Protocol Operation Control Register (POCR)*.

As a result, the PE stops its operation within the next 500 ns. At the same time the FlexRay bus ports are put into the reset state given in *Table 215*. The protocol and message buffer configuration data are not affected by the RESET command and will not be changed.

The duration of the RESET command must be at least 500 ns.

Before accessing the message buffers, the application must put the protocol into the *POC:default config* state by sending the Protocol Command DEFAULT_CONFIG via *Section 20.5.2.9: Protocol Operation Control Register (POCR)*.

20.7.6 Message buffer search on simple message buffer configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The receive FIFO behavior is not considered in this section.

20.7.6.1 Simple message buffer configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has following configuration

Table 331. Transmit buffer configuration

Register	Field	Value	Description
MBCCSR*t*	MCM	—	used only for double buffers
	MBT	0	single transmit buffer
	MTD	1	transmit buffer
MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
MBFIDR*t*	FID	S	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit MBCCSRt[CMT] and the lock bit MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and has following configuration

Table 332. Receive buffer configuration

Register	Field	Value	Description
MBCCSR*r*	MCM	—	n/a
	MBT	—	n/a
	MTD	0	receive buffer
MBCCF*r*	MTM	—	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
MBFIDR*r*	FID	S	subscribed slot

Furthermore the assumption is that both message buffers are enabled (MBCCSR*t*[EDS] = 1 and MBCCSR*r*[EDS] = 1)

Note: The cycle set {4n+2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

20.7.6.2 Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S.* In any case, the result of the message buffer search will be the transmit message buffer *t.* The receive message buffer *r* will not be found, no reception is possible.

20.7.6.3 Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

20.7.6.3.1 Transmit data not available

If transmit data are *not available*, i.e., the transmit buffer is not committed MBCCSR*t*[CMT] = 0 and/or locked MBCCSR*t*[LCKS] = 1,

a) for the cycles in the set {4n}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in *Figure 361*.

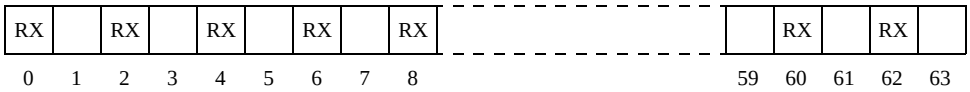


Figure 361. Transmit data not available

20.7.6.3.2 Transmit data available

If transmit data are *available,* i.e., the transmit buffer is committed MBCCSR*t*[CMT] = 1 and not locked MBCCSR*t*[LCKS] = 0,

a) for the cycles in the set {4*n*}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4*n* + 2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in *Figure 362*.

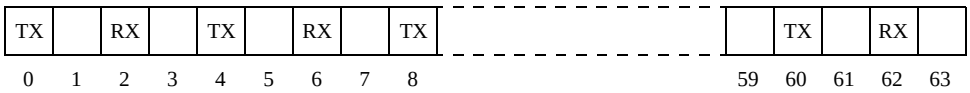


Figure 362. Transmit data not available

21 Deserial Serial Peripheral Interface (DSPI)

21.1 Introduction

This chapter describes the deserial serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device.

The SPC560P44Lx, SPC560P50Lx implements the modules DSPI0 to 3. The “x” appended to signal names signifies the module to which the signal applies. Thus CS0_x specifies that the CS0 signal applies to DSPI module 0, 1, etc.

21.2 Block diagram

A block diagram of the DSPI is shown in *Figure 363*.

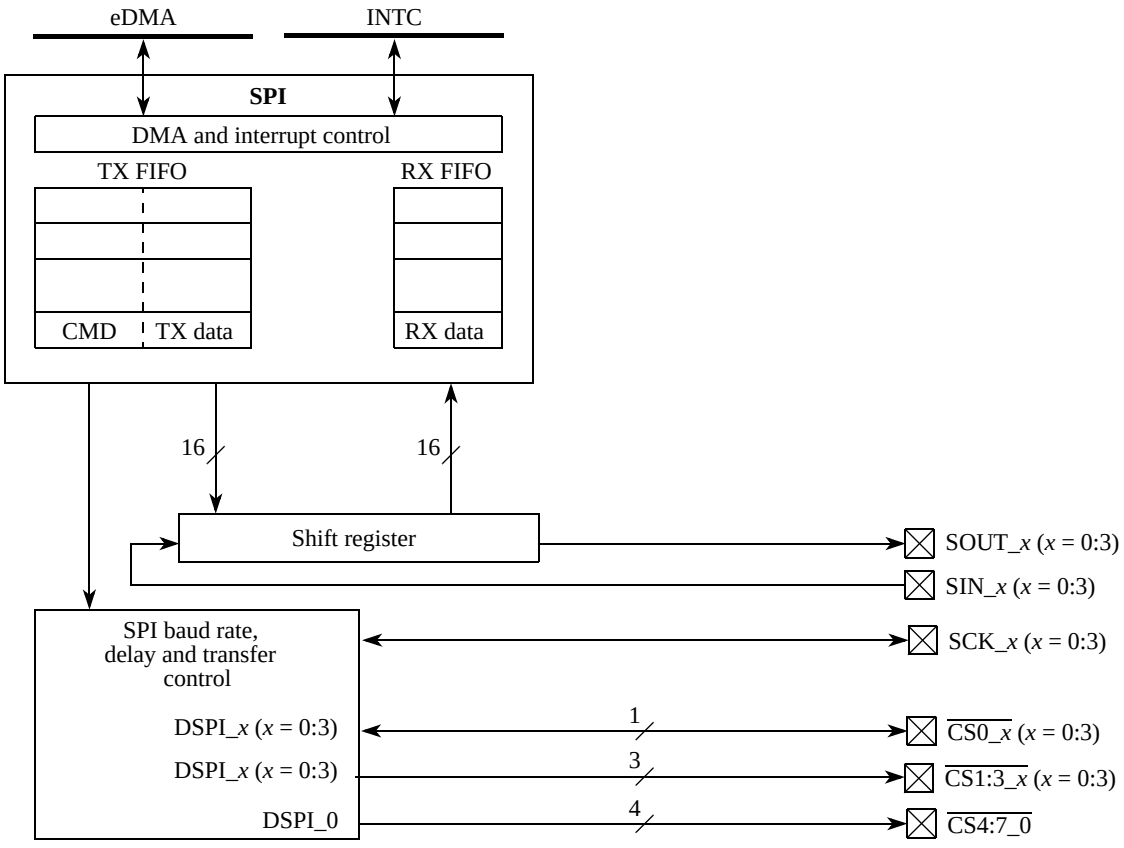


Figure 363. DSPI block diagram

21.3 Overview

The register content is transmitted using an SPI protocol. There are four DSPI modules (DSPI_0, DSPI_1, DSPI_2, and DSPI_3) on the device. The modules are identical except that DSPI_0 has four additional chip select ($\overline{CS}$) lines.

For queued operations, the SPI queues reside in internal SRAM that is external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

Figure 364 shows a DSPI with external queues in internal SRAM.

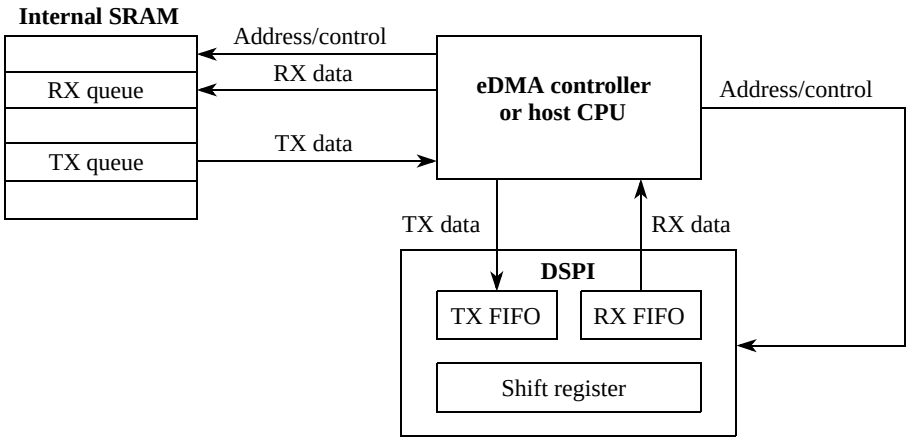


Figure 364. DSPI with queues and eDMA

21.4 Features

The DSPI supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave modes
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 5 entries
- Visibility into TX and RX FIFOs for ease of debugging
- FIFO bypass mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis
  - 8 clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    CS to SCK delay
    SCK to CS delay
    Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- 8 peripheral chip selects, expandable up to 256 with external demultiplexer
- Deglitching support for as many as 32 peripheral chip selects with external demultiplexer
- 2 DMA conditions for SPI queues residing in RAM or flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - RX FIFO is not empty (RFDF)
  - FIFO overrun (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)
  - FIFO under flow (slave only and SPI mode, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (SCK)

21.5 Modes of operation

The DSPI has four modes of operation. These modes can be divided into two categories; module-specific modes such as master, slave, and module disable modes, and a second category that is an MCU-specific mode: debug mode. All four modes are implemented on this device.

The module-specific modes are entered by host software writing to a register. The MCU-specific mode is controlled by signals external to the DSPI. The MCU-specific mode is a

mode that the entire device may enter, in parallel to the DSPI being in one of its module-specific modes.

21.5.1 Master mode

Master mode allows the DSPI to initiate and control serial communication. In this mode, the SCK, CS*n* and SOUT signals are controlled by the DSPI and configured as outputs.

For more information, refer to *Section 21.8.1.1: Master mode*.

21.5.2 Slave mode

Slave mode allows the DSPI to communicate with SPI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in slave mode. In slave mode, the SCK signal and the CS0_*x* signal are configured as inputs and provided by a bus master. CS0_*x* must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

For more information, refer to *Section 21.8.1.2: Slave mode*.

21.5.3 Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

For more information, refer to *Section 21.8.1.3: Module disable mode*.

21.5.4 Debug mode

Debug mode is used for system development and debugging. If the device enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected.

For more information, refer to *Section 21.8.1.4: Debug mode*.

21.6 External signal description

21.6.1 Signal overview

Table 333 lists off-chip DSPI signals.

Table 333. Signal properties

Name	I/O type	Function	
		Master mode	Slave mode
CS0_*x*	Output / input	Peripheral chip select 0	Slave select
CS1:3_*x* (DSPI 0: CS1:3_0, CS5_0)	Output	Peripheral chip select 1–3	Unused(1)
CS4_x	Output	Peripheral chip select 4	Master trigger
CS5_*x* (DSPI 0: CS7_0)	Output	Peripheral chip select 5 / Peripheral chip select strobe	Unused(1)
SIN_*x*	Input	Serial data in	Serial data in
SOUT_*x*	Output	Serial data out	Serial data out
SCK_*x*	Output / input	Serial clock (output)	Serial clock (input)

1. The SIUL allows you to select alternate pin functions for the device.

21.6.2 Signal names and descriptions

21.6.2.1 Peripheral Chip Select / Slave Select (CS_0)

In master mode, the CS_0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended.

In slave mode, the CS_0 signal is a slave select input signal that allows an SPI master to select the DSPI as the target for transmission. CS_0 must be configured as input and pulled high. If the internal pullup is being used then the appropriate bits in the relevant SIU_PCR must be set (SIU_PCR [WPE = 1], [WPS = 1]).

Set the IBE and OBE bits in the PCR register for all CS_0 pins when the DSPI chip select or slave select primary function is selected for that pin. When the pin is used for DSPI master mode as a chip select output, set the OBE bit. When the pin is used in DSPI slave mode as a slave select input, set the IBE bit.

Refer to *Section 11.5.2.8: Pad Configuration Registers (PCR[0:107])* for more information.

21.6.2.2 Peripheral Chip Selects 1–3 ($\overline{\text{CS1:3}}$)

$\overline{\text{CS1:3}}$ are peripheral chip select output signals in master mode. In slave mode these signals are not used. On DSPI_0, these are $\overline{\text{CS1:3}}$ and $\overline{\text{CS5:6}}$.

21.6.2.3 Peripheral Chip Select 4 ($\overline{\text{CS4}}$)

$\overline{\text{CS4}}$ is a peripheral chip select output signal in master mode. In slave mode this signal is not used.

21.6.2.4 Peripheral Chip Select 5/Peripheral Chip Select Strobe ($\overline{\text{CS\_5}}$)

$\overline{\text{CS5}}$ is a peripheral chip select output signal. When the DSPI is in master mode and PCSSE bit in the DSPI*x*_MCR is cleared, the $\overline{\text{CS5}}$ signal selects the slave device that receives the current transfer.

$\overline{CS5}$ is a strobe signal used by external logic for deglitching of the CS signals. When the DSPI is in master mode and the PCSSE bit in the DSPI*x*_MCR is set, the $\overline{CS5}$ signal indicates the timing to decode $\overline{CS0}$:4 signals, which prevents glitches from occurring.

$\overline{CS5}$ is not used in slave mode. On DSPI_0, this is $\overline{CS7}$.

21.6.2.5 Serial Input (SIN_*x*)

SIN_*x* is a serial data input signal.

21.6.2.6 Serial Output (SOUT_*x*)

SOUT_*x* is a serial data output signal.

21.6.2.7 Serial Clock (SCK_*x*)

SCK_*x* is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK_*x* is an input from an external bus master.

21.7 Memory map and registers description

21.7.1 Memory map

Table 334 shows the DSPI memory map.

Table 334. DSPI memory map

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3)	Register	Location
0x0000	DSPI_MCR—DSPI module configuration register	*on page 622*
0x0004	Reserved	
0x0008	DSPI_TCR—DSPI transfer count register	*on page 625*
0x000C	DSPI_CTAR0—DSPI clock and transfer attributes register 0	*on page 626*
0x0010	DSPI_CTAR1—DSPI clock and transfer attributes register 1	*on page 626*
0x0014	DSPI_CTAR2—DSPI clock and transfer attributes register 2	*on page 626*
0x0018	DSPI_CTAR3—DSPI clock and transfer attributes register 3	*on page 626*
0x001C	DSPI_CTAR4—DSPI clock and transfer attributes register 4	*on page 626*
0x0020	DSPI_CTAR5—DSPI clock and transfer attributes register 5	*on page 626*
0x0024	DSPI_CTAR6—DSPI clock and transfer attributes register 6	*on page 626*
0x0028	DSPI_CTAR7—DSPI clock and transfer attributes register 7	*on page 626*
0x002C	DSPI_SR—DSPI status register	*on page 632*
0x0030	DSPI_RSER—DSPI DMA/interrupt request select and enable register	*on page 634*

Table 334. DSPI memory map(Continued)

Offset from DSPI_BASE 0xFFF9_0000 (DSPI_0) 0xFFF9_4000 (DSPI_1) 0xFFF9_8000 (DSPI_2) 0xFFF9_C000 (DSPI_3)	Register	Location
0x0034	DSPI_PUSHR—DSPI push TX FIFO register	*on page 636*
0x0038	DSPI_POPR—DSPI pop RX FIFO register	*on page 637*
0x003C	DSPI_TXFR0—DSPI transmit FIFO register 0	*on page 638*
0x0040	DSPI_TXFR1—DSPI transmit FIFO register 1	*on page 638*
0x0044	DSPI_TXFR2—DSPI transmit FIFO register 2	*on page 638*
0x0048	DSPI_TXFR3—DSPI transmit FIFO register 3	*on page 638*
0x004C	DSPI_TXFR4—DSPI transmit FIFO register 4	*on page 638*
0x0050–0x007B	Reserved	
0x007C	DSPI_RXFR0—DSPI receive FIFO register 0	*on page 639*
0x0080	DSPI_RXFR1—DSPI receive FIFO register 1	*on page 639*
0x0084	DSPI_RXFR2—DSPI receive FIFO register 2	*on page 639*
0x0088	DSPI_RXFR3—DSPI receive FIFO register 3	*on page 639*
0x008C	DSPI_RXFR4—DSPI receive FIFO register 4	*on page 639*
0x0090–0x3FFF	Reserved	

21.7.2 Registers description

21.7.2.1 DSPI Module Configuration Register (DSPI*x*_MCR)

The DSPI*x*_MCR contains bits that configure attributes of the DSPI operation. The values of the HALT and MDIS bits can be changed at any time, but their effect begins on the next frame boundary. The HALT and MDIS bits in the DSPI*x*_MCR are the only bit values software can change while the DSPI is running.

Address: Base + 0x0000 Access: User read/write

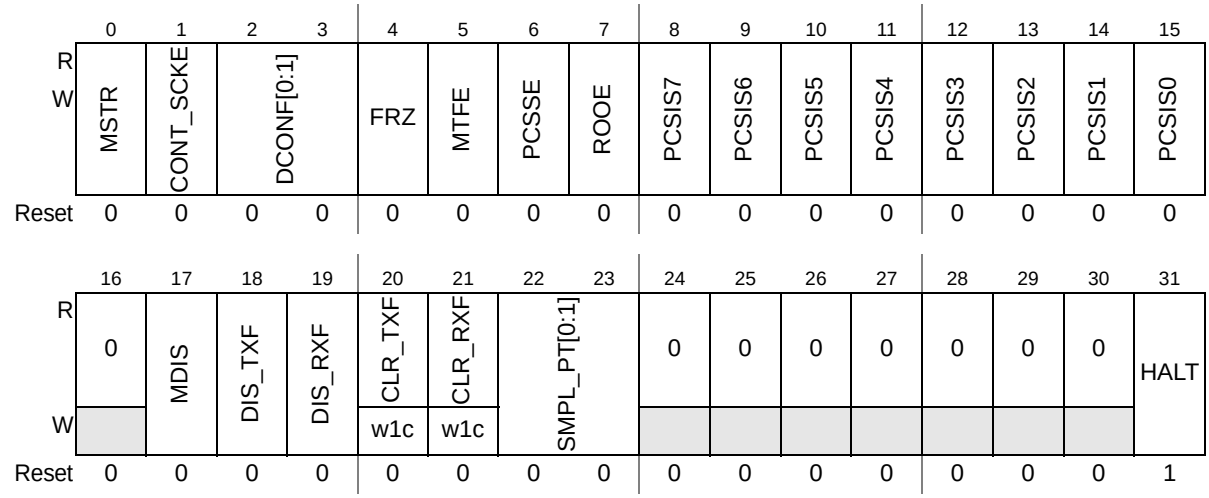


Figure 365. DSPI Module Configuration Register (DSPI*x*_MCR)

Table 335. DSPI*x*_MCR field descriptions

Field	Description
0 MSTR	Master/slave mode select Configures the DSPI for master mode or slave mode. 0 DSPI is in slave mode. 1 DSPI is in master mode.
1 CONT_SCKE	Continuous SCK enable Enables the serial communication clock (SCK) to run continuously. Refer to *Section 21.8.6: Continuous Serial communications clock* for details. 0 Continuous SCK disabled 1 Continuous SCK enabled **Note:** If the FIFO is enabled with continuous SCK mode, before setting the CONT_SCKE bit, the TX-FIFO should be cleared and only CTAR0 register should be used for transfer attributes otherwise a change in SCK frequency occurs.
2–3 DCONF [0:1]	DSPI configuration The following table lists the DCONF values for the various configurations. **DCONF** / **Configuration** 00 / SPI 01 / Invalid value 10 / Invalid value 11 / Invalid value
4 FRZ	Freeze Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers. 1 Halt serial transfers.

Table 335. DSPI*x*_MCR field descriptions(Continued)

Field	Description
5 MTFE	Modified timing format enable Enables a modified transfer format to be used. Refer to *Section 21.8.5.4: Modified SPI transfer format (MTFE = 1, CPHA = 1)* for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
6 PCSSE	Peripheral chip select strobe enable Enables the $\overline{CS5\_x}$ to operate as a $\overline{CS}$ strobe output signal. Refer to *Section 21.8.4.5: Peripheral Chip Select strobe enable (CS5_x)* for more information. 0 $\overline{CS5\_x}$ is used as the Peripheral chip select 5 signal. 1 $\overline{CS5\_x}$ is used as an active-low $\overline{CS}$ strobe signal.
7 ROOE	Receive FIFO overflow overwrite enable Enables an RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored. Refer to *Section 21.8.7.6: Receive FIFO overflow interrupt request (RFOF)* for more information. 0 Incoming data is ignored. 1 Incoming data is put in the shift register.
8–9	Reserved, but implemented. These bits are writable, but have no effect.
10–15 PCSIS*n*	Peripheral chip select inactive state Determines the inactive state of the $\overline{CS0\_x}$ signal. $\overline{CS0\_x}$ must be configured as inactive high for slave mode operation. 0 The inactive state of $\overline{CS0\_x}$ is low. 1 The inactive state of $\overline{CS0\_x}$ is high. **Note:** PCSIS7 and PSCIS6 are implemented only on DSPI_0.
16	Reserved
17 MDIS	Module disable Allows the clock to stop to the non-memory mapped logic in the DSPI, effectively putting the DSPI in a software controlled power-saving state. Refer to *Section 21.8.8: Power saving features* for more information." The reset value of the MDIS bit is parameterized, with a default reset value of 0. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
18 DIS_TXF	Disable transmit FIFO Enables and disables the TX FIFO. When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section 21.8.3.3: FIFO disable operation* for details. 0 TX FIFO enabled 1 TX FIFO disabled
19 DIS_RXF	Disable receive FIFO Enables and disables the RX FIFO. When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. Refer to *Section 21.8.3.3: FIFO disable operation* for details. 0 RX FIFO enabled 1 RX FIFO disabled

Table 335. DSPI*x*_MCR field descriptions(Continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>20
CLR_TXF</td><td>Clear TX FIFO
Flushes the TX FIFO. Write a 1 to the CLR_TXF bit to clear the TX FIFO counter. The CLR_TXF bit is always read as 0.
0 Do not clear the TX FIFO counter.
1 Clear the TX FIFO counter.</td></tr>
<tr><td>21
CLR_RXF</td><td>Clear RX FIFO
Flushes the RX FIFO. Write a 1 to the CLR_RXF bit to clear the RX counter. The CLR_RXF bit is always read as 0.
0 Do not clear the RX FIFO counter.
1 Clear the RX FIFO counter.</td></tr>
<tr><td>22–23
SMPL_PT
[0:1]</td><td>Sample point
Allows the host software to select when the DSPI master samples SIN in modified transfer format. Figure 380 shows where the master can sample the SIN pin. The following table lists the delayed sample points.
<table>
<tr><th>SMPL_PT</th><th>Number of system clock cycles between odd-numbered edge of SCK_x and sampling of SIN_x</th></tr>
<tr><td>00</td><td>0</td></tr>
<tr><td>01</td><td>1</td></tr>
<tr><td>10</td><td>2</td></tr>
<tr><td>11</td><td>Invalid value</td></tr>
</table>
</td></tr>
<tr><td>24–30</td><td>Reserved</td></tr>
<tr><td>31
HALT</td><td>Halt
Provides a mechanism for software to start and stop DSPI transfers. Refer to Section 21.8.2: Start and stop of DSPI transfers for details on the operation of this bit.
0 Start transfers.
1 Stop transfers.</td></tr>
</table>

21.7.2.2 DSPI Transfer Count Register (DSPI*x*_TCR)

The DSPI*x*_TCR contains a counter that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. The user must not write to the DSPI*x*_TCR while the DSPI is running.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SPI_TCNT[0:15]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 366. DSPI Transfer Count Register (DSPIx_TCR)

Table 336. DSPI*x*_TCR field descriptions

Field	Description
0–15 SPI_TCNT [0:15]	SPI transfer counter Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field is incremented every time the last bit of an SPI frame is transmitted. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter 'wraps around,' incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved

21.7.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTAR*n*)

The DSPI modules each contain eight clock and transfer attribute registers (DSPI*x*_CTAR*n*) that define different transfer attribute configurations. Each DSPI*x*_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB or LSB first

DSPI*x*_CTARs support compatibility with the QSPI module used in certain members of the SPC56x family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI*x*_CTAR that contains the transfer's attributes. Do not write to the DSPI*x*_CTARs while the DSPI is running.

In master mode, the DSPI*x*_CTAR*n* registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit fields in the DSPI*x*_CTAR0 and DSPI*x*_CTAR1 registers sets the slave transfer attributes. Refer to the individual bit descriptions for details on which bits are used in slave modes.

When the DSPI is configured as an SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI*x*_CTAR registers is used on a per-frame basis. When the DSPI is configured as an SPI bus slave, the DSPI*x*_CTAR0 register is used.

Address: Base + 0x000C (DSPI*x*_CTAR0)
Base + 0x0010 (DSPI*x*_CTAR1)
Base + 0x0014 (DSPI*x*_CTAR2)
Base + 0x0018 (DSPI*x*_CTAR3)
Base + 0x001C (DSPIx_CTAR4)
Base + 0x0020 (DSPI*x*_CTAR5)
Base + 0x0024 (DSPI*x*_CTAR6)
Base + 0x0028 (DSPI*x*_CTAR7)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSB FE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 367. DSPI Clock and Transfer Attributes Registers 0–7 (DSPI*x*_CTARn)

Table 337. DSPI*x*_CTARn field descriptions

Field	Descriptions
0 DBR	Double Baud Rate The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in Master Mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in *Table 338*. See the BR[0:3] field description for details on how to compute the baud rate. If the overall baud rate is divide by two or divide by three of the system clock then neither the Continuous SCK Enable or the Modified Timing Format Enable bits should be set. 0 The baud rate is computed normally with a 50/50 duty cycle. 1 The baud rate is doubled with the duty cycle depending on the baud rate prescaler.
1–4 FMSZ[0:3]	Frame Size The FMSZ field selects the number of bits transferred per frame. The FMSZ field is used in Master Mode and Slave Mode. *Table 339* lists the frame size encodings.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous Selection Format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low. 1 The inactive state value of SCK is high.
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both Master and Slave Mode. For successful communication between serial devices, the devices must have identical clock phase settings. Continuous SCK is only supported for CPHA = 1. 0 Data is captured on the leading edge of SCK and changed on the following edge. 1 Data is changed on the leading edge of SCK and captured on the following edge.

Table 337. DSPI*x*_CTARn field descriptions(Continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>7
LSBFE</td><td>LSB First
The LSBFE bit selects if the LSB or MSB of the frame is transferred first. This bit is only used in Master Mode.
0 Data is transferred MSB first.
1 Data is transferred LSB first.</td></tr>
<tr><td>8–9
PCSSCK[0:1]</td><td>PCS to SCK Delay Prescaler
The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. This field is only used in Master Mode. The table lists the prescaler values. See the CSSCK[0:3] field description for details on how to compute the PCS to SCK delay.
<table>
<tr><th>PCSSCK</th><th>PCS to SCK delay prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>10–11
PASC[0:1]</td><td>After SCK Delay Prescaler
The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. This field is only used in Master Mode. The table lists the prescaler values. See the ASC[0:3] field description for details on how to compute the After SCK delay.
<table>
<tr><th>PASC</th><th>After SCK delay prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>12–13
PDT[0:1]</td><td>Delay after Transfer Prescaler
The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in Master Mode. The table lists the prescaler values. See the DT[0:3] field description for details on how to compute the delay after transfer.
<table>
<tr><th>PDT</th><th>Delay after transfer prescaler value</th></tr>
<tr><td>00</td><td>1</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
</table>

Table 337. DSPI*x*_CTARn field descriptions(Continued)

<table>
<tr><th>Field</th><th>Descriptions</th></tr>
<tr><td>14–15
PBR[0:1]</td><td>Baud Rate Prescale
The PBR field selects the prescaler value for the baud rate. This field is only used in Master Mode. The baud rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The baud rate prescaler values are listed in the table. See the BR[0:3] field description for details on how to compute the baud rate.
<table>
<tr><th>PBR</th><th>Baud rate prescaler value</th></tr>
<tr><td>00</td><td>2</td></tr>
<tr><td>01</td><td>3</td></tr>
<tr><td>10</td><td>5</td></tr>
<tr><td>11</td><td>7</td></tr>
</table></td></tr>
<tr><td>16–19
CSSCK[0:3]</td><td>PCS to SCK Delay Scaler
The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in Master Mode. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. Table 340 lists the scaler values. The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation:
Equation 47
$$t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$$
See Section 21.8.4.2: CS to SCK delay (tCSC) for more details.</td></tr>
<tr><td>20–23
ASC[0:3]</td><td>After SCK Delay Scaler
The ASC field selects the scaler value for the After SCK delay. This field is only used in Master Mode. The After SCK delay is the delay between the last edge of SCK and the negation of PCS. Table 341 lists the scaler values. The After SCK delay is a multiple of the system clock period, and it is computed according to the following equation:
Equation 48
$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$
See Section 21.8.4.3: After SCK delay (tASC) for more details.</td></tr>
</table>

Table 337. DSPI*x*_CTARn field descriptions(Continued)

Field	Descriptions
24–27 DT[0:3]	Delay after Transfer Scaler The DT field selects the delay after transfer scaler. This field is only used in Master Mode. The delay after transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. *Table 342* lists the scaler values. In continuous serial communications clock operation, the DT value is fixed to one TSCK,. The delay after transfer is a multiple of the system clock period and it is computed according to the following equation: **Equation 49** $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ See *Section 21.8.4.4: Delay after transfer (tDT)* for more details.
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in Master Mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the SCK. *Table 343* lists the baud rate scaler values. The baud rate is computed according to the following equation: **Equation 50** $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ See *Section 21.8.4.1: Baud rate generator* for more details.

Table 338. DSPI SCK duty cycle

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 339. DSPI transfer frame size

FMSZ	Frame size	FMSZ	Frame size
0000	Reserved	1000	9
0001	Reserved	1001	10
0010	Reserved	1010	11
0011	4	1011	12
0100	5	1100	13
0101	6	1101	14
0110	7	1110	15
0111	8	1111	16

Table 340. DSPI PCS to SCK delay scaler

CSSCK	PCS to SCK delay scaler value	CSSCK	PCS to sck delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 341. DSPI after SCK delay scaler

ASC	After SCK delay scaler value	ASC	After SCK delay scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 342. DSPI delay after transfer scaler

DT	Delay after transfer scaler value	DT	Delay after transfer scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 343. DSPI baud rate scaler

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

21.7.2.4 DSPI Status Register (DSPI*x*_SR)

The DSPI*x*_SR contains status and flag bits. The bits are set by the hardware and reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI*x*_SR by writing a 1 to clear it (w1c). Writing a 0 to a flag bit has no effect.

Address: Base + 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	0	0	0	RFOF	0	RFDF	0
W	w1c			w1c	w1c		w1c						w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 368. DSPI Status Register (DSPI*x*_SR)

Table 344. DSPI*x*_SR field descriptions

Field	Description
0 TCF	Transfer complete flag Indicates that all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section 21.8.5.1: Classic SPI transfer format (CPHA = 0)* for details. The TCF bit is cleared by writing 1 to it. 0 Transfer not complete. 1 Transfer complete.
1 TXRXS	TX and RX status Reflects the status of the DSPI. Refer to *Section 21.8.2: Start and stop of DSPI transfers* for information on what clears and sets this bit. 0 TX and RX operations are disabled (DSPI is in STOPPED state). 1 TX and RX operations are enabled (DSPI is in RUNNING state).
2	Reserved
3 EOQF	End of queue flag Indicates that transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section 21.8.5.1: Classic SPI transfer format (CPHA = 0)* for details. The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executing command. 1 EOQ bit is set in the executing SPI command. **Note:** EOQF does not function in slave mode.
4 TFUF	Transmit FIFO underflow flag Indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred. 1 TX FIFO underflow has occurred.
5	Reserved

Table 344. DSPI*x*_SR field descriptions(Continued)

Field	Description
6 TFFF	Transmit FIFO fill flag Indicates that the TX FIFO can be filled. Provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it, or an by acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full. 1 TX FIFO is not full.
7–11	Reserved
12 RFOF	Receive FIFO overflow flag Indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred. 1 RX FIFO overflow has occurred.
13	Reserved
14 RFDF	Receive FIFO drain flag Indicates that the RX FIFO can be drained. Provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it, or by acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty. 1 RX FIFO is not empty. **Note:** In the interrupt service routine, RFDF must be cleared only after the DSPIx_POPR register is read.
15	Reserved
16–19 TXCTR [0:3]	TX FIFO counter Indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time an SPI command is executed and the SPI data is transferred to the shift register.
20–23 TXNXTPTR [0:3]	Transmit next pointer Indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. Refer to *Section 21.8.3.4: Transmit First In First Out (TX FIFO) buffering mechanism* for more details.
24–27 RXCTR [0:3]	RX FIFO counter Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the tASC delay starts. Refer to *Section 21.8.5.1: Classic SPI transfer format (CPHA = 0)* for details.
28–31 POPNXTPTR [0:3]	Pop next pointer Contains a pointer to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR is updated when the DSPI*x*_POPR is read. Refer to *Section 21.8.3.5: Receive First In First Out (RX FIFO) buffering mechanism* for more details.

21.7.2.5 DSPI DMA / Interrupt Request Select and Enable Register (DSPI*x*_RSER)

The DSPI*x*_RSER serves two purposes: enables flag bits in the DSPI*x*_SR to generate DMA requests or interrupt requests, and selects the type of request to generate. Refer to the bit descriptions for the type of requests that are supported. Do not write to the DSPI*x*_RSER while the DSPI is running.

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQF_RE	TFUF_RE	0	TFFF_RE	TFFF_DIRS	0	0	0	0	RFOF_RE	0	RFDF_RE	RFDF_DIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 369. DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)

Table 345. DSPI*x*_RSER field descriptions

Field	Description
0 TCF_RE	Transmission complete request enable Enables TCF flag in the DSPI*x*_SR to generate an interrupt request. 0 TCF interrupt requests are disabled. 1 TCF interrupt requests are enabled.
1–2	Reserved
3 EOQF_RE	DSPI finished request enable Enables the EOQF flag in the DSPI*x*_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled. 1 EOQF interrupt requests are enabled.
4 TFUF_RE	Transmit FIFO underflow request enable The TFUF_RE bit enables the TFUF flag in the DSPI*x*_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled. 1 TFUF interrupt requests are enabled.
5	Reserved
6 TFFF_RE	Transmit FIFO fill request enable Enables the TFFF flag in the DSPI*x*_SR to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled. 1 TFFF interrupt requests or DMA requests are enabled.
7 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI*x*_SR is set, and the TFFF_RE bit in the DSPI*x*_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
8–11	Reserved
12 RFOF_RE	Receive FIFO overflow request enable Enables the RFOF flag in the DSPI*x*_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled. 1 RFOF interrupt requests are enabled.
13	Reserved

Table 345. DSPI*x*_RSER field descriptions(Continued)

Field	Description
14 RFDF_RE	Receive FIFO drain request enable Enables the RFDF flag in the DSPI*x*_SR to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled. 1 RFDF interrupt requests or DMA requests are enabled.
15 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select Selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI*x*_SR is set, and the RFDF_RE bit in the DSPI*x*_RSER is set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request is selected. 1 DMA request is selected.
16–31	Reserved

21.7.2.6 DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

The DSPI*x*_PUSHR provides a means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. Refer to *Section 21.8.3.4: Transmit First In First Out (TX FIFO) buffering mechanism* for more information. Write accesses of 8 or 16 bits to the DSPI*x*_PUSHR transfer 32 bits to the TX FIFO.

Note: TXDATA is used in master and slave modes.

Address Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CONT	CTAS			EOQ	CTCNT	0	0	PCS7	PCS6	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 370. DSPI PUSH TX FIFO Register (DSPI*x*_PUSHR)

Table 346. DSPI*x*_PUSHR field descriptions

Field	Description
0 CONT	Continuous peripheral chip select enable Selects a continuous selection format. The bit is used in SPI master mode. The bit enables the selected $\overline{CS}$ signals to remain asserted between transfers. Refer to *Section 21.8.5.5: Continuous selection format* for more information. 0 Return peripheral chip select signals to their inactive state between transfers. 1 Keep peripheral chip select signals asserted between transfers.

Table 346. DSPI*x*_PUSHR field descriptions(Continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>1–3
CTAS
[0:2]</td><td>Clock and transfer attributes select
Selects which of the DSPIx_CTARs sets the transfer attributes for the SPI frame. In SPI slave mode, DSPIx_CTAR0 is used. The following table shows how the CTAS values map to the DSPIx_CTARs. There are eight DSPIx_CTARs in the device DSPI implementation.
Note: Use in SPI master mode only.

<table>
<tr><th>CTAS</th><th>Use Clock and Transfer Attributes from</th></tr>
<tr><td>000</td><td>DSPIx_CTAR0</td></tr>
<tr><td>001</td><td>DSPIx_CTAR1</td></tr>
<tr><td>010</td><td>DSPIx_CTAR2</td></tr>
<tr><td>011</td><td>DSPIx_CTAR3</td></tr>
<tr><td>100</td><td>DSPIx_CTAR4</td></tr>
<tr><td>101</td><td>DSPIx_CTAR5</td></tr>
<tr><td>110</td><td>DSPIx_CTAR6</td></tr>
<tr><td>111</td><td>DSPIx_CTAR7</td></tr>
</table>
</td></tr>
<tr><td>4
EOQ</td><td>End of queue
Provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPIx_SR is set.
0 The SPI data is not the last data to transfer.
1 The SPI data is the last data to transfer.
Note: Use in SPI master mode only.</td></tr>
<tr><td>5
CTCNT</td><td>Clear SPI_TCNT
Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the SPI_TCNT field in the DSPIx_TCR. The SPI_TCNT field is cleared before transmission of the current SPI frame begins.
0 Do not clear SPI_TCNT field in the DSPIx_TCR.
1 Clear SPI_TCNT field in the DSPIx_TCR.
Note: Use in SPI master mode only.</td></tr>
<tr><td>6–7</td><td>Reserved</td></tr>
<tr><td>10–15
PCSx</td><td>Peripheral chip select x
Selects which $\overline{\text{CS}x}$ signals are asserted for the transfer.
0 Negate the $\overline{\text{CS}x}$ signal.
1 Assert the $\overline{\text{CS}x}$ signal.
Note: Use in SPI master mode only.</td></tr>
<tr><td>16–31
TXDATA
[0:15]</td><td>Transmit data
Holds SPI data for transfer according to the associated SPI command.
Note: Use TXDATA in master and slave modes.</td></tr>
</table>

21.7.2.7 DSPI POP RX FIFO Register (DSPI*x*_POPR)

The DSPI*x*_POPR allows you to read the RX FIFO. Refer to *Section 21.8.3.5: Receive First In First Out (RX FIFO) buffering mechanism* for a description of the RX FIFO operations.

Eight or 16-bit read accesses to the DSPI*x*_POPR fetch the RX FIFO data, and update the counter and pointer.

Note: Reading the RX FIFO field fetches data from the RX FIFO. Once the RX FIFO is read, the read data pointer is moved to the next entry in the RX FIFO. Therefore, read DSPIx_POPR only when you need the data. For compatibility, configure the TLB (MMU table) entry for DSPIx_POPR as guarded.

Address: Base + 0x0038 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 371. DSPI POP RX FIFO Register (DSPI*x*_POPR)

Table 347. DSPI*x*_POPR field descriptions

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [0:15]	Received data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (POPNXTPTR).

21.7.2.8 DSPI Transmit FIFO Registers 0–4 (DSPI*x*_TXFR*n*)

The DSPI*x*_TXFR*n* registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI*x*_TXFR*n* registers does not alter the state of the TX FIFO. The MCU uses five registers to implement the TX FIFO, that is DSPI*x*_TXFR0–DSPI*x*_TXFR4 are used.

Address: Base + 0x003C (DSPI*x*_TXFR0) Base + 0x0048 (DSPI*x*_TXFR3) Access: User read-only
Base + 0x0040 (DSPI*x*_TXFR1) Base + 0x004C (DSPI*x*_TXFR4)
Base + 0x0044 (DSPI*x*_TXFR2)

	0	1	2	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD														
W															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 372. DSPI Transmit FIFO Register 0–4 (DSPI*x*_TXFR*n*)

Table 348. DSPI*x*_TXFR*n* field descriptions

Field	Description
0–15 TXCMD [0:15]	Transmit command Contains the command that sets the transfer attributes for the SPI data. Refer to *Section 21.7.2.6: DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the command field.
16–31 TXDATA [0:15]	Transmit data Contains the SPI data to be shifted out.

21.7.2.9 DSPI Receive FIFO Registers 0–4 (DSPI*x*_RXFR*n*)

The DSPI*x*_RXFR*n* registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI*x*_RXFR registers are read-only. Reading the DSPI*x*_RXFR*n* registers does not alter the state of the RX FIFO. The device uses five registers to implement the RX FIFO, that is DSPI*x*_RXFR0–DSPI*x*_RXFR4 are used.

Address: Base + 0x007C (DSPI*x*_RXFR0) Base + 0x0088 (DSPI*x*_RXFR3) Access: User read-only
Base + 0x0080 (DSPI*x*_RXFR1) Base + 0x008C (DSPI*x*_RXFR4)
Base + 0x0084 (DSPI*x*_RXFR2)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 373. DSPI Receive FIFO Registers 0–4 (DSPIx_RXFRn)

Table 349. DSPI*x*_RXFR*n* field description

Field	Description
0–15	Reserved, must be cleared.
16–31 RXDATA [15:0]	Receive data Contains the received SPI data.

21.8 Functional description

The DSPI supports full-duplex, synchronous serial communications between the MCU and peripheral devices. All communications are through an SPI-like protocol.

The DSPI supports only the serial peripheral interface (SPI) configuration in which the DSPI operates as a basic SPI or a queued SPI.

The DCONF field in the DSPI*x*_MCR register determines the DSPI configuration. Refer to *Table 335* for the DSPI configuration values.

The DSPI*x*_CTAR0–DSPI*x*_CTAR7 registers hold clock and transfer attributes.The SPI configuration can select which CTAR to use on a frame by frame basis by setting the CTAS field in the DSPI*x*_PUSHR.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT_*x* and SIN_*x* signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data is exchanged between the master and the slave; the data that was in the master's shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate a completed transfer. *Figure 374* illustrates how master and slave data is exchanged.

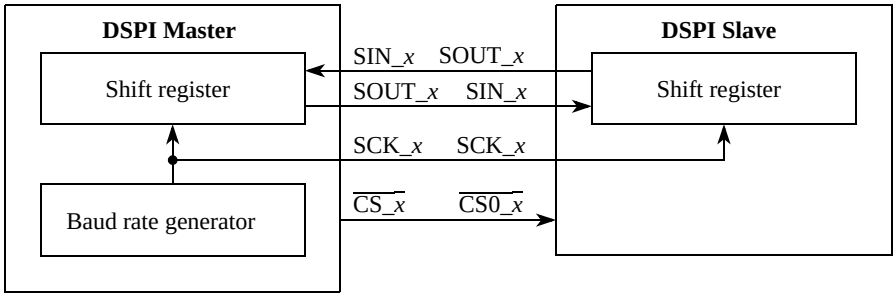


Figure 374. SPI serial protocol overview

Each DSPI has four peripheral chip select ($\overline{CSx}$) signals that select the slaves with which to communicate (DSPI_0 has eight $\overline{CSx}$ signals.)

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in *Section 21.8.5: Transfer formats*. The transfer rate and delay settings are described in *Section 21.8.4: DSPI baud rate and clock delay generation*.

Refer to *Section 21.8.8: Power saving features* for information on the power-saving features of the DSPI.

21.8.1 Modes of operation

The DSPI modules have four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode. All four modes are implemented on this device.

The module-specific modes are determined by bits in the DSPI*x*_MCR. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

21.8.1.1 Master mode

In master mode the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the MSTR bit in the DSPI*x*_MCR is set. The serial communications clock (SCK) is controlled by the master DSPI. All three DSPI configurations are valid in master mode.

In SPI configuration, master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI*x*_CTARs set the transfer attributes. Transfer attribute control is on a frame by frame basis.

Refer to *Section 21.8.3: Serial Peripheral Interface (SPI) configuration* for more details.

21.8.1.2 Slave mode

In slave mode the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the MSTR bit in the DSPI*x*_MCR is negated. The DSPI slave is selected by a bus master by having the slave's $\overline{CS0\_x}$ asserted. In slave mode, the SCK is provided by the bus master. All transfer attributes are controlled by the bus master, except the clock polarity, clock phase, and the number of bits to transfer. These must be configured in the DSPI slave for correct communications.

21.8.1.3 Module disable mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in module disable mode. The DSPI enters the module disable mode when the MDIS bit in DSPI*x*_MCR is set.

Refer to *Section 21.8.8: Power saving features* for more details on the module disable mode.

21.8.1.4 Debug mode

The debug mode is used for system development and debugging. If the MCU enters debug mode while the FRZ bit in the DSPI*x*_MCR is set, the DSPI stops all serial transfers and enters a stopped state. If the MCU enters debug mode while the FRZ bit is cleared, the

DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. The DSPI enters debug mode when a debug request is asserted by an external controller.

Refer to *Figure 375* for a state diagram.

21.8.2 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. The TXRXS bit in the DSPI*x*_SR is cleared in this state. In the RUNNING state, serial transfers take place. The TXRXS bit in the DSPI*x*_SR is set in the RUNNING state.

Figure 375 shows a state diagram of the start and stop mechanism.

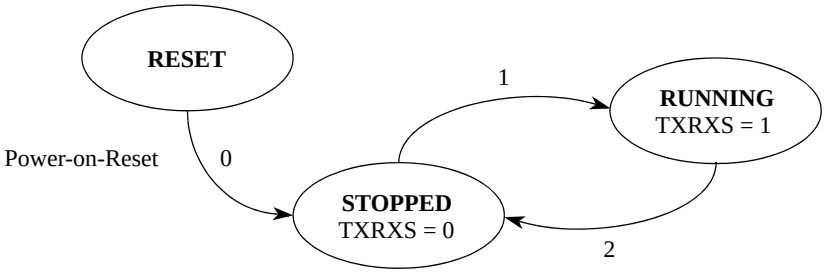


Figure 375. DSPI start and stop state diagram

The transitions are described in *Table 350*.

Table 350. State transitions for start and stop of DSPI transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI starts (transitions from STOPPED to RUNNING) when all of the following conditions are true: – EOQF bit is clear – Debug mode is unselected or the FRZ bit is clear – HALT bit is clear
2	RUNNING	STOPPED	The DSPI stops (transitions from RUNNING to STOPPED) after the current frame for any one of the following conditions: – EOQF bit is set – Debug mode is selected and the FRZ bit is set – HALT bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or on the next system clock cycle if no transfers are in progress.

21.8.3 Serial Peripheral Interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when the DCONF field in the DSPI*x*_MCR is 0b00. The SPI frames can be from 4 to 16 bits long. The data to be transmitted can come from queues stored in RAM external to the DSPI. Host software or an eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or an eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI.

The FIFO buffer operations are described in *Section 21.8.3.4: Transmit First In First Out (TX FIFO) buffering mechanism* and *Section 21.8.3.5: Receive First In First Out (RX FIFO) buffering mechanism*.

The interrupt and DMA request conditions are described in *Section 21.8.7: Interrupts/DMA requests*.

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for the master mode and slave mode. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field of the TX FIFO entry is ignored.

21.8.3.1 SPI master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the serial communications clock (SCK_*x*) and the peripheral chip select ($\overline{\text{CS}}$*x*) signals. The SPI command field in the executing TX FIFO entry determines which CTARs set the transfer attributes and which $\overline{\text{CS}}$*x* signals to assert. The command field also contains various bits that help with queue management and transfer protocol. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the serial out (SOUT_*x*) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

Refer to *Section 21.7.2.6: DSPI PUSH TX FIFO Register (DSPIx_PUSHR)* for details on the SPI command fields.

21.8.3.2 SPI slave mode

In SPI slave mode the DSPI responds to transfers initiated by an SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with an SPI master. The SPI slave mode transfer attributes are set in the DSPI*x*_CTAR0.

21.8.3.3 FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The TX and RX FIFOs are disabled separately. The TX FIFO is disabled by writing a 1 to the DIS_TXF bit in the DSPI*x*_MCR. The RX FIFO is disabled by writing a 1 to the DIS_RXF bit in the DSPI*x*_MCR.

The FIFO disable mechanisms are transparent to the user and to host software; transmit data and commands are written to the DSPI*x*_PUSHR and received data is read from the

DSPI*x*_POPR. When the TX FIFO is disabled, the TFFF, TFUF, and TXCTR fields in DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_TXFRs and TXNXTPTR are undefined. When the RX FIFO is disabled, the RFDF, RFOF, and RXCTR fields in the DSPI*x*_SR behave as if there is a one-entry FIFO but the contents of the DSPI*x*_RXFRs and POPNXTPTR are undefined.

Disable the TX and RX FIFOs only if the FIFO must be disabled as a requirement of the application's operating mode. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported, and can result in incorrect results.

21.8.3.4 Transmit First In First Out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds five entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI*x*_PUSHR). For more information on DSPI*x*_PUSHR refer to *Section 21.7.2.6: DSPI PUSH TX FIFO Register (DSPIx_PUSHR)*. TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO counter field (TXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

Refer to *Section 21.7.2.4: DSPI Status Register (DSPIx_SR)* for more information on DSPI*x*_SR.

The TXNXTPTR field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI*x*_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI*x*_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field is incremented every time SPI data is transferred from the TX FIFO to the shift register.

21.8.3.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI*x*_PUSHR. When the TX FIFO is not full, the TX FIFO fill flag (TFFF) in the DSPI*x*_SR is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI*x*_PUSHR is complete or alternatively by host software writing a 1 to the TFFF in the DSPI*x*_SR. The TFFF can generate a DMA request or an interrupt request.

Refer to *Section 21.8.7.2: Transmit FIFO fill interrupt or DMA request (TFFF)* for details.

The DSPI ignores attempts to push data to a full TX FIFO; that is, the state of the TX FIFO is unchanged. No error condition is indicated.

21.8.3.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO counter is decremented by one. At the end of a transfer, the TCF bit in the DSPI*x*_SR is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a 1 to the CLR_TXF bit in DSPI*x*_MCR.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave's DSPI TX FIFO is empty, the transmit FIFO underflow flag (TFUF) in the slave's DSPI*x*_SR is set.

Refer to *Section 21.8.7.4: Transmit FIFO underflow interrupt request (TFUF)* for details.

21.8.3.5 Receive First In First Out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds five received SPI data frames. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data is removed (popped) from the RX FIFO by reading the DSPI*x*_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI*x*_POPR or by flushing the RX FIFO.

Refer to *Section 21.7.2.7: DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on the DSPI*x*_POPR.

The RX FIFO counter field (RXCTR) in the DSPI status register (DSPI*x*_SR) indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

The POPNXTPTR field in the DSPI*x*_SR points to the RX FIFO entry that is returned when the DSPI*x*_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPI*x*_RXFR0. For example, POPNXTPTR equal to two means that the DSPI*x*_RXFR2 contains the received SPI data that is returned when DSPI*x*_POPR is read. The POPNXTPTR field is incremented every time the DSPI*x*_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

21.8.3.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time an SPI frame is transferred to the RX FIFO the RX FIFO counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI*x*_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is ignored or put in the shift register. If the ROOE bit is set, the incoming data is put in the shift register. If the ROOE bit is cleared, the incoming data is ignored.

21.8.3.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI*x*_POPR. A read of the DSPI*x*_POPR decrements the RX FIFO counter by one. Attempts to pop data from an empty RX FIFO are ignored, the RX FIFO counter remains unchanged. The data returned from reading an empty RX FIFO is undetermined.

Refer to *Section 21.7.2.7: DSPI POP RX FIFO Register (DSPIx_POPR)* for more information on DSPI*x*_POPR.

When the RX FIFO is not empty, the RX FIFO drain flag (RFDF) in the DSPI*x*_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the eDMA controller indicates that a read from DSPI*x*_POPR is complete; alternatively the RFDF bit can be cleared by the host writing a 1 to it.

21.8.4 DSPI baud rate and clock delay generation

The SCK_*x* frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option of doubling the baud rate.

Figure 376 shows conceptually how the SCK signal is generated.

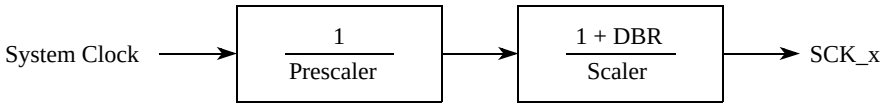


Figure 376. Communications clock prescalers and scalers

21.8.4.1 Baud rate generator

The baud rate is the frequency of the serial communication clock (SCK_*x*). The system clock is divided by a baud rate prescaler (defined by DSPI*x*_CTAR[PBR]) and baud rate scaler (defined by DSPI*x*_CTAR[BR]) to produce SCK_*x* with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI*x*_CTARs select the frequency of SCK_*x* using the following formula:

Equation 51

$$\text{SCK baud rate} = \frac{f_{SYS}}{\text{PBRPrescalerValue}} \times \frac{1 + \text{DBR}}{\text{BRScalerValue}}$$

Table 351 shows an example of a computed baud rate.

Table 351. Baud rate computation example

f_{SYS}	PBR	Prescaler value	BR	Scaler value	DBR value	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mbit/s
20 MHz	0b00	2	0b0000	2	1	10 Mbit/s

21.8.4.2 *CS* to SCK delay (t_{CSC})

The $\overline{\text{CS}}$_*x* to SCK_*x* delay is the length of time from assertion of the $\overline{\text{CS}}$_*x* signal to the first SCK_*x* edge. Refer to *Figure 378* for an illustration of the $\overline{\text{CS}}$_*x* to SCK_*x* delay. The PCSSCK and CSSCK fields in the DSPI*x*_CTAR*n* registers select the $\overline{\text{CS}}$_*x* to SCK_*x* delay, and the relationship is expressed by the following formula:

Equation 52

$$t_{CSC} = \frac{1}{f_{SYS}} \times \text{PCSSCK} \times \text{CSSCK}$$

Table 352 shows an example of the computed $\overline{\text{CS}}$ to SCK_*x* delay.

Table 352. $\overline{CS}$ to SCK delay computation example

PCSSCK	Prescaler value	CSSCK	Scaler value	f_{SYS}	$\overline{CS}$ to SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

21.8.4.3 After SCK delay (t_{ASC})

The after SCK_*x* delay is the length of time between the last edge of SCK_*x* and the deassertion of $\overline{CS}$_*x*. Refer to *Figure 378* and *Figure 379* for illustrations of the after SCK_*x* delay. The PASC and ASC fields in the DSPI*x*_CTAR*n* registers select the after SCK delay. The relationship between these variables is given in the following formula:

Equation 53

$$t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$$

Table 353 shows an example of the computed after SCK delay.

Table 353. After SCK delay computation example

PASC	Prescaler value	ASC	Scaler value	f_{SYS}	After SCK delay
0b01	3	0b0100	32	100 MHz	0.96 µs

21.8.4.4 Delay after transfer (t_{DT})

The delay after transfer is the length of time between negation of the $\overline{CSx}$ signal for a frame and the assertion of the $\overline{CSx}$ signal for the next frame. The PDT and DT fields in the DSPI*x*_CTAR*n* registers select the delay after transfer.

Refer to *Figure 378* for an illustration of the delay after transfer.

The following formula expresses the PDT/DT/delay after transfer relationship:

Equation 54

$$t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$$

Table 354 shows an example of the computed delay after transfer.

Table 354. Delay after transfer computation example

PDT	Prescaler value	DT	Scaler value	f_{SYS}	Delay after transfer
0b01	3	0b1110	32768	100 MHz	0.98 ms

21.8.4.5 Peripheral Chip Select strobe enable ($\overline{CS5\_x}$)

The $\overline{CS5\_x}$ signal provides a delay to allow the CS*x* signals to settle after transitioning, thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI*x*_MCR, CS5_*x* provides a signal for an external demultiplexer to decode the CS4_*x* signals into as many as 32 glitch-free CS*x* signals.

Figure 377 shows the timing of the CS5_*x* signal relative to CS signals.

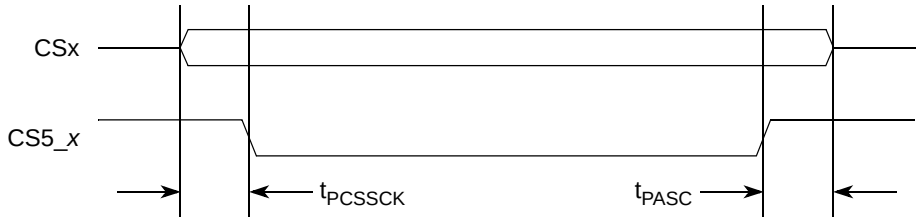


Figure 377. Peripheral Chip Select strobe timing

The delay between the assertion of the CS*x* signals and the assertion of CS5_*x* signal is selected by the PCSSCK field in the DSPI*x*_CTAR based on the following formula:

Equation 55

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK$$

At the end of the transfer the delay between CS5_*x* negation and CS*x* negation is selected by the PASC field in the DSPI*x*_CTAR based on the following formula:

Equation 56

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC$$

Table 355 shows an example of the computed t_{PCSSCK} delay.

Table 355. Peripheral Chip Select strobe assert computation example

PCSSCK	Prescaler	f_{SYS}	Delay before transfer
0b11	7	100 MHz	70.0 ns

Table 356 shows an example of the computed the t_{PASC} delay.

Table 356. Peripheral Chip Select strobe negate computation example

PASC	Prescaler	f_{SYS}	Delay after transfer
0b11	7	100 MHz	70.0 ns

21.8.5 Transfer formats

The SPI serial communication is controlled by the serial communications clock (SCK_*x*) signal and the CS*x* signals. The SCK_*x* signal provided by the master device synchronizes shifting and sampling of the data by the SIN_*x* and SOUT_*x* pins. The CS*x* signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI clock and transfer attributes registers (DSPI*x*_CTAR*n*) select the polarity and phase of the serial clock, SCK_*x*. The polarity bit selects the idle state of the SCK_*x*. The clock phase bit selects if the data on SOUT_*x* is valid before or on the first SCK_*x* edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI*x*_CTAR0 (SPI slave mode) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master device and the slave device to ensure proper transmission.

The DSPI supports four different transfer formats:

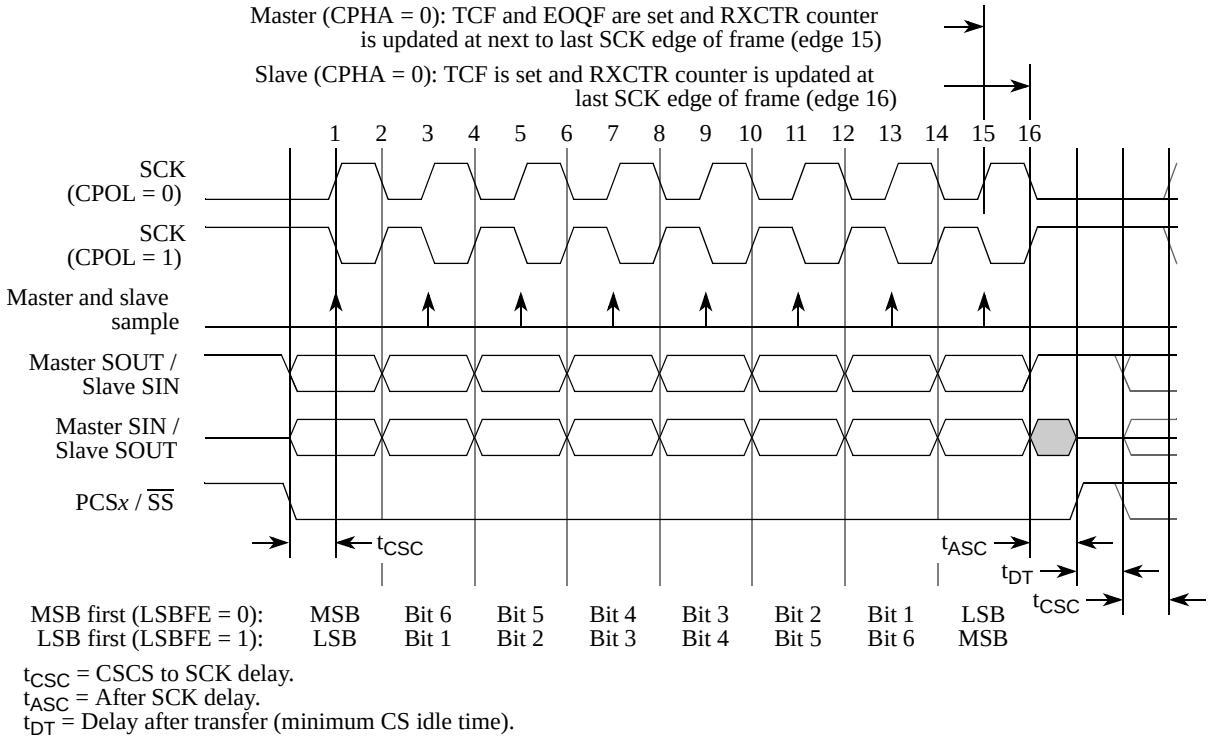
- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified transfer format with CPHA = 0
- Modified transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI*x*_MCR selects between classic SPI format and modified transfer format. The classic SPI formats are described in *Section 21.8.5.1: Classic SPI transfer format (CPHA = 0)* and *Section 21.8.5.2: Classic SPI transfer format (CPHA = 1)*." The modified transfer formats are described in *Section 21.8.5.3: Modified SPI transfer format (MTFE = 1, CPHA = 0)* and *Section 21.8.5.4: Modified SPI transfer format (MTFE = 1, CPHA = 1)*."

In the SPI configuration, the DSPI provides the option of keeping the CS signals asserted between frames. Refer to *Section 21.8.5.5: Continuous selection format* for details.

21.8.5.1 Classic SPI transfer format (CPHA = 0)

The transfer format shown in *Figure 378* is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN_*x* pins on the odd-numbered SCK_*x* edges and change the data on their SOUT_*x* pins on the even-numbered SCK_*x* edges.



t_{CSC} = CSCS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS idle time).

Figure 378. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

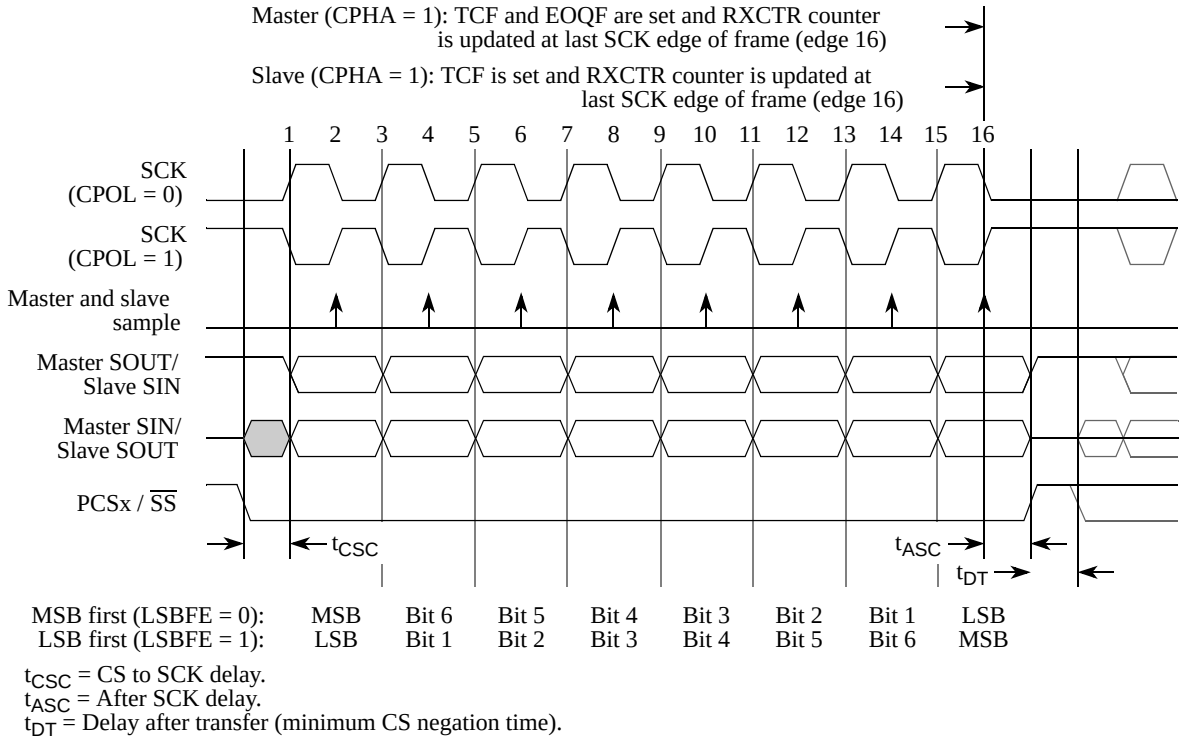
The master initiates the transfer by placing its first data bit on the SOUT_*x* pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT_*x* pin. After the t_{CSC} delay has elapsed, the master outputs the first edge of SCK_*x*. This is the edge used by the master and slave devices to sample the first input data bit on their serial data input signals. At the second edge of the SCK_*x* the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN_*x* pins on the odd-numbered clock edges and changes the data on their SOUT_*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the CS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For the CPHA = 0 condition of the master, TCF and EOQF are set and the RXCTR counter is updated at the next to last serial clock edge of the frame (edge 15) of *Figure 378*.

For the CPHA = 0 condition of the slave, TCF is set and the RXCTR counter is updated at the last serial clock edge of the frame (edge 16) of *Figure 378*.

21.8.5.2 Classic SPI transfer format (CPHA = 1)

The transfer format shown in *Figure 379* is used to communicate with peripheral SPI slave devices that require the first SCK_*x* edge before the first data bit becomes available on the slave SOUT_*x* pin. In this format the master and slave devices change the data on their SOUT_*x* pins on the odd-numbered SCK_*x* edges and sample the data on their SIN_*x* pins on the even-numbered SCK_*x* edges.



t_{CSC} = CS to SCK delay.
t_{ASC} = After SCK delay.
t_{DT} = Delay after transfer (minimum CS negation time).

Figure 379. DSPI transfer timing diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the CS*x* signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK_*x* edge and at the same time places valid data on the master SOUT_*x* pin. The slave responds to the first SCK_*x* edge by placing its first data bit on its slave SOUT_*x* pin.

At the second edge of the SCK_*x* the master and slave sample their SIN_*x* pins. For the rest of the frame the master and the slave change the data on their SOUT_*x* pins on the odd-numbered clock edges and sample their SIN_*x* pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the CS*x* signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

For CPHA = 1 the master EOQF and TCF and slave TCF are set at the last serial clock edge (edge 16) of *Figure 379*. For CPHA = 1 the master and slave RXCTR counters are updated on the same clock edge.

21.8.5.3 Modified SPI transfer format (MTFE = 1, CPHA = 0)

In this modified transfer format both the master and the slave sample later in the SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

The master and the slave place data on the SOUT_*x* pins at the assertion of the CS*x* signal. After the CS*x* to SCK_*x* delay has elapsed the first SCK_*x* edge is generated. The slave

samples the master SOUT_*x* signal on every odd numbered SCK_*x* edge. The slave also places new data on the slave SOUT_*x* on every odd numbered clock edge.

The master places its second data bit on the SOUT_*x* line one system clock after odd numbered SCK_*x* edge. The point where the master samples the slave SOUT_*x* is selected by writing to the SMPL_PT field in the DSPI*x*_MCR. *Table 357* lists the number of system clock cycles between the active edge of SCK_*x* and the master sample point for different values of the SMPL_PT bit field. The master sample point can be delayed by one or two system clock cycles.

Table 357. Delayed master sample point

SMPL_PT	Number of system clock cycles between odd-numbered edge of SCK and sampling of SIN
00	0
01	1
10	2
11	Invalid value

Figure 380 shows the modified transfer format for CPHA = 0. Only the condition where CPOL = 0 is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.

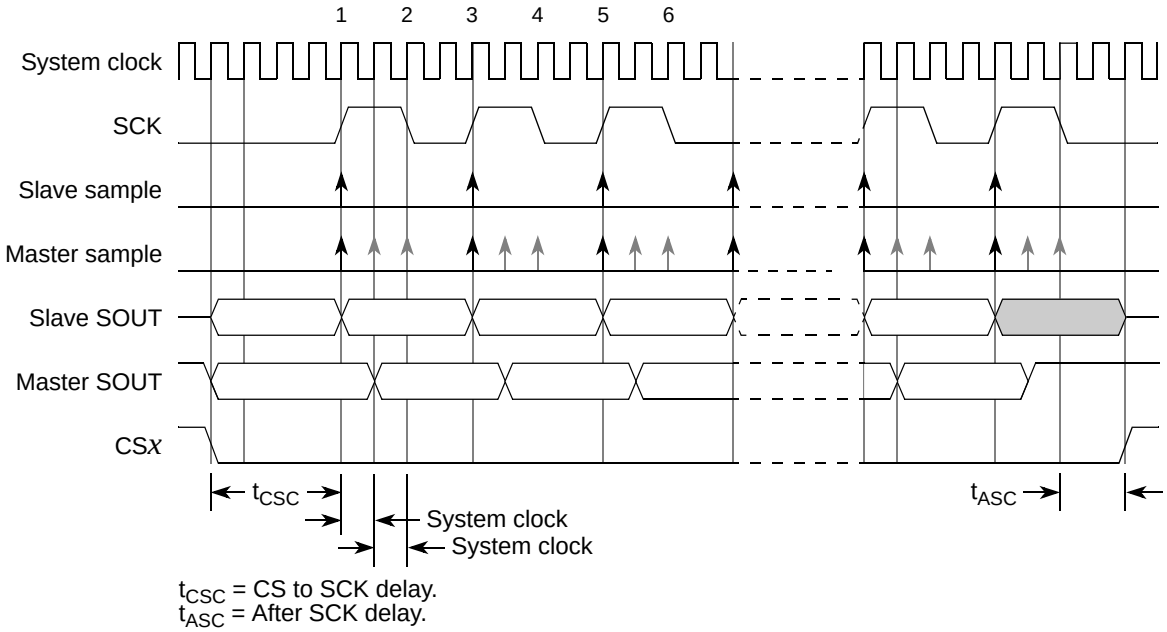


Figure 380. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{SCK} = f_{SYS} / 4$)

21.8.5.4 Modified SPI transfer format (MTFE = 1, CPHA = 1)

At the start of a transfer the DSPI asserts the CS signal to the slave device. After the CS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of

SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the 3rd SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge is visible on the master SCK pin during the sampling of the last bit. The SCK to CS delay must be programmed to be greater than or equal to half the SCK period.

Note: For the modified transfer format to operate correctly, you must thoroughly analyze the SPI link timing budget.

Figure 381 shows the modified transfer format for CPHA = 1. Only the condition where CPOL = 0 is shown.

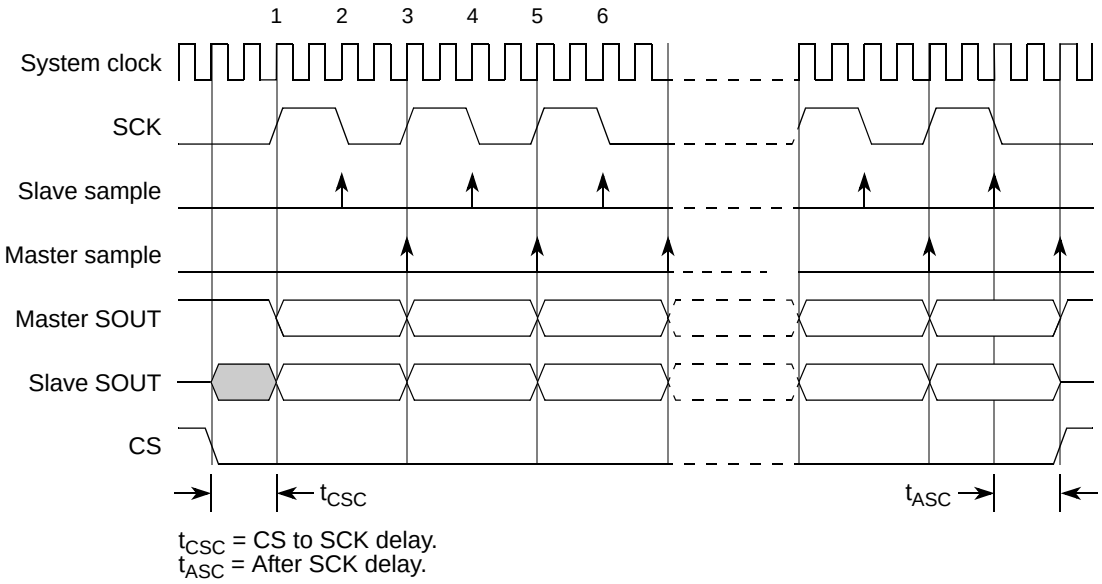


Figure 381. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{SCK} = f_{SYS} / 4$)

21.8.5.5 Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The continuous selection format provides the flexibility to handle both cases. The continuous selection format is enabled for the SPI configuration by setting the CONT bit in the SPI command.

When the CONT bit = 0, the DSPI drives the asserted chip select signals to their idle states in between frames. The idle states of the chip select signals are selected by the PCSIS field in the DSPI*x*_MCR.

Figure 382 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

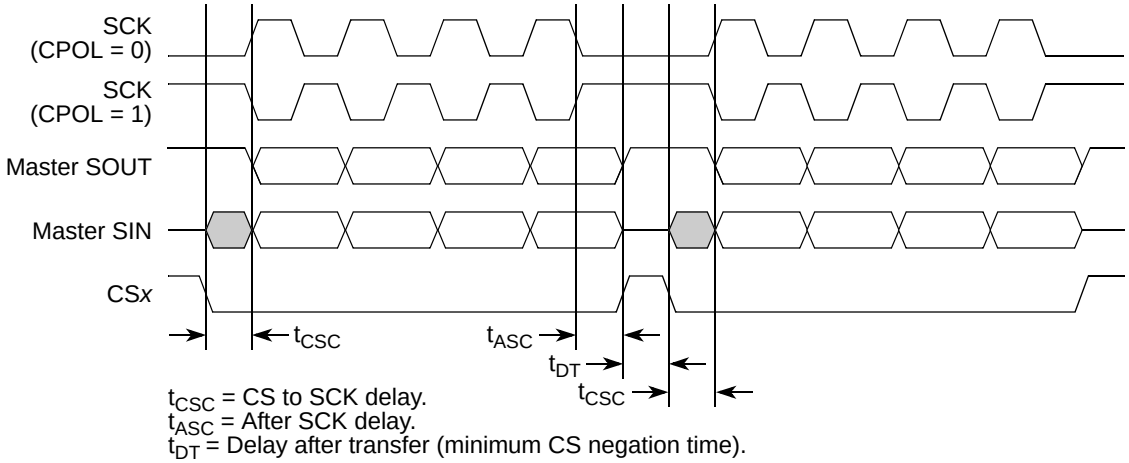


Figure 382. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT = 1 and the CS signal for the next transfer is the same as for the current transfer, the CS signal remains asserted for the duration of the two transfers. The delay between transfers (t_{DT}) is not inserted between the transfers.

Figure 383 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.

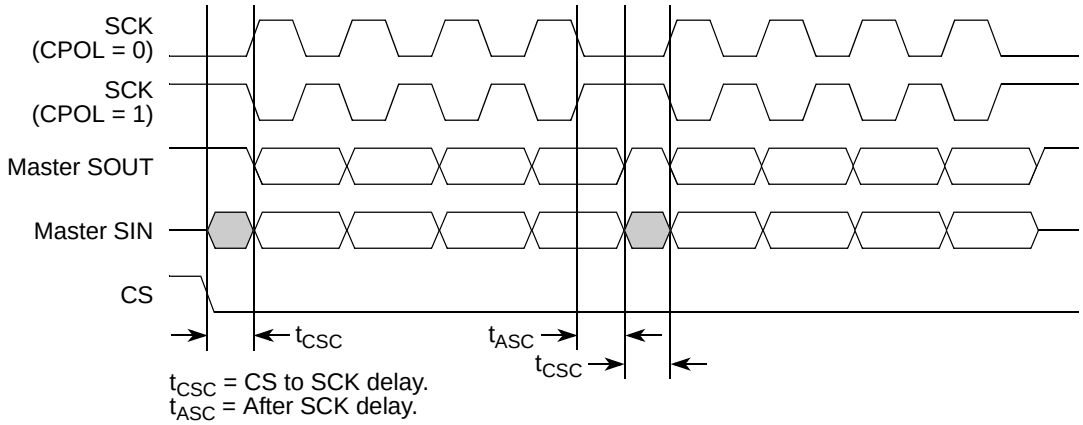


Figure 383. Example of continuous transfer (CPHA = 1, CONT = 1)

In *Figure 383*, the period length at the start of the next transfer is the sum of t_{ASC} and t_{CSC}; i.e., it does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations, t_{ASC} and t_{CSC} must be increased if a full half-clock period is required.

Switching CTARs between frames while using continuous selection can cause errors in the transfer. The CS signal must be negated before CTAR is switched.

When the CONT bit = 1 and the CS signals for the next transfer are different from the present transfer, the CS signals behave as if the CONT bit was not set.

Note: It is mandatory to fill the TXFIFO with the number of entries that will be concatenated together under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example; while transmitting in master mode, it should be ensured that the last

entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmited (i.e. the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be de-selected for any further serial communication; else an underflow error occurs

21.8.5.6 Clock polarity switching between DSPI transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame.

Refer to *Section 21.7.2.3: DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn)*.

In *Figure 384*, time 'A' shows the one clock interval. Time 'B' is user programmable from a minimum of two system clocks.

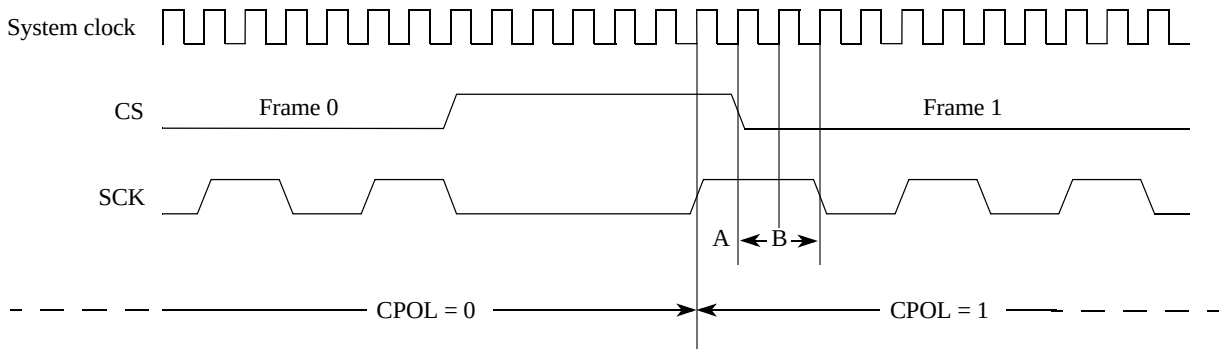


Figure 384. Polarity switching between frames

21.8.6 Continuous Serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting the CONT_SCKE bit in the DSPI*x*_MCR. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Setting CPHA = 0 is ignored if the CONT_SCKE bit is set. Continuous SCK is supported for modified transfer format.

Clock and transfer attributes for the continuous SCK mode are set according to the following rules:

- When the DSPI is in SPI configuration, CTAR0 is used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame is used.
- In all configurations, the currently selected CTAR remains in use until the start of a frame with a different CTAR specified, or the continuous SCK mode is terminated.

The device is designed to use the same baud rate for all transfers made while using the continuous SCK. Switching clock polarity between frames while using continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into module disable mode.

Enabling continuous SCK disables the CS to SCK delay and the After SCK delay. The delay after transfer is fixed at one SCK cycle. *Figure 385* shows timing diagram for continuous SCK format with continuous selection disabled.

Note: When in Continuous SCK mode, for the SPI transfer CTAR0 should always be used, and the TX-FIFO must be clear using the MCR.CLR_TXF field before initiating transfer.

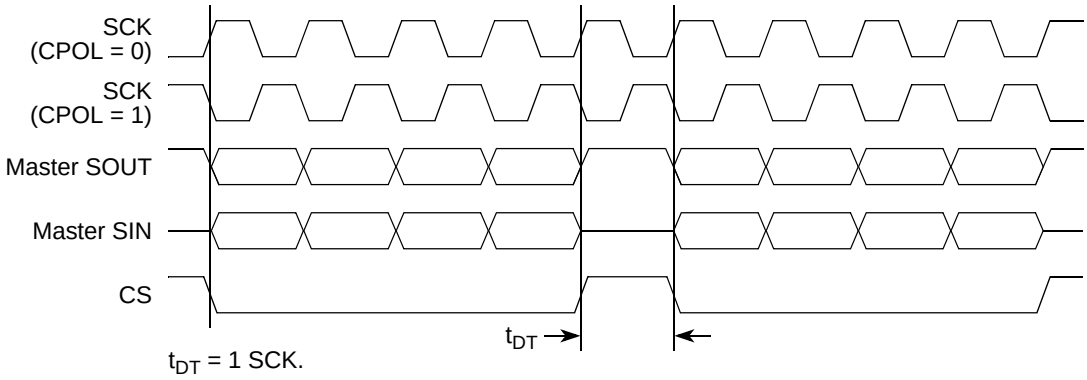


Figure 385. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set, CS remains asserted between the transfers when the CS signal for the next transfer is the same as for the current transfer. *Figure 386* shows timing diagram for continuous SCK format with continuous selection enabled.

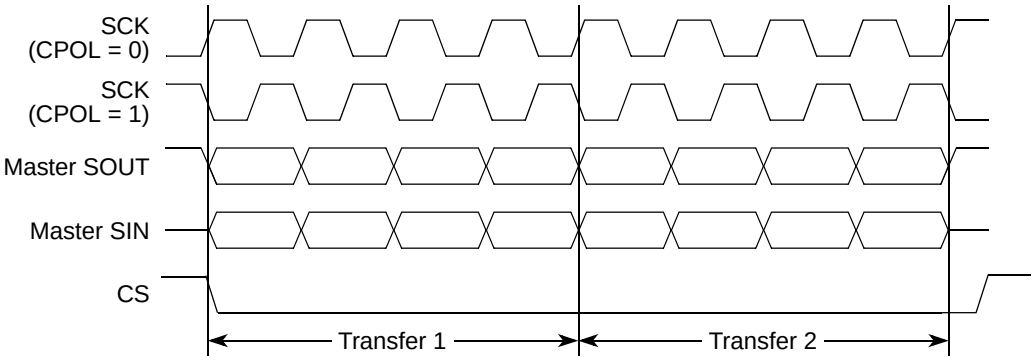


Figure 386. Continuous SCK timing diagram (CONT = 1)

21.8.7 Interrupts/DMA requests

The DSPI has conditions that can generate interrupt requests only, and conditions that can generate interrupts or DMA requests. *Table 358* lists these conditions.

Table 358. Interrupt and DMA request conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	
TX FIFO underflow has occurred	TFUF	X	
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow occurred	RFOF	X	
A FIFO overrun occurred(1)	TFUF ORed with RFOF	X	

1. The FIFO overrun condition is created by ORing the TFUF and RFOF flags together.

Each condition has a flag bit and a request enable bit. The flag bits are described in the *Section 21.7.2.4: DSPI Status Register (DSPIx_SR)* and the request enable bits are described in the *Section 21.7.2.5: DSPI DMA / Interrupt Request Select and Enable Register (DSPIx_RSER)*. The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the TFFF_DIRS and RFDF_DIRS bits in the DSPI*x*_RSER.

21.8.7.1 End of queue interrupt request (EOQF)

The end of queue request indicates that the end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the EOQF_RE bit in the DSPI*x*_RSER is set. Refer to the EOQ bit description in *Section 21.7.2.4: DSPI Status Register (DSPIx_SR)*. Refer to *Figure 378* and *Figure 379* that illustrate when EOQF is set.

21.8.7.2 Transmit FIFO fill interrupt or DMA request (TFFF)

The transmit FIFO fill request indicates that the TX FIFO is not full. The transmit FIFO fill request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFF_RE bit in the DSPI*x*_RSER is set. The TFFF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

21.8.7.3 Transfer complete interrupt request (TCF)

The transfer complete request indicates the end of the transfer of a serial frame. The transfer complete request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI*x*_RSER. Refer to the TCF bit description in *Section 21.7.2.4: DSPI Status Register (DSPIx_SR)*. Refer to *Figure 378* and *Figure 379*, which show when TCF is set.

21.8.7.4 Transmit FIFO underflow interrupt request (TFUF)

The transmit FIFO underflow request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode and SPI configuration is empty, and a transfer is initiated from an

external SPI master. If the TFUF bit is set while the TFUF_RE bit in the DSPI*x*_RSER is set, an interrupt request is generated.

21.8.7.5 Receive FIFO drain interrupt or DMA request (RFDF)

The receive FIFO drain request indicates that the RX FIFO is not empty. The receive FIFO drain request is generated when the number of entries in the RX FIFO is not zero, and the RFDF_RE bit in the DSPI*x*_RSER is set. The RFDF_DIRS bit in the DSPI*x*_RSER selects whether a DMA request or an interrupt request is generated.

21.8.7.6 Receive FIFO overflow interrupt request (RFOF)

The receive FIFO overflow request indicates that an overflow condition in the RX FIFO has occurred. A receive FIFO overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOF_RE bit in the DSPI*x*_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI*x*_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is negated, the incoming data is ignored.

21.8.7.7 FIFO overrun request (TFUF) or (RFOF)

The FIFO overrun request indicates that at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing together the RX FIFO overflow and TX FIFO underflow signals.

21.8.8 Power saving features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

21.8.8.1 Module disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host software can initiate the module disable mode by writing a 1 to the MDIS bit in the DSPI*x*_MCR.In module disable mode, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different affect when the DSPI is in the module disable mode. Reading the RX FIFO pop register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO push register does not change the state of the TX FIFO. Clearing either of the FIFOs does not have any effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI*x*_MCR does not have any affect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no affect. Writing to the DSPI*x*_TCR during module disable mode does not have an effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

21.9 Initialization and application information

21.9.1 Managing queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration. This section presents an example of how to manage queues for the DSPI.

1. The last command word from a queue is executed. The EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag (EOQF) in the DSPI*x*_SR is set.
3. The setting of the EOQF flag disables both serial transmission, and serial reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is negated to indicate the STOPPED state.
4. The eDMA continues to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the RXCNT in DSPI*x*_SR or by checking RFDF in the DSPI*x*_SR after each read operation of the DSPI*x*_POPR.
7. Modify DMA descriptor of TX and RX channels for “new” queues.
8. Flush TX FIFO by writing a 1 to the CLR_TXF bit in the DSPI*x*_MCR register and flush the RX FIFO by writing a 1 to the CLR_RXF bit in the DSPI*x*_MCR register.
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to SPI_TCNT field in the DSPI*x*_TCR.
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

21.9.2 Baud rate settings

Table 359 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI*x*_CTARs. The values are calculated at a 100 MHz system frequency.

Table 359. Baud rate values

		Baud Rate divider prescaler values (DSPI_CTAR[PBR])			
		2	**3**	**5**	**7**
Baud rate scaler values (DSPI_CTAR[BR])	**2**	25.0 MHz	16.7 MHz	10.0 MHz	7.14 MHz
	4	12.5 MHz	8.33 MHz	5.00 MHz	3.57 MHz
	6	8.33 MHz	5.56 MHz	3.33 MHz	2.38 MHz
	8	6.25 MHz	4.17 MHz	2.50 MHz	1.79 MHz
	16	3.12 MHz	2.08 MHz	1.25 MHz	893 kHz
	32	1.56 MHz	1.04 MHz	625 kHz	446 kHz
	64	781 kHz	521 kHz	312 kHz	223 kHz
	128	391 kHz	260 kHz	156 kHz	112 kHz
	256	195 kHz	130 kHz	78.1 kHz	55.8 kHz
	512	97.7 kHz	65.1 kHz	39.1 kHz	27.9 kHz
	1024	48.8 kHz	32.6 kHz	19.5 kHz	14.0 kHz
	2048	24.4 kHz	16.3 kHz	9.77 kHz	6.98 kHz
	4096	12.2 kHz	8.14 kHz	4.88 kHz	3.49 kHz
	8192	6.10 kHz	4.07 kHz	2.44 kHz	1.74 kHz
	16384	3.05 kHz	2.04 kHz	1.22 kHz	872 Hz
	32768	1.53 kHz	1.02 kHz	610 Hz	436 Hz

21.9.3 Delay settings

Table 360 shows the values for the delay after transfer (t_{DT}) and CS to SCK delay (t_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI*x*_CTARs. The values calculated assume a 100 MHz system frequency.

Table 360. Delay values

		Delay prescaler values (DSPI_CTAR[PBR])			
		1	3	5	7
Delay scaler values (DSPI_CTAR[DT])	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

21.9.4 Calculation of FIFO pointer addresses

The user has complete visibility of the TX and RX FIFO contents through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the transmit next pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the pop next pointer (POPNXTPTR).

Refer to *Section 21.8.3.4: Transmit First In First Out (TX FIFO) buffering mechanism* and *Section 21.8.3.5: Receive First In First Out (RX FIFO) buffering mechanism* for details on the FIFO operation. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO.

Figure 387 illustrates the concept of first-in and last-in FIFO entries along with the FIFO counter.

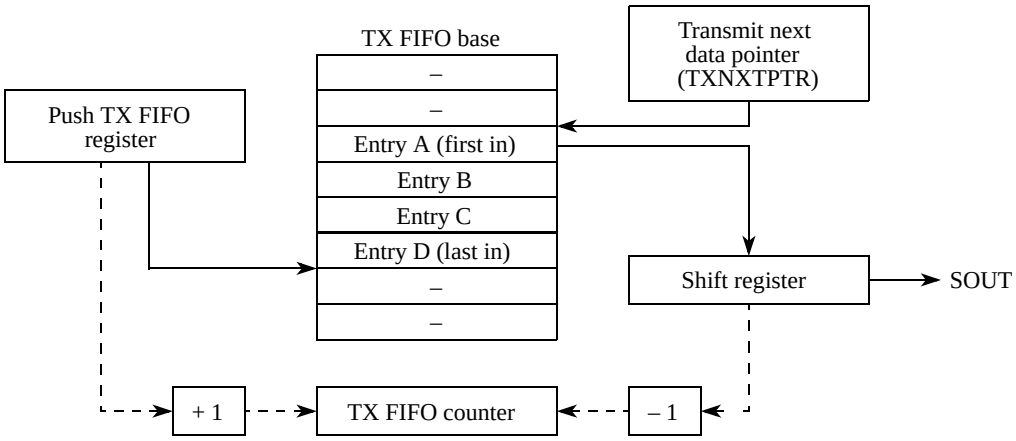


Figure 387. TX FIFO pointers and counter

21.9.4.1 Address calculation for first-in entry and last-in entry in TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

Equation 57

$$\text{First-in entry address} = \text{TXFIFO base} + 4\ (\text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

Equation 58

$$\text{Last-in entry address} = \text{TXFIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXTPTR} - 1) \text{ modulo TXFIFO depth}]$$

where:

TXFIFO base = base address of transmit FIFO

TXCTR = transmit FIFO counter

TXNXTPTR = transmit next pointer

TX FIFO depth = transmit FIFO depth (depth is 5)

21.9.4.2 Address calculation for first-in entry and last-in entry in RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

Equation 59

$$\text{First-in entry address} = \text{RXFIFO base} + 4 \times (\text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

Equation 60

$$\text{Last-in entry address} = \text{RXFIFO base} + 4 \times [(\text{RXCTR} + \text{POPNXTPTR} - 1) \text{ modulo RXFIFO depth}]$$

where:

RXFIFO base = base address of receive FIFO

RXCTR = receive FIFO counter

POPNXTPTR = pop next pointer

RX FIFO depth = receive FIFO depth (depth is 5)

22 LIN Controller (LINFlex)

22.1 Introduction

The LINFlex (Local Interconnect Network Flexible) controller interfaces the LIN network and supports the LIN protocol versions 1.3; 2.0 2.1; and J2602 in both Master and Slave modes. LINFlex includes a LIN mode that provides additional features (compared to standard UART) to ease LIN implementation, improve system robustness, minimize CPU load and allow slave node resynchronization.

22.2 Main features

22.2.1 LIN mode features

- Supports LIN protocol versions 1.3, 2.0, 2.1, and J2602
- Master mode with autonomous message handling
- Classic and enhanced checksum calculation and check
- Single 8-byte buffer for transmission/reception
- Extended frame mode for In-Application Programming (IAP) purposes
- Wake-up event on dominant bit detection
- True LIN field state machine
- Advanced LIN error detection
- Header, response and frame timeout
- Slave mode
  - Autonomous header handling
  - Autonomous transmit/receive data handling
- LIN automatic resynchronization, allowing operation with 16 MHz fast internal RC oscillator as clock source
- 16 identifier filters for autonomous message handling in Slave mode

22.2.2 UART mode features

- Full duplex communication
- 8- or 9-bit with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

22.2.3 Features common to LIN and UART

- Fractional baud rate generator
- 3 operating modes for power saving and configuration registers lock:
  - Initialization
  - Normal
  - Sleep
- 2 test modes:
  - Loop Back
  - Self Test
- Maskable interrupts

22.3 General description

The increasing number of communication peripherals embedded on microcontrollers, for example CAN, LIN and SPI, requires more and more CPU resources for communication management. Even a 32-bit microcontroller is overloaded if its peripherals do not provide high-level features to autonomously handle the communication.

Even though the LIN protocol with a maximum baud rate of 20 Kbit/s is relatively slow, it still generates a non-negligible load on the CPU if the LIN is implemented on a standard UART, as usually the case.

To minimize the CPU load in Master mode, LINFlex handles the LIN messages autonomously.

In Master mode, once the software has triggered the header transmission, LINFlex does not request any software intervention until the next header transmission request in transmission mode or until the checksum reception in reception mode.

To minimize the CPU load in Slave mode, LINFlex requires software intervention only to:

- Trigger transmission or reception or data discard depending on the identifier
- Write data into the buffer (transmission mode) or read data from the buffer (reception mode) after checksum reception

If filter mode is activated for Slave mode, LINFlex requires software intervention only to write data into the buffer (transmission mode) or read data from the buffer (reception mode)

The software uses the control, status and configuration registers to:

- Configure LIN parameters (for example, baud rate or mode)
- Request transmissions
- Handle receptions
- Manage interrupts
- Configure LIN error and timeout detection
- Process diagnostic information

The message buffer stores transmitted or received LIN frames.

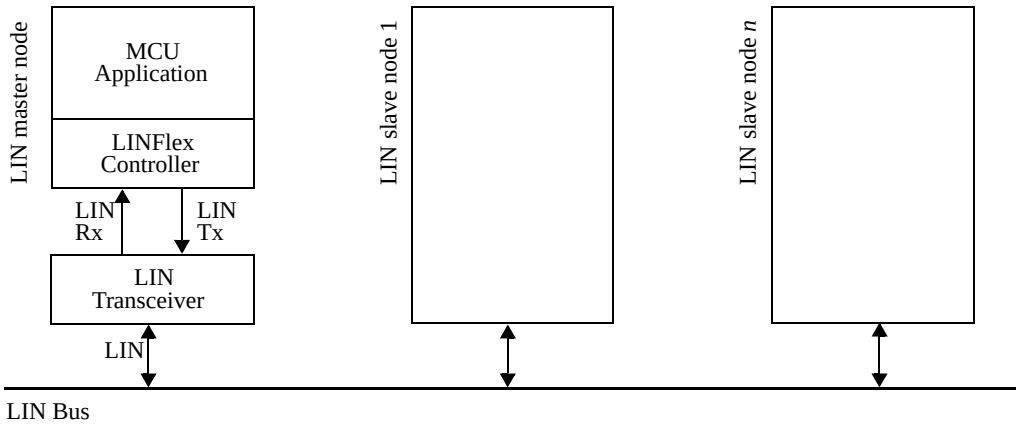
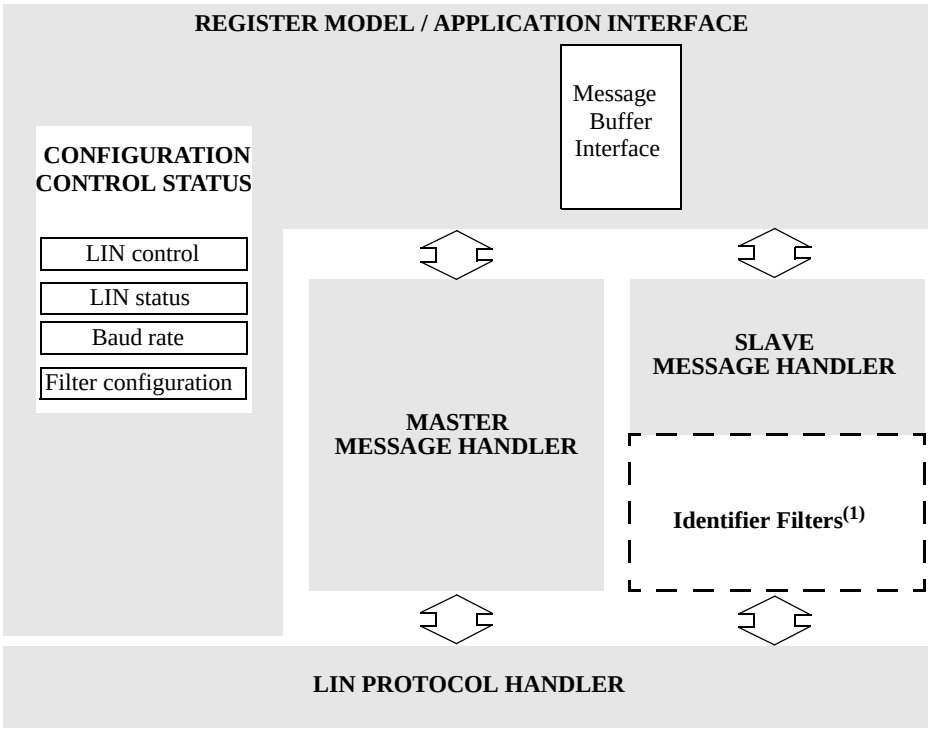


Figure 388. LIN topology network



1. Filter activation optional

Figure 389. LINFlex block diagram

22.4 Fractional baud rate generation

The baud rates for the receiver and transmitter are both set to the same value as programmed in the Mantissa (LINIBRR) and Fraction (LINFBRR) registers.

Equation 61

$$\text{Tx/ Rx baud} = \frac{f_{periph\_set\_1\_clk}}{(16 \times LFDIV)}$$

LFDIV is an unsigned fixed point number. The 12-bit mantissa is coded in the LINIBRR and the fraction is coded in the LINFBRR.

The following examples show how to derive LFDIV from LINIBRR and LINFBRR register values:

Example 1 Deriving LFDIV from LINIBRR and LINFBRR register values

If LINIBRR = 27d and LINFBRR = 12d, then

Mantissa (LFDIV) = 27d

Fraction (LFDIV) = 12/16 = 0.75d

Therefore LFDIV = 27.75d

Example 2 Programming LFDIV from LINIBRR and LINFBRR register values

To program LFDIV = 25.62d,

LINFBRR = 16 × 0.62 = 9.92, nearest real number 10d = 0xA

LINIBRR = mantissa (25.620d) = 25d = 0x19

Note: The baud counters are updated with the new value of the baud registers after a write to LINIBRR. Hence the baud register value must not be changed during a transaction. The LINFBRR (containing the Fraction bits) must be programmed before the LINIBRR.

Note: LFDIV must be greater than or equal to 1.5d, i.e. LINIBRR = 1 and LINFBRR = 8. Therefore, the maximum possible baudrate is fperiph_set_1_clk / 24.

Table 361. Error calculation for programmed baud rates

Baud rate	$f_{periph\_set\_1\_clk}$ = 64 MHz				$f_{periph\_set\_1\_clk}$ = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate
		LINIBRR	LINFBRR			LINIBRR	LINFBRR	
2400	2399.97	1666	11	–0.001	2399.88	416	11	–0.005
9600	9599.52	416	11	–0.005	9598.08	104	3	–0.02
10417	10416.7	384	0	–0.003	10416.7	96	0	–0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	–0.08
115200	115108	34	12	–0.08	115108	8	11	–0.08
230400	230216	17	6	–0.08	231884	4	5	0.644
460800	460432	8	11	–0.08	457143	2	3	–0.794
921600	927536	4	5	0.644	941176	1	1	2.124

22.5 Operating modes

LINFlex has three main operating modes: Initialization, Normal and Sleep. After a hardware reset, LINFlex is in Sleep mode to reduce power consumption. The software instructs LINFlex to enter Initialization mode or Sleep mode by setting the INIT bit or SLEEP bit in the LINCR1.

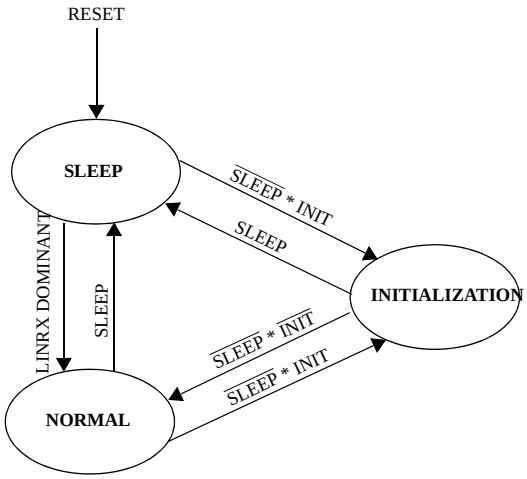


Figure 390. LINFlex operating modes

22.5.1 Initialization mode

The software can be initialized while the hardware is in Initialization mode. To enter this mode the software sets the INIT bit in the LINCR1.

To exit Initialization mode, the software clears the INIT bit.

While in Initialization mode, all message transfers to and from the LIN bus are stopped and the status of the LIN bus output LINTX is recessive (high).

Entering Initialization mode does not change any of the configuration registers.

To initialize the LINFlex controller, the software selects the mode (LIN Master, LIN Slave or UART), sets up the baud rate register and, if LIN Slave mode with filter activation is selected, initializes the identifier list.

22.5.2 Normal mode

Once initilization is complete, software clears the INIT bit in the LINCR1 to put the hardware into Normal mode.

22.5.3 Low power mode (Sleep)

To reduce power consumption, LINFlex has a low power mode called Sleep mode. To enter Sleep mode, software sets the SLEEP bit in the LINCR1. In this mode, the LINFlex clock is

stopped. Consequently, the LINFlex will not update the status bits but software can still access the LINFlex registers.

LINFlex can be awakened (exit Sleep mode) either by software clearing the SLEEP bit or on detection of LIN bus activity if automatic wake-up mode is enabled (AWUM bit is set).

On LIN bus activity detection, hardware automatically performs the wake-up sequence by clearing the SLEEP bit if the AWUM bit in the LINCR1 is set. To exit from Sleep mode if the AWUM bit is cleared, software clears the SLEEP bit when a wake-up event occurs.

22.6 Test modes

Two test modes are available to the user: Loop Back mode and Self Test mode. They can be selected by the LBKM and SFTM bits in the LINCR1. These bits must be configured while LINFlex is in Initialization mode. Once one of the two test modes has been selected, LINFlex must be started in Normal mode.

22.6.1 Loop Back mode

LINFlex can be put in Loop Back mode by setting the LBKM bit in the LINCR. In Loop Back mode, the LINFlex treats its own transmitted messages as received messages.

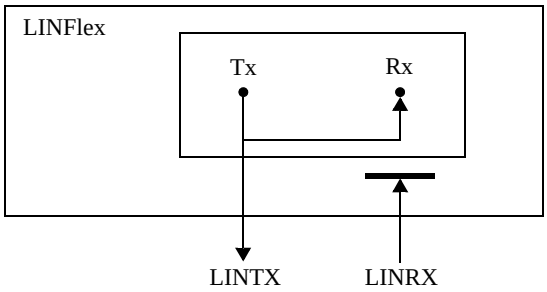


Figure 391. LINFlex in loop back mode

This mode is provided for self test functions. To be independent of external events, the LIN core ignores the LINRX signal. In this mode, the LINFlex performs an internal feedback from its Tx output to its Rx input. The actual value of the **LINRX** input pin is disregarded by the LINFlex. The transmitted messages can be monitored on the **LINTX** pin.

22.6.2 Self Test mode

LINFlex can be put in Self Test mode by setting the LBKM and SFTM bits in the LINCR. This mode can be used for a “Hot Self Test”, meaning the LINFlex can be tested as in Loop Back mode but without affecting a running LIN system connected to the LINTX and LINRX pins. In this mode, the LINRX pin is disconnected from the LINFlex and the LINTX pin is held recessive.

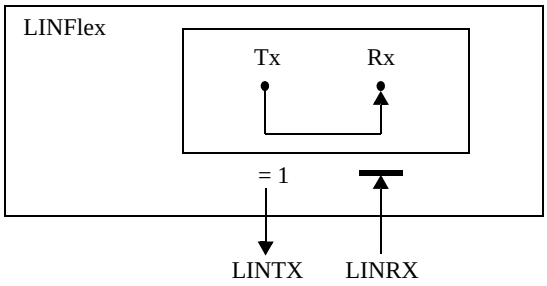


Figure 392. LINFlex in self test mode

22.7 Memory map and registers description

22.7.1 Memory map

See the “Memory map” chapter of this reference manual for the base addresses for the LINFlex modules.

Table 362 shows the LINFlex memory map.

Table 362. LINFlex memory map

Address offset	Register	Location
0x0000	LIN control register 1 (LINCR1)	*on page 671*
0x0004	LIN interrupt enable register (LINIER)	*on page 674*
0x0008	LIN status register (LINSR)	*on page 676*
0x000C	LIN error status register (LINESR)	*on page 679*
0x0010	UART mode control register (UARTCR)	*on page 680*
0x0014	UART mode status register (UARTSR)	*on page 681*
0x0018	LIN timeout control status register (LINTCSR)	*on page 683*
0x001C	LIN output compare register (LINOCR)	*on page 684*
0x0020	LIN timeout control register (LINTOCR)	*on page 685*
0x0024	LIN fractional baud rate register (LINFBRR)	*on page 685*
0x0028	LIN integer baud rate register (LINIBRR)	*on page 686*
0x002C	LIN checksum field register (LINCFR)	*on page 687*
0x0030	LIN control register 2 (LINCR2)	*on page 687*
0x0034	Buffer identifier register (BIDR)	*on page 688*
0x0038	Buffer data register LSB (BDRL)(1)	*on page 689*
0x003C	Buffer data register MSB (BDRM)(2)	*on page 690*

Table 362. LINFlex memory map(Continued)

Address offset	Register	Location
0x0040	Identifier filter enable register (IFER)	*on page 690*
0x0044	Identifier filter match index (IFMI)	*on page 691*
0x0048	Identifier filter mode register (IFMR)	*on page 691*
0x004C	Identifier filter control register 0 (IFCR0)	*on page 692*
0x0050	Identifier filter control register 1 (IFCR1)	*on page 693*
0x0054	Identifier filter control register 2 (IFCR2)	*on page 693*
0x0058	Identifier filter control register 3 (IFCR3)	*on page 693*
0x005C	Identifier filter control register 4 (IFCR4)	*on page 693*
0x0060	Identifier filter control register 5 (IFCR5)	*on page 693*
0x0064	Identifier filter control register 6 (IFCR6)	*on page 693*
0x0068	Identifier filter control register 7 (IFCR7)	*on page 693*
0x006C	Identifier filter control register 8 (IFCR8)	*on page 693*
0x0070	Identifier filter control register 9 (IFCR9)	*on page 693*
0x0074	Identifier filter control register 10 (IFCR10)	*on page 693*
0x0078	Identifier filter control register 11 (IFCR11)	*on page 693*
0x007C	Identifier filter control register 12 (IFCR12)	*on page 693*
0x0080	Identifier filter control register 13 (IFCR13)	*on page 693*
0x0084	Identifier filter control register 14 (IFCR14)	*on page 693*
0x0088	Identifier filter control register 15 (IFCR15)	*on page 693*
0x008C–0x000F	Reserved	

1. LSB: Least significant byte
2. MSB: Most significant byte

22.7.1.1 LIN control register 1 (LINCR1)

Offset: 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CCD	CFD	LASE	AWUM	MBL				BF	SFTM	LBKM	MME	SBDT	RBLM	SLEEP	INIT
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Figure 393. LIN control register 1 (LINCR1)

Table 363. LINCR1 field descriptions

Field	Description
CCD	Checksum calculation disable This bit disables the checksum calculation (see *Table 364*). 0 Checksum calculation is done by hardware. When this bit is 0, the LINCFR is read-only. 1 Checksum calculation is disabled. When this bit is set the LINCFR is read/write. User can program this register to send a software-calculated CRC (provided CFD is 0). **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
CFD	Checksum field disable This bit disables the checksum field transmission (see *Table 364*). 0 Checksum field is sent after the required number of data bytes is sent. 1 No checksum field is sent. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
LASE	LIN Slave Automatic Resynchronization Enable 0 Automatic resynchronization disable. 1 Automatic resynchronization enable. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
AWUM	Automatic Wake-Up Mode This bit controls the behavior of the LINFlex hardware during Sleep mode. 0 The Sleep mode is exited on software request by clearing the SLEEP bit of the LINCR. 1 The Sleep mode is exited automatically by hardware on LINRX dominant state detection. The SLEEP bit of the LINCR is cleared by hardware whenever WUF bit in the LINSR is set. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
MBL	LIN Master Break Length This field indicates the Break length in Master mode (see *Table 365*). **Note:** This field can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
BF	Bypass filter 0 No interrupt if identifier does not match any filter. 1 An RX interrupt is generated on identifier not matching any filter. **Note:** – If no filter is activated, this bit is reserved. – This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SFTM	Self Test Mode This bit controls the Self Test mode. For more details, see *Section 22.6.2: Self Test mode*. 0 Self Test mode disable. 1 Self Test mode enable. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
LBKM	Loop Back Mode This bit controls the Loop Back mode. For more details see *Section 22.6.1: Loop Back mode*. 0 Loop Back mode disable. 1 Loop Back mode enable. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode
MME	Master Mode Enable 0 Slave mode enable. 1 Master mode enable. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.

Table 363. LINCR1 field descriptions(Continued)

Field	Description
SBDT	Slave Mode Break Detection Threshold 0 11-bit break. 1 10-bit break. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
RBLM	Receive Buffer Locked Mode 0 Receive Buffer not locked on overrun. Once the Slave Receive Buffer is full the next incoming message overwrites the previous one. 1 Receive Buffer locked against overrun. Once the Receive Buffer is full the next incoming message is discarded. **Note:** This bit can be written in Initialization mode only. It is read-only in Normal or Sleep mode.
SLEEP	Sleep Mode Request This bit is set by software to request LINFlex to enter Sleep mode. This bit is cleared by software to exit Sleep mode or by hardware if the AWUM bit in LINCR1 and the WUF bit in LINSR are set (see *Table 366*).
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see *Table 366*).

Table 364. Checksum bits configuration

CFD	CCD	LINCFR	Checksum sent
1	1	Read/Write	None
1	0	Read-only	None
0	1	Read/Write	Programmed in LINCFR by bits CF[0:7]
0	0	Read-only	Hardware calculated

Table 365. LIN master break length selection

MBL	Length
0000	10-bit
0001	11-bit
0010	12-bit
0011	13-bit
0100	14-bit
0101	15-bit
0110	16-bit
0111	17-bit
1000	18-bit
1001	19-bit
1010	20-bit
1011	21-bit

Table 365. LIN master break length selection(Continued)

MBL	Length
1100	22-bit
1101	23-bit
1110	36-bit
1111	50-bit

Table 366. Operating mode selection

SLEEP	INIT	Operating mode
1	0	Sleep (reset value)
x	1	Initialization
0	0	Normal

22.7.1.2 LIN interrupt enable register (LINIER)

Offset: 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZIE	OCIE	BEIE	CEIE	HEIE	0	0	FEIE	BOIE	LSIE	WUIE	DBFIE	DBEIE	DRIE	DTIE	HRIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 394. LIN interrupt enable register (LINIER)

Table 367. LINIER field descriptions

Field	Description
SZIE	Stuck at Zero Interrupt Enable 0 No interrupt when SZF bit in LINESR or UARTSR is set. 1 Interrupt generated when SZF bit in LINESR or UARTSR is set.
OCIE	Output Compare Interrupt Enable 0 No interrupt when OCF bit in LINESR or UARTSR is set. 1 Interrupt generated when OCF bit in LINESR or UARTSR is set.
BEIE	Bit Error Interrupt Enable 0 No interrupt when BEF bit in LINESR is set. 1 Interrupt generated when BEF bit in LINESR is set.
CEIE	Checksum Error Interrupt Enable 0 No interrupt on Checksum error. 1 Interrupt generated when checksum error flag (CEF) in LINESR is set.

Table 367. LINIER field descriptions(Continued)

Field	Description
HEIE	Header Error Interrupt Enable 0 No interrupt on Break Delimiter error, Synch Field error, Identifier field error. 1 Interrupt generated on Break Delimiter error, Synch Field error, Identifier field error.
FEIE	Framing Error Interrupt Enable 0 No interrupt on Framing error. 1 Interrupt generated on Framing error.
BOIE	Buffer Overrun Interrupt Enable 0 No interrupt on Buffer overrun. 1 Interrupt generated on Buffer overrun.
LSIE	LIN State Interrupt Enable 0 No interrupt on LIN state change. 1 Interrupt generated on LIN state change. This interrupt can be used for debugging purposes. It has no status flag but is reset when writing '1111' into LINS[0:3] in the LINSR.
WUIE	Wake-up Interrupt Enable 0 No interrupt when WUF bit in LINSR or UARTSR is set. 1 Interrupt generated when WUF bit in LINSR or UARTSR is set.
DBFIE	Data Buffer Full Interrupt Enable 0 No interrupt when buffer data register is full. 1 Interrupt generated when data buffer register is full.
DBEIE	Data Buffer Empty Interrupt Enable 0 No interrupt when buffer data register is empty. 1 Interrupt generated when data buffer register is empty.
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt when data reception is completed. 1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.
DTIE	Data Transmitted Interrupt Enable 0 No interrupt when data transmission is completed. 1 Interrupt generated when data transmitted flag (DTF) is set in LINSR or UARTSR.
HRIE	Header Received Interrupt Enable 0 No interrupt when a valid LIN header has been received. 1 Interrupt generated when a valid LIN header has been received, that is, HRF bit in LINSR is set.

22.7.1.3 LIN status register (LINSR)

Offset: 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LINS				0	0	RMB	0	RBSY	RPS	WUF	DBFF	DBEF	DRF	DTF	HRF
W							w1c		w1c		w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 395. LIN status register (LINSR)

Table 368. LINSR field descriptions

Field	Description
LINS	LIN modes / normal mode states **0000: Sleep mode** LINFlex is in Sleep mode to save power consumption. **0001: Initialization mode** LINFlex is in Initialization mode. Normal mode states **0010: Idle** This state is entered on several events: – SLEEP bit and INIT bit in LINCR1 have been cleared by software, – A falling edge has been received on RX pin and AWUM bit is set, – The previous frame reception or transmission has been completed or **aborted.** **0011: Break** In Slave mode, a falling edge followed by a dominant state has been detected. Receiving Break. **Note:** In Slave mode, in case of error new LIN state can be either Idle or Break depending on last bit state. If last bit is dominant new LIN state is Break, otherwise Idle. In Master mode, Break transmission ongoing. **0100: Break Delimiter** In Slave mode, a valid Break has been detected. See *Section 22.7.1.1: LIN control register 1 (LINCR1)* for break length configuration (10-bit or 11-bit). Waiting for a rising edge. In Master mode, Break transmission has been completed. Break Delimiter transmission is ongoing. **0101: Synch Field** **I**n Slave mode, a valid Break Delimiter has been detected (recessive state for at least one bit time). Receiving Synch Field. In Master mode, Synch Field transmission is ongoing. **0110: Identifier Field** In Slave mode, a valid Synch Field has been received. Receiving Identifier Field. In Master mode, identifier transmission is ongoing. **0111: Header reception/transmission completed** In Slave mode, a valid header has been received and identifier field is available in the BIDR. In Master mode, header transmission is completed. **1000: Data reception/transmission** Response reception/transmission is ongoing. **1001: Checksum** Data reception/transmission completed. Checksum reception/transmission ongoing. In UART mode, only the following states are flagged by the LIN state bits: – Init – Sleep – Idle – Data transmission/reception
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.

Table 368. LINSR field descriptions(Continued)

Field	Description
RBSY	Receiver Busy Flag 0 Receiver is idle 1 Reception ongoing **Note:** In Slave mode, after header reception, if BIDR[DIR] = 0 and reception starts then this bit is set. In this case, user cannot program LINCR2[DTRQ] = 1.
RPS	LIN receive pin state This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin when: – Slave is in Sleep mode – Master is in Sleep mode or idle state This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if WUIE bit in LINIER is set.
DBFF	Data Buffer Full Flag This bit is set by hardware and indicates the buffer is full. It is set only when receiving extended frames (DFL > 7). This bit must be cleared by software. It is reset by hardware in Initialization mode.
DBEF	Data Buffer Empty Flag This bit is set by hardware and indicates the buffer is empty. It is set only when transmitting extended frames (DFL > 7). This bit must be cleared by software, once buffer has been filled again, in order to start transmission. This bit is reset by hardware in Initialization mode.
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. **Note:** This flag is not set in case of bit error or framing error.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed. This bit must be cleared by software. It is reset by hardware in Initialization mode. **Note:** This flag is not set in case of bit error if IOBE bit is reset.
HRF	Header Reception Flag This bit is set by hardware and indicates a valid header reception is completed. This bit must be cleared by software. This bit is reset by hardware in Initialization mode and at end of completed or aborted frame. **Note:** If filters are enabled, this bit is set only when identifier software filtering is required, that is to say: – All filters are inactive and BF bit in LINCR1 is set – No match in any filter and BF bit in LINCR1 is set – TX filter match

22.7.1.4 LIN error status register (LINESR)

Offset: 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	BEF	CEF	SFEF	BDEF	IDPEF	FEF	BOF	0	0	0	0	0	0	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 396. LIN error status register (LINESR)

Table 369. LINESR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. If the dominant state continues, SZF flag is set again after 87-bit time. It is cleared by software.
OCF	Output Compare Flag 0 No output compare event occurred 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. If this bit is set and IOT bit in LINTCSR is set, LINFlex moves to Idle state. If LTOM bit in LINTCSR is set, then OCF is cleared by hardware in Initialization mode. If LTOM bit is cleared, then OCF maintains its status whatever the mode is.
BEF	Bit Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a bit error. This error can occur during response field transmission (Slave and Master modes) or during header transmission (in Master mode). This bit is cleared by software.
CEF	Checksum Error Flag This bit is set by hardware and indicates that the received checksum does not match the hardware calculated checksum. This bit is cleared by software. **Note:** This bit is never set if CCD or CFD bit in LINCR1 is set.
SFEF	Synch Field Error Flag This bit is set by hardware and indicates that a Synch Field error occurred (inconsistent Synch Field).
BDEF	Break Delimiter Error Flag This bit is set by hardware and indicates that the received Break Delimiter is too short (less than one bit time).
IDPEF	Identifier Parity Error Flag This bit is set by hardware and indicates that a Identifier Parity error occurred. **Note:** Header interrupt is triggered when SFEF or BDEF or IDPEF bit is set and HEIE bit in LINIER is set.

Table 369. LINESR field descriptions(Continued)

Field	Description
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit). This error can occur during reception of any data in the response field (Master or Slave mode) or during reception of Synch Field or Identifier Field in Slave mode.
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites the buffer. It can be cleared by software.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

22.7.1.5 UART mode control register (UARTCR)

Offset: 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	TDFL		0	RDFL		0	0	0	0	RXEN	TXEN	OP	PCE	WL	UART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 397. UART mode control register (UARTCR)

Table 370. UARTCR field descriptions

Field	Description
TDFL	Transmitter Data Field length This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1. 00 Transmit buffer size = 1. 01 Transmit buffer size = 2. 10 Transmit buffer size = 3. 11 Transmit buffer size = 4.
RDFL	Receiver Data Field length This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1. 00 Receive buffer size = 1. 01 Receive buffer size = 2. 10 Receive buffer size = 3. 11 Receive buffer size = 4.

Table 370. UARTCR field descriptions(Continued)

Field	Description
RXEN	Receiver Enable 0 Receiver disable. 1 Receiver enable. This bit can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disable. 1 Transmitter enable. This bit can be programmed only when the UART bit is set. **Note:** Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
OP	Odd Parity 0 Sent parity is even. 1 Sent parity is odd. This bit can be programmed in Initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disable. 1 Parity transmit/check enable. This bit can be programmed in Initialization mode only when the UART bit is set.
WL	Word Length in UART mode 0 7-bit data + parity bit. 1 8-bit data (or 9-bit if PCE is set). This bit can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0 LIN mode. 1 UART mode. This bit can be programmed in Initialization mode only.

22.7.1.6 UART mode status register (UARTSR)

Offset: 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	0	DRF	DTF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c			w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 398. UART mode status register (UARTSR)

Table 371. UARTSR field descriptions

Field	Description
SZF	Stuck at Zero Flag This bit is set by hardware when the bus is dominant for more than a 100-bit time. It is cleared by software.
OCF	OCF Output Compare Flag 0 No output compare event occurred. 1 The content of the counter has matched the content of OC1[0:7] or OC2[0:7] in LINOCR. An interrupt is generated if the OCIE bit in LINIER register is set.
PE3	Parity Error Flag Rx3 This bit indicates if there is a parity error in the corresponding received byte (Rx3). See *Section 22.8.1.1: Buffer in UART mode*. No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE2	Parity Error Flag Rx2 This bit indicates if there is a parity error in the corresponding received byte (Rx2). See *Section 22.8.1.1: Buffer in UART mode*. No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE1	Parity Error Flag Rx1 This bit indicates if there is a parity error in the corresponding received byte (Rx1). See *Section 22.8.1.1: Buffer in UART mode*. No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
PE0	Parity Error Flag Rx0 This bit indicates if there is a parity error in the corresponding received byte (Rx0). See *Section 22.8.1.1: Buffer in UART mode*. No interrupt is generated if this error occurs. 0 No parity error. 1 Parity error.
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.
FEF	Framing Error Flag This bit is set by hardware and indicates to the software that LINFlex has detected a framing error (invalid stop bit).
BOF	Buffer Overrun Flag This bit is set by hardware when a new data byte is received and the buffer full flag is not cleared. If RBLM in LINCR1 is set then the new byte received is discarded. If RBLM is reset then the new byte overwrites buffer. it can be cleared by software.
RPS	LIN Receive Pin State This bit reflects the current status of LINRX pin for diagnostic purposes.
WUF	Wake-up Flag This bit is set by hardware and indicates to the software that LINFlex has detected a falling edge on the LINRX pin in Sleep mode. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt i generated if WUIE bit in LINIER is set.

Table 371. UARTSR field descriptions(Continued)

Field	Description
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DRIE bit in LINIER is set. **Note:** In UART mode, this flag is set in case of framing error, parity error or overrun.
DTF	Data Transmission Completed Flag This bit is set by hardware and indicates the data transmission is completed, that is, the number of bytes programmed in TDFL[0:1] have been transmitted. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DTIE bit in LINIER is set.
NF	Noise Flag This bit is set by hardware when noise is detected on a received character. This bit is cleared by software.

22.7.1.7 LIN timeout control status register (LINTCSR)

Offset: 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	LTOM	IOT	TOCE	CNT							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 399. LIN timeout control status register (LINTCSR)

Table 372. LINTCSR field descriptions

Field	Description
LTOM	LIN timeout mode 0 LIN timeout mode (header, response and frame timeout detection). 1 Output compare mode. This bit can be set/cleared in Initialization mode only.
IOT	Idle on Timeout 0 LIN state machine not reset to Idle on timeout event. 1 LIN state machine reset to Idle on timeout event. This bit can be set/cleared in Initialization mode only.

Table 372. LINTCSR field descriptions(Continued)

Field	Description
TOCE	Timeout counter enable 0 Timeout counter disable. OCF bit in LINESR or UARTSR is not set on an output compare event. 1 Timeout counter enable. OCF bit is set if an output compare event occurs. TOCE bit is configurable by software in Initialization mode. If LIN state is not Init and if timer is in LIN timeout mode, then hardware takes control of TOCE bit.
CNT	Counter Value This field indicates the LIN timeout counter value.

22.7.1.8 LIN output compare register (LINOCR)

Offset: 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OC2[1]								OC1[1]							
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

1 If LINTCSR[LTOM] = 0, this field is read-only.

Figure 400. LIN output compare register (LINOCR)

Table 373. LINOCR field descriptions

Field	Description
OC2	Output compare 2 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.
OC1	Output compare 1 value These bits contain the value to be compared to the value of bits CNT[0:7] in LINTCSR.

22.7.1.9 LIN timeout control register (LINTOCR)

Offset: 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	RTO				0	HTO						
W																
Reset	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	0

Figure 401. LIN timeout control register (LINTOCR)

Table 374. LINTOCR field descriptions

Field	Description
RTO	Response timeout value This field contains the response timeout duration (in bit time) for 1 byte. The reset value is 0xE = 14, corresponding to $T_{Response\_Maximum} = 1.4 \times T_{Response\_Nominal}$
HTO	Header timeout value This field contains the header timeout duration (in bit time). This value does not include the Break and the Break Delimiter. The reset value is the 0x2C = 44, corresponding to $T_{Header\_Maximum}$. Programming LINSR[MME] = 1 changes the HTO value to 0x1C = 28. This field can be written only in Slave mode.

22.7.1.10 LIN fractional baud rate register (LINFBRR)

Offset: 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIV_F			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 402. LIN fractional baud rate register (LINFBRR)

Table 375. LINFBRR field descriptions

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This field can be written in Initialization mode only.

22.7.1.11 LIN integer baud rate register (LINIBRR)

Offset: 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DIV_M												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 403. LIN integer baud rate register (LINIBRR)

Table 376. LINIBRR field descriptions

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see *Table 377*). This field can be written in Initialization mode only.

Table 377. Integer baud rate selection

DIV_M[0:12]	Mantissa
0x0000	LIN clock disabled
0x0001	1
...	...
0x1FFE	8190
ox1FFF	8191

22.7.1.12 LIN checksum field register (LINCFR)

Offset: 0x002C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	CF							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 404. LIN checksum field register (LINCFR)

Table 378. LINCFR field descriptions

Field	Description
CF	Checksum bits When LINCR1[CCD] = 0, this field is read-only. When LINCR1[CCD] = 1, this field is read/write. See *Table 364*.

22.7.1.13 LIN control register 2 (LINCR2)

Offset: 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	IOBE	IOPE	0	0	0	0	0	0	0	0	0	0	0	0	0
W				WURQ	DDRQ	DTRQ	ABRQ	HTRQ								
Reset	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 405. LIN control register 2 (LINCR2)

Table 379. LINCR2 field descriptions

Field	Description
IOBE	Idle on Bit Error 0 Bit error does not reset LIN state machine. 1 Bit error reset LIN state machine. This bit can be set/cleared in Initialization mode only.
IOPE	Idle on Identifier Parity Error 0 Identifier Parity error does not reset LIN state machine. 1 Identifier Parity error reset LIN state machine. This bit can be set/cleared in Initialization mode only.

Table 379. LINCR2 field descriptions(Continued)

Field	Description
WURQ	Wake-up Generation Request Setting this bit generates a wake-up pulse. It is reset by hardware when the wake-up character has been transmitted. The character sent is copied from DATA0 in BDRL buffer. Note that this bit cannot be set in Sleep mode. Software has to exit Sleep mode before requesting a wake-up. Bit error is not checked when transmitting the wake-up request.
DDRQ	Data Discard Request Set by software to stop data reception if the frame does not concern the node. This bit is reset by hardware once LINFlex has moved to idle state. In Slave mode, this bit can be set only when HRF bit in LINSR is set and identifier did not match any filter.
DTRQ	Data Transmission Request Set by software in Slave mode to request the transmission of the LIN Data field stored in the Buffer data register. This bit can be set only when HRF bit in LINSR is set. Cleared by hardware when the request has been completed or aborted or on an error condition. In Master mode, this bit is set by hardware when BIDR[DIR] = 1 and header transmission is completed.
ABRQ	Abort Request Set by software to abort the current transmission. Cleared by hardware when the transmission has been aborted. LINFlex aborts the transmission at the end of the current bit. This bit can also abort a wake-up request. It can also be used in UART mode.
HTRQ	Header Transmission Request Set by software to request the transmission of the LIN header. Cleared by hardware when the request has been completed or aborted. This bit has no effect in UART mode.

22.7.1.14 Buffer identifier register (BIDR)

Offset: 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DFL						DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 406. Buffer identifier register (BIDR)

Table 380. BIDR field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1. Normally, LIN uses only DFL[2:0] to manage frames with a maximum of 8 bytes of data. Identifier filters are compatible with DFL[2:0] only. DFL[5:3] are provided to manage extended frames.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDR registers. 1 LINFlex transmits the data from the BDR registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier. In LIN slave mode (MME bit cleared in LINCR1), this bit must be configured before the header reception. If the slave has to manage frames with 2 types of checksum, filters must be configured.
ID	Identifier Identifier part of the identifier field without the identifier parity.

22.7.1.15 Buffer data register LSB (BDRL)

Offset: 0x0038 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA3								DATA2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA1								DATA0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 407. Buffer data register LSB (BDRL)

Table 381. BDRL field descriptions

Field	Description
DATA3	Data Byte 3 Data byte 3 of the data field.
DATA2	Data Byte 2 Data byte 2 of the data field.
DATA1	Data Byte 1 Data byte 1 of the data field.
DATA0	Data Byte 0 Data byte 0 of the data field.

22.7.1.16 Buffer data register MSB (BDRM)

Offset: 0x003C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA7								DATA6							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DATA5								DATA4							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 408. Buffer data register MSB (BDRM)

Table 382. BDRM field descriptions

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field.
DATA6	Data Byte 6 Data byte 6 of the data field.
DATA5	Data Byte 5 Data byte 5 of the data field.
DATA4	Data Byte 4 Data byte 4 of the data field.

22.7.1.17 Identifier filter enable register (IFER)

Offset: 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FACT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 409. Identifier filter enable register (IFER)

Table 383. IFER field descriptions

Field	Description
FACT	Filter activation The software sets the bit FACT[x] to activate the filters x in identifier list mode. In identifier mask mode bits FACT(2n + 1) have no effect on the corresponding filters as they act as masks for the Identifiers 2n. 0 Filter x is deactivated. 1 Filter x is activated. This field can be set/cleared in Initialization mode only.

22.7.1.18 Identifier filter match index (IFMI)

Address: Base + 0x0044 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	IFMI[0:4]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 410. Identifier filter match index (IFMI)

Table 384. IFMI field descriptions

Field	Description
0:26	Reserved
IFMI[0:4] 27:31	Filter match index This register contains the index corresponding to the received identifier. It can be used to directly write or read the data in SRAM (see *Section 22.8.2.2: Slave mode* for more details). When no filter matches, IFMI[0:4] = 0. When Filter *n* is matching, IFMI[0:4] = *n* + 1.

22.7.1.19 Identifier filter mode register (IFMR)

Offset: 0x0048 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	IFM							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 411. Identifier filter mode register (IFMR)

Table 385. IFMR field descriptions

Field	Description
IFM	Filter mode (see *Table 386*). 0 Filters 2*n* and 2*n* + 1 are in identifier list mode. 1 Filters 2*n* and 2*n* + 1 are in mask mode (filter 2*n* + 1 is the mask for the filter 2*n*).

Table 386. IFMR[IFM] configuration

Bit	Value	Result
IFM[0]	0	Filters 0 and 1 are in identifier list mode.
	1	Filters 0 and 1 are in mask mode (filter 1 is the mask for the filter 0).
IFM[1]	0	Filters 2 and 3 are in identifier list mode.
	1	Filters 2 and 3 are in mask mode (filter 3 is the mask for the filter 2).
IFM[2]	0	Filters 4 and 5 are in identifier list mode.
	1	Filters 4 and 5 are in mask mode (filter 5 is the mask for the filter 4).
IFM[3]	0	Filters 6 and 7 are in identifier list mode.
	1	Filters 6 and 7 are in mask mode (filter 7 is the mask for the filter 6).
IFM[4]	0	Filters 8 and 9 are in identifier list mode.
	1	Filters 8 and 9 are in mask mode (filter 9 is the mask for the filter 8).
IFM[5]	0	Filters 10 and 11 are in identifier list mode.
	1	Filters 10 and 11 are in mask mode (filter 11 is the mask for the filter 10).
IFM[6]	0	Filters 12 and 13 are in identifier list mode.
	1	Filters 12 and 13 are in mask mode (filter 13 is the mask for the filter 12).
IFM[7]	0	Filters 14 and 15 are in identifier list mode.
	1	Filters 14 and 15 are in mask mode (filter 15 is the mask for the filter 14).

22.7.1.20 Identifier filter control register (IFCR2*n*)

Offsets: 0x004C–0x0084 (8 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 412. Identifier filter control register (IFCR2*n*)

Note: Register bit can be read in any mode, written only in Initialization mode

Table 387. IFCR2*n* field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data fields only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity.

22.7.1.21 Identifier filter control register (IFCR2*n* + 1)

Offsets: 0x0050–0x0088 (8 registers) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	DFL			DIR	CCS	0	0	ID					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 413. Identifier filter control register (IFCR2*n* + 1)

Note: Register bit can be read in any mode, written only in Initialization mode

Table 388. IFCR2*n* + 1 field descriptions

Field	Description
DFL	Data Field Length This field defines the number of data bytes in the response part of the frame. DFL = Number of data bytes – 1.
DIR	Direction This bit controls the direction of the data field. 0 LINFlex receives the data and copies them in the BDRL and BDRM registers. 1 LINFlex transmits the data from the BDRL and BDRM registers.

Table 388. IFCR2*n* + 1 field descriptions(Continued)

Field	Description
CCS	Classic Checksum This bit controls the type of checksum applied on the current message. 0 Enhanced Checksum covering Identifier and Data fields. This is compatible with LIN specification 2.0 and higher. 1 Classic Checksum covering Data field only. This is compatible with LIN specification 1.3 and earlier.
ID	Identifier Identifier part of the identifier field without the identifier parity

22.8 Functional description

22.8.1 UART mode

The main features in the UART mode are

- Full duplex communication
- 8- or 9-bit data with parity
- 4-byte buffer for reception, 4-byte buffer for transmission
- 8-bit counter for timeout management

8-bit data frames: The 8th bit can be a data or a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 7 data bits is 1. An odd parity is cleared in this case.

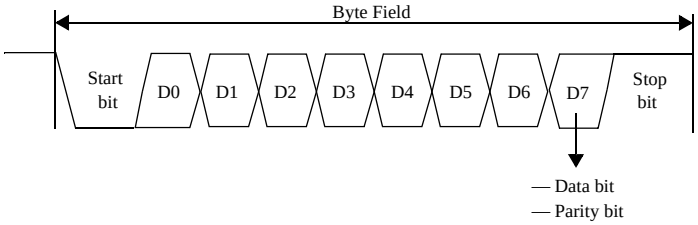


Figure 414. UART mode 8-bit data frame

9-bit frames: The 9th bit is a parity bit. Even/Odd Parity can be selected by the Odd Parity bit in the UARTCR. An even parity is set if the modulo-2 sum of the 8 data bits is 1. An odd parity is cleared in this case.

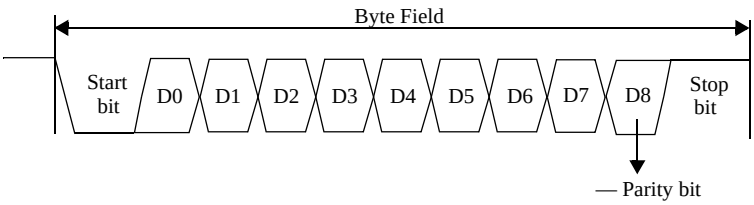


Figure 415. UART mode 9-bit data frame

22.8.1.1 Buffer in UART mode

The 8-byte buffer is divided into two parts: one for receiver and one for transmitter as shown in *Table 389*.

Table 389. Message buffer

Buffer data register	LIN mode		UART mode	
BDRL[0:31]	Transmit/Receive buffer	DATA0[0:7]	Transmit buffer	Tx0
		DATA1[0:7]		Tx1
		DATA2[0:7]		Tx2
		DATA3[0:7]		Tx3
BDRM[0:31]		DATA4[0:7]	Receive buffer	Rx0
		DATA5[0:7]		Rx1
		DATA6[0:7]		Rx2
		DATA7[0:7]		Rx3

22.8.1.2 UART transmitter

In order to start transmission in UART mode, you must program the UART bit and the transmitter enable (TXEN) bit in the UARTCR to 1. Transmission starts when DATA0 (least significant data byte) is programmed. The number of bytes transmitted is equal to the value configured by UARTCR[TDFL] (see *Table 370*).

The Transmit buffer is 4 bytes, hence a 4-byte maximum transmission can be triggered. Once the programmed number of bytes has been transmitted, the UARTSR[DTF] bit is set. If UARTCR[TXEN] is reset during a transmission then the current transmission is completed and no further transmission can be invoked.

22.8.1.3 UART receiver

The UART receiver is active as soon as the user exits Initialization mode and programs UARTCR[RXEN] = 1. There is a dedicated 4-byte data buffer for received data bytes. Once the programmed number (RDFL bits) of bytes has been received, the UARTSR[DRF] bit is set. If the RXEN bit is reset during a reception then the current reception is completed and no further reception can be invoked until RXEN is set.

If a parity error occurs during reception of any byte, then the corresponding PE*x* bit in the UARTSR is set. No interrupt is generated in this case. If a framing error occurs in any byte (UARTSR[FE] = 1) then an interrupt is generated if the LINIER[FEIE] bit is set.

If the last received frame has not been read from the buffer (that is, RMB bit is not reset by the user) then upon reception of the next byte an overrun error occurs (UARTSR[BOF] = 1) and one message will be lost. Which message is lost depends on the configuration of LINCR1[RBLM].

- If the buffer lock function is disabled (LINCR1[RBLM] = 0) the last message stored in the buffer is overwritten by the new incoming message. In this case the latest message is always available to the application.
- If the buffer lock function is enabled (LINCR1[RBLM] = 1) the most recent message is discarded and the previous message is available in the buffer.

An interrupt is generated if the LINIER[BOIE] bit is set.

22.8.1.4 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In UART mode, the LINFlex controller acknowledges a clock gating request once the data transmission and data reception are completed, that is, once the Transmit buffer is empty and the Receive buffer is full.

22.8.2 LIN mode

LIN mode comprises four submodes:

- Master mode
- Slave mode
- Slave mode with identifier filtering
- Slave mode with automatic resynchronization

These submodes are described in the following pages.

22.8.2.1 Master mode

In Master mode the application uses the message buffer to handle the LIN messages. Master mode is selected when the LINCR1[MME] bit is set.

22.8.2.1.1 LIN header transmission

According to the LIN protocol any communication on the LIN bus is triggered by the Master sending a header. The header is transmitted by the Master task while the data is transmitted by the Slave task of a node.

To transmit a header with LINFlex the application must set up the identifier, the data field length and configure the message (direction and checksum type) in the BIDR before requesting the header transmission by setting LINCR2[HTRQ].

22.8.2.1.2 Data transmission (transceiver as publisher)

When the master node is publisher of the data corresponding to the identifier sent in the header, then the slave task of the master has to send the data in the Response part of the LIN frame. Therefore, the application must provide the data to LINFlex before requesting the header transmission. The application stores the data in the message buffer BDR. According to the data field length, LINFlex transmits the data and the checksum. The application uses the BDR[CCS] bit to configure the checksum type (classic or enhanced) for each message.

If the response has been sent successfully, the LINSR[DTF] bit is set. In case of error, the DTF flag is not set and the corresponding error flag is set in the LINESR (see *Section 22.8.2.1.6: Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBEF] bit is set after the first 8 bytes have been transmitted. The application has to update the buffer BDR before resetting the DBEF bit. The transmission of the next bytes starts when the DBEF bit is reset.

After the last data byte (or the checksum byte) has been sent, the DTF flag is set.

The direction of the message buffer is controlled by the BIDR[DIR] bit. When the application sets this bit the response is sent by LINFlex (publisher). Resetting this bit configures the message buffer as subscriber.

22.8.2.1.3 Data reception (transceiver as subscriber)

To receive data from a slave node, the master sends a header with the corresponding identifier. LINFlex stores the data received from the slave in the message buffer and stores the message status in the LINSR.

If the response has been received successfully, the LINSR[DRF] is set. In case of error, the DRF flag is not set and the corresponding error flag is set in the LINESR (see *Section 22.8.2.1.6: Error handling*).

It is possible to handle frames with a Response size larger than 8 bytes of data (extended frames). If the data field length in the BIDR is configured with a value higher than 8 data bytes, the LINSR[DBFF] bit is set once the first 8 bytes have been received. The application has to read the buffer BDR before resetting the DBFF bit. Once the last data byte (or the checksum byte) has been received, the DRF flag is set.

22.8.2.1.4 Data discard

To discard data from a slave, the BIDR[DIR] bit must be reset and the LINCR2[DDRQ] bit must be set before starting the header transmission.

22.8.2.1.5 Error detection

LINFlex is able to detect and handle LIN communication errors. A code stored in the LIN error status register (LINESR) signals the errors to the software.

In Master mode, the following errors are detected:

- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.
- **Response and Frame timeout**: See *Section 22.8.3: 8-bit timeout counter*, for more details.

22.8.2.1.6 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if LINIER[BEIE] = 1.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

22.8.2.1.7 Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting

the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer is overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.

22.8.2.2 Slave mode

In Slave mode the application uses the message buffer to handle the LIN messages. Slave mode is selected when LINCR1[MME] = 0.

22.8.2.2.1 Data transmission (transceiver as publisher)

When LINFlex receives the identifier, the LINSR[HRF] is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR, fill the BDR registers, specify the data field length using the BIDR[DFL] and trigger the data transmission by setting the LINCR2[DTRQ] bit.

One or several identifier filters can be configured for transmission by setting the IFCR*x*[DIR] bit and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in transmission and activated, and if the received identifier matches the filter, a specific TX interrupt (instead of an RX interrupt) is generated.

Typically, the application has to copy the data from SRAM locations to the BDR. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data to the BDR (see *Figure 417*).

Using a filter avoids the software having to configure the direction, the data field length and the checksum type in the BIDR. The software fills the BDR and triggers the data transmission by programming LINCR2[DTRQ] = 1.

If LINFlex cannot provide enough TX identifier filters to handle all identifiers the software has to transmit data for, then a filter can be configured in mask mode (see *Section 22.8.2.3: Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

22.8.2.2.2 Data reception (transceiver as subscriber)

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. The software must read the received identifier in the BIDR and specify the data field length using the BIDR[DFL] field before receiving the stop bit of the first byte of data field.

When the checksum reception is completed, an RX interrupt is generated to allow the software to read the received data in the BDR registers.

One or several identifier filters can be configured for reception by programming IFCR*x*[DIR] = 0 and activated by setting one or several bits in the IFER.

When at least one identifier filter is configured in reception and activated, and if the received identifier matches the filter, an RX interrupt is generated after the checksum reception only.

Typically, the application has to copy the data from the BDR to SRAM locations. To copy the data to the right location, the application has to identify the data by means of the identifier. To avoid this and to ease the access to the SRAM locations, the LINFlex controller provides a Filter Match Index. This index value is the number of the filter that matched the received identifier.

The software can use the index in the IFMI register to directly access the pointer that points to the right data array in the SRAM area and copy this data from the BDR to the SRAM (see *Figure 417*).

Using a filter avoids the software reading the ID value in the BIDR, and configuring the direction, the data field length and the checksum type in the BIDR.

If LINFlex cannot provide enough RX identifier filters to handle all identifiers the software has to receive the data for, then a filter can be configured in mask mode (see *Section 22.8.2.3: Slave mode with identifier filtering*) in order to manage several identifiers with one filter only.

22.8.2.2.3 Data discard

When LINFlex receives the identifier, the LINSR[HRF] bit is set and, if LINIER[HRIE] = 1, an RX interrupt is generated. If the received identifier does not concern the node, you must program LINCR2[DDRQ] = 1. LINFlex returns to idle state after bit DDRQ is set.

22.8.2.2.4 Error detection

In Slave mode, the following errors are detected:

- **Header error**: An error occurred during header reception (Break Delimiter error, Inconsistent Synch Field, Header Timeout).
- **Bit error**: During transmission, the value read back from the bus differs from the transmitted value.
- **Framing error**: A dominant state has been sampled on the stop bit of the currently received character (synch field, identifier field or data field).
- **Checksum error**: The computed checksum does not match the received one.

22.8.2.2.5 Error handling

In case of Bit Error detection during transmission, LINFlex stops the transmission of the frame after the corrupted bit. LINFlex returns to idle state and an interrupt is generated if the BEIE bit in the LINIER is set.

During reception, a Framing Error leads LINFlex to discard the current frame. LINFlex returns immediately to idle state. An interrupt is generated if LINIER[FEIE] = 1.

During reception, a Checksum Error leads LINFlex to discard the received frame. LINFlex returns to idle state. An interrupt is generated if LINIER[CEIE] = 1.

During header reception, a Break Delimiter error, an Inconsistent Synch Field or a Timeout error leads LINFlex to discard the header. An interrupt is generated if LINIER[HEIE] = 1. LINFlex returns to idle state.

22.8.2.2.6 Valid header

A received header is considered as valid when it has been received correctly according to the LIN protocol.

If a valid Break Field and Break Delimiter come before the end of the current header or at any time during a data field, the current header or data is discarded and the state machine synchronizes on this new break.

22.8.2.2.7 Valid message

A received or transmitted message is considered as valid when the data has been received or transmitted without error according to the LIN protocol.

22.8.2.2.8 Overrun

Once the messages buffer is full (LINSR[RMB] = 1) the next valid message reception leads to an overrun and message is lost. The hardware signals the overrun condition by setting the BOF bit in the LINESR. Which message is lost depends on the buffer lock function control bit RBLM.

- If the buffer lock function control bit is cleared (LINCR1[RBLM] = 0) the old message in the buffer will be overwritten by the most recent message.
- If buffer lock function control bit is set (LINCR1[RBLM] = 1) the most recent message is discarded, and the oldest message is available in the buffer.
- If buffer is not released (LINSR[RMB] = 1) before reception of next Identifier and if RBLM is set then ID along with the data is discarded.

22.8.2.3 Slave mode with identifier filtering

In the LIN protocol the identifier of a message is not associated with the address of a node but related to the content of the message. Consequently a transmitter broadcasts its message to all receivers. On header reception a slave node decides—depending on the identifier value—whether the software needs to receive or send a response. If the message does not target the node, it must be discarded without software intervention.

To fulfill this requirement, the LINFlex controller provides configurable filters in order to request software intervention only if needed. This hardware filtering saves CPU resources that would otherwise be needed by software for filtering.

22.8.2.3.1 Filter mode

Usually each of the sixteen IFCR registers filters one dedicated identifier, but this limits the number of identifiers LINFlex can handle to the number of IFCR registers implemented in the device. Therefore, in order to be able to handle more identifiers, the filters can be configured in mask mode.

In **identifier list mode** (the default mode), both filter registers are used as identifier registers. All bits of the incoming identifier must match the bits specified in the filter register.

In **mask mode**, the identifier registers are associated with mask registers specifying which bits of the identifier are handled as “must match” or as “don’t care”. For the bit mapping and registers organization, please see *Figure 416*.

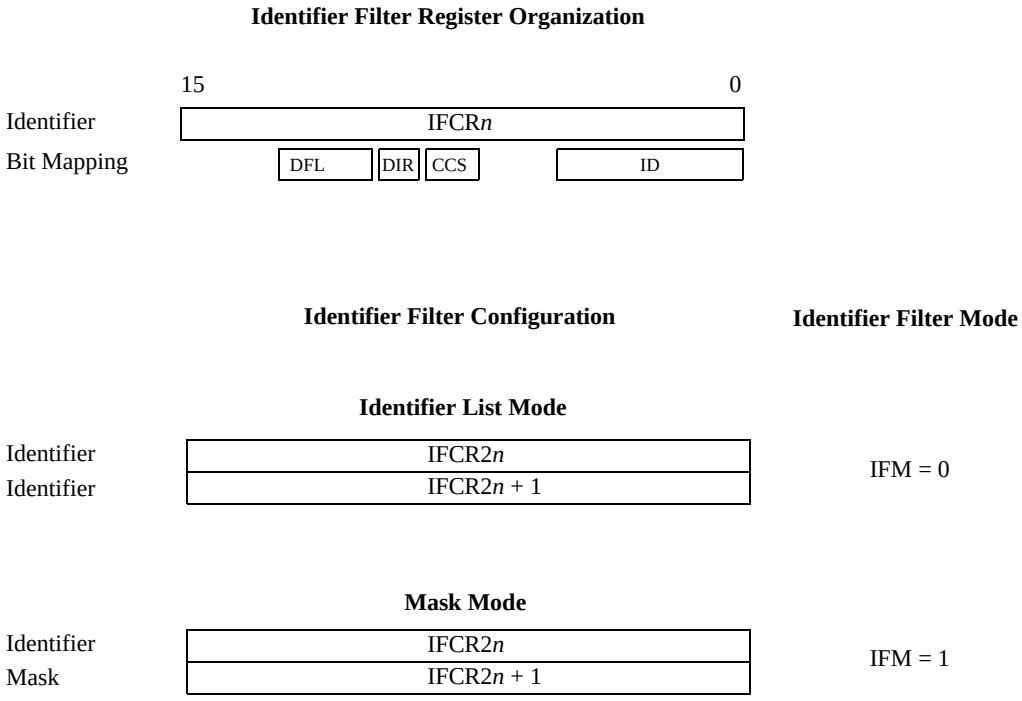


Figure 416. Filter configuration—register organization

22.8.2.3.2 Identifier filter mode configuration

The identifier filters are configured in the IFCR*x* registers. To configure an identifier filter the filter must first be activated by programming IFER[FACT] = 1. The **identifier list** or **identifier mask** mode for the corresponding IFCR*x* registers is configured by the IFMR[IFM] bit. For each filter, the IFCR*x* register configures the ID (or the mask), the direction (TX or RX), the data field length, and the checksum type.

If no filter is active, an RX interrupt is generated on any received identifier event.

If at least one active filter is configured as TX, all received identifiers matching this filter generate a TX interrupt.

If at least one active filter is configured as RX, all received identifiers matching this filter generate an RX interrupt.

If no active filter is configured as RX, all received identifiers not matching TX filter(s) generate an RX interrupt.

Table 390. Filter to interrupt vector correlation

Number of active filters	Number of active filters configured as TX	Number of active filters configured as RX	Interrupt vector
0	0	0	RX interrupt on all identifiers
a (a > 0)	a	0	— TX interrupt on identifiers matching the filters, — RX interrupt on all other identifiers if BF bit is set, no RX interrupt if BF bit is reset
n (n = a + b)	a (a > 0)	b (b > 0)	— TX interrupt on identifiers matching the TX filters, — RX interrupt on identifiers matching the RX filters, — all other identifiers discarded (no interrupt)
b (b > 0)	0	b	— RX interrupt on identifiers matching the filters, — TX interrupt on all other identifiers if BF bit is set, no TX interrupt if BF bit is reset

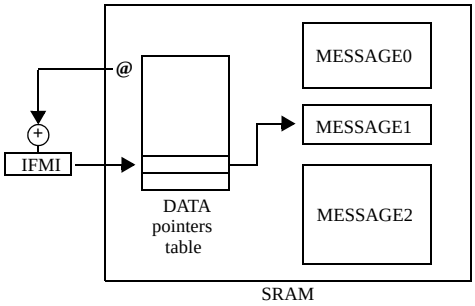


Figure 417. Identifier match index

22.8.2.4 Slave mode with automatic resynchronization

Automatic resynchronization must be enabled in Slave mode if $f_{periph\_set\_1\_clk}$ tolerance is greater than 1.5%. This feature compensates a $f_{periph\_set\_1\_clk}$ deviation up to 14%, as specified in LIN standard.

This mode is similar to Slave mode as described in *Section 22.8.2.2: Slave mode*, with the addition of automatic resynchronization enabled by the LASE bit. In this mode LINFlex adjusts the fractional baud rate generator after each Synch Field reception.

22.8.2.4.1 Automatic resynchronization method

When automatic resynchronization is enabled, after each LIN Break, the time duration between five falling edges on RDI is sampled on $f_{periph\_set\_1\_clk}$ and the result of this measurement is stored in an internal 19-bit register called SM (not user accessible) (see *Figure 418*). Then the LFDIV value (and its associated registers LINIBRR and LINFBRR) is

automatically updated at the end of the fifth falling edge. During LIN Synch Field measurement, the LINFlex state machine is stopped and no data is transferred to the data register.

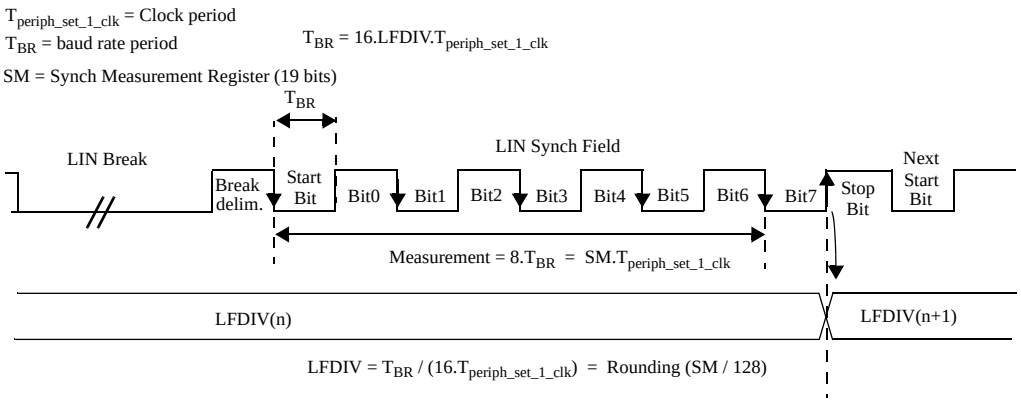


Figure 418. LIN synch field measurement

LFDIV is an unsigned fixed point number. The mantissa is coded on 12 bits in the LINIBRR and the fraction is coded on 4 bits in the LINFBRR.

If LASE bit = 1 then LFDIV is automatically updated at the end of each LIN Synch Field.

Three internal registers (not user-accessible) manage the auto-update of the LINFlex divider (LFDIV):

- LFDIV_NOM (nominal value written by software at LINIBRR and LINFBRR addresses)
- LFDIV_MEAS (results of the Field Synch measurement)
- LFDIV (used to generate the local baud rate)

On transition to idle, break or break delimiter state due to any error or on reception of a complete frame, hardware reloads LFDIV with LFDIV_NOM.

22.8.2.4.2 Deviation error on the Synch Field

The deviation error is checked by comparing the current baud rate (relative to the slave oscillator) with the received LIN Synch Field (relative to the master oscillator). Two checks are performed in parallel.

The first check is based on a measurement between the first falling edge and the last falling edge of the Synch Field:

- If D1 > 14.84%, LHE is set.
- If D1 < 14.06%, LHE is not set.
- If 14.06% < D1 < 14.84%, LHE can be either set or reset depending on the dephasing between the signal on LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

The second check is based on a measurement of time between each falling edge of the Synch Field:

- If D2 > 18.75%, LHE is set.
- If D2 < 15.62%, LHE is not set.
- If 15.62% < D2 < 18.75%, LHE can be either set or reset depending on the dephasing between the signal on LINFlex_RX pin the $f_{periph\_set\_1\_clk}$ clock.

Note that the LINFlex does not need to check if the next edge occurs slower than expected. This is covered by the check for deviation error on the full synch byte.

22.8.2.5 Clock gating

The LINFlex clock can be gated from the Mode Entry module (MC_ME). In LIN mode, the LINFlex controller acknowledges a clock gating request once the frame transmission or reception is completed.

22.8.3 8-bit timeout counter

22.8.3.1 LIN timeout mode

Resetting the LTOM bit in the LINTCSR enables the LIN timeout mode. The LINOCR becomes read-only, and OC1 and OC2 output compare values in the LINOCR are automatically updated by hardware.

This configuration detects header timeout, response timeout, and frame timeout.

Depending on the LIN mode (selected by the LINCR1[MME] bit), the 8-bit timeout counter will behave differently.

LIN timeout mode must not be enabled during LIN extended frames transmission or reception (that is, if the data field length in the BIDR is configured with a value higher than 8 data bytes).

22.8.3.1.1 LIN Master mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO = 28-bit time.

Field OC1 checks T_{Header} and $T_{Response}$ and field OC2 checks T_{Frame} (see *Figure 419*).

When LINFlex moves from Break delimiter state to Synch Field state (see *Section 22.7.1.3: LIN status register (LINSR)*):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + 28),
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + 28 + RTO × 9 (frame timeout value for an 8-byte frame),
- the TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO × 9 (response timeout value for an 8-byte frame)).

On the first response byte is received, OC1 and OC2 are automatically updated to check $T_{Response}$ and T_{Frame} according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or response, the TOCE bit is reset.

If there is no response, frame timeout value does not take into account the DFL value, and an 8-byte response (DFL = 7) is always assumed.

22.8.3.1.2 LIN Slave mode

The LINTOCR[RTO] field can be used to tune response timeout and frame timeout values. Header timeout value is fixed to HTO.

OC1 checks T_{Header} and $T_{Response}$ and OC2 checks T_{Frame} (see *Figure 419*).

When LINFlex moves from Break state to Break Delimiter state (see *Section 22.7.1.3: LIN status register (LINSR)*):

- OC1 is updated with the value of OC_{Header} (OC_{Header} = CNT + HTO),
- OC2 is updated with the value of OC_{Frame} (OC_{Frame} = CNT + HTO + RTO × 9 (frame timeout value for an 8-byte frame)),
- The TOCE bit is set.

On the start bit of the first response data byte (and if no error occurred during the header reception), OC1 is updated with the value of $OC_{Response}$ ($OC_{Response}$ = CNT + RTO × 9 (response timeout value for an 8-byte frame)).

Once the first response byte is received, OC1 and OC2 are automatically updated to check $T_{Response}$ and T_{Frame} according to RTO (tolerance) and DFL.

On the checksum reception or in case of error in the header or data field, the TOCE bit is reset.

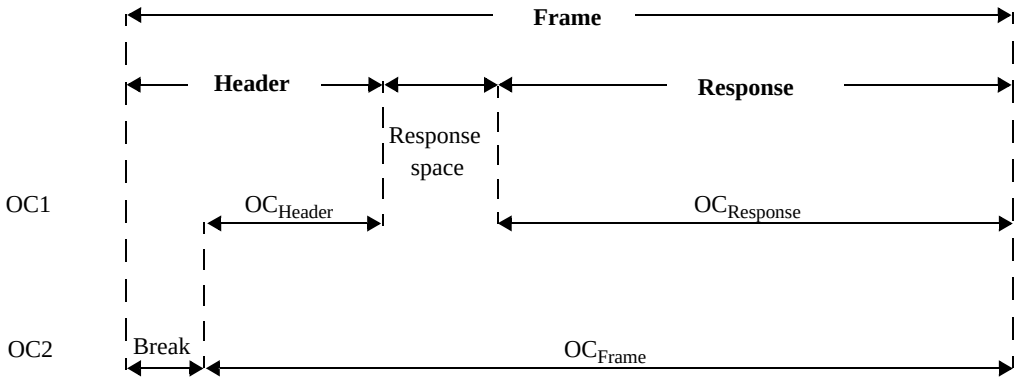


Figure 419. Header and response timeout

22.8.3.2 Output compare mode

Programming LINTCSR[LTOM] = 1 enables the output compare mode. This mode allows the user to fully customize the use of the counter.

OC1 and OC2 output compare values can be updated in the LINTOCR by software.

22.8.4 Interrupts

Table 391. LINFlex interrupt control

Interrupt event	Event flag bit	Enable control bit	Interrupt vector
Header Received interrupt	HRF	HRIE	RXI (1)
Data Transmitted interrupt	DTF	DTIE	TXI
Data Received interrupt	DRF	DRIE	RXI
Data Buffer Empty interrupt	DBEF	DBEIE	TXI
Data Buffer Full interrupt	DBFF	DBFIE	RXI
Wake-up interrupt	WUPF	WUPIE	RXI
LIN State interrupt (2)	LSF	LSIE	RXI
Buffer Overrun interrupt	BOF	BOIE	ERR
Framing Error interrupt	FEF	FEIE	ERR
Header Error interrupt	HEF	HEIE	ERR
Checksum Error interrupt	CEF	CEIE	ERR
Bit Error interrupt	BEF	BEIE	ERR
Output Compare interrupt	OCF	OCIE	ERR
Stuck at Zero interrupt	SZF	SZIE	ERR

1. In Slave mode, if at least one filter is configured as TX and enabled, header received interrupt vector is RXI or TXI depending on the value of identifier received.
2. For debug and validation purposes

23 FlexCAN

23.1 Introduction

The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification. A general block diagram is shown in *Figure 420*, which describes the main subblocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. Support for 32 MBs is provided. The functions of the submodules are described in subsequent sections.

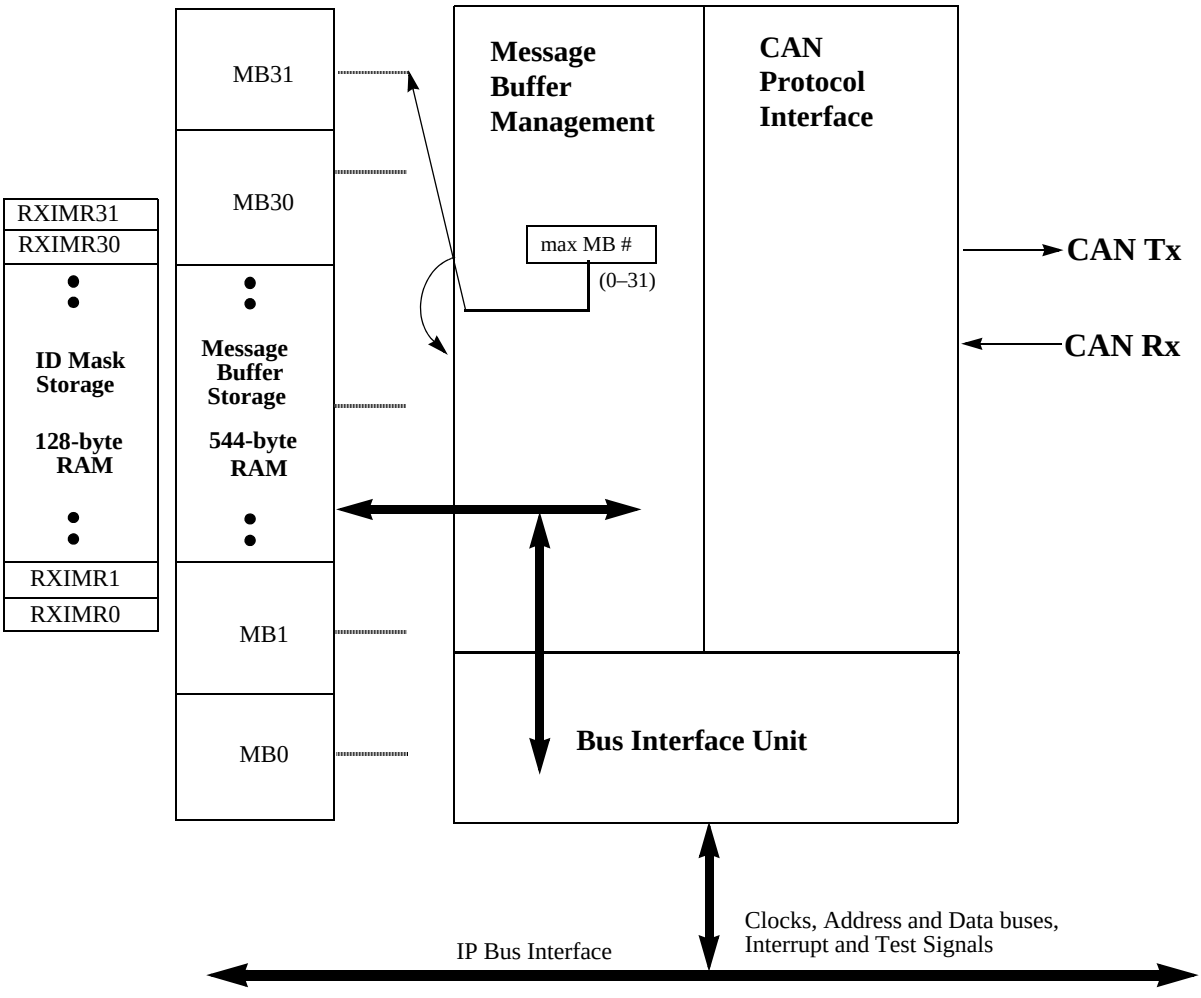


Figure 420. FlexCAN block diagram

23.1.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation

in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B, which supports both standard and extended message frames. 32 Message Buffers are supported. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) submodule manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) submodule handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) submodule controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

One FlexCAN module is implemented on the SPC560P44Lx, SPC560P50Lx device.

23.1.2 FlexCAN module features

The FlexCAN module includes these features:

- Full Implementation of the CAN protocol specification, Version 2.0B
  - Standard data and remote frames
  - Extended data and remote frames
  - 0 to 8 bytes data length
  - Programmable bit rate as high as 1 Mbit/s
  - Content-related addressing
- 32 Flexible Message Buffers (MBs) of 0 to 8 bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes 544 bytes (32 MBs) of RAM used for MB storage
- Includes 128 bytes (32 MBs) of RAM used for individual Rx Mask Registers
- Full-featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard, or 32 partial (8-bit) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen-only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes

23.1.3 Modes of operation

The FlexCAN module has four functional modes: Normal mode (User and Supervisor), Freeze mode, Listen-Only Mode, and Loop-Back mode. There are also these low power modes: Disable mode and Stop mode, and a Test mode.

- Normal mode (User or Supervisor)

  In Normal mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor modes differ in the access to some restricted control registers.

- Freeze mode

  It is enabled when the FRZ bit in the Module Configuration Register (MCR) is asserted. If enabled, Freeze Mode is entered when the HALT bit in the MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of

frames is done and synchronicity to the CAN bus is lost. See *Section 23.4.9.1: Freeze mode* for more information.

- Listen-Only mode

  The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

- Loop-Back mode

  The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

- Module Disable mode

  This low power mode is entered when the MDIS bit in the MCR Register is asserted by the CPU. When disabled, the module requests to disable the clocks to the CAN Protocol Interface and Message Buffer Management submodules. Exit from this mode is done by negating the MDIS bit in the MCR. See *Section 23.4.9.2: Module disable mode* for more information.

- Stop mode

  This low power mode is entered when Stop mode is requested at device level. When in Stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop mode request is removed. See *Section 23.4.9.3: Stop mode* for more information.

23.2 External signal description

23.2.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in *Table 392* and described in more detail in the next subsections.

Table 392. FlexCAN signals

Signal name	Direction	Description
RXD	Input	CAN receive pin
TXD	Output	CAN transmit pin

23.2.2 Signal Descriptions

23.2.2.1 RXD

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

23.2.2.2 TXD

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level 0. Recessive state is represented by logic level 1.

23.3 Memory map and registers description

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the device. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

23.3.1 FlexCAN memory mapping

The complete memory map for a FlexCAN module with 32 MBs capability is shown in *Table 393*. The access type can be Supervisor (S), Test (T) or Unrestricted (U), also called User access. Most of the registers can be configured to have either Supervisor or Unrestricted access by programming the SUPV bit in the MCR.

The Rx Global Mask (RXGMASK), Rx Buffer 14 Mask (RX14MASK) and the Rx Buffer 15 Mask (RX15MASK) registers are provided for backwards compatibility, and are not used when the BCC bit in the MCR is asserted.

The address ranges 0x0060–0x027F and 0x0880–0x08FF are occupied by two separate embedded memories of RAM, of 544 bytes and 128 bytes, respectively. When the FlexCAN is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the BCC bit in the MCR is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the address range 0x0880–0x097F is considered reserved space, independent of the value of the BCC bit.

Table 393. FlexCAN module memory map

Offset from FlexCAN_BASE 0xFFFC_0000	Register	Location
0x0000	Module Configuration Register (MCR)	*on page 718*
0x0004	Control Register (CTRL)	*on page 722*
0x0008	Free Running Timer (TIMER)	*on page 725*
0x000C	Reserved	
0x0010	Rx Global Mask (RXGMASK)	*on page 726*
0x0014	Rx Buffer 14 Mask (RX14MASK)	*on page 727*
0x0018	Rx Buffer 15 Mask (RX15MASK)	*on page 727*
0x001C	Error Counter Register (ECR)	*on page 728*
0x0020	Error and Status Register (ESR)	*on page 729*
0x0024	Reserved	
0x0028	Interrupt Masks 1 (IMASK1)	*on page 732*
0x002C	Reserved	
0x0030	Interrupt Flags 1 (IFLAG1)	*on page 733*
0x0034–0x005F	Reserved	
0x0060–0x007F	Serial Message Buffers (SMB0–SMB1) – Reserved	—
0x0080–0x017F	Message Buffers MB0–MB15	*on page 713*
0x0180–0x027F	Message Buffers MB16–MB31	*on page 713*
0x0280–0x087F	Reserved	
0x0880–0x08BF	Rx Individual Mask Registers RXIMR0–RXIMR15	*on page 734*
0x08C0–0x08FF	Rx Individual Mask Registers RXIMR16–RXIMR31	*on page 734*
0x0900–0x3FFF	Reserved	

Table 394. FlexCAN register reset status

Register	Affected by hard reset	Affected by soft reset
Module Configuration Register (MCR)	Yes	Yes
Control Register (CTRL)	Yes	No
Free Running Timer (TIMER)	Yes	Yes
Reserved		
Rx Global Mask (RXGMASK)	Yes	No
Rx Buffer 14 Mask (RX14MASK)	Yes	No
Rx Buffer 15 Mask (RX15MASK)	Yes	No
Error Counter Register (ECR)	Yes	Yes
Error and Status Register (ESR)	Yes	Yes

Table 394. FlexCAN register reset status(Continued)

Register	Affected by hard reset	Affected by soft reset
Interrupt Masks 1 (IMASK1)	Yes	Yes
Interrupt Flags 1 (IFLAG1)	Yes	Yes
Serial Message Buffers (SMB0–SMB1) – Reserved	No	No
Message Buffers MB0–MB15	No	No
Message Buffers MB16–MB31	No	No
Rx Individual Mask Registers RXIMR0–RXIMR15	No	No
Rx Individual Mask Registers RXIMR16–RXIMR31	No	No

The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped on memory as described in *Table 395*. The module also implements two additional Message Buffers called SMB0 and SMB1 (Serial Message Buffers), in the address ranges 0x60–0x6F and 0x70–0x7F, which are not accessible to the end user. They are used for temporary storage of frames during the reception and transmission processes. *Table 395* shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 395. Message Buffer MB0 memory mapping

Address offset	MB field
0x0080	Control and Status (C/S)
0x0084	Identifier Field
0x0088–0x008F	Data Field 0 – Data Field 7 (1 byte each)

23.3.2 Message buffer structure

The Message Buffer structure used by the FlexCAN module is represented in *Table 421*. Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

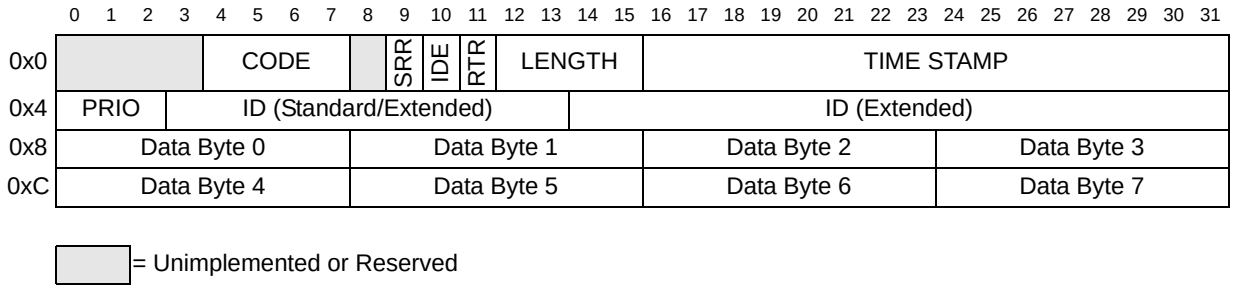


Figure 421. Message buffer structure

Table 396. Message Buffer structure field description

Field	Description
CODE	Message Buffer Code This 4-bit field can be accessed (read or write) by the CPU and by the Flexcan module itself, as part of the message buffer matching and arbitration process. The encoding is shown in *Table 397* and *Table 398*. See *Section 23.4: Functional description* for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to 1 by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames. 1 Recessive value is compulsory for transmission in Extended Format frames.
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as 1 (recessive) and receives it as 0 (dominant), it is interpreted as arbitration loss. If this bit is transmitted as 0 (dominant), then if it is received as 1 (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a Data Frame to be transmitted 1 Indicates the current MB has a Remote Frame to be transmitted
LENGTH	Length of Data in Bytes This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see *Table 421*). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.
TIME STAMP	Free-Running Counter Time Stamp This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
PRIO	Local priority This 3-bit field is only used when LPRIO_EN bit is set in MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See *Section 23.4.3: Arbitration process*.
ID	Frame Identifier In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field As many as 8 bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 397. Message buffer code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	–	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to *Section 23.4.5: Matching process* for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to *Section 23.4.5: Matching process* for details about overrun behavior.
0XY1(1)	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

1. Note that for Tx MBs (see *Table 398*), the BUSY bit should be ignored upon read, except when AEN bit is set in the MCR.

Table 398. Message Buffer code for Tx buffers

RTR	Initial Tx code	Code after successful transmission	Description
X	1000	–	INACTIVE: MB does not participate in the arbitration process.
X	1001	–	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

Table 399. MB0–MB31 addresses

Address	Register	Address	Register
Base + 0x0080	MB0	Base + 0x0180	MB16
Base + 0x0090	MB1	Base + 0x0190	MB17
Base + 0x00A0	MB2	Base + 0x01A0	MB18
Base + 0x00B0	MB3	Base + 0x01B0	MB19
Base + 0x00C0	MB4	Base + 0x01C0	MB20
Base + 0x00D0	MB5	Base + 0x01D0	MB21
Base + 0x00E0	MB6	Base + 0x01E0	MB22
Base + 0x00F0	MB7	Base + 0x01F0	MB23
Base + 0x0100	MB8	Base + 0x0200	MB24
Base + 0x0110	MB9	Base + 0x0210	MB25
Base + 0x0120	MB10	Base + 0x0220	MB26
Base + 0x0130	MB11	Base + 0x0230	MB27
Base + 0x0140	MB12	Base + 0x0240	MB28

Table 399. MB0–MB31 addresses(Continued)

Address	Register	Address	Register
Base + 0x0150	MB13	Base + 0x0250	MB29
Base + 0x0160	MB14	Base + 0x0260	MB30
Base + 0x0170	MB15	Base + 0x0270	MB31

23.3.3 Rx FIFO structure

When the FEN bit is set in the MCR, the memory area from 0x80 to 0xFF (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. *Figure 422* shows the Rx FIFO data structure. The region 0x80–0x8F contains an MB structure that is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90–0xDF is reserved for internal use of the FIFO engine. The region 0xE0–0xFF contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. *Table 400* shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See *Section 23.4.7: Rx FIFO* for more information.

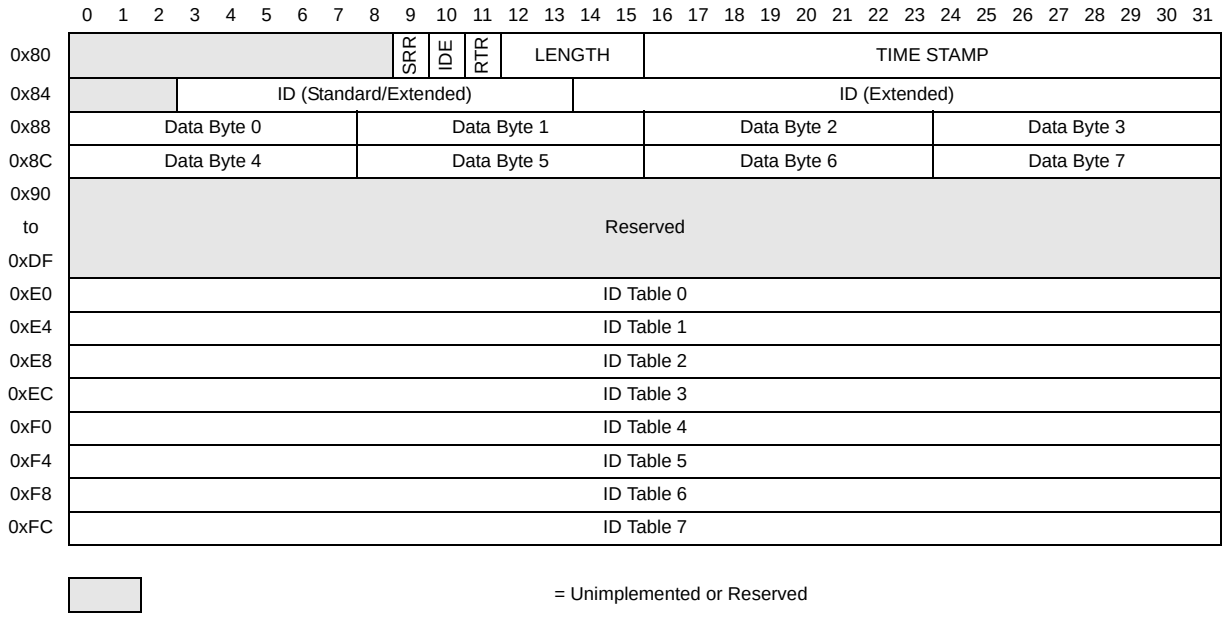


Figure 422. Rx FIFO structure

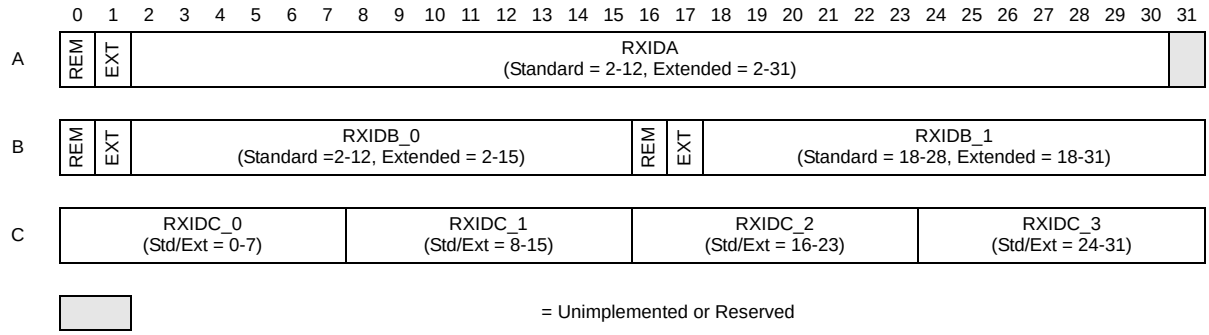


Table 400. ID Table 0 - 7

Table 401. Rx FIFO Structure field description

Field	Description
REM	Remote Frame This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 0 Remote Frames are rejected and data frames can be accepted. 1 Remote Frames can be accepted and data frames are rejected.
EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted. 1 Extended frames can be accepted and standard frames are rejected.
RXIDA	Rx Frame Identifier (Format A) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13) are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0 RXIDB_1	Rx Frame Identifier (Format B) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13) are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0 RXIDC_1 RXIDC_2 RXIDC_3	Rx Frame Identifier (Format C) Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

23.3.4 Registers description

The FlexCAN registers are described in this section in ascending address order.

23.3.4.1 Module Configuration Register (MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Address: Base + 0x0000 Access: User read/write

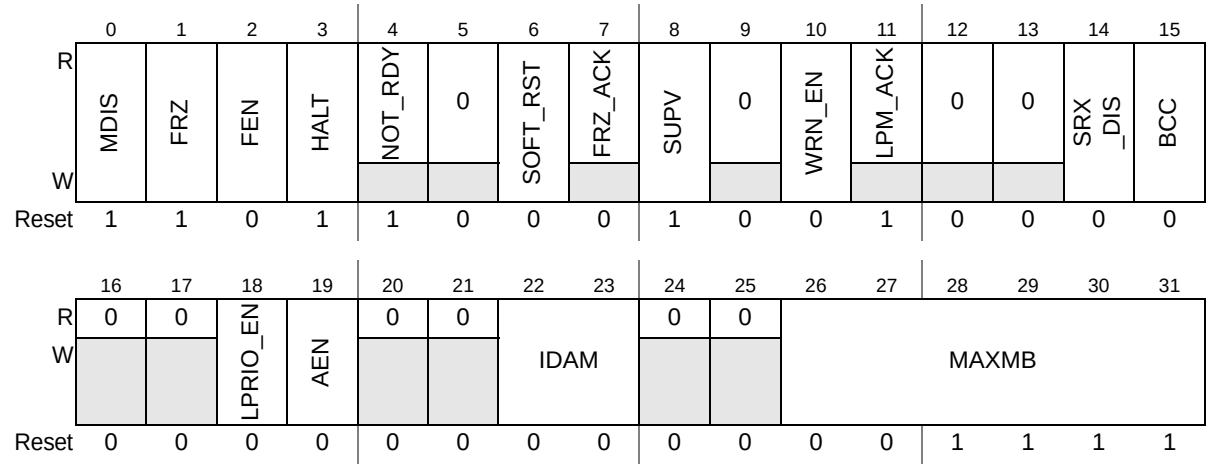


Figure 423. Module Configuration Register (MCR)

Table 402. MCR field descriptions

Field	Description
0 MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management submodules. This is the only bit in MCR not affected by soft reset. See *Section 23.4.9.2: Module disable mode* for more information. 0 Enable the FlexCAN module. 1 Disable the FlexCAN module.
1 FRZ	Freeze Enable The FRZ bit specifies the FlexCAN behavior when the HALT bit in the MCR is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is able to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 0 Not able to enter Freeze Mode. 1 Able to enter Freeze Mode.
2 FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80–0xFF) is used by the FIFO engine. See *Section 23.3.3: Rx FIFO structure* and *Section 23.4.7: Rx FIFO* for more information. 0 FIFO disabled. 1 FIFO enabled.
3 HALT	Halt FlexCAN Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See *Section 23.4.9.1: Freeze mode* for more information. 0 No Freeze Mode request. 1 Enters Freeze Mode if the FRZ bit is asserted.

Table 402. MCR field descriptions(Continued)

Field	Description
4 NOT_RDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode. 1 FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode.
6 SOFT_RST	Soft Reset When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: MCR (except the MDIS bit), TIMER, ECR, ESR, IMASK1, IFLAG1. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: – CTRL – RXIMR0–RXIMR31 – RXGMASK, RX14MASK, RX15MASK – all Message Buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the MCR, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 0 No reset request. 1 Resets the registers marked as “affected by soft reset” in *Table 393*.
7 FRZ_ACK	Freeze Mode Acknowledge This read-only bit indicates that FlexCAN is in Freeze mode and its prescaler is stopped. The Freeze mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See *Section 23.4.9.1: Freeze mode* for more information. 0 FlexCAN not in Freeze mode, prescaler running. 1 FlexCAN in Freeze mode, prescaler stopped.
8 SUPV	Supervisor Mode This bit configures some of the FlexCAN registers to be either in Supervisor or Unrestricted memory space. The registers affected by this bit are marked as S/U in the Access Type column of *Table 393*. The reset value of this bit is 1, so the affected registers start with Supervisor access restrictions. 0 Affected registers are in Unrestricted memory space. 1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location.
10 WRN_EN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥ 96.

Table 402. MCR field descriptions(Continued)

Field	Description
11 LPM_ACK	Low Power Mode Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the LPM_ACK bit to know when FlexCAN has actually entered low power mode. See *Section 23.4.9.2: Module disable mode* and *Section 23.4.9.3: Stop mode* for more information. 0 FlexCAN not in any of the low power modes. 1 FlexCAN is either in Disable Mode or Stop mode.
14 SRX_DIS	Self Reception Disable This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 0 Self reception enabled 1 Self reception disabled
15 BCC	Backwards Compatibility Configuration This bit is provided to support Backwards Compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: – For MCUs supporting individual Rx ID masking, this feature is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK and RX15MASK. – The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to ‘0110’ (overrun). Upon reset this bit is negated, allowing legacy software to work without modification. 0 Individual Rx masking and queue feature are disabled. 1 Individual Rx masking and queue feature are enabled.
18 LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It extends the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. 0 Local Priority disabled 1 Local Priority enabled
19 AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 0 Abort disabled 1 Abort enabled

Table 402. MCR field descriptions(Continued)

Field	Description
22–23 IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in *Table 403*. Note that all elements of the table are configured at the same time by this field (they are all the same format). See *Section 23.3.3: Rx FIFO structure*.
26–31 MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze Mode. **Maximum MBs in use = MAXMB + 1** **Note:** MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages. Maximum number of MBs is 32 => MAXMB = 6b011111 (do not use higher values).

Table 403. IDAM coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

23.3.4.2 Control Register (CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 424. Control Register (CTRL)

Table 404. CTRL field descriptions

Field	Description
0–7 PRESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to *Section 23.4.8.4: Protocol timing*. **Sclock frequency = CPI clock frequency / (PRESDIV + 1).**
8–9 RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta(1) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3. **Resync Jump Width = RJW + 1.**
10–12 PSEG1	PSEG1 — Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. **Phase Buffer Segment 1(PSEG1 + 1) x Time-Quanta.**
13–15 PSEG2	PSEG2 — Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. **Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.**
16 BOFF_MSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 0 Bus Off interrupt disabled. 1 Bus Off interrupt enabled.
17 ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled. 1 Error interrupt enabled.
18 CLK_SRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the peripheral clock (driven by the PLL) or the crystal oscillator clock. The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See *Section 23.4.8.4: Protocol timing* for more information. 0 The CAN engine clock source is the oscillator clock. 1 The CAN engine clock source is the bus clock. **Note:** This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, this bit has no effect on the module operation.

Table 404. CTRL field descriptions(Continued)

Field	Description
19 LPB	Loop Back This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic 1). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loop Back disabled. 1 Loop Back enabled.
20 TWRN_MSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Tx Warning Interrupt disabled. 1 Tx Warning Interrupt enabled.
21 RWRN_MSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in MCR is negated and it is read as zero when WRN_EN is negated. 0 Rx Warning Interrupt disabled. 1 Rx Warning Interrupt enabled.
24 SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx input. 0 Just one sample determines the bit value. 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and two preceding samples, a majority rule is used.
25 BOFF_REC	Bus Off Recovery Mode This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will resynchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery. 0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B. 1 Automatic recovering from Bus Off state disabled.
26 TSYN	Timer Sync Mode This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (that is, global network time). If the FEN bit in MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer Sync feature disabled 1 Timer Sync feature enabled

Table 404. CTRL field descriptions(Continued)

Field	Description
27 LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first. 1 Lowest number buffer is transmitted first.
28 LOM	Listen-Only Mode This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 Listen Only Mode is deactivated. 1 FlexCAN module operates in Listen Only Mode.
29–31 PROPSEG	Propagation Segment This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) × Time-Quanta. Time-Quantum = one Sclock period.

1. One time quantum is equal to the Sclock period.

23.3.4.3 Free Running Timer (TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TIMER															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 425. Free Running Timer (TIMER)

Table 405. TIMER field descriptions

Field	Description
TIMER	Holds the value for this timer.

23.3.4.4 Rx Global Mask register (RXGMASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RXGMASK register to have no effect on the module operation. For MCUs not supporting individual masks per MB, this register is always effective.

RXGMASK is used as an acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in the MCR is set (FIFO enabled), the RXGMASK also applies to all elements of the ID filter table, except elements 6–7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 426. Rx Global Mask register (RXGMASK)

Table 406. RXGMASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

23.3.4.5 Rx 14 Mask (RX14MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RX14MASK register to have no effect on the module operation.

RX14MASK is used as an acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in the MCR is set (FIFO enabled), the RX14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 427. Rx Buffer 14 Mask register (RX14MASK)

Table 407. RX14MASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

23.3.4.6 Rx 15 Mask (RX15MASK)

This register is provided for legacy support and for low cost MCUs that do not have the individual masking per Message Buffer feature. For MCUs supporting individual masks per MB, setting the BCC bit in the MCR causes the RX15MASK register to have no effect on the module operation.

When the BCC bit is negated, RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in the MCR is set (FIFO enabled), the RX15MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 428. Rx Buffer 15 Mask register (RX15MASK)

Table 408. RX15MASK field description

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

23.3.4.7 Error Counter Register (ECR)

This register has two 8-bit fields that reflect the value of two FlexCAN error counters:

- Transmit Error Counter (Tx_Err_Counter field)
- Receive Error Counter (Rx_Err_Counter field)

The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Test Mode or Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g., transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any

influence on the bus when in ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to 0.
- If FlexCAN is in ‘Bus Off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to 0 and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the FLT_CONF field in the Error and Status Register is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.
- If during system start-up, only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in the Error and Status Register). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Rx_Err_Counter								Tx_Err_Counter							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 429. Error Counter Register (ECR)

23.3.4.8 Error and Status Register (ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16–21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16–21. Bits 22–28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT, and ERR_INT, that are interrupt flags that can be cleared by writing 1 to them (writing 0 has no effect). See *Section 23.4.10: Interrupts* for more details.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 430. Error and Status Register (ESR)

Table 409. Error and Status Register (ESR) field description

Field	Description
14 TWRN_INT	Tx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from 0 to 1, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Tx error counter transitioned from < 96 to ≥ 96.
15 RWRN_INT	Rx Warning Interrupt Flag If the WRN_EN bit in MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from 0 to 1, meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 The Rx error counter transitioned from < 96 to ≥ 96.
16 BIT1_ERR	Bit1 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as recessive is received as dominant. **Note:** This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.
17 BIT0_ERR	Bit0 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence. 1 At least one bit sent as dominant is received as recessive.

Table 409. Error and Status Register (ESR) field description(Continued)

Field	Description
18 ACK_ERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, that is, a dominant bit has not been detected during the ACK SLOT. 0 No such occurrence. 1 An ACK error occurred since last read of this register
19 CRC_ERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, that is, the calculated CRC is different from the received. 0 No such occurrence. 1 A CRC error occurred since last read of this register.
20 FRM_ERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, that is, a fixed-form bit field contains at least one illegal bit. 0 No such occurrence. 1 A Form Error occurred since last read of this register.
21 STF_ERR	Stuffing Error This bit indicates that a Stuffing Error has been detected. 0 No such occurrence. 1 A Stuffing Error occurred since last read of this register.
22 TX_WRN	TX Error Counter This bit indicates when repetitive errors are occurring during message transmission. 0 No such occurrence. 1 TX_Err_Counter $\geq$ 96.
23 RX_WRN	Rx Error Counter This bit indicates when repetitive errors are occurring during message reception. 0 No such occurrence. 1 Rx_Err_Counter $\geq$ 96.
24 IDLE	CAN bus IDLE state This bit indicates when the CAN bus is in IDLE state. 0 No such occurrence. 1 CAN bus is now IDLE.
25 TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE = 0). 1 FlexCAN is transmitting a message (IDLE = 0).
26–27 FLT_CONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown in *Table 410*. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted.

Table 409. Error and Status Register (ESR) field description(Continued)

Field	Description
29 BOFF_INT	Bus Off Interrupt This bit is set when FlexCAN enters ‘Bus Off’ state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1. Writing 0 has no effect. 0 No such occurrence. 1 FlexCAN module entered ‘Bus Off’ state.
30 ERR_INT	Error Interrupt This bit indicates that at least one of the Error Bits (bits 16–21) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to 1.Writing 0 has no effect. 0 No such occurrence. 1 Indicates setting of any Error Bit in the Error and Status Register.

Table 410. Fault confinement state

Value	Meaning
00	Error Active
01	Error Passive
1X	Bus Off

23.3.4.9 Interrupt Masks 1 Register (IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (that is, when the corresponding IFLAG1 bit is set).

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 431. Interrupt Masks 1 Register (IMASK1)

Table 411. IMASK1 field descriptions

Field	Description
0–31 BUF31M – BUF0M	BUF31M–BUF0M — Buffer MB_i Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 0 The corresponding buffer Interrupt is disabled. 1 The corresponding buffer Interrupt is enabled. **Note:** Setting or clearing a bit in the IMASK1 Register can assert or negate an interrupt request, if the corresponding IFLAG1 bit is set.

23.3.4.10 Interrupt Flags 1 Register (IFLAG1)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding IFLAG1 bit. If the corresponding IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to 1. Writing 0 has no effect.

When the AEN bit in the MCR is set (Abort enabled), while the IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the MCR is set (FIFO enabled), the function of the eight least significant interrupt flags (BUF7I – BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 432. Interrupt Flags 1 Register (IFLAG1)

Table 412. IFLAG1 field descriptions

Field	Description
0–23 BUF31I – BUF8I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 0 No such occurrence. 1 The corresponding MB has successfully completed transmission or reception.
24 BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence. 1 MB7 completed transmission/reception or FIFO overflow.

Table 412. IFLAG1 field descriptions(Continued)

Field	Description
25 BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 4 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence. 1 MB6 completed transmission/reception or FIFO almost full.
26 BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence. 1 MB5 completed transmission/reception or frames available in the FIFO.
27–31 BUF4I – BUF0I	Buffer MB_i Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence. 1 Corresponding MB completed transmission/reception.

23.3.4.11 Rx Individual Mask Registers (RXIMR0–RXIMR31)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (FEN bit in MCR is set), the first 8 Mask Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the BCC bit in the MCR is negated, any read or write operation to these registers results in access error.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK and RX15MASK registers are available, regardless of the value of the BCC bit.

Address: See *Table 414* Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 433. Rx Individual Mask Registers (RXIMR0–RXIMR31)

Table 413. RXIMR0–RXIMR31 field descriptions

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care.” 1 The corresponding bit in the filter is checked against the one received.

Table 414. RXIMR0–RXIMR31 addresses

Address	Register	Address	Register
Base + 0x0880	RXIMR0	Base + 0x08C0	RXIMR16
Base + 0x0884	RXIMR1	Base + 0x08C4	RXIMR17
Base + 0x0888	RXIMR2	Base + 0x08C8	RXIMR18
Base + 0x088C	RXIMR3	Base + 0x08CC	RXIMR19
Base + 0x0890	RXIMR4	Base + 0x08D0	RXIMR20
Base + 0x0894	RXIMR5	Base + 0x08D4	RXIMR21
Base + 0x0898	RXIMR6	Base + 0x08D8	RXIMR22
Base + 0x089C	RXIMR7	Base + 0x08DC	RXIMR23
Base + 0x08A0	RXIMR8	Base + 0x08E0	RXIMR24
Base + 0x08A4	RXIMR9	Base + 0x08E4	RXIMR25
Base + 0x08A8	RXIMR10	Base + 0x08E8	RXIMR26
Base + 0x08AC	RXIMR11	Base + 0x08EC	RXIMR27
Base + 0x08B0	RXIMR12	Base + 0x08F0	RXIMR28
Base + 0x08B4	RXIMR13	Base + 0x08F4	RXIMR29
Base + 0x08B8	RXIMR14	Base + 0x08F8	RXIMR30
Base + 0x08BC	RXIMR15	Base + 0x08FC	RXIMR31

23.4 Functional description

23.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of as many as 32 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see *Section 23.3.2: Message buffer structure*). The memory corresponding to the first eight Message Buffers can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (as many as 8 extended IDs or 16 standard IDs or 32 eight-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it

possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a 0000 code is inactive (refer to *Table 397*). Similarly, a Tx MB with a 1000 or 1001 code is also inactive (refer to *Table 398*). An MB not programmed with 0000, 1000, or 1001 will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see *Section 23.4.6.2: Message Buffer deactivation*).

23.4.2 Transmit process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

1. If the message buffer is active (transmission pending), write ‘1000’ to the Code field to inactivate the message buffer. The deactivated message buffer can transmit without setting IFLAG and without updating the CODE field. (see *Section 23.4.6.2: Message Buffer deactivation*).
2. Write the ID word.
3. Write the data bytes.
4. Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step 4 (see *Table 397* and *Table 398* in *Section 23.3.2: Message buffer structure*). When the Abort feature is enabled (AEN in MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

23.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[(d)] or the lowest MB number or the

d. Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

highest priority, depending on the LBUF and LPRIO_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame
- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in the MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits as many as 8 data bytes, even if the DLC (Data Length Code) value is bigger.

23.4.4 Receive process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

1. If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see *Section 23.4.6.1: Transmission abort mechanism*). If backwards compatibility is desired (AEN in MCR negated), just write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see *Section 23.4.6.2: Message Buffer deactivation*). If the MB already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.
2. Write the ID word
3. Write ‘0100’ to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see *Table 397* and *Table 398* in *Section 23.3.2: Message buffer structure*)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (mandatory – activates an internal lock for this buffer)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see *Section 23.4.6: Data coherence*).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in *Table 397*. If the CPU tries to work around this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: **never perform polling by directly reading the C/S word of the MBs. Instead, read the IFLAG registers**.

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see *Section 23.4.7: Rx FIFO*). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

1. Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)
2. Read the ID field (optional – needed only if a mask was used)
3. Read the Data field
4. Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

23.4.5 Matching process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see *Section 23.4.6.3: Message Buffer lock mechanism*)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to *Table 397* and *Table 398*). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see *Section 23.4.6.3: Message Buffer lock mechanism*).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the BCC bit in the MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to *Section 23.3.4.11: Rx Individual Mask Registers (RXIMR0–RXIMR31)*. During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is

"don't care". Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the BCC bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, RX14MASK and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the BCC bit in the MCR is negated.

Note: The individual Rx Mask per Message Buffer feature may not be available in low cost MCUs. Please consult the specific MCU documentation to find out if this feature is supported. If not supported, the RXGMASK, RX14MASK, and RX15MASK registers are available, regardless of the value of the BCC bit.

23.4.6 Data coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in *Section 23.4.2: Transmit process* and *Section 23.4.4: Receive process*. Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

23.4.6.1 Transmission abort mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

1. CPU writes 1001 into the code field of the C/S word.
2. CPU reads the CODE field and compares it to the value that was written.
3. If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE = 1001) or it was transmitted (CODE = 1000).

Note: An abort request to a TxMB can block any write operation into its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a transmission by winning the CAN bus arbitration.

23.4.6.2 Message Buffer deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.
- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in *Section 23.4.6.1: Transmission abort mechanism* should be used.

23.4.6.3 Message Buffer lock mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an “active not empty” Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and

thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

Note: The locking mechanism only applies to Rx MBs that have a code different than INACTIVE ('0000') or EMPTY[e] ('0100'). Also, Tx MBs can not be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no "free to receive" MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

Note: If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

23.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see *Section 23.3.3: Rx FIFO structure*). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store as many as six frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by

e. In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. In current FlexCAN versions, this behavior is maintained when the BCC bit is negated.

reading one or more frames. A warning interrupt is also generated when 4frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also *Section 23.3.3: Rx FIFO structure*):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

Note: A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0–RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the BCC bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

23.4.7.1 Precautions when using Global Mask and Individual Mask registers

Mask filtering alignment is affected based on the setting of the FEN and BCC of MCR. *Table 415* table shows recommended actions depending on FEN and BCC settings.

Table 415. Recommended FEN and BCC settings

Case	MCR[FEN] RxFIFO	MCR[BCC] Rx Individual Mask	Notes
Case 1	FEN = 0	BCC = 0	RXGMASK, RX14MASK, and RX15MASK can safely be used. This allows backwards compatibility to older devices (e.g., devices without the individual masks feature). In this case, individual masks are not used.
Case 2	FEN = 1	BCC = 0	1st alternative: Do not use RXGMASK, RX14MASK, and RX15MASK in this case, leave the masks in their reset state.
Case 3	FEN = 1	BCC = 0	2nd alternative: Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive). In this case, RXGMASK, RX14MASK, and RX15MASK can be used to affect ID Tables without affecting the filtering process for Rx MBs.
Case 4	Don’t care	BCC = 1	If MCR[BCC] = 1, then the RXIMRs are enabled. Thus, RXGMASK, RX14MASK, and RX15MASK are not used. Particularly, when MCR[FEN] = 0, RxFIFO is disabled; RXGMASK, RX14MASK, and RX15MASK do not affect filtering. Individual masks are used.

23.4.8 CAN protocol related features

23.4.8.1 Remote frames

Remote frame is a special kind of frame. The user can program a MB to be a Request Remote Frame by writing the MB as Transmit with the RTR bit set to 1. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by FlexCAN, its ID is compared to the IDs of the transmit message buffers with the Code field '1010'. If there is a matching ID, then this MB frame will be transmitted. Note that if the matching MB has the RTR bit set, then FlexCAN will transmit a Remote Frame as a response.

A received Remote Request Frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request Frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

If the Rx FIFO is enabled (bit FEN set in MCR), FlexCAN will not generate an automatic response for Remote Request Frames that match the FIFO filtering criteria. If the remote frame matches one of the target IDs, it will be stored in the FIFO and presented to the CPU. Note that for filtering formats A and B, it is possible to select whether remote frames are accepted or not. For format C, remote frames are always accepted (if they match the ID).

23.4.8.2 Overload frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission
- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

23.4.8.3 Time stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of "move-in" in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in *Section 23.3.4.2: Control Register (CTRL)*.

23.4.8.4 Protocol timing

Figure 434 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral Clock (generally from a PLL). In order to guarantee reliable operation,

the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

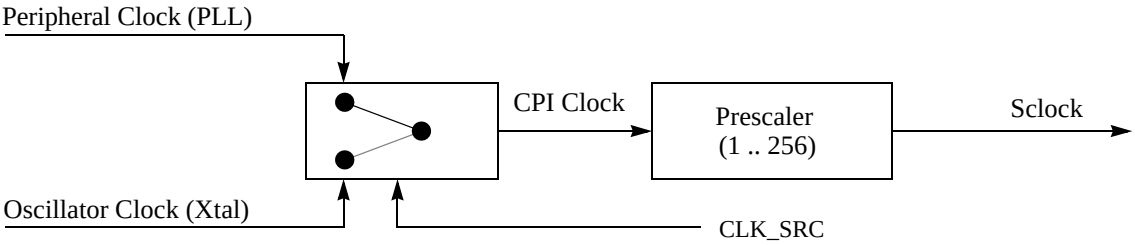


Figure 434. CAN engine clocking scheme

The crystal oscillator clock should be selected whenever a tight tolerance (to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

Note: This clock selection feature may not be available in all MCUs. A particular MCU may not have a PLL, in which case it would have only the oscillator clock, or it may use only the PLL clock feeding the FlexCAN module. In these cases, the CLK_SRC bit in the CTRL Register has no effect on the module operation.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See *Section 23.3.4.2: Control Register (CTRL)*.

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the 'time quantum' used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

Equation 62

$$f_{Tq} = \frac{f_{CANCLK}}{(\text{Prescaler value})}$$

A bit time is subdivided into three segments[f] (reference *Figure 435* and *Table 416*):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section.
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CTRL Register (plus 1) to be 2 to 8 time quanta long.

f. For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Equation 63

$$\text{Bit Rate} = \frac{f_{Tq}}{(\text{number of Time Quanta})}$$

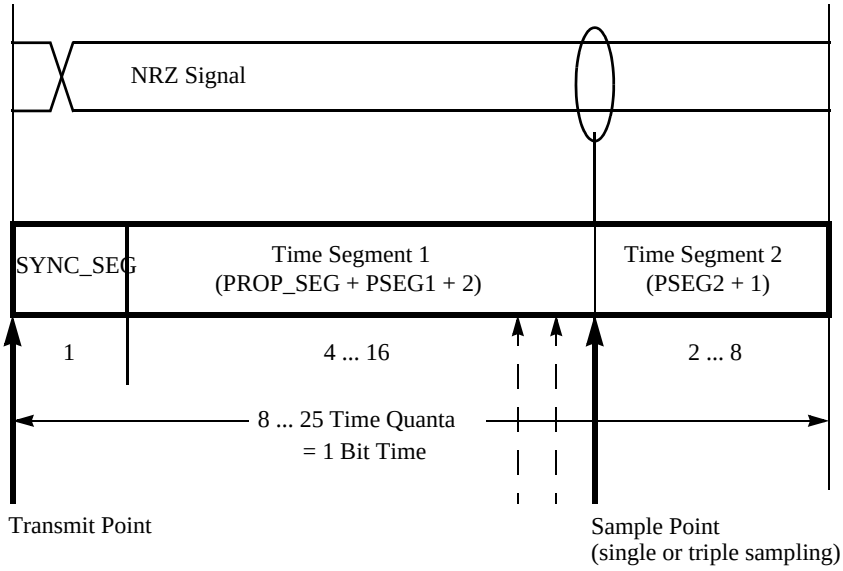


Figure 435. Segments within the bit time

Table 416. Time segment syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 417 gives an overview of the CAN compliant segment settings and the related parameter values.

Table 417. CAN standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4

Table 417. CAN standard compliant bit time segment settings

Time Segment 1	Time Segment 2	Resynchronization Jump Width
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

Note: It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

23.4.8.5 Arbitration and matching timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in *Figure 436*.

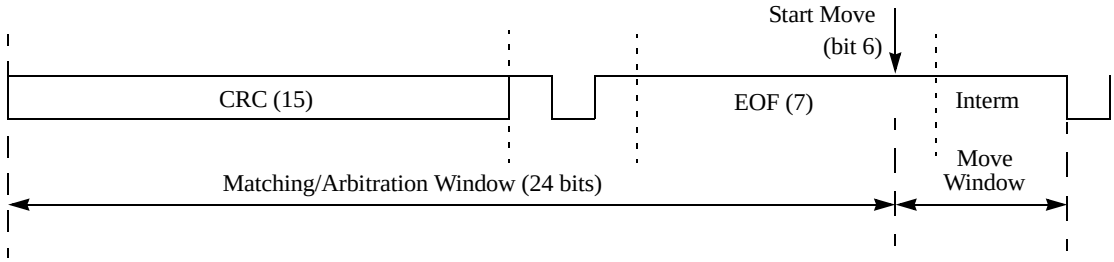


Figure 436. Arbitration, match, and move time windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in *Table 417*
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, that is, the PLL can not be programmed to divide down the oscillator clock
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in *Table 418*

Table 418. Minimum ratio between peripheral clock frequency and CAN bit rate

Number of message buffers	Minimum ratio
16	8
32	8

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in *Table 418* can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1,

PSEG2). As an example, taking the case of 32 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to 1 and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

23.4.9 Modes of operation details

23.4.9.1 Freeze mode

This mode is entered by asserting the HALT bit in the MCR or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the MCR and the module is not in any of the low power modes. When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in MCR

After requesting Freeze Mode, the user must wait for the FRZ_ACK bit to be asserted in the MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in the MCR
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to resynchronize to the CAN bus by waiting for 11 consecutive recessive bits.

23.4.9.2 Module disable mode

This low power mode is entered when the MDIS bit in the MCR Register is asserted by the CPU. When the FlexCAN module is disabled during Freeze Mode, the module sends a request to disable the clocks to the CPI and MBM submodules, sets CAN_MCR[MDISACK] and negates CAN_MCR[FRZACK]. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and LPM_ACK bits in MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register, and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the LPM_ACK bit.

23.4.9.3 Stop mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the LPM_ACK bit, negates the FRZ_ACK bit and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and LPM_ACK bits in MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Stop Mode is exited by the CPU resuming the clocks and removing the Stop Mode request.

23.4.10 Interrupts

The module can generate as many as 38 interrupt sources (32 interrupts due to message buffers and 6 interrupts due to ORed interrupts from MBs, Bus Off, Error, Tx Warning, Rx Warning, and Wake Up). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to 1 (unless another interrupt is generated at the same time).

Note: It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags that are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the IFLAG1 becomes the FIFO Overflow flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the Frames Available in FIFO flag and bits 4:0 are unused. See *Section 23.3.4.10: Interrupt Flags 1 Register (IFLAG1)* for more information.

A combined interrupt for all MBs is also generated by an OR of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG Registers to determine which MB caused the interrupt.

The other five interrupt sources (Bus Off, Error, Tx Warning, Rx Warning, and Wakeup) generate interrupts like the MB ones, and can be read from the ESR register. The Bus Off, Error, Tx Warning, and Rx Warning interrupt mask bits are located in the CTRL register, and the Wake-Up interrupt mask bit is located in the MCR.

23.4.11 Bus interface

The CPU access to FlexCAN registers is subject to the following rules:

- All reads and writes to test registers must be qualified with ips_test_access signal. Read and write access to test mode registers in non test mode results in access error.
- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the BCC bit in the MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 32 MBs and MAXMB is programmed with 0. The maximum number of MBs in this case becomes 1. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x027F. The available memory in the Mask Registers space would be from 0x0884 to 0x08FF.

Note: Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

23.5 Initialization/application information

This section provide instructions for initializing the FlexCAN module.

23.5.1 FlexCAN initialization sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to *Table 393* to see which registers are affected by soft reset)
- SOFT_RST bit in MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is unsynchronized to the CAN bus, the HALT and FRZ bits in the MCR are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the MCR are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see *Section 23.4.9.1: Freeze mode*). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the BCC bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)
- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in CTRL Register (for Bus Off and Error interrupts) and in MCR for Wake-Up interrupt
- Negate the HALT bit in MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

24 Analog-to-Digital Converter (ADC)

24.1 Overview

24.1.1 Device-specific features

- 2 ADC units
  - 26 input channels (2 × 11 plus 4 shared)
  - ADC_0: channel 15 dedicated for the internal 1.2 V rail
  - Channels 11 to 14 shared between the two converters
- 10-bit resolution
- Conversion time < 1 µs including sampling time at full precision (conversion time target of 700 ns for the analog section)
- Cross triggering unit (CTU)
- 4 analog watchdogs with interrupt capability for continuous hardware monitoring of as many as 4 analog input channels
- Clock stretching (with CTU pulse)
- Sampling and conversion time register CTR0 (internal precision channels)
- Left-aligned result format
- Right-aligned result format
- One Shot/Scan Modes
- Chain Injection Mode
- Power-down mode
- 2 different Abort functions allow aborting either single-channel conversion or chain conversion
- As many as 16 data registers for storing converted data. Conversion information, such as mode of operation (normal, injected or CTU), is associated to data value.
- Auto-clock-off
- 2 modes of operation, each with DMA compatible interface
  - Normal Mode
  - CTU Control Mode

These features are absent on the device:

- Support of external channels
- Presampling
- Alternate analog thresholds
- Offset Cancellation and Offset Refresh Control
- External start and triggering

24.1.2 Device-specific pin configuration features

- For *Section 24.3.3: ADC sampling and conversion timing*, f_{ck} = (1/2) MC_PLL_CLK is true where the bit ADCLKSEL would be always 0 (default value), meaning that AD_clk is half of MC_PLL_CLK. A clock prescaler (1 or 2) can be configured. The AD_clk has

the same frequency of MC_PLL_CLK or is half of MC_PLL_CLK, depending on the value of the bit ADCLKSEL.

- CTUEN field in the MCR: Enables or disables CTU control mode
- Registers CDR[16..95] not used

24.1.3 Device-specific implementation

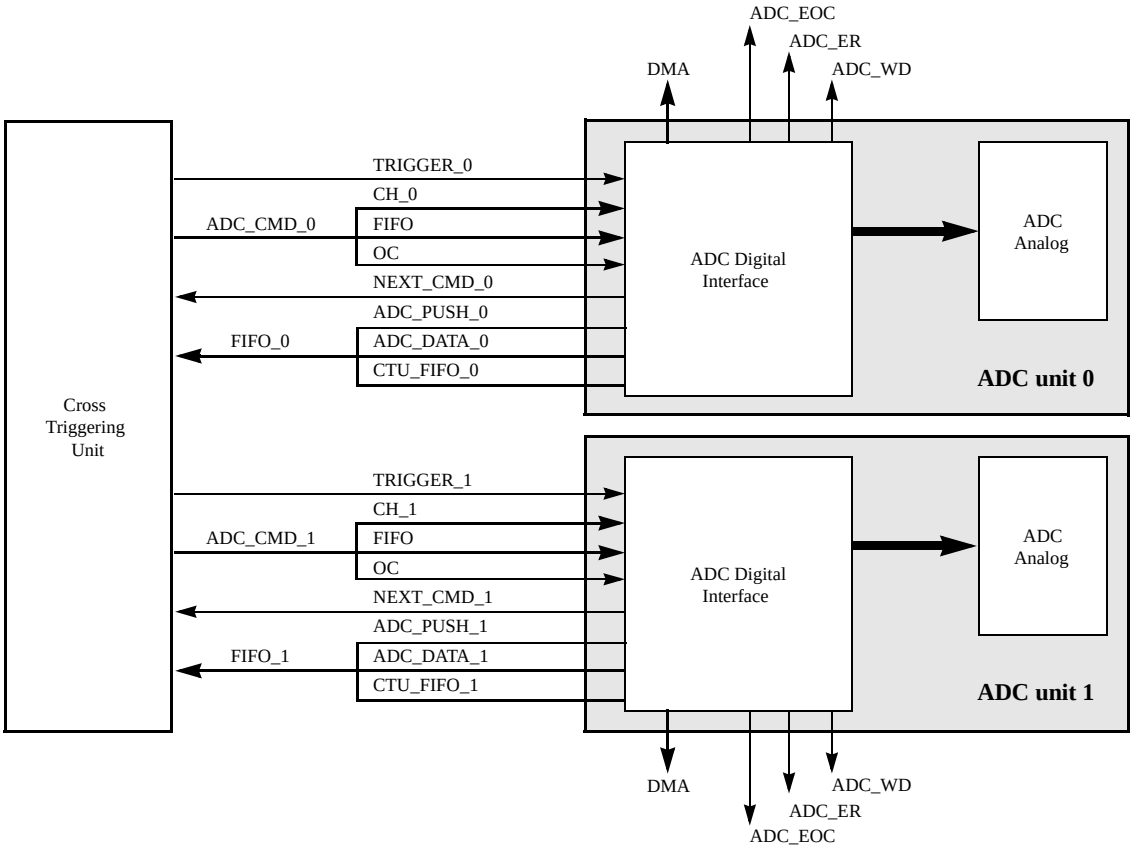


Figure 437. ADC implementation

24.2 Introduction

The analog-to-digital converter (ADC) block provides accurate and fast conversions for a wide range of applications.

The ADC contains advanced features for normal or injected conversion. It provides support for eDMA (direct memory access) mode operation. A conversion can be triggered by software or hardware (Cross Triggering Unit or PIT).

The mask registers present within the ADC can be programmed to configure the channel to be converted.

A conversion timing register for configuring different sampling and conversion times is associated to each channel type.

Analog watchdogs allow continuous hardware monitoring.

24.3 Functional description

24.3.1 Analog channel conversion

Two conversion modes are available within the ADC:

- Normal conversion
- Injected conversion

24.3.1.1 Normal conversion

This is the normal conversion that the user programs by configuring the normal conversion mask registers (NCMR). Each channel can be individually enabled by setting '1' in the corresponding field of NCMR registers. Mask registers must be programmed before starting the conversion and cannot be changed until the conversion of all the selected channels ends (NSTART bit in the Main Status Register (MSR) is reset).

24.3.1.2 Start of normal conversion

By programming the configuration bits in the Main Configuration Register (MCR), the normal conversion can be started in two ways:

- By software — The conversion chain starts when the MCR[NSTART] bit is set.
- By trigger — An on-chip internal signal triggers an ADC conversion. The settings in the MCR select how conversions are triggered based on these internal signals:
  - A rising/falling edge detected in the signal sets the MSR[NSTART] bit and starts the programmed conversion.
  - The conversion is started if and only if the MCR[NSTART] bit is set and the programmed level on the trigger signal is detected.

Table 419. Configurations for starting normal conversion

Type of conversion start	NSTART (in MCR)	NSTART (in MSR)	Result
Software	1	1	Conversion chain starts
Trigger	—	1	A falling or rising edge detected in a trigger signal sets the NSTART bit in the MSR and starts the programmed conversion.
	1	1	The conversion is started if the programmed level on the trigger signal is detected: the start of conversion is enabled if the external pin is low or high.

The MSR[NSTART] status bit is automatically set when the normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

If the content of all the normal conversion mask registers is zero (that is, no channel is selected) the conversion operation is considered completed and the interrupt ECH (see interrupt controller chapter for further details) is immediately issued after the start of conversion.

24.3.1.3 Normal conversion operating modes

Two operating modes are available for the normal conversion:

- One Shot
- Scan

To enter one of these modes, it is necessary to program the MCR[MODE] bit. The first phase of the conversion process involves sampling the analog channel and the next phase involves the conversion phase when the sampled analog value is converted to digital as shown in *Figure 438*.

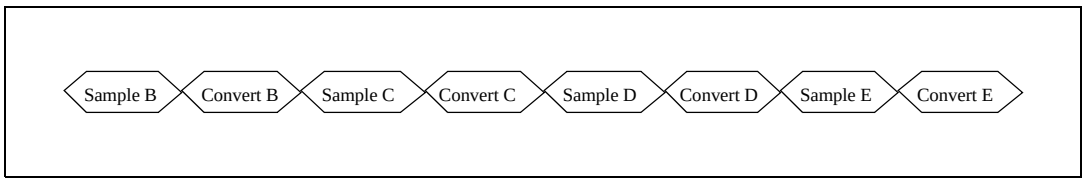


Figure 438. Normal conversion flow

In **One Shot Mode** (MODE = 0) a sequential conversion specified in the NCMR registers is performed only once. At the end of each conversion, the digital result of the conversion is stored in the corresponding data register.

Example 1. One Shot Mode (MODE = 0)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the One Shot Mode. MODE = 0 is set for One Shot mode. Conversion starts from the channel B followed by conversion of channels D-E. At the end of conversion of channel E the scanning of channels stops.

The NSTART status bit in the MSR is automatically set when the Normal conversion starts. At the same time the MCR[NSTART] bit is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running conversion is completed.

In **Scan Mode** (MODE = 1), a sequential conversion of N channels specified in the NCMR registers is continuously performed. As in the previous case, at the end of each conversion the digital result of the conversion is stored into the corresponding data register.

The MSR[NSTART] status bit is automatically set when the Normal conversion starts. Unlike One Shot Mode, the MCR[NSTART] bit is not reset. It can be reset by software when the user needs to stop scan mode. In that case, the ADC completes the current scan conversion and, after the last conversion, also resets the MSR[NSTART] bit.

Example 2. Scan Mode (MODE = 1)

Channels A-B-C-D-E-F-G-H are present in the device where channels B-D-E are to be converted in the Scan Mode. MODE = 1 is set for Scan Mode. Conversion starts from the channel B followed by conversion of the channels D-E. At the end of conversion of channel E the scanning of channel B starts followed by conversion of the channels D-E. This sequence repeats itself till the MCR[NSTART] bit is cleared by software.

At the end of each conversion an End Of Conversion interrupt is issued (if enabled by the corresponding mask bit) and at the end of the conversion sequence an End Of Chain interrupt is issued (if enabled by the corresponding mask bit in the IMR register).

24.3.1.4 Injected channel conversion

A conversion chain can be injected into the ongoing normal conversion by configuring the Injected Conversion Mask Registers (JCMR). As with normal conversion, each channel can be selected individually. This injected conversion (which can occur only in One Shot mode) interrupts the normal conversion (which can be in One Shot or Scan mode). When an injected conversion is inserted, ongoing normal channel conversion is aborted and the injected channel request is processed. After the last channel in the injected chain is converted, normal conversion resumes from the channel at which the normal conversion was aborted, as shown in *Figure 439*.

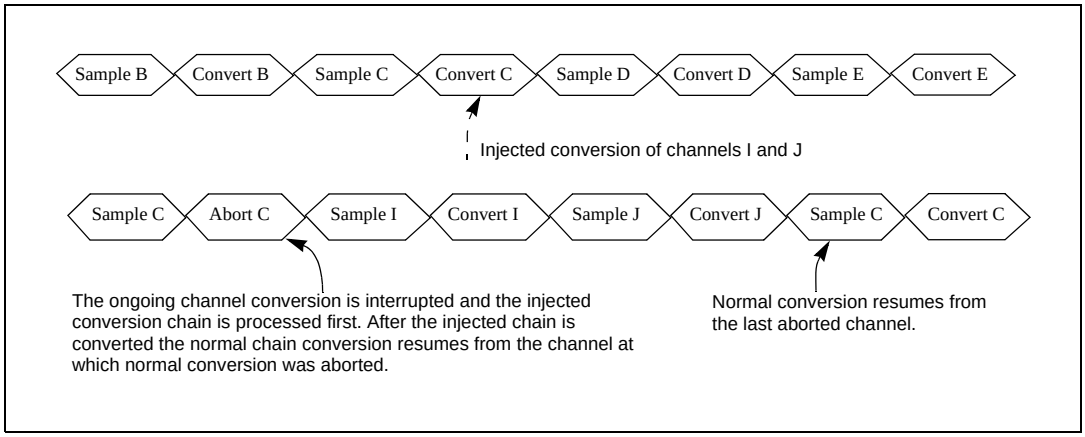


Figure 439. Injected sample/conversion sequence

The injected conversion can be started using two options:

- By software setting the MCR[JSTART]; the current conversion is suspended and the injected chain is converted. At the end of the chain, the JSTART bit in the MSR is reset and the normal chain conversion is resumed.
- By an internal trigger signal from the PIT when MCR[JTRGEN] is set; a programmed event (rising/falling edge depending on MCR[JEDGE]) on the signal coming from PIT or CTU starts the injected conversion by setting the MSR[JSTART]. At the end of the chain, the MSR[JSTART] is cleared and the normal conversion chain is resumed.

The MSR[JSTART] is automatically set when the Injected conversion starts. At the same time the MCR[JSTART] is reset, allowing the software to program a new start of conversion. In that case the new requested conversion starts after the running injected conversion is completed.

At the end of each injected conversion, an End Of Injected Conversion (JEOC) interrupt is issued (if enabled by the IMR[MSKJEOC]) and at the end of the sequence an End Of Injected Chain (JECH) interrupt is issued (if enabled by the IMR[MSKJEOC]).

If the content of all the injected conversion mask registers (JCMR) is zero (that is, no channel is selected) the JECH interrupt is immediately issued after the start of conversion.

Once started, injected chain conversion cannot be interrupted by any other conversion type (it can, however, be aborted; see *Section 24.3.1.5: Abort conversion*).

24.3.1.5 Abort conversion

Two different abort functions are provided.

- The user can abort the ongoing conversion by setting the MCR[ABORT] bit. The current conversion is aborted and the conversion of the next channel of the chain is immediately started. In the case of an abort operation, the NSTART/JSTART bit remains set and the ABORT bit is reset after the conversion of the next channel starts. The EOC interrupt corresponding to the aborted channel is not generated. This behavior is true for normal or triggered/Injected conversion modes. If the last channel of a chain is aborted, the end of chain is reported generating an ECH interrupt.
- It is also possible to abort the current chain conversion by setting the MCR[ABORTCHAIN] bit. In that case the behavior of the ADC depends on the MODE bit. If scan mode is disabled, the NSTART bit is automatically reset together with the MCR[ABORTCHAIN] bit. Otherwise, if the scan mode is enabled, a new chain conversion is started. The EOC interrupt of the current aborted conversion is not generated but an ECH interrupt is generated to signal the end of the chain.

  When a chain conversion abort is requested (ABORTCHAIN bit is set) while an injected conversion is running over a suspended Normal conversion, both injected chain and Normal conversion chain are aborted (both the NSTART and JSTART bits are also reset).

24.3.2 Analog clock generator and conversion timings

The clock frequency can be selected by programming the MCR[ADCLKSEL]. When this bit is set to ‘1’ the ADC clock has the same frequency as the MC_PLL_CLK. Otherwise, the ADC clock is half of the MC_PLL_CLK frequency. The ADCLKSEL bit can be written only in power-down mode.

When the internal divider is not enabled (ADCCLKSEL = 1), it is important that the associated clock divider in the clock generation module is ‘1’. This is needed to ensure 50% clock duty cycle.

The direct clock should basically be used only in low power mode when the device is using only the 16 MHz fast internal RC oscillator, but the conversion still requires a 16 MHz clock (an 8 MHz clock is not fast enough).

In all other cases, the ADC should use the clock divided by two internally.

Depending on the position of the rising edge of the signal internal trigger signal coming from the CTU, the ADC clock could also be stretched as illustrated in *Figure 440*.

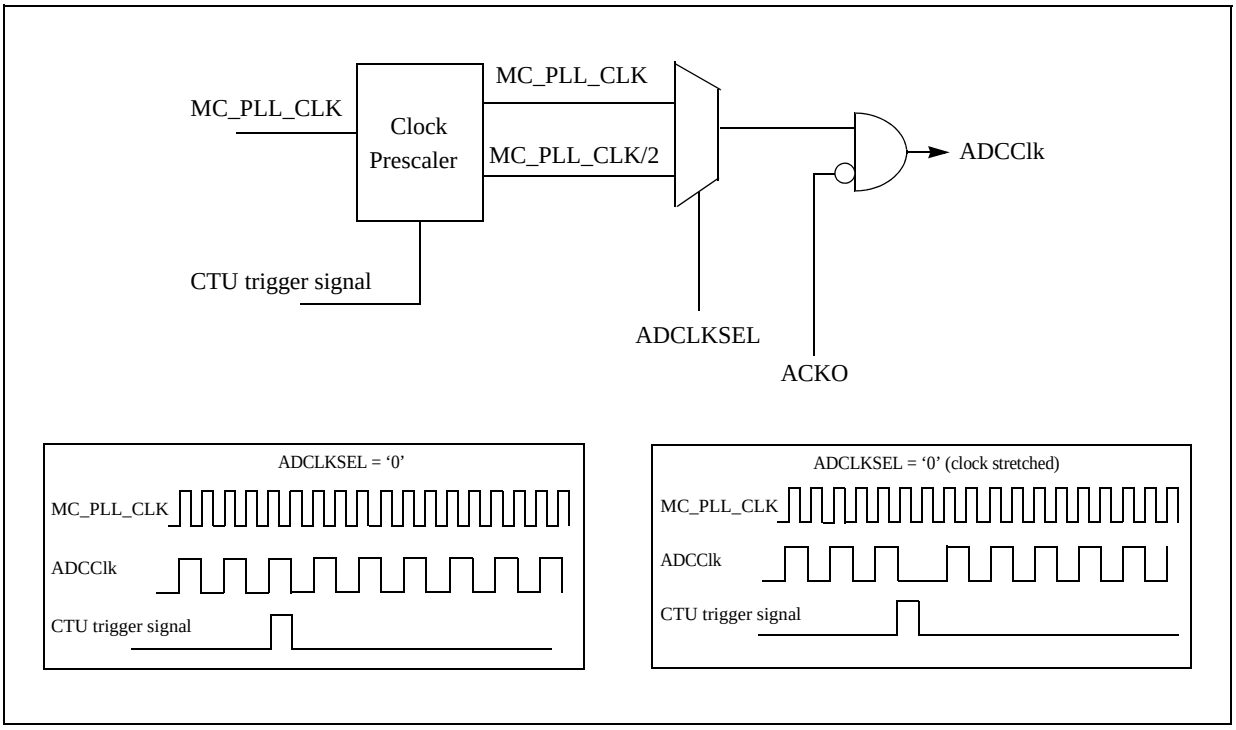


Figure 440. Prescaler simplified block diagram

The clock stretching is implemented if and only if ADCLKSEL = 0 (and clock is half of the MC_PLL_CLK).

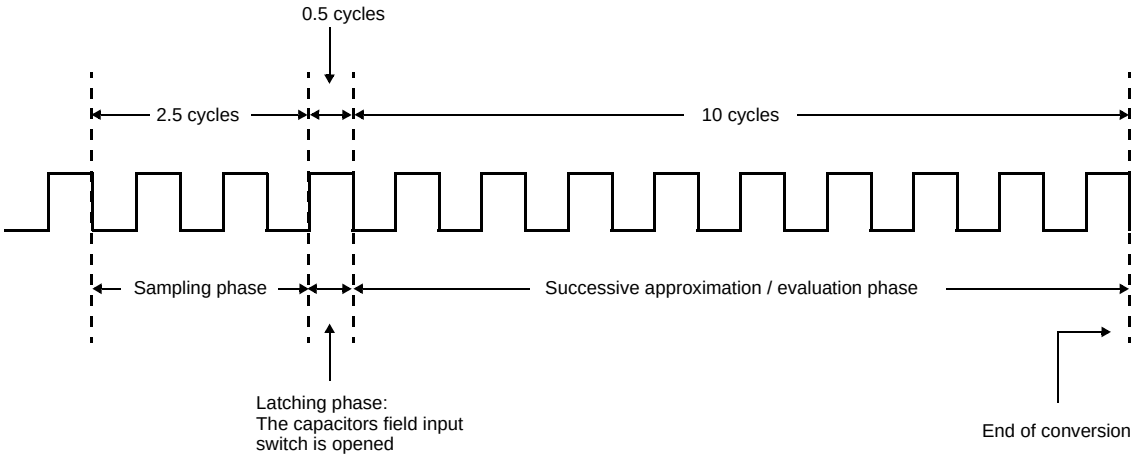
24.3.3 ADC sampling and conversion timing

In order to support different loading and switching times, several different Conversion Timing registers (CTR) are present. There is one register per channel type. INPLATCH and INPCMP configurations are limited when the system clock frequency is greater than 20 MHz.

When a conversion is started, the ADC connects the internal sampling capacitor to the respective analog input pin, allowing the capacitance to charge up to the input voltage value. The time to load the capacitor is referred to as sampling time. After completion of the sampling phase, the evaluation phase starts and all the bits corresponding to the resolution of the ADC are estimated to provide the conversion result.

The conversion times are programmed via the bit fields of the CTR. Bit fields INPLATCH, INPCMP, and INPSAMP define the total conversion duration (T_{conv}) and in particular the partition between sampling phase duration (T_{sample}) and total evaluation phase duration (T_{eval}).

Figure 441 represents the sampling and conversion sequence.



Note: Operating conditions — INPLATCH = 0, INPSAMP = 3, INPCMP = 1 and Fadc clk = 20 MHz

Figure 441. Sampling and conversion timings

In the following equation, the unit Tck refers to the reciprocal of the motor control clock which is then modified by value of the MCR.ADCCLKSEL field:

T_{CK} = (2 - MCR.ADCCLKSEL) * T_{MOTC_CLK}

The sampling phase duration is:

$$T_{SAMPLE} = (INPSAMPLES - nDELAY) \bullet T_{ck}$$

$$INPSAMPLES \geq 3$$

INPSAMPLES must be greater or equal to 8 (hardware requirement). If INPSAMPLES is < 3, the sampling time remains as if INPSAMPLES = 3.

nDELAY = 0.5 if INPSAMPLES <= 6, = 1 otherwise

The total evaluation phase duration is:

$$T_{eval} = 10 \bullet T_{biteval}$$

INPCMP must be greater than or equal to 1 and INPLATCH must be less than INCMP (hardware requirements).

Table 420. Relation between INPCMP and $T_{biteval}$

INPCMP	$T_{biteval}$
00/01	$1*T_{CK}$
10	$2*T_{CK}$
11	$3*T_{CK}$

Note: 1. The Tsample and Teval must respect the minimum value specified in the Data Sheet to allow ADC to reach TUE performance.

2. For Tbiteval/INPCMP explanation, please refer to Table 420: Relation between INPCMP and Tbiteval.

The total conversion duration T_{conv} is (not including external multiplexing) the time to perform the total evaluation:

$$T_{CONV} = T_{SAMPLE} + T_{EVAL} + T_{CK} + T_{DIGSYNC} + T_{CTUSYNC}$$

where

$T_{DIGSYNC}$ = (1 + MCR.ADCCLKSEL) * T_{CK}

Table 421. Relation between MCR.CTUEN, MCR.ADCCLKSEL, and $T_{CTUSYNC}$

MCR.CTUEN	MCR.ADCCLKSEL	$T_{CTUSYNC}$
0	-	0
1	0	0.5*T_{CK}
1	1	2*T_{CK}

Table 422. Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V

INPCMP	INPLATCH	Max f_{ADC_clk}	Min f_{ADC_clk}
00/01	0	20 +/- 4%	3
	1	—	—
10	0	—	—
	1	40 +/- 4%	6
11	0	—	—
	1	60 +/- 4%	9

24.3.4 ADC CTU (Cross Triggering Unit)

Table 423. ADC sampling and conversion timing at 5 V / 3.3 V

Clock (MHz)	T_{ck} (µs)	INP SAMPLE (1)	INP CMP	Ndelay (2)	T_{sample} (3)	T_{sample}/T_{ck}	T_{eval} (µs)	T_{conv} (µs)	T_{conv} (T_{ck})	CPU Mode				TCU Mode			
										Tsample + Teval + Ndelay+Tck		Tsample + Teval + Ndelay + 2*Tck(4)		Tsample + Teval + Ndelay + 1.5*Tck		Tsample + Teval + Ndelay + 4*Tck(5)	
										MCR.ADCLK SEL = 0		MCR.ADCLK SEL = 1		MCR.ADCLK SEL = 0		MCR.ADCLKS EL = 1	
										T_{conv} (µs)	T_{conv} (T_{ck})	T_{conv} (µs)	T_{conv} (T_{ck})	T_{conv} (µs)	T_{conv} (T_{ck})	T_{conv} (µs)	T_{conv} (T_{ck})
3	0.333	4	1	0.5	1.167	3.5	3.333	4.667	14	5.000	15	5.333	16	5.167	15.5	6.000	18
6	0.167	4	1	0.5	0.583	3.5	1.667	2.333	14	2.500	15	2.667	16	2.583	15.5	3.000	18
15	0.067	4	1	0.5	0.233	3.5	0.667	0.933	14	1.000	15	1.067	16	1.033	15.5	1.200	18
20	0.050	4	2	0.5	0.175	3.5	1.000	1.200	24	1.250	25	1.300	26	1.275	25.5	1.400	28
30	0.033	5	2	0.5	0.150	4.5	0.667	0.833	25	0.867	26	0.900	27	0.883	26.5	0.967	29
60	0.017	9	3	1	0.133	8	0.500	0.650	39	0.667	40	0.683	41	0.675	40.5	0.717	43

1. Where: INPSAMPLE ≥ 3.
2. Where: INPSAMP ≤ 6, N = 0.5; INPSAMP > 6, N = 1.
3. Where: T_{sample} = (INPSAMP-N)T_{ck}; Must be ≥ 125 ns.
4. Tck with 50% duty cycle is requested.
5. Tck with 50% duty cycle is requested.

24.3.4.1 Overview

The ADC cross triggering unit (CTU) is added to enhance the injected conversion capability of the ADC. The CTU contains multiple event inputs that can be used to select the channels to be converted from the appropriate event configuration register. The CTU generates a trigger output pulse of one clock cycle and outputs onto an internal data bus the channel to be converted. A single channel is converted for each request. After performing the conversion, the ADC returns the result on internal bus.

The conversion result is also saved in the corresponding data register and it is compared with watchdog thresholds if requested.

The CTU can be enabled by setting MCR[CTUEN].

The CTU and the ADC are synchronous with the MC_PLL_CLK in both cases.

24.3.4.2 CTU in control mode

In CTU control mode, the CPU is able to write in the ADC registers but it cannot start any conversion. Conversion requests can be generated only by the CTU trigger pulse. If a normal or injected conversion is requested, it is automatically discarded.

When a CTU trigger pulse is received with the injected channel number, the conversion starts. The CTUSTART bit is set automatically at this point and it is also automatically reset when CTU Control mode is disabled (CTUEN = '0').

24.3.5 Programmable analog watchdog

24.3.5.1 Introduction

The analog watchdogs are used for determining whether the result of a channel conversion lies within a given guarded area (as shown in *Figure 442*) specified by an upper and a lower threshold value named THRH and THRL respectively.

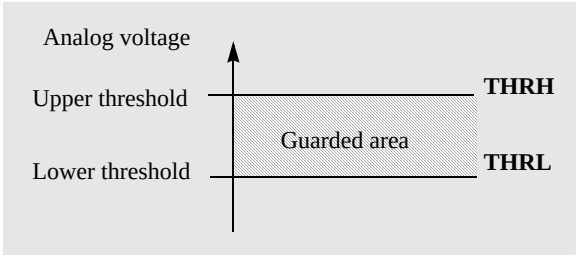


Figure 442. Guarded area

After the conversion of the selected channel, a comparison is performed between the converted value and the threshold values. If the converted value lies outside that guarded area then corresponding threshold violation interrupts are generated. The comparison result is stored as WTISR[WDGxH] and WTISR[WDGxL] as explained in *Table 424*. Depending on the mask bits WTIMR[MSKWDGxL] and WTIMR[MSKWDGxH], an interrupt is generated on threshold violation.

Table 424. Values of WDGxH and WDGxL fields

WDGxH	WDGxL	Converted data
1	0	converted data > THRH
0	1	converted data < THRL
0	0	THRL <= converted data <= THRH

The TRC[THRCH] field specifies the channel on which the analog watchdog is applied. The analog watchdog is enabled by setting the corresponding TRC[THREN].

The lower and higher threshold values for the analog watchdog are programmed using the THRHLR registers.

For example, if channel number 3 is to be monitored with threshold values in THRHLR1, then the TRC[THRCH] field is programmed to select channel number 3.

A set of threshold registers (THRHLRx and TRCx) can be linked only to a single channel for a particular THRCH value. If another channel is to be monitored with same threshold values, then the TRCx[THRCH] must be programmed again.

Note: If the higher threshold for the analog watchdog is programmed lower than the lower threshold and the converted value is less than the lower threshold, then the WDGxL interrupt for the low threshold violation is set, else if the converted value is greater than the lower threshold (consequently also greater than the higher threshold) then the interrupt WDGxH for high threshold violation is set. Thus, the user should avoid that situation as it could lead to misinterpretation of the watchdog interrupts.

24.3.5.2 Analog watchdog functionality

For each input channel the result of the comparison is reflected in the THROP bit in TRC register based on the converted analog values received by the analog watchdogs:

- If the converted data value is lower than the lower threshold then the THROP bit in TRC register will be set to 1.
- If the converted voltage is higher than the higher threshold then the THROP bit in TRC register will be set to 0.
- If the converted voltage lies between the upper and the lower threshold guard window then THROP bit in TRC register will keep its logic value.

The logic level of the THROP bit can be programmed by software. In fact, the user can decide to keep the behavior described or to invert the output logic level by setting the THRINV bit in the TRC register.

An example of the operation is shown in *Table 425*.

Table 425. Example for Analog watchdog operation

Converted data watchdog[x]	Upper threshold watchdog[x]	Lower threshold watchdog[x]	THRINV watchdog[x]	THROP[x]
155h	055h	000h	0	0
055h	1ffh	088h	0	1
155h	055h	000h	1	1
055h	1ffh	088h	1	0

24.3.6 DMA functionality

A DMA request can be programmed after the conversion of every channel by setting the respective masking bit in the DMAR registers. The DMAR masking registers must be programmed before starting any conversion. There is one DMAR per channel type.

The DMA transfers can be enabled using the DMAEN bit of DMAE register. When the DCLR bit of DMAE register is set, the DMA request is cleared the register enabled for DMA transfer has been read.

24.3.7 Interrupts

The ADC generates the following maskable interrupt signals:

- EOC (end of conversion) interrupt request
- ECH (end of chain) interrupt request
- JEOC (end of injected conversion) interrupt request
- JECH (end of injected chain) interrupt request
- EOCTU (end of CTU conversion) interrupt request
- WDGxL and WDGxH (watchdog threshold) interrupt requests

Interrupts are generated during the conversion process to signal events such as End Of Conversion as explained in register description for ISR and IMR.

The analog watchdog interrupts are handled by two registers WTISR (Watchdog Threshold Interrupt Status Register) and WTIMR (Watchdog Threshold Interrupt Mask Register) in order to check and enable the interrupt request to the INTC module. The Watchdog interrupt source sets two pending bits WDGxH and WDGxL in the WTISR for each of the channels being monitored.

24.3.8 Power-down mode

The analog part of the ADC can be put in low power mode by setting the MCR[PWDN]. After releasing the reset signal the ADC analog module is kept in power-down mode by default, so this state must be exited before starting any operation by resetting the appropriate bit in the MCR.

The power-down mode can be requested at any time by setting the MCR[PWDN]. If a conversion is ongoing, the ADC must complete the conversion before entering the power down mode. In fact, the ADC enters power-down mode only after completing the ongoing conversion. Otherwise, the ongoing operation should be aborted manually by resetting the NSTART bit and using the ABORTCHAIN bit.

MSR[ADCSTATUS] bit is set only when ADC enters power-down mode.

After the power-down phase is completed the process ongoing before the power-down phase must be restarted manually by setting the appropriate MCR[START] bit.

Resetting MCR[PWDN] bit and setting MCR[NSTART] or MCR[JSTART] bit during the same cycle is forbidden.

If a CTU trigger pulse is received during power-down, it is discarded.

If the CTU is enabled and the MSR[CTUSTART] bit is ‘1’, then the MCR[PWDN] bit cannot be set.

24.3.9 Auto-clock-off mode

To reduce power consumption during the IDLE mode of operation (without going into power-down mode), an “auto-clock-off” feature can be enabled by setting the MCR[ACKO] bit. When enabled, the analog clock is automatically switched off when no operation is ongoing, that is, no conversion is programmed by the user.

Note: The auto-clock-off feature cannot operate when the digital interface runs at the same rate as the analog interface. This means that when MCR.ADCCLKSEL = 1, the analog clock will not shut down in IDLE mode.

24.4 Register descriptions

24.4.1 Introduction

Table 426 lists ADC registers with their address offsets and reset values.

Table 426. ADC digital registers

Offset from base address ADC_0: 0xFFE0_0000 ADC_1: 0xFFE0_4000	Register name	Location
0x0000	Main Configuration Register (MCR)	*on page 767*
0x0004	Main Status Register (MSR)	*on page 768*
0x0008–0x000F	Reserved	
0x0010	Interrupt Status Register (ISR)	*on page 769*
0x0014–0x001F	Reserved	
0x0020	Interrupt Mask Register (IMR)	*on page 770*
0x0024–0x002F	Reserved	
0x0030	Watchdog Threshold Interrupt Status Register (WTISR)	*on page 771*
0x0034	Watchdog Threshold Interrupt Mask Register (WTIMR)	*on page 772*
0x0038–0x003F	Reserved	
0x0040	DMA Enable Register (DMAE)	*on page 772*
0x0044	DMA Channel Select Register 0 (DMAR0)	*on page 773*
0x0048–0x004F	Reserved	
0x0050	Threshold Control Register 0 (TRC0)	*on page 774*
0x0054	Threshold Control Register 1 (TRC1)	*on page 774*
0x0058	Threshold Control Register 2 (TRC2)	*on page 774*
0x005C	Threshold Control Register 3 (TRC3)	*on page 774*
0x0060	Threshold Register 0 (THRHLR0)	*on page 774*
0x0064	Threshold Register 1 (THRHLR1)	*on page 774*

Table 426. ADC digital registers

Offset from base address ADC_0: 0xFFE0_0000 ADC_1: 0xFFE0_4000	Register name	Location
0x0068	Threshold Register 2 (THRHLR2)	*on page 774*
0x006C	Threshold Register 3 (THRHLR3)	*on page 774*
0x0070–0x0093	Reserved	
0x0094	Conversion Timing Register 0 (CTR0)	*on page 775*
0x0098–0x00A3	Reserved	
0x00A4	Normal Conversion Mask Register 0 (NCMR0)	*on page 775*
0x00A8–0x00B3	Reserved	
0x00B4	Injected Conversion Mask Register 0 (JCMR0)	*on page 776*
0x00B8–00C7	Reserved	
0x00C8	Power-down Exit Delay Register (PDEDR)	*on page 777*
0x00CC–0x00FF	Reserved	
0x0100	Channel 0 Data Register (CDR0)	*on page 777*
0x0104	Channel 1 Data Register (CDR1)	*on page 777*
0x0108	Channel 2 Data Register (CDR2)	*on page 777*
0x010C	Channel 3 Data Register (CDR3)	*on page 777*
0x0110	Channel 4 Data Register (CDR4)	*on page 777*
0x0114	Channel 5 Data Register (CDR5)	*on page 777*
0x0118	Channel 6 Data Register (CDR6)	*on page 777*
0x011C	Channel 7 Data Register (CDR7)	*on page 777*
0x0120	Channel 8 Data Register (CDR8)	*on page 777*
0x0124	Channel 9 Data Register (CDR9)	*on page 777*
0x0128	Channel 10 Data Register (CDR10)	*on page 777*
0x012C	Channel 11 Data Register (CDR11)	*on page 777*
0x0130	Channel 12 Data Register (CDR12)	*on page 777*
0x0134	Channel 13 Data Register (CDR13)	*on page 777*
0x0138	Channel 14 Data Register (CDR14)	*on page 777*
0x013C	Channel 15 Data Register (CDR15)(1)	*on page 777*

1. available only on ADC_0

24.4.2 Control logic registers

24.4.2.1 Main Configuration Register (MCR)

The Main Configuration Register (MCR) provides configuration settings for the ADC.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OWREN	WLSIDE	MODE	0	0	0	0	NSTART	0	JTRGEN	JEDGE	JSTART	0	0	CTUEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	ADCLK SEL	ABORT CHAIN	ABORT	ACKO	0	0	0	0	PWDN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 443. Main Configuration Register (MCR)

Table 427. MCR field descriptions

Field	Description
OWREN	Overwrite enable This bit enables or disables the functionality to overwrite unread converted data. 0 Prevents overwrite of unread converted data; new result is discarded 1 Enables converted data to be overwritten by a new conversion
WLSIDE	Write left/right-aligned 0 The conversion data is written right-aligned. 1 Data is left-aligned (from 15 to (15 – resolution + 1)). The WLSIDE bit affects all the CDR registers simultaneously. See *Figure 457*.
MODE	One Shot/Scan 0 One Shot Mode—Configures the normal conversion of one chain. 1 Scan Mode—Configures continuous chain conversion mode; when the programmed chain conversion is finished it restarts immediately.
NSTART	Normal Start conversion Setting this bit starts the chain or scan conversion. Resetting this bit during scan mode causes the current chain conversion to finish, then stops the operation. This bit stays high while the conversion is ongoing (or pending during injection mode). 0 Causes the current chain conversion to finish and stops the operation 1 Starts the chain or scan conversion
JTRGEN	Injection external trigger enable 0 External trigger disabled for channel injection 1 External trigger enabled for channel injection
JEDGE	Injection trigger edge selection Edge selection for external trigger, if JTRGEN = 1. 0 Selects falling edge for the external trigger 1 Selects rising edge for the external trigger

Table 427. MCR field descriptions(Continued)

Field	Description
JSTART	Injection start Setting this bit will start the configured injected analog channels to be converted by software. Resetting this bit has no effect, as the injected chain conversion cannot be interrupted.
CTUEN	Cross trigger unit conversion enable 0 CTU triggered conversion disabled 1 CTU triggered conversion enabled
ADCLKSEL	Analog clock select This bit can only be written when ADC in Power-Down mode 0 ADC clock frequency is half Peripheral Set Clock frequency 1 ADC clock frequency is equal to Peripheral Set Clock frequency
ABORTCHAIN	Abort Chain When this bit is set, the ongoing Chain Conversion is aborted. This bit is reset by hardware as soon as a new conversion is requested. 0 Conversion is not affected 1 Aborts the ongoing chain conversion
ABORT	Abort Conversion When this bit is set, the ongoing conversion is aborted and a new conversion is invoked. This bit is reset by hardware as soon as a new conversion is invoked. If it is set during a scan chain, only the ongoing conversion is aborted and the next conversion is performed as planned. 0 Conversion is not affected 1 Aborts the ongoing conversion
ACKO	Auto-clock-off enable If set, this bit enables the Auto clock off feature. 0 Auto clock off disabled 1 Auto clock off enabled
PWDN	Power-down enable When this bit is set, the analog module is requested to enter Power Down mode. When ADC status is PWDN, resetting this bit starts ADC transition to IDLE mode. 0 ADC is in normal mode 1 ADC has been requested to power down

24.4.2.2 Main Status Register (MSR)

The Main Status Register (MSR) provides status bits for the ADC.

Address: Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	NSTART	JABORT	0	0	JSTART	0	0	0	CTUSTART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CHADDR							0	0	0	ACK0	0	0	ADCSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 444. Main Status Register (MSR)

Table 428. MSR field descriptions

Field	Description
NSTART	This status bit is used to signal that a Normal conversion is ongoing.
JABORT	This status bit is used to signal that an Injected conversion has been aborted. This bit is reset when a new injected conversion starts.
JSTART	This status bit is used to signal that an Injected conversion is ongoing.
CTUSTART	This status bit is used to signal that a CTU conversion is ongoing.
CHADDR	Current conversion channel address This status field indicates current conversion channel address.
ACKO	Auto-clock-off enable This status bit is used to signal if the Auto-clock-off feature is on.
ADCSTATUS	The value of this parameter depends on ADC status: 000 IDLE — The ADC is powered up but idle. 001 Power-down — The ADC is powered down. 010 Wait state — The ADC is waiting for an external multiplexer. This occurs only when the DSDR register is non-zero. 011 Reserved 100 Sample — The ADC is sampling the analog signal. 101 Reserved 110 Conversion — The ADC is converting the sampled signal. 111 Reserved

Note: MSR[JSTART] is automatically set when the injected conversion starts. At the same time MCR[JSTART] is reset, allowing the software to program a new start of conversion.

The JCMR registers do not change their values.

24.4.3 Interrupt registers

24.4.3.1 Interrupt Status Register (ISR)

The Interrupt Status Register (ISR) contains interrupt status bits for the ADC.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	EOCTU	JEOC	JECH	EOC	ECH
W												w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 445. Interrupt Status Register (ISR)

Table 429. ISR field descriptions

Field	Description
EOCTU	End of CTU Conversion interrupt flag When this bit is set, an EOCTU interrupt has occurred.
JEOC	End of Injected Channel Conversion interrupt flag When this bit is set, a JEOC interrupt has occurred.
JECH	End of Injected Chain Conversion interrupt flag When this bit is set, a JECH interrupt has occurred.
EOC	End of Channel Conversion interrupt flag When this bit is set, an EOC interrupt has occurred.
ECH	End of Chain Conversion interrupt flag When this bit is set, an ECH interrupt has occurred.

24.4.3.2 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) contains the interrupt enable bits for the ADC.

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	MSKEOCTU	MSKJEOC	MSKJECH	MSKEOC	MSKECH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 446. Interrupt Mask Register (IMR)

Table 430. IMR field descriptions

Field	Description
MSKEOCTU	Mask for end of CTU conversion (EOCTU) interrupt When set, the EOCTU interrupt is enabled.
MSKJEOC	Mask for end of injected channel conversion (JEOC) interrupt When set, the JEOC interrupt is enabled.
MSKJECH	Mask for end of injected chain conversion (JECH) interrupt When set, the JECH interrupt is enabled.
MSKEOC	Mask for end of channel conversion (EOC) interrupt When set, the EOC interrupt is enabled.
MSKECH	Mask for end of chain conversion (ECH) interrupt When set, the ECH interrupt is enabled.

24.4.3.3 Watchdog Threshold Interrupt Status Register (WTISR)

Address: Base + 0x0030 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	WDG 3H	WDG 2H	WDG 1H	WDG 0H	WDG 3L	WDG 2L	WDG 1L	WDG 0L
W									w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 447. Watchdog Threshold Interrupt Status Register (WTISR)

Table 431. WTISR field descriptions

Field	Description
WDGxH	This corresponds to the status flag generated on the converted value being higher than the programmed higher threshold (for [x = 0...3]).
WDGxL	This corresponds to the status flag generated on the converted value being lower than the programmed lower threshold (for [x = 0...3]).

24.4.3.4 Watchdog Threshold Interrupt Mask Register (WTIMR)

Address: Base + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	MSKWDG3H	MSKWDG2H	MSKWDG1H	MSKWDG0H	MSKWDG3L	MSKWDG2L	MSKWDG1L	MSKWDG0L
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 448. Watchdog Threshold Interrupt Mask Register (WTIMR)

Table 432. WTIMR field descriptions

Field	Description
MSKWDGxH	This corresponds to the mask bit for the interrupt generated on the converted value being higher than the programmed higher threshold (for [x = 0...3]). When set the interrupt is enabled.
MSKWDGxL	This corresponds to the mask bit for the interrupt generated on the converted value being lower than the programmed lower threshold (for [x = 0...3]). When set the interrupt is enabled.

24.4.4 DMA registers

24.4.4.1 DMA Enable (DMAE) register

The DMA Enable (DMAE) register sets up the DMA for use with the ADC.

Address: Base + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DCLR	DMAEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 449. DMA Enable (DMAE) register

Table 433. DMAE field descriptions

Field	Description
DCLR	DMA clear sequence enable 0 DMA request cleared by Acknowledge from DMA controller 1 DMA request cleared on read of data registers
DMAEN	DMA global enable 0 DMA feature disabled 1 DMA feature enabled

24.4.4.2 DMA Channel Select Register (DMAR[0])

DMAR0 = Enable bits for channel 0 to 15 (precision channels)

Address: Base + 0x0044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DMA 15	DMA 14	DMA 13	DMA 12	DMA 11	DMA 10	DMA 9	DMA 8	DMA 7	DMA 6	DMA 5	DMA 4	DMA 3	DMA 2	DMA 1	DMA 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 450. DMA Channel Select Register 0 (DMAR0)

Table 434. DMARx field descriptions

Field	Description
DMAn	DMA enable When set (DMAn = 1), channel n is enabled to transfer data in DMA mode.

24.4.5 Threshold registers

24.4.5.1 Introduction

The Threshold registers store the user programmable lower and upper thresholds' values. The inverter bit and the mask bit for mask the interrupt are stored in the TRC registers.

24.4.5.2 Threshold Control Register (TRC*x*, *x* = [0..3])

Address: Base + 0x0050 (TRC0)
Base + 0x0054 (TRC1)
Base + 0x0058 (TRC2)
Base + 0x005C (TRC3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	THR EN	THR INV	THR OP	0	0	0	0	0	0	THRCH						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 451. Threshold Control Register (TRC*x*, *x* = [0..3])

Table 435. TRC*x* field descriptions

Field	Description
THREN	Threshold enable When set, this bit enables the threshold detection feature for the selected channel.
THRINV	Invert the output pin Setting this bit inverts the behavior of the threshold output pin.
THROP	This bit reflects the output pin status. See *Section 24.3.5.2: Analog watchdog functionality*. It reflects the output pin status.
THRCH	Choose the channel for threshold comparison.

24.4.5.3 Threshold Register (THRHLR[0:3])

The four THRHLR*n* registers store the user-programmable thresholds' 10-bit values.

Address: Base + 0x0060 (THRHLR0)
Base + 0x0064 (THRHLR1)
Base + 0x0068 (THRHLR2)
Base + 0x006C (THRHLR3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	THRH									
W																
Reset	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	THRL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 452. Threshold Register (THRHLR[0:3])

Table 436. THRHLR*x* field descriptions

Field	Description
THRH	High threshold value for channel *n*.
THRL	Low threshold value for channel *n*.

24.4.6 Conversion Timing Registers CTR[0]

CTR0 = associated to internal precision channels (from 0 to 15)

Address: Base + 0x0094 (CTR0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPLATCH	0	OFFSHIFT		0	INPCMP		0	INPSAMP							
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1

Figure 453. Conversion Timing Registers CTR[0]

Table 437. CTR field descriptions

Field	Description
INPLATCH	Configuration bit for latching phase duration
OFFSHIFT	Configuration for offset shift characteristic 00 No shift (that is the transition between codes 000h and 001h) is reached when the A_{VIN} (analog input voltage) is equal to 1 LSB. 01 Transition between code 000h and 001h is reached when the A_{VIN} is equal to1/2 LSB 10 Transition between code 00h and 001h is reached when the A_{VIN} is equal to 0 11 Not used **Note:** Available only on CTR0
INPCMP	Configuration bits for comparison phase duration
INPSAMP	Configuration bits for sampling phase duration

24.4.7 Mask registers

24.4.7.1 Introduction

The Mask registers are used to program the 16 input channels that are converted during Normal and Injected conversion.

24.4.7.2 Normal Conversion Mask Registers (NCMR[0])

NCMR0 = Enable bits of normal sampling for channel 0 to 15 (precision channels)

Address: Base + 0x00A4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 454. Normal Conversion Mask Register 0 (NCMR0)

Table 438. NCMR field descriptions

Field	Description
CHn	Sampling enable When set Sampling is enabled for channel n.

24.4.7.3 Injected Conversion Mask Registers (JCMR[0])

JCMR0 = Enable bits of injected sampling for channel 0 to 15 (precision channels)

Address: Base + 0x00B4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CH15	CH14	CH13	CH12	CH11	CH10	CH9	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	CH0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 455. Injected Conversion Mask Register 0 (JCMR0)

Table 439. JCMR field descriptions

Field	Description
CHn	Sampling enable When set, sampling is enabled for channel n.

24.4.8 Delay registers

24.4.8.1 Power-Down Exit Delay Register (PDEDR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	PDED							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 456. Power-Down Exit Delay Register (PDEDR)

Table 440. PDEDR field descriptions

Field	Description
PDED	Delay between the power-down bit reset and the start of conversion. The delay is to allow time for the ADC power supply to settle before commencing conversions. The power down delay is calculated as: PDED x 1/frequency of ADC clock.

24.4.9 Data registers

24.4.9.1 Introduction

ADC conversion results are stored in data registers. There is one register per channel.

24.4.9.2 Channel Data Registers (CDR[0..15])

CDR[0..15] = precision channels

Each data register also gives information regarding the corresponding result as described below.

Address: See *Table 426* Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	VALID	OVERW	RESULT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	CDATA[0:9] (MCR[WLSIDE] = 0)									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDATA[0:9] (MCR[WLSIDE] = 1)										0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 457. Channel Data Registers (CDR[0..26])

Table 441. CDR field descriptions

Field	Description
VALID	Used to notify when the data is valid (a new value has been written). It is automatically cleared when data is read.
OVERW	Overwrite data This bit signals that the previous converted data has been overwritten by a new conversion. This functionality depends on the value of MCR[OWREN]: – When OWREN = 0, then OVERW is frozen to 0 and CDATA field is protected against being overwritten until being read. – When OWREN = 1, then OVERW flags the CDATA field overwrite status. 0 Converted data has not been overwritten 1 Previous converted data has been overwritten before having been read
RESULT	This bit reflects the mode of conversion for the corresponding channel. 00 Data is a result of Normal conversion mode 01 Data is a result of Injected conversion mode 10 Data is a result of CTU conversion mode 11 Reserved
CDATA	Channel 0-15 converted data. Depending on the value of the MCR[WLSIDE] bit, the position of this field can be changed as shown in *Figure 457*.

25 Cross Triggering Unit (CTU)

25.1 Introduction

In PWM driven systems it is important to schedule the acquisition of the state variables with respect to PWM cycle. State variables are obtained through the following peripherals: ADC, position counter (for example, quadrature decoder, resolver and sine-cos sensor) and PWM duty cycle decoder.

The cross triggering unit (CTU) is intended to completely avoid CPU involvement in the time acquisitions of state variables during the control cycle that can be the PWM cycle, the half PWM cycle or a number of PWM cycles. In such cases the pre-setting of the acquisition times needs to be completed during the previous control cycle, where the actual acquisitions are to be made, and a double-buffered structure for the CTU registers is used, in order to activate the new settings at the beginning of the next control cycle. Additionally, four FIFOs inside the CTU are available to store the ADC results.

25.2 CTU overview

The CTU receives various incoming signals from different sources (PWM, timers, position decoder and/or external pins). These signals are then processed to generate as many as eight trigger events. An input can be a rising edge, a falling edge or both, edges of each incoming signal. The output can be a pulse or a command (or a stream of consecutive commands for over-sampling support) or both, to one or more peripherals (for example, ADC or timers).

The CTU interfaces to the following peripherals:

- PWM—13 inputs
- Timers—2 inputs
- GPIO—1 external input signal

The 16 input signals are digital signals and the CTU must be able to detect a rising and/or a falling edge for each of them.

The CTU comprises the following:

- Input signals interface
- User interface (such as configuration registers)
- ADC interface
- Timers interface

The block diagram of the CTU is shown in *Figure 458*.

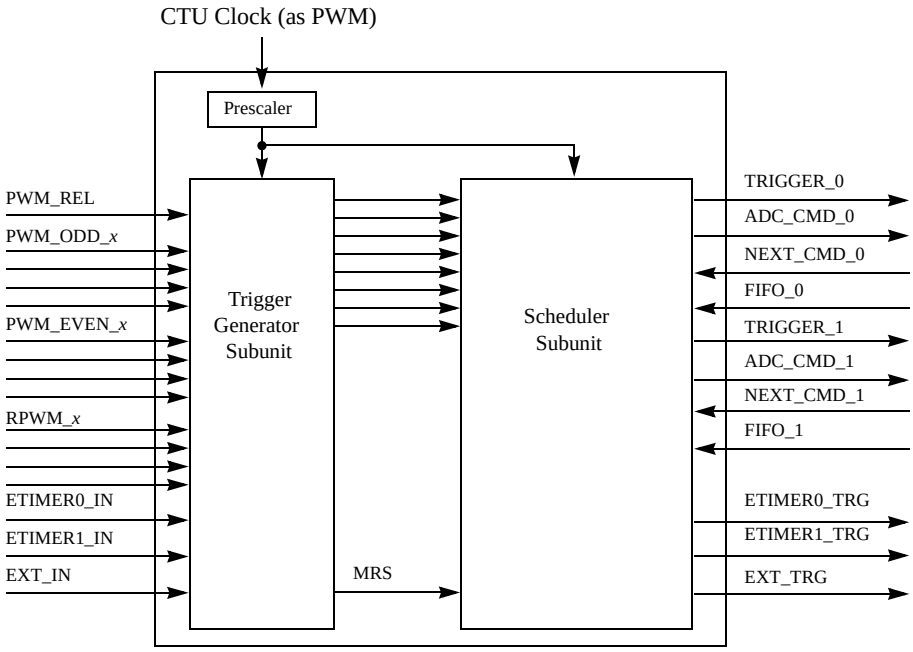


Figure 458. Cross triggering unit diagram

The CTU consists of two subunits:

- Trigger generator
- Scheduler

The trigger generator subunit handles incoming signals, selecting for each signal, the active edges to generate the Master Reload signal, and generates as many as eight trigger events (signals). The scheduler subunit generates the trigger event output according to the occurred trigger event (signal).

25.3 Functional description

The following describes the functionality of the CTU.

25.3.1 Trigger events features

The TGS is capable of generating as many as eight trigger events. Each trigger event has the following characteristics:

- generation of the trigger event is sequential in time
- the triggers list uses eight 16-bit double-buffered registers
- the new triggers list is loaded on each Master Reload Signal (MRS)

Note: The triggers list is only reloaded on an MRS occurrence if the reload enable bit is set.

25.3.2 Trigger generator subunit (TGS)

The trigger generator subunit has the following two modes:

- Triggered mode—Each event source for the incoming signals can generate as many as eight trigger event outputs. For the ADC, a commands list is received from the CPU, and each event source can generate as many as eight commands or command streams.
- Sequential mode—An event source from the incoming signals can generate one trigger event output, the next event source generates the next trigger event output, and so on in a predefined sequence. For the ADC, a commands list is received from the CPU and the sequence of the selected incoming trigger events generate commands or command streams.

The TGS Mode is selected using the TGS_M bit in the TGS Control Register.

25.3.3 TGS in triggered mode

The structure of the TGS in Triggered mode is shown in *Figure 459*.

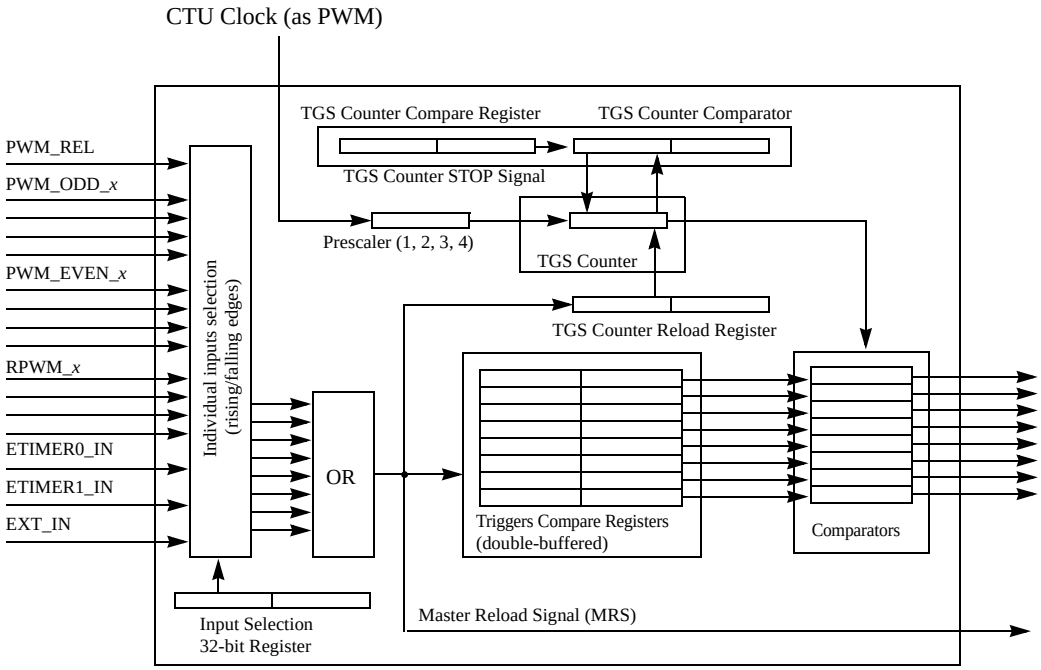


Figure 459. TGS in triggered mode

The TGS has 16 input signals, each of which is selected from the input selection register (TGSISR) which enables or disables rising or falling edge signals independently – for a maximum total 32 enabled input events. These signals are ORed in order to generate the MRS. At the beginning of control cycle *n* (defined by the MRS occurrence), the MRS preloads the TGS counter register using the preload value written to the double-buffered

register (TGSCRR) during the control cycle *n* – 1, and reloads all the double-buffered registers (such as Trigger Compare registers, TGSCR, TGSCRR itself).

The triggers list registers consist of eight compare registers. Each triggers list register is associated with a comparator. On reload (MRS occurrence), the comparators are disabled. One TGS clock cycle is necessary to enable them and to start the counting. The MRS is output together with individual trigger signals. The MRS can be performed by hardware or by software. The MRS_SG bit in the CTU control register, if set to 1, generates equivalent software MRS (that is, resets/reloads TGS Counter and reloads all double-buffered registers). This bit is cleared by each hardware or software MRS occurrence.

The TGS counter compare register and the TGS counter comparator are used to stop the TGS counter when it reaches the value stored in the TGS counter compare register before an MRS occurs.

The prescaler for TGS and SU can be 1,2,3,4 (PRES bits in the TGS Control Register).

An example timing for the TGS in Triggered Mode is shown in *Figure 460*. The red arrows indicate the MRS occurrences, while the black arrows indicate the trigger event occurrences, with the relevant delay with respect to the last MRS occurrence.

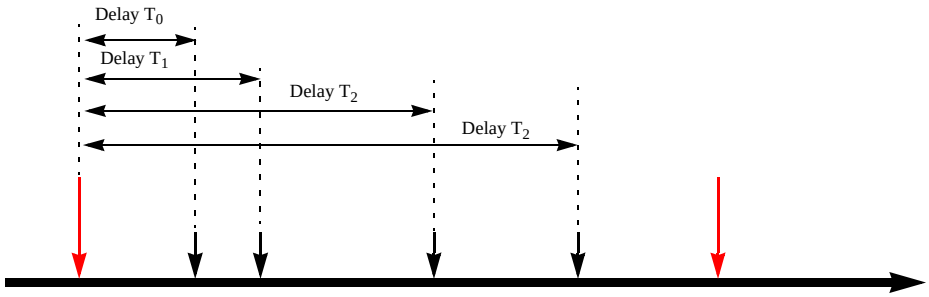


Figure 460. Example timing for TGS in triggered mode

25.3.4 TGS in sequential mode

The structure of the TGS in sequential mode is shown in *Figure 461*.

The 32 input events (16 signals with two edges for signal), which can be individually enabled, are ORed in order to generate the event signal (ES). The ES enables the reload of the TGS counter register and pilots the 3-bit counter in order to select the next active trigger. One of the 32 input events can be selected, through the MRS_SM (master reload selection sequential mode) bits in the TGS control register, to be the MRS, that enables the reload of the triggers list and resets the 3-bit counter (incoming events counter), that is, the MRS is the signal linked with the control cycle defined as the time window between two consecutive MRSs. In this mode, each incoming event sequentially enables only one trigger event through the 3-bit counter and the MUX. The MUX is a selection switch that enables, according to the number of event signals occurred, only one of the eight trigger signals to the scheduler subunit. Sequences of as many as eight trigger events can be supported within this control cycle.

For the other features see the previous paragraphs.

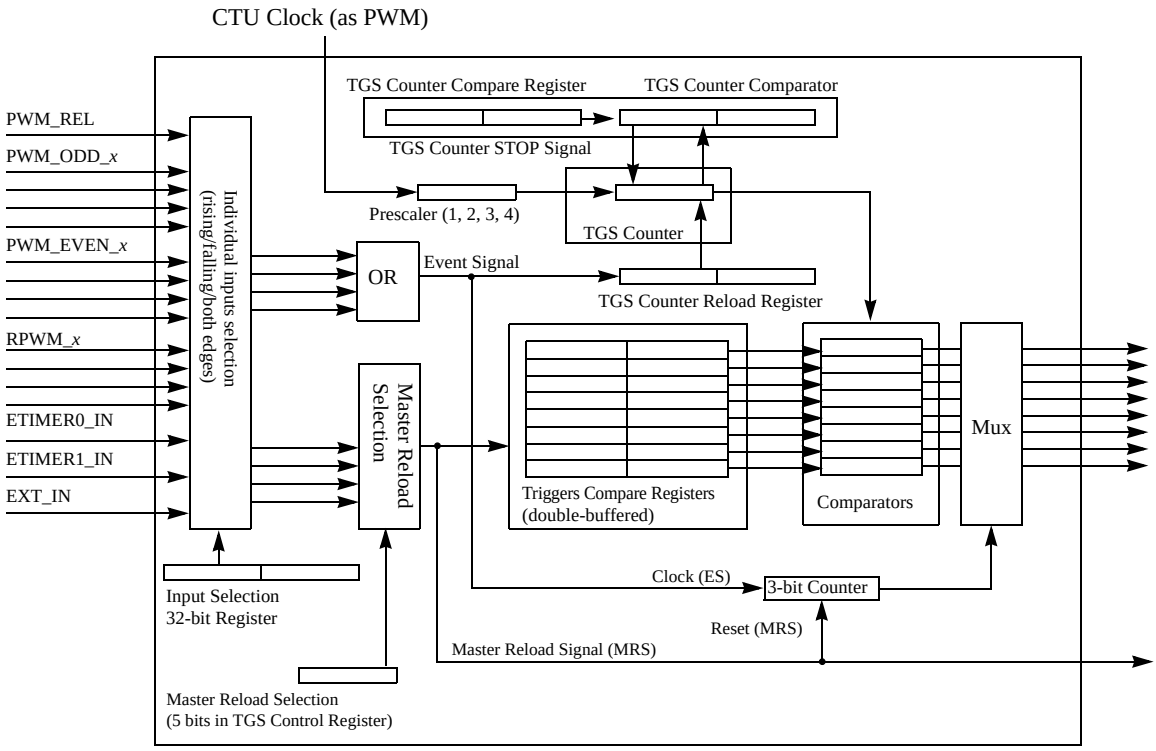


Figure 461. TGS in sequential mode

An example timing diagram for TGS in sequential mode is shown in *Figure 462*. The red arrows indicate the MRS occurrences and ES occurrences, while the black arrows indicate the trigger event occurrences with the relevant delay with respect to the ES occurrence. The first red arrow indicates the first ES occurrence, which is also the MRS.

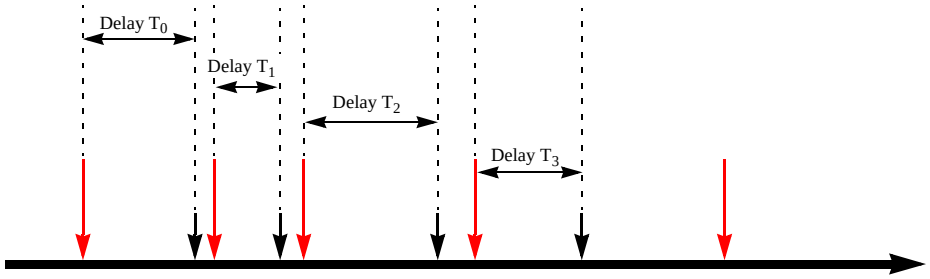


Figure 462. Example timing for TGS in sequential mode

25.3.5 TGS counter

The TGS counter is able to count from negative to positive, that is, from 0x8000 to 0x7FFF. *Figure 463* shows examples in order to explain the TGS counter counts. The compare operation to stop the TGS counter is not enabled during the first counting cycle, in order to allow the counting, if the value of the TGSCRR is the same as the value of the TGSCCR.

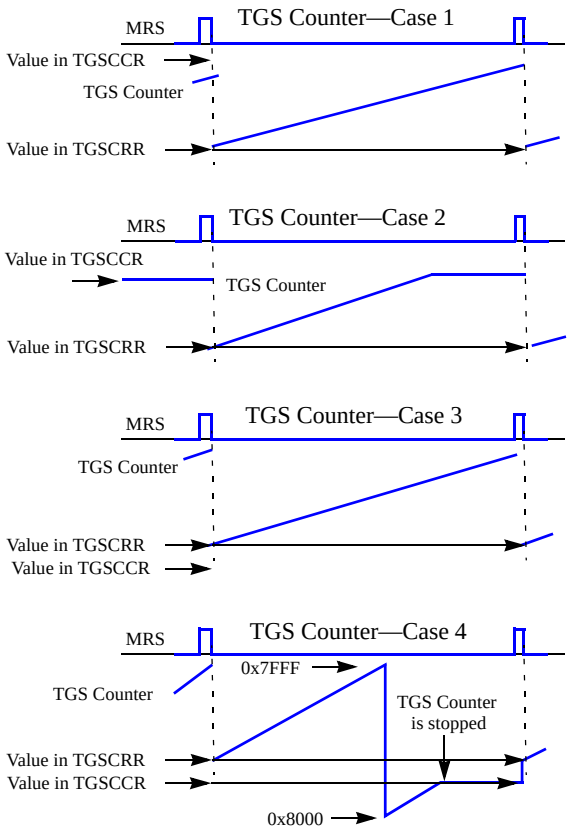


Figure 463. TGS counter cases

25.4 Scheduler subunit (SU)

The structure of the SU is shown in *Figure 464*.

The SU generates the trigger event output according to the occurred trigger event, and it has the same functionality in both TGS modes (triggered mode and sequential mode). Each of the four SU outputs:

- ADC command or ADC stream of commands
- eTimer1 pulse (ETIMER0_TRG internally connected to eTimer_0 AUX0)
- eTimer2 pulse (ETIMER1_TRG internally connected to eTimer_1 AUX0
- External trigger pulse

can be linked to any of eight trigger events by the Trigger Handler block. Each trigger event can be linked to one or more SU outputs.

If two events at the same time are linked to the same output only one output is generated and an error is provided. The output is generated using the trigger with the lowest index. For example, if trigger 0 and trigger 1 are linked to the ADC output and they occur together, an error is generated and the output linked with the trigger 0 is generated.

When a trigger is linked to the ADC, an associated ADC command (or stream of commands) is generated. The ADC Commands List Control Register (CLCR*x*) sets the assignment to an ADC command or to a stream of commands. When a trigger is linked to a timer or to the external trigger, a pulse with an active rising edge is generated. Additional features for the external triggers are available:

The external trigger output has:

- Pulse mode
- Toggle mode

In Toggle Mode, each trigger event is linked to the external trigger, the external trigger pin toggles. The ON-Time for both modes (Pulse mode and Toggle mode) of the triggers is defined from a COTR register (Control On Time Register). A guard time is also defined from the same register at the same value of the ON-Time. A new trigger will be generated only if the ON time + Guard Time has past. The ON-Time and the Guard Time are only used for external Triggers.

External signals can be asynchronous with motor control clock. For this reason a programmable digital filter is available. The external signal is considered at 1 if it is latched N time at 1, and is considered at 0 if it is latched N time at 0, where N is a value in the digital filter control register.

Trigger events in the SU can be initiated by hardware or by software, and an additional software control is possible for each trigger event (as for the MRS), so 1 bit for each trigger event in the CTU Control Register is used to generate an equivalent software trigger event. Each of these bits is cleared by a respective hardware or software trigger event.

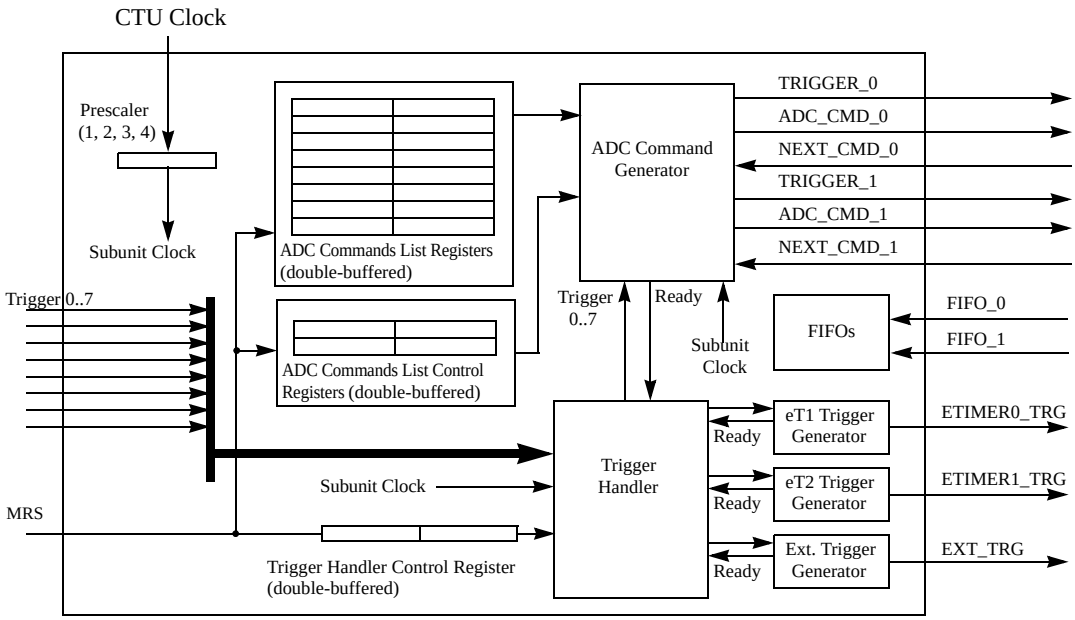


Figure 464. Scheduler subunit

25.4.1 ADC commands list

The ADC can be controlled by the CPU (CPU Control Mode) and by the CTU (CTU Control Mode). The CTU can control the ADC by sending an ADC command only when the ADC is in CTU control mode. During the CTU control mode, the CPU is able to write to the ADC registers but it can not start a new conversion. A control bit is allowed to select from the classic interface of the CTU control mode. Once selected, no change is possible unless a reset occurs.

The SU uses a Commands List in order to select the command to send to the ADC when a trigger event occurs. The commands list can hold twenty-four 16-bit commands (see *Section 25.4.2: ADC commands list format*) and it is double-buffered, that is, the commands list can be updated at any time between two consecutive MRS, but the changes become workable only after the next MRS occurs, and a correct reload is performed. In order to manage the commands list, 5 bits are available in the CLCRx (ADC Commands List Control Register *x*), for the position of the first command in the list of commands for each trigger event. The number of commands piloted by the same trigger event is defined directly in the commands list. For each command, a bit defines whether or not it is the first command of a commands list.

25.4.2 ADC commands list format

The two ADCs support the Single Conversion Mode (1 bit in the ADC command format allows selection of the conversion mode), and the Dual Conversion Mode (the sampling phases and the conversion phases are performed at the same time; the storage of the results are performed in series). The result of each conversion, in both modes, can be stored in one of the four available FIFOs. In dual conversion mode, both ADCs must store the result of their conversion in the same FIFO. If the access to the FIFO is in the same clock cycle, the ADC unit 0 has the priority, otherwise the first ADC that ends its conversion will write as first in the FIFO. Four analog channels are shared across the two ADCs and the total number of channels is 28 (12 + 12 + 4 shared channels), that is, 16 channels for each ADC (12 + 4 shared channels). The dual conversion mode on the same physical channel is not allowed, but the dual conversion mode on the same channel number is allowed. According to this, if, in dual conversion mode, the channel number is the same for both the ADCs and the selected channel is one of the shared channels, the CTU will detect an invalid command. In Dual Conversion Mode, 4 bits select each channel number, and the conversion mode selection bit selects the Dual Conversion Mode. If the Single Conversion Mode is selected, 5 of the 8 bits reserved to select the channels in Dual Conversion Mode are re-used to select the channel (4 bits) and the ADC unit (1 bit). See *Section 25.8.10: Commands list register x (x = 1,...,24) (CLRx)*.

The interrupt request bit (CIR) is used as an interrupt request when ADC will start the command execution with this bit set and it is only for CTU internal use. Before the next command to the CTU controls is sent, the value of the first command (FC) bit is checked to see if it is the current command is the first command of a new stream of consecutive commands or not. If not, the CTU sends the command.

According to the previous considerations, the commands in the list allow control on:

- Channel 0: number of ADC channel to sample from ADC unit 0 (4 bits)
- Channel 1: number of ADC channel to sample from ADC unit 1 (4 bits)
- FIFO selection bits for the ADC unit 0/1 (2 bits)
- Conversion Mode selection bit
- First command bit (only for CTU internal use)
- Interrupt request bit (only for CTU internal use)

25.4.3 ADC results

ADC results can be stored in the channel relevant standard result register and/or in one of the four FIFOs: the different FIFOs allow to dispatch ADC results according to their type of acquisition (for example phase currents, rotor position or ground-noise). Each FIFO has its own interrupt line and DMA request signal (plus an individual overflow error bit in the FIFO status register). The store location is specified in the ADC command, that is, the FIFOs are available only in CTU Control Mode. Each entry of a FIFO is 32-bits.

The size of the FIFOs are the following:

- FIFO1 and FIFO2—16 entries (sized to avoid overflow during a full PWM period for current acquisitions)
- FIFO3 and FIFO4—4 entries (low acquisition rate FIFOs)

Results in each FIFO can be read by a 16-bit read transaction (only the result is read in order to minimize the CPU load before computing on results) or by a 32-bit read transaction (both the result and the channel number are read in order to avoid blind acquisitions), 5 bits in the upper 16 bits indicate the ADC unit (1 bit) and the channel number (4 bits). The result registers (only for the FIFOs) can be read from two different addresses in the ADC memory map. The format of the result depends on the address from which it is read. The available formats are:

- Unsigned right-justified

  Conversion result is unsigned right-justified data. Bits [9:0] are used for 10-bit resolution and bits [15:10] always return zero when read.
- Signed left-justified

  Conversion result is signed left-justified data. Bit [15] is reserved for sign and is always read as zero for this ADC, bits [14:5] are used for 10-bit resolution, and bits [4:0] always return zero when read.

25.5 Reload mechanism

Some CTU registers are double-buffered, and the reload is controlled by a reload enable bit, as the TGSISR_RE bit or the DFE bit, but for the most of the double-buffered registers, the reload is controlled by the MRS occurrence, and it is synchronized with the beginning of the CTU control period.

If the MRS occurs while the user is updating some double-buffered registers, eg. some registers of the triggers list, the new triggers list will be a mix of the old triggers list and the new triggers list, because the user has not ended the update of the triggers list before the MRS occurrence.

In order to avoid this case, 1 bit enables the reload operation, that is, to inform the CTU that the user has ended updates to the double-buffered registers, and the reload can be performed without problems of mixed scenarios. In order to guarantee the coherency, the reload of all double-buffered registers is enabled by setting GRE (General Reload Enable) bit in the CTU Control Register. The user must ensure that all intended double-buffered registers are updated before a new MRS occurrence. If an MRS occurs before a GRE bit is set (for example, wrong application timing), the update is not performed, the previous values of all double-buffered registers remain active, the error flag is set (the MRS_RE bit in the CTU Error Flag Register) and, if enabled, CTU performs an interrupt request.

All the double-buffered registers use the General Reload Enable (GRE) bit to enable the reload when the MRS occurs. The GRE bit is R/S (Read/Set) and if this bit is 1, the reload can be performed, while if this bit is 0, the reload is not performed. A correct reload resets the GRE bit. None of the double-buffered registers can be written while the GRE bit remains set. The GRE bit can be reset by the occurrence of the next MRS (that is, a correct reload) or by software setting the CGRE bit.

The CGRE is reset by hardware after that GRE bit is reset. If the user sets the CGRE bit and at the same time a MRS occurs, CGRE has the priority so GRE is reset and the reload is not performed. In the same way, the GRE has the priority when compared with the MRS occurrence, and the CGRE has the priority compared with the GRE (the two bits are in the same register so they can be set in the same time). MRS has the priority compared with the re-synchronization bit of the TGSISR.

In order to verify if a reload error occurs, FGRE (Flag GRE bit in the CTU Control Register) bit is used. When one of the double-buffered registers is written, this flag is set to 1 and it is reset by a correct reload. When the MRS occurs while FGRE is 1 and GRE is 1, a correct reload is performed (because all intended registers have been updated before the MRS occurs). If FGRE is 1 and GRE is 0, a reload is not performed, the error flag (MRS_RE) is set and (if enabled) an interrupt for an error is performed (in this case at least one register was written but the update has not ended before the MRS occurrence). If FGRE is 0 it is not necessary to perform a reload because all the double-buffered registers are unchanged (see *Figure 465*).

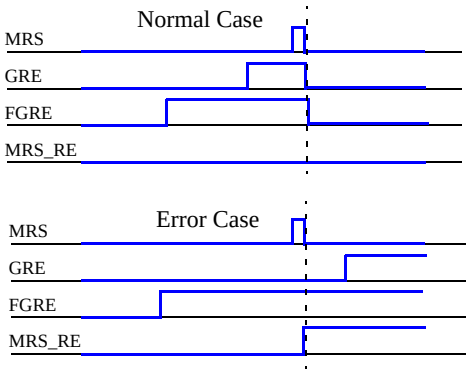


Figure 465. Reload error scenario

25.6 Power safety mode

To reduce power consumption two mechanisms are implemented:

- MDIS bit in the CTUPCR
- STOP mode

25.6.1 MDIS bit

The MDIS bit in the CTUPCR is used for stopping the clock to all non memory mapped registers.

25.6.2 STOP mode

To reduce consumption, it is also possible to enable a stop request from the Mode Entry module. The FIFOs are considered a lot like memory mapped registers, otherwise there could be some problems if a read operation occurs during the MDIS bit set period. When the clock is started after an MDIS bit setting or a stop signal, some mistakes could occur. For example, a wrong trigger could be provided because it was programmed before the stop signal was performed, and some incorrect write operations into the FIFOs could happen. For this reason after a stop signal after a MDIS bit setting the FIFO have to be empty. In order to avoid the problems linked to a wrong trigger, the CTU output can be disabled by the CTU_ODIS bit and the ADC interface state machine can be reset by the CRU_ADC_R (see *Section 25.8.21: Cross triggering unit control register (CTUCR)*).

25.7 Interrupts and DMA requests

25.7.1 DMA support

The DMA can be used to configure the CTU registers. One DMA channel is reserved for performing a block transfer, and the MRS can be used as an optional DMA request signal (MRS_DMAE bit in the CTU Interrupt/DMA Register).

Note: If enabled, the DMA request on the MRS occurrence is performed only if a reload is performed, that is, only if the GRE bit is set.

Moreover, this CTU implementation requires DMA support for reading the data from the FIFOs. One DMA channel is available for each FIFO. Each FIFO can perform a DMA request when the number of words stored in the FIFO reaches the threshold value.

25.7.2 CTU faults and errors

Faults and errors that could occur during the programming include:

- An MRS occurs while user is updating the double-buffered registers and the MRS_RE bit is set.
- Receiving more than eight EVs before that the next MRS occurs in TGS sequential mode and the SM_TO bit is set.
- A trigger event occurs during the time when the actions of the previous trigger event are not completed (user ensures no trigger event occurs during another one is

processed, but if user makes a mistake and a trigger event occurs when another one is processed, the incoming trigger event will be lost and an error occurs).

There are four overrun flags (one for each type of output). The general mechanism shall be as in *Figure 464*.

The Trigger Handler, when a trigger event occurs, and the corresponding Ready signal is high, presents the respective trigger signal (one cycle high time + one cycle low time) to the respective generator sub-block (ADC Command Generator, eT0 Trigger Generator, eT1 Trigger Generator or Ext. Trigger Generator). This generator sub-block then generates the requested signal. Until this real signal is generated (including guard time) the Ready signal is kept low.

In the case of ADC command generator, the Ready signal shall be kept low until the last conversion in the batch is finished. The respective overrun flag is set at the following conditions:

– Ready signal is low.
– The rising edge of the respective trigger signal (from Trigger Handler to generator sub-block) occurs.

This architecture allows the user to pre-set a trigger to the eTimer0/1 in the middle of an ADC conversion, that is, the SU will be considered busy only if a request to perform the same action that the SU is already performing occurs. One of the following bits is set: ADC_OE, T0_OE, T1_OE, or ET_OE.

- Invalid (unrecognized) ADC command and the ICE bit is set.
- The MRS occurs before the enabled trigger events occur and the MRS_O bit is set.
- TGS overrun in sequential mode: a new incoming EV occurs before than the trigger event selected by the previous EV occurs. The incoming EV sets an internal busy flag. The outgoing trigger event (all line are ORed) resets this flag to 0. TGS Overrun in the sequential mode shall be generated under the following conditions:
  – TGS is in sequential mode
  – there is an incoming EV while the busy flag is high. the TGS_OSM bit is set.

The faults/errors flags in the CTU error flag register and in the CTU interrupt flag register can be cleared by writing a 1 while writing a 0 has no effect. The CTU does not support a write-protection mechanism.

25.7.3 CTU interrupt/DMA requests

The CTU can perform the following interrupt/DMA requests (15 interrupt lines):

- Error interrupt request (see *Section 25.7.2: CTU faults and errors*) (1 interrupt line)
- ADC command interrupt request (1 interrupt line)
- Interrupt request on MRS occurrence (1 interrupt line)
- Interrupt request on each trigger event occurrence (1 interrupt line for each trigger event)
- FIFOs interrupt requests and/or DMA transfer request (1 interrupt line for each FIFO)
- DMA transfer request on the MRS occurrence if GRE bit is set

The interrupt flags are shown in *Table 442*.

Table 442. CTU interrupts

Category	Interrupt	Interrupt function
Managed individually	MRS_I	MRS Interrupt flag (IRQ193)
	T0_I	Trigger 0 interrupt flag (IRQ194)
	T1_I	Trigger 1 interrupt flag (IRQ195)
	T2_I	Trigger 2 interrupt flag (IRQ196)
	T3_I	Trigger 3 interrupt flag (IRQ197)
	T4_I	Trigger 4 interrupt flag (IRQ198)
	T5_I	Trigger 5 interrupt flag (IRQ199)
	T6_I	Trigger 6 interrupt flag (IRQ200)
	T7_I	Trigger 7 interrupt flag (IRQ201)
	ADC_I	ADC command interrupt flag (IRQ206)
ORed onto FIFO1_I (IRQ202)	FIFO_FULL0	This bit is set to 1 if the FIFO 0 is full.
	FIFO_EMPTY0	This bit is set to 1 if the FIFO 0 is empty.
	FIFO_OVERFLOW0	This bit is set to 1 if the number of words exceeds the value set in the threshold 0.
	FIFO_OVERRUN0	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL0 flag is set.
ORed onto FIFO2_I (IRQ203)	FIFO_FULL1	This bit is set to 1 if the FIFO 1 is full.
	FIFO_EMPTY1	This bit is set to 1 if the FIFO 1 is empty.
	FIFO_OVERFLOW1	This bit is set to 1 if the number of words exceeds the value set in the threshold 1.
	FIFO_OVERRUN1	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL1 flag is set.
ORed onto FIFO3_I (IRQ204)	FIFO_FULL2	This bit is set to 1 if the FIFO 2 is full.
	FIFO_EMPTY2	This bit is set to 1 if the FIFO 2 is empty.
	FIFO_OVERFLOW2	This bit is set to 1 if the number of words exceeds the value set in the threshold 2.
	FIFO_OVERRUN2	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL2 flag is set.
ORed onto FIFO4_I (IRQ205)	FIFO_FULL3	This bit is set to 1 if the FIFO 3 is full.
	FIFO_EMPTY3	This bit is set to 1 if the FIFO 3 is empty.
	FIFO_OVERFLOW3	This bit is set to 1 if the number of words exceeds the value set in the threshold 3.
	FIFO_OVERRUN3	This bit is set to 1 if a write operation occurs when corresponding FIFO_FULL3 flag is set.

Table 442. CTU interrupts(Continued)

Category	Interrupt	Interrupt function
ORed onto ERR_I (IRQ207)	MRS_RE	Master Reload Signal Reload Error
	SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode
	ICE	Invalid Command Error
	MRS_O	Master Reload Signal Overrun
	TGS_OSM	TGS Overrun in Sequential Mode
	ADC_OE	ADC command generation Overrun Error
	T0_OE	Timer 0 trigger generation Overrun Error
	T1_OE	Timer 1 trigger generation Overrun Error
	ET_OE	External Trigger generation Overrun Error

25.8 Memory map

Table 443. CTU memory map

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	*on page 796*
0x0004	TGSCR — Trigger Generator Subunit Control Register	*on page 799*
0x0006	T0CR — Trigger 0 Compare Register	*on page 799*
0x0008	T1CR — Trigger 1 Compare Register	*on page 799*
0x000A	T2CR — Trigger 2 Compare Register	*on page 799*
0x000C	T3CR — Trigger 3 Compare Register	*on page 799*
0x000E	T4CR — Trigger 4 Compare Register	*on page 799*
0x0010	T5CR — Trigger 5 Compare Register	*on page 799*
0x0012	T6CR — Trigger 6 Compare Register	*on page 799*
0x0014	T7CR — Trigger 7 Compare Register	*on page 799*
0x0016	TGSCCR — TGS Counter Compare Register	*on page 800*
0x0018	TGSCRR — TGS Counter Reload Register	*on page 800*
0x001A	Reserved	
0x001C	CLCR1 — Commands List Control Register 1	*on page 800*
0x0020	CLCR2 — Commands List Control Register 2	*on page 801*
0x0024	THCR1 — Trigger Handler Control Register 1	*on page 801*
0x0028	THCR2 — Trigger Handler Control Register 2	*on page 803*

Table 443. CTU memory map(Continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x002C	CLR1—Commands List Register 1	*on page 805*
0x002E	CLR2—Commands List Register 2	*on page 805*
0x0030	CLR3—Commands List Register 3	*on page 805*
0x0032	CLR4—Commands List Register 4	*on page 805*
0x0034	CLR5—Commands List Register 5	*on page 805*
0x0036	CLR6—Commands List Register 6	*on page 805*
0x0038	CLR7—Commands List Register 7	*on page 805*
0x003A	CLR8—Commands List Register 8	*on page 805*
0x003C	CLR9—Commands List Register 9	*on page 805*
0x003E	CLR10—Commands List Register 10	*on page 805*
0x0040	CLR11—Commands List Register 11	*on page 805*
0x0042	CLR12—Commands List Register 12	*on page 805*
0x0044	CLR13—Commands List Register 13	*on page 805*
0x0046	CLR14—Commands List Register 14	*on page 805*
0x0048	CLR15—Commands List Register 15	*on page 805*
0x004A	CLR16—Commands List Register 16	*on page 805*
0x004C	CLR17—Commands List Register 17	*on page 805*
0x004E	CLR18—Commands List Register 18	*on page 805*
0x0050	CLR19—Commands List Register 19	*on page 805*
0x0052	CLR20—Commands List Register 20	*on page 805*
0x0054	CLR21—Commands List Register 21	*on page 805*
0x0056	CLR22—Commands List Register 22	*on page 805*
0x0058	CLR23—Commands List Register 23	*on page 805*
0x005A	CLR24—Commands List Register 24	*on page 805*
0x005C—0x006B	Reserved	
0x006C	FDCR — FIFO DMA Control Register	*on page 806*
0x0070	FCR — FIFO Control Register	*on page 807*
0x0074	FTH — FIFO Threshold Register	*on page 808*
0x0078–0x007B	Reserved	
0x007C	FST — FIFO Status Register	*on page 809*
0x0080	FR0 — FIFO Right aligned data register 0	*on page 810*

Table 443. CTU memory map(Continued)

Offset from CTU_BASE (0xFFE0_C000)	Register	Location
0x0084	FR1 — FIFO Right aligned data register 1	*on page 810*
0x0088	FR2 — FIFO Right aligned data register 2	*on page 810*
0x008C	FR3 — FIFO Right aligned data register 3	*on page 810*
0x0080–0x009F	Reserved	
0x00A0	FL0 — FIFO Left aligned data register 0	*on page 811*
0x00A4	FL1 — FIFO Left aligned data register 1	*on page 811*
0x00A8	FL2 — FIFO Left aligned data register 2	*on page 811*
0x00AC	FL3 — FIFO Left aligned data register 3	*on page 811*
0x00B0–0x00BF	Reserved	
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	*on page 811*
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	*on page 812*
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	*on page 813*
0x00C6	COTR — Control ON-Time Register	*on page 814*
0x00C8	CTUCR — Cross triggering unit control register	*on page 815*
0x00CA	CTUDF — Cross Triggering Unit Digital Filter register	*on page 816*
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	*on page 817*
0x00CE–0x3FFF	Reserved	

Table 444. TGS registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x0000	TGSISR — Trigger Generator Subunit Input Selection Register	Yes	TGSISR_RE	0x0000_0000
0x0004	TGSCR — Trigger Generator Subunit Control Register	Yes	MRS	0x0000
0x0006	T0CR — Trigger 0 Compare Register	Yes	MRS	0x0000
0x0008	T1CR — Trigger 1 Compare Register	Yes	MRS	0x0000
0x000A	T2CR — Trigger 2 Compare Register	Yes	MRS	0x0000
0x000C	T3CR — Trigger 3 Compare Register	Yes	MRS	0x0000
0x000E	T4CR — Trigger 4 Compare Register	Yes	MRS	0x0000
0x0010	T5CR — Trigger 5 Compare Register	Yes	MRS	0x0000

Table 444. TGS registers(Continued)

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x0012	T6CR — Trigger 6 Compare Register	Yes	MRS	0x0000
0x0014	T7CR — Trigger 7 Compare Register	Yes	MRS	0x0000
0x0016	TGSCCR — TGS Counter Compare Register	Yes	MRS	0x0000
0x0018	TGSCRR — TGS Counter Reload Register	Yes	MRS	0x0000

Table 445. SU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x001C	CLCR1 — Commands List Control Register 1	Yes	MRS	0x0000_0000
0x0020	CLCR2 — Commands List Control Register 2	Yes	MRS	0x0000_0000
0x0024	THCR1 — Trigger Handler Control Register 1	Yes	MRS	0x0000_0000
0x0028	THCR2 — Trigger Handler Control Register 2	Yes	MRS	0x0000_0000
0x002C – 0x005A	CLRx — Commands List Register x (x = 1,...,24)	Yes	MRS	0x0000

Table 446. CTU registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x00C0	CTUEFR — Cross Triggering Unit Error Flag Register	No	—	0x0000
0x00C2	CTUIFR — Cross Triggering Unit Interrupt Flag Register	No	—	0x0000
0x00C4	CTUIR — Cross Triggering Unit Interrupt Register	No	—	0x0000
0x00C6	COTR — Control ON-Time Register	Yes	MRS	0x0000
0x00C8	CTUCR — Cross triggering unit control register	No	—	0x0000
0x00CA	CTUDF — Cross Triggering Unit Digital Filter	Yes	DFE	0x0000
0x00CC	CTUPCR — Cross Triggering Unit Power Control Register	No	—	0x0000

Table 447. FIFO registers

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x006C	FDCR — FIFO DMA Control Register	No	—	0x0000
0x0070	FCR — FIFO Control Register	No	—	0x0000_0000
0x0074	FTH — FIFO Threshold Register	No	—	0x0000_0000
0x007C	FST — FIFO Status Register	No	—	0x0000_0000
0x0080	FR0 — FIFO Right aligned data 0	No	—	0x0000_0000

Table 447. FIFO registers(Continued)

Offset from CTU_BASE	Register	Double-buffered	Synchronization	Reset value
0x0084	FR1 — FIFO Right aligned data 1	No	—	0x0000_0000
0x0088	FR2 — FIFO Right aligned data 2	No	—	0x0000_0000
0x008C	FR3 — FIFO Right aligned data 3	No	—	0x0000_0000
0x00A0	FL0 — FIFO Left aligned data 0	No	—	0x0000_0000
0x00A4	FL1 — FIFO Left aligned data 1	No	—	0x0000_0000
0x00A8	FL2 — FIFO Left aligned data 2	No	—	0x0000_0000
0x00AC	FL3 — FIFO Left aligned data 3	No	—	0x0000_0000

25.8.1 Trigger Generator Sub-unit Input Selection Register (TGSISR)

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	I15_FE	I15_RE	I14_FE	I14_RE	I13_FE	I13_RE	I12_FE	I12_RE	I11_FE	I11_RE	I10_FE	I10_RE	I9_FE	I9_RE	I8_FE	I8_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	I7_FE	I7_RE	I6_FE	I6_RE	I5_FE	I5_RE	I4_FE	I4_RE	I3_FE	I3_RE	I2_FE	I2_RE	I1_FE	I1_RE	I0_FE	I0_RE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 466. Trigger Generator Sub-unit Input Selection Register (TGSISR)

Table 448. TGSISR field descriptions

Field	Description
I15_FE	Input 15 — ext_signals Falling edge Enable 0 Disabled 1 Enabled
I15_RE	Input 15 — ext_signals Rising edge Enable 0 Disabled 1 Enabled
I14_FE	Input 14 — eTimer1 [ETC2] Falling edge Enable 0 Disabled 1 Enabled
I14_RE	Input 14 — eTimer1 [ETC2] Rising edge Enable 0 Disabled 1 Enabled
I13_FE	Input 13 — eTimer0 [ETC2] Falling edge Enable 0 Disabled 1 Enabled

Table 448. TGSISR field descriptions(Continued)

Field	Description
I13_RE	Input 13 — eTimer0 [ETC2] Rising edge Enable 0 Disabled 1 Enabled
I12_FE	Input 12 —PWM X[3] Falling edge Enable 0 Disabled 1 Enabled
I12_RE	Input 12 — PWM X[3] Rising edge Enable 0 Disabled 1 Enabled
I11_FE	Input 11 — PWM X[2] Falling edge Enable 0 Disabled 1 Enabled
I11_RE	Input 11 — PWM X[2] Rising edge Enable 0 Disabled 1 Enabled
I10_FE	Input 10 — PWM X[1] Falling edge Enable 0 Disabled 1 Enabled
I10_RE	Input 10 — PWM X[1] Rising edge Enable 0 Disabled 1 Enabled
I9_FE	Input 9 — PWM X[0] Falling edge Enable 0 Disabled 1 Enabled
I9_RE	Input 9 — PWM X[0] Rising edge Enable 0 Disabled 1 Enabled
I8_FE	Input 8 — PWM OUT_TRIG 1 [3] Falling edge Enable 0 Disabled 1 Enabled
I8_RE	Input 8 — PWM OUT_TRIG 1 [3] Rising edge Enable 0 Disabled 1 Enabled
I7_FE	Input 7 — PWM OUT_TRIG 1 [2] Falling edge Enable 0 Disabled 1 Enabled
I7_RE	Input 7 — PWM OUT_TRIG 1 [2] Rising edge Enable 0 Disabled 1 Enabled
I6_FE	Input 6 — PWM OUT_TRIG 1 [1] Falling edge Enable 0 Disabled 1 Enabled
I6_RE	Input 6 — PWM OUT_TRIG 1 [1] Rising edge Enable 0 Disabled 1 Enabled

Table 448. TGSISR field descriptions(Continued)

Field	Description
I5_FE	Input 5 — PWM OUT_TRIG 1 [0] Falling edge Enable 0 Disabled 1 Enabled
I5_RE	Input 5 — PWM OUT_TRIG 1 [0] Rising edge Enable 0 Disabled 1 Enabled
I4_FE	Input 4 — PWM OUT_TRIG 0 [3] Falling edge Enable 0 Disabled 1 Enabled
I4_RE	Input 4 — PWM OUT_TRIG 0 [3] Rising edge Enable 0 Disabled 1 Enabled
I3_FE	Input 3 — PWM OUT_TRIG 0 [2] Falling edge Enable 0 Disabled 1 Enabled
I3_RE	Input 3 — PWM OUT_TRIG 0 [2] Rising edge Enable 0 Disabled 1 Enabled
I2_FE	Input 2 — PWM OUT_TRIG 0 [1] Falling edge Enable 0 Disabled 1 Enabled
I2_RE	Input 2 — PWM OUT_TRIG 0 [1] Rising edge Enable 0 Disabled 1 Enabled h
I1_FE	Input 1 — PWM OUT_TRIG 0 [0] Falling edge Enable 0 Disabled 1 Enabled
I1_RE	Input 1 — PWM OUT_TRIG 0 [0] Rising edge Enable 0 Disabled 1 Enabled
I0_FE	Input 0 — PWM Reload Falling edge Enable 0 Disabled 1 Enabled
I0_RE	Input 0 — PWM Reload Rising edge Enable 0 Disabled 1 Enabled

25.8.2 Trigger Generator Sub-unit Control Register (TGSCR)

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	ET_TM	PRES		MRS_SM					TGS_M
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 467. Trigger Generator Sub-unit Control Register (TGSCR)

Table 449. TGSCR field descriptions

Field	Description
ET_TM	This bit enables toggle mode for external triggers.
PRES	TGS and SU prescaler selection bits 00 1 01 2 10 3 11 4
MRS_SM	Master Reload Selection in Sequential Mode (5 bits to select one of 32 inputs)
TGS_M	Trigger Generator Subunit Mode 0 Triggered Mode 1 Sequential Mode

25.8.3 Trigger x Compare Register (TxCR, *x* = 0...7)

Address: Base + 0x0006 (T0CR) Base + 0x000E (T4CR)
Base + 0x0008 (T1CR) Base + 0x0010 (T5CR)
Base + 0x000A (T2CR) Base + 0x0012 (T6CR)
Base + 0x000C (T3CR) Base + 0x0014 (T7CR)
Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TxCRV															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 468. Trigger x Compare Register (TxCR, *x* = 0...7)

Table 450. TxCR field descriptions

Field	Description
TxCRV	Trigger x Compare Register Value

25.8.4 TGS Counter Compare Register (TGSCCR)

Address: Base + 0x0016 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCCV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 469. TGS Counter Compare Register (TGSCCR)

Table 451. TGSCCR field format

Field	Description
TGSCCV	TGS Counter Compare Value

25.8.5 TGS Counter Reload Register (TGSCRR)

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TGSCRV															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 470. TGS Counter Reload Register (TGSCRR)

Table 452. TGSCRR field descriptions

Field	Description
TGSCRV	TGS Counter Reload Value

25.8.6 Commands list control register 1 (CLCR1)

Address: Base + 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_INDEX					0	0	0	T2_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_INDEX					0	0	0	T0_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 471. Commands list control register 1 (CLCR1)

Table 453. CLCR1 field descriptions

Field	Description
T3_INDEX	Trigger 3 Commands List first command address
T2_INDEX	Trigger 2 Commands List first command address
T1_INDEX	Trigger 1 Commands List first command address
T0_INDEX	Trigger 0 Commands List first command address

25.8.7 Commands list control register 2 (CLCR2)

Address: Base + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T7_INDEX					0	0	0	T6_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T5_INDEX					0	0	0	T4_INDEX				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 472. Commands list control register 2 (CLCR2)

Table 454. CLCR2 field descriptions

Field	Description
T7_INDEX	Trigger 7 Commands List first command address
T6_INDEX	Trigger 6 Commands List first command address
T5_INDEX	Trigger 5 Commands List first command address
T4_INDEX	Trigger 4 Commands List first command address

25.8.8 Trigger handler control register 1 (THCR1)

Address: Base + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T3_E	T3_ ETE	T3_ T1E	T3_ T0E	T3_ ADCE	0	0	0	T2_E	T2_ ETE	T2_ T1E	T2_ T0E	T2_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T1_E	T1_ ETE	T1_ T1E	T1_ T0E	T1_ ADCE	0	0	0	T0_E	T0_ ETE	T0_ T1E	T0_ T0E	T0_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 473. Trigger handler control register 1 (THCR1)

Table 455. THCR1 field descriptions

Field	Description
T3_E	Trigger 3 enable 0 Disabled 1 Enabled
T3_ETE	Trigger 3 External Trigger output enable 0 Disabled 1 Enabled
T3_T1E	Trigger 3 Timer 1 output enable 0 Disabled 1 Enabled
T3_T0E	Trigger 3 Timer 0 output enable 0 Disabled 1 Enabled
T3_ADCE	Trigger 3 ADC command output enable 0 Disabled 1 Enabled
T2_E	Trigger 2 enable 0 Disabled 1 Enabled
T2_ETE	Trigger 2 External Trigger output enable 0 Disabled 1 Enabled
T2_T1E	Trigger 2 Timer 1 output enable 0 Disabled 1 Enabled
T2_T0E	Trigger 2 Timer 0 output enable 0 Disabled 1 Enabled
T2_ADCE	Trigger 2 ADC command output enable 0 Disabled 1 Enabled
T1_E	Trigger 1 enable 0 Disabled 1 Enabled
T1_ETE	Trigger 1 External Trigger output enable 0 Disabled 1 Enabled
T1_T1E	Trigger 1 Timer 1 output enable 0 Disabled 1 Enabled
T1_T0E	Trigger 1 Timer 0 output enable 0 Disabled 1 Enabled
T1_ADCE	Trigger 1 ADC command output enable 0 Disabled 1 Enabled

Table 455. THCR1 field descriptions(Continued)

Field	Description
T0_E	Trigger 0 enable 0 Disabled 1 Enabled
T0_ETE	Trigger 0 External Trigger output enable 0 Disabled 1 Enabled
T0_T1E	Trigger 0 Timer 1 output enable 0 Disabled 1 Enabled
T0_T0E	Trigger 0 Timer 0 output enable 0 Disabled 1 Enabled
T0_ADCE	Trigger 0 ADC command output enable 0 Disabled 1 Enabled

25.8.9 Trigger handler control register 2 (THCR2)

Address: Base + 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	T7_E	T7_ ETE	T7_ T1E	T7_ T0E	T7_ ADC E	0	0	0	T6_E	T6_ ETE	T6_ T1E	T6_ T0E	T6_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	T5_E	T5_ ETE	T5_ T5E	T5_ T0E	T5_ ADCE	0	0	0	T4_E	T4_ ETE	T4_ T1E	T4_ T0E	T4_ ADCE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 474. Trigger handler control register 2 (THCR2)

Table 456. THCR2 field descriptions

Field	Description
T7_E	Trigger 7 enable 0 Disabled 1 Enabled
T7_ETE	Trigger 7 External Trigger output enable 0 Disabled 1 Enabled
T7_T1E	Trigger 7 Timer 1 output enable 0 Disabled 1 Enabled

Table 456. THCR2 field descriptions(Continued)

Field	Description
T7_T0E	Trigger 7 Timer 0 output enable 0 Disabled 1 Enabled
T7_ADCE	Trigger 7 ADC command output enable 0 Disabled 1 Enabled
T6_E	Trigger 6 enable 0 Disabled 1 Enabled
T6_ETE	Trigger 6 External Trigger output enable 0 Disabled 1 Enabled
T6_T1E	Trigger 6 Timer 1 output enable 0 Disabled 1 Enabled
T6_T0E	Trigger 6 Timer 0 output enable 0 Disabled 1 Enabled
T6_ADCE	Trigger 6 ADC command output enable 0 Disabled 1 Enabled
T5_E	Trigger 5 enable 0 Disabled 1 Enabled
T5_ETE	Trigger 5 External Trigger output enable 0 Disabled 1 Enabled
T5_T1E	Trigger 5 Timer 1 output enable 0 Disabled 1 Enabled
T5_T0E	Trigger 5 Timer 0 output enable 0 Disabled 1 Enabled
T5_ADCE	Trigger 5 ADC command output enable 0 Disabled 1 Enabled
T4_E	Trigger 4 enable 0 Disabled 1 Enabled
T4_ETE	Trigger 4 External Trigger output enable 0 Disabled 1 Enabled
T4_T1E	Trigger 4 Timer 1 output enable 0 Disabled 1 Enabled

Table 456. THCR2 field descriptions(Continued)

Field	Description
T4_T0E	Trigger 4 Timer 0 output enable 0 Disabled 1 Enabled
T4_ADCE	Trigger 4 ADC command output enable 0 Disabled 1 Enabled

25.8.10 Commands list register x (x = 1,...,24) (CLR*x*)

Figure 475 and *Table 457* show the register configured for ADC command format in single conversion mode (CMS = 0) (CLR*x*).

Address: Base + 0x002C,...,0x005A (See *Table 443*)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	0	0	0	SU	0	CH			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 475. Commands list register *x* (*x* = 1,...,24) (CMS = 0)

Table 457. CLR*x* (CMS = 0) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection bit 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit 0/1 00 FIFO 0 selected 01 FIFO 1 selected 10 FIFO 2 selected 11 FIFO 3 selected
SU	Selection Unit bit 0 ADC unit 0 selected 1 ADC unit 1 selected
CH	ADC unit channel number

Figure 476 and *Table 458* show the register configured for ADC command format in dual conversion mode (CMS = 1).

Address: Base + 0x002C ... 0x005A (See *Table 443*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIR	FC	CMS	0	FIFO		0	CH_B				0	CH_A			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 476. Commands list register x (x = 1,...,24) (CMS = 1)

Table 458. CLRx (CMS = 1) field descriptions

Field	Description
CIR	Command Interrupt Request bit 0 Disabled 1 Enabled
FC	First command bit 0 Not first command 1 First command
CMS	Conversion mode selection 0 Single conversion mode 1 Dual conversion mode
FIFO	FIFO for ADC unit A/B
CH_B	ADC unit B channel number
CH_A	ADC unit A channel number

25.8.11 FIFO DMA control register (FDCR)

Address: Base + 0x006C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	DE3	DE2	DE1	DE0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 477. FIFO DMA control register (FDCR)

Table 459. FDCR field descriptions

Name	Description
DEx	This bit enables DMA for the FIFOx 0 Disabled 1 Enabled

25.8.12 FIFO control register (FCR)

Address: Base + 0x0070 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OR_EN3	OF_EN3	EMPTY_EN3	FULL_EN3	OR_EN2	OF_EN2	EMPTY_EN2	FULL_EN2	OR_EN1	OF_EN1	EMPTY_EN1	FULL_EN1	OR_EN0	OF_EN0	EMPTY_EN0	FULL_EN0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 478. FIFO control register (FCR)

Table 460. FCR field descriptions

Field	Description
OR_EN3	FIFO 3 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN3	FIFO 3 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN3	FIFO 3 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN3	FIFO 3 Full interrupt enable 0 Disabled 1 Enabled
OR_EN2	FIFO 2 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN2	FIFO 2 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN2	FIFO 2 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN2	FIFO 2 Full interrupt enable 0 Disabled 1 Enabled
OR_EN1	FIFO 1 Overrun interrupt enable 0 Disabled 1 Enabled

Table 460. FCR field descriptions(Continued)

Field	Description
OF_EN1	FIFO 1 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN1	FIFO 1 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN1	FIFO 1 Full interrupt enable 0 Disabled 1 Enabled
OR_EN0	FIFO 0 Overrun interrupt enable 0 Disabled 1 Enabled
OF_EN0	FIFO 0 threshold Overflow interrupt enable 0 Disabled 1 Enabled
EMPTY_EN0	FIFO 0 Empty interrupt enable 0 Disabled 1 Enabled
FULL_EN0	FIFO 0 Full interrupt enable 0 Disabled 1 Enabled

25.8.13 FIFO threshold register (FTH)

Address: Base + 0x0074 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TH3								TH2							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TH1								TH0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 479. FIFO threshold register (FTH)

Table 461. FTH field descriptions

Field	Description
TH3	FIFO 3 Threshold
TH2	FIFO 2 Threshold
TH1	FIFO 1 Threshold
TH0	FIFO 0 Threshold

25.8.14 FIFO status register (FST)

Address: Base + 0x007C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OR3	OF3	EMP3	FULL3	OR2	OF2	EMP2	FULL2	OR1	OF1	EMP1	FULL1	OR0	OF0	EMP0	FULL0
W	r1c				r1c				r1c				r1c			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 480. FIFO status register (FST)

Table 462. FST field descriptions

Field	Description
OR3	FIFO 3 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF3	FIFO 3 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP3	FIFO 3 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL3	FIFO 3 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR2	FIFO 2 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF2	FIFO 2 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP2	FIFO 2 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL2	FIFO 2 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR1	FIFO 1 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 462. FST field descriptions(Continued)

Field	Description
OF1	FIFO 1 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP1	FIFO 1 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL1	FIFO 1 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
OR0	FIFO 0 Overrun interrupt flag A read of this bit clears it. 0 Interrupt has not occurred. 1 Interrupt has occurred.
OF0	FIFO 0 threshold Overflow interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
EMP0	FIFO 0 Empty interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
FULL0	FIFO 0 Full interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

25.8.15 FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Address: Base + 0x0080,...,0x008C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	N_CH[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	DATA									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 481. FIFO Right aligned data *x* (*x* = 0,...,3) (FR*x*)

Table 463. FR*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

25.8.16 FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Address: Base + 0x00A0,...,0x00AC Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	N_CH[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	DATA										0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 482. FIFO signed Left aligned data *x* (*x* = 0,...,3) (FL*x*)

Table 464. FL*x* field descriptions

Field	Description
N_CH[4:0]	Number of stored channel 0xxxx: Result comes from an ADC_1 channel 1xxxx: Result comes from an ADC_0 channel
DATA	Data of stored channel

25.8.17 Cross triggering unit error flag register (CTUEFR)

Address: Base + 0x00C0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	ET_OE	T1_OE	T0_OE	ADC_OE	TGS_OSM	MRS_O	ICE	SM_TO	MRS_RE
W								w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 483. Cross triggering unit error flag register (CTUEFR)

Table 465. CTUEFR field descriptions

Field	Description
ET_OE	External Trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
T1_OE	Timer 1 trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
T0_OE	Timer 0 trigger generation Overrun Error 0 Error has not occurred. 1 Error has occurred.

Table 465. CTUEFR field descriptions(Continued)

Field	Description
ADC_OE	ADC command generation Overrun Error 0 Error has not occurred. 1 Error has occurred.
TGS_OSM	TGS Overrun in Sequential Mode 0 Error has not occurred. 1 Error has occurred.
MRS_O	Master Reload Signal Overrun 0 Error has not occurred. 1 Error has occurred.
ICE	Invalid Command Error 0 Error has not occurred. 1 Error has occurred.
SM_TO	Trigger Overrun (more than 8 EV) in TGS Sequential Mode 0 Error has not occurred. 1 Error has occurred.
MRS_RE	Master Reload Signal Reload Error 0 Error has not occurred. 1 Error has occurred.

25.8.18 Cross triggering unit interrupt flag register (CTUIFR)

Address: Base + 0x00C2 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	ADC_I	T7_I	T6_I	T5_I	T4_I	T3_I	T2_I	T1_I	T0_I	MRS_I
W							r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c	r1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 484. Cross triggering unit interrupt flag register (CTUIFR)

Table 466. CTUIFR field descriptions

Field	Description
ADC_I	ADC command interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T7_I	Trigger 7 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T6_I	Trigger 6 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T5_I	Trigger 5 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

Table 466. CTUIFR field descriptions(Continued)

Field	Description
T4_I	Trigger 4 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T3_I	Trigger 3 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T2_I	Trigger 2 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T1_I	Trigger 1 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
T0_I	Trigger 0 interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.
MRS_I	MRS Interrupt flag 0 Interrupt has not occurred. 1 Interrupt has occurred.

25.8.19 Cross triggering unit interrupt/DMA register (CTUIR)

Address: Base + 0x00C4 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_IE	T6_IE	T5_IE	T4_IE	T3_IE	T2_IE	T1_IE	T0_IE	0	0	0	0	0	MRS_DMAE	MRS_IE	IEE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 485. Cross triggering unit interrupt/DMA register (CTUIR)

Table 467. CTUIR field descriptions

Field	Description
T7_IE	Trigger 7 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T6_IE	Trigger 6 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T5_IE	Trigger 5 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T4_IE	Trigger 4 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled

Table 467. CTUIR field descriptions(Continued)

Field	Description
T3_IE	Trigger 3 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T2_IE	Trigger 2 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T1_IE	Trigger 1 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
T0_IE	Trigger 0 Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
MRS_DMAE	DMA transfer Enable on MRS occurrence if GRE bit is set 0 Interrupt disabled 1 Interrupt enabled
MRS_IE	MRS Interrupt Enable 0 Interrupt disabled 1 Interrupt enabled
IEE	Interrupt Error Enable 0 Interrupt disabled 1 Interrupt enabled

25.8.20 Control ON time register (COTR)

Address: Base + 0x00C6 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	COTR							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 486. Control ON time register (COTR)

Table 468. COTR field descriptions

Field	Description
COTR	Control ON-Time and Guard Time for external trigger

25.8.21 Cross triggering unit control register (CTUCR)

Address: Base + 0x00C8 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	T7_SG	T6_SG	T5_SG	T4_SG	T3_SG	T2_SG	T1_SG	T0_SG	CRU_ADC_R	CTU_ODIS	DFE	CGRE	FGRE	MRS_SG	GRE	TGSISR_RE
W														w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 487. Cross triggering unit control register (CTUCR)

Table 469. CTUCR field descriptions

Field	Description
T7_SG	Trigger 7 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T6_SG	Trigger 6 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T5_SG	Trigger 5 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T4_SG	Trigger 4 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T3_SG	Trigger 3 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T2_SG	Trigger 2 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T1_SG	Trigger 1 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
T0_SG	Trigger 0 Software Generated 0 Trigger has not been generated. 1 Trigger has been generated.
CRU_ADC_R	CTU/ADC state machine Reset 0 Reset has not been generated. 1 Reset has been generated.
CTU_ODIS	CTU Output Disable CTU_ODISCTU output disable 0 CTU output is enabled. 1 CTU output is disabled.

Table 469. CTUCR field descriptions(Continued)

Field	Description
DFE	Digital Filter Enable 0 Digital filter is disabled. 1 Digital filter is enabled. **Note:** This bit can be read by software and can only be set to 1 by software.
CGRE	Clear GRE Writing 1 clears GRE bit in this register. This bit will be cleard after GRE is cleared. **Note:** This bit can only be set to 1 by software.
FGRE	Flag GRE 0 General Reload has not occurred. 1 General Reload has occurred.
MRS_SG	MRS Software Generated Writing 1 generates MRS signal by software. This bit will be cleared after MRS is generated.
GRE	General Reload Enable 0 General reload is disabled. 1 General reload is enabled. **Note:** This bit can be read by software and can only be set to 1 by software.
TGSISR_RE	TGS Input Selection Register Reload Enable 0 Register reload is disabled. 1 Register reload is enabled. **Note:** This bit can be read by software and can only be set to 1 by software.

25.8.22 Cross triggering unit digital filter (CTUDF)

Address: Base + 0x00CA Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	N							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 488. Cross triggering unit digital filter (CTUDF)

Table 470. CTUDF field descriptions

Field	Description
0-7	Reserved
8-15 N	Digital Filter value (the external signal is considered at 1 if it is latched N time at 1 and is considered at 0 if it is latched N time at 0)

25.8.23 Cross triggering unit power control register (CTUPCR)

Address: Base + 0x00CC Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 489. Cross triggering unit power control register (CTUPCR)

Table 471. CTUPCR field descriptions

Field	Description
0-14	Reserved
15 MDIS	Module Disable

26 FlexPWM

26.1 Overview

The pulse width modulator module (PWM) contains four PWM submodules, each of which capable of controlling a single half-bridge power stage and four fault input channels.

This PWM is capable of controlling most motor types: AC induction motors (ACIM), Permanent Magnet AC motors (PMAC), both brushless (BLDC) and brush DC motors (BDC), switched (SRM) and variable reluctance motors (VRM), and stepper motors.

26.2 Features

- 16-bit resolution for center, edge-aligned, and asymmetrical PWMs
- PWM outputs can operate as complimentary pairs or independent channels
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half cycle reload capability
- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a “Force Out” event
- PWMX pin can optionally output a third PWM signal from each submodule
- Channels not used for PWM generation can be used for buffered output compare functions
- PWMXn channel not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- The option to supply the source for each complementary PWM signal pair from any of the following:
  - External digital pin
  - Internal timer channel
  - External ADC input, taking into account values set in ADC high and low limit registers.

26.3 Modes of operation

Care must be exercised when using this module in certain device operating modes. Some motors (such 3-phase AC motors) require regular software updates for proper operation. Failure to do so could result in destroying the motor or inverter. Because of this, PWM outputs are placed in their inactive states in STOP mode, and optionally under WAIT/HALT and debug modes. PWM outputs will be reactivated (assuming they were active to begin with) when these modes are exited.

Table 472. Modes when PWM operation is restricted

Mode	Description
STOP	Peripheral and CPU clocks are stopped. PWM outputs are driven inactive.
WAIT/HALT	CPU clocks are stopped while peripheral clocks continue to run. PWM outputs are driven inactive as a function of the WAITEN bit.(1)
DEBUG	CPU and peripheral clocks continue to run, but CPU maybe stalled for periods of time. PWM outputs are driven inactive as a function of the DBGEN bit.

1. WAIT mode may be called HALT mode at the SoC level.

26.4 Block diagrams

26.4.1 Module level

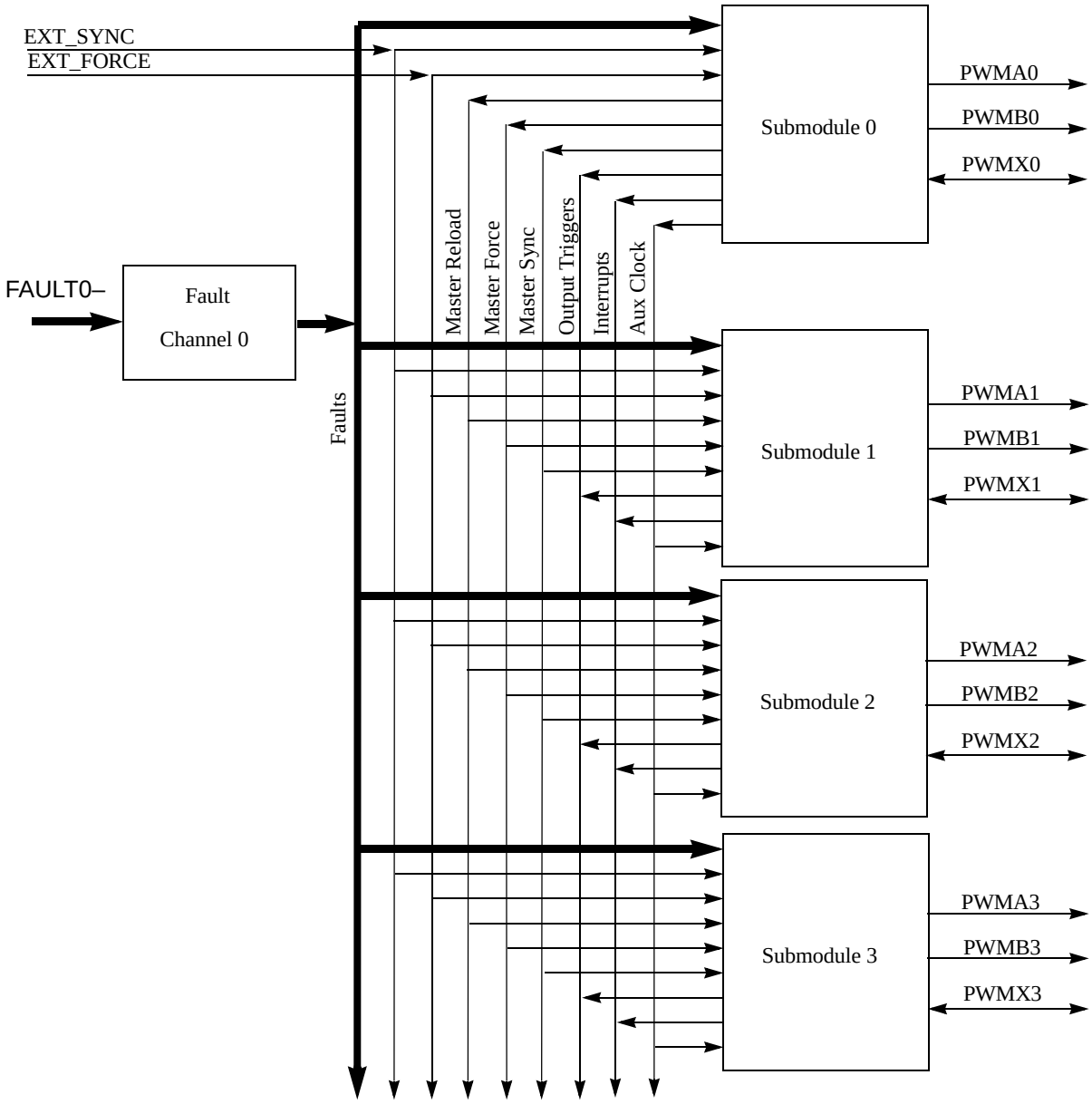


Figure 490. PWM block diagram

26.4.2 PWM submodule

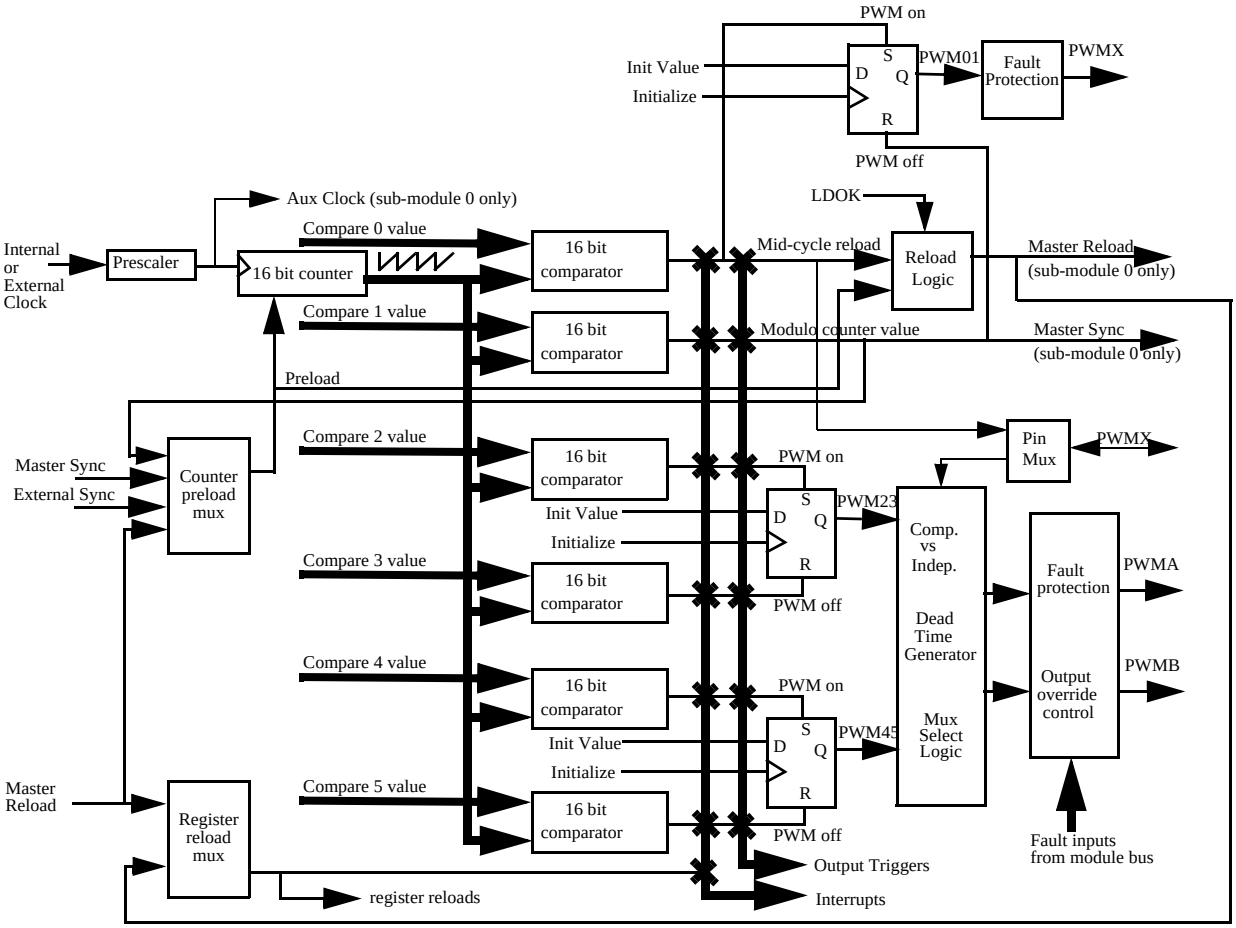


Figure 491. PWM submodule block diagram

26.5 External signal descriptions

The PWM module has external pins named PWMA[*n*], PWMB[*n*], PWMX[*n*], FAULT[*n*], EXT_SYNC. The PWM module also has on-chip inputs called EXT_CLK, EXT_FORCE and on-chip outputs called OUT_TRIG[*n*].

26.5.1 PWMA[*n*] and PWMB[*n*] — external PWM pair

These pins are the output pins of the PWM channels. They can be independent PWM signals or a complementary pair.

26.5.2 PWMX[*n*] — auxiliary PWM signal

These pins are the auxiliary output pins of the PWM channels. They can be independent PWM signals. When not needed as an output, they can be used as inputs to the input capture circuitry or they can be used to generate the IPOL bit during deadtime correction.

26.5.3 FAULT[*n*] — fault inputs

These are input pins for disabling selected PWM outputs.

26.5.4 EXT_SYNC — external synchronization signal

This input signal allows a source external to the PWM to initialize the PWM counter. In this manner the PWM can be synchronized to external circuitry.

26.5.5 EXT_FORCE — external output force signal

This input signal allows a source external to the PWM to force an update of the PWM outputs. In this manner the PWM can be synchronized to external circuitry. An example would be to simultaneously switch all of the PWM outputs on a commutation boundary for trapezoidal control of a BLDC motor. The source of EXT_FORCE signal is eTimer_0 OFLAG.

26.5.6 OUT_TRIG0[n] and OUT_TRIG1[n] — output triggers

These outputs allow the PWM submodules to control timing of the ADC conversions. See *Section 26.6.3.15: Output Trigger Control register (TCTRL)* for a description of how to enable these outputs and how the compare registers match up to the output triggers.

26.5.7 EXT_CLK — external clock signal

This on-chip input signal allows an on-chip source external to the PWM (typically a Timer) to control the PWM clocking. In this manner the PWM can be synchronized to the Timer. This signal must be generated synchronously to the PWM's clock since it is not resynchronized in the PWM.

26.6 Memory map and registers

26.6.1 FlexPWM module memory map

Table 473. FlexPWM memory map

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Location
0x0000	CNT—Counter Register (Submodule 0)	*on page 826*
0x0002	INIT—Initial Count Register (Submodule 0)	*on page 827*
0x0004	CTRL2—Control 2 Register (Submodule 0)	*on page 827*
0x0006	CTRL1—Control 1 Register (Submodule 0)	*on page 829*
0x0008	VAL0—Value Register 0 (Submodule 0)	*on page 831*
0x000A	VAL1—Value Register 1 (Submodule 0)	*on page 832*
0x000C	VAL2—Value Register 2 (Submodule 0)	*on page 832*
0x000E	VAL3—Value Register 3 (Submodule 0)	*on page 833*
0x0010	VAL4—Value Register 4 (Submodule 0)	*on page 833*

Table 473. FlexPWM memory map(Continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Location
0x0012	VAL5—Value Register 5 (Submodule 0)	on page 834
0x0014–0x0017	Reserved	
0x0018	OCTRL—Output Control Register (Submodule 0)	on page 834
0x001A	STS—Status Register (Submodule 0)	on page 835
0x001C	INTEN—Interrupt Enable Register (Submodule 0)	on page 836
0x001E	DMAEN—DMA Enable Register (Submodule 0)	on page 837
0x0020	TCTRL—Output Trigger Control Register (Submodule 0)	on page 838
0x0022	DISMAP—Fault Disable Mapping Register (Submodule 0)	on page 839
0x0024	DTCNT0—Deadtime Count Register 0 (Submodule 0)	on page 839
0x0026	DTCNT1—Deadtime Count Register 1 (Submodule 0)	on page 839
0x0028–0x002F	Reserved	
0x0030	CAPTCTRLX—Capture Control Register X (Submodule 0)	on page 840
0x0032	CAPTCOMPX—Capture Compare Register X (Submodule 0)	on page 842
0x0034	CVAL0—Capture Value 0 Register (Submodule 0)	on page 843
0x0036	CVAL0C—Capture Value 0 Cycle Register (Submodule 0)	on page 843
0x0038	CVAL1—Capture Value 1 Register (Submodule 0)	on page 843
0x003A	CVAL1C—Capture Value 1 Cycle Register (Submodule 0)	on page 844
0x003C–0x004F	Reserved	
0x0050	CNT—Counter Register (Submodule 1)	on page 826
0x0052	INIT—Initial Count Register (Submodule 1)	on page 827
0x0054	CTRL2—Control 2 Register (Submodule 1)	on page 827
0x0056	CTRL1—Control 1 Register (Submodule 1)	on page 829
0x0058	VAL0—Value Register 0 (Submodule 1)	on page 831
0x005A	VAL1—Value Register 1 (Submodule 1)	on page 832
0x005C	VAL2—Value Register 2 (Submodule 1)	on page 832
0x005E	VAL3—Value Register 3 (Submodule 1)	on page 833
0x0060	VAL4—Value Register 4 (Submodule 1)	on page 833
0x0062	VAL5—Value Register 5 (Submodule 1)	on page 834
0x0064–0x0067	Reserved	
0x0068	OCTRL—Output Control Register (Submodule 1)	on page 834
0x006A	STS—Status Register (Submodule 1)	on page 835
0x006C	INTEN—Interrupt Enable Register (Submodule 1)	on page 836
0x006E	DMAEN—DMA Enable Register (Submodule 1)	on page 837

Table 473. FlexPWM memory map(Continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Location
0x0070	TCTRL—Output Trigger Control Register (Submodule 1)	on page 838
0x0072	DISMAP—Fault Disable Mapping Register (Submodule 1)	on page 839
0x0074	DTCNT0—Deadtime Count Register 0 (Submodule 1)	on page 839
0x0076	DTCNT1—Deadtime Count Register 1 (Submodule 1)	on page 839
0x0078–0x007F	Reserved	
0x0080	CAPTCTRLX—Capture Control Register X (Submodule 1)	on page 840
0x0082	CAPTCOMPX—Capture Compare Register X (Submodule 1)	on page 842
0x0084	CVAL0—Capture Value 0 Register (Submodule 1)	on page 843
0x0086	CVAL0C—Capture Value 0 Cycle Register (Submodule 1)	on page 843
0x0088	CVAL1—Capture Value 1 Register (Submodule 1)	on page 843
0x008A	CVAL1C—Capture Value 1 Cycle Register (Submodule 1)	on page 844
0x008C–0x009F	Reserved	
0x00A0	CNT—Counter Register (Submodule 2)	on page 826
0x00A2	INIT—Initial Count Register (Submodule 2)	on page 827
0x00A4	CTRL2—Control 2 Register (Submodule 2)	on page 827
0x00A6	CTRL1—Control 1 Register (Submodule 2)	on page 829
0x00A8	VAL0—Value Register 0 (Submodule 2)	on page 831
0x00AA	VAL1—Value Register 1 (Submodule 2)	on page 832
0x00AC	VAL2—Value Register 2 (Submodule 2)	on page 832
0x00AE	VAL3—Value Register 3 (Submodule 2)	on page 833
0x00B0	VAL4—Value Register 4 (Submodule 2)	on page 833
0x00B2	VAL5—Value Register 5 (Submodule 2)	on page 834
0x00B4–0x00B7	Reserved	
0x00B8	OCTRL—Output Control Register (Submodule 2)	on page 834
0x00BA	STS—Status Register (Submodule 2)	on page 835
0x00BC	INTEN—Interrupt Enable Register (Submodule 2)	on page 836
0x00BE	DMAEN—DMA Enable Register (Submodule 2)	on page 837
0x00C0	TCTRL—Output Trigger Control Register (Submodule 2)	on page 838
0x00C2	DISMAP—Fault Disable Mapping Register (Submodule 2)	on page 839
0x00C4	DTCNT0—Deadtime Count Register 0 (Submodule 2)	on page 839
0x00C6	DTCNT1—Deadtime Count Register 1 (Submodule 2)	on page 839
0x00C8–0x00CF	Reserved	
0x00D0	CAPTCTRLX—Capture Control Register X (Submodule 2)	on page 840

Table 473. FlexPWM memory map(Continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Location
0x00D2	CAPTCOMPX—Capture Compare Register X (Submodule 2)	on page 842
0x00D4	CVAL0—Capture Value 0 Register (Submodule 2)	on page 843
0x00D6	CVAL0C—Capture Value 0 Cycle Register (Submodule 2)	on page 843
0x00D8	CVAL1—Capture Value 1 Register (Submodule 2)	on page 843
0x00DA	CVAL1C—Capture Value 1 Cycle Register (Submodule 2)	on page 844
0x00DC–0x00EF	Reserved	
0x00F0	CNT—Counter Register (Submodule 3)	on page 826
0x00F2	INIT—Initial Count Register (Submodule 3)	on page 827
0x00F4	CTRL2—Control 2 Register (Submodule 3)	on page 827
0x00F6	CTRL1—Control 1 Register (Submodule 3)	on page 829
0x00F8	VAL0—Value Register 0 (Submodule 3)	on page 831
0x00FA	VAL1—Value Register 1 (Submodule 3)	on page 832
0x00FC	VAL2—Value Register 2 (Submodule 3)	on page 832
0x00FE	VAL3—Value Register 3 (Submodule 3)	on page 833
0x0100	VAL4—Value Register 4 (Submodule 3)	on page 833
0x0102	VAL5—Value Register 5 (Submodule 3)	on page 834
0x0104–0x0107	Reserved	
0x0108	OCTRL—Output Control Register (Submodule 3)	on page 834
0x010A	STS—Status Register (Submodule 3)	on page 835
0x010C	INTEN—Interrupt Enable Register (Submodule 3)	on page 836
0x010E	DMAEN—DMA Enable Register (Submodule 3)	on page 837
0x0110	TCTRL—Output Trigger Control Register (Submodule 3)	on page 838
0x0112	DISMAP—Fault Disable Mapping Register (Submodule 3)	on page 839
0x0114	DTCNT0—Deadtime Count Register 0 (Submodule 3)	on page 839
0x0116	DTCNT1—Deadtime Count Register 1 (Submodule 3)	on page 839
0x0118–0x011F	Reserved	
0x0120	CAPTCTRLX—Capture Control Register X (Submodule 3)	on page 840
0x0122	CAPTCOMPX—Capture Compare Register X (Submodule 3)	on page 842
0x0124	CVAL0—Capture Value 0 Register (Submodule 3)	on page 843
0x0126	CVAL0C—Capture Value 0 Cycle Register (Submodule 3)	on page 843
0x0128	CVAL1—Capture Value 1 Register (Submodule 3)	on page 843
0x012A	CVAL1C—Capture Value 1 Cycle Register (Submodule 3)	on page 844
0x012C–0x013F	Reserved	

Table 473. FlexPWM memory map(Continued)

Offset from FlexPWM_BASE (0xFFE2_4000)	Register	Location
0x0140	OUTEN—Output Enable Register	*on page 844*
0x142	MASK—Output Mask Register	*on page 845*
0x0144	SWCOUT—Software Controlled Output Register	*on page 846*
0x0146	DTSRCSEL—Deadtime Source Select Register	*on page 847*
0x0148	MCTRL—Master Control Register	*on page 849*
0x014A	Reserved	
0x014C	FCTRL—Fault Control Register	*on page 850*
0x014E	FSTS—Fault Status Register	*on page 851*
0x0150	FFILT—Fault Filter Register	*on page 852*
0x0152–0x3FFF	Reserved	

26.6.2 Register descriptions

The address of a register is the sum of a base address and an address offset. The base address is defined at the core level and the address offset is defined at the module level. There are a set of registers for each PWM submodule, for the configuration logic, and for each Fault channel.

26.6.3 Submodule registers

These registers are repeated for each PWM submodule.

26.6.3.1 Counter Register (CNT)

Address: Base + 0x0000 (Submodule 0)
Base + 0x0050 (Submodule 1)
Base + 0x00A0 (Submodule 2)
Base + 0x00F0 (Submodule 3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 492. Counter Register (CNT)

This read-only register displays the state of the signed 16-bit submodule counter. This register is not byte accessible.

26.6.3.2 Initial Count Register (INIT)

Address: Base + 0x0002 (Submodule 0)
Base + 0x0052 (Submodule 1)
Base + 0x00A2 (Submodule 2)
Base + 0x00F2 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INIT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 493. Initial Count Register (INIT)

The 16-bit signed value in this register defines the initial count value for the PWM in PWM clock periods. This is the value loaded into the submodule counter when local sync, master sync, or master reload is asserted (based on the value of INIT_SEL) or when FORCE is asserted and force init is enabled. For PWM operation, the buffered contents of this register are loaded into the counter at the start of every PWM cycle. This register is not byte accessible.

Note: The INIT register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. This register cannot be written when LDOK is set. Reading INIT reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.3 Control 2 Register (CTRL2)

Address: Base + 0x0004 (Submodule 0)
Base + 0x0054 (Submodule 1)
Base + 0x00A4 (Submodule 2)
Base + 0x00F4 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBGEN	WAITEN	INDEP	PWMA_INIT	PWMB_INIT	PWMX_INIT	INIT_SEL		FRCEN	0 FORCE	FORCE_SEL			RELOAD_SEL	CLK_SEL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 494. Control 2 Register (CTRL2)

Table 474. CTRL2 field descriptions

Field	Description
0 DBGEN	Debug Enable When this bit is set, the PWM will continue to run while the device is in debug mode. If the device enters debug mode and this bit is cleared, then the PWM outputs are disabled until debug mode is exited. At that point, the PWM pins resume operation as programmed in the PWM registers. For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in debug mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (such as DC motors), this bit might safely be set, enabling the PWM in debug mode. The key point is that PWM parameter updates will not occur in debug mode. Any motors requiring such updates should be disabled during debug mode. If in doubt, leave this bit cleared.
1 WAITEN	WAIT Enable When this bit is set, the PWM continues to run while the device is in WAIT/HALT mode. In this mode, the peripheral clock continues to run, but the CPU clock does not. If the device enters WAIT/HALT mode and this bit is cleared, then the PWM outputs are disabled until WAIT/HALT mode is exited. At that point, the PWM pins resume operation as programmed in the PWM registers. For certain types of motors (such as 3-phase AC), it is imperative that this bit be left in its default state (in which the PWM is disabled in WAIT/HALT mode). Failure to do so could result in damage to the motor or inverter. For other types of motors (such as DC motors), this bit might safely be set, enabling the PWM in WAIT/HALT mode. The key point is PWM parameter updates will not occur in this mode. Any motors requiring such updates should be disabled during WAIT/HALT mode. If in doubt, leave this bit cleared.
2 INDEP	Independent or Complementary Pair Operation This bit determines whether the PWMA and PWMB channels will be independent PWMs or a complementary PWM pair. 0 PWMA and PWMB form a complementary PWM pair. 1 PWMA and PWMB outputs are independent PWMs.
3 PWMA_INIT	PWMA Initial Value This read/write bit determines the initial value for PWMA and the value to which it is forced when FORCE_INIT is asserted.
4 PWMB_INIT	PWMB Initial Value This read/write bit determines the initial value for PWMB and the value to which it is forced when FORCE_INIT is asserted.
5 PWMX_INIT	PWMX Initial Value This read/write bit determines the initial value for PWMX and the value to which it is forced when FORCE_INIT is asserted.
6:7 INIT_SEL	Initialization Control Select These read/write bits control the source of the INIT signal that goes to the counter. 00 Local sync (PWMX) causes initialization. 01 Master reload from submodule 0 causes initialization. This setting should not be used in submodule 0 as it will force the INIT signal to logic 0. 10 Master sync from submodule 0 causes initialization. This setting should not be used in submodule 0 as it will force the INIT signal to logic 0. 11 EXT_SYNC causes initialization.
8 FRCEN	Force Initialization Enable This bit allows the FORCE_OUT signal to initialize the counter without regard to the signal selected by INIT_SEL. This is a software controlled initialization. 0 Initialization from a Force Out event is disabled. 1 Initialization from a Force Out event is enabled.

Table 474. CTRL2 field descriptions(Continued)

Field	Description
9 FORCE	Force Initialization If the FORCE_SEL bits = 000, writing a 1 to this bit results in a Force Out event. This causes the following actions to be taken: – The PWMA and PWMB output pins will assume values based on the SELA and SELB bits. – If the FRCEN bit is set, the counter value will be initialized with the INIT register value.
10:12 FORCE_SEL	Force Source Select This read/write bit determines the source of the FORCE OUTPUT signal for this submodule. 000 The local force signal, FORCE, from this submodule is used to force updates. 001 The master force signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it will hold the FORCE OUTPUT signal to logic 0. 010 The local reload signal from this submodule is used to force updates. 011 The master reload signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it will hold the FORCE OUTPUT signal to logic 0. 100 The local sync signal from this submodule is used to force updates. 101 The master sync signal from submodule 0 is used to force updates. This setting should not be used in submodule 0 as it will hold the FORCE OUTPUT signal to logic 0. 110 The external force signal, EXT_FORCE, from outside the PWM module causes updates. 111 Reserved.
13 RELOAD_SEL	Reload Source Select This read/write bit determines the source of the RELOAD signal for this submodule. When this bit is set, the LDOK bit in submodule 0 should be used since the local LDOK bit will be ignored. 0 The local RELOAD signal is used to reload registers. 1 The master RELOAD signal (from submodule 0) is used to reload registers. This setting should not be used in submodule 0 as it will force the RELOAD signal to logic 0.
14:15 CLK_SEL	Clock Source Select These read/write bits determine the source of the clock signal for this submodule. 00 The IPBus clock is used as the clock for the local prescaler and counter. 01 EXT_CLK is used as the clock for the local prescaler and counter. 10 Submodule 0's clock (AUX_CLK) is used as the source clock for the local prescaler and counter. This setting should not be used in submodule 0 as it will force the clock to logic 0. 11 Reserved.

26.6.3.4 Control 1 Register (CTRL1)

Address: Base + 0x0006 (Submodule 0)
Base + 0x0056 (Submodule 1)
Base + 0x00A6 (Submodule 2)
Base + 0x00F6 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LDFQ				HALF	FULL	DT		0	PRSC			0	0	0	DBL EN
W																
Reset	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 495. Control 1 Register (CTRL1)

Table 475. CTRL1 field descriptions

Field	Description
0:3 LDFQ	Load Frequency These buffered read/write bits select the PWM load frequency according to *Table 476*. Reset clears the LDFQ bits, selecting loading every PWM opportunity. A PWM opportunity is determined by the HALF and FULL bits. The LDFQ*x* bits take effect when the current load cycle is complete regardless of the state of the LDOK bit. Reading the LDFQx bits reads the buffered values and not necessarily the values currently in effect. See *Table 476*.
4 HALF	Half Cycle Reload This read/write bit enables half-cycle reloads. A half cycle is defined by when the submodule counter matches the VAL0 register and does not have to be half way through the PWM cycle. 0 Half-cycle reloads disabled. 1 Half-cycle reloads enabled.
5 FULL	Full Cycle Reload This read/write bit enables full-cycle reloads. A full cycle is defined by when the submodule counter matches the VAL1 register. Either the HALF or FULL bit must be set in order to move the buffered data into the registers used by the PWM generators. If both the HALF and FULL bits are set, then reloads can occur twice per cycle. 0 Full-cycle reloads disabled. 1 Full-cycle reloads enabled.
6:7 DT	Deadtime These read only bits reflect the sampled values of the PWMX input at the end of each deadtime. Sampling occurs at the end of deadtime 0 for DT[0] and the end of deadtime 1 for DT[1]. Reset clears these bits.
9:11 PRSC	Prescaler These buffered read/write bits select the divide ratio of the PWM clock frequency selected by CLK_SEL as illustrated in *Table 477*
15 DBLEN	Double Switching Enable This read/write bit enables the double switching PWM behavior. 0 Double switching disabled. 1 Double switching enabled.

Table 476. PWM reload frequency

LDFQ	PWM reload frequency
0000	Every PWM opportunity
0001	Every 2 PWM opportunities
0010	Every 3 PWM opportunities
0011	Every 4 PWM opportunities
0100	Every 5 PWM opportunities
0101	Every 6 PWM opportunities
0110	Every 7 PWM opportunities
0111	Every 8 PWM opportunities
1000	Every 9 PWM opportunities
1001	Every 10 PWM opportunities

Table 476. PWM reload frequency(Continued)

LDFQ	PWM reload frequency
1010	Every 11 PWM opportunities
1011	Every 12 PWM opportunities
1100	Every 13 PWM opportunities
1101	Every 14 PWM opportunities
1110	Every 15 PWM opportunities
1111	Every 16 PWM opportunities

Table 477. PWM prescaler

PRSC	PWM clock frequency
000	f_{clk}
001	$f_{clk}/2$
010	$f_{clk}/4$
011	$f_{clk}/8$
100	$f_{clk}/16$
101	$f_{clk}/32$
110	$f_{clk}/64$
111	$f_{clk}/128$

Note: Reading the PRSCx bits reads the buffered values and not necessarily the values currently in effect. The PRSCx bits take effect at the beginning of the next PWM cycle and only when the load okay bit, LDOK, is set. This field cannot be written when LDOK is set.

26.6.3.5 Value register 0 (VAL0)

Address: Base + 0x0008 (Submodule 0)
Base + 0x0058 (Submodule 1)
Base + 0x00A8 (Submodule 2)
Base + 0x00F8 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 496. Value Register 0 (VAL0)

The 16-bit signed value in this buffered, read/write register defines the mid-cycle reload point for the PWM in PWM clock periods. This register is not byte accessible.

Note: The VAL0 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL0 cannot be written when LDOK is set. Reading VAL0 reads the value in a buffer. It is not necessarily the value the PWM generator is currently using.

26.6.3.6 Value register 1 (VAL1)

Address: Base + 0x000A (Submodule 0)
Base + 0x005A (Submodule 1)
Base + 0x00AA (Submodule 2)
Base + 0x00FA (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 497. Value Register 1 (VAL1)

The 16-bit signed value written to this register defines the modulo count value (maximum count) for the submodule counter. Upon reaching this count value, the counter will reload itself with the contents of the INIT register. This register is not byte accessible.

Note: The VAL1 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL1 cannot be written when LDOK is set. Reading VAL1 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.7 Value register 2 (VAL2)

Address: Base + 0x000C (Submodule 0)
Base + 0x005C (Submodule 1)
Base + 0x00AC (Submodule 2)
Base + 0x00FC (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL2															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 498. Value register 2 (VAL2)

The 16-bit signed value in this register defines the count value to set PWMA high (*Figure 491*). This register is not byte accessible.

Note: The VAL2 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL2 cannot be written when LDOK is set. Reading VAL2 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.8 Value register 3 (VAL3)

Address: Base + 0x000E (Submodule 0)
Base + 0x005E (Submodule 1)
Base + 0x00AE (Submodule 2)
Base + 0x00FE (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL3															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 499. Value register 3 (VAL3)

The 16-bit signed value in this register defines the count value to set PWMA low (*Figure 491*). This register is not byte accessible.

Note: The VAL3 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL3 cannot be written when LDOK is set. Reading VAL3 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.9 Value register 4 (VAL4)

Address: Base + 0x0010 (Submodule 0)
Base + 0x0060 (Submodule 1)
Base + 0x00B0 (Submodule 2)
Base + 0x0100 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL4															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 500. Value register 4 (VAL4)

The 16-bit signed value in this register defines the count value to set PWMB high (*Figure 491*). This register is not byte accessible.

Note: The VAL4 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL4 cannot be written when LDOK is set. Reading VAL4 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.10 Value register 5 (VAL5)

Address: Base + 0x0012 (Submodule 0)
Base + 0x0062 (Submodule 1)
Base + 0x00B2 (Submodule 2)
Base + 0x0102 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VAL5															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 501. Value register 5 (VAL5)

The 16-bit signed value in this register defines the count value to set PWMB low (*Figure 491*). This register is not byte accessible.

Note: The VAL5 register is buffered. The value written does not take effect until the LDOK bit is set and the next PWM load cycle begins. VAL5 cannot be written when LDOK is set. Reading VAL5 reads the value in a buffer and not necessarily the value the PWM generator is currently using.

26.6.3.11 Output Control register (OCTRL)

Address: Base + 0x0018 (Submodule 0)
Base + 0x0068 (Submodule 1)
Base + 0x00B8 (Submodule 2)
Base + 0x0108 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PWMA_IN	PWMB_IN	PWMX_IN	0	0	POLA	POLB	POLX	0	0	PWMAFS		PWMBFS		PWMXFS	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 502. Output Control register (OCTRL)

Table 478. OCTRL field descriptions

Field	Description
0 PWMA_IN	PWMA Input This read only bit shows the logic value currently being driven into the PWMA input.
1 PWMB_IN	PWMB Input This read only bit shows the logic value currently being driven into the PWMB input.
2 PWMX_IN	PWMX Input This read only bit shows the logic value currently being driven into the PWMX input.
5 POLA	PWMA Output Polarity This bit inverts the PWMA output polarity. 0 PWMA output not inverted. A high level on the PWMA pin represents the “on” or “active” state. 1 PWMA output inverted. A low level on the PWMA pin represents the “on” or “active” state.

Table 478. OCTRL field descriptions(Continued)

Field	Description
6 POLB	PWMB Output Polarity This bit inverts the PWMB output polarity. 0 PWMB output not inverted. A high level on the PWMB pin represents the “on” or “active” state. 1 PWMB output inverted. A low level on the PWMB pin represents the “on” or “active” state.
7 POLX	PWMX Output Polarity This bit inverts the PWMX output polarity. 0 PWMX output not inverted. A high level on the PWMX pin represents the “on” or “active” state. 1 PWMX output inverted. A low level on the PWMX pin represents the “on” or “active” state.
10:11 PWMAFS	PWMA Fault State These bits determine the fault state for the PWMA output during fault conditions and STOP mode. It may also define the output state during WAIT/HALT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1x Output is tristated.
12:13 PWMBFS	PWMB Fault State These bits determine the fault state for the PWMB output during fault conditions and STOP mode. It may also define the output state during WAIT/HALT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1x Output is tristated.
14:15 PWMXFS	PWMX Fault State These bits determine the fault state for the PWMX output during fault conditions and STOP mode. It may also define the output state during WAIT/HALT and DEBUG modes depending on the settings of WAITEN and DBGEN. 00 Output is forced to logic 0 state prior to consideration of output polarity control. 01 Output is forced to logic 1 state prior to consideration of output polarity control. 1x Output is tristated.

26.6.3.12 Status register (STS)

Address: Base + 0x001A (Submodule 0)
Base + 0x006A (Submodule 1)
Base + 0x00BA (Submodule 2)
Base + 0x010A (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	RUF	REF	RF	0	0	0	0	CFX1	CFX0	CMPF					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 503. Status register (STS)

Table 479. STS field descriptions

Field	Description
1 RUF	Registers Updated Flag This read only flag is set when one of the INIT, VALx, or PRSC fields has been written resulting in non-coherent data in the set of double buffered registers. Clear RUF by a proper reload sequence consisting of a reload signal while LDOK = 1. Reset clears RUF. 0 No register update has occurred since last reload. 1 At least one of the double buffered registers has been updated since the last reload.
2 REF	Reload Error Flag This read/write flag is set when a reload cycle occurs while LDOK is 0 and the double buffered registers are in a non-coherent state (RUF = 1). Clear REF by writing a logic one to the REF bit. Reset clears REF. 0 No reload error occurred. 1 Reload signal occurred with non-coherent data and LDOK = 0.
3 RF	Reload Flag This read/write flag is set at the beginning of every reload cycle regardless of the state of the LDOK bit. Clear RF by writing a logic one to the RF bit when VALDE is clear (non-DMA mode). RF can also be cleared by the DMA done signal when VALDE is set (DMA mode). Reset clears RF. 0 No new reload cycle since last RF clearing. 1 New reload cycle since last RF clearing.
8 CFX1	Capture Flag X1 This bit is set when the word count of the Capture X1 FIFO (CX1CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX1DE is clear (non-DMA mode) or by the DMA done signal if CX1DE is set (DMA mode). Reset clears this bit.
9 CFX0	Capture Flag X0 This bit is set when the word count of the Capture X0 FIFO (CX0CNT) exceeds the value of the CFXWM field. This bit is cleared by writing a one to this bit position if CX0DE is clear (non-DMA mode) or by the DMA done signal if CX0DE is set (DMA mode). Reset clears this bit.
10:15 CMPF	Compare Flags These bits are set when the submodule counter value matches the value of one of the VALx registers. Clear these bits by writing a 1 to a bit position. 0 No compare event has occurred for a particular VALx value. 1 A compare event has occurred for a particular VALx value.

26.6.3.13 Interrupt Enable register (INTEN)

Address: Base + 0x001C (Submodule 0)
Base + 0x006C (Submodule 1)
Base + 0x00BC (Submodule 2)
Base + 0x010C (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	REIE	RIE	0	0	0	0	CX1IE	CX0IE	CMPIE					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 504. Interrupt Enable register (INTEN)

Table 480. INTEN field descriptions

Field	Description
2 REIE	Reload Error Interrupt Enable This read/write bit enables the reload error flag (REF) to generate CPU interrupt requests. Reset clears RIE. 0 REF CPU interrupt requests disabled. 1 REF CPU interrupt requests enabled.
3 RIE	Reload Interrupt Enable This read/write bit enables the reload flag (RF) to generate CPU interrupt requests. Reset clears RIE. 0 RF CPU interrupt requests disabled. 1 RF CPU interrupt requests enabled.
8 CX1IE	Capture X 1 Interrupt Enable This bit allows the CFX1 flag to create an interrupt request to the CPU. Do not set both this bit and the CX1DE bit. 0 Interrupt request disabled for CFX1. 1 Interrupt request enabled for CFX1.
9 CX0IE	Capture X 0 Interrupt Enable This bit allows the CFX0 flag to create an interrupt request to the CPU. Do not set both this bit and the CX0DE bit. 0 Interrupt request disabled for CFX0. 1 Interrupt request enabled for CFX0.
10:15 CMPIE	Compare Interrupt Enables These bits enable the CMPF flags to cause a compare interrupt request to the CPU. 0 The corresponding CMPF bit will not cause an interrupt request 1 The corresponding CMPF bit will cause an interrupt request.

26.6.3.14 DMA Enable register (DMAEN)

Address: Base + 0x001E (Submodule 0)
Base + 0x006E (Submodule 1)
Base + 0x00BE (Submodule 2)
Base + 0x010E (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	VADE	FAND	CAPTDE		0	0	0	0	CX1DE	CX0DE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 505. DMA Enable register (DMAEN)

Table 481. DMAEN field descriptions

Field	Description
6 VALDE	Value Registers DMA Enable This read/write bit enables DMA write requests for the VALx registers when RF is set. Reset clears VALDE. 0 DMA write requests disabled. 1 DMA write requests for the VALx registers enabled.
7 FAND	FIFO Watermark AND Control This read/write bit works in conjunction with the CAPTDE field when it is set to watermark mode (CAPTDE = 01). While the CXxDE bits determine which FIFO watermarks the DMA read request is sensitive to, this bit determines if the selected watermarks are ANDed together or ORed together in order to create the request. 0 Selected FIFO watermarks are ORed together. 1 Selected FIFO watermarks are ANDed together.
8:9 CAPTDE	Capture DMA Enable Source Select These read/write bits select the source of enabling the DMA read requests for the capture FIFOs. Reset clears these bits. 00 Read DMA requests disabled. 01 Exceeding a FIFO watermark sets the DMA read request. This requires at least 1 of the CX1DE or CX0DE bits to also be set in order to determine which watermark(s) the DMA request is sensitive to. 10 A local sync (VAL1 matches counter) sets the read DMA request. 11 A local reload (RF being set) sets the read DMA request.
14 CX1DE	Capture X1 FIFO DMA Enable This read/write bit enables DMA read requests for the Capture X1 FIFO data when CFX1 is set. Reset clears CX1DE. Do not set both this bit and the CX1IE bit.
15 CX0DE	Capture X0 FIFO DMA Enable This read/write bit enables DMA read requests for the Capture X0 FIFO data when CFX0 is set. Reset clears CX0DE. Do not set both this bit and the CX0IE bit.

26.6.3.15 Output Trigger Control register (TCTRL)

Address: Base + 0x0020 (Submodule 0)
Base + 0x0070 (Submodule 1)
Base + 0x00C0 (Submodule 2)
Base + 0x0110 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	OUT_TRIG_EN[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 506. Output Trigger Control register (TCTRL)

Table 482. TCTRL field descriptions

Field	Description
10:15 OUT_TRIG_EN[5:0]	Output Trigger Enables These bits enable the generation of OUT_TRIG0 and OUT_TRIG1 outputs based on the counter value matching the value in one or more of the VAL0-5 registers where OUT_TRIG_EN[0] refers to VAL0, OUT_TRIG_EN[1] refers to VAL1 and so on. VAL0, VAL2, and VAL4 are used to generate OUT_TRIG0 and VAL1, VAL3, and VAL5 are used to generate OUT_TRIG1. The OUT_TRIGx signals are only asserted as long as the counter value matches the VALx value, therefore as many as six triggers can be generated (three each on OUT_TRIG0 and OUT_TRIG1) per PWM cycle per submodule. 0 OUT_TRIGx will not set when the counter value matches the VALx value. 1 OUT_TRIGx will set when the counter value matches the VALx value.

26.6.3.16 Fault Disable Mapping register (DISMAP)

This register determines which PWM pins are disabled by the fault protection inputs, illustrated in *Figure 548* in *Section 26.8.13: Fault protection*. Reset sets all of the bits in the fault disable mapping register.

Address: Base + 0x0022 (Submodule 0)
Base + 0x0072 (Submodule 1)
Base + 0x00C2 (Submodule 2)
Base + 0x0112 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	1	DISX[3:0]				DISB[3:0]				DISA[3:0]			
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 507. Fault Disable Mapping register (DISMAP)

Table 483. DISMAP field descriptions

Field	Description
4:7 DISX	PWMX Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMX output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISX field. A reset sets all DISX bits.
8:11 DISB	PWMB Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMB output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISB field. A reset sets all DISB bits.
12:15 DISA	PWMA Fault Disable Mask Each of the 4 bits of this read/write field is one-to-one associated with the four FAULTx inputs. The PWMA output will be turned off if there is a logic 1 on a FAULTx input and a 1 in the corresponding bit of the DISA field. A reset sets all DISA bits.

26.6.3.17 Deadtime Count registers (DTCNT0, DTCNT1)

Deadtime operation is only applicable to complementary channel operation. Reset sets the deadtime count registers to a default value of 0x07FF, selecting a deadtime of 4095 peripheral clock cycles. These registers are not byte accessible.

Address: Base + 0x0024 (Submodule 0)
Base + 0x0074 (Submodule 1)
Base + 0x00C4 (Submodule 2)
Base + 0x0114 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT0										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 508. Deadtime Count Register 0 (DTCNT0)

Table 484. DTCNT0 field descriptions

Field	Description
5:15 DTCNT0	Deadtime Count 0 This field controls the deadtime during 0 to 1 transitions of the PWMA output (assuming normal polarity).

Address: Base + 0x0026 (Submodule 0)
Base + 0x0076 (Submodule 1)
Base + 0x00C6 (Submodule 2)
Base + 0x0116 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	DTCNT1										
W																
Reset	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1

Figure 509. Deadtime Count register 1 (DTCNT1)

Table 485. DTCNT1 field descriptions

Field	Description
5:15 DTCNT1	Deadtime Count 1 This field controls the deadtime during 0 to 1 transitions of the complementary PWMB output.

26.6.3.18 Capture Control X register (CAPTCTRLX)

Address: Base + 0x0030 (Submodule 0)
Base + 0x0080 (Submodule 1)
Base + 0x00D0 (Submodule 2)
Base + 0x0120 (Submodule 3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CX1CNT			CX0CNT			CFXWM		EDGCNTX_EN	INP_SELX	EDGX1		EDGX0		ONESHOTX	ARMX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 510. Capture Control X register (CAPTCTRLX)

Table 486. CAPTCTRLX field descriptions

Field	Description
0:2 CX1CNT	Capture X1 FIFO Word Count This field reflects the number of words in the CVAL1 FIFO.
3:5 CX0CNT	Capture X0 FIFO Word Count This field reflects the number of words in the CVAL0 FIFO.
6:7 CFXWM	Capture X FIFOs Water Mark This field represents the water mark level for capture X FIFOs. The capture flags, CFX1 and CFX0, are not set until the word count of the corresponding FIFO is greater than this water mark level.
8 EDGCNTX_EN	Edge Counter X Enable This bit enables the edge counter, which counts rising and falling edges on the PWMX input signal. 0 Edge counter disabled and held in reset. 1 Edge counter enabled.
9 INPSELX	Input Select X This bit selects between the raw PWMX input signal and the output of the edge counter/compare circuitry as the source for the input capture circuit. 0 Raw PWMX input signal selected as source. 1 Output of edge counter/compare selected as source. **Note:** When INPSELX = 1, the internal edge counter is enabled and the rising and/or falling edges specified by the EDGX0 and EDGX1 fields are ignored. The software must still place a value other than 00 in either or both of the EDGX0 and/or EDGX1 fields in order to enable one or both of the capture registers.
10:11EDGX1	Edge X 1 These bits control the input capture 1 circuitry by determining which input edges cause a capture event. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
12:13 EDGX0	Edge X 0 These bits control the input capture 0 circuitry by determining which input edges cause a capture event. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.

Table 486. CAPTCTRLX field descriptions(Continued)

Field	Description
14 ONESHOTX	One Shot Mode Aux This bit selects between free running and one shot mode for the input capture circuitry. 0 Free running mode is selected If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and capture circuit 0 is re-armed. The process continues indefinitely. If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit. 1 One shot mode is selected. If both capture circuits are enabled, then capture circuit 0 is armed first after the ARMX bit is set. Once a capture occurs, capture circuit 0 is disarmed and capture circuit 1 is armed. After capture circuit 1 performs a capture, it is disarmed and the ARMX bit is cleared. No further captures will be performed until the ARMX bit is set again. If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARMX bit is then cleared.
15 ARMX	Arm X Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one shot mode and the enabled capture circuit(s) has had a capture event(s). 0 Input capture operation is disabled. 1 Input capture operation as specified by the EDGX bits is enabled.

26.6.3.19 Capture Compare X register (CAPTCMPX)

Address: Base + 0x0032 (Submodule 0)
Base + 0x0082 (Submodule 1)
Base + 0x00D2 (Submodule 2)
Base + 0x0122 (Submodule 3)

Access: User read/write

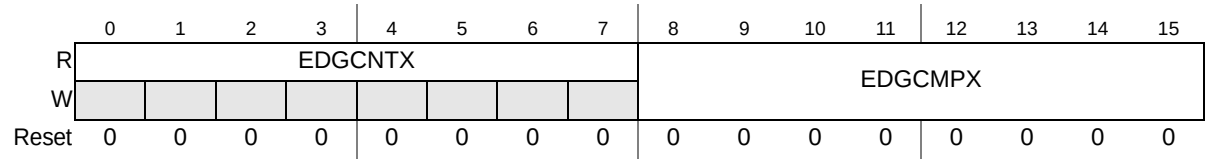


Figure 511. Capture Compare X register (CAPTCMPX)

Table 487. CAPTCMPX field descriptions

Field	Description
0:7 EDGCNTX	Edge Counter X This read only field contains the edge counter value for the PWMX input capture circuitry.
8:15 EDGCMPX	Edge Compare X This read/write field is the compare value associated with the edge counter for the PWMX input capture circuitry.

26.6.3.20 Capture Value 0 register (CVAL0)

Address: Base + 0x0034 (Submodule 0)
Base + 0x0084(Submodule 1)
Base + 0x00D4 (Submodule 2)
Base + 0x0124 (Submodule 3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPTVAL0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 512. Capture Value 0 register (CVAL0)

This read only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX0 bits. This is actually a 4-deep FIFO and not a single register. This register is not byte accessible.

26.6.3.21 Capture Value 0 Cycle register (CVAL0CYC)

Address: Base + 0x0036 (Submodule 0)
Base + 0x0086 (Submodule 1)
Base + 0x00D6 (Submodule 2)
Base + 0x0126 (Submodule 3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL0CYC		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 513. Capture Value 0 Cycle register (CVAL0CYC)

This read only register stores the cycle number corresponding to the value captured in CVAL0. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. This is actually a 4-deep FIFO and not a single register.

26.6.3.22 Capture Value 1 register (CVAL1)

Address: Base + 0x0038 (Submodule 0)
Base + 0x0088 (Submodule 1)
Base + 0x00D8 (Submodule 2)
Base + 0x0128 (Submodule 3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CAPTVAL1															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 514. Capture Value 1 register (CVAL1)

This read only register stores the value captured from the submodule counter. Exactly when this capture occurs is defined by the EDGX1 bits. This is actually a 4-deep FIFO and not a single register. This register is not byte accessible.

26.6.3.23 Capture Value 1 Cycle register (CVAL1CYC)

Address: Base + 0x003A (Submodule 0)
Base + 0x008A (Submodule 1)
Base + 0x00DA (Submodule 2)
Base + 0x012A (Submodule 3)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	CVAL1CYC		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 515. Capture Value 1 Cycle register (CVAL1CYC)

This read only register stores the cycle number corresponding to the value captured in CVAL1. The PWM cycle is reset to 0 and is incremented each time the counter is loaded with the INIT value. This is actually a 4-deep FIFO and not a single register.

26.6.4 Configuration registers

The base address of the configuration registers is equal to the base address of the PWM plus as offset of 0x140.

26.6.4.1 Output Enable register (OUTEN)

Address: Base + 0x0140

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	PWMA_EN[3:0]				PWMB_EN[3:0]				PWMX_EN[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 516. Output Enable register (OUTEN)

The relationship between the fields of OUTEN and the submodules is as follows:

- PWM*x*_EN[3] enables/disables submodule 3
- PWM*x*_EN[2] enables/disables submodule 2
- PWM*x*_EN[1] enables/disables submodule 1
- PWM*x*_EN[0] enables/disables submodule 0

Table 488. OUTEN field descriptions

Field	Description
4:7 PWMA_EN[3:0]	PWMA Output Enables These bits enable the PWMA outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMA pin is being used for input capture. 0 PWMA output disabled. 1 PWMA output enabled.
8:11 PWMB_EN[3:0]	PWMB Output Enables These bits enable the PWMB outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMB pin is being used for input capture. 0 PWMB output disabled. 1 PWMB output enabled.
12:15 PWMX_EN[3:0]	PWMX Output Enables These bits enable the PWMX outputs of each submodule. These bits should be set to 0 (output disabled) when a PWMX pin is being used for input capture or deadtime correction. 0 PWMX output disabled. 1 PWMX output enabled.

26.6.4.2 Mask register (MASK)

Address: Base + 0x0142 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	MASKA[3:0]				MASKB[3:0]				MASKX[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 517. Mask register (MASK)

The relationship between the fields of MASK and the submodules is as follows:

- MASK*x*[3] enables/disables submodule 3
- MASK*x*[2] enables/disables submodule 2
- MASK*x*[1] enables/disables submodule 1
- MASK*x*[0] enables/disables submodule 0

Table 489. MASK field descriptions

Field	Description
4:7 MASKA[3:0]	PWMA Masks These bits mask the PWMA outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMA output normal. 1 PWMA output masked.

Table 489. MASK field descriptions(Continued)

Field	Description
8:11 MASKB[3:0]	PWMB Masks These bits mask the PWMB outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMB output normal. 1 PWMB output masked.
12:15 MASKX[3:0]	PWMX Masks These bits mask the PWMX outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 PWMX output normal. 1 PWMX output masked.

Note: The MASKx bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 538 to see how FORCE_OUT is generated. Reading the MASK bits reads the buffered value and not necessarily the value currently in effect.

26.6.4.3 Software Controlled Output Register (SWCOUT)

Address: Base + 0x0144 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	OUTA_3	OUTB_3	OUTA_2	OUTB_2	OUTA_1	OUTB_1	OUTA_0	OUTB_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 518. Software Controlled Output Register (SWCOUT)

Table 490. SWCOUT field descriptions

Field	Description
8 OUTA_3	Software Controlled Output A_3 This bit is only used when SELA for submodule 3 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWMA.
9 OUTB_3	Software Controlled Output B_3 This bit is only used when SELB for submodule 3 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWMB.
10 OUTA_2	Software Controlled Output A_2 This bit is only used when SELA for submodule 2 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWMA.

Table 490. SWCOUT field descriptions(Continued)

Field	Description
11 OUTB_2	Software Controlled Output B_2 This bit is only used when SELB for submodule 2 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 2 instead of PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 2 instead of PWMB.
12 OUTA_1	Software Controlled Output A_1 This bit is only used when SELA for submodule 1 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWMA.
13 OUTB_1	Software Controlled Output B_1 This bit is only used when SELB for submodule 1 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 1 instead of PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 1 instead of PWMB.
14 OUTA_0	Software Controlled Output A_0 This bit is only used when SELA for submodule 0 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWMA. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWMA.
15 OUTB_0	Software Controlled Output BA_0 This bit is only used when SELB for submodule 0 is set to 0b10. It allows software control of which signal is supplied to the deadtime generator of that submodule. 0 A logic 0 is supplied to the deadtime generator of submodule 0 instead of PWMB. 1 A logic 1 is supplied to the deadtime generator of submodule 0 instead of PWMB.

Note: These bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 538 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

26.6.4.4 Deadtime Source Select Register (DTSRCSEL)

Address: Base + 0x0146 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SELA_3		SELB_3		SELA_2		SELB_2		SELA_1		SELB_1		SELA_0		SELB_0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 519. Deadtime Source Select Register (DTSRCSEL)

Table 491. DTSRCSEL field descriptions

Field	Description
1:0 SELA_3	PWMA_3 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_3 signal is used by the deadtime logic. 01 Inverted generated PWMA_3 signal is used by the deadtime logic. 10 OUTA_3 bit is used by the deadtime logic. 11 Reserved
2:3 SELB_3	PWMB_3 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 3 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_3 signal is used by the deadtime logic. 01 Inverted generated PWMB_3 signal is used by the deadtime logic. 10 OUTB_3 bit is used by the deadtime logic. 11 Reserved
4:5 SELA_2	PWMA_2 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_2 signal is used by the deadtime logic. 01 Inverted generated PWMA_2 signal is used by the deadtime logic. 10 OUTA_2 bit is used by the deadtime logic. 11 Reserved
6:7 SELB_2	PWMB_2 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 2 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_2 signal is used by the deadtime logic. 01 Inverted generated PWMB_2 signal is used by the deadtime logic. 10 OUTB_2 bit is used by the deadtime logic. 11 Reserved
8:9 SELA_1	PWMA_1 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_1 signal is used by the deadtime logic. 01 Inverted generated PWMA_1 signal is used by the deadtime logic. 10 OUTA_1 bit is used by the deadtime logic. 11 Reserved
10:11 SELB_1	PWMB_1 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 1 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_1 signal is used by the deadtime logic. 01 Inverted generated PWMB_1 signal is used by the deadtime logic. 10 OUTB_1 bit is used by the deadtime logic. 11 Reserved

Table 491. DTSRCSEL field descriptions(Continued)

Field	Description
12:13 SELA_0	PWMA_0 Control Select This field selects possible over-rides to the generated PWMA signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMA_0 signal is used by the deadtime logic. 01 Inverted generated PWMA_0 signal is used by the deadtime logic. 10 OUTA_0 bit is used by the deadtime logic. 11 Reserved
14:15 SELB_0	PWMB_0 Control Select This field selects possible over-rides to the generated PWMB signal in submodule 0 that will be passed to the deadtime logic upon the occurrence of a “Force Out” event in that submodule. 00 Generated PWMB_0 signal is used by the deadtime logic. 01 Inverted generated PWMB_0 signal is used by the deadtime logic. 10 OUTB_0 bit is used by the deadtime logic. 11 Reserved

Note: The deadtime source select bits are double buffered and do not take effect until a FORCE_OUT event occurs within the appropriate submodule. Refer to Figure 538 to see how FORCE_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

26.6.4.5 Master Control Register (MCTRL)

Address: Base + 0x0148 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IPOL				RUN				0	0	0	0	LDOK			
W									CLDOK							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 520. Master Control Register (MCTRL)

The relationship between the fields of MCTRL and the submodules is as follows:

- *Field*[3] refers to submodule 3
- *Field*[2] refers to submodule 2
- *Field*[1] refers to submodule 1
- *Field*[0] refers to submodule 0

Table 492. MCTRL field descriptions

Field	Description
0:3 IPOL[3:0]	Current Polarity This buffered read/write bit selects between PWMA and PWMB as the source for the generation of the complementary PWM pair output. IPOL is ignored in independent mode. PWMB (*Figure 491*) generates complementary PWM pairs. PWMA (*Figure 491*) generates complementary PWM pairs. **Note:** The IPOL bit does not take effect until a FORCE_OUT event takes place in the appropriate submodule. Reading the IPOL bit reads the buffered value and not necessarily the value currently in effect.
4:7 RUN[3:0]	Run This read/write bit enables the clocks to the PWM generator. When RUN equals zero, the submodule counter is reset. A reset clears RUN. 0 Do not load new values. 1 PWM generator enabled. **Note:** For proper initialization of the LDOK and RUN bits, see *Section 26.9.5: Initialization*.
8:11 CLDOK[3:0]	Clear Load Okay This write only bit clears the LDOK bit. Write a 1 to this location to clear the corresponding LDOK. If a reload occurs with LDOK set at the same time that CLDOK is written, then the reload will not be performed and LDOK will be cleared. This bit is self clearing and always reads as a 0.
12:15 LDOK[3:0]	Load Okay This read/set bit loads the PRSC bits of CTRL1 and the INIT, and VALx registers into a set of buffers. The buffered prescaler divisor, submodule counter modulus value, and PWM pulse width take effect at the next PWM reload. Set LDOK by reading it when it is logic zero and then writing a logic one to it. The VALx, INIT, and PRSC fields cannot be written while LDOK is set. LDOK is automatically cleared after the new values are loaded, or can be manually cleared before a reload by writing a logic 1 to CLDOK. This bit cannot be written with a zero. LDOK can be set in DMA mode when the DMA indicates that it has completed the update of all PRSC, INIT, VALx, fields. Reset clears LDOK. 0 Do not load new values. 1 Load prescaler, modulus, and PWM values. **Note:** For proper initialization of the LDOK and RUN bits, see *Section 26.9.5: Initialization*.

26.6.5 Fault channel registers

26.6.5.1 Fault Control Register (FCTRL)

Address: Base + 0x014C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FLVL				FAUTO				FSAFE				FIE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 521. Fault Control Register (FCTRL)

Table 493. FCTRL field descriptions

Field	Description
0:3 FLVL	Fault Level These read/write bits select the active logic level of the individual fault inputs. A reset clears FLVL. 0 A logic 0 on the fault input indicates a fault condition. 1 A logic 1 on the fault input indicates a fault condition.
4:7 FAUTO	Automatic Fault Clearing These read/write bits select automatic or manual clearing of faults. A reset clears FAUTO. 0 Manual fault clearing. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear at the start of a half cycle. This is further controlled by the FSAFE bits. 1 Automatic fault clearing. PWM outputs disabled by this fault are enabled when the FFPINx bit is clear at the start of a half cycle without regard to the state of FFLAGx bit.
8:11 FSAFE	Fault Safety Mode These read/write bits select the safety mode during manual fault clearing. A reset clears FSAFE. 0 Normal mode. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear at the start of a half cycle without regard to the state of the FFPINx bit. The PWM outputs disabled by this fault input will not be re-enabled until the actual FAULTx input signal de-asserts since the fault input will combinationally disable the PWM outputs (as programmed in DISMAP). 1 Safe mode. PWM outputs disabled by this fault are not enabled until the FFLAGx bit is clear and the FFPINx bit is clear at the start of a half cycle. **Note:** The FFPINx bit may indicate a fault condition still exists even though the actual fault signal at the FAULTx pin is clear due to the fault filter latency.
12:15 FIE	Fault Interrupt Enables This read/write bit enables CPU interrupt requests generated by the FAULTx pins. A reset clears FIE. 0 FAULTx CPU interrupt requests disabled. 1 FAULTx CPU interrupt requests enabled. **Note:** The fault protection circuit is independent of the FIEx bit and is always active. If a fault is detected, the PWM outputs are disabled according to the disable mapping register.

26.6.5.2 Fault Status Register (FSTS)

Address: Base + 0x014E Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	FTEST	FFPIN				0	0	0	0	FFLAG			
W																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Figure 522. Fault Status Register (FSTS)

Table 494. FSTS field descriptions

Field	Description
3 FTEST	Fault Test These read/write bits simulate a fault condition. Setting this bit will cause a simulated fault to be sent into all of the fault filters. The condition will propagate to the fault flags and possibly the PWM outputs depending on the DISMAP settings. Clearing this bit removes the simulated fault condition. 0 No fault. 1 Cause a simulated fault.
4:7 FFPIN	Filtered Fault Pins These read-only bits reflect the current state of the filtered FAULTx pins converted to high polarity. A logic 1 indicates a fault condition exists on the filtered FAULTx pin. A reset has no effect on FFPIN.
12:15 FFLAG	Fault Flags These read-only flags are set within 2 CPU cycles after a transition to active on the FAULTx pin. Clear FFLAGx by writing a logic one to it. A reset clears FFLAG. 0 No fault on the FAULTx pin. 1 Fault on the FAULTx pin. **Note:** The FFLAG[3:0] flags will be set out of reset. They should be cleared before enabling the Fault Control feature.

26.6.5.3 Fault Filter Register (FFILT)

Address: Base + 0x0150 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FILT_CNT			FILT_PER							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 523. Fault Filter Register (FFILT)

The settings in this register are shared among each of the fault input filters.

Table 495. FFILT field descriptions

Field	Description
5:7 FILT_CNT	Fault Filter Count These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in *Section 26.6.5.4: Input filter considerations*.
8:15 FILT_PER	Fault Filter Period These bits represent the sampling period (in IPBus clock cycles) of the fault pin input filter. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in *Section 26.6.5.4: Input filter considerations*.

26.6.5.4 Input filter considerations

The FILT_PER value should be set such that the sampling period is larger than the period of the expected noise. This way a noise spike will only corrupt one sample. The FILT_CNT

value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT+3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of ((FILT_CNT+4) x FILT_PER x IPBus clock period). Note that even when the filter is enabled, there is a combinational path to disable the PWM outputs. This is to ensure rapid response to fault conditions and also to ensure fault response if the PWM module loses its clock. The latency induced by the filter will be seen in the time to set the FFLAG and FFPIN bits of the FSTS register.

26.7 Functional description

26.7.1 Center-aligned PWMs

Each submodule has its own timer that is capable of generating PWM signals on two output pins. The edges of each of these signals are controlled independently as shown in *Figure 524*.

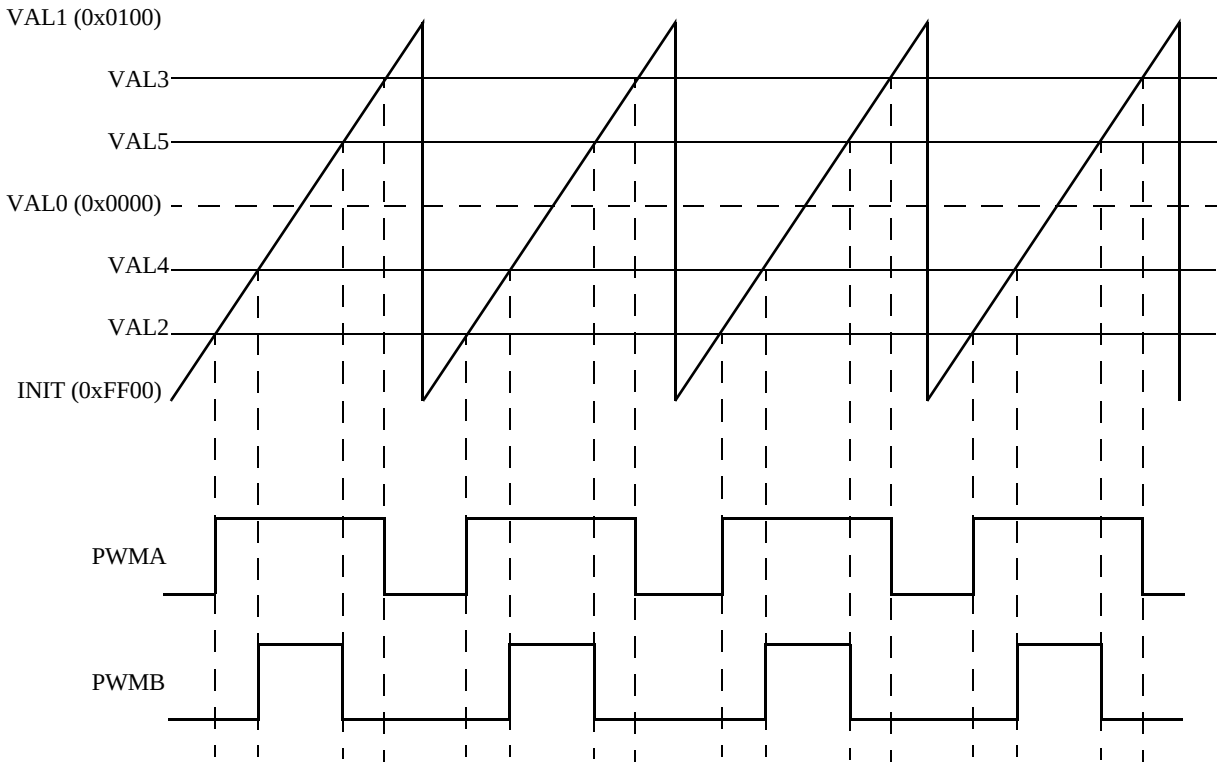


Figure 524. Center-aligned example

The submodule timers only count in the up direction and then reset to the INIT value. Instead of having a single value that determines pulse width, there are two values that must be specified: the turn on edge and the turn off edge. This double action edge generation not only gives the user control over the pulse width, but over the relative alignment of the signal

as well. As a result, there is no need to support separate PWM alignment modes since the PWM alignment mode is inherently a function of the turn on and turn off edge values.

Figure 524 also illustrates an additional enhancement to the PWM generation process. When the counter resets, it is reloaded with a user-specified value, which may or may not be zero. If the value chosen happens to be the 2's complement of the modulus value, then the PWM generator operates in "signed" mode. This means that if each PWM's turn on and turn off edge values are also the same number but only different in their sign, the "on" portion of the output signal will be centered around a count value of zero. Therefore, only one PWM value needs to be calculated in software and then this value and its negative are provided to the submodule as the turn off and turn on edges respectively. This technique will result in a pulse width that always consists of an odd number of timer counts. If all PWM signal edge calculations follow this same convention, then the signals will be center-aligned with respect to each other, which is the goal. Of course, center alignment between the signals is not restricted to symmetry around the zero count value, as any other number would also work. However, centering on zero provides the greatest range in signed mode and also simplifies the calculations.

26.7.2 Edge-aligned PWMs

When the turn on edge for each pulse is specified to be the INIT value, then edge-aligned operation results, as illustrated in *Figure 525*. Therefore, only the turn off edge value needs to be periodically updated to change the pulse width.

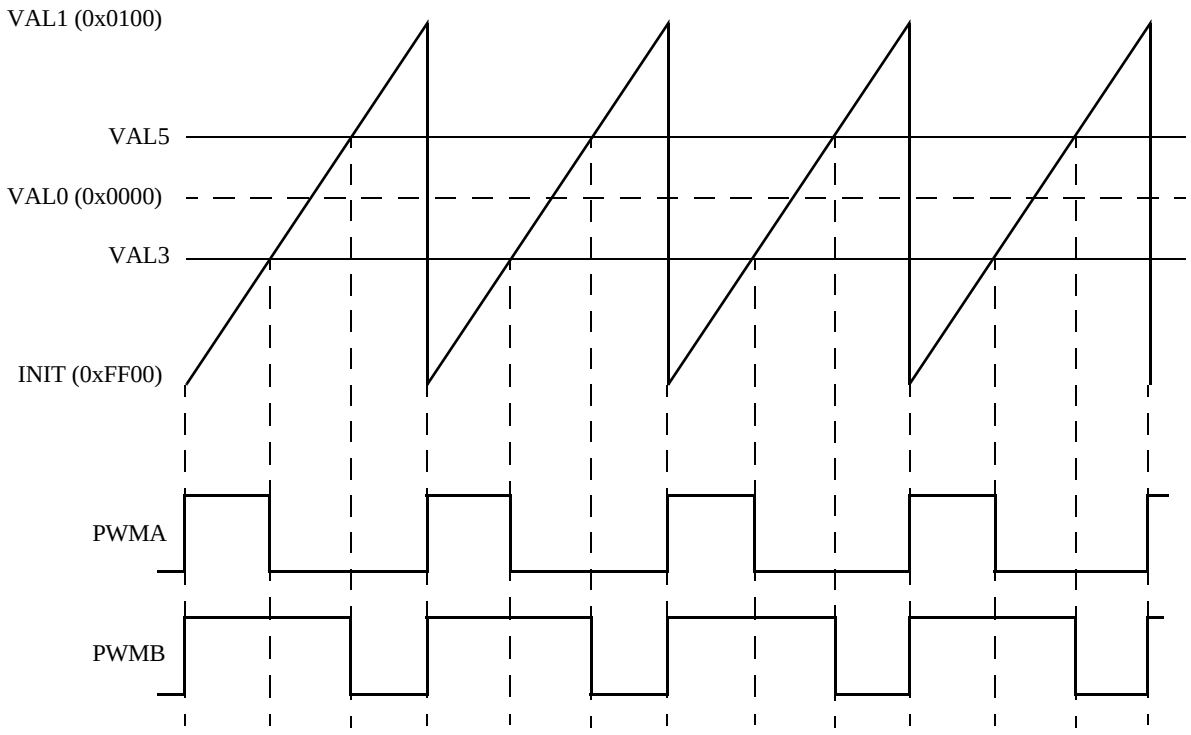


Figure 525. Edge-aligned example (INIT = VAL2 = VAL4)

With edge-aligned PWMs, another example of the benefits of signed mode can be seen. A common way to drive an H-bridge is to use a technique called "bipolar" PWMs where a 50%

duty cycle results in 0 volts on the load. Duty cycles less than 50% result in negative load voltages and duty cycles greater than 50% generate positive load voltages. If the module is set to signed mode operation (the INIT and VAL1 values are the same number with opposite signs), then there is a direct proportionality between the PWM turn off edge value and the motor voltage, INCLUDING the sign. So once again, signed mode of operation simplifies the software interface to the PWM module since no offset calculations are required to translate the output variable control algorithm to the voltage on an H-Bridge load.

26.7.3 Phase-shifted PWMs

In the previous sections, the benefits of signed mode of operation were discussed in the context of simplifying the required software calculations by eliminating the requirement to bias up signed variables before applying them to the module. However, if numerical biases are applied to the turn on and turn off edges of different PWM signal, the signals will be phase shifted with respect to each other, as illustrated in *Figure 526*. This results in certain advantages when applied to a power stage. For example, when operating a multi-phase inverter at a low modulation index, all of the PWM switching edges from the different phases occur at nearly the same time. This can be troublesome from a noise standpoint, especially if ADC readings of the inverter must be scheduled near those times. Phase shifting the PWM signals can open up timing windows between the switching edges to allow a signal to be sampled by the ADC. However, phase shifting does not affect the duty cycle so average load voltage is not affected.

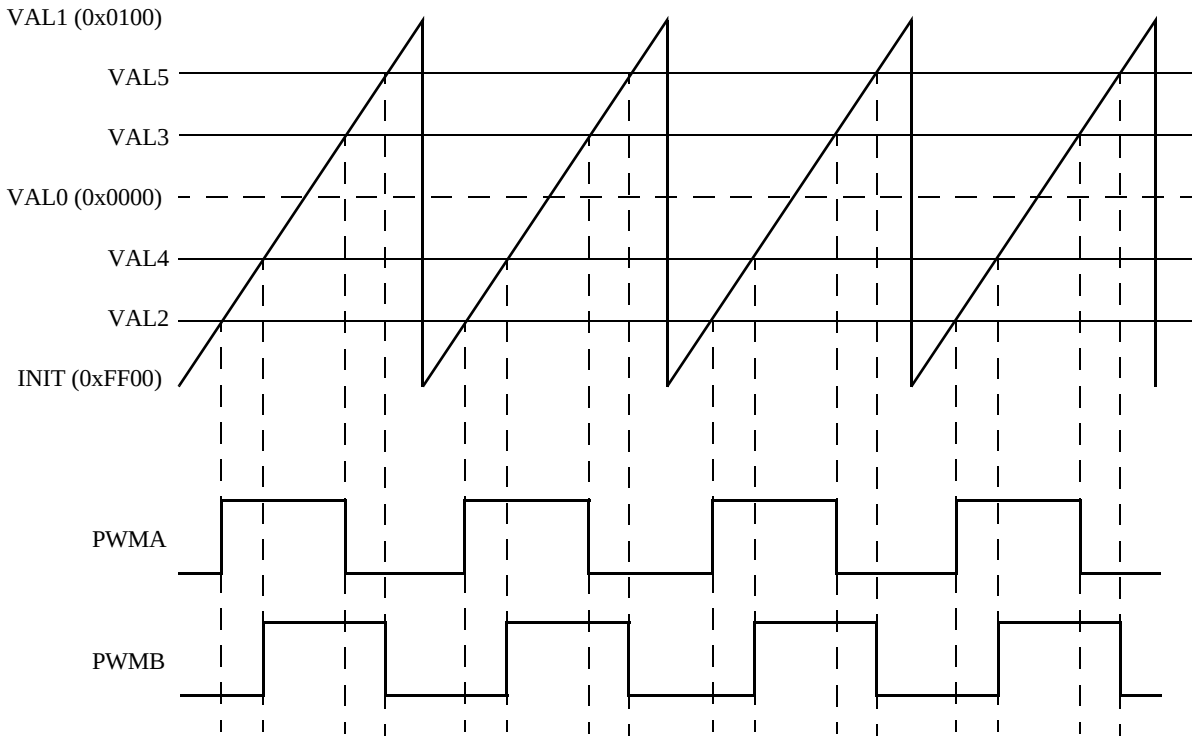


Figure 526. Phase-shifted outputs example

An additional benefit of phase-shifted PWMs can be seen in *Figure 527*. In this case, an H-bridge circuit is driven by four PWM signals to control the voltage waveform on the primary of a transformer. Both left and right side PWMs are configured to always generate a square

wave with 50% duty cycle. This works for the H-bridge since no narrow pulse widths are generated, reducing the high-frequency switching requirements of the transistors. Notice that the square wave on the right side of the H-Bridge is phase-shifted compared to the left side of the H-Bridge. As a result, the transformer primary sees the bottom waveform across its terminals. The RMS value of this waveform is directly controlled by the amount of phase shift of the square waves. Regardless of the phase shift, no DC component appears in the load voltage as long as the duty cycle of each square wave remains at 50%, making this technique ideally suited for transformer loads. As a result, this topology is frequently used in industrial welders to adjust the amount of energy delivered to the weld arc.

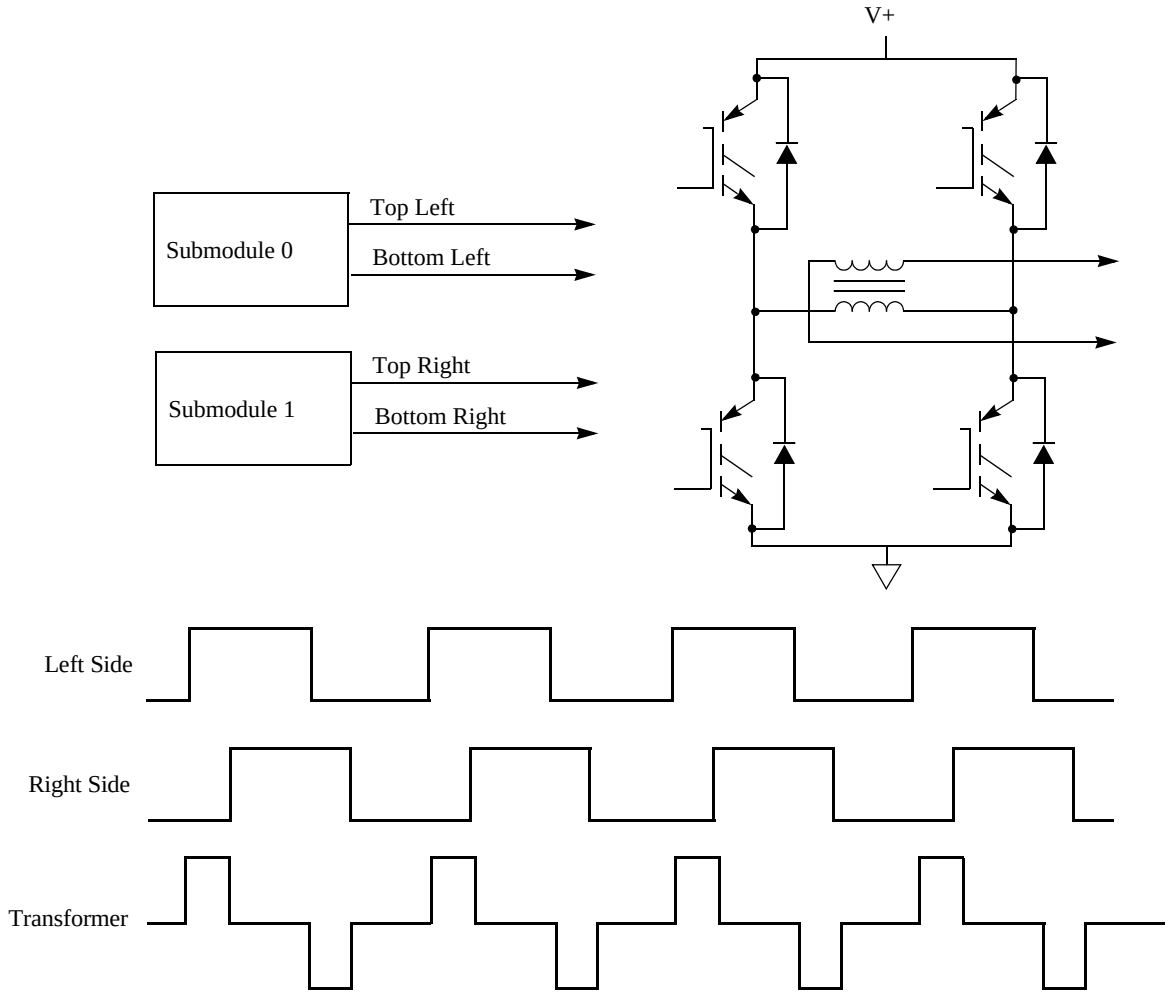


Figure 527. Phase-shifted PWMs applied to a transformer primary

26.7.4 Double switching PWMs

Double switching PWM output is supported to aid in single shunt current measurement and three phase reconstruction. This method support two independent rising edges and two independent falling edges per PWM cycle. The VAL2 and VAL3 registers generate the even channel (labeled as PWMA in the figure) while VAL4 and VAL5 generate the odd channel.

The two channels are combined using XOR logic (see *Figure 538*) as shown in *Figure 528*. The DBLPWM signal can be run through the deadtime insertion logic.

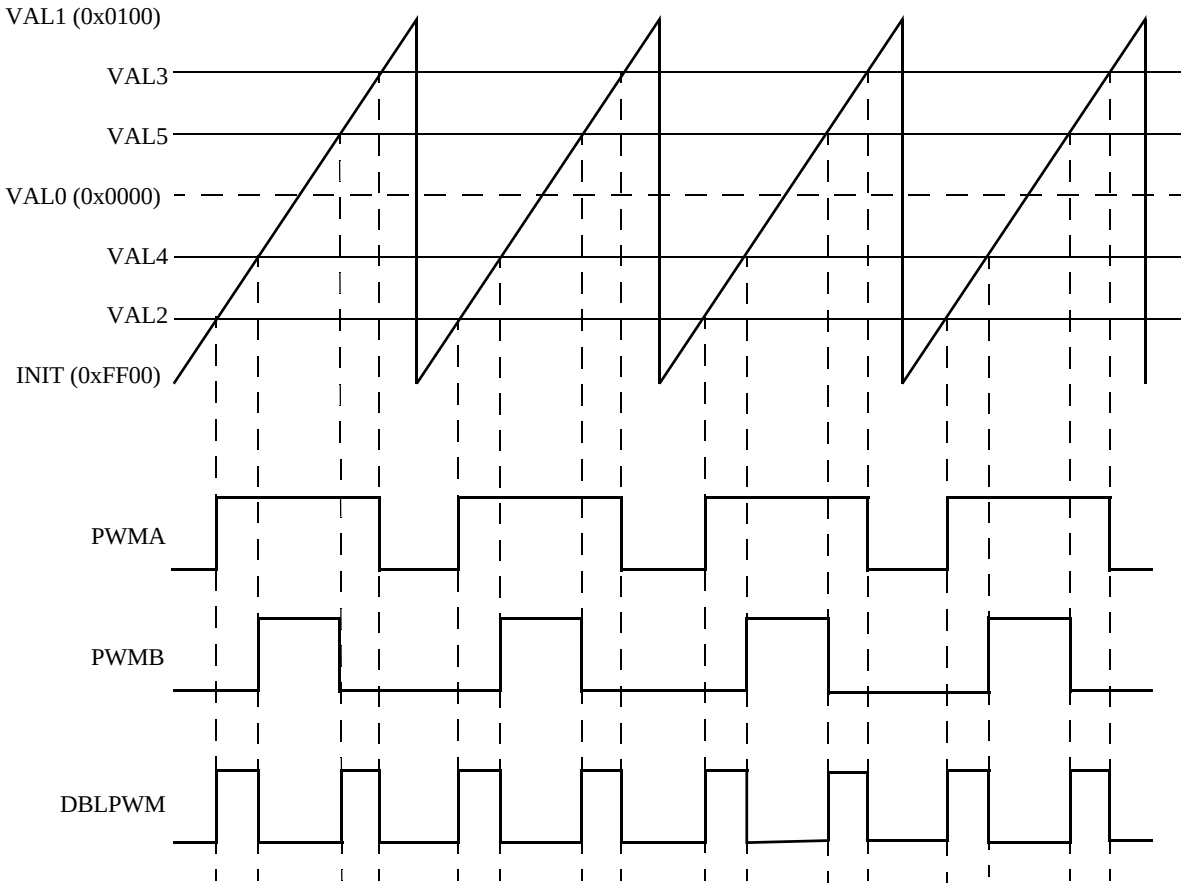


Figure 528. Double switching output example

26.7.5 ADC triggering

In cases where the timing of the ADC triggering is critical, it must be scheduled as a hardware event instead of software activated. With this PWM module, multiple ADC triggers can be generated in hardware per PWM cycle without the requirement of another timer module. *Figure 529* shows how this is accomplished. When specifying complimentary mode of operation, only two edge comparators are required to generate the output PWM signals for a given submodule. This means that the other comparators are free to perform other functions. In this example, the software does not need to respond quickly after the first conversion to set up other conversions that must occur in the same PWM cycle.

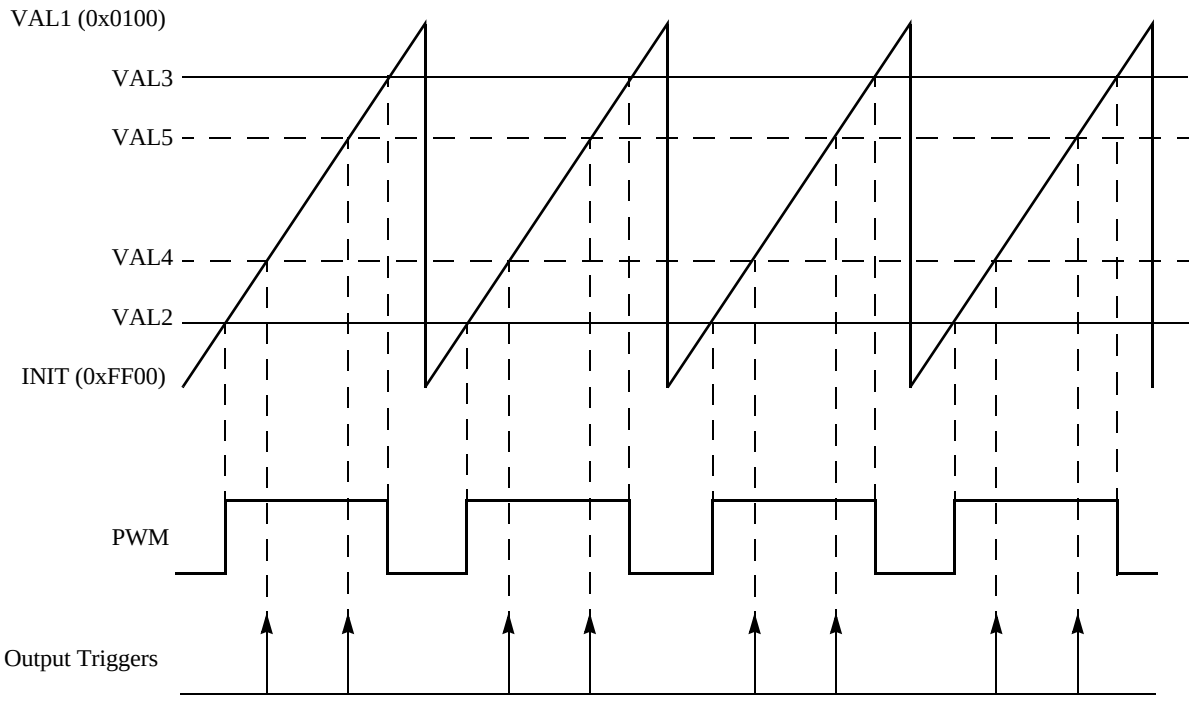


Figure 529. Multiple output trigger generation in hardware

Since each submodule has its own timer, it is possible for each submodule to run at a different frequency. One of the options possible with this PWM module is to have one or more submodules running at a lower frequency, but still synchronized to the timer in submodule 0. *Figure 530* shows how this feature can be used to schedule ADC triggers over multiple PWM cycles. A suggested use for this configuration would be to use the lower frequency submodule to control the sampling frequency of the software control algorithm where multiple ADC triggers can now be scheduled over the entire sampling period. In *Figure 530*, ALL submodule comparators are shown being used for ADC trigger generation.

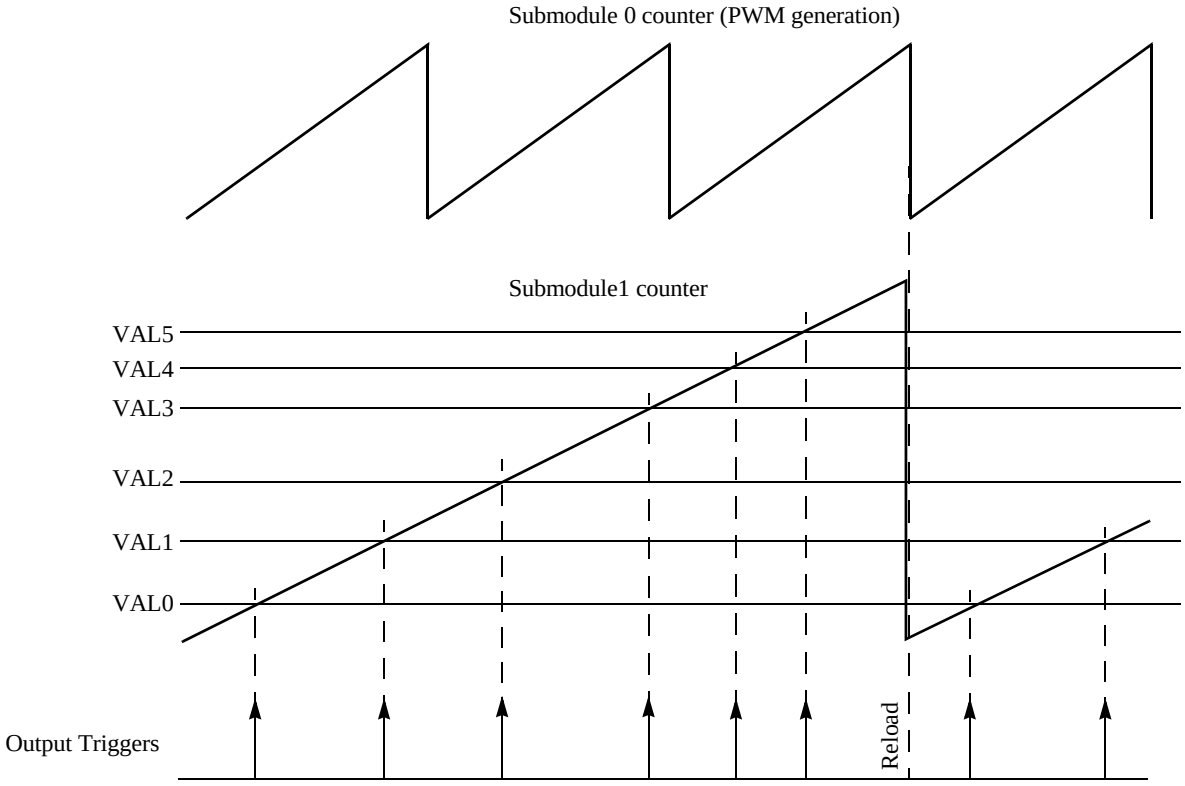


Figure 530. Multiple output triggers over several PWM cycles

26.7.6 Enhanced capture capabilities (E-Capture)

When a PWM pin is not being used for PWM generation, it can be used to perform input captures. Recall that for PWM generation BOTH edges of the PWM signal are specified via separate compare register values. When programmed for input capture, both of these registers work on the same pin to capture multiple edges, toggling from one to the other in either a free running or one-shot fashion. By programming the desired edge of each capture circuit, period and pulse width of an input signal can easily be measured without the requirement to re-arm the circuit. In addition, each edge of the input signal can clock an 8-bit counter where the counter output is compared to a user specified value (EDGCMP). When the counter output equals EDGCMP, the value of the submodule timer is captured and the counter is automatically reset. This feature allows the module to count a specified number of edge events and then perform a capture and interrupt. *Figure 531* illustrates some of the functionality of the E-Capture circuit.

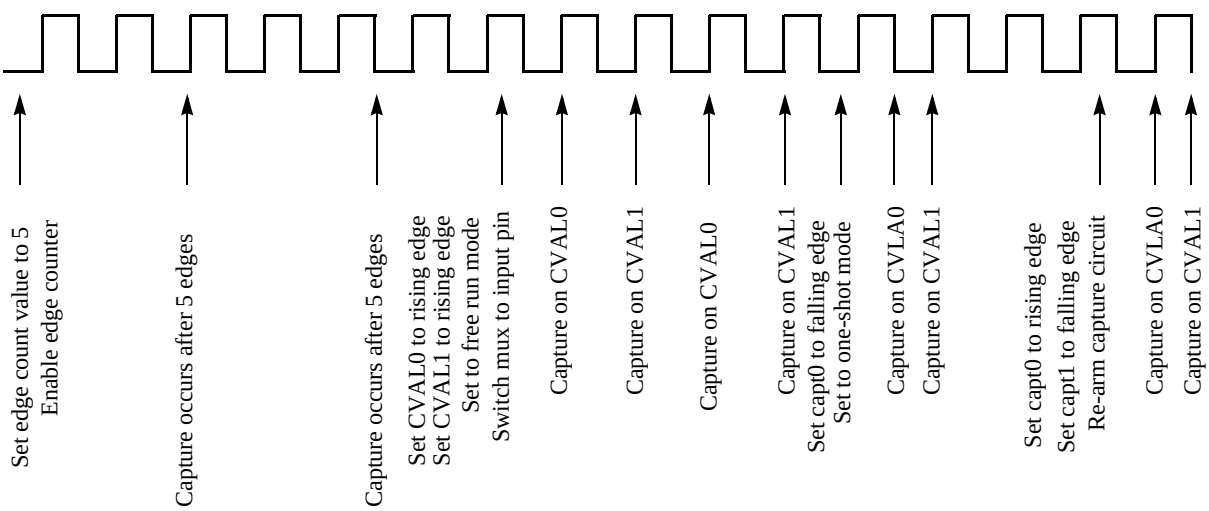


Figure 531. Capture capabilities of the E-Capture circuit

When a submodule is being used for PWM generation, its timer counts up to the modulus value used to specify the PWM frequency and then is re-initialized. Therefore, using this timer for input captures on one of the other pins (e.g, PWMX) has limited utility since it does not count through all of the numbers and the timer reset represents a discontinuity in the 16-bit number range. However, when measuring a signal that is synchronous to the PWM frequency, the timer modulus range is perfectly suited for the application. As an example, consider *Figure 532*. In this application the output of a PWM power stage is connected to the PWMX pin that is configured for free running input captures. Specifically, the CVAL0 capture circuitry is programmed for rising edges and the CVAL1 capture circuitry is set for falling edges. This will result in new load pulse width data being acquired every PWM cycle. To calculate the pulse width, subtract the CVAL0 register value from the CVAL1 register value. This measurement is extremely beneficial when performing dead-time distortion correction on a half bridge circuit driving an inductive load. Also, these values can be directly compared to the VALx registers responsible for generating the PWM outputs to obtain a measurement of system propagation delays.

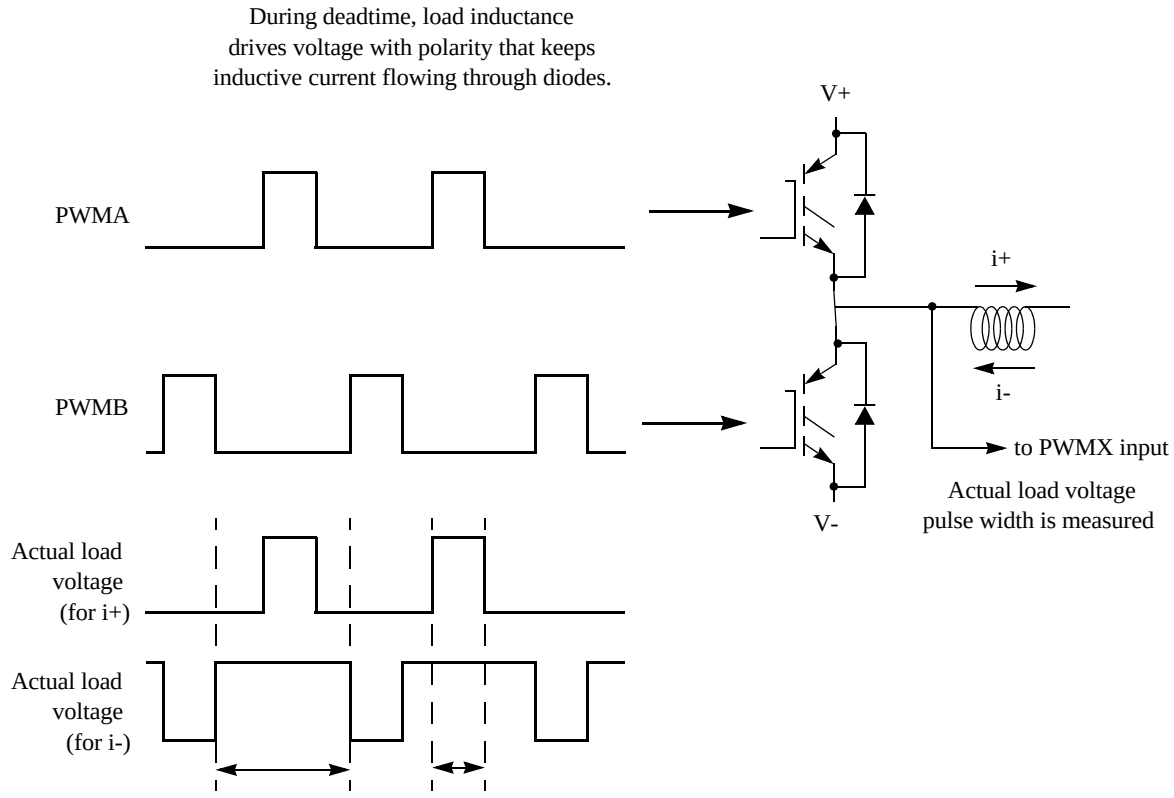


Figure 532. Output pulse width measurement possible with the E-Capture circuit

26.7.7 Synchronous switching of multiple outputs

Before the PWM signals are routed to the output pins, they are processed by a hardware block that permits all submodule outputs to be switched synchronously. This feature can be extremely useful in commutated motor applications where the next commutation state can be laid in ahead of time and then immediately switched to the outputs when the appropriate condition or time is reached. Not only do all the changes occur synchronously on all submodule outputs, but they occur IMMEDIATELY after the trigger event occurs eliminating any interrupt latency.

The synchronous output switching is accomplished via a signal called FORCE_OUT. This signal originates from the local FORCE bit within the submodule, from submodule 0, or from external to the PWM module and, in most cases, is supplied from an external timer channel configured for output compare. In a typical application, software sets up the desired states of the output pins in preparation for the next FORCE_OUT event. This selection lays dormant until the FORCE_OUT signal transitions and then all outputs are switched simultaneously. The signal switching is performed upstream from the deadtime generator so that any abrupt changes that might occur do not violate deadtime on the power stage when in complementary mode.

Figure 533 shows a popular application that can benefit from this feature. On a brushless DC motor it is desirable on many cases to spin the motor without need of hall-effect sensor feedback. Instead, the back EMF of the motor phases is monitored and this information is used to schedule the next commutation event. The top waveforms of *Figure 533* are a

simplistic representation of these back EMF signals. Timer compare events (represented by the long vertical lines in the diagram) are scheduled based on the zero crossings of the back-EMF waveforms. The PWM module is configured via software ahead of time with the next state of the PWM pins in anticipation of the compare event. When it happens, the output compare of the timer drives the FORCE_OUT signal, which immediately changes the state of the PWM pins to the next commutation state with no software latency.

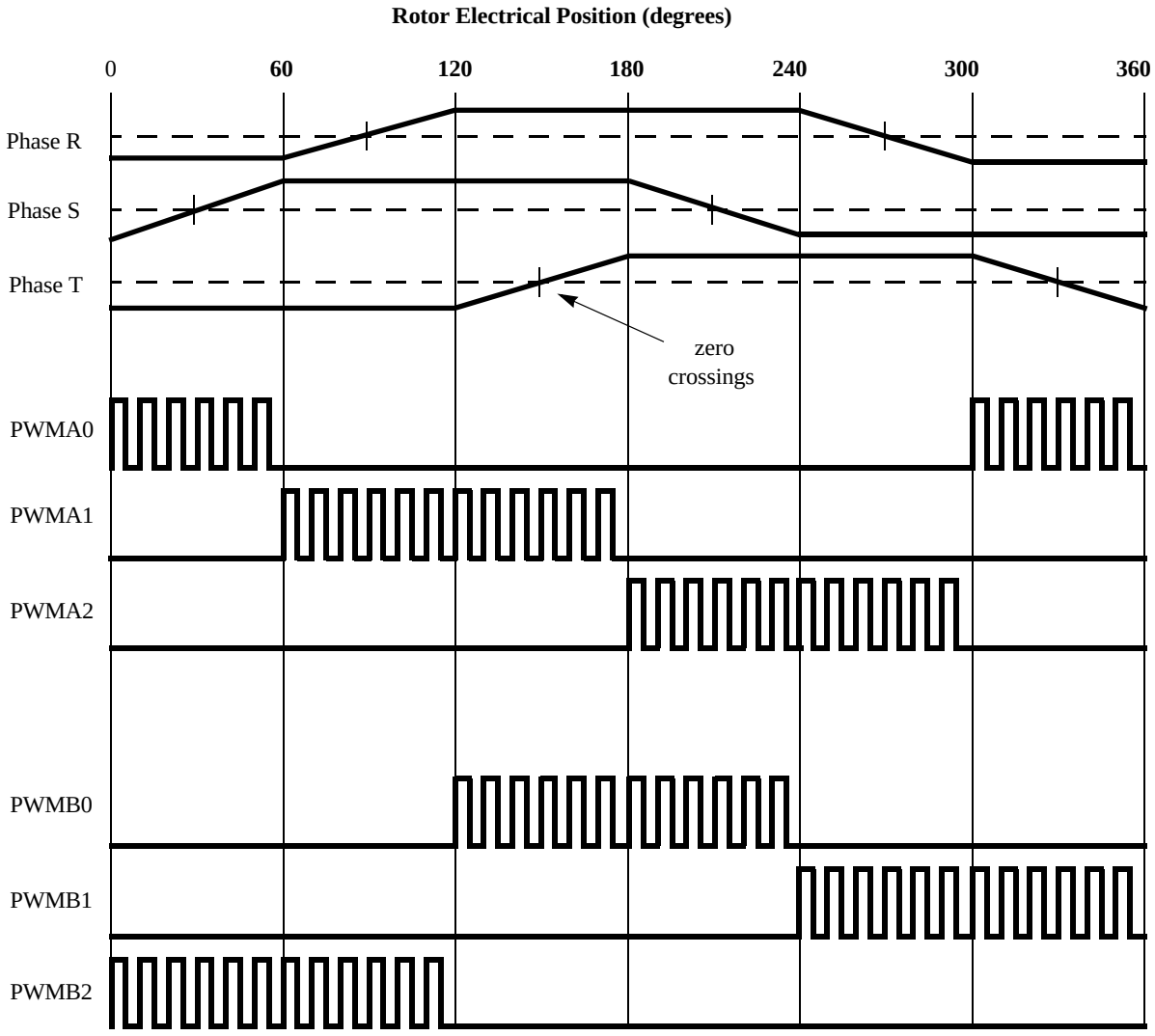


Figure 533. Sensorless BLDC commutation using the force out function

26.8 Functional details

This section describes the implementation of various features of the PWM in greater detail.

26.8.1 PWM clocking

Figure 534 shows the logic used to generate the main counter clock. Each submodule can select between three clock signals: the IPBus clock, EXT_CLK, and AUX_CLK. The EXT_CLK is generated by an on-chip resource such as a Timer module and goes to all of the submodules. The AUX_CLK signal is broadcast from submodule 0 and can be selected as the clock source by other submodules so that the 8-bit prescaler and RUN bit from submodule 0 can control all of the submodules. When AUX_CLK is selected as the source for the submodule clock, the RUN bit from submodule 0 is used instead of the local RUN bit from this submodule.

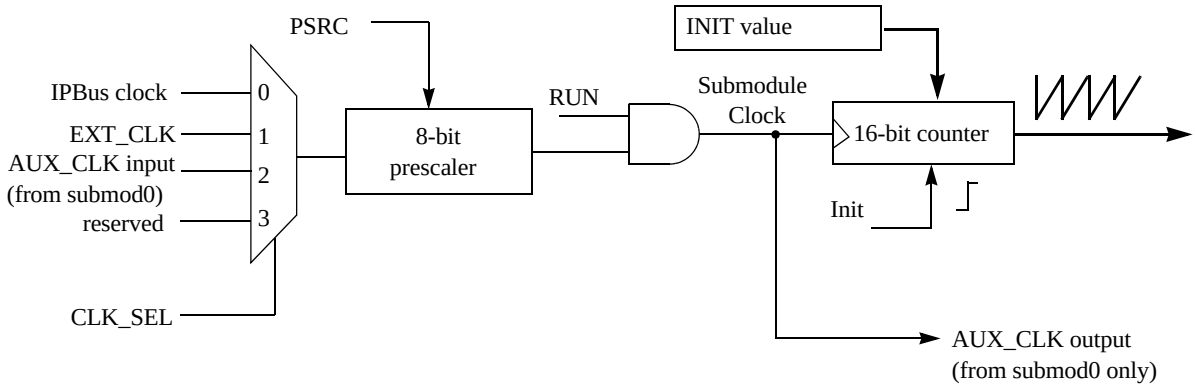


Figure 534. Clocking block diagram for each PWM submodule

To permit lower PWM frequencies, the prescaler produces the PWM clock frequency by dividing the IPBus clock frequency by 1-128. The prescaler bits, PRSC, in the control register (CTRL1), select the prescaler divisor. This prescaler is buffered and will not be used by the PWM generator until the LDOK bit is set and a new PWM reload cycle begins.

26.8.2 Register reload logic

The register reload logic determines when the outer set of registers for all double buffered register pairs will be transferred to the inner set of registers. The register reload event can be scheduled to occur every *n* PWM cycles using the LDFQ bits and the FULL bit. A half cycle reload option is also supported (HALF) where the reload can take place in the middle of a PWM cycle. The half cycle point is defined by the VAL0 register and does not have to be exactly in the middle of the PWM cycle.

As illustrated in *Figure 535* the reload signal from submodule 0 can be broadcast as the Master Reload signal allowing the reload logic from submodule 0 to control the reload of registers in other submodules.

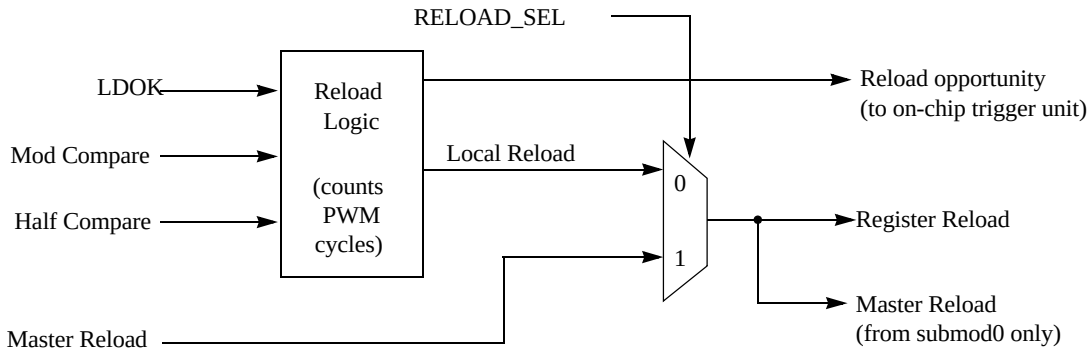


Figure 535. Register reload logic

26.8.3 Counter synchronization

Referring to *Figure 536*, the 16-bit counter will count up until its output equals VAL1, which specifies the counter modulus value. The resulting compare causes a rising edge to occur on the Local Sync signal, which is one of four possible sources used to cause the 16-bit counter to be initialized with INIT. If Local Sync is selected as the counter initialization signal, then VAL1 within the submodule effectively controls the timer period (and thus the PWM frequency generated by that submodule) and everything works on a local level.

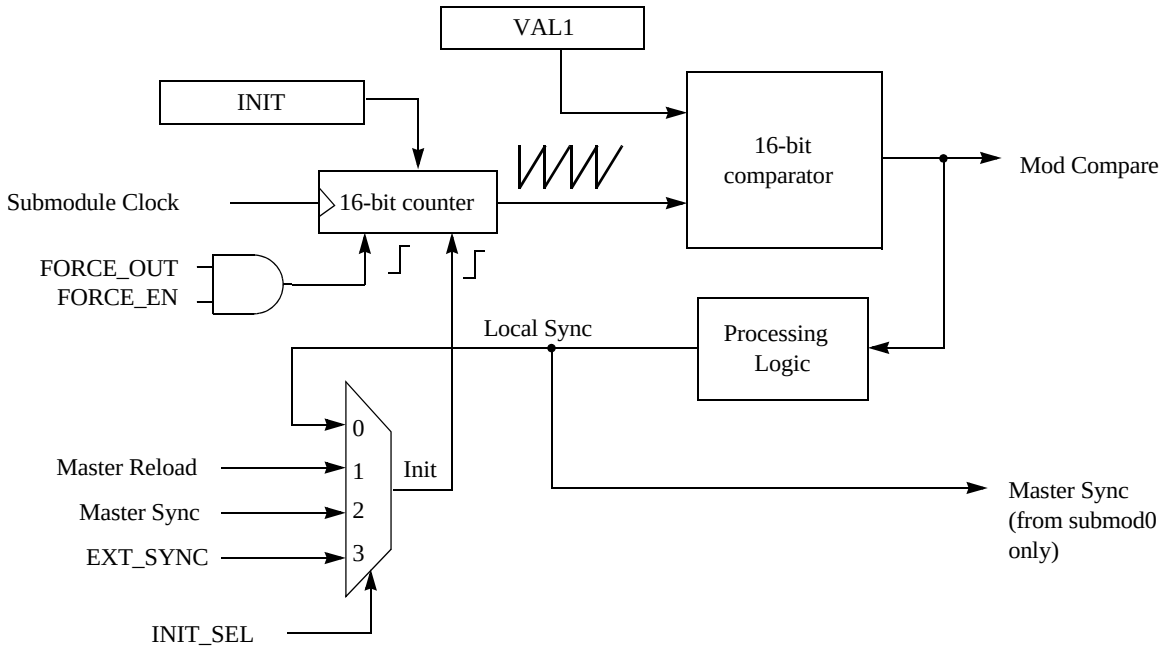


Figure 536. Submodule timer synchronization

The Master Sync signal originates as the Local Sync from submodule 0. If configured to do so, the timer period of any submodule can be locked to the period of the timer in submodule 0. The VAL1 register and associated comparator of the other submodules can then be freed

up for other functions such as PWM generation, input captures, output compares, or output triggers.

The EXT_SYNC signal originates either on- or off-chip, depending on the system architecture. This signal may be selected as the source for counter initialization so that an external source can control the period of all submodules.

If the Master Reload signal is selected as the source for counter initialization, then the period of the counter will be locked to the register reload frequency of submodule 0. Since the reload frequency is usually commensurate to the sampling frequency of the software control algorithm, the submodule counter period will therefore equal the sampling period. As a result, this timer can be used to generate output compares or output triggers over the entire sampling period, which may consist of several PWM cycles. The Master Reload signal can only originate from submodule 0.

The counter can optionally initialize upon the assertion of the FORCE_OUT signal assuming that the FORCE_EN bit is set. As indicated by *Figure 536*, this constitutes a second init input into the counter, which causes the counter to initialize regardless of which signal is selected as the counter init signal. The FORCE_OUT signal is provided mainly for commutated applications. When PWM signals are commutated on an inverter controlling a brushless DC motor, it is necessary to restart the PWM cycle at the beginning of the commutation interval. This action effectively resynchronizes the PWM waveform to the commutation timing. Otherwise, the average voltage applied to a motor winding integrated over the entire commutation interval will be a function of the timing between the asynchronous commutation event with respect to the PWM cycle. The effect is more critical at higher motor speeds where each commutation interval may consist of only a few PWM cycles. If the counter is not initialized at the start of each commutation interval, the result will be an oscillation caused by the beating between the PWM frequency and the commutation frequency.

26.8.4 PWM generation

Figure 537 illustrates how PWM generation is accomplished in each submodule. In each case, two comparators and associated VALx registers are utilized for each PWM output signal. One comparator and VAL*x* register control the turn-on edge, while a second comparator and VAL*y* register control the turn-off edge.

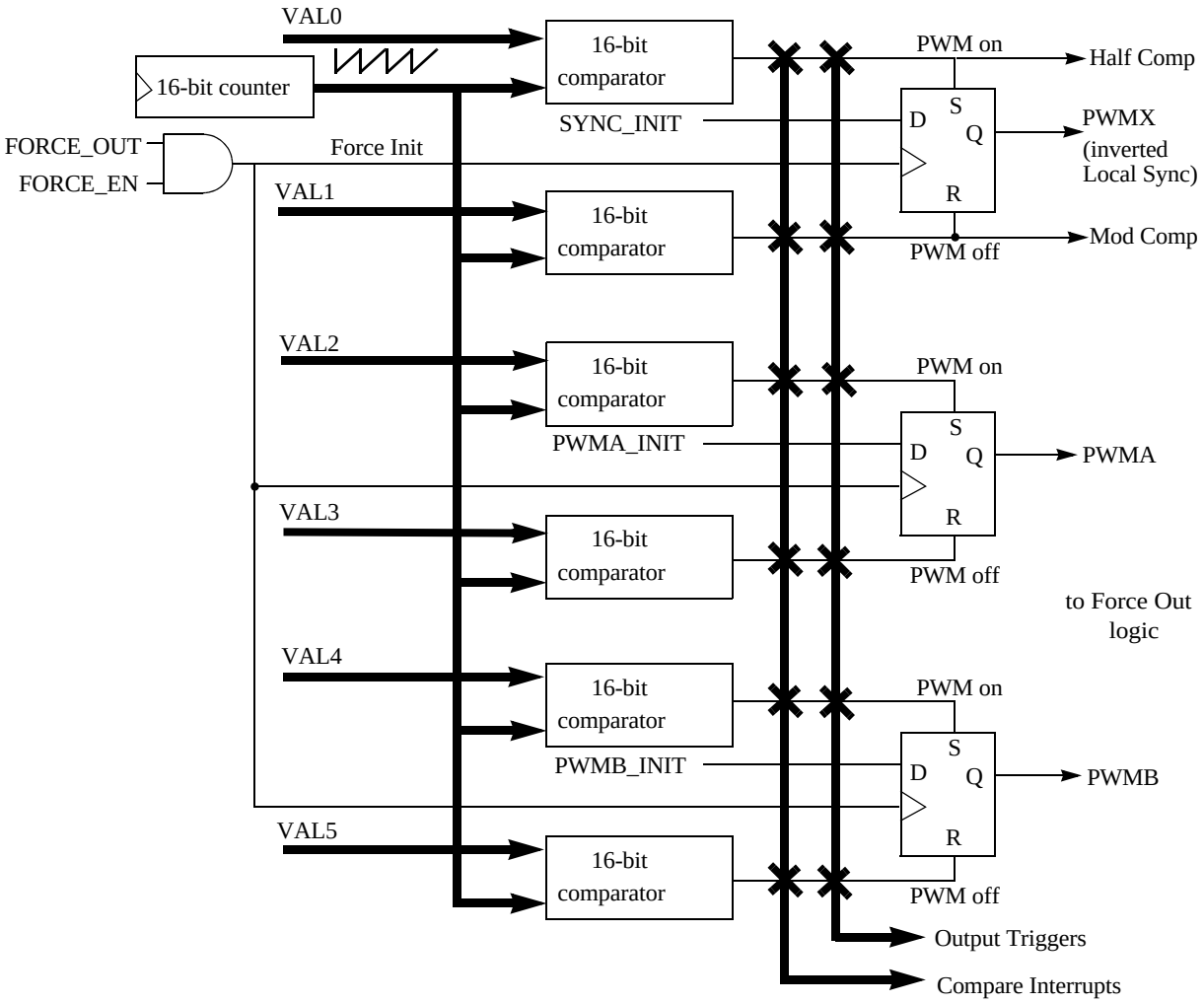


Figure 537. PWM generation hardware

The generation of the Local Sync signal is performed exactly the same way as the other PWM signals in the submodule. While comparator 0 causes a rising edge of the Local Sync signal, comparator 1 generates a falling edge. Comparator 1 is also hardwired to the reload logic to generate the half cycle reload indicator.

If VAL1 is controlling the modulus of the counter and VAL0 is half of the VAL1 register minus the INIT value, then the half cycle reload pulse will occur exactly half way through the timer count period and the Local Sync will have a 50% duty cycle. On the other hand, if the VAL1 and VAL0 registers are not required for register reloading or counter initialization, they can be used to modulate the duty cycle of the Local Sync signal effectively turning it into an auxiliary PWM signal (PWMX) assuming that the PWMX pin is not being used for another function such as input capture or deadtime distortion correction. Including the Local Sync signal, each submodule is capable of generating three PWM signals where software has complete control over each edge of each of the signals.

If the comparators and edge value registers are not required for PWM generation, they can also be used for other functions such as output compares, generating output triggers, or generating interrupts at timed intervals.

The 16-bit comparators shown in *Figure 537* are “equal to or greater than” not just “equal to” comparators. In addition, if both the set and reset of the flip-flop are both asserted, then the flop output goes to 0.

26.8.5 Output compare capabilities

By using the VALx registers in conjunction with the submodule timer and 16-bit comparators, buffered output compare functionality can be achieved with no additional hardware required. Specifically, the following output compare functions are possible:

- An output compare sets the output high
- An output compare sets the output low
- An output compare generates an interrupt
- An output compare generates an output trigger

Referring again to *Figure 537*, an output compare is initiated by programming a VALx register for a timer compare, which in turn causes the output of the D flip-flop to either set or reset. For example, if an output compare is desired on the PWMA signal that sets it high, VAL2 would be programmed with the counter value where the output compare should take place. However, to prevent the D flip-flop from being reset again after the compare has occurred, the VAL3 register must be programmed to a value outside of the modulus range of the counter. Therefore, a compare that would result in resetting the D flip-flop output would never occur. Conversely, if an output compare is desired on the PWMA signal that sets it low, the VAL3 register is programmed with the appropriate count value and the VAL2 register is programmed with a value outside the counter modulus range. Regardless of whether a high compare or low compare is programmed, an interrupt or output trigger can be generated when the compare event occurs.

26.8.6 Force out logic

For each submodule software can select between seven signal sources for the FORCE_OUT signal: the local FORCE bit, the Master Force signal from submodule 0, the local Reload signal, the Master Reload signal from submodule 0, the Local Sync signal, the Master Sync signal from submodule 0, or the EXT_FORCE signal from on or off chip depending on the device architecture. The local signals are used when the user wants to change the signals on the output pins of the submodule without regard for synchronization with other submodules. However, if it is required that all signals on all submodule outputs change at the same time, the Master signals or EXT_FORCE signal should be selected.

Figure 538 illustrates the Force logic. The SELA and SELB fields each choose from one of four signals that can be supplied to the submodule outputs: the PWM signal, the inverted PWM signal, a binary level specified by software via the OUTA and OUTB bits. The selection can be determined ahead of time and, when a FORCE_OUT event occurs, these values are presented to the signal selection mux, which immediately switches the requested signal to the output of the mux for further processing downstream.

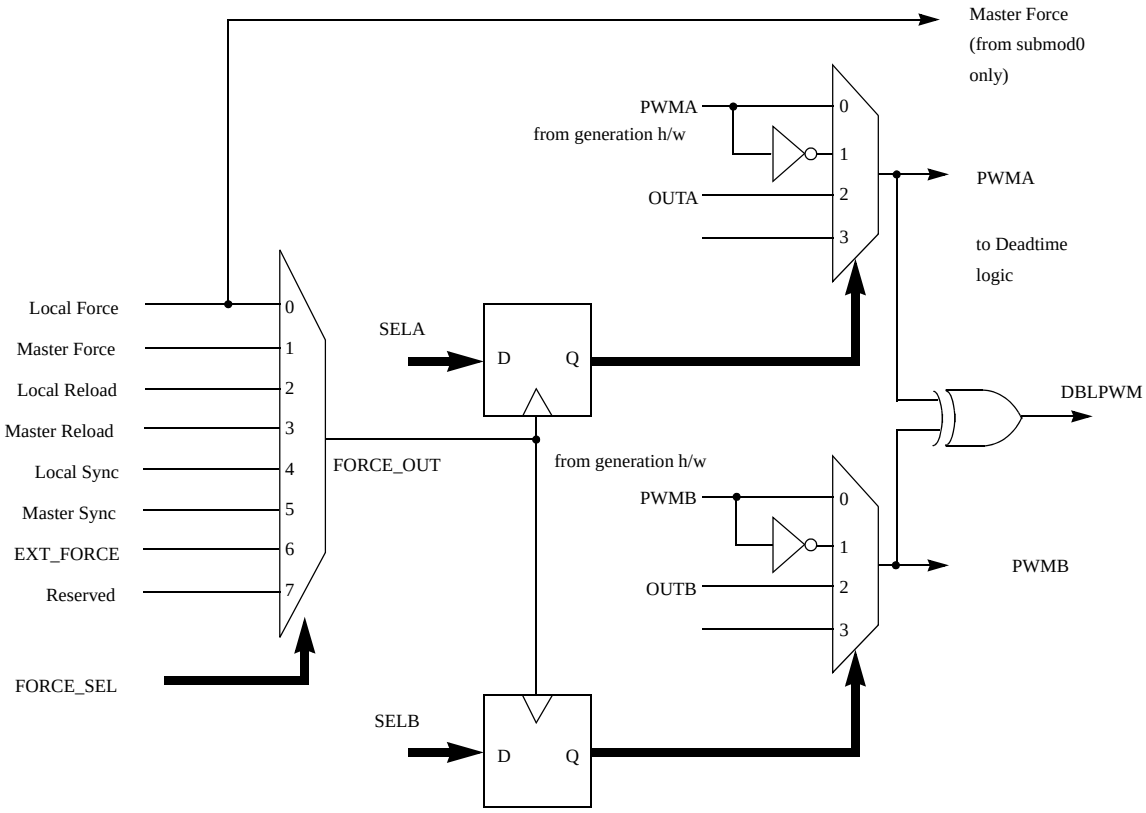


Figure 538. Force out logic

The local Force signal of submodule 0 can be broadcast as the Master Force signal to other submodules. This feature allows the local FORCE bit of submodule 0 to synchronously update all of the submodule outputs at the same time. The EXT_FORCE signal originates from outside the PWM module from a source such as a timer or digital comparators in the analog-to-digital converter.

26.8.7 Independent or complementary channel operation

Writing a logic one to the INDEP bit of the CNFG register configures the pair of PWM outputs as two independent PWM channels. Each PWM output is controlled by its own VALx pair operating independently of the other output.

Writing a logic zero to the INDEP bit configures the PWM output as a pair of complementary channels. The PWM pins are paired as shown in *Figure 539* in complementary channel operation. The IPOL bit determines which signal is connected to the output pin (PWMA or PWMB).

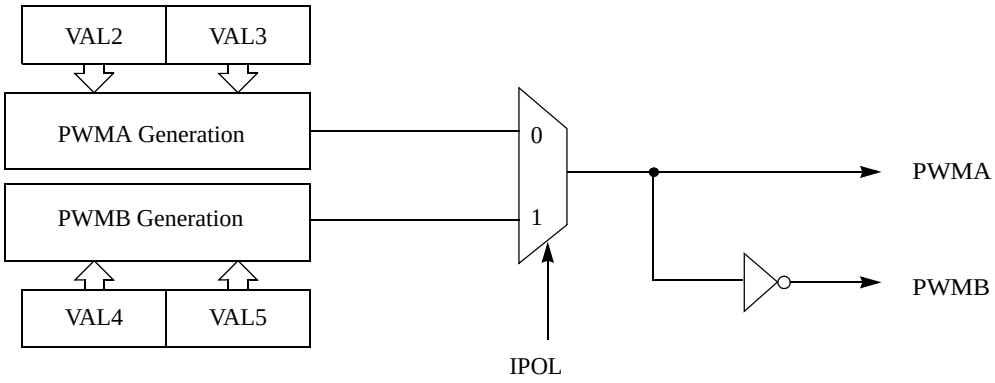


Figure 539. Complementary channel pair

The complementary channel operation is for driving top and bottom transistors in a motor drive circuit, such as the one in *Figure 540*.

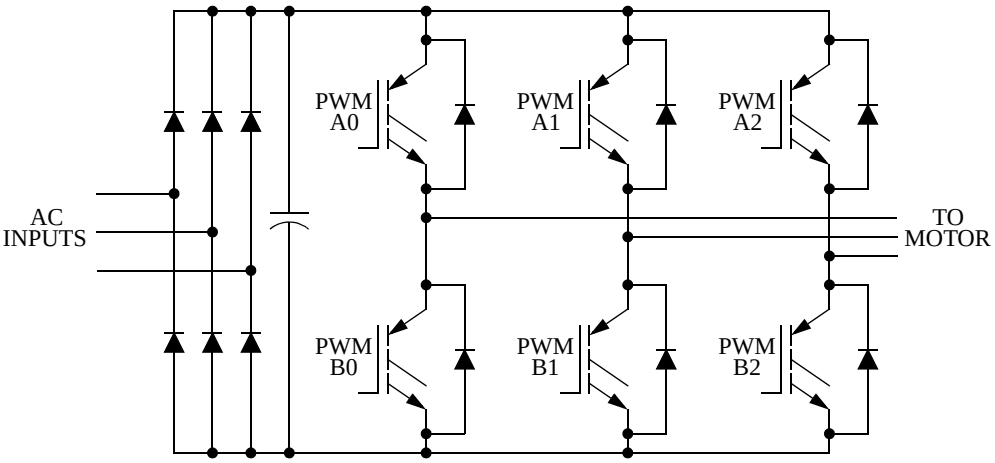


Figure 540. Typical 3-phase AC motor drive

Complementary operation allows the use of the deadtime insertion feature.

26.8.8 Deadtime insertion logic

Figure 541 shows the deadtime insertion logic of each submodule, which creates non-overlapping complementary signals when not in independent mode.

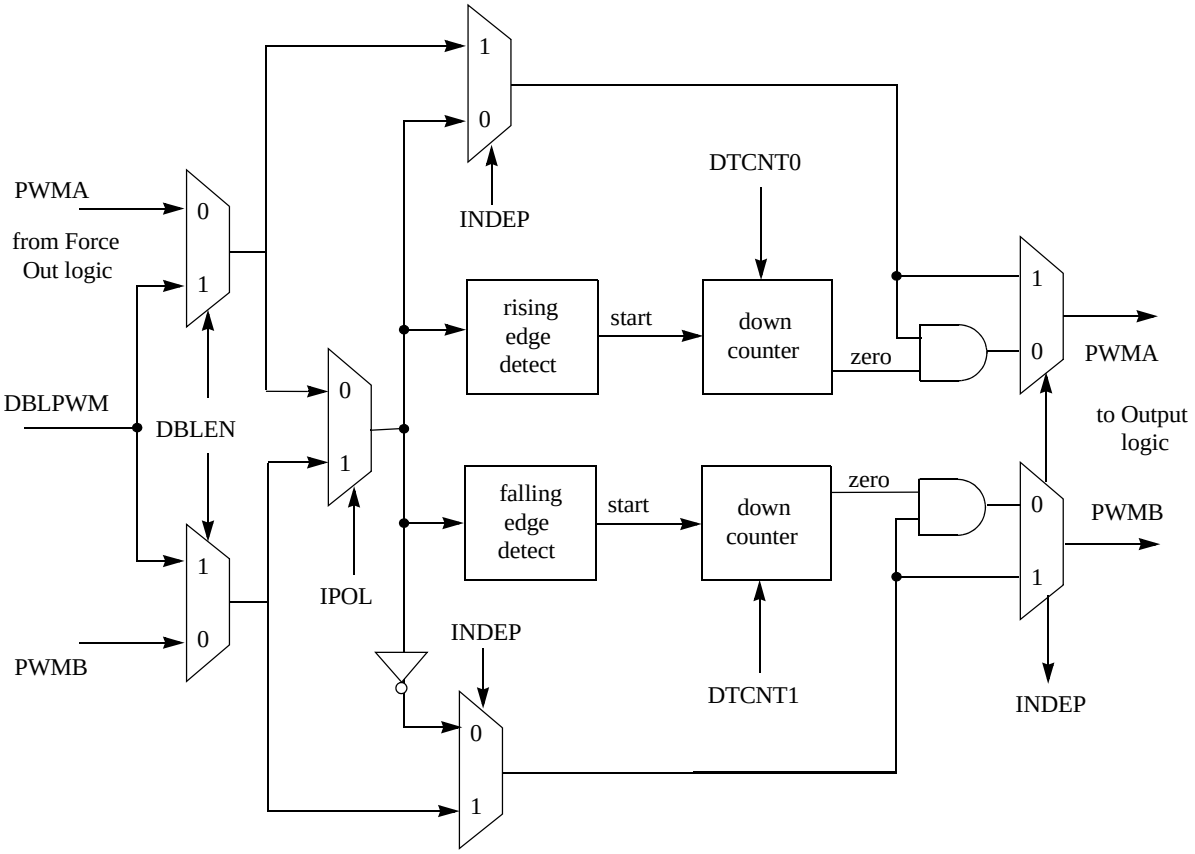


Figure 541. Deadtime insertion and fine control logic

While in the complementary mode, a PWM pair can be used to drive top/bottom transistors, as shown in *Figure 541*. When the top PWM channel is active, the bottom PWM channel is inactive, and vice versa.

Note: To avoid short-circuiting the DC bus and endangering the transistor, there must be no overlap of conducting intervals between top and bottom transistor. However, the transistor's characteristics may cause its switching-off time to be longer than its switching-on time. To avoid the conducting overlap of top and bottom transistors, deadtime needs to be inserted in the switching period, as illustrated in Figure 542.

The deadtime generators automatically insert software-selectable activation delays into the pair of PWM outputs. The deadtime registers (DTCNT0 and DTCNT1) specify the number of IPBus clock cycles to use for deadtime delay. Every time the deadtime generator inputs change state, deadtime is inserted. Deadtime forces both PWM outputs in the pair to the inactive state.

When deadtime is inserted in complementary PWM signals connected to an inverter driving an inductive load, the PWM waveform on the inverter output will have a different duty cycle than what appears on the output pins of the PWM module. This results in a distortion in the voltage applied to the load. A method of correcting this, adding to or subtracting from the PWM value used, is discussed next.

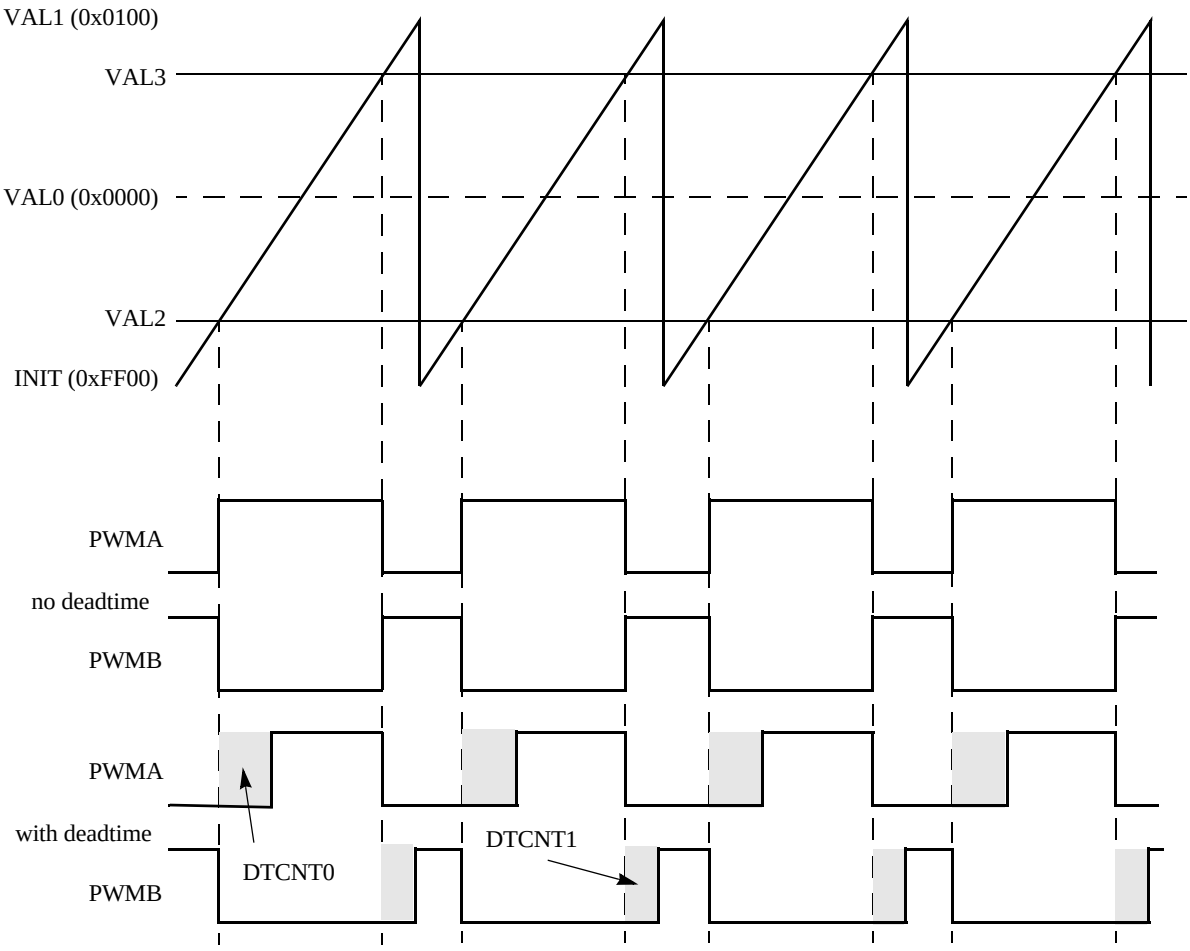


Figure 542. Deadtime insertion

26.8.9 Top/bottom correction

In complementary mode, either the top or the bottom transistor controls the output voltage. However, deadtime has to be inserted to avoid overlap of conducting interval between the top and bottom transistor. Both transistors in complementary mode are off during deadtime, allowing the output voltage to be determined by the current status of load and introduce distortion in the output voltage. See *Figure 543*. On AC induction motors running open-loop, the distortion typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

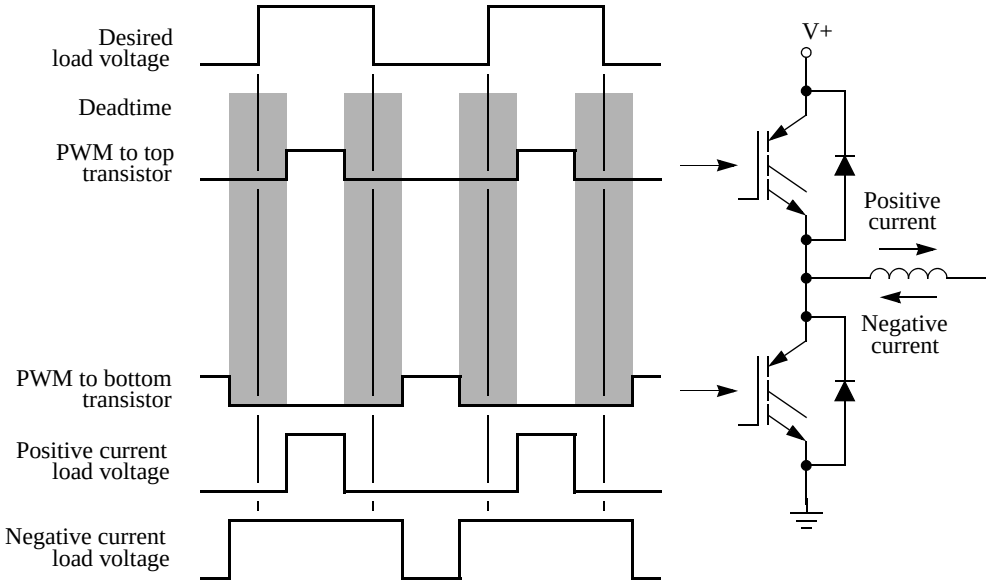


Figure 543. Deadtime distortion

During deadtime, load inductance distorts output voltage by keeping current flowing through the diodes. This deadtime current flow creates a load voltage that varies with current direction. With a positive current flow, the load voltage during deadtime is equal to the bottom supply, putting the top transistor in control. With a negative current flow, the load voltage during deadtime is equal to the top supply putting the bottom transistor in control.

Remembering that the original PWM pulse widths where shortened by deadtime insertion, the averaged sinusoidal output will be less than the desired value. However, when deadtime is inserted, it creates a distortion in the motor current waveform. This distortion is aggravated by dissimilar turn-on and turn-off delays of each of the transistors. By giving the PWM module information on which transistor is controlling at a given time this distortion can be corrected.

For a typical circuit in complementary channel operation, only one of the transistors will be effective in controlling the output voltage at any given time. This depends on the direction of the motor current for that pair. See *Figure 544*. To correct distortion one of two different factors must be added to the desired PWM value, depending on whether the top or bottom transistor is controlling the output voltage. Therefore, the software is responsible for calculating both compensated PWM values prior to placing them in the VALx registers. Either the VAL2/VAL3 or the VAL4/VAL5 register pair controls the pulse width at any given time. For a given PWM pair, whether the VAL2/VAL3 or VAL4/VAL5 pair is active depends on either:

- The state of the current status pin, PWMx, for that driver
- The state of the odd/even correction bit, IPOL, for that driver

To correct deadtime distortion, software can decrease or increase the value in the appropriate VALx register.

- In edge-aligned operation, decreasing or increasing the PWM value by a correction value equal to the deadtime typically compensates for deadtime distortion.
- In center-aligned operation, decreasing or increasing the PWM value by a correction value equal to one-half the deadtime typically compensates for deadtime distortion.

26.8.10 Manual correction

To detect the current status, the voltage on each PWMx pin is sampled twice in a PWM period, at the end of each deadtime. The value is stored in the DTx bits in the CTRL1 register. The DTx bits are a timing marker especially indicating when to toggle between PWM value registers. Software can then set the IPOL bit to switch between VAL2/VAL3 and VAL4/VAL5 register pairs according to DTx values.

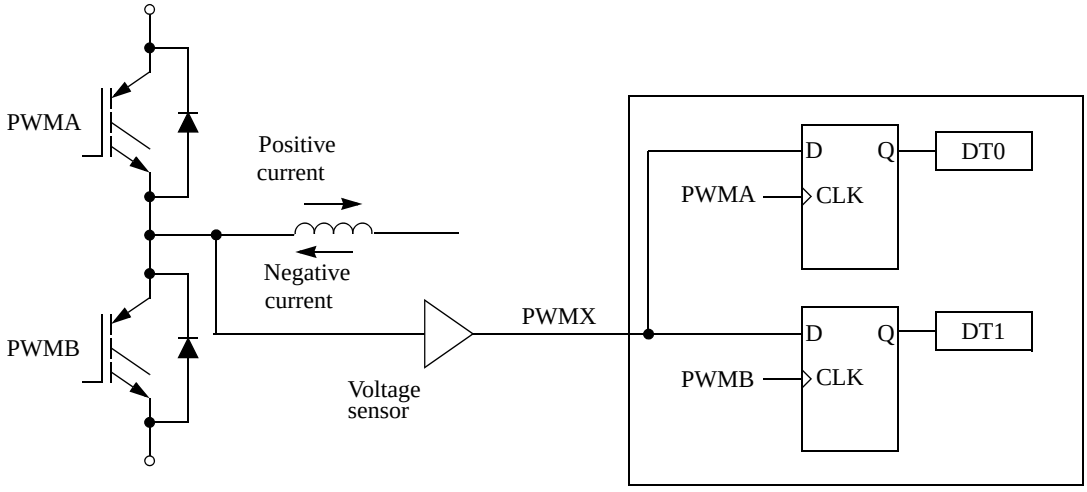


Figure 544. Current-status sense scheme for deadtime correction

Both D flip-flops latch low, DT0 = 0, DT1 = 0, during deadtime periods if current is large and flowing out of the complementary circuit. See *Figure 544*. Both D flip-flops latch the high, DT0 = 1, DT1 = 1, during deadtime periods if current is also large and flowing into the complementary circuit.

However, under low-current, the output voltage of the complementary circuit during deadtime is somewhere between the high and low levels. The current cannot free-wheel through the opposition anti-body diode, regardless of polarity, giving additional distortion when the current crosses zero. Sampled results will be DT0 = 0 and DT1 = 1. Thus, the best time to change one PWM value register to another is just before the current zero crossing.

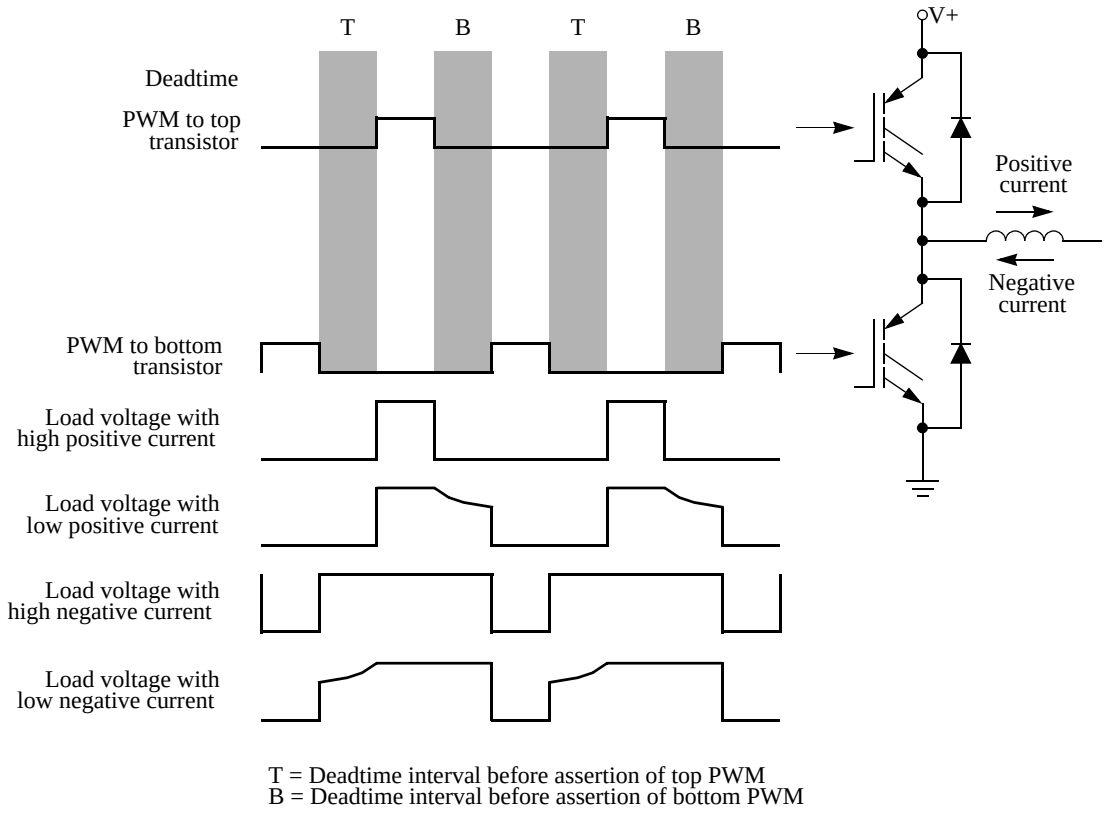


Figure 545. Output voltage waveforms

26.8.11 Output logic

Figure 546 shows the output logic of each submodule including how each PWM output has individual fault disabling, polarity control, and output enable. This allows for maximum flexibility when interfacing to the external circuitry.

The PWMA and PWMB signals that are output from the deadtime logic in *Figure 546* are positive true signals. In other words, a high level on these signals should result in the corresponding transistor in the PWM inverter being turned ON. The voltage level required at the PWM output pin to turn the transistor ON or OFF is a function of the logic between the pin and the transistor. Therefore, it is imperative that the user program the POLA and POLB bits before enabling the output pins. A fault condition can result in the PWM output being tristated, forced to a logic 1, or forced to a logic 0 depending on the values programmed into the PWMxFS fields.

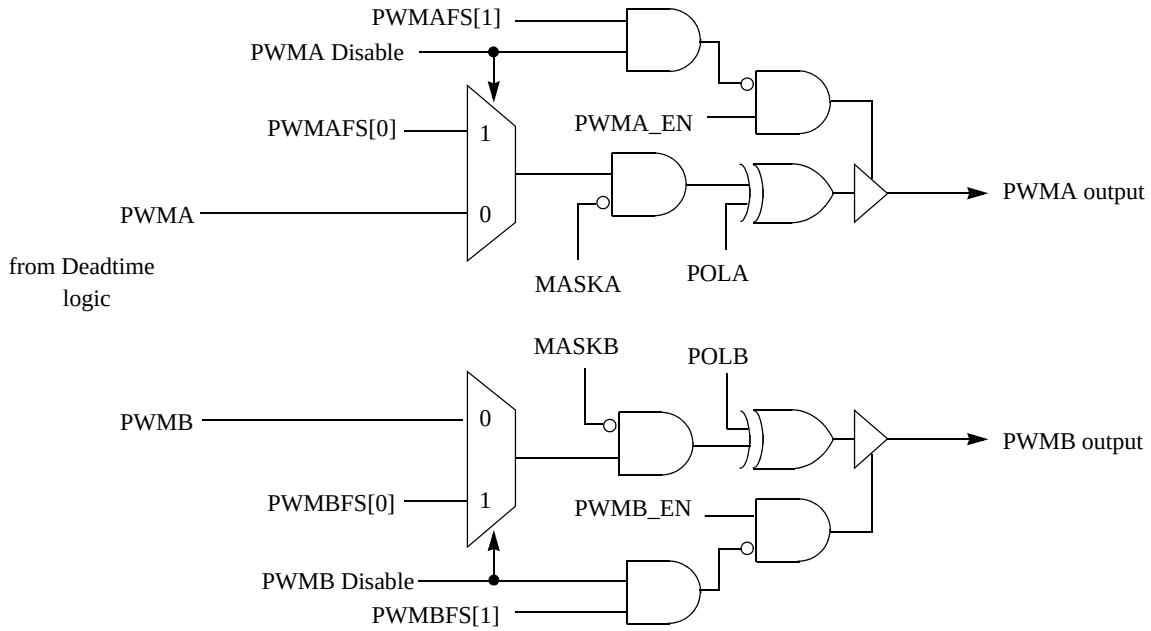


Figure 546. Output logic section

26.8.12 E-Capture

Commensurate with the idea of controlling both edges of an output signal, the Enhanced Capture (E-Capture) logic is designed to measure both edges of an input signal. As a result, when a submodule pin is configured for input capture, the CVALx registers associated with that pin record the edge values.

Figure 547 illustrates the block diagram of the E-Capture circuit. Upon entering the pin input, the signal is split into two paths. One goes straight to a mux input where software can select to pass the signal directly to the capture logic for processing. The other path connects the signal to an 8-bit counter that counts both the rising and falling edges of the signal. The output of this counter is compared to an 8-bit value that is specified by the user (EDGCMP*x*) and when the two values are equal, the comparator generates a pulse that resets the counter. This pulse is also supplied to the mux input where software can select it to be processed by the capture logic. This feature permits the E-Capture circuit to count as many as 256 edge events before initiating a capture event. This feature is useful for dividing down high frequency signals for capture processing so that capture interrupts do not overwhelm the CPU. Also, this feature can be used to generate an interrupt after *n* events have been counted.

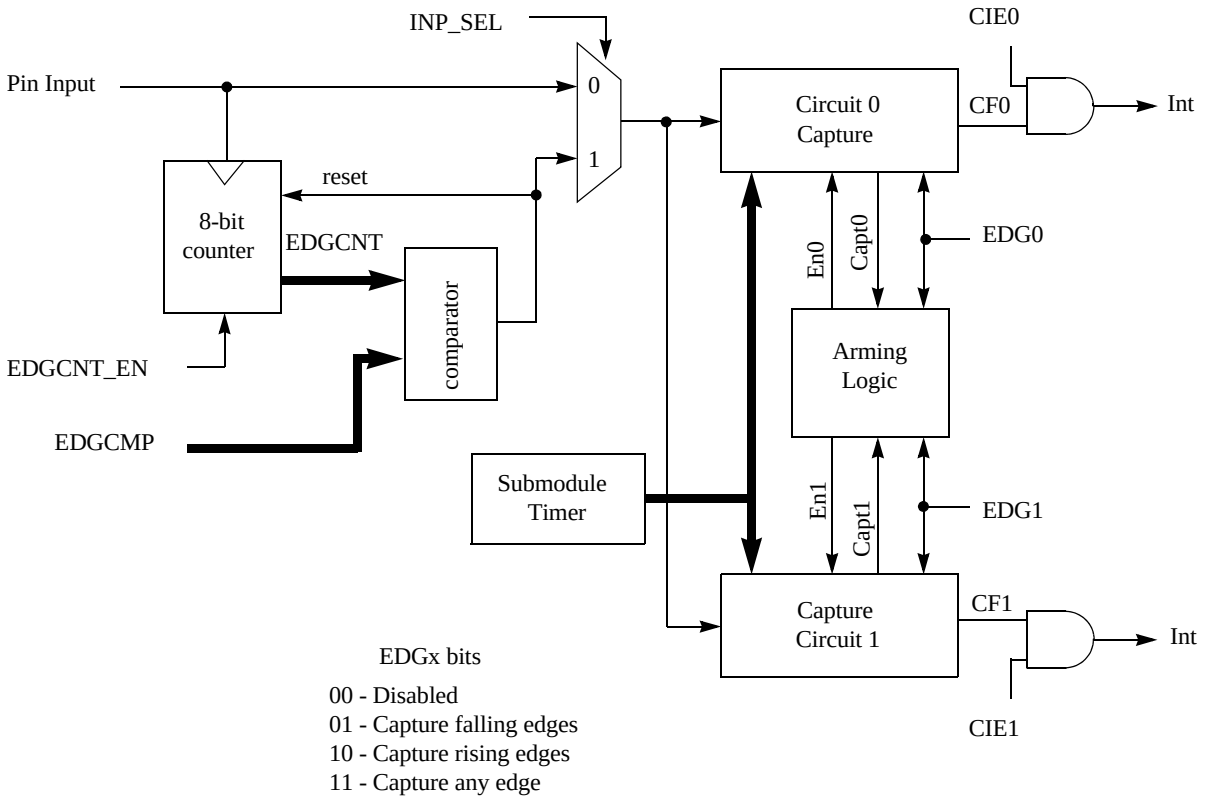


Figure 547. Enhanced Capture (E-Capture) logic

Based on the mode selection, the mux selects either the pin input or the compare output from the count/compare circuit to be processed by the capture logic. The selected signal is routed to two separate capture circuits that work in tandem to capture sequential edges of the signal. The type of edge to be captured by each circuit is determined by the EDGx1 and EDGx0 bits whose functionality is listed in *Figure 547*. Also, controlling the operation of the capture circuits is the arming logic, which allows captures to be performed in a free running (continuous) or one shot fashion. In free running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one shot mode, only one capture sequence will be performed. If both capture circuits are enabled, capture circuit 0 is first armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt to the CPU.

26.8.13 Fault protection

Fault protection can control any combination of PWM output pins. Faults are generated by a logic one on any of the FAULTx pins. This polarity can be changed via the FLVL bits. Each FAULTx pin can be mapped arbitrarily to any of the PWM outputs. When fault protection hardware disables PWM outputs, the PWM generator continues to run, only the output pins are forced to logic 0, logic 1, or tristated depending the values of the PWMxFS bits.

The fault decoder disables PWM pins selected by the fault logic and the disable mapping register (DISMAP). See *Figure 548* for an example of the fault disable logic. Each bank of bits in DISMAP control the mapping for a single PWM pin. Refer to *Table 496*.

The fault protection is enabled even when the PWM module is not enabled; therefore, a fault will be latched in and must be cleared in order to prevent an interrupt when the PWM is enabled.

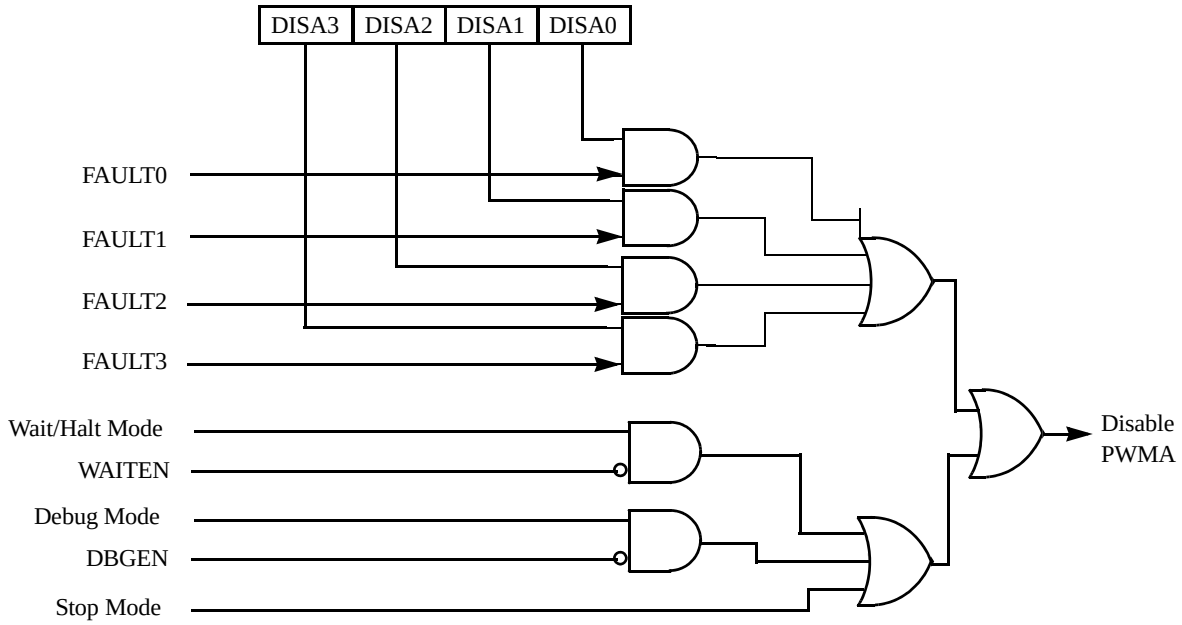


Figure 548. Fault decoder for PWMA

Table 496. Fault mapping

PWM pin	Controlling register bits
PWMA	DISA[3:0]
PWMB	DISB[3:0]
PWMX	DISX[3:0]

26.8.14 Fault pin filter

Each fault pin has a programmable filter that can be bypassed. The sampling period of the filter can be adjusted with the FILT_PER field of the FFILTx register. The number of consecutive samples that must agree before an input transition is recognized can be adjusted using the FILT_CNT field of the same register. Setting FILT_PER to all 0 disables the input filter for a given FAULTx pin.

Upon detecting a logic 0 on the filtered FAULTx pin (or a logic 1 if FLVLx is set), the corresponding FFPINx and fault flag, FFLAGx, bits are set. The FFPINx bit remains set as long as the filtered FAULTx pin is zero. Clear FFLAGx by writing a logic 1 to FFLAGx.

If the FIEx, FAULTx pin interrupt enable bit is set, the FFLAGx flag generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears the FFLAGx flag by writing a logic one to the bit
- Software clears the FIEx bit by writing a logic zero to it
- A reset occurs

Even with the filter enabled, there is a combinational path from the FAULTx inputs to the PWM pins. This logic is also capable of holding a fault condition in the event of loss of clock to the PWM module.

26.8.15 Automatic fault clearing

Setting an automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for automatic clearing.

When FAUTOx is set, disabled PWM pins are enabled when the FAULTx pin returns to logic one and a new PWM either full or half cycle begins. See *Figure 549*. Clearing the FFLAGx flag does not affect disabled PWM pins when FAUTOx is set.

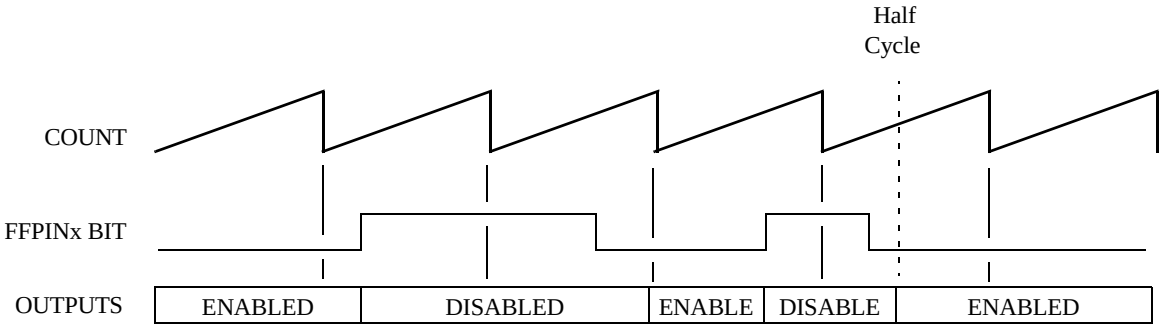


Figure 549. Automatic fault clearing

26.8.16 Manual fault clearing

Clearing the automatic clearing mode bit, FAUTOx, configures faults from the FAULTx pin for manual clearing:

- If the fault safety mode bits, FSAFEx, are clear, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The pins are enabled when the next PWM half cycle begins regardless of the logic level detected by the filter at the FAULTx pin. See *Figure 550*.
- If the fault safety mode bits, FSAFEx, are set, then PWM pins disabled by the FAULTx pins are enabled when:
  - Software clears the corresponding FFLAGx flag
  - The filter detects a logic one on the FAULTx pin at the start of the next PWM half cycle boundary. See *Figure 551*.

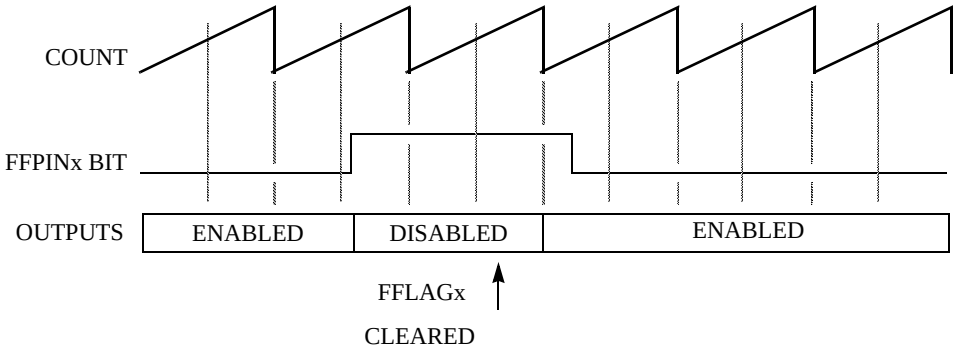


Figure 550. Manual fault clearing (FSAFE = 0)

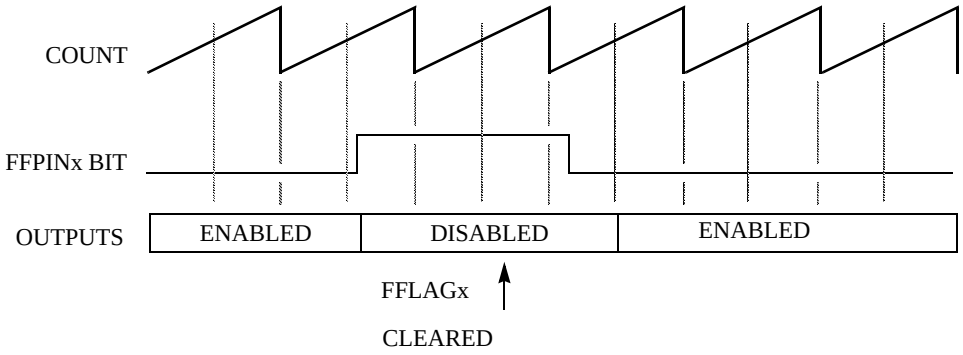


Figure 551. Manual fault clearing (FSAFE = 1)

Note: Fault protection also applies during software output control when the SELA and SELB fields are set to select OUTA and OUTB bits. Fault clearing still occurs at half PWM cycle boundaries while the PWM generator is engaged, RUN equals one. But the OUTx bits can control the PWM pins while the PWM generator is off, RUN equals zero. Thus, fault clearing occurs at IPBus cycles while the PWM generator is off and at the start of PWM cycles when the generator is engaged.

26.8.17 Fault testing

The FTEST bit simulates a fault condition on each of the fault inputs.

26.9 PWM generator loading

26.9.1 Load enable

The LDOK bit enables loading of the following PWM generator parameters:

- The prescaler divisor—from the PRSC bits in the CTRL1 register
- The PWM period and pulse width—from the INIT and VALx registers

LDOK allows software to finish calculating all of these PWM parameters so they can be synchronously updated. The PSRC, INIT, and VALx registers are loaded by software into a set of outer buffers. When LDOK is set, these values are transferred to an inner set of registers at the beginning of the next PWM reload cycle to be used by the PWM generator. Set LDOK by reading it when it is a logic zero and then writing a logic one to it. After loading, LDOK is automatically cleared.

26.9.2 Load frequency

The LDFQ bits in the CTRL1 register select an integral loading frequency of one to 16 PWM reload opportunities. The LDFQ bits take effect at every PWM reload opportunity, regardless the state of the LDOK bit. The HALF and FULL bits in the CTRL1 register control reload timing. If FULL is set, a reload opportunity occurs at the end of every PWM cycle when the count equals VAL1. If HALF is set, a reload opportunity occurs at the half cycle when the count equals VAL0. If both HALF and FULL are set, a reload opportunity occurs twice per PWM cycle when the count equals VAL1 and when it equals VAL0.

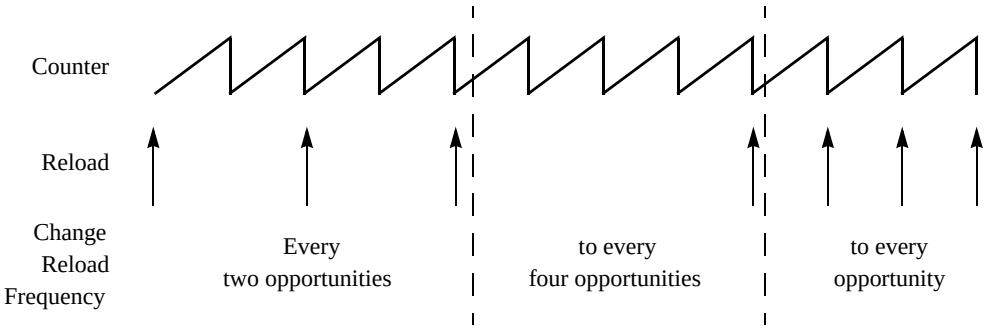


Figure 552. Full cycle reload frequency change

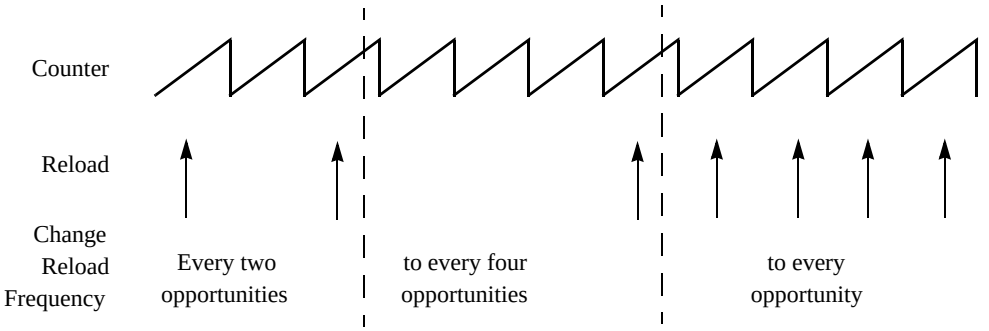


Figure 553. Half cycle reload frequency change

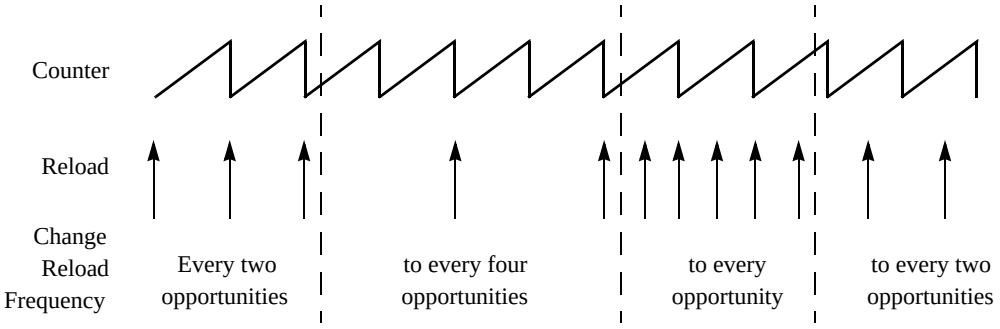


Figure 554. Full and half cycle reload frequency change

26.9.3 Reload flag

At every reload opportunity the PWM Reload Flag (RF) in the CTRL1 register is set. Setting RF happens even if an actual reload is prevented by the LDOK bit. If the PWM reload interrupt enable bit, RIE is set, the RF flag generates CPU interrupt requests allowing software to calculate new PWM parameters in real time. When RIE is not set, reloads still occur at the selected reload rate without generating CPU interrupt requests.

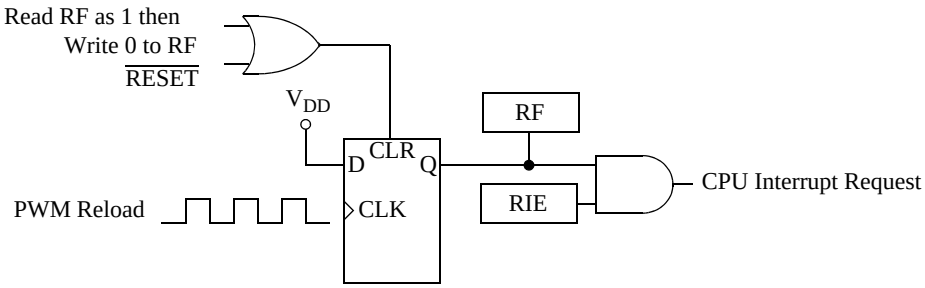


Figure 555. PWMF reload interrupt request

26.9.4 Reload errors

Whenever one of the VALx, or PSRC registers is updated, the RUF flag is set to indicate that the data is not coherent. RUF will be cleared by a successful reload, which consists of the reload signal while LDOK is set. If RUF is set and LDOK is clear when the reload signal occurs, a reload error has taken place and the REF bit is set. If RUF is clear when a reload signal asserts, then the data is coherent and no error will be flagged.

26.9.5 Initialization

Initialize all registers and set the LDOK bit before setting the RUN bit.

Note: Even if LDOK is not set, setting RUN also sets the RF flag. To prevent a CPU interrupt request, clear the RIE bit before setting RUN.

The PWM generator uses the last values loaded if RUN is cleared and then set while LDOK equals zero.

When the RUN bit is cleared:

- The RF flag and pending CPU interrupt requests are not cleared
- All fault circuitry remains active
- Software/external output control remains active
- Deadtime insertion continues during software/external output control

26.10 Clocks

The PWM module runs at a frequency ≥120 MHz.

26.11 Interrupts

Each of the submodules within the PWM can generate an interrupt from several sources. The fault logic can also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 497. Interrupt summary

Core interrupt flag	Interrupt flag	Interrupt enable	Name	Description
COF0	CMPF_0	CMPIE_0	Submodule 0 compare interrupt	Compare event has occurred
CAF0	CFX1_0, CFX0_0	CFX1IE_0, CFX0IE_0	Submodule 0 input capture interrupt	Input capture event has occurred
RF0	RF_0	RIE_0	Submodule 0 reload interrupt	Reload event has occurred
COF1	CMPF_1	CMPIE_1	Submodule 1 compare interrupt	Compare event has occurred
CAF1	CFX1_1, CFX0_1	CFX1IE_1, CFX0IE_1	Submodule 1 input capture interrupt	Input capture event has occurred
RF1	RF_1	RIE_1	Submodule 1 reload interrupt	Reload event has occurred
COF2	CMPF_2	CMPIE_2	Submodule 2 compare interrupt	Compare event has occurred
CAF2	CFX1_2, CFX0_2	CFX1IE_2, CFX0IE_2	Submodule 2 input capture interrupt	Input capture event has occurred
RF2	RF_2	RIE_2	Submodule 2 reload interrupt	Reload event has occurred
COF3	CMPF_3	CMPIE_3	Submodule 3 compare interrupt	Compare event has occurred
CAF3	CFX1_3, CFX0_3	CFX1IE_3, CFX0IE_3	Submodule 3 input capture interrupt	Input capture event has occurred
RF3	RF_3	RIE_3	Submodule 3 reload interrupt	Reload event has occurred

Table 497. Interrupt summary(Continued)

Core interrupt flag	Interrupt flag	Interrupt enable	Name	Description
REF	REF_0	REIE_0	Submodule 0 reload error interrupt	Reload error has occurred
	REF_1	REIE_1	Submodule 1 reload error interrupt	
	REF_2	REIE_2	Submodule 2 reload error interrupt	
	REF_3	REIE_3	Submodule 3 reload error interrupt	
FFLAG	FFLAG	FIE	Fault input interrupt	Fault condition has been detected

26.12 DMA

Each submodule can request a DMA read access for its capture FIFOs and a DMA write request for its double buffered VALx registers.

Table 498. DMA summary

DMA request	DMA enable	Name	Description
Submodule 0 read request	CX0DE_0	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_0	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_0	Capture FIFO read request source select	Selects source of read DMA request
Submodule 0 write request	VALDE_0	VALx write request	VALx registers need to be updated
Submodule 1 read request	CX0DE_1	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_1	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_1	Capture FIFO read request source select	Selects source of read DMA request
Submodule 1 write request	VALDE_1	VALx write request	VALx registers need to be updated
	CX0DE_2	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_2	Capture FIFO X1 read request	CVAL1 contains a value to be read
	CAPTDE_2	Capture FIFO read request source select	Selects source of read DMA request
Submodule 2 write request	VALDE_2	VALx write request	VALx registers need to be updated
Submodule 3 read request	CX0DE_3	Capture FIFO X0 read request	CVAL0 contains a value to be read
	CX1DE_3	Capture FIFO X1 read request	CVAL1 contains a value to be read

Table 498. DMA summary(Continued)

DMA request	DMA enable	Name	Description
	CAPTDE_3	Capture FIFO read request source select	Selects source of read DMA request
Submodule 3 write request	VALDE_3	VALx write request	VALx registers need to be updated

27 eTimer

27.1 Introduction

The eTimer module contains six identical counter/timer channels and one watchdog timer function only for eTimer_0. Each 16-bit counter/timer channel contains a prescaler, a counter, a load register, a hold register, two queued capture registers, two compare registers, two compare preload registers, and four control registers.

The Load register provides the initialization value to the counter when the counter's terminal value has been reached. For true modulo counting the counter can also be initialized by the CMPLD1 or CMPLD2 registers.

The Hold register captures the counter's value when other counters are being read. This feature supports the reading of cascaded counters coherently.

The Capture registers enable an external signal to take a "snapshot" of the counter's current value.

The COMP1 and COMP2 registers provide the values to which the counter is compared. If a match occurs, the OFLAG signal can be set, cleared, or toggled. At match time, an interrupt is generated if enabled, and the new compare value is loaded into the COMP1 or COMP2 registers from CMPLD1 and CMPLD2 if enabled.

The Prescaler provides different time bases useful for clocking the counter/timer.

The Counter provides the ability to count internal or external events.

Within the eTimer module (set of six timer/counter channels) the input pins are shareable.

27.2 Features

The eTimer module design includes these features:

- Six 16-bit counters/timers
- Count up/down
- Cascadeable counters
- Enhanced programmable up/down modulo counting
- Max count rate equals peripheral clock/2 for external clocks
- Max count rate equals peripheral clock for internal clocks
- Count once or repeatedly
- Preloadable counters
- Preloadable compare registers
- Counters can share available input pins
- Separate prescaler for each counter
- Each counter has capture and compare capability
- Continuous and single shot capture for enhanced speed measurement
- DMA support of capture registers and compare registers
- 32-bit watchdog capability to detect stalled quadrature counting (implemented only on eTimer_0)
- OFLAG comparison for safety critical applications
- Programmable operation during debug mode and stop mode
- Programmable input filter
- Counting start can be synchronized across counters

27.3 Module block diagram

The eTimer block diagram is shown in *Figure 556*.

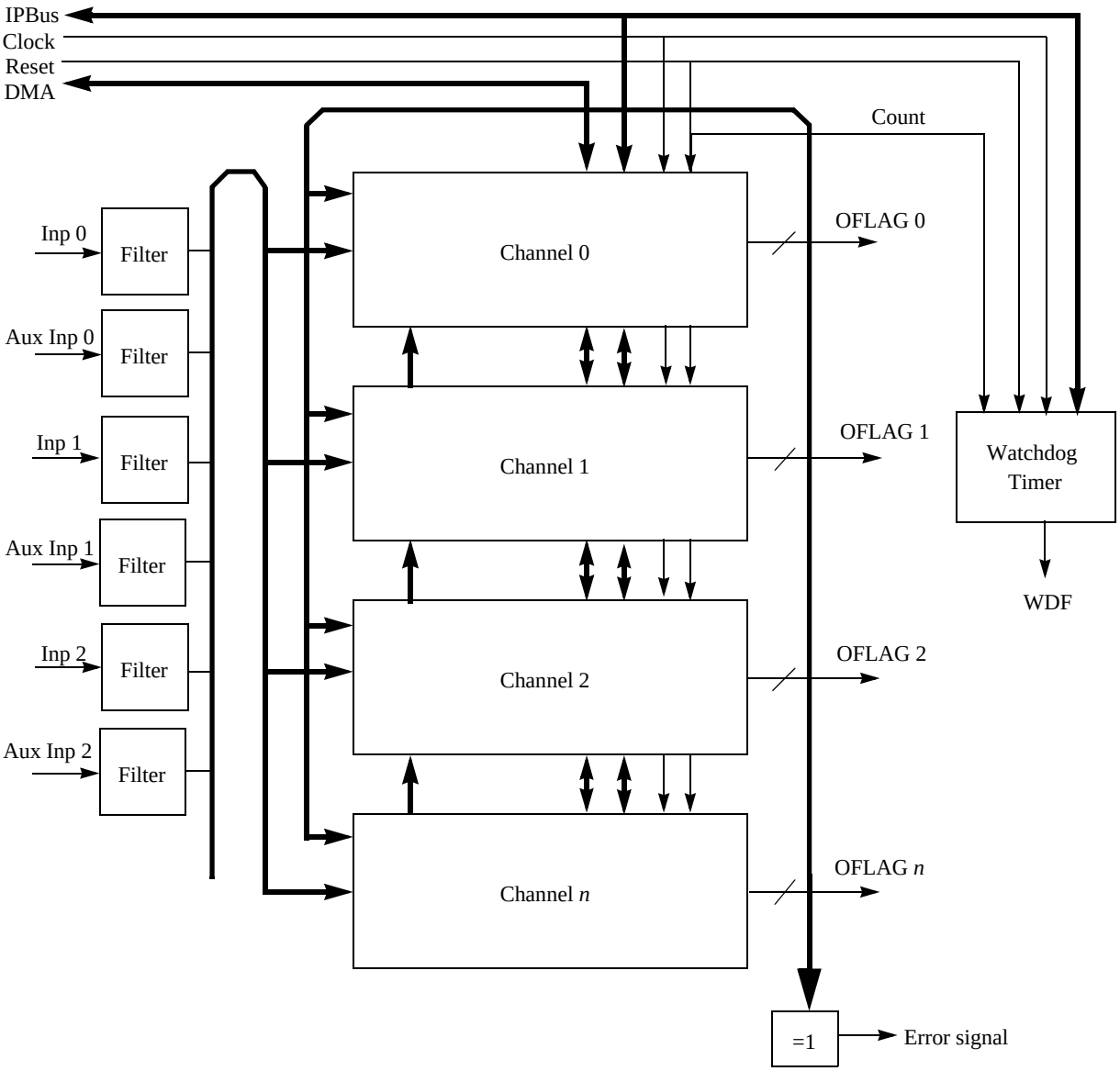


Figure 556. eTimer block diagram

27.4 Channel block diagram

Each of the timer/counter channels within the eTimer are shown in *Figure 557*.

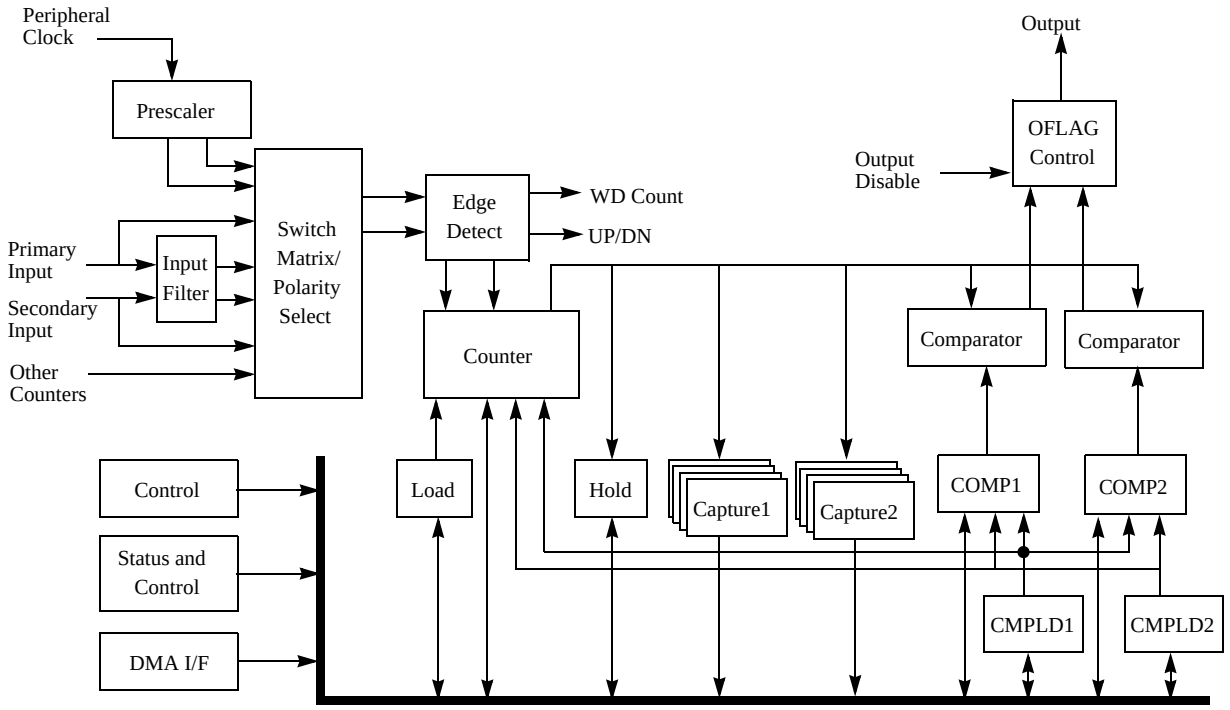


Figure 557. eTimer channel block diagram

27.5 External signal descriptions

The eTimer module has six external signals that can be used as either inputs or outputs.

27.5.1 ETC[5:0]—eTimer input/outputs

These pins can be independently configured to be either timer input sources or output flags.

27.6 Memory map and registers

27.6.1 Overview

Table 499 shows the memory map for the eTimer modules.

Table 499. eTimer memory map

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
eTimer Channel 0		
0x0000	COMP1—Compare Register 1	*on page 893*
0x0002	COMP2—Compare Register 2	*on page 893*
0x0004	CAPT1—Capture Register 1	*on page 893*
0x0006	CAPT2—Capture Register 2	*on page 894*
0x0008	LOAD—Load Register	*on page 894*
0x000A	HOLD—Hold Register	*on page 895*
0x000C	CNTR—Counter Register	*on page 895*
0x000E	CTRL1—Control Register 1	*on page 896*
0x0010	CTRL2—Control Register 2	*on page 898*
0x0012	CTRL3—Control Register 3	*on page 900*
0x0014	STS—Status Register	*on page 901*
0x0016	INTDMA—Interrupt and DMA Enable Register	*on page 902*
0x0018	CMPLD1—Comparator Load Register 1	*on page 903*
0x001A	CMPLD2—Comparator Load Register 2	*on page 904*
0x001C	CCCTRL—Compare and Capture Control Register	*on page 904*
0x001E	FILT—Input Filter Register	*on page 906*
eTimer Channel 1		
0x0020	COMP1—Compare Register 1	*on page 893*
0x0022	COMP2—Compare Register 2	*on page 893*
0x0024	CAPT1—Capture Register 1	*on page 893*
0x0026	CAPT2—Capture Register 2	*on page 894*
0x0028	LOAD—Load Register	*on page 894*
0x002A	HOLD—Hold Register	*on page 895*
0x002C	CNTR—Counter Register	*on page 895*
0x002E	CTRL1—Control Register 1	*on page 896*
0x0030	CTRL2—Control Register 2	*on page 898*
0x0032	CTRL3—Control Register 3	*on page 900*
0x0034	STS—Status Register	*on page 901*

Table 499. eTimer memory map(Continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x0036	INTDMA—Interrupt and DMA Enable Register	on page 902
0x0038	CMPLD1—Comparator Load Register 1	on page 903
0x003A	CMPLD2—Comparator Load Register 2	on page 904
0x003C	CCCTRL—Compare and Capture Control Register	on page 904
0x003E	FILT—Input Filter Register	on page 906
eTimer Channel 2		
0x0040	COMP1—Compare Register 1	on page 893
0x0042	COMP2—Compare Register 2	on page 893
0x0044	CAPT1—Capture Register 1	on page 893
0x0046	CAPT2—Capture Register 2	on page 894
0x0048	LOAD—Load Register	on page 894
0x004A	HOLD—Hold Register	on page 895
0x004C	CNTR—Counter Register	on page 895
0x004E	CTRL1—Control Register 1	on page 896
0x0050	CTRL2—Control Register 2	on page 898
0x0052	CTRL3—Control Register 3	on page 900
0x0054	STS—Status Register	on page 901
0x0056	INTDMA—Interrupt and DMA Enable Register	on page 902
0x0058	CMPLD1—Comparator Load Register 1	on page 903
0x005A	CMPLD2—Comparator Load Register 2	on page 904
0x005C	CCCTRL—Compare and Capture Control Register	on page 904
0x005E	FILT—Input Filter Register	on page 906
eTimer Channel 3		
0x0060	COMP1—Compare Register 1	on page 893
0x0062	COMP2—Compare Register 2	on page 893
0x0064	CAPT1—Capture Register 1	on page 893
0x0066	CAPT2—Capture Register 2	on page 894
0x00068	LOAD—Load Register	on page 894
0x006A	HOLD—Hold Register	on page 895
0x006C	CNTR—Counter Register	on page 895

Table 499. eTimer memory map(Continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x006E	CTRL1—Control Register 1	on page 896
0x0070	CTRL2—Control Register 2	on page 898
0x0072	CTRL3—Control Register 3	on page 900
0x0074	STS—Status Register	on page 901
0x0076	INTDMA—Interrupt and DMA Enable Register	on page 902
0x0078	CMPLD1—Comparator Load Register 1	on page 903
0x007A	CMPLD2—Comparator Load Register 2	on page 904
0x007C	CCCTRL—Compare and Capture Control Register	on page 904
0x007E	FILT—Input Filter Register	on page 906
eTimer Channel 4		
0x0080	COMP1—Compare Register 1	on page 893
0x0082	COMP2—Compare Register 2	on page 893
0x0084	CAPT1—Capture Register 1	on page 893
0x0086	CAPT2—Capture Register 2	on page 894
0x0088	LOAD—Load Register	on page 894
0x008A	HOLD—Hold Register	on page 895
0x008C	CNTR—Counter Register	on page 895
0x008E	CTRL1—Control Register 1	on page 896
0x0090	CTRL2—Control Register 2	on page 898
0x0092	CTRL3—Control Register 3	on page 900
0x0094	STS—Status Register	on page 901
0x0096	INTDMA—Interrupt and DMA Enable Register	on page 902
0x0098	CMPLD1—Comparator Load Register 1	on page 903
0x009A	CMPLD2—Comparator Load Register 2	on page 904
0x009C	CCCTRL—Compare and Capture Control Register	on page 904
0x009E	FILT—Input Filter Register	on page 906
eTimer Channel 5		
0x00A0	COMP1—Compare Register 1	on page 893
0x00A2	COMP2—Compare Register 2	on page 893
0x00A4	CAPT1—Capture Register 1	on page 893

Table 499. eTimer memory map(Continued)

Offset from eTIMER0_BASE (FFE1_8000) eTIMER1_BASE (FFE1_C000)	Register	Location
0x00A6	CAPT2—Capture Register 2	on page 894
0x00A8	LOAD—Load Register	on page 894
0x00AA	HOLD—Hold Register	on page 895
0x00AC	CNTR—Counter Register	on page 895
0x00AE	CTRL1—Control Register 1	on page 896
0x00B0	CTRL2—Control Register 2	on page 898
0x00B2	CTRL3—Control Register 3	on page 900
0x00B4	STS—Status Register	on page 901
0x00B6	INTDMA—Interrupt and DMA Enable Register	on page 902
0x00B8	CMPLD1—Comparator Load Register 1	on page 903
0x00BA	CMPLD2—Comparator Load Register 2	on page 904
0x00BC	CCCTRL—Compare and Capture Control Register	on page 904
0x00BE	FILT—Input Filter Register	on page 906
0x00C0–0x00FF	Reserved	
0x0100	WDTOL—Watchdog Time-out Low Register	on page 907
0x0102	WDTOH—Watchdog Time-out High Register	on page 907
0x0104–0x010B	Reserved	
Watchdog and Configuration registers		
0x010C	ENBL—Channel Enable Register	on page 908
0x0110	DREQ0—DMA Request 0 Select Register	on page 908
0x0112	DREQ1—DMA Request 1 Select Register	on page 908
0x0114–0x3FFF	Reserved	

27.6.2 Timer channel registers

These registers are repeated for each timer channel. The base address of channel 0 is the same as the base address of the eTimer module as a whole. The base address of channel 1 is 0x20. This is the base address of the eTimer module plus an offset based on the number of bytes of registers in a timer channel. The base address of each subsequent timer channel is equal to the base address of the previous channel plus this same offset of 0x20.

27.6.2.1 Compare register 1 (COMP1)

The COMP1 register stores the value used for comparison with the counter value. More explanation on the use of COMP1 can be found in *Section 27.7.2.13: Usage of compare registers*.

Address: Base + 0x0000 (eTimer0), Base + 0x0020 (eTimer1), Base + 0x0040 (eTimer2), Base + 0x0060 (eTimer3), Base + 0x0080 (eTimer4), Base + 0x00A0 (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	COMP1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 558. Compare register 1 (COMP1)

Table 500. COMP1 field descriptions

Field	Description
COMP1[15:0]	Compare 1 Stores the value used for comparison with the counter value. **Note:** This register is not byte accessible.

27.6.2.2 Compare register 2 (COMP2)

The COMP2 register stores the value used for comparison with the counter value. More explanation on the use of COMP2 can be found in *Section 27.7.2.13: Usage of compare registers*.

Address: Base + 0x0002 (eTimer0), Base + 0x0022 (eTimer1), Base + 0x0042 (eTimer2), Base + 0x0062 (eTimer3), Base + 0x0082 (eTimer4), Base + 0x00A2 (eTimer5) — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	COMP2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 559. Compare register 2 (COMP2)

Table 501. COMP2 field descriptions

Field	Description
COMP2[15:0]	Compare 2 Stores the value used for comparison with the counter value. **Note:** This register is not byte accessible.

27.6.2.3 Capture register 1 (CAPT1)

The CAPT1 register stores the value captured from the counter while the counter is enabled (CNTMODE != 000). Exactly when a capture occurs is defined by the CPT1MODE bits in the Compare and Capture Control (CCCTRL) register. This is actually a 2-deep FIFO and not a single register.

Address: Base + 0x0004 (eTimer0) Base + 0x0064 (eTimer3)
Base + 0x0024 (eTimer1) Base + 0x0084 (eTimer4)
Base + 0x0044 (eTimer2) Base + 0x00A4 (eTimer5) Access: User read-only

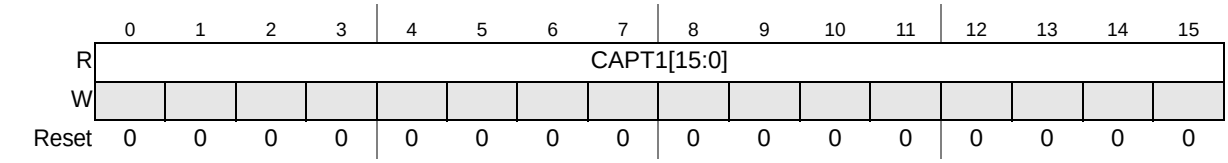


Figure 560. Capture register 1 (CAPT1)

Table 502. CAPT1 field descriptions

Field	Description
CAPT1[15:0]	Capture 1 Stores the value captured from the counter. **Note:** This register is not byte accessible.

27.6.2.4 Capture register 2 (CAPT2)

This read only register stores the value captured from the counter while the counter is enabled (CNTMODE != 000). Exactly when a capture occurs is defined by the CPT2MODE bits. This is actually a 2-deep FIFO and not a single register. This register is not byte accessible.

Address: Base + 0x0006 (eTimer0) Base + 0x0066 (eTimer3) Access: User read-only
Base + 0x0026 (eTimer1) Base + 0x0086 (eTimer4)
Base + 0x0046 (eTimer2) Base + 0x00A6 (eTimer5)

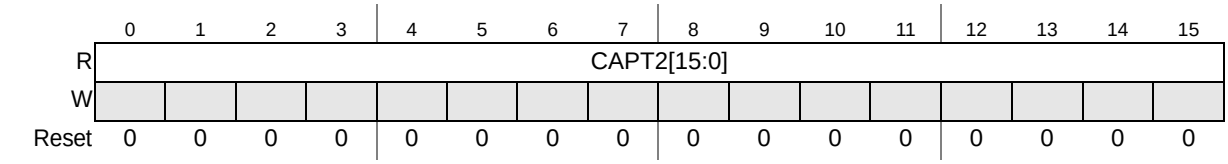


Figure 561. Capture register 2 (CAPT2)

Table 503. CAPT2 field descriptions

Field	Description
CAPT2[15:0]	Capture 2 Stores the value captured from the counter. **Note:** This register is not byte accessible.

27.6.2.5 Load register (LOAD)

This read/write register stores the value used to initialize the counter. This register is not byte accessible.

Address: Base + 0x0008 (eTimer0) Base + 0x0068 (eTimer3) Access: User read/write
Base + 0x0028(eTimer1) Base + 0x0088 (eTimer4)
Base + 0x0048 (eTimer2) Base + 0x00A8 (eTimer5)

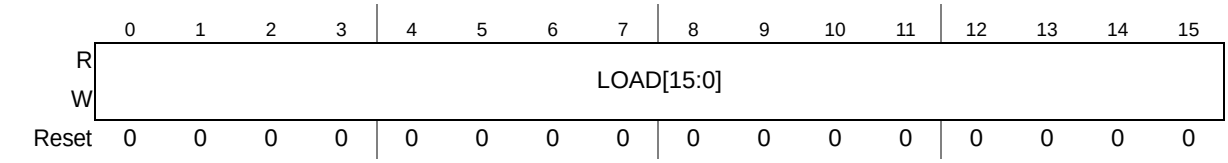


Figure 562. Load register (LOAD)

Table 504. LOAD field descriptions

Field	Description
LOAD[15:0]	Load Stores the value used to initialize the counter. **Note:** This register is not byte accessible.

27.6.2.6 Hold register (HOLD)

This read-only register stores the counter's value whenever any of the other counters within a module are read. This supports coherent reading of cascaded counters.

Address: Base + 0x000A (eTimer0) Base + 0x006A (eTimer3) Access: User read-only
Base + 0x002A (eTimer1) Base + 0x008A (eTimer4)
Base + 0x004A (eTimer2) Base + 0x00AA (eTimer5)

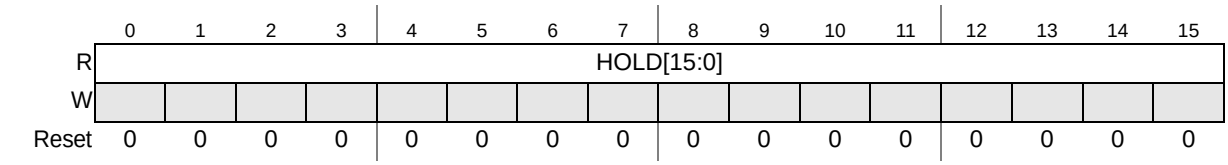


Figure 563. Hold register (HOLD)

Table 505. HOLD field descriptions

Field	Description
HOLD[15:0]	Stores the counter's value whenever any of the other counters within a module are read. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQRL[ERQn] bit.

27.6.2.7 Counter register (CNTR)

This read/write register is the counter for this channel of the timer module. This register is not byte accessible.

Address: Base + 0x000C (eTimer0) Base + 0x006C (eTimer3) Access: User read/write
Base + 0x002C (eTimer1) Base + 0x008C (eTimer4)
Base + 0x004C (eTimer2) Base + 0x00AC (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTR[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 564. Counter register (CNTR)

Table 506. CNTR field descriptions

Field	Description
CNTR[15:0]	Contains the count value for this channel of the eTimer module. **Note:** This register is not byte accessible.

27.6.2.8 Control register 1 (CTRL1)

Address: Base + 0x000E(eTimer0) Base + 0x006E (eTimer3) Access: User read/write
Base + 0x002E (eTimer1) Base + 0x008E (eTimer4)
Base + 0x004E (eTimer2) Base + 0x00AE (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CNTMODE[2:0]			PRISRC[4:0]					ONCE	LENGTH	DIR	SECSRC[4:0]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 565. Control register 1 (CTRL1)

Table 507. CTRL1 field descriptions

Field	Description
CNTMODE[2:0]	Count Mode These bits control the basic counting and behavior of the counter. 000 No Operation. 001 Count rising edges of primary source. Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. If primary count source is IP bus clock, only rising edges are counted regardless of PIPS value. 010 Count rising and falling edges of primary source. IP Bus clock divide by 1 can not be used as a primary count source in edge count mode. 011 Count rising edges of primary source while secondary input high active. 100 Quadrature count mode, uses primary and secondary sources. 101 Count primary source rising edges, secondary source specifies direction (1 = minus). Rising edges counted only when PIPS = 0. Falling edges counted when PIPS = 1. 110 Edge of secondary source triggers primary count till compare. 111 Cascaded counter mode, up/down. Primary count source must be set to one of the counter outputs.
PRISRC	Primary Count Source These bits select the primary count source. See *Table 508*. **Note:** A timer cannot select its own output as its primary count source. If this is done, the timer will not count.

Table 507. CTRL1 field descriptions(Continued)

Field	Description
ONCE	Count Once This bit selects continuous or one-shot counting mode. 0 Count repeatedly. 1 Count until compare and then stop.
LENGTH	Count Length This bit determines whether the counter counts to the compare value and then reinitializes itself to the value specified in the LOAD, CMPLD1, or CMPLD2 registers, or the counter continues counting past the compare value, to the binary roll over. 0 Continue counting to roll over. 1 Count until compare, then reinitialize. The value that the counter is reinitialized with depends on the settings of CLC1 and CLC2. If neither of these indicates the counter is to be loaded from one of the CMPLD registers, then the LOAD register reinitializes the counter upon matching either COMP register. If one of CLC1 or CLC2 indicates that the counter is to be loaded from one of the CMPLD registers, then the counter will reinitialize to the value in the appropriate CMPLD register upon a match with the appropriate COMP register. If both of the CLC1 and CLC2 fields indicate that the counter is to be loaded from the CMPLD registers, then CMPLD1 will have priority if both compares happen at the same value. When output mode 0x4 is used, alternating values of COMP1 and COMP2 are used to generate successful comparisons. For example, the counter counts until COMP1 value is reached, reinitializes, then counts until COMP2 value is reached, reinitializes, then counts until COMP1 value is reached, etc.
DIR	Count Direction This bit selects either the normal count direction up, or the reverse direction, down. 0 Count up. 1 Count down.
SECSRC	Secondary Count Source These bits identify the source to be used as a count command or timer command. The selected input can trigger the timer to capture the current value of the CNTR register. The selected input can also be used to specify the count direction. The polarity of the signal can be inverted by the SIPS bit of the CTRL2 register.

Table 508. Count source values

Value	Meaning	Value	Meaning
00000	Counter #0 input pin	10000	Counter #0 output
00001	Counter #1 input pin	10001	Counter #1 output
00010	Counter #2 input pin	10010	Counter #2 output
00011	Counter #3 input pin	10011	Counter #3 output
00100	Counter #4 input pin	10100	Counter #4 output
00101	Counter #5 input pin	10101	Counter #5 output
00110	*Reserved*	10110	*Reserved*
00111	*Reserved*	10111	*Reserved*
01000	Auxiliary input #0 pin	11000	IP Bus clock divide by 1 prescaler
01001	Auxiliary input #1 pin	11001	IP Bus clock divide by 2 prescaler

Table 508. Count source values(Continued)

Value	Meaning	Value	Meaning
01010	Auxiliary input #2 pin	11010	IP Bus clock divide by 4 prescaler
01011	*Reserved*	11011	IP Bus clock divide by 8 prescaler
01100	*Reserved*	11100	IP Bus clock divide by 16 prescaler
01101	*Reserved*	11101	IP Bus clock divide by 32 prescaler
01110	*Reserved*	11110	IP Bus clock divide by 64 prescaler
01111	*Reserved*	11111	IP Bus clock divide by 128 prescaler

27.6.2.9 Control register 2 (CTRL2)

Address: Base + 0x0010 (eTimer0) Base + 0x0070 (eTimer3) Access: User read/write
Base + 0x0030 (eTimer1) Base + 0x0090 (eTimer4)
Base + 0x0050 (eTimer2) Base + 0x00B0 (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OEN	RDNT	INPUT	VAL	0	CO FRC	COINIT		SIPS	PIPS	OPS	MSTR	OUTMODE[3:0]			
W					FORCE											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 566. Control register 2 (CTRL2)

Table 509. CTRL2 field descriptions

Field	Description
OEN	Output Enable This bit determines the direction of the external pin. 0 The external pin is configured as an input. 1 OFLAG output signal is driven on the external pin. Other timer channels using this external pin as their input will see the driven value. The polarity of the signal will be determined by the OPS bit.
RDNT	Redundant Channel Enable This bit enables redundant channel checking between adjacent channels (0 and 1, 2 and 3, 4 and 5). When this bit is cleared, the RCF bit in this channel cannot be set. When this bit is set, the RCF bit is set by a miscompare between the OFLAG of this channel and the OFLAG of its redundant adjacent channel, which causes the output of this channel to go inactive (logic 0 prior to consideration of the OPS bit). 0 Disable redundant channel checking. 1 Enable redundant channel checking.
INPUT	External input signal This read only bit reflects the current state of the signal selected via SECSRC after application of the SIPS bit and filtering.
VAL	Forced OFLAG Value This bit determines the value of the OFLAG output signal when a software triggered FORCE command occurs.

Table 509. CTRL2 field descriptions(Continued)

Field	Description
FORCE	Force the OFLAG output This write only bit forces the current value of the VAL bit to be written to the OFLAG output. This bit always reads as a zero. The VAL and FORCE bits can be written simultaneously in a single write operation. Write to the FORCE bit only if OUTMODE is 0000 (software controlled). Setting this bit while the OUTMODE is a different value may yield unpredictable results.
COFRC	Co-channel OFLAG Force This bit enables the compare from another channel within the module to force the state of this counter's OFLAG output signal. 0 Other channels cannot force the OFLAG of this channel. 1 Other channels may force the OFLAG of this channel.
COINIT	Co-channel Initialization These bits enable another channel within the module to force the reinitialization of this channel when the other channel has an active compare event. 00 Other channels cannot force reinitialization of this channel. 01 Other channels may force a reinitialization of this channel's counter using the LOAD reg. 10 Other channels may force a reinitialization of this channel's counter with the CMPLD2 reg when this channel is counting down or the CMPLD1 reg when this channel is counting up. 11 Reserved.
SIPS	Secondary Source Input Polarity Select This bit inverts the polarity of the signal selected by the SECSRC bits. 0 True polarity. 1 Inverted polarity.
PIPS	Primary Source Input Polarity Select This bit inverts the polarity of the signal selected by the PRISRC bits. This only applies if the signal selected by PRISRC is not the prescaled IP Bus clock. 0 True polarity. 1 Inverted polarity.
OPS	Output Polarity Select This bit inverts the OFLAG output signal polarity. 0 True polarity. 1 Inverted polarity.

Table 509. CTRL2 field descriptions(Continued)

Field	Description
MSTR	Master Mode This bit enables the compare function's output to be broadcast to the other channels in the module. The compare signal then can be used to reinitialize the other counters and/or force their OFLAG signal outputs. 0 Disable broadcast of compare events from this channel. 1 Enable broadcast of compare events from this channel.
OUTMODE	Output Mode These bits determine the mode of operation for the OFLAG output signal. 0000 Software controlled 0001 Clear OFLAG output on successful compare (COMP1 or COMP2) 0010 Set OFLAG output on successful compare (COMP1 or COMP2) 0011 Toggle OFLAG output on successful compare (COMP1 or COMP2) 0100 Toggle OFLAG output using alternating compare registers 0101 Set on compare with COMP1, cleared on secondary source input edge 0110 Set on compare with COMP2, cleared on secondary source input edge 0111 Set on compare, cleared on counter roll-over 1000 Set on successful compare on COMP1, clear on successful compare on COMP2 1001 Asserted while counter is active, cleared when counter is stopped. 1010 Asserted when counting up, cleared when counting down. 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Enable gated clock output while counter is active

27.6.2.10 Control register 3 (CTRL3)

Address: Base + 0x0012 (eTimer0) Base + 0x0072 (eTimer3) Access: User read/write
Base + 0x0032 (eTimer1) Base + 0x0092 (eTimer4)
Base + 0x0052 (eTimer2) Base + 0x00B2 (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STPEN	ROC		0	1	1	1	1	C2FCNT[2:0]			C1FCNT[2:0]			DBGEN [1:0]	
W																
Reset	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0

Figure 567. Control register 3 (CTRL3)

Table 510. CTRL3 field descriptions

Field	Description
STPEN	Stop Actions Enable This bit allows the tristating of the timer output during stop mode. 0 Output enable is unaffected by stop mode. 1 Output enable is disabled during stop mode.

Table 510. CTRL3 field descriptions(Continued)

Field	Description
ROC	Reload on Capture These bits enable the capture function to cause the counter to be reloaded from the LOAD register. 00 Do not reload the counter on a capture event. 01 Reload the counter on a capture 1 event. 10 Reload the counter on a capture 2 event. 11 Reload the counter on both a capture 1 event and a capture 2 event.
C2FCNT	CAPT2 FIFO Word Count This field reflects the number of words in the CAPT2 FIFO.
C1FCNT	CAPT1 FIFO Word Count This field reflects the number of words in the CAPT1 FIFO.
DBGEN	Debug Actions Enable These bits allow the counter channel to perform certain actions in response to the device entering debug mode. 00 Continue with normal operation during debug mode. (default) 01 Halt channel counter during debug mode. 10 Force OFLAG to logic 0 (prior to consideration of the OPS bit) during debug mode. 11 Both halt counter and force OFLAG to 0 during debug mode.

27.6.2.11 Status register (STS)

Address: Base + 0x0014 (eTimer0) Base + 0x0074 (eTimer3) Access: User read/write
Base + 0x0034 (eTimer1) Base + 0x0094 (eTimer4)
Base + 0x0054 (eTimer2) Base + 0x00B4 (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	WDF	RCF	ICF2	ICF1	IEHF	IELF	TOF	TCF2	TCF1	TCF
W							w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 568. Status register (STS)

Table 511. STS field descriptions

Field	Description
WDF	Watchdog Time-out Flag This bit is set when the watchdog times out by counting down to zero. The watchdog must be enabled for time-out to occur and channel 0 must be in quadrature decode count mode (CNTMODE = 100). This bit is cleared by writing a 1 to this bit. This bit is used in channel 0 only.
RCF	Redundant Channel Flag This bit is set when there is a miscompare between this channel's OFLAG value and the OFLAG value of the corresponding redundant channel. Corresponding channels are grouped together in the following pairs: 0 and 1, 2 and 3, 4 and 5, or 6 and 7. This bit can only be set if the RDNT bit is set. This bit is cleared by writing a 1 to this bit. This bit is used in even channels (0, 2, 4, and 6) only.
ICF2	Input Capture 2 Flag This bit is set when an input capture event (as defined by CPT2MODE) occurs and the word count of the CAPT2 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a 1 to this bit if ICF2DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF2DE is set (DMA).

Table 511. STS field descriptions(Continued)

Field	Description
ICF1	Input Capture 1 Flag This bit is set when an input capture event (as defined by CPT1MODE) occurs and the word count of the CAPT1 FIFO exceeds the value of the CFWM field. This bit is cleared by writing a 1 to this bit if ICF1DE is clear (no DMA) or it is cleared automatically by the DMA access if ICF1DE is set (DMA).
IEHF	Input Edge High Flag This bit is set when a positive input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a 1 to this bit.
IELF	Input Edge Low Flag This bit is set when a negative input transition occurs (on an input selected by SECSRC) while the counter is enabled. This bit is cleared by writing a 1 to this bit.
TOF	Timer Overflow Flag This bit is set when the counter rolls over its maximum value 0xFFFF or 0x0000 (depending on count direction). This bit is cleared by writing a 1 to this bit.
TCF2	Timer Compare 2 Flag This bit is set when a successful compare occurs with COMP2. This bit is cleared by writing a 1 to this bit.
TCF1	Timer Compare 1 Flag This bit is set when a successful compare occurs with COMP1. This bit is cleared by writing a 1 to this bit.
TCF	Timer Compare Flag This bit is set when a successful compare occurs. This bit is cleared by writing a 1 to this bit.

27.6.2.12 Interrupt and DMA enable register (INTDMA)

Address: Base + 0x0016 (eTimer0) Base + 0x0076 (eTimer3) Access: User read/write
Base + 0x0036 (eTimer1) Base + 0x0096 (eTimer4)
Base + 0x0056 (eTimer2) Base + 0x00B6 (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ICF2DE	ICF1DE	CMPLD2DE	CMPLD1DE	0	0	WDFIE	RCFIE	ICF2IE	ICF1IE	IEHFIE	IELFIE	TOFIE	TCF2IE	TCF1IE	TCFIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 569. Interrupt and DMA enable register (INTDMA)

Table 512. INTDMA field descriptions

Field	Description
ICF2DE	Input Capture 2 Flag DMA Enable Setting this bit enables DMA read requests for CAPT2 when the ICF2 bit is set. Do not set both this bit and the ICF2IE bit.
ICF1DE	Input Capture 1 Flag DMA Enable Setting this bit enables DMA read requests for CAPT1 when the ICF1 bit is set. Do not set both this bit and the ICF1IE bit.

Table 512. INTDMA field descriptions(Continued)

Field	Description
CMPLD2DE	Comparator Load Register 2 Flag DMA Enable Setting this bit enables DMA write requests to the CMPLD2 register whenever data is transferred out of the CMPLD2 reg into either the CNTR, COMP1, or COMP2 registers.
CMPLD1DE	Comparator Load Register 1 Flag DMA Enable Setting this bit enables DMA write requests to the CMPLD1 register whenever data is transferred out of the CMPLD1 reg into either the CNTR, COMP1, or COMP2 registers.
WDFIE	Watchdog Flag Interrupt Enable Setting this bit enables interrupts when the WDF bit is set. This bit is used in channel 0 only.
RCFIE	Redundant Channel Flag Interrupt Enable Setting this bit enables interrupts when the RCF bit is set. This bit is used in even channels (0, 2, 4) only.
ICF2IE	Input Capture 2 Flag Interrupt Enable Setting this bit enables interrupts when the ICF2 bit is set. Do not set both this bit and the ICF2DE bit.
ICF1IE	Input Capture 1 Flag Interrupt Enable Setting this bit enables interrupts when the ICF1 bit is set. Do not set both this bit and the ICF1DE bit.
IEHFIE	Input Edge High Flag Interrupt Enable Setting this bit enables interrupts when the IEHF bit is set.
IELFIE	Input Edge Low Flag Interrupt Enable Setting this bit enables interrupts when the IELF bit is set.
TOFIE	Timer Overflow Flag Interrupt Enable Setting this bit enables interrupts when the TOF bit is set.
TCF2IE	Timer Compare 2 Flag Interrupt Enable Setting this bit enables interrupts when the TCF2 bit is set.
TCF1IE	Timer Compare 1 Flag Interrupt Enable Setting this bit enables interrupts when the TCF1 bit is set.
TCFIE	Timer Compare Flag Interrupt Enable Setting this bit enables interrupts when the TCF bit is set.

27.6.2.13 Comparator Load register 1 (CMPLD1)

This read/write register is the preload value for the COMP1 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section 27.7.2.14: Usage of Compare Load registers*.

Address: Base + 0x0018 (eTimer0) Base + 0x0078 (eTimer3) Access: User read/write
Base + 0x0038 (eTimer1) Base + 0x0098 (eTimer4)
Base + 0x0058 (eTimer2) Base + 0x00B8 (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD1[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 570. Comparator Load 1 (CMPLD1)

Table 513. CMPLD1 field descriptions

Field	Description
CMPLD1[15:0]	Specifies the preload value for the COMP1 register.

27.6.2.14 Comparator Load register 2 (CMPLD2)

This read/write register is the preload value for the COMP2 register. This register can also be used to load into the CNTR register. This register is not byte accessible. More information on the use of this register can be found in *Section 27.7.2.14: Usage of Compare Load registers*.

Address: Base + 0x001A (eTimer0) Base + 0x007A (eTimer3) Access: User read/write
Base + 0x003A (eTimer1) Base + 0x009A (eTimer4)
Base + 0x005A (eTimer2) Base + 0x00BA (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPLD2[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 571. Comparator Load 2 (CMPLD2)

Table 514. CMPLD2 field descriptions

Field	Description
CMPLD2[15:0]	Specifies the preload value for the COMP2 register.

27.6.2.15 Compare and Capture Control register (CCCTRL)

Address: Base + 0x001C (eTimer0) Base + 0x007C (eTimer3) Access: User read/write
Base + 0x003C (eTimer1) Base + 0x009C (eTimer4)
Base + 0x005C (eTimer2) Base + 0x00BC (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CLC2[1:0]				CLC1[1:0]		CMPMODE[1:0]		CPT2 MODE[1:0]		CPT1 MODE[1:0]		CFWM[1:0]		ONE SHOT	ARM
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 572. Compare and Capture Control register (CCCTRL)

Table 515. CCCTRL field descriptions

Field	Description
CLC2	Compare Load Control 2 These bits control when COMP2 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved 010 Load COMP2 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP2 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP2 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP2 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD2 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD2 upon successful compare with the value in COMP2.
CLC1	Compare Load Control 1 These bits control when COMP1 is preloaded. It also controls the loading of CNTR. 000 Never preload. 001 Reserved 010 Load COMP1 with CMPLD1 upon successful compare with the value in COMP1. 011 Load COMP1 with CMPLD1 upon successful compare with the value in COMP2. 100 Load COMP1 with CMPLD2 upon successful compare with the value in COMP1. 101 Load COMP1 with CMPLD2 upon successful compare with the value in COMP2. 110 Load CNTR with CMPLD1 upon successful compare with the value in COMP1. 111 Load CNTR with CMPLD1 upon successful compare with the value in COMP2.
CMPMODE	Compare Mode These bits control when the COMP1 and COMP2 registers are used in regards to the counting direction. 00 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting up. 01 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting up. 10 COMP1 register is used when the counter is counting up. COMP2 register is used when the counter is counting down. 11 COMP1 register is used when the counter is counting down. COMP2 register is used when the counter is counting down.
CPT2MODE	Capture 2 Mode Control These bits control the operation of the CAPT2 register as well as the operation of the ICF2 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CPT1MODE	Capture 1 Mode Control These bits control the operation of the CAPT1 register as well as the operation of the ICF1 flag by defining which input edges cause a capture event. The input source is the secondary count source. 00 Disabled. 01 Capture falling edges. 10 Capture rising edges. 11 Capture any edge.
CFWM	Capture FIFO Water Mark This field represents the water mark level for the CAPT1 and CAPT2 FIFOs. The capture flags, ICF1 and ICF2, are not set until the word count of the corresponding FIFO is greater than this water mark level.

Table 515. CCCTRL field descriptions(Continued)

Field	Description
ONESHOT	One-Shot Capture Mode This bit selects between free-running and one-shot mode for the input capture circuitry. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and the ARM bit is cleared. No further captures will be performed until the ARM bit is set again. If only one of the capture circuits is enabled, then a single capture will occur on the enabled capture circuit and the ARM bit is then cleared. If both capture circuits are enabled, then capture circuit 1 is armed first after the ARM bit is set. Once a capture occurs, capture circuit 1 is disarmed and capture circuit 2 is armed. After capture circuit 2 performs a capture, it is disarmed and capture circuit 1 is re-armed. The process continues indefinitely. If only one of the capture circuits is enabled, then captures continue indefinitely on the enabled capture circuit. 0 Free-running mode is selected 1 One-shot mode is selected.
ARM	Arm Capture Setting this bit high starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self cleared when in one-shot mode and the enabled capture circuit(s) has had a capture event(s). 0 Input capture operation is disabled. 1 Input capture operation as specified by the CPT1MODE and CPT2MODE bits is enabled.

27.6.2.16 Input Filter Register (FILT)

Address: Base + 0x001E (eTimer0) Base + 0x007E (eTimer3) Access: User read/write
Base + 0x003E (eTimer1) Base + 0x009E (eTimer4)
Base + 0x005E (eTimer2) Base + 0x00BE (eTimer5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FILT_CNT[2:0]			FILT_PER[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 573. Input Filter register (FILT)

Table 516. FILT field descriptions

Field	Description
FILT_CNT[2:0]	Input Filter Sample Count These bits represent the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0 represents 3 samples. A value of 7 represents 10 samples. The value of FILT_CNT affects the input latency as described in *Section 27.6.2.17: Input filter considerations*.
FILT_PER[7:0]	Input Filter Sample Period These bits represent the sampling period (in IPBus clock cycles) of the eTimer input signal. Each input is sampled multiple times at the rate specified by FILT_PER. If FILT_PER is 0x00 (default), then the input filter is bypassed. The value of FILT_PER affects the input latency as described in *Section 27.6.2.17: Input filter considerations*.

27.6.2.17 Input filter considerations

The FILT_PER value should be set such that the sampling period is larger the period of the expected noise. This way a noise spike will only corrupt one sample. The FILT_CNT value should be chosen to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as the probability of an incorrect sample raised to the FILT_CNT + 3 power.

The values of FILT_PER and FILT_CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT_PER to a non-zero value) introduces a latency of: $\{[(\text{FILT\_CNT} + 3) \times \text{FILT\_PER}] + 2\}$ IPBus clock periods.

27.6.3 Watchdog timer registers

The base address of the Watchdog Timer registers is equal to the base address of the eTimer plus an offset of 0x100. These registers are implemented only on eTimer_0.

27.6.3.1 Watchdog Time-Out registers (WDTOL and WDTOH)

Address: Base + 0x0100 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WDTOL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 574. Watchdog Time-out Low Word register (WDTOL)

Address: Base + 0x0102 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	WDTOH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 575. Watchdog Time-Out High Word register (WDTOH)

Table 517. WDTOL, WDTOH field descriptions

Field	Description
WDTO	Watchdog Time-out These registers are combined to form the 32-bit time-out count for the Timer watchdog function. This time-out count is used to monitor for inactivity on the inputs when channel 0 is in the quadrature decode count mode. The watchdog function is enabled whenever WDTO contains a non-zero value (although actual counting only occurs if channel 0 is in quadrature decode counting mode). The watchdog time-out down counter is loaded whenever WDTOH is written. These registers are not byte accessible. See *Section 27.7.3.5: Watchdog timer* for more information on the use of the watchdog timer. **Note:** The Watchdog registers are implemented only on eTimer_0.

27.6.4 Configuration registers

The base address of the configuration registers is equal to the base address of the eTimer plus an offset of 0x010C.

27.6.4.1 Channel Enable register (ENBL)

Address: Base + 0x010C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	ENBL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 576. Channel Enable register (ENBL)

Table 518. ENBL field descriptions

Field	Description
ENBL	Timer Channel Enable These bits enable the prescaler (if it is being used) and counter in each channel. Multiple ENBL bits can be set at the same time to synchronize the start of separate channels. If an ENBL bit is set, then the corresponding channel will start counting as soon as the CNTMODE field has a value other than 000. When an ENBL bit is clear, the corresponding channel maintains its current value. 0 Timer channel is disabled. 1 Timer channel is enabled. (default)

27.6.4.2 DMA Request Select registers (DREQ0, DREQ1)

Address: Base + 0x0110 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	DREQ0[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 577. DMA Request 0 Select register (DREQ0)

Address: Base + 0x0112 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	DREQ1[4:0]				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 578. DMA Request 1 Select register (DREQ1)

Table 519. DREQ*n* field descriptions

Field	Description
DREQ*n*_EN	DMA Request Enable Use these bits to enable each of the four module level DMA request outputs. Program the DREQ fields prior to setting the corresponding enable bit. Clearing this enable bit will remove the request but wíll not clear the flag that is causing the request. 1 = DMA request enabled. 0 = DMA request disabled.
DREQ*n*	DMA Request Select Use these fields to select which DMA request source will be muxed onto one of the two module level DMA request outputs. Make sure each of the DREQ registers is programmed with a different value else a single DMA source will cause multiple DMA requests. Enable a DMA request in the channel specific INTDMA register after the DREQ registers are programmed. 00000Channel 0 CAPT1 DMA read request 00001Channel 0 CAPT2 DMA read request 00010 Channel 0 CMPLD1 DMA write request 00011 Channel 0 CMPLD2 DMA write request 00100 Channel 1 CAPT1 DMA read request 00101 Channel 1 CAPT2 DMA read request 00110 Channel 1 CMPLD1 DMA write request 00111 Channel 1 CMPLD2 DMA write request 01000 Channel 2 CAPT1 DMA read request 01001 Channel 2 CAPT2 DMA read request 01010 Channel 2 CMPLD1 DMA write request 01011 Channel 2 CMPLD2 DMA write request 01100 Channel 3 CAPT1 DMA read request 01101 Channel 3 CAPT2 DMA read request 01110 Channel 3 CMPLD1 DMA write request 01111 Channel 3 CMPLD2 DMA write request 10000 Channel 4 CAPT1 DMA read request 10001 Channel 4 CAPT2 DMA read request 10010 Channel 4 CMPLD1 DMA write request 10011 Channel 4 CMPLD2 DMA write request 10100 Channel 5 CAPT1 DMA read request 10101 Channel 5 CAPT2 DMA read request 10110 Channel 5 CMPLD1 DMA write request 10111 Channel 5 CMPLD2 DMA write request

27.7 Functional description

27.7.1 General

Each channel has two basic modes of operation: it can count internal or external events, or it can count an internal clock source while an external input signal is asserted, thus timing the width of the external input signal.

- The counter can count the rising, falling, or both edges of the selected input pin.
- The counter can decode and count quadrature encoded input signals.
- The counter can count up and down using dual inputs in a “count with direction” format.
- The counter’s terminal count value (modulo) is programmable.
  - The value that is loaded into the counter after reaching its terminal count is programmable.
- The counter can count repeatedly, or it can stop after completing one count cycle.
- The counter can be programmed to count to a programmed value and then immediately reinitialize, or it can count through the compare value until the count “rolls over” to zero.

The external inputs to each counter/timer are shareable among each of the six channels within the module. The external inputs can be used as:

- Count commands
- Timer commands
- They can trigger the current counter value to be “captured”
- They can be used to generate interrupt requests

The polarity of the external inputs is selectable.

The primary output of each channel is the output signal OFLAG. The OFLAG output signal can be:

- Set, cleared, or toggled when the counter reaches the programmed value.
- The OFLAG output signal may be output to an external pin instead of having that pin serve as a timer input.
- The OFLAG output signal enables each counter to generate square waves, PWM, or pulse stream outputs.
- The polarity of the OFLAG output signal is programmable.

Any channel can be assigned as a Master. A master’s compare signal can be broadcast to the other channels within the module. The other channels can be configured to reinitialize their counters and/or force their OFLAG output signals to predetermined values when a Master channel’s compare event occurs.

27.7.2 Counting modes

The selected external signals are sampled at the eTimer’s base clock rate and then run through a transition detector. The maximum count rate is one-half of the eTimer’s base clock rate when using an external signal. Internal clock sources can be used to clock the counters at the eTimer’s base clock rate.

If a counter is programmed to count to a specific value and then stop, the CNTMODE field in the CTRL1 register is cleared when the count terminates.

27.7.2.1 STOP mode

When the CNTMODE field is set to 000, the counter is inert. No counting will occur. Stop mode will also disable the interrupts caused by input transitions on a selected input pin.

27.7.2.2 COUNT mode

When the CNTMODE field is set to 001, the counter will count the rising edges of the selected clock source. This mode is useful for generating periodic interrupts for timing purposes, or counting external events such as “widgets” on a conveyor belt passing a sensor. If the selected input is inverted by setting the PIPS bit, then the negative edge of the selected external input signal is counted.

See *Section 27.7.2.9: CASCADE-COUNT mode* through *Section 27.7.2.12: VARIABLE-FREQUENCY PWM mode* for additional capabilities of this operating mode.

27.7.2.3 EDGE-COUNT mode

When the CNTMODE field is set to 010, the counter will count both edges of the selected external clock source. This mode is useful for counting the changes in the external environment such as a simple encoder wheel.

27.7.2.4 GATED-COUNT mode

When the CNTMODE field is set to 011, the counter will count while the selected secondary input signal is high. This mode is used to time the duration of external events. If the selected input is inverted by setting the PIPS bit, then the counter will count while the selected secondary input is low.

27.7.2.5 QUADRATURE-COUNT mode

When the CNTMODE field is set to 100, the counter will decode the primary and secondary external inputs as quadrature encoded signals. Quadrature signals are usually generated by rotary or linear sensors used to monitor movement of motor shafts or mechanical equipment. The quadrature signals are square waves that are 90 degrees out of phase. The decoding of quadrature signal provides both count and direction information.

Figure 579 shows a timing diagram illustrating the basic operation of a quadrature incremental position encoder.

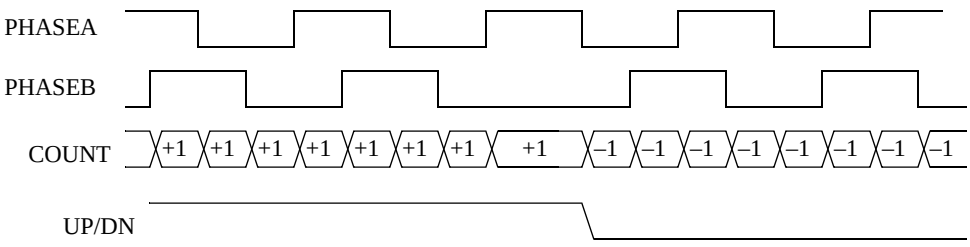


Figure 579. Quadrature incremental position encoder

27.7.2.6 SIGNED-COUNT mode

When the CNTMODE field is set to 101, the counter counts the primary clock source while the selected secondary source provides the selected count direction (up/down).

27.7.2.7 TRIGGERED-COUNT mode

When the CNTMODE field is set to 110, the counter will begin counting the primary clock source after a positive transition (negative if SIPS = 1) of the secondary input occurs. The counting will continue until a compare event occurs or another positive input transition is detected. Subsequent secondary positive input transitions will continue to restart and stop the counting until a compare event occurs.

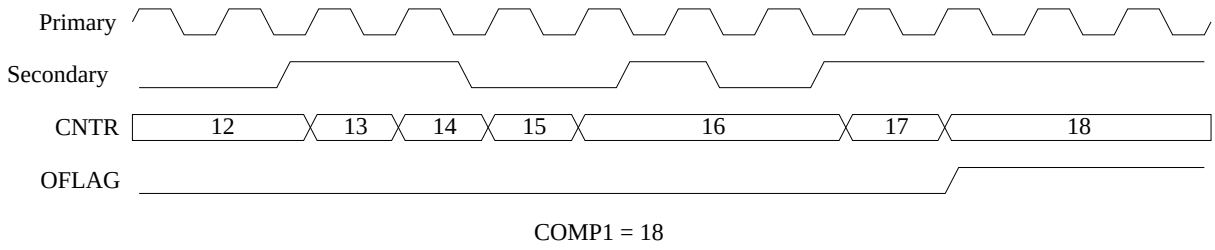


Figure 580. Triggered Count mode (length = 1)

27.7.2.8 ONE-SHOT mode

When the CNTMODE field is set to 110 and the counter is set to reinitialize at a compare event (LENGTH = 1), and the OFLAG OUTMODE is set to 0101 (cleared on init, set on compare), the counter works in ONE-SHOT mode. If an external events causes the counter to count, when terminal count is reached, the output is asserted. This delayed output can be used to provide timing delays.

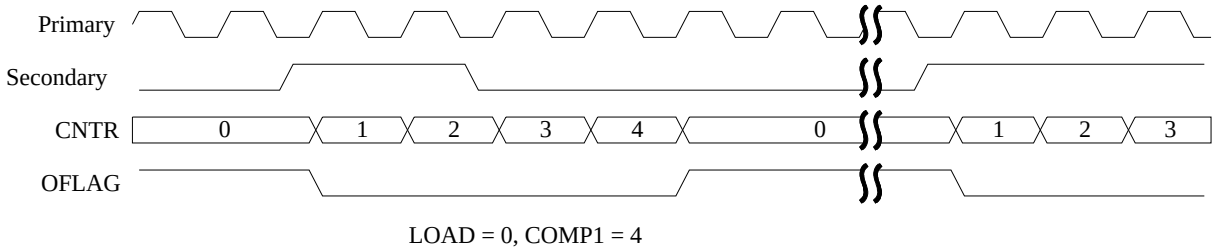


Figure 581. One-Shot mode (length = 1)

27.7.2.9 CASCADE-COUNT mode

When the CNTMODE field is set to 111, the counter's input is connected to the output of another selected counter. The counter will count up and down as compare events occur in the selected source counter. This cascaded or "daisy-chained" mode enables multiple counters to be cascaded to yield longer counter lengths. When operating in cascade mode, a special high speed signal path is used between modules rather than the OFLAG output signal. If the selected source counter is counting up and it experiences a compare event, the counter will be incremented. If the selected source counter is counting down and it experiences a compare event, the counter will be decremented.

Either one or two counters may be cascaded to create a 32-bit wide synchronous counter.

Whenever any counter is read within a counter module, all of the counters' values within the module are captured in their respective HOLD registers. This action supports the reading of a cascaded counter chain. First read any counter of a cascaded counter chain, then read the HOLD registers of the other counters in the chain. The cascaded counter mode is synchronous.

Note: It is possible to connect counters together by using the other (non-cascade) counter modes and selecting the outputs of other counters as a clock source. In this case, the counters are operating in a "ripple" mode, where higher order counters will transition a clock later than a purely synchronous design.

One channel can be cascaded with any other channel, but channels cannot be cascaded more than two deep. Separate cascades of pairs of channels can be created. For example, channels 0 and 1 can be cascaded, and channels 5 and 4 cascaded separately. However, channels 0, 1, and 5 cannot be cascaded.

27.7.2.10 PULSE-OUTPUT mode

When the counter is set up with CNTMODE = 001, and the OFLAG OUTMODE is set to 1111 (gated clock output), and the ONCE bit is set, then the counter will output a pulse stream of pulses that has the same frequency of the selected clock source, and the number of output pulses is equal to the compare value minus the init value. This mode is useful for driving step motor systems.

Note: This does not work if the PRISRC is set to 11000.

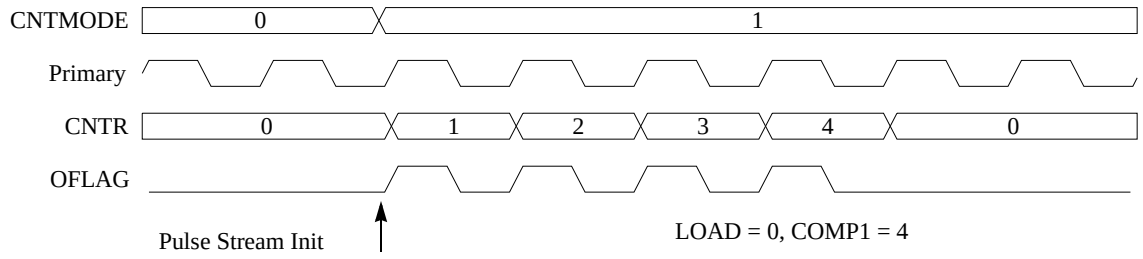


Figure 582. Pulse Output mode

27.7.2.11 FIXED-FREQUENCY PWM mode

When the counter is set up for CNTMODE = 001, count through roll-over (LENGTH = 0), continuous count (ONCE = 0) and the OFLAG OUTMODE is 0111 (set on compare, cleared on counter roll-over) then the counter output yields a Pulse Width Modulated (PWM) signal with a frequency equal to the count clock frequency divided by 65,536 and a pulse width duty cycle equal to the compare value divided by 65,536. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters.

27.7.2.12 VARIABLE-FREQUENCY PWM mode

When the counter is setup for CNTMODE = 001, count till compare (LENGTH = 1), continuous count (ONCE = 0) and the OFLAG OUTMODE is 0100 (toggle OFLAG and alternate compare registers) then the counter output yields a Pulse Width Modulated (PWM) signal whose frequency and pulse width is determined by the values programmed into the COMP1 and COMP2 registers, and the input clock frequency. This method of PWM

generation has the advantage of allowing almost any desired PWM frequency and/or constant on or off periods. This mode of operation is often used to drive PWM amplifiers used to power motors and inverters. The CMPLD1 and CMPLD2 registers are especially useful for this mode, as they allow the programmer time to calculate values for the next PWM cycle while the PWM current cycle is underway.

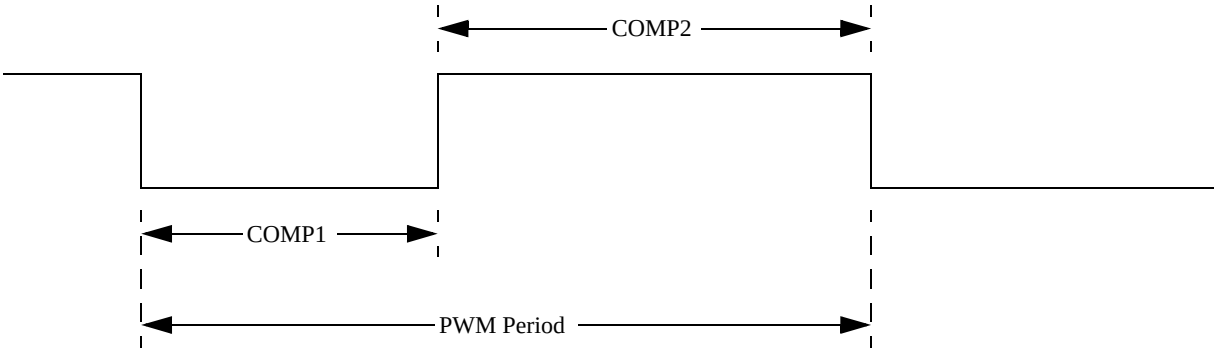


Figure 583. Variable PWM waveform

27.7.2.13 Usage of compare registers

The dual compare registers (COMP1 and COMP2) provide a bidirectional modulo count capability.

The COMP1 register should be set to the desired maximum count value or 0xFFFF to indicate the maximum unsigned value prior to roll-over, and the COMP2 register should be set to the minimum count value or 0x0000 to indicate the minimum unsigned value prior to roll-under.

When the output mode is set to 0100, the OFLAG will toggle while using alternating compare registers. In this variable frequency PWM mode, the COMP2 value defines the desired pulse width of the on time, and the COMP1 register defines the off time. COMP1 is used when OFLAG == 0 and COMP2 is used when OFLAG == 1.

Use caution when changing COMP1 and COMP2 while the counter is active. If the counter has already passed the new value, it will count to 0xFFFF or 0x0000, roll over, then begin counting toward the new value. The check is: CNTR = COMPx, *not* CNTR > COMP1 or CNTR < COMP2.

Using the CMPLD1 and CMPLD2 registers to preload compare values helps to minimize this problem.

27.7.2.14 Usage of Compare Load registers

The CMPLD1, CMPLD2, and CCCTRL registers offer a high degree of flexibility for loading compare registers with user-defined values on different compare events. To ensure correct functionality while using these registers, the following methods are recommended.

The purpose of the compare load feature is to allow quicker updating of the compare registers. A compare register can be updated using interrupts. However, because of the latency between an interrupt event occurring and the service of that interrupt, there is the possibility that the counter may have already counted past the new compare value by the

time the compare register is updated by the interrupt service routine. The counter would then continue counting until it rolled over and reached the new compare value.

To address this, the compare registers are updated in hardware in the same way the counter register is reinitialized to the value stored in the LOAD register. The compare load feature allows the user to calculate new compare values and store them in to the comparator load registers. When a compare event occurs, the new compare values in the comparator load registers are written to the compare registers eliminating the use of software to do this.

The compare load feature is intended to be used in variable frequency PWM mode. The COMP1 register determines the pulse width for the logic low part of OFLAG and COMP2 determines the pulse width for the logic high part of OFLAG. The period of the waveform is determined by the COMP1 and COMP2 values and the frequency of the primary clock source. See *Figure 583*.

27.7.2.15 MODULO COUNTING mode

To create a modulo counter using COMP1 and COMP2 as the counter boundaries (instead of 0x0000 and 0xFFFF), set the registers in the following manner. Set CNTMODE to either 100 (quadrature count mode) or 101 (count with direction mode). Use count through roll-over (LENGTH = 0) and continuous count (ONCE = 0). Set COMP1 and CMPLD1 to the upper boundary value. Set COMP2 and CMPLD2 to the lower boundary value. Set CMPMODE = 10 (COMP1 is used when counting up and COMP2 is used when counting down). Set CLC2 = 110 (load CNTR with value of CMPLD2 on COMP1 compare) and CLC1 = 111 (load CNTR with value of CMPLD1 on COMP2 compare).

27.7.3 Other features

27.7.3.1 Redundant OFLAG checking

This mode allows the user to bundle two timer functions generating any pattern to compare their resulting OFLAG behaviors (output signal).

The redundant mode is used to support online checks for functional safety reasons. Whenever a mismatch between the two adjacent channels occurs, it is reported via an interrupt to the core and the two outputs are put into their inactive states. An error is flagged via the RCF flag.

This feature can be tested by forcing a transition on one of the OFLAGs using the VAL and FORCE bits of the channel.

27.7.3.2 Loopback checking

This mode is always available in that one channel can generate an OFLAG while another channel uses the first channels' OFLAG as its input to be measured and verified to be as expected.

27.7.3.3 Input capture mode

Input capture measures pulse width (by capturing the counter value on two successive input edges) or waveform period (by capturing the counter value on two consecutive rising edges or two consecutive falling edges). The capture registers store a copy of the counter's value when an input edge (positive, negative, or both) is detected. The type of edge to be

captured by each circuit is determined by the CPT1MODE and CPT2MODE bits whose functionality is shown in *Figure 572*.

The arming logic controls the operation of the capture circuits to allow captures to be performed in a free-running (continuous) or one-shot fashion. In free-running mode, the capture sequences will be performed indefinitely. If both capture circuits are enabled, they will work together in a ping-pong style where a capture event from one circuit leads to the arming of the other and vice versa. In one-shot mode, only one capture sequence will be performed. If both capture circuits are enabled, capture circuit 0 is armed first. When a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are capable of generating an interrupt to the CPU.

27.7.3.4 Master/Slave mode

Any timer channel can be assigned as a Master (MSTR = 1). A Master's compare signal can be broadcast to the other channels within the module. The other counters can be configured to reinitialize their counters (COINIT = 1) and/or force their OFLAG output signals (COFRC = 1) to predetermined values when a Master counter compare event occurs.

27.7.3.5 Watchdog timer

The watchdog timer monitors for a stalled count when channel 0 is in quadrature count mode. When the watchdog is enabled, it loads the time-out value into a down counter. The down counter counts as long as channel 0 is in quadrature decode count mode. If this down counter reaches 0, an interrupt is asserted. The down counter is reloaded to the time-out value each time the counter value from channel 0 changes. If the channel 0 count value is toggling between two values (indicating a possibly stalled encoder), then the down counter is not reloaded.

27.8 Clocks

The eTimer module implements a protocol clock running at a frequency ≥ 120 MHz.

27.9 Interrupts

Each of the channels within the eTimer can generate an interrupt from several sources. The watchdog also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt.

Table 520. Interrupt summary

Core Interrupt	Interrupt Flag	Interrupt Enable	Name	Description
eTimer_0				
Channels 0–5	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter roll-over or roll-under
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2
Watchdog	WDF	WDFIE	Watchdog time-out interrupt	Watchdog has timed out
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel
eTimer_1				
Channels 0–5	TCF	TCFIE	Compare interrupt	Compare of counter and related compare register
	TCF1	TCF1IE	Compare 1 interrupt	Compare of the counter and COMP1 register
	TCF2	TCF2IE	Compare 2 interrupt	Compare of the counter and COMP2 register
	TOF	TOFIE	Overflow interrupt	Generated on counter roll-over or roll-under
	IELF	IELFIE	Input Low Edge interrupt	Falling edge of the secondary input signal
	IEHF	IEHFIE	Input High Edge interrupt	Rising edge of the secondary input signal
	ICF1	ICF1IE	Input Capture 1 interrupt	Input capture event for CAPT1
	ICF2	ICF2IE	Input Capture 2 interrupt	Input capture event for CAPT2
Redundant Channel Checking	RCF	RCFIE	Redundant Channel Fault interrupt	Miscompare with redundant channel

27.10 DMA

Table 521. DMA summary

DMA Request	DMA Enable	Name	Description
Channels 0–5	ICF1DE	CAPT1 read request	CAPT1 contains a value
	ICF2DE	CAPT2 read request	CAPT2 contains a value
	CMPLD1DE	CMPLD1 write request	CMPLD1 needs an update
	CMPLD2DE	CMPLD2 write request	CMPLD2 needs an update

28 Functional Safety

28.1 Introduction

This chapter describes the following modules that help add reliability to the SPC560B54/6x.

- Register protection module
- Software watchdog timer (SWT)

28.2 Register protection module

28.2.1 Overview

The register protection module offers a mechanism to protect defined memory-mapped address locations in a module under protection from being written. The address locations that can be protected are module-specific.

The register protection module is located between the module under protection and the PBRIDGE. This is shown in *Figure 584*.

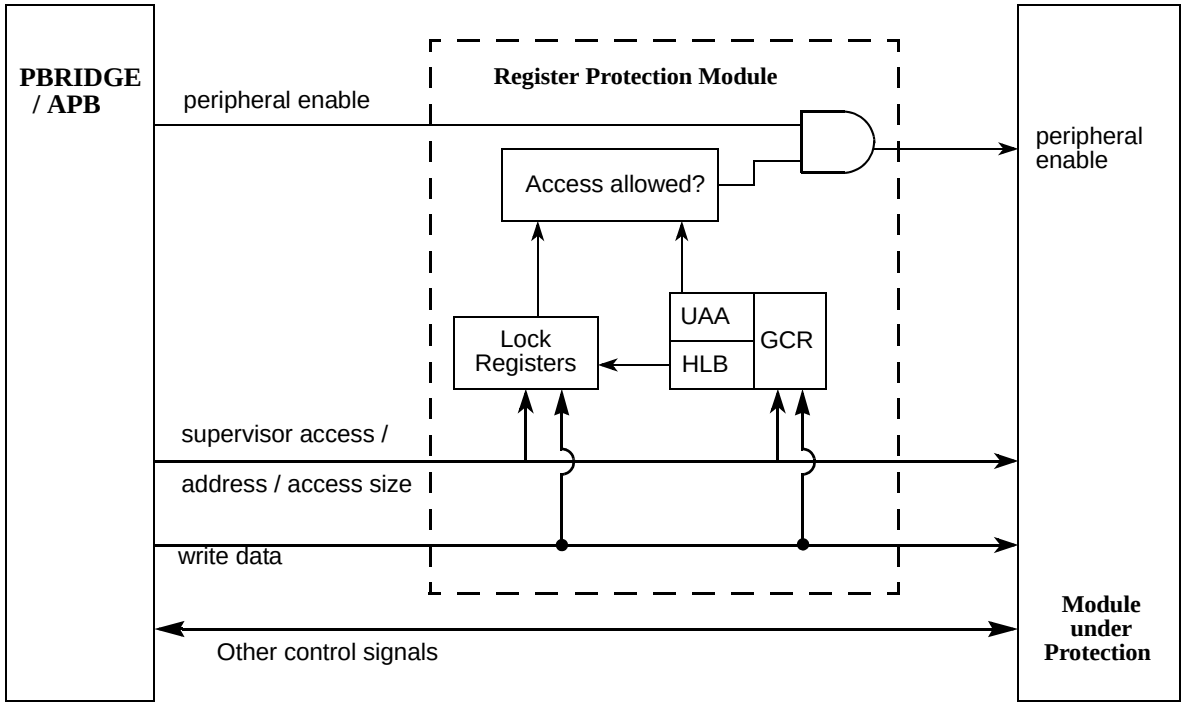


Figure 584. Register protection module block diagram

28.2.2 Features

The register protection module includes these features:

- Restrict write accesses for the module under protection to supervisor mode only
- Lock registers for first 6 KB of memory-mapped address space
- Address mirror automatically sets corresponding lock bit
- Once configured lock bits can be protected from changes

28.2.3 Modes of operation

The register protection module is operable when the module under protection is operable.

28.2.4 External signal description

There are no external signals.

28.2.5 Memory map and registers description

This section provides a detailed description of the memory map of a module using the Register protection. The original 16 KB module memory space is divided into five areas as shown in *Figure 585*.

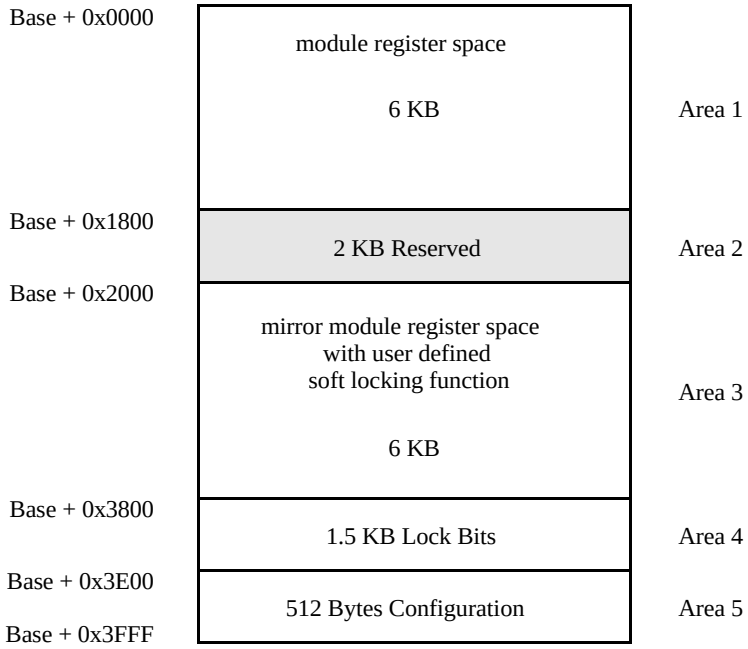


Figure 585. Register protection memory diagram

Area 1 is 6 KB and holds the normal functional module registers and is transparent for all read/write operations.

Area 2, 2 KB starting at address 0x1800, is reserved.

Area 3 is 6 KB, starting at address 0x2000 and is a mirror of area 1. A read/write access to these 0x2000 + X addresses will read/write the register at address X. As a side effect, a write access to address 0x2000 + X will set the optional Soft Lock Bits for this address X in the same cycle as the register at address X is written. Not all registers in area 1 need to have protection defined by associated Soft Lock Bits. For unprotected registers at address Y, accesses to address 0x2000 + Y will be identical to accesses at address Y. Only for registers implemented in area 1 and defined as protectable Soft Lock Bits will be available in area 4.

Area 4 is 1.5 KB and holds the Soft Lock Bits, one bit per byte in area 1. The four Soft Lock Bits associated with one module register word are arranged at byte boundaries in the memory map. The Soft Lock Bit registers can be directly written using a bit mask.

Area 5 is 512 bytes and holds the configuration bits of the protection mode. There is one configuration hard lock bit per module that prevents all further modifications to the Soft Lock Bits and can only be cleared by a system reset once set. The other bits, if set, will allow user access to the protected module.

If any locked byte is accessed with a write transaction, a transfer error will be issued to the system and the write transaction will not be executed. This is true even if not all accessed bytes are locked.

Accessing unimplemented 32-bit registers in areas 4 and 5 will result in a transfer error.

28.2.5.1 Register protection memory map

Table 522 shows the registers in the Safety Port.

Table 522. Register protection memory map

Offset from REG_ PROT_BASE (0xFFFE_8000)	Register	Location
0x0000–0x17FF	Module Register 0 (MR0)–Module Register 6143(MR6143)	*on page 921*
0x1800–0x1FFF	Reserved	
0x2000–0x37FF	Module Register 0 (MR0) + Set Soft Lock Bit 0 (LMR0)–Module Register 6143 (MR6143) + Set Soft Lock Bit 6143 (LMR6143)	*on page 921*
0x3800–0x3DFF	Soft Lock Bit Register 0 (SLBR0): Soft Lock Bits 0:3–Soft Lock Bit Register 1535 (SLBR1535): Soft Lock Bits 6140:6143	*on page 921*
0x3E00–0x3FFB	Reserved	
0x3FFC	Global Configuration Register (GCR)	*on page 922*

Note: Reserved registers in area #2 will be handled according to the protected IP (module under protection).

28.2.5.2 Registers description

This section describes in address order all the register protection registers. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

28.2.5.2.1 Module registers (MR0–6143)

This is the lower 6 KB module memory space that holds all the functional registers of the module that is protected by the register protection module.

28.2.5.2.2 Module Register and Set Soft Lock Bit (LMR0–6143)

This is memory area #3 that provides mirrored access to the MR0–6143 registers with the side effect of setting Soft Lock Bits in case of a write access to a MR that is defined as protectable by the locking mechanism. Each MR is protectable by one associated bit in a SLBR*n*[SLB*m*], according to the mapping described in *Table 523*.

28.2.5.2.3 Soft Lock Bit Register (SLBR0–1535)

These registers hold the Soft Lock Bits for the protected registers in memory area #1.

Address: Base + 0x3800–0x3DFF Access: User read-only; Supervisor read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	SLB0	SLB1	SLB2	SLB3
W	WE0	WE1	WE2	WE3				
Reset	0	0	0	0	0	0	0	0

Figure 586. Soft Lock Bit Register (SLBR*n*)

Table 523. SLBR*n* field descriptions

Field	Description
WE0 WE1 WE2 WE3	Write Enable Bits for Soft Lock Bits (SLB): WE0 enables writing to SLB0. WE1 enables writing to SLB1. WE2 enables writing to SLB2. WE3 enables writing to SLB3. 0 SLB is not modified. 1 Value is written to SLB.
SLB0 SLB1 SLB2 SLB3	Soft Lock Bits for one MR*n* register: SLB0 can block accesses to MR[(*n* × 4) + 0] SLB1 can block accesses to MR[(*n* × 4) + 1] SLB2 can block accesses to MR[(*n* × 4) + 2] SLB3 can block accesses to MR[(*n* × 4) + 3] 0 Associated MR*n* byte is unprotected and writeable. 1 Associated MR*n* byte is locked against write accesses.

Table 524 gives some examples how SLBR*n*[SLB] and SLBR*n*[MR*n*] go together:

Table 524. Soft Lock Bits vs. Protected Address

Soft Lock Bit	Protected address
SLBR0[SLB0]	MR0
SLBR0[SLB1]	MR1
SLBR0[SLB2]	MR2
SLBR0[SLB3]	MR3

Table 524. Soft Lock Bits vs. Protected Address(Continued)

Soft Lock Bit	Protected address
SLBR1[SLB0]	MR4
SLBR1[SLB1]	MR5
SLBR1[SLB2]	MR6
SLBR1[SLB3]	MR7
SLBR2[SLB0]	MR8
...	...

28.2.5.2.4 Global Configuration Register (GCR)

The Global Configuration Register (GCR) controls global configurations related to register protection.

Address: Base + 0x3FFC Access: Read Always; Supervisor write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HLB	0	0	0	0	0	0	0	UAA	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 587. Global Configuration Register (GCR)

Table 525. GCR field descriptions

Field	Description
HLB	Hard Lock Bit This register cannot be cleared once it is set by software. It can only be cleared by a system reset. 0 All SLB bits are accessible and can be modified. 1 All SLB bits are write protected and can not be modified.
UAA	User Access Allowed 0 The registers in the module under protection can only be written in supervisor mode. All write accesses in non-supervisor mode are not executed and a transfer error is issued. This access restriction is in addition to any access restrictions imposed by the protected IP module. 1 The registers in the module under protection can be accessed in the mode defined for the module registers without any additional restrictions.

Note: The GCR[UAA] bit has no effect on the allowed access modes for the registers in the Register protection module.

28.2.6 Functional description

28.2.6.1 General

This module provides a generic register (address) write-protection mechanism. The protection size can be:

- 32-bit (address == multiples of 4)
- 16-bit (address == multiples of 2)
- 8-bit (address == multiples of 1)
- unprotected (address == multiples of 1)

For all addresses that are protected there are SLBR*n*[SLB*m*] bits that specify whether the address is locked. When an address is locked it can only be read but not written in any mode (supervisor/normal). If an address is unprotected the corresponding SLBR*n*[SLB*m*] bit is always 0b0 no matter what software is writing to.

28.2.6.2 Change lock settings

To change the setting whether an address is locked or unlocked, the corresponding SLBR*n*[SLB*m*] bit needs to be changed. This can be done using the following methods:

- Modify the SLBR*n*[SLB*m*] bit directly by writing to area #4
- Set the SLBR*n*[SLB*m*] bit(s) by writing to the mirror module space (area #3)

Both methods are explained in the following sections.

28.2.6.2.1 Change lock settings directly via area #4

In memory area #4 the lock bits are located. They can be modified by writing to them. Each SLBR*n*[SLB*m*] bit has a corresponding SLBR*n*[WE*m*] mask bit, which protects it from being modified. This masking makes clear-modify-write operations unnecessary.

Figure 588 shows two modification examples. In the left example there is a write access to the SLBR*n* register specifying a mask value that allows modification of all SLBR*n*[SLB*m*] bits. The example on the right specifies a mask that only allows modification of the bits SLBR*n*[SLB[3:1]].

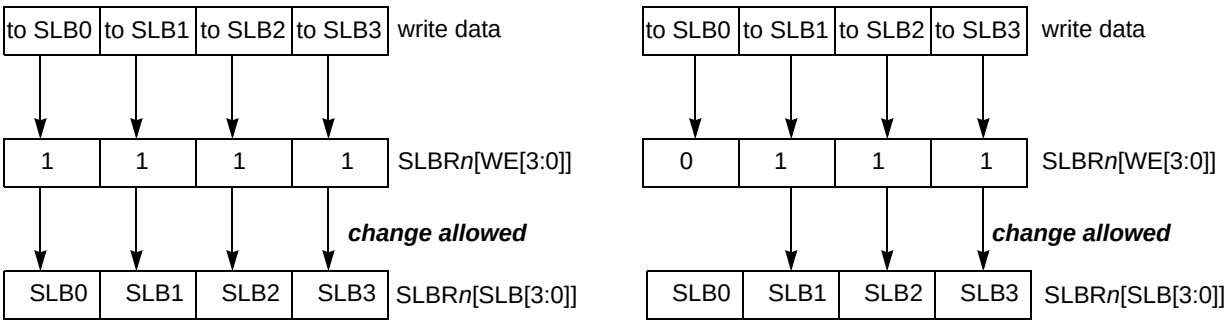


Figure 588. Change lock settings directly via area #4

Figure 588 showed four registers that can be protected 8-bit wise. In *Figure 589* registers with 16-bit protection and in *Figure 590* registers with 32-bit protection are shown.

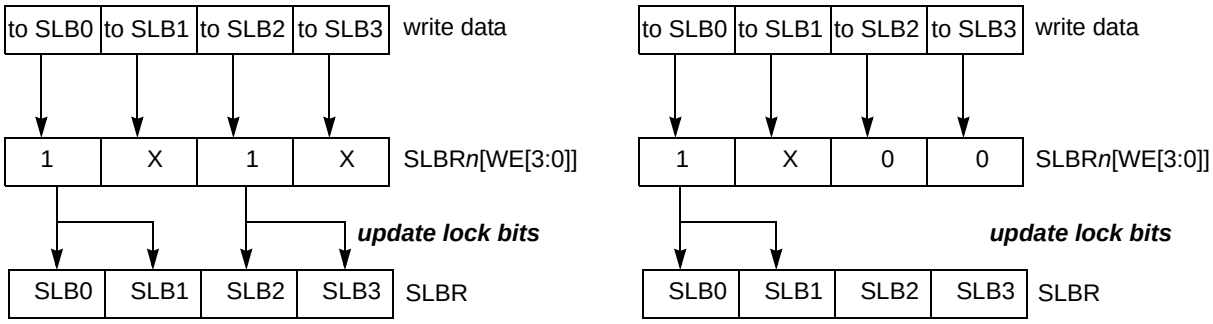


Figure 589. Change lock settings for 16-bit protected addresses

On the right side of *Figure 589* it is shown that the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB1] also. This is done as the address reflected by SLBR*n*[SLB0] is protected 16-bit wise. Note that in this case the write enable SLBR*n*[WE0] must be set while SLBR*n*[WE1] does not matter. As the enable bits SLBR*n*[WE[3:2]] are cleared the lock bits SLBR*n*[SLB[3:2]] remain unchanged.

In the example on the left side of *Figure 589* the data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] and the data written to SLBR*n*[SLB2] is mirrored to SLBR*n*[SLB3] as for both registers the write enables are set.

In *Figure 590* a 32-bit wise protected register is shown. When SLBR*n*[WE0] is set the data written to SLBR*n*[SLB0] is automatically written to SLBR*n*[SLB[3:1]] also. Otherwise SLBR*n*[SLB[3:0]] remains unchanged.

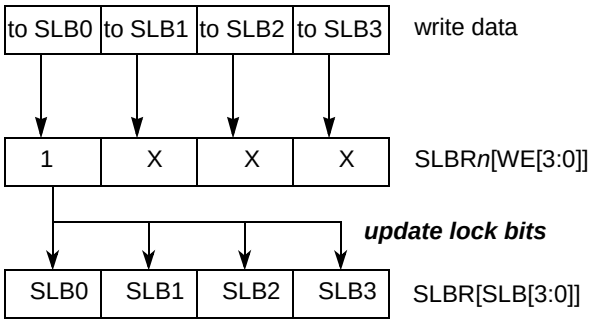


Figure 590. Change lock settings for 32-bit protected addresses

Figure 591 shows an example that has a mixed protection size configuration.

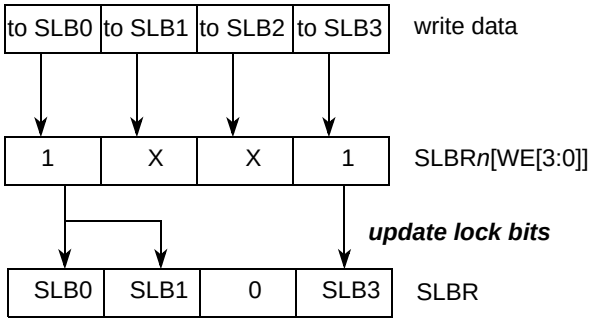


Figure 591. Change lock settings for mixed protection

The data written to SLBR*n*[SLB0] is mirrored to SLBR*n*[SLB1] as the corresponding register is 16-bit protected. The data written to SLBR*n*[SLB2] is blocked as the corresponding register is unprotected. The data written to SLBR*n*[SLB3] is written to SLBR*n*[SLB3].

28.2.6.2.2 Enable locking via mirror module space (area #3)

It is possible to enable locking for a register after writing to it. To do so the mirrored module address space must be used. *Figure 592* shows one example.

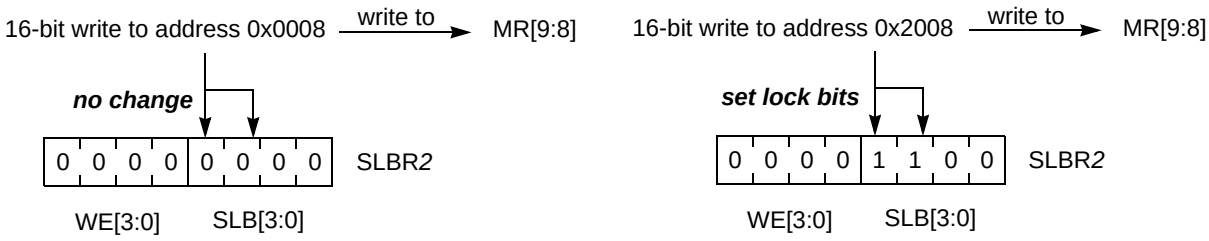


Figure 592. Enable locking via mirror module space (area #3)

When writing to address 0x0008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits remain unchanged (left part of *Figure 589*).

When writing to address 0x2008 the registers MR9 and MR8 in the protected module are updated. The corresponding lock bits SLBR2.SLB[1:0] are set while the lock bits SLBR2.SLB[3:2] remain unchanged (right part of *Figure 589*).

Figure 593 shows an example where some addresses are protected and some are not.

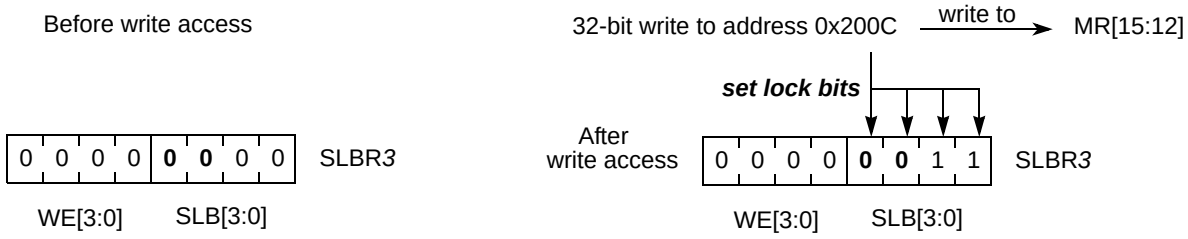


Figure 593. Enable locking for protected and unprotected addresses

In the example in *Figure 593*, addresses 0x0C and 0x0D are unprotected. Therefore their corresponding lock bits SLBR3.SLB[1:0] are always 0b0 (shown in bold). When doing a 32-bit write access to address 0x200C only lock bits SLBR3.SLB[3:2] are set while bits SLBR3.SLB[1:0] stay 0b0.

Note: Lock bits can only be set via writes to the mirror module space. Reads from the mirror module space will not change the lock bits.

28.2.6.2.3 Write protection for locking bits

Changing the locking bits through any of the procedures mentioned in *Section 28.2.6.2.1: Change lock settings directly via area #4*, and *Section 28.2.6.2.2: Enable locking via mirror module space (area #3)* is only possible as long as the GCR[HLB] bit is cleared. Once this bit is set the locking bits can no longer be modified until there was a system reset.

28.2.6.3 Access errors

The protection module generates transfer errors under several circumstances. For the area definition refer to *Figure 585*.

1. If accessing area #1 or area #3, the protection module will pass on any access error from the underlying Module under Protection.
2. If user mode is not allowed, user writes to all areas will assert a transfer error and the writes will be blocked.
3. If accessing the reserved area #2, a transfer error will be asserted.
4. If accessing unimplemented 32-bit registers in area #4 and area #5 a transfer error will be asserted.
5. If writing to a register in area #1 and area #3 with Soft Lock Bit set for any of the affected bytes a transfer error is asserted and the write will be blocked. Also the complete write operation to non-protected bytes in this word is ignored.
6. If writing to a Soft Lock Register in area #4 with the Hard Lock Bit being set a transfer error is asserted.
7. Any write operation in any access mode to area #3 while Hard Lock Bit GCR[HLB] is set

28.2.7 Reset

The reset state of each individual bit is shown in *Section 28.2.5.2: Registers description*. In summary, after reset, locking for all MR*n* registers is disabled. The registers can be accessed in Supervisor Mode only.

28.3 Software Watchdog Timer (SWT)

28.3.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT requires periodic execution of a watchdog servicing sequence. Writing the sequence resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or

hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out, a reset is always generated on a second consecutive time-out.

The SWT provides a window functionality. When this functionality is programmed, the servicing action should take place within the defined window. When occurring outside the defined period, the SWT will generate a reset.

28.3.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- The unique SWT counter clock is the undivided low power internal oscillator (IRC 16 MHz), no other clock source can be selected
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Master access protection
- Hard and soft configuration lock bits
- The SWT is started on exit of power-on phase (RGM phase 2) to monitor flash boot sequence phase. It is then reset during RGM phase 3 and optionally enabled when platform reset is released depending on value of flash user option bit 31 (WATCHDOG_EN).

28.3.3 Modes of operation

The SWT supports three device modes of operation: normal, debug and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. In stop mode, operation of the counter is controlled by the STP bit in the SWT_CR. If the STP bit is set, the counter is stopped in stop mode, otherwise it continues to run. As soon as out of stop mode, SWT will continue from the state it was before entering this mode.

Software watchdog is not available during stand-by. As soon as out of stand-by, the SWT behaves as in a usual “out of reset” situation.

28.3.4 External signal description

The SWT module does not have any external interface signals.

28.3.5 SWT memory map and registers description

The SWT programming model has six 32-bit registers, listed in *Table 526*. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the RIA bit in the SWT_CR is set, the SWT generates a system reset on an invalid access. Otherwise, a bus error is generated. If either the HLK or SLK bits in the SWT_CR are set, then the SWT_CR, SWT_TO and SWT_WN registers are read only.

Table 526. SWT memory map

Offset from SWT_BASE 0xFFF3_8000	Register	Location
0x0000	SWT_CR—SWT Control Register	*on page 928*
0x0004	SWT_IR—SWT Interrupt Register	*on page 929*
0x0008	SWT_TO—SWT Time-Out register	*on page 930*
0x000C	SWT_WN—SWT Window Register	*on page 931*
0x0010	SWT_SR—SWT Service Register	*on page 931*
0x0014	SWT_CO—SWT Counter Output register	*on page 932*
0x0018	SWT_SK—SWT Service Key register	*on page 932*
0x001C–0x03FF	Reserved	

28.3.5.1 SWT Control Register (SWT_CR)

The SWT_CR contains fields for configuring and controlling the SWT. The reset value of this register is device specific. Some devices can be configured to automatically clear the SWT_CR[WEN] bit during the boot process. This register is read-only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP 0	MAP 1	MAP 2	MAP 3	MAP 4	MAP 5	MAP 6	MAP 7	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	STP	FRZ	WEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	1	1	0	1	1

Figure 594. SWT Control Register (SWT_CR)

The default reset value for SWT_CR is 0xFF00_011B, corresponding to MAPn = 1 (all master access allowed), RIA = 1 (reset on invalid SWT access), SLK = 1 (soft lock), CSL = 1 (IRC clock source for counter), FRZ = 1 (freeze on debug), WEN = 1 (watchdog enable). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is 0.

Table 527. SWT_CR field descriptions

Field	Description
MAP*n*	Master Access Protection for Master *n*. The platform bus master assignments are device specific. 0 Access for the master is disabled. 1 Access for the master is enabled. **Note:** Master n refers to "Logical Master ID" (please consult *Table 120*).
KEY	Keyed Service Mode 0 Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog. 1 Keyed Service Mode, two pseudorandom key values are used to service the watchdog.
RIA	Reset on Invalid Access 0 Invalid access to the SWT generates a bus error. 1 Invalid access to the SWT causes a system reset if WEN = 1.
WND	Window Mode 0 Regular mode, service sequence can be done at any time. 1 Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
ITR	Interrupt Then Reset 0 Generate a reset on a time-out. 1 Generate an interrupt on an initial time-out, reset on a second consecutive time-out.
HLK	Hard Lock This bit is only cleared at reset. 0 SWT_CR, SWT_TO and SWT_WN are read/write registers if SLK = 0. 1 SWT_CR, SWT_TO and SWT_WN are read only registers.
SLK	Soft Lock This bit is cleared by writing the unlock sequence to the service register. 0 SWT_CR, SWT_TO and SWT_WN are read/write registers if HLK = 0. 1 SWT_CR, SWT_TO and SWT_WN are read only registers.
CSL	Clock Selection Selects the internal 16 MHz IRC oscillator clock that drives the internal timer. CSL bit can be written.The status of the bit has no effect on counter clock selection on the device. 0 System clock. (Not applicable in SPC560B54/6x). 1 Oscillator clock.
STP	Stop Mode Control Allows the watchdog timer to be stopped when the device enters stop mode. 0 SWT counter continues to run in stop mode. 1 SWT counter is stopped in stop mode.
FRZ	Debug Mode Control Allows the watchdog timer to be stopped when the device enters debug mode. 0 SWT counter continues to run in debug mode. 1 SWT counter is stopped in debug mode.
WEN	Watchdog Enabled 0 SWT is disabled. 1 SWT is enabled.

28.3.5.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 595. SWT Interrupt Register (SWT_IR)

Table 528. SWT_IR field descriptions

Field	Description
TIF	Time-out Interrupt Flag The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to an initial time-out.

28.3.5.3 SWT Time-Out register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. The reset value for this register is device specific. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WTO															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WTO															
W																
Reset	1	0	1	0	1	0	0	1	1	0	0	0	0	0	0	0

Figure 596. SWT Time-Out register (SWT_TO)

The default counter value (SWT_TO_RST) is 0x0003_A980, which corresponds to approximately 15 ms with the 16 MHz IRC clock.

Table 529. SWT_TO field descriptions

Field	Description
WTO	Watchdog time-out period in clock cycles An internal 32-bit down counter is loaded with this value or 0x0100, whichever is greater when the service sequence is written or when the SWT is enabled.

28.3.5.4 SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WST															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WST															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 597. SWT Window register (SWT_WN)

Table 530. SWT_WN field descriptions

Field	Description
WST	Window start value When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

28.3.5.5 SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service sequence writes used to reset the watchdog timer.

Address: Base + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WSC															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 598. SWT Service Register (SWT_SR)

Table 531. SWT_SR field descriptions

Field	Description
WSC	Watchdog Service Code This field services the watchdog and clears the SWT_CR[SLK] soft lock bit. To service the watchdog, the value 0xA602 followed by 0xB480 is written to the WSC field. To clear the SWT_CR[SLK] soft lock bit, the value 0xC520 followed by 0xD928 is written to the WSC field.

28.3.5.6 SWT Counter Output register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read only register that shows the value of the internal down counter when the SWT is disabled.

Address: Base + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 599. SWT Counter Output register (SWT_CO)

Table 532. SWT_CO field descriptions

Field	Description
CNT	Watchdog Count When the watchdog is disabled (SWT_CR.[WEN]=0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for as many as 6 system plus 8 counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

28.3.5.7 SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read only if either the SWT_CR[HLK] or SWT_CR[SLK] bits are set.

Address: Base + 0x0018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SK															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 600. SWT Service Register (SWT_SR)

Table 533. SWT_SR field descriptions

Field	Description
SK	Service Key.This field is the previous (or initial) service key value used in keyed service mode. If SWT_CR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

28.3.6 Functional description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a a control register (SWT_CR), an interrupt register (SWT_IR), time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR) and a counter output register (SWT_CO).

The SWT_CR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT_CR.WEN bit. The reset value of the SWT_CR.WEN bit is device specific1 (enabled). This last bit is cleared when exiting ME RESET mode in case flash user option bit 31 (WATCHDOG_EN) is 0. If the reset value of this bit is 1, the watchdog starts operation automatically after reset is released. Some devices can be configured to clear this bit automatically during the boot process.

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x0100, in which case the time-out period is set to 0x0100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and a valid service sequence is written. The SWT_CR[CSL] bit selects which clock (system or oscillator) drives the down counter.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_CR, SWT_TO and SWT_WN registers are read only. The hard lock is enabled by setting the SWT_CR[HLK] bit, which can only be cleared by a reset. The soft lock is enabled by setting the SWT_CR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_CR.WEN bit to be set.

When enabled, the SWT requires periodic execution of the watchdog servicing sequence. The service sequence is a write of 0xA602 followed by a write of 0xB480 to the

SWT_SR[WSC] field. Writing the service sequence loads the internal down counter with the time-out period. There is no timing requirement between the two writes. The service sequence logic ignores unlock sequence writes and recognizes the 0xA602, 0xB480 sequence regardless of previous writes. Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require as many as 3 system plus 7 counter clock cycles.

If window mode is enabled (SWT_CR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_CR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be as many as 3 system plus 4 counter clock cycles.

The SWT_CR[ITR] interrupt then reset bit controls the action taken when a time-out occurs. If the SWT_CR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_CR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for as many as 6 system plus 8 counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_CR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

29 Fault Collection Unit (FCU)

29.1 Introduction

The Fault Collection Unit (FCU) module provides functional safety to the device.

29.1.1 Overview

The FCU provides a central capability to collect faults reported by the individual modules of the device. It represents the minimum blocking unit to develop a coherent safety strategy for the chassis family. Selected critical faults are reported to the external device via output pins, if no recovery is provided by the device. The operation of the FCU is independent from the CPU. The FCU provides an independent fault reporting mechanism even in case the CPU behavior is abnormal. The FCU always starts up in init mode. As long as the FCU remains in init mode, testing of the FCU logic (for dormant fault detection) can be performed under software control.

The FCU is developed to increase the level of safety of the system/MCU level.

Functional safety features of the FCU include:

- It is an independent module: If other control modules are behaving abnormally, the user can still trigger actions to prevent a critical situation.
- Collection and external reporting of faults occurring on the device
- Centralized fault collection
- Each fault cause can be treated in a different way
  - No action
  - Alarm—allows hardware or software to recover from fault
  - Fault—communicates directly to an external device that something went wrong
- Three different output protocols available
- Possible to inject fake faults on user request during initialization phase to test the peripheral (dormant fault detection)

29.1.1.1 General description

The FCU is logically divided in three blocks:

- Input unit—captures faults reported from the device
- Control unit—implemented by a finite state machine (see *Figure 615* for details)
- Output unit—generates output signals

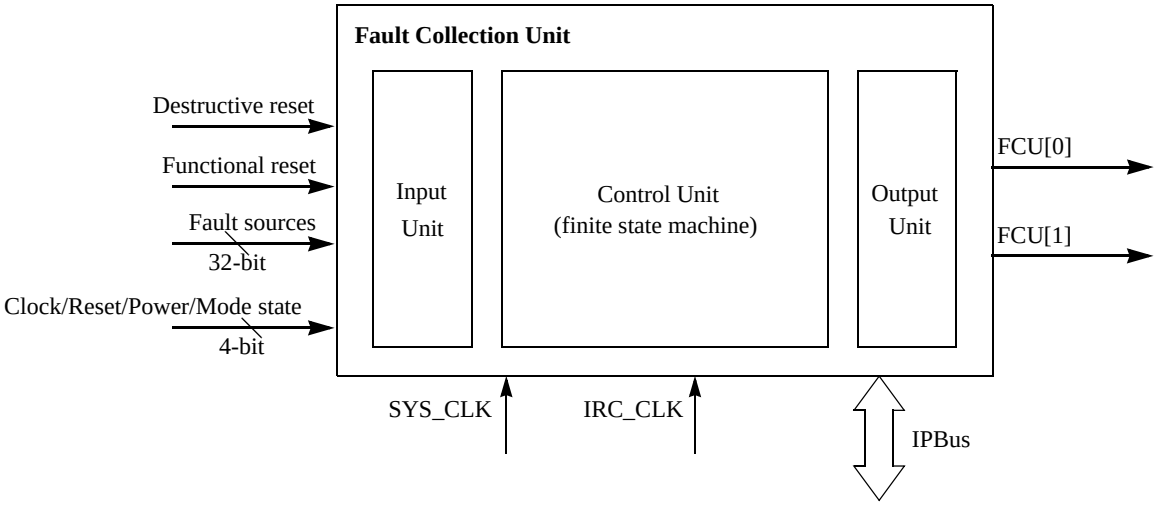


Figure 601. Fault Collection Unit (FCU) block diagram

Figure 602 shows the flow chart of FCU fault handling.

The FCU module and its state machine run on the system clock, making the two modules synchronous. The high speed RC clock (16 MHz) is used only in the Alarm state, in order to compute a deterministic timeout.

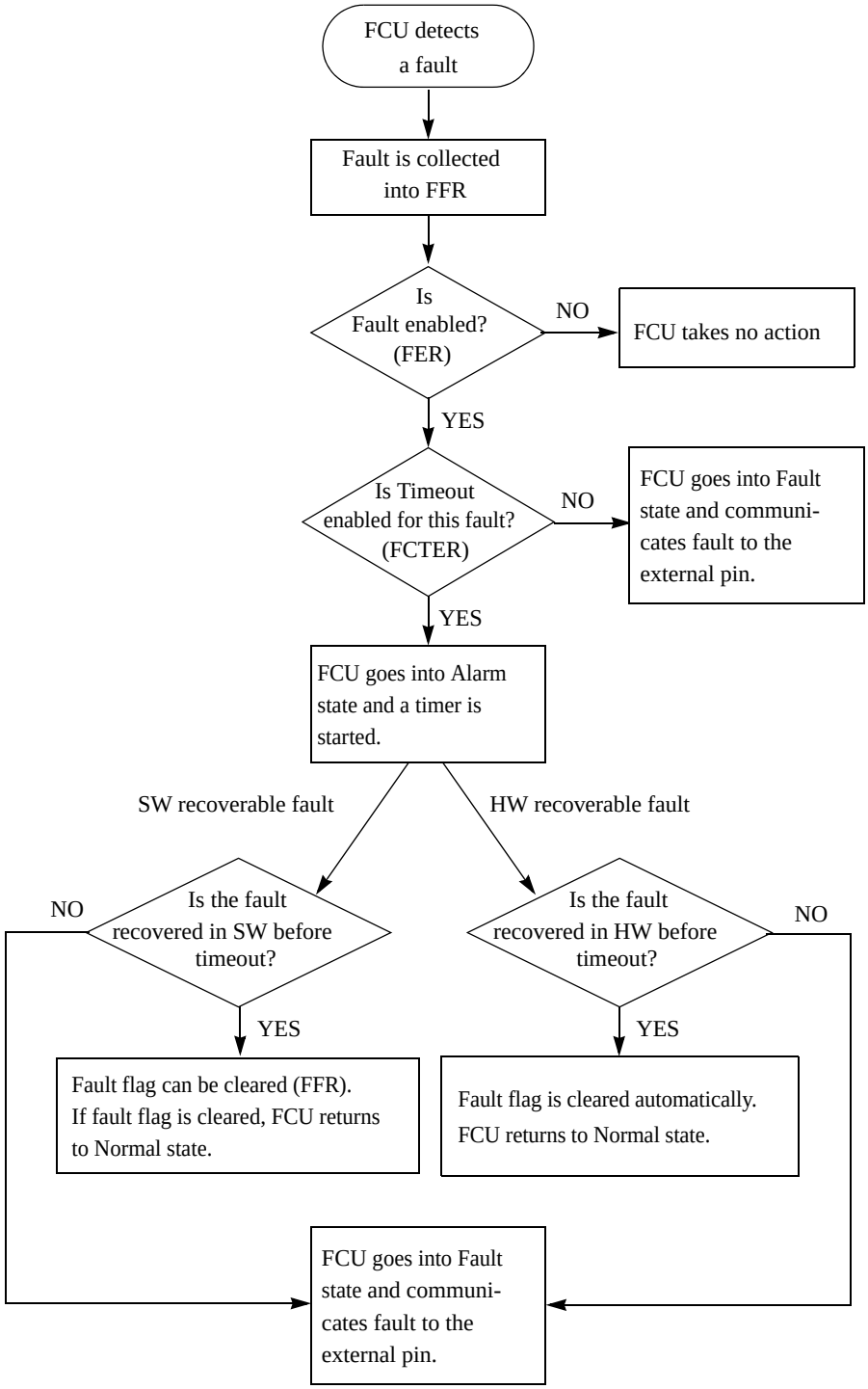


Figure 602. FCU fault handling

29.1.2 Features

The FCU includes the following features:

- Collection of critical faults
- Reporting of selected critical faults to external pins
- Fault flag status kept over non-destructive reset for later analysis (in a “Freeze” register)
- Continuous and synchronous latch of MC state
- MC state kept over non-destructive reset for later analysis (in a “Freeze” register)
- 4-state (Init, Normal, Alarm, Fault) finite state machine
- Different actions can be taken depending on fault type
- Selectable protocols for fault signal indication in Fault state (dual-rail, time-switching, bi-stable)
- Programmable clock prescaler for time-switching output signal generation
- Protection mechanism to avoid unwanted clearing of fault flags
- Internal logic testing, by using a fake fault generator during initialization phase

29.1.3 Modes of operation

This section describes the basic functional modes of the FCU module.

29.1.3.1 Normal mode

In Normal operation, the FCU captures all the faults in real time and processes them according to the fault type and the configuration set by the user.

29.1.3.2 Test mode

Test mode provides a testing mechanism of the FCU (for dormant fault detection). Test mode can be entered during the configuration (Init) phase. The user can write to the Fake Fault Generation Register (FCU_FFGR) and can check the behavior while staying in Test mode. During Test mode, the state machine behaves normally. In Test mode, real faults are not detected.

The user can inject fake faults by writing to the FCU_FFGR during Test mode to test the peripheral. Fault flags are cleared when Test mode is exited. Asynchronous software faults written to the FCU_FFGR during Init phase will not be latched. In order to test software faults, the FCU_FFGR needs to be cleared during Init phase and written to during Normal phase.

29.2 Memory map and register definition

The following sections define the FCU memory map, register layout and functionality.

29.2.1 Memory map

Table 534. FCU memory map

Offset from FCU_BASE (0xFFE6_C000)	Register	Location
0x0000	Module Configuration Register (FCU_MCR)	*on page 941*
0x0004	Fault Flag Register (FCU_FFR)	*on page 942*
0x0008	Frozen Fault Flag Register (FCU_FFFR)	*on page 944*
0x000C	Fake Fault Generation Register (FCU_FFGR)	*on page 945*
0x0010	Fault Enable Register (FCU_FER)	*on page 946*
0x0014	Key Register (FCU_KR)	*on page 946*
0x0018	Timeout Register (FCU_TR)	*on page 947*
0x001C	Timeout Enable Register (FCU_TER)	*on page 947*
0x0020	Module State Register (FCU_MSR)	*on page 948*
0x0024	Microcontroller State Register (FCU_MCSR)	*on page 948*
0x0028	Frozen MC State Register (FCU_FMCSR)	*on page 950*
0x002C–0x3FFF	Reserved	

29.2.2 Register summary

Table 535 shows the register summary.

Table 535. Register summary

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0000_0000 FCU_MCR	R	MCL	TM[1:0]		0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	PS[1:0]		FOM[1:0]		FOP[5:0]					
	W																
0x0000_0004 FCU_FFR	R	SRF 0	0	SRF 2	SRF 3	SRF 4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	HRF 15	HRF 14	HRF 13	HRF 12	HRF 11	HRF 10	HRF 9	HRF 8	HRF 7	HRF 6	HRF 5	HRF 4	HRF 3	HRF 2	HRF 1	HRF 0
	W																

Table 535. Register summary(Continued)

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0000_0008 FCU_FFFR	R	FRSRF0	0	FRSRF2	FRSRF3	FRSRF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	FRHRF15	FRHRF14	FRHRF13	FRHRF12	FRHRF11	FRHRF10	FRHRF9	FRHRF8	FRHRF7	FRHRF6	FRHRF5	FRHRF4	FRHRF3	FRHRF2	FRHRF1	FRHRF0
	W																
0x0000_000C FCU_FFGR	R	FSRF0	F1RF3	FSRF2	FSRF3	FSRF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	FHRF15	FHRF14	FHRF13	FHRF12	FHRF11	FHRF10	FHRF9	FHRF8	FHRF7	FHRF6	FHRF5	FHRF4	FHRF3	FHRF2	FHRF1	FHRF0
	W																
0x0000_0010 FCU_FER	R	ESF0	ESF1	ESF2	ESF3	ESF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	EHF15	EHF14	EHF13	EHF12	EHF11	EHF10	EHF9	EHF8	EHF7	EHF6	EHF5	EHF4	EHF3	EHF2	EHF1	EHF0
	W																
0x0000_0014 FCU_KR	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
0x0000_0018 FCU_TR	R	TR[31:16]															
	W																
	R	TR[15:0]															
	W																
0x0000_001C FCU_TER	R	TESF0	TESF1	TESF2	TESF3	TESF4	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	TEHF15	TEHF14	TEHF13	TEHF12	TEHF11	TEHF10	TEHF9	TEHF8	TEHF7	TEHF6	TEHF5	TEHF4	TEHF3	TEHF2	TEHF1	TEHF0
	W																
0x0000_0020 FCU_MSR	R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	S0	S1	S2	S3
	W																

Table 535. Register summary(Continued)

Name		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0x0000_0024 FCU_MCSR	R	0	0	0	0	0	0	0	0	0	0	0	0	MCPS[3:0]			
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	MCAS[3:0]			
	W																
0x0000_0028 FCU_FMCSR	R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCPS[3:0]			
	W																
	R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCAS[3:0]			
	W																

29.2.3 Register descriptions

29.2.3.1 Module Configuration Register (FCU_MCR)

The FCU_MCR does the following:

- Locks the configuration and lets the FCU go into Normal behavior state
- Enters Test mode (only possible during the Init state) but can be exited during any phase
- Configures protocol, prescaler, and polarity for FCU output signals (only possible before configuration is locked)

In Test mode, the FCU_FFGR can be accessed to emulate software/hardware faults. Fake faults can be generated only when Test mode is entered.

In Test mode, output pin(s) can be enabled or disabled, depending on the value of field TM[1:0]. To exit from Test mode, field TM must be written either ‘00’ or ‘11’. While exiting the Test mode, the FCU must return to the Init state and automatically clear all the fault flags.

Address: 0x0000 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MCL	TM[1:0]		0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	—

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	PS[1:0]		FOM[1:0]		FOP[5:0]					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 603. Module Configuration Register (FCU_MCR)

Table 536. FCU_MCR field description

Field	Description
0 MCL	Module Configuration Lock 0: Configuration not locked, FCU remains in Init state 1: Configuration locked, FCU moves to Normal state
1-2 TM[1:0]	Test Mode 00: Test Mode not entered 01: Test Mode entered (fake faults can be generated), output pins disabled 10: Test Mode entered (fake faults can be generated), output pins enabled 11: Test Mode not entered
22-23 PS[1:0]	Polarity select 00: FCU[0] has normal polarity, FCU[1] has normal polarity. 01: FCU[0] has inverted polarity, FCU[1] has normal polarity. 10: FCU[0] has normal polarity, FCU[1] has inverted polarity. 11: FCU[0] has inverted polarity, FCU[1] has inverted polarity.
24-25 FOM[1:0]	Fault output mode selection 00: Dual-Rail (default state) 01: Time Switching 10: Bi-Stable 11: Reserved
26-31 FOP[5:0]	Fault Output Prescaler 000000: Input clock frequency is divided by [4096 × (0 + 1) × 2], where 4096 is a fixed prescaler. 000001: Input clock frequency is divided by [4096 × (1 + 1) × 2], where 4096 is a fixed prescaler. 000010: Input clock frequency is divided by [4096 × (2 + 1) × 2], where 4096 is a fixed prescaler. 000011: Input clock frequency is divided by [4096 × (3 + 1) × 2], where 4096 is a fixed prescaler. 000100: Input clock frequency is divided by [4096 × (4 + 1) × 2], where 4096 is a fixed prescaler. ... Default at reset is 0x07 (input clock frequency is divided by [4096 × (7 + 1) × 2]).

29.2.3.2 Fault Flag Register (FCU_FFR)

The FCU_FFR contains the latched fault indication coming from the device. The FCU reacts on faults only if the respective enable bit for a fault is set in the Fault Enable Register (FCU_FER). In this case, the Alarm or Fault state is entered, depending on the user's selection. To enter Alarm state, the bit for the fault has to be set in the Timeout Enable Register (FCU_TER), otherwise Fault state is entered and output pins are communicating the internal fault.

No faults are latched in Init state (refer to state machine *Figure 615*).

The FCU_FFR is copied into Frozen Fault Flag Register (FCU_FFFR) only when the FCU goes into Fault state.

Each single flag can be cleared:

- In hardware, if the flag disappears due to hardware intervention. Bits 16:31 of the FCU_FFR store hardware-recoverable flags.
- In software, if the user application can recover from a faulty situation. Bits 0:4 of the FCU_FFR store software-recoverable flags.

Notice that software recoverable fault flags must be kept high also if the relative signal does not show anymore a fault. Hardware recoverable fault flags are updated in real time. Reset requests are assumed as hardware-recoverable.

In order to clear a flag in the FCU_FFR via software, the application should access first time the Key Register (FCU_KR) by writing the value of a key (0x618B_7A50) second time resetting the appropriate flag. The following errors are ignored:

- Wrong key inserted in FCU_KR
- Attempt to clear a hardware recoverable flag

If user software tries to clear an already cleared software-recoverable flag, SRF0 is set.

Address: Base + 0x0004 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SRF 0	0	SRF 2	SRF 3	SRF 4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HRF 15	HRF 14	HRF 13	HRF 12	HRF 11	HRF 10	HRF 9	HRF 8	HRF 7	HRF 6	HRF 5	HRF 4	HRF 3	HRF 2	HRF 1	HRF 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 604. Fault Flag Register (FCU_FFR)

In addition to the detailed field descriptions in *Table 537*, *Table 538* provides the hardware/software fault descriptions.

Table 537. FCU_FFR field descriptions

Field	Description
0 SRF0	Software Recoverable Fault [0] 0: No error latched 1: Error latched
2:4 SRF2–SRF4	Software Recoverable Fault [2:4] 0: No error latched 1: Error latched
16:31 HRF15–HRF0	Hardware Recoverable Fault [15:0] 0: No error latched 1: Error latched

Table 538. Hardware/software fault description

Label	Module	Fault type
HRF0	Core	Checkstop mode entered
HRF1	Core	z0h core reset output
HRF2	CMU_0	Loss of crystal
HRF3	FMPLL_0	Loss of lock
HRF4	CMU_0	Frequency out of range
HRF5	FMPLL_1	Loss of lock

Table 538. Hardware/software fault description(Continued)

Label	Module	Fault type
HRF6	CMU_1	Frequency out of range
HRF7	Flash	Flash Fatal Error
HRF8	SWT	SW Watchdog reset
HRF9	JTAG	JTAG reset (TAP controller)
HRF10	VREG	Comparators
HRF11	VREG	LVD 4.5
HRF12	VREG	LVD 2.7 VREG
HRF13	VREG	LVD 2.7 FLASH
HRF14	VREG	LVD 2.7 I/O
HRF15	VREG	LVD 1.2 digital
SRF0	FCU	FCU error
SRF1	FCU	Not used
SRF2	SRAM	ECC multi-bit error
SRF3	Data Flash	ECC multi-bit error
SRF4	Code Flash	ECC multi-bit error

29.2.3.3 Frozen Fault Flag Register (FCU_FFFR)

The Fault Flag Register (FCU_FFR) is copied into Frozen Fault Flag Register (FFFR) every time the FCU goes into Fault state, in order to take a snapshot of the fault flags.

The bit description of the FCU_FFFR is the same as that of the FCU_FFR. The FCU_FFR is read/clear whereas the FCU_FFFR is read-only. See *Figure 605* and *Table 539* for details.

Address: Base + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRS RF0	0	FRS RF2	FRS RF3	FRS RF4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FRH RF15	FRH RF14	FRH RF13	FRH RF12	FRH RF11	FRH RF10	FRH RF9	FRH RF8	FRH RF7	FRH RF6	FRH RF5	FRH RF4	FRH RF3	FRH RF2	FRH RF1	FRH RF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 605. Frozen Fault Flag Register (FCU_FFFR)

Table 539 provides a detailed bit description. *Table 538* provides a detailed list of recoverable faults.

Table 539. FCU_FFFR field descriptions

Field	Description
0 FRSRF0	Software Recoverable Fault 0: No error latched 1: Error latched
2:4 FRSRF2– FRSRF4	Software Recoverable Fault 0: No error latched 1: Error latched
16:31 FRHRF15 – FRHRF0	Hardware Recoverable Fault 0: No error latched 1: Error latched

29.2.3.4 Fake Fault Generation Register (FCU_FFGR)

The FCU_FFGR allows the user to emulate a software/hardware recoverable fault in order to test the FCU logic. Once a bit in the FCU_FFGR is set, the FCU should behave exactly in the same way after detecting a real fault. This register can be accessed only when Test mode is entered (check TM field in FCU_MCR). In Test mode, real faults are not detected and fake faults for SRF0 cannot be generated.

Address: Base + 0x000C Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FSRF 0	FSRF 1	FSRF 2	FSRF 3	FSRF 4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FHRF 15	FHRF 14	FHRF 13	FHRF 12	FHRF 11	FHRF 10	FHRF 9	FHRF 8	FHRF 7	FHRF 6	FHRF 5	FHRF 4	FHRF 3	FHRF 2	FHRF 1	FHRF 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 606. Fake Fault Generation Register (FCU_FFGR)

Table 540. FCU_FFGR field description

Field	Description
0:4 FSRF0– FSRF4	Fake Software Recoverable Fault[0:4] 0: No error latched 1: Error latched
16:31 FHRF15– FHRF0	Fake Hardware Recoverable Fault[15:0] 0: No error latched 1: Error latched

29.2.3.5 Fault Enable Register (FCU_FER)

When a fault occurs, the FCU goes into either Alarm or Fault state (state is selected in the FCU_TER), if the respective fault enable bit is set in the Fault Enable Register (FCU_FER). This register can be configured only during the Init phase before the configuration is locked.

Address: Base + 0x0010 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ESF 0	ESF 1	ESF 2	ESF 3	ESF 4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EHF 15	EHF 14	EHF 13	EHF 12	EHF 11	EHF 10	EHF 9	EHF 8	EHF 7	EHF 6	EHF 5	EHF 4	EHF 3	EHF 2	EHF 1	EHF 0
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 607. Fault Enable Register (FCU_FER)

Table 541. FCU_FER field descriptions

Field	Description
0:4 ESF0–ESF4	Enable Software Recoverable Fault 0: FCU takes no action on Software recoverable Fault [0:4] 1: FCU goes into Alarm/Fault state on Software recoverable Fault [0:4] Note. ESF1 not implemented or usable on this device
16:31 EHF15–EHF0	Enable Hardware Recoverable Fault 0: FCU takes no action on Hardware recoverable Fault [15:0] 1: FCU goes into Alarm/Fault state on Hardware recoverable Fault [15:0]

29.2.3.6 Key Register (FCU_KR)

In order to clear a software fault flag in the Alarm state, the FCU_KR must be set to the following value: 0x618B_7A50. Then the software fault flag can be cleared.

User software can clear the software fault flag only after the following instruction is executed. If something else is done, instead of clearing the flag, the key must be re-entered.

Any wrong key error is ignored.

Address: Base + 0x0014 Access: User read-only, Supervisor read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 608. Key Register (FCU_KR)

29.2.3.7 Timeout Register (FCU_TR)

Once the FCU goes into Alarm state, a fault can be recovered before the timeout elapses. This timeout should be long enough for hardware or software to recover from the fault. If the fault is not recovered before the timeout elapses, the FCU goes into Fault state.

Address: Base + 0x0018 Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TR[0:15]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TR[31:16]															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 609. Timeout Register (FCU_TR)

Table 542. FCU_TR field descriptions

Field	Description
0-31 TR	FCU Timeout 00000: Timeout is one clock (16 MHz) cycle 00001: Timeout is one clock (16 MHz) cycle 00002: Timeout is two clock (16 MHz) cycles 00003: Timeout is three clock (16 MHz) cycles ... Default at reset is 0x0000_FFFF. Timeout is 65,535 clock cycles (about 4.1 ms at 16 MHz, high speed RC clock)

29.2.3.8 Timeout Enable Register (FCU_TER)

Once a specific fault is enabled, the user can select to move to Alarm or Fault state when a fault occurs. A timeout enable has no effect if the related fault enable flag is not set.

Address: Base + 0x001C Access: User read/write, Supervisor read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TESF0	TESF1	TESF2	TESF3	TESF4	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TEHF15	TEHF14	TEHF13	TEHF12	TEHF11	TEHF10	TEHF9	TEHF8	TEHF7	TEHF6	TEHF5	TEHF4	TEHF3	TEHF2	TEHF1	TEHF0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 610. Timeout Enable Register (FCU_TER)

Table 543. FCU_TER field descriptions

Field	Description
0:4 TESF0– TESF4	Timeout Enable for Software Recoverable Fault 0: FCU goes into Fault state on Software recoverable Fault[0:4] 1: FCU goes into Alarm state on Software recoverable Fault[0:4] Note. TESF1 not implemented or usable on this device
16:31 TEHF15– TEHF0	Timeout Enable for Hardware Recoverable Fault 0: FCU goes into Fault state on Hardware recoverable Fault[15:0] 1: FCU goes into Alarm state on Hardware recoverable Fault[15:0]

29.2.3.9 Module State Register (FCU_MSR)

The FCU_MSR indicates the current state of the FCU. Only one of these bits can be set at a time.

Address: Base + 0x0020 Access: User read-only, Supervisor read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	S3	S2	S1	S0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 611. Module State Register (FCU_MSR)

Table 544. FCU_MSR field descriptions

Field	Description
28 S3	When this bit is set, the FCU is in the Fault state.
29 S2	When this bit is set, the FCU is in the Alarm state.
30 S1	When this bit is set, the FCU is in the Normal state.
31 S0	When this bit is set, the FCU is in the Init state.

29.2.3.10 Microcontroller State Register (FCU_MCSR)

The FCU_MCSR indicates the current state of the microcontroller in the MCSA field and the previous state (before state change) in the MCSP field. The contents of this register are copied into the Frozen MC State Register (FCU_FMCSR) when the FCU goes into Fault state.

Address: Base + 0x0024 Access: User read-only, Supervisor read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	MCPS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	MCAS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 612. MC State Register (FCU_MCSR)

Table 545. FCU_MCSR field description

Field	Description
12:15 MCPS[3:0]	MC Previous State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved
28:31 MCAS[3:0]	MC Actual State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved

29.2.3.11 Frozen MC State Register (FCU_FMCSR)

The FCU_MCSR is copied into the FCU_FMCSR each time the Fault state is entered. The FCU_FMCSR bit description is the same as that of the FCU_MCSR.

Address: Base + 0x0028 Access: User read-only, Supervisor read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCPS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	FRMCAS[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 613. Frozen MC State Register (FCU_FMCSR)

Table 546. FCU_FMCSR field description

Field	Description
12:15 FRMCPS [3:0]	MC Previous State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved
28:31 FRMCAS [3:0]	MC Actual State 0000: RESET 0001: TEST 0010: SAFE 0011: DRUN 0100: RUN0 0101: RUN1 0110: RUN2 0111: RUN3 1000: HALT0 1001: Reserved 1010: STOP 1011: Reserved 1100: Reserved 1101: Reserved 1110: Reserved 1111: Reserved

29.3 Functional description

The FCU module contains six functional blocks as shown in *Figure 614*.

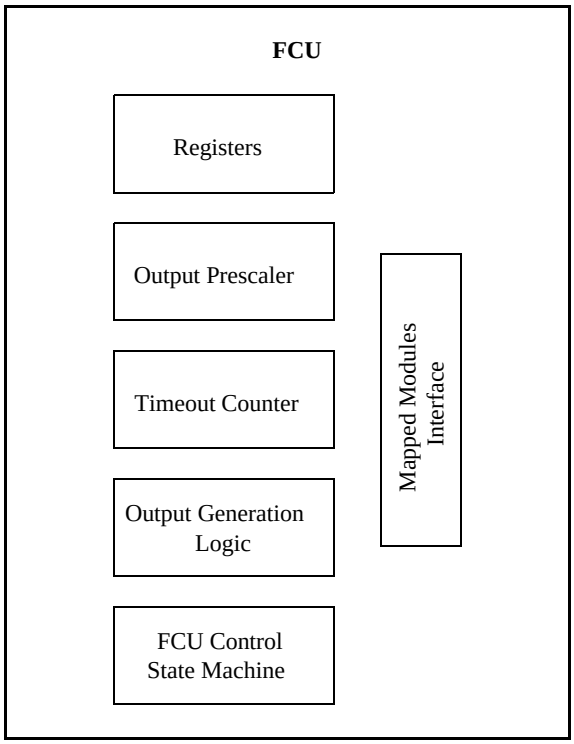


Figure 614. Functional block diagram

The register block implements all the FCU registers including input capture logic. The interface implements the read functionality and generated write and register enable signals. The timeout counter implements the counter to calculate timeout to switch from Alarm state to Fault state. The output prescaler module generates the prescaled clock to be used to generate output sequence. The FCU control implements the FCU state machine. The output generation logic generates two external output signals according to the output generation protocol.

29.3.1 State machine

The FCU provides four states: Init, Normal, Alarm, and Fault.

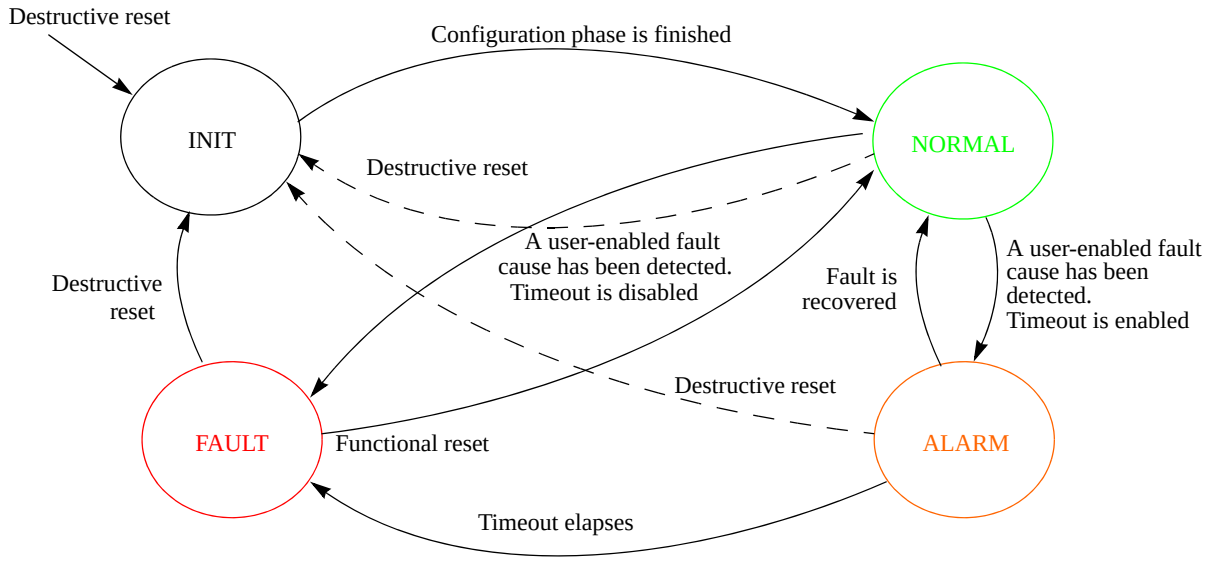


Figure 615. Finite state machine

The Init state is entered after any destructive reset assertion. Once the FCU goes into Fault state, both functional and destructive reset assertion lets the FCU move into Init state.

Once a configuration is locked (see *Section 29.2.3.1: Module Configuration Register (FCU_MCR)*), the FCU goes into Normal state. Then, when a fault occurs, the FCU can move into Alarm/Fault states depending on the Fault Enable Register (FCU_FER). The Fault Flag Register (FCU_FFR) stores any fault that has occurred, even if the FCU is not entering into the Alarm or the Fault state.

When the FCU is in Alarm state, a timer starts counting up to a fixed timeout (see *Section 29.2.3.7: Timeout Register (FCU_TR)*) before going into Fault state.

When the FCU goes into Fault state, the status of the FCU_MCR is copied into the FCU_FMCSR. Also, the FCU_FFR is copied into the FCU_FFFR.

The FCU can be tested by setting field TM[1:0] in the FCU_MCR during initialization. In this way, the FCU_FFGR can be accessed and faults can be emulated, since FCU behavior is the same whether real or fake faults occur. Optionally, the output pins can be disabled in Test mode (by setting the TM field to ‘01’). To exit from Test mode, the TM field must be set to ‘00’ or ‘11’.

After a functional reset, the FCU_FFR (not the Frozen Fault Flag Register (FCU_FFFR)) must be cleared and the FCU must return to Normal state.

29.3.2 Output generation protocol

The FCU provides two external output signals. The FCU supports different protocols for fault indication to the external device. Both external signals are used only in dual-rail protocol. In other protocols, the second output is the inverted version of the first output.

In all the protocols, depending on the polarity field (PS) in the MCR, the outputs can be normal polarity (if PS = 0) or inverted polarity (if PS = 1). The LSB of PS sets the polarity of the first signal; the MSB sets the polarity of the second signal.

29.3.2.1 Dual-rail protocol

Dual-rail encoding is an alternate method for encoding bits. In contrast to classical encoding, where each wire carries a single bit-value, dual-rail encoded circuits use two wires to carry each bit. *Table 547* summarizes the encoding.

Table 547. Dual-rail coding

Logical value	Dual-rail encoding (FCU[0], FCU[1])
non-faulty	(0,1)
non-faulty	(1,0)
faulty	(0,0)
faulty	(1,1)

As long as the FCU is in Normal or Alarm state, the output shows a “non-faulty” signal. FCU[0], FCU[1] toggles between (0,1) and (1,0) with a given frequency, set in the FOP field of the MCR. By default, this value is f = 976 Hz @ 64 MHz (about 1 kHz, see *Equation 64*). The same frequency is used to show a faulty situation (FCU in Fault state).

In the Init state, output pins are set as high impedance by clearing the OBE bits for each pad in its respective PCR in the SIUL module.

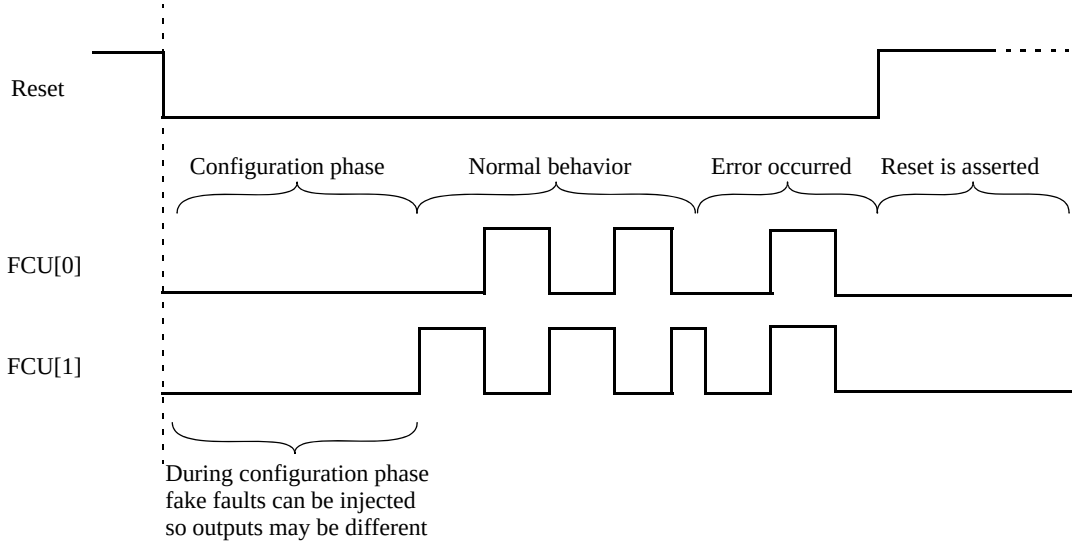


Figure 616. Dual rail coding example

29.3.2.2 Time switching protocol

FCU[0] is toggled between logic 0 and logic 1 with a defined frequency f = 1 kHz @ 64 MHz (f is approximated, as shown by *Equation 64*) and duty cycle d = 50%.

Equation 64

$$\text{EOUT freq} = \text{SYS\_CLK} \div \{4096 \times [(\text{FOP} + 1) \times 2]\}$$

$$64\ \text{MHz} \div \{4096 \times [(7 + 1) \times 2]\} = 976.5\ \text{Hz}$$

Frequency can be varied by using the same prescaler as used for the dual-rail protocol (FOP field in FCU_MCR). This frequency modulation protocol is violated when the FCU goes into Fault state.

During initialization phase, FCU[0] is set high. When a fault is detected, FCU[0] is set low.

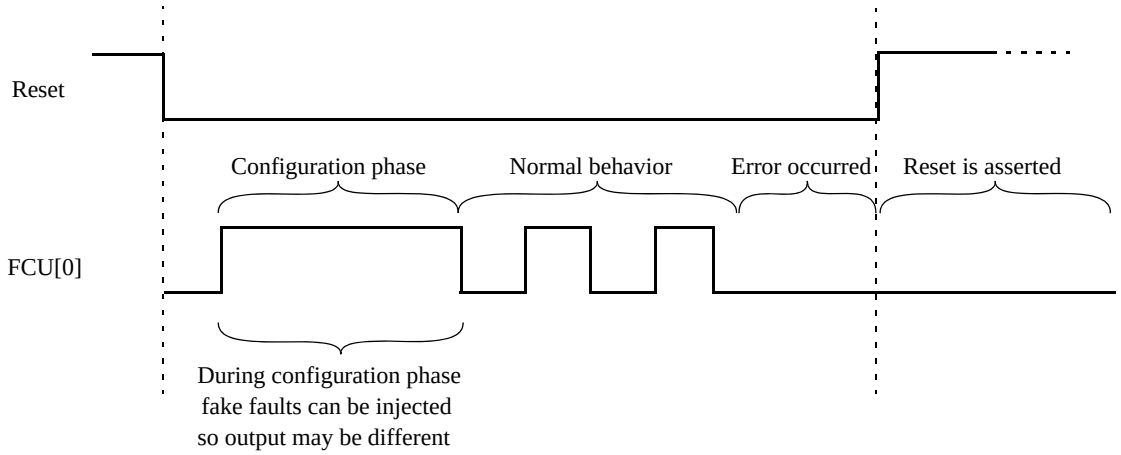


Figure 617. Time switching protocol example

29.3.2.3 Bi-Stable protocol

In this protocol during the Init and the Fault state, faulty state is indicated. In the Normal/Alarm state, non-faulty state is indicated. *Table 548* shows bi-stable encoding for FCU[0].

Table 548. Bi-stable coding

Logical value	Bi-stable encoding
faulty	0
non-faulty	1

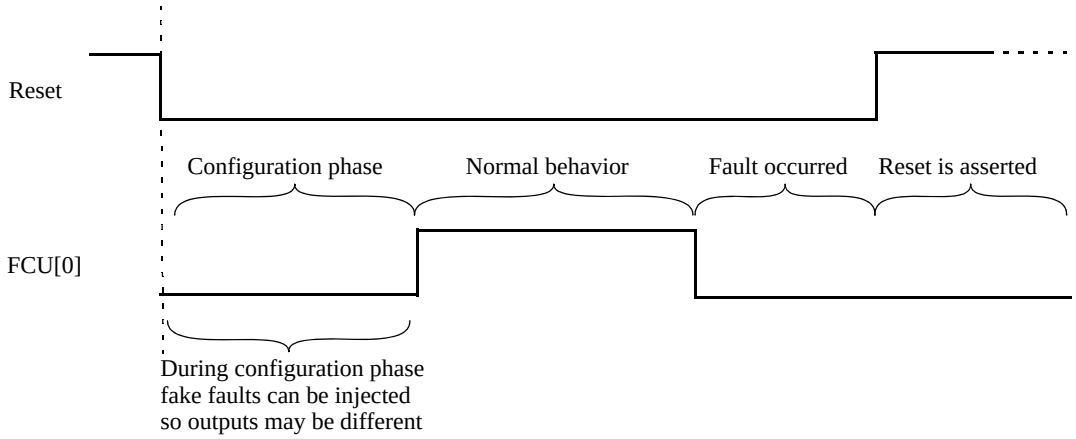


Figure 618. Bi-stable coding example

30 Wakeup Unit (WKPU)

30.1 Overview

The Wakeup Unit (WKPU) supports one external source that causes non-maskable interrupt requests.

30.2 Features

The WKPU provides non-maskable interrupt support with these features:

- 1 NMI source
- 1 analog glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request
- Edge detection

30.3 External signal description

The WKPU has one signal input that can be used as non-maskable interrupt.

Note: The user should be aware that the Wake-up pins are enabled in ALL modes, therefore, the Wake-up pins should be correctly terminated to ensure minimal current consumption. Any unused Wake-up signal input should be terminated by using an external pull-up or pull-down, or by internal pull-up enabled at WKUP_WIPUER. Also care has to be taken on packages where the Wake-up signal inputs are not bonded. For these packages the user must ensure the internal pull-up are enabled for those signals not bonded.

30.4 Memory map and registers description

This section provides a detailed description of all registers accessible in the WKPU module.

30.4.1 Memory map

Table 549 lists the WKPU registers.

Table 549. WKPU memory map

Offset from WKPU_BASE (0xC3F9_4000)	Register	Location
0x0000	NSR—NMI Status Flag Register	*on page 958*
0x0004–0x0007	Reserved	
0x0008	NCR—NMI Configuration Register	*on page 958*
0x000C–0x3FFF	Reserved	

30.4.2 Registers description

This section describes the Wakeup Unit registers.

30.4.2.1 NMI Status Flag Register (NSR)

This register holds the non-maskable interrupt status flags.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NIF	NOVF	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 619. NMI Status Flag Register (NSR)

Table 550. NSR field descriptions

Field	Description
0 NIF	NMI Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. If enabled (NCR.NREEor NCR.NFEE is set), NIFcauses an interrupt request. 0: No event has occurred on the pad. 1: An event as defined by NRRC.NREE or NCR.NFEE has occurred.
1 NOVF	NMI Overrun Status Flag This flag can be cleared only by writing a 1. Writing a 0 has no effect. It will be a copy of the current NIF value whenever an NMI event occurs, thereby indicating to the software that an NMI occurred while the last one was not yet serviced. If enabled (NCR.NREEor NCR.NFEE set), NOVF causes an interrupt request. 0: No overrun has occurred on NMI input. 1: An overrun has occurred on NMI input.

30.4.2.2 NMI Configuration Register (NCR)

This register holds the configuration bits for the non-maskable interrupt settings.

Address: Base + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NLOCK	NDSS		0	0	NREE	NFEE	NFE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 620. NMI Configuration Register (NCR)

Table 551. NCR field descriptions

Field	Description
0 NLOCK	NMI Configuration Lock Register Writing a 1 to this bit locks the configuration for the NMI until it is unlocked by a system reset. Writing a 0 has no effect.
1-2 NDSS	NMI Destination Source Select 00: Non-maskable interrupt 01: Critical interrupt 10: Machine check request 11: Reserved
5 NREE	NMI Rising-edge Events Enable 0: Rising-edge event is disabled 1: Rising-edge event is enabled
6 NFEE	NMI Falling-edge Events Enable 0: Falling-edge event is disabled 1: Falling-edge event is enabled
7 NFE	NMI Filter Enable Enable analog glitch filter on the NMI pad input. 0: Filter is disabled 1: Filter is enabled

Note: Writing a 0 to both NREEand NFEE disables the NMI functionality completely (that is, no system wakeup or interrupt will be generated on any pad activity)!

30.5 Functional description

30.5.1 General

This section provides a complete functional description of the Wakeup Unit.

30.5.2 Non-Maskable Interrupts

The Wakeup Unit supports one non-maskable interrupt, which is allocated to pin 1.

The Wakeup Unit supports the generation of three types of interrupts from the NMI input to the device. The Wakeup Unit supports the capturing of a second event per NMI input before the interrupt is cleared, thus reducing the chance of losing an NMI event.

Each NMI passes through a bypassable analog glitch filter.

Note: Glitch filter control and pad configuration should be done while the NMI is disabled in order to avoid erroneous triggering by glitches caused by the configuration process itself.

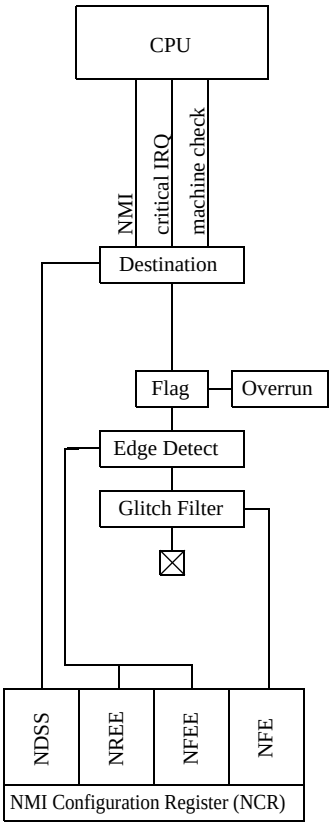


Figure 621. NMI pad diagram

30.5.2.1 NMI management

The NMI can be enabled or disabled using the single NCR register laid out to contain all configuration bits for an NMI in a single byte (see *Figure 620*). The pad defined as an NMI can be configured by the user to recognize interrupts with an active rising edge, an active falling edge or both edges being active. A setting of having both edge events disabled results in no interrupt being detected and should not be configured.

The active NMI edge is controlled by the user through the configuration of the NREE and NFEE bits.

Note: After reset, NREE and NFEE are set to 0, therefore the NMI functionality is disabled after reset and must be enabled explicitly by software.

Once the pad's NMI functionality has been enabled, the pad cannot be reconfigured in the IOMUX to override or disable the NMI.

The NMI destination interrupt is controlled by the user through the configuration of the NDSS bits. See *Table 551* for details.

An NMI supports a status flag and an overrun flag, which are located in the NSR register (see *Figure 619*). This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register. The status flag is set whenever an NMI event is detected. The overrun flag is set whenever an NMI event is detected and the status flag is set (that is, has not yet been cleared).

Note: The overrun flag is cleared by writing a 1 to the appropriate overrun bit in the NSR register. If the status bit is cleared and the overrun bit is still set, the pending interrupt will not be cleared.

31 Periodic Interrupt Timer (PIT)

31.1 Introduction

The Periodic Interrupt Timer (PIT) block is an array of four timers that can be used for DMA triggering, general purpose interrupts and system wakeup.

Figure 622 shows the PIT block diagram.

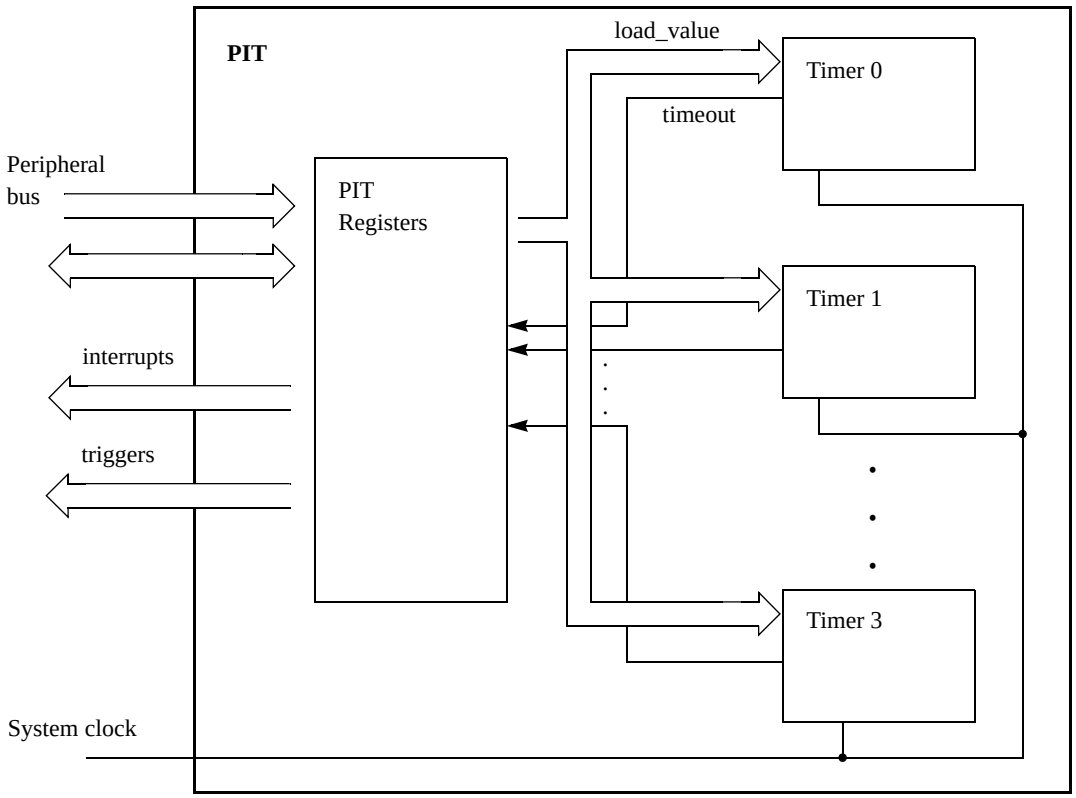


Figure 622. PIT block diagram

31.2 Features

The main features of this block are:

- Timers can generate DMA trigger pulses to initiate DMA transfers with other peripherals (ex: initiate a SPI message transfer sequence)
- Timers can generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer

31.3 Signal description

The PIT module has no external pins.

31.4 Memory map and registers description

31.4.1 Memory map

Table 552 gives an overview of the implemented PIT registers.

Table 552. PIT memory map

Offset from PIT_BASE (0xC3FF_0000)	Register	Location
0x0000	PITMCR—PIT Module Control Register	*on page 964*
0x0004–0x00FF	Reserved	
Timer Channel 0		
0x0100	LDVAL0—Timer 0 Load Value Register	*on page 964*
0x0104	CVAL0—Timer 0 Current Value Register	*on page 965*
0x0108	TCTRL0—Timer 0 Control Register	*on page 966*
0x010C	TFLG0—Timer 0 Flag Register	*on page 966*
Timer Channel 1		
0x0110	LDVAL1—Timer 1 Load Value Register	*on page 964*
0x0114	CVAL1—Timer 1 Current Value Register	*on page 965*
0x0118	TCTRL1—Timer 1 Control Register	*on page 966*
0x011C	TFLG1—Timer 1 Flag Register	*on page 966*
Timer Channel 2		
0x0120	LDVAL2—Timer 2 Load Value Register	*on page 964*
0x0124	CVAL2—Timer 2 Current Value Register	*on page 965*
0x0128	TCTRL2—Timer 2 Control Register	*on page 966*
0x012C	TFLG2—Timer 2 Flag Register	*on page 966*
Timer Channel 3		
0x0130	LDVAL3—Timer 3 Load Value Register	*on page 964*
0x0134	CVAL3—Timer 3 Current Value Register	*on page 965*
0x0138	TCTRL3—Timer 3 Control Register	*on page 966*
0x013C	TFLG3—Timer 3 Flag Register	*on page 966*
0x0140–0x3FFF	Reserved	

Note: Reserved registers read as 0. Writes have no effect.

31.4.2 Registers description

This section describes in address order all the PIT registers and their individual bits. PIT registers are accessible only when the core is in supervisor mode (see *Section 15.4.3: ECSM_reg_protection*).

31.4.2.1 PIT Module Control Register (PITMCR)

This register controls whether the timer clocks are enabled and whether the timers run in debug mode.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 623. PIT Module Control Register (PITMCR)

Table 553. PITMCR field descriptions

Field	Description
MDIS	Module Disable Used to disable the module clock. This bit should be enabled before any other setup is done. 0: Clock for PIT Timers is enabled 1: Clock for PIT Timers is disabled (default)
FRZ	Freeze Allows the timers to be stopped when the device enters debug mode. 0: Timers continue to run in debug mode. 1: Timers are stopped in debug mode.

31.4.2.2 Timer Load Value Register *n* (LDVAL*n*)

These registers select the timeout period for the timer interrupts.

Address: Channel Base + 0x0000 Access: User read/write

LDVAL0 = PIT_BASE + 0x0100
LDVAL1 = PIT_BASE + 0x0110
LDVAL2 = PIT_BASE + 0x0120
LDVAL3 = PIT_BASE + 0x0130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TSV 31	TSV 30	TSV 29	TSV 28	TSV 27	TSV 26	TSV 25	TSV 24	TSV 23	TSV 22	TSV 21	TSV 20	TSV 19	TSV 18	TSV 17	TSV 16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TSV 15	TSV 14	TSV 13	TSV 12	TSV 11	TSV 10	TSV 9	TSV 8	TSV7	TSV6	TSV5	TSV4	TSV3	TSV2	TSV1	TSV0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 624. Timer Load Value Register *n* (LDVAL*n*)

Table 554. LDVAL*n* field descriptions

Field	Description
TSV*n*	Time Start Value Bits These bits set the timer start value. The timer will count down until it reaches 0, then it will generate an interrupt and load this register value again. Writing a new value to this register will not restart the timer, instead the value will be loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see *Figure 629*).

31.4.2.3 Current Timer Value Register *n* (CVAL*n*)

These registers indicate the current timer position.

Address: Channel Base + 0x0004 Access: User read-only

CVAL0 = PIT_BASE + 0x0104
CVAL1 = PIT_BASE + 0x0114
CVAL2 = PIT_BASE + 0x0124
CVAL3 = PIT_BASE + 0x0134

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TVL31	TVL30	TVL29	TVL28	TVL27	TVL26	TVL25	TVL24	TVL23	TVL22	TVL21	TVL20	TVL19	TVL18	TVL17	TVL16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL15	TVL14	TVL13	TVL12	TVL11	TVL10	TVL9	TVL8	TVL7	TVL6	TVL5	TVL4	TVL3	TVL2	TVL1	TVL0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 625. Current Timer Value register *n* (CVAL*n*)

Table 555. CVAL*n* field descriptions

Field	Description
TVL*n*	Current Timer Value These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values will be frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see *Figure 623*).

31.4.2.4 Timer Control Register *n* (TCTRL*n*)

The TCTRL register contains the control bits for each timer.

Address: Channel Base + 0x0008 Access: User read/write

TCTRL0 = PIT_BASE + 0x0108
TCTRL1 = PIT_BASE + 0x0118
TCTRL2 = PIT_BASE + 0x0128
TCTRL3 = PIT_BASE + 0x0138

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 626. Timer Control register *n* (TCTRL*n*)

Table 556. TCTRL*n* field descriptions

Field	Description
TIE	Timer Interrupt Enable Bit 0: Interrupt requests from Timer x are disabled 1: Interrupt will be requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt will immediately cause an interrupt event. To avoid this, the associated TIF flag must be cleared first.
TEN	Timer Enable Bit 0: Timer will be disabled 1: Timer will be active

31.4.2.5 Timer Flag Register *n* (TFLG*n*)

These registers contain the PIT interrupt flags.

Address: Channel Base + 0x000C Access: User read/write

TFLG0 = PIT_BASE + 0x010C
TFLG1 = PIT_BASE + 0x011C
TFLG2 = PIT_BASE + 0x012C
TFLG3 = PIT_BASE + 0x013C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 627. Timer Flag register *n* (TFLG*n*)

Table 557. TFLG*n* field descriptions

Field	Description
TIF	Time Interrupt Flag TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0: Time-out has not yet occurred 1: Time-out has occurred

31.5 Functional description

31.5.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate trigger pulses as well as to generate interrupts, each interrupt will be available on a separate interrupt line.

31.5.1.1 Timers

The timers generate triggers at periodic intervals, when enabled. They load their start values, as specified in their LDVAL registers, then count down until they reach 0. Then they load their respective start value again. Each time a timer reaches 0, it will generate a trigger pulse, and set the interrupt flag.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see *Figure 628*).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see *Figure 629*).

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value will then be loaded after the next trigger event (see *Figure 630*).

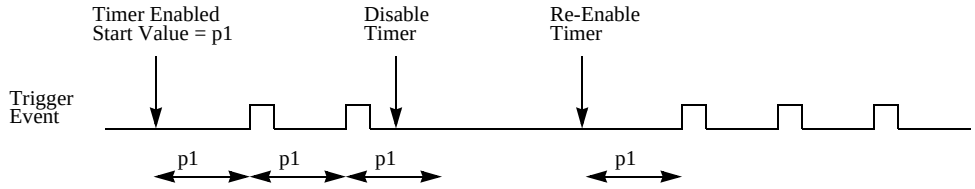


Figure 628. Stopping and starting a timer

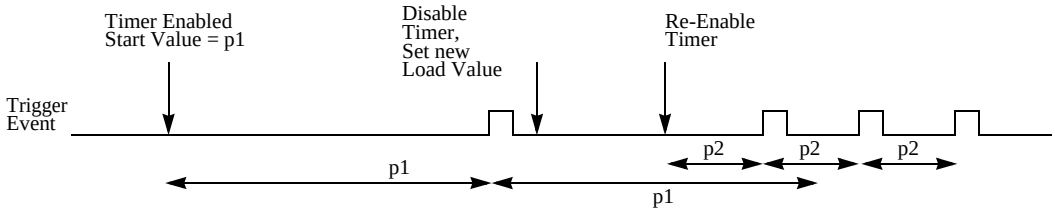


Figure 629. Modifying running timer period

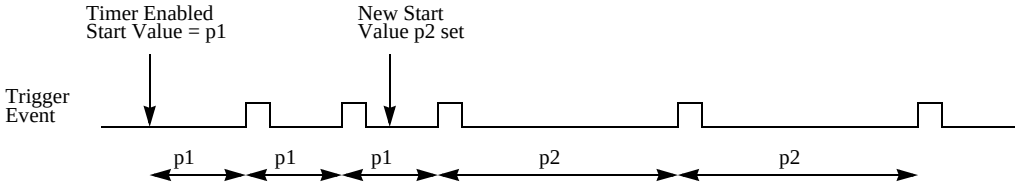


Figure 630. Dynamically setting a new load value

31.5.1.2 Debug mode

In Debug mode, the timers are frozen. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g., timer values) and then continue the operation.

31.5.2 Interrupts

All of the timers support interrupt generation. Refer to *Chapter 9: Interrupt Controller (INTC)* for related vector addresses and priorities.

Timer interrupts can be disabled by setting the TIE bits to zero. The timer interrupt flags (TIF) are set to 1 when a timeout occurs on the associated timer, and are cleared to 0 by writing a 1 to that TIF bit.

31.6 Initialization and application information

31.6.1 Example configuration

In the example configuration:

- The PIT clock has a frequency of 50 MHz
- Timer 1 creates an interrupt every 5.12 ms
- Timer 3 creates a trigger event every 30 ms

First the PIT module needs to be activated by writing a 0 to the MDIS bit in the PITMCR.

The 50 MHz clock frequency equates to a clock period of 20 ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles and timer 3 every 30 ms/20 ns = 1500000 cycles. The value for the LDVAL register trigger would be calculated as (period / clock period) – 1.

The LDVAL registers must be configured as follows:

- LDVAL for Timer 1: 0x0003_E7FF
- LDVAL for Timer 3: 0x0016_E35F

The interrupt for Timer 1 is enabled by setting TIE in the TCTRL1 register. The timer is started by writing a 1 to bit TEN in the TCTRL1 register.

Timer 3 shall be used only for triggering. Therefore Timer 3 is started by writing a 1 to bit TEN in the TCTRL3 register, bit TIE stays at 0.

The following example code matches the described setup:

```
// turn on PIT
PIT_CTRL = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL = PIT_TIE; // let RTI generate interrupts
PIT_RTI_TCTRL |= PIT_TEN; // start RTI

// Timer 1
PIT_LDVAL1 = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_TCTRL1 = TIE; // enable Timer 1 interrupts
PIT_TCTRL1 |= TEN; // start timer 1

// Timer 3
PIT_LDVAL3 = 0x0016E35F; // setup timer 3for 1500000 cycles
PIT_TCTRL3 = TEN; // start timer 3
```

32 System Timer Module (STM)

32.1 Overview

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

32.2 Features

The STM has the following features:

- One 32-bit up counter with 8-bit prescaler
- Four 32-bit compare channels
- Independent interrupt source for each channel
- Counter can be stopped in debug mode

32.3 Modes of operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

32.4 External signal description

The STM does not have any external interface signals.

32.5 Memory map and registers description

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination. STM registers are accessible only when the core is in supervisor mode (see *Section 15.4.3: ECSM_reg_protection*).

32.5.1 Memory map

The STM memory map is shown in *Table 558*.

Table 558. STM memory map

Offset from STM_BASE 0xFFF3_C000	Register	Location
0x0000	STM_CR—STM Control Register	*on page 971*
0x0004	STM_CNT—STM Counter Value	*on page 972*
0x0008–0x000F	Reserved	
0x0010	STM_CCR0—STM Channel 0 Control Register	*on page 973*
0x0014	STM_CIR0—STM Channel 0 Interrupt Register	*on page 973*
0x0018	STM_CMP0—STM Channel 0 Compare Register	*on page 974*
0x001C	Reserved	
0x0020	STM_CCR1—STM Channel 1 Control Register	*on page 973*
0x0024	STM_CIR1—STM Channel 1 Interrupt Register	*on page 973*
0x0028	STM_CMP1—STM Channel 1 Compare Register	*on page 974*
0x002C	Reserved	
0x0030	STM_CCR2—STM Channel 2 Control Register	*on page 973*
0x0034	STM_CIR2—STM Channel 2 Interrupt Register	*on page 973*
0x0038	STM_CMP2—STM Channel 2 Compare Register	*on page 974*
0x003C	Reserved	
0x0040	STM_CCR3—STM Channel 3 Control Register	*on page 973*
0x0044	STM_CIR3—STM Channel 3 Interrupt Register	*on page 973*
0x0048	STM_CMP3—STM Channel 3 Compare Register	*on page 974*
0x004C–0x3FFF	Reserved	

32.5.2 Registers description

The following sections detail the individual registers within the STM programming model.

32.5.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Address: Base + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS[7:0]								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 631. STM Control Register (STM_CR)

Table 559. STM_CR field descriptions

Field	Description
CPS[7:0]	Counter Prescaler Selects the clock divide value for the prescaler (1 - 256). 0x00 Divide system clock by 1. 0x01 Divide system clock by 2. ... 0xFF Divide system clock by 256.
FRZ	Freeze Allows the timer counter to be stopped when the device enters debug mode. 0 STM counter continues to run in debug mode. 1 STM counter is stopped in debug mode.
TEN	Timer Counter Enabled 0 Counter is disabled. 1 Counter is enabled.

32.5.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Address: Base + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 632. STM Count Register (STM_CNT)

Table 560. STM_CNT field descriptions

Field	Description
CNT	Timer count value used as the time base for all channels. When enabled, the counter increments at the rate of the system clock divided by the prescale value.

32.5.2.3 STM Channel Control Register (STM_CCR*n*)

The STM Channel Control Register (STM_CCR*n*) enables and services channel *n* of the timer.

Address: Base + 0x0010 (STM_CCR0)
Base + 0x0020 (STM_CCR1)
Base + 0x0030 (STM_CCR2)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 633. STM Channel Control Register (STM_CCR*n*)

Table 561. STM_CCR*n* field descriptions

Field	Description
CEN	Channel Enable 0 The channel is disabled. 1 The channel is enabled.

32.5.2.4 STM Channel Interrupt Register (STM_CIR*n*)

The STM Channel Interrupt Register (STM_CIR*n*) enables and services channel *n* of the timer.

Address: Base + 0x0014 (STM_CIR0)
Base + 0x0024 (STM_CIR1)
Base + 0x0034 (STM_CIR2)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 634. STM Channel Interrupt Register (STM_CIR*n*)

Table 562. STM_CIR*n* field descriptions

Field	Description
CIF	Channel Interrupt Flag The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 No interrupt request. 1 Interrupt request due to a match on the channel.

32.5.2.5 STM Channel Compare Register (STM_CMP*n*)

The STM Channel Compare Register (STM_CMP*n*) holds the compare value for channel *n*.

Address: Base + 0x0018 (STM_CMP0)
Base + 0x0028 (STM_CMP1)
Base + 0x0038 (STM_CMP2)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CMP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 635. STM Channel Compare Register (STM_CMP*n*)

Table 563. STM_CMP*n* field descriptions

Field	Description
CMP	Compare value for channel n If the STM_CCR*n*[CEN] bit is set and the STM_CMP*n* register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIR*n*[CIF] bit is set.

32.6 Functional description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCR*n*), a channel interrupt register (STM_CIR*n*) and a channel compare register (STM_CMP*n*). The channel is enabled by setting the STM_CCR*n*[CEN] bit. When enabled, the channel will set the STM_CIR[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a 1 to the STM_CIR*n*[CIF] bit. A write of 0 to the STM_CIR*n*[CIF] bit has no effect.

33 Cyclic Redundancy Check (CRC)

33.1 Introduction

The Cyclic Redundancy Check (CRC) computing unit is dedicated to the computation of CRC, thus off-loading the CPU. The SPC560P44Lx, SPC560P50Lx CRC supports two contexts. Each context has a separate CRC computation engine in order to allow the concurrent computation of the CRC of multiple data streams. The CRC computation is performed at speed without wait states insertion. Bit-swap and bit-inversion operations can be applied on the final CRC signature. Each context can be configured with one of two hard-wired polynomials, normally used for most of the standard communication protocols. The data stream supports multiple data width (byte/half-word/word) formats.

33.1.1 Glossary

- CRC: cyclic redundancy check
- CPU: central processing unit
- DMA: direct memory access
- CCITT: ITU-T (for Telecommunication Standardization Sector of the International Telecommunications Union)
- SW: software
- WS: wait state
- SPI: serial peripheral interface

33.2 Main features

- 2 contexts for the concurrent CRC computation
- Separate CRC engine for each context
- Zero-wait states during the CRC computation (pipeline scheme)
- 2 hard-wired polynomials (CRC-16-CCITT and CRC-32 ethernet)
- Support for byte/half-word/word width of the input data stream

33.2.1 Standard features

- IPS bus interface
- CRC-16-CCITT
- CRC-32 ethernet

33.3 Block diagram

Figure 636 shows the top level diagram of the CRC unit.

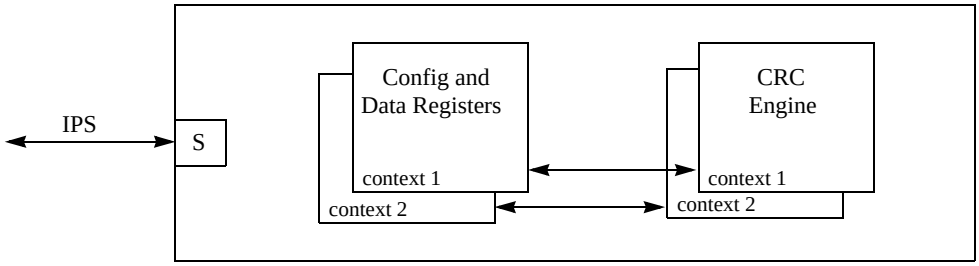


Figure 636. CRC top level diagram

33.3.1 IPS bus interface

The IPS bus interface is a slave bus used for configuration and data streaming (CRC computation) purposes via CPU or DMA. The following bus operations (contiguous byte enables) are supported:

- Word (32-bit) data write/read operations to any registers
- Low and high half-words (16-bit, data[31:16] or data[15:0]) data write/read operations to any registers
- Byte (8-bit, data[31:24] or data[23:16] or data[15:8] or data[7:0]) data write/read operations to any registers
- Any other operation (free byte enables or other operations) must be avoided.

The CRC generates a transfer error in the following cases:

- Any write/read access to the register addresses not mapped on the peripheral but included in the address space of the peripheral.
- Any write/read operation different from byte/hword/word (free byte enables or other operations) on each register.

The registers of the CRC module are accessible (read/write) in each access mode: user, supervisor, or test.

In terms of bus performance of the operations, following the summary:

- 0 WS (single bus cycle) for each write/read operations to the CRC_CFG and CRC_INP registers
- 0 WS (single bus cycle) for each write operation to the CRC_ CSTAT register
- Double WS (3 bus cycles) for each read operation to the CRC_ CSTAT or CRC_OUTP registers immediately following (next clock cycle) a write operation to the CRC_CSTAT, CRC_INP or CRC_CFG registers belonging to the same context. In all the other cases no WS are inserted.

33.4 Functional description

The CRC module supports the CRC computation for each context. Each context has a own complete set of registers including the CRC engine. The data flow of each context can be interleaved. The data stream can be structured as a sequence of byte, half-words or words. The input data sequence is provided, eventually mixing the data formats (byte, half-word, word), writing to the input data register (CRC_INP).

The data stream is generally executed by N concurrent DMA data transfers (mem2mem) where N is less or equal to the number of contexts.

Two standard generator polynomials are given in *Equation 65* and *Equation 66* for the CRC computation of each context.

Equation 65 CRC-16-CCITT (x25 protocol)

$$X^{16} + X^{12} + X^5 + 1$$

Equation 66 CRC-32 (ethernet protocol)

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

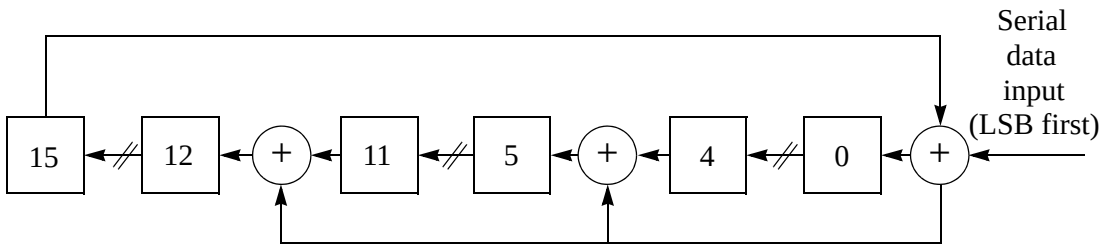


Figure 637. CRC-CCITT engine concept scheme

The initial seed value of the CRC can be programmed initializing the CRC_CSTAT register. The concept scheme (serial data loading) of the CRC engine is given in *Figure 637* for the CRC-CCITT. The design implementation executes the CRC computation in a single clock cycle (parallel data loading). A pipeline scheme has been adopted to de-couple the IPS bus interface from the CRC engine in order to allow the computation of the CRC at speed (zero wait states).

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (e.g., SPI, ..) a bit swap (MSB → LSB, LSB → MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register) before to transmit the CRC.

The usage of the CRC is summarized in the flow-chart given in *Figure 638*.

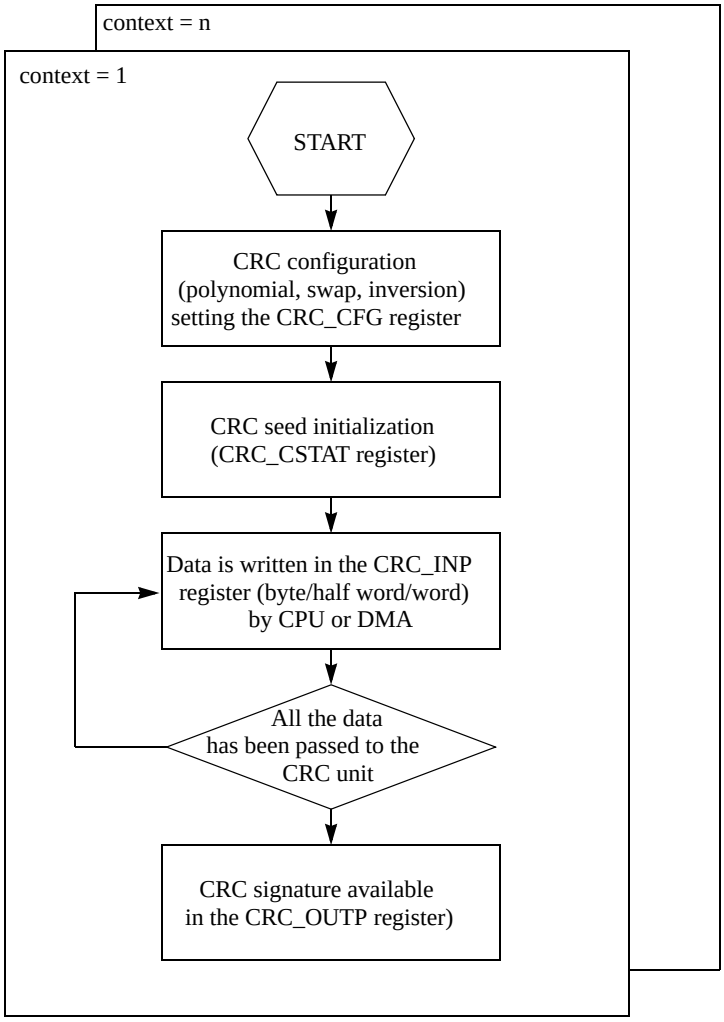


Figure 638. CRC computation flow

33.5 Memory map and registers description

Table 564 shows the CRC memory map.

Table 564. CRC memory map

Offset from CRC_BASE 0xFFE6_8000	Register	Location
0x0000	CRC_CFG—CRC Configuration Register, Context 1	*on page 980*
0x0004	CRC_INP—CRC Input Register, Context 1	*on page 981*
0x0008	CRC_CSTAT—CRC Current Status Register, Context 1	*on page 982*

Table 564. CRC memory map(Continued)

Offset from CRC_BASE 0xFFE6_8000	Register	Location
0x000C	CRC_OUTP—CRC Output Register, Context 1	*on page 982*
0x0010	CRC_CFG—CRC Configuration Register, Context 2	*on page 980*
0x0014	CRC_INP—CRC Input Register, Context 2	*on page 981*
0x0018	CRC_CSTAT—CRC Current Status Register, Context 2	*on page 982*
0x001C	CRC_OUTP—CRC Output Register, Context 2	*on page 982*
0x0020–0x3FFF	Reserved	

33.5.1 CRC Configuration Register (CRC_CFG)

Address: Context 1: Base + 0x0000
Context 2: Base + 0x0010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	POLYG(1)	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 639. CRC Configuration Register (CRC_CFG)

1. Only for CRC module 2 POLIG is "1", CRC_CFG - correct reset value is 0x0004.

Table 565. CRC_CFG field descriptions

Field	Description
0:28	Reserved These are reserved bits. These bits are always read as 0 and must always be written with 0.
29	POLYG: *Polynomial selection* 0: CRC-CCITT polynomial. 1: CRC-32 polynomial. This bit can be read and written by the software. This bit can be written only during the configuration phase. **Note:** To get the correct CRC signature selecting CRC-16-CCITT protocol, the input data has to be bit reversed and entered into the CRC Input Register (CRC_INP); INV and SWAP bits are programmed to zero. (Given input data of 0xABCDEF98, the user software must enter 0x19F7B3D5 into the CRC_INP register) **Note:** To get the correct CRC signature selecting CRC-32 protocol, the input data has to be byte reversed and entered into the CRC Input Register (CRC_INP); INV and SWAP bits are to be set to 1. (Given input data of 0xABCDEF98, the user software must enter 0x98EFCDAB into the CRC_INP register)

Table 565. CRC_CFG field descriptions(Continued)

Field	Description
30	SWAP: *SWAP selection* 0: No swap selection applied on the CRC_OUTP content 1: Swap selection (MSB -> LSB, LSB -> MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the swap operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.
31	INV: *INV selection* 0: No inversion selection applied on the CRC_OUTP content 1: Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by the software. This bit can be written only during the configuration phase.

33.5.2 CRC Input Register (CRC_INP)

Address: Context 1: Base + 0x0004
Context 2: Base + 0x0014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 640. CRC Input Register (CRC_INP)

Table 566. CRC_INP field descriptions

Field	Description
0:31	INP: *Input data for the CRC computation* The INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by the software.

33.5.3 CRC Current Status Register (CRC_CSTAT)

Address: Context 1: Base + 0x0008
Context 2: Base + 0x0018

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSTAT															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 641. CRC Current Status Register (CRC_CSTAT)

Table 567. CRC_CSTAT field descriptions

Field	Description
0:31	CSTAT: *Status of the CRC signature* The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 LSB bits are significant. The 16 MSB bits are tied at 0b during the computation. The CSTAT register can be written at byte, half-word or word. This register can be read and written by the software. This register can be written only during the configuration phase.

33.5.4 CRC Output Register (CRC_OUTP)

Address: Context 1: Base + 0x000C
Context 2: Base + 0x001C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	OUTP															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OUTP															
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 642. CRC Output Register (CRC_OUTP)

Table 568. CRC_OUTP field descriptions

Field	Description
0:31	OUTP: *Final CRC signature* The OUTP register includes the final signature corresponding to the CRC_CSTAT register value eventually swapped and inverted. In case of CRC-CCITT polynomial only the16 LSB bits are significant. The 16 MSB bits are tied at 0b during the computation. This register can be read by the software.

33.6 Use cases and limitations

Two main use cases shall be considered:

- Calculation of the CRC of the configuration registers during the process safety time
- Calculation of the CRC on the incoming/outgoing frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral.

The signature of the configuration registers is computed in a correct way only if these registers do not contain any status bit. Assuming that the DMA engine has N channels (greater or equal to the number of contexts) configurable for the following type of data transfer: mem2mem, periph2mem, mem2periph, the following sequence, as given in *Figure 643*, shall be applied to manage the transmission data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Payload transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase1) by DMA (mem2mem data transfer, channel x)
- CRC signature copy from the CRC module (CRC_OUTP register) to the MEM (phase 2) by CPU
- Data block (payload + CRC) transfer from the MEM to the PERIPH module (e.g., SPI Tx fifo) (phase 3) by DMA (mem2periph data transfer, channel x)

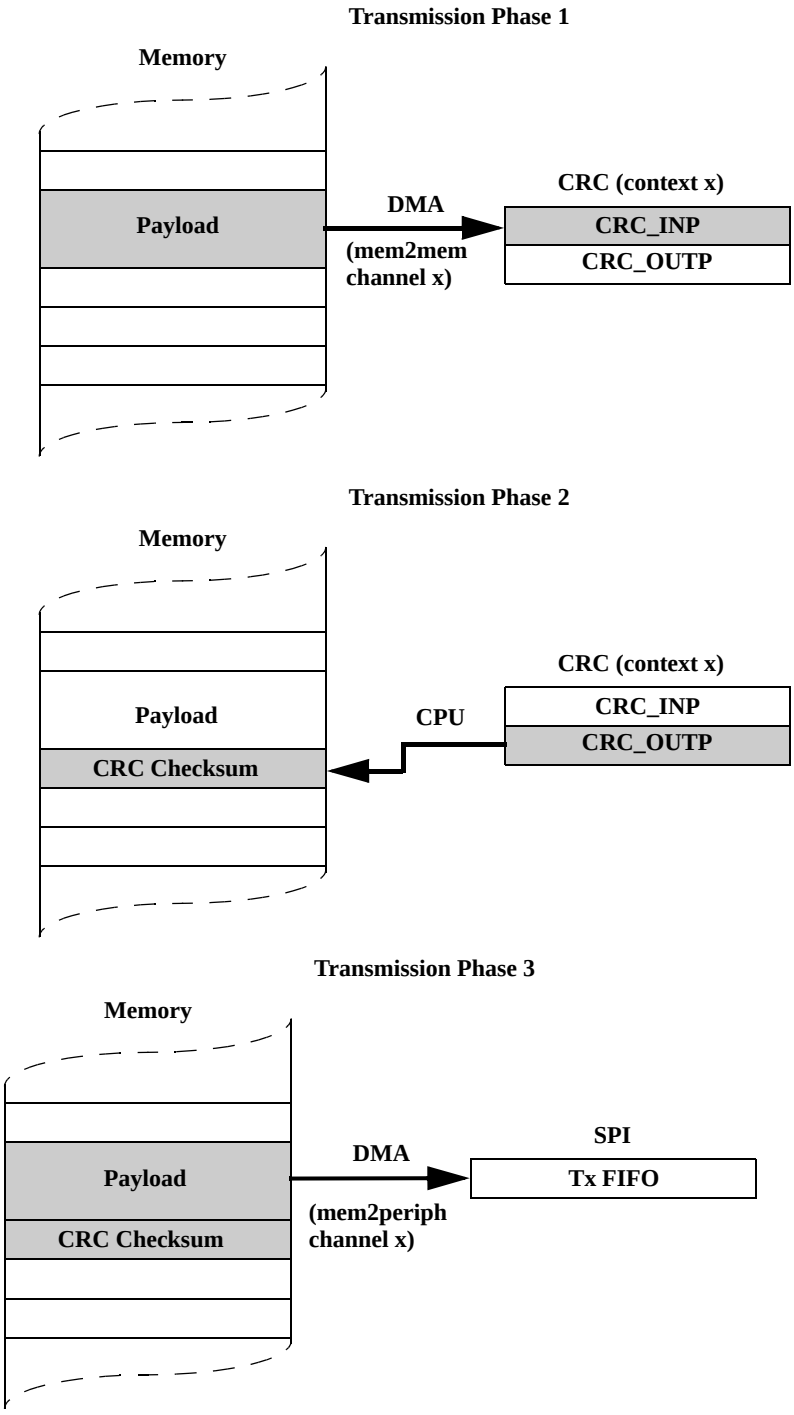


Figure 643. DMA-CRC Transmission Sequence

The following sequence, as given in *Figure 644*, shall be applied to manage the reception data flow:

- DMA/CRC module configuration (context x, channel x) by CPU
- Data block (payload + CRC) transfer from the PERIPH (e.g., SPI Rx fifo) module to the MEM (phase 1) by DMA (periph2mem data transfer, channel x)
- Data block transfer (payload + CRC) transfer from the MEM to the CRC module (CRC_INP register) to calculate the CRC signature (phase 2) by DMA (mem2mem data transfer, channel x)
- CRC signature check from the CRC module (CRC_OUTP register) by CPU (phase 3)1

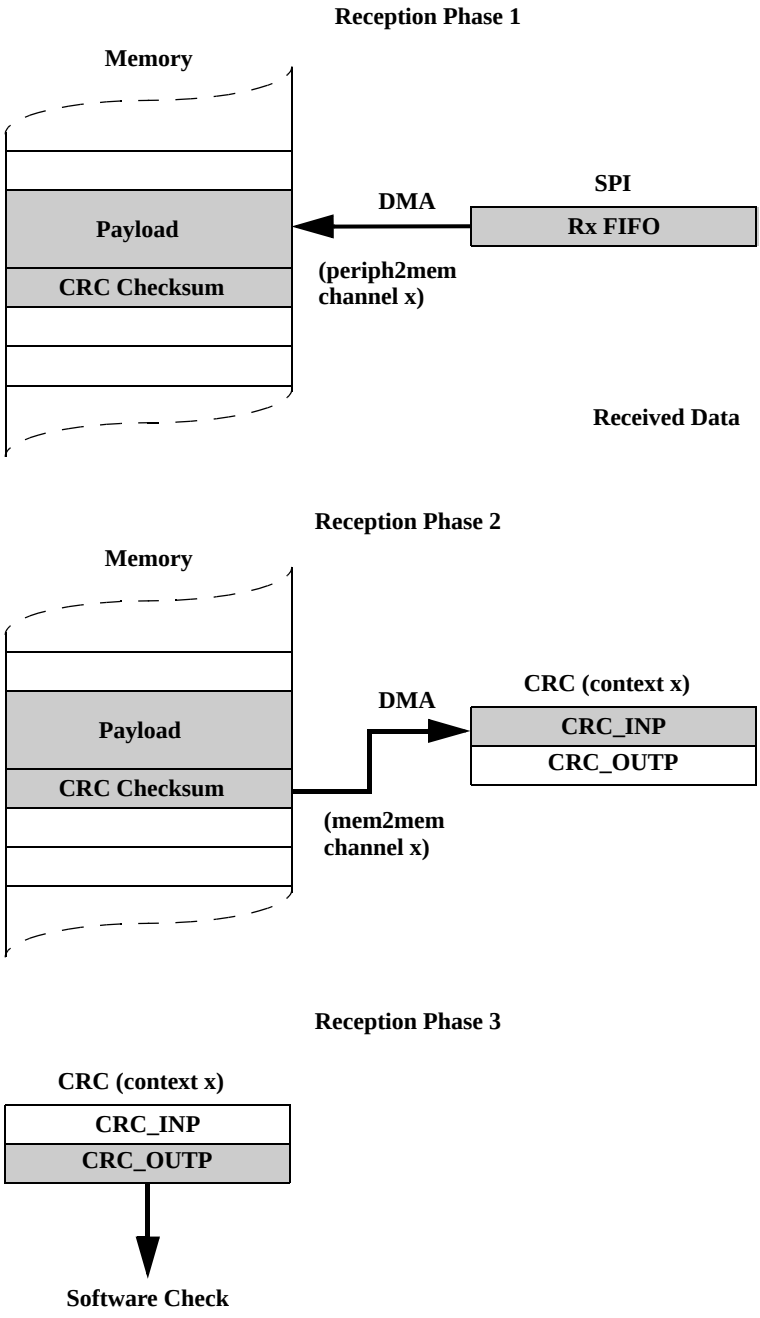


Figure 644. DMA-CRC Reception Sequence

34 Boot Assist Module (BAM)

34.1 Overview

The Boot Assist Module is a block of read-only memory containing VLE code that is executed according to the boot mode of the device.

The BAM allows downloading boot code via the FlexCAN or LINFlex interfaces into internal SRAM and then executing it.

34.2 Features

The BAM provides the following features:

- SPC560P44Lx, SPC560P50Lx in static mode if internal flash is not initialized or invalid
- Programmable 64-bit password protection for serial boot mode
- Serial boot loads the application boot code from a FlexCAN or LINFlex bus into internal SRAM
- Censorship protection for internal flash module

34.3 Boot modes

The SPC560P44Lx, SPC560P50Lx device supports the following boot modes:

- Single Chip (SC) — The device boots from the first bootable section of the Flash main array.
- Serial Boot (SBL) — The device downloads boot code from either LINFlex or FlexCAN interface and then execute it[g].

If booting is not possible with the selected configuration (e.g., if no Boot ID is found in the selected boot location) then the device enters the static mode.

34.4 Memory map

The BAM code resides in a reserved 8 KB ROM mapped from address 0xFFFF_C000. The address space and memory used by the BAM application is shown in *Table 569*.

Table 569. BAM memory organization

Parameter	Address
BAM entry point	0xFFFF_C000
Downloaded code base address	0x4000_0100

The RAM location where to download the code can be any 4-byte-aligned location starting from the address 0x4000_0100.

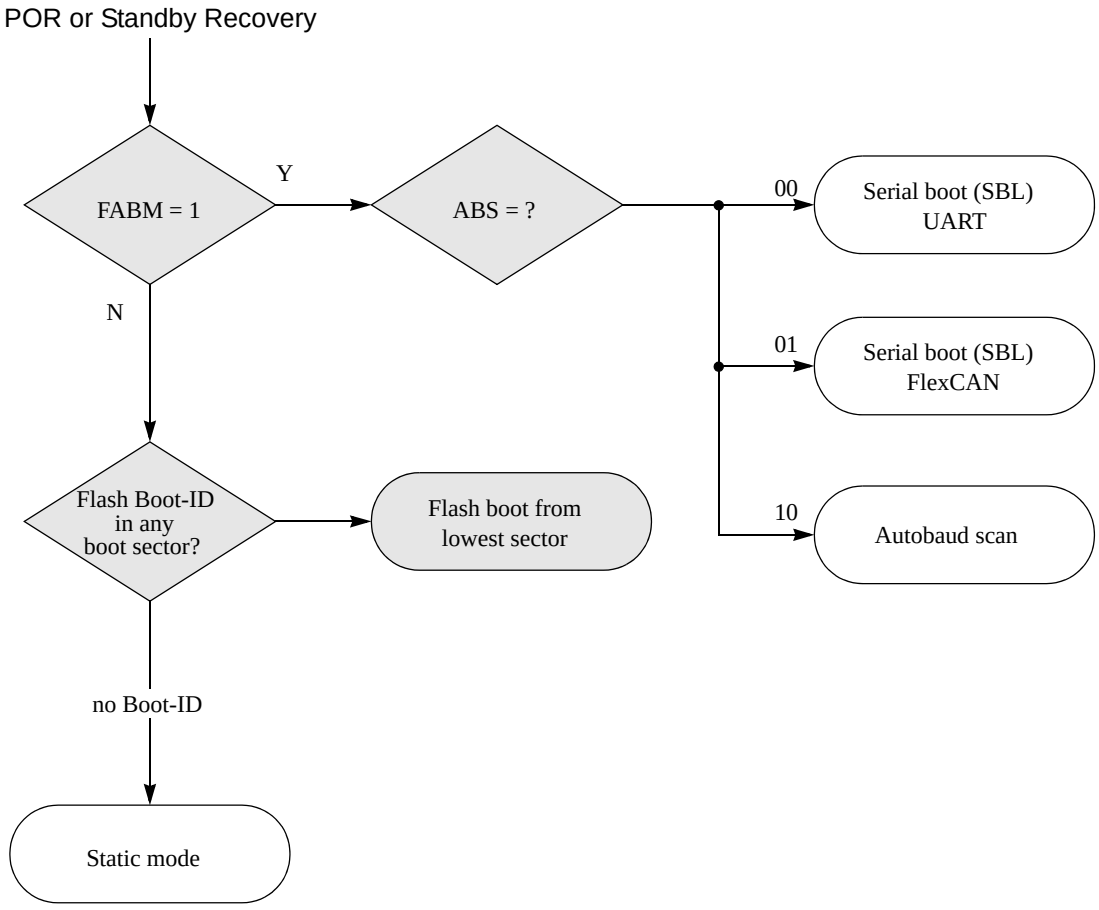
g. Support for the FlexRay interface is planned on future devices.

34.5 Functional description

34.5.1 Entering boot modes

The SPC560P44Lx, SPC560P50Lx detects the boot mode based on external pins and device status. The following sequence applies (see *Figure 645*):

- To boot either from FlexCAN or LINFlex, the device must be forced into an Alternate Boot Loader Mode via the FAB (Force Alternate Boot Mode), which must be asserted before initiating the reset sequence. The type of alternate boot mode is selected according to the ABS (Alternate Boot Selector) pins (see *Table 570*).
- If FAB is not asserted, the device boots from the lowest Flash sector that contains a valid boot signature.
- If no Flash sector contains a valid boot signature, the device will go into static mode.



Note: The gray blocks in this figure represent hardware actions; white blocks represent software (BAM) actions.

Figure 645. Boot mode selection

Boot configuration pins are:

- PAD A[2] - ABS[0],
- PAD A[3] - ABS[1],
- PAD A[4] - FAB

Table 570. Hardware configuration to select boot mode

FAB	ABS[1:0](1)	Standby-RAM Boot Flag	Boot ID	Boot Mode
1	00	0	—	Serial Boot viaLINFlex without autobaud
1	01	0	—	Serial Boot viaFlexCAN without autobaud
1	10	0	—	Scan of both serial interfaces (FlexCAN and LINFlex) for Serial Boot with autobaud

1. During reset the boot configuration pins are weak pull down.

When autobaud scan is selected by ABS and FAB pins, an autoscan procedure starts. This procedure listens to the active bus protocol and automatically selects between:

- FlexCAN with autobaud
- LINFlex with autobaud

Note: Autobaud boot code feature is not provided when the device is secured.

34.5.2 SPC560P44Lx, SPC560P50Lx boot pins

The TX/RX pin (LINFlex_0 and FlexCAN_0) used for serial boot and configuration boot pins to select the serial boot mode are described in the *Table 571* for 100-pin and 144-pin LQFP packages.

Table 571. SPC560P44Lx, SPC560P50Lx boot pins

Port pin	Function	Pin	
		100-pin	144-pin
A[2](1)	ABS[0]	57	84
A[3](1)	ABS[2]	64	92
A[4](1)	FAB	75	108
B[0]	CAN_0 TX	76	109
B[1]	CAN_0 RX	77	110
B[2]	LIN_0 TX	79	114
B[3]	LIN_0 RX	80	116

1. Weak pull down during reset.

34.5.3 Reset Configuration Half Word (RCHW)

The SPC560P44Lx, SPC560P50Lx Flash is partitioned into boot sectors as shown in *Table 573*.

Each boot sector contains the Reset Configuration Half-Word (RCHW) at offset 0x00.

Address: Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	VLE	BOOT_ID[0:7]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 646. Reset Configuration Half Word (RCHW)

Table 572. RCHW field descriptions

Field	Description
0-6	Reserved
7 VLE	VLE Indicator This bit configures the MMU for the boot block to execute as either Power Architecture techmology code or as VLE code. 0 Boot code executes as Power Architecture techmology code 1 Boot code executes as VLE code
8-15 BOOT_ID[0:7]	Is valid if its value is 0x5A, then the sector is considered bootable.

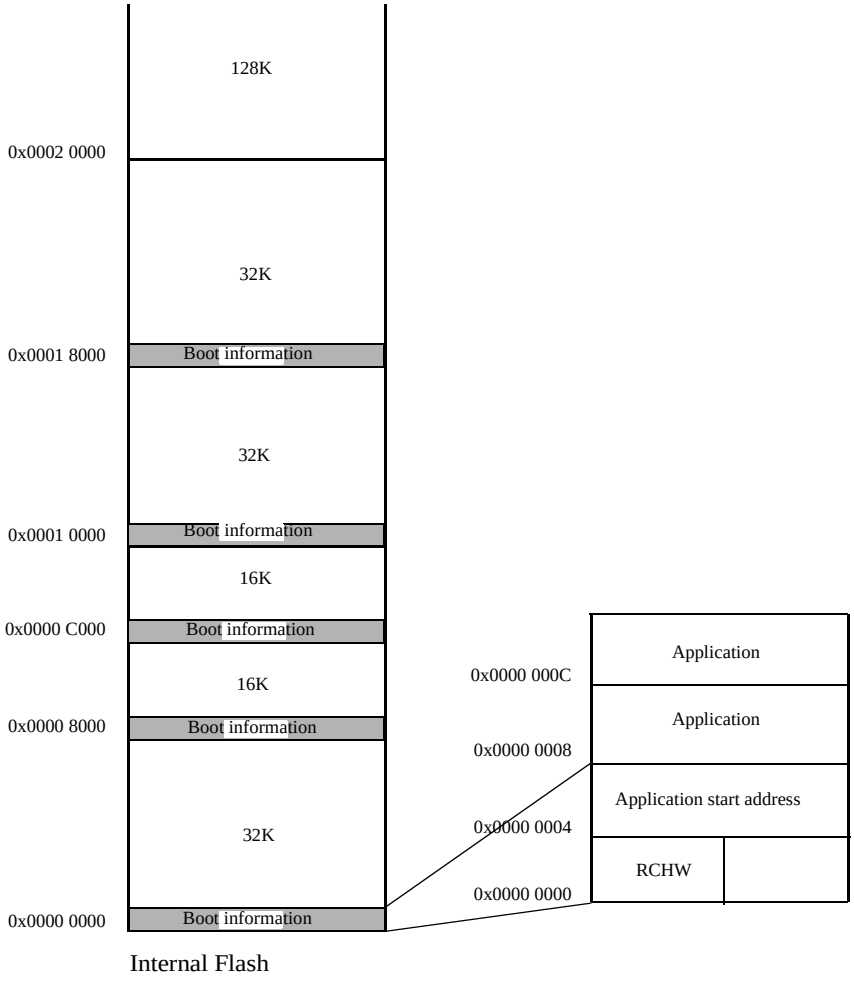


Figure 647. SPC560P44Lx, SPC560P50Lx Flash partitioning and RCHW search

Table 573. Flash boot sector

Block	Address
0	0x0000_0000
1	0x0000_8000
2	0x0000_C000
3	0x0001_0000
4	0x0001_8000

34.5.4 Single chip boot mode

In single chip boot mode, the hardware searches the flash boot sector for a valid boot ID. As soon the device detects a bootable sector, it jumps within this sector and reads the 32-bit word at offset 0x4. This word is the address where the startup code is located (reset boot vector).

Then the device executes this startup code. A user application should have a valid instruction at the reset boot vector address.

If a valid RCHW is not found, the BAM code is executed. In this case BAM moves the SPC560P44Lx, SPC560P50Lx into static mode.

34.5.4.1 Boot and alternate boot

Some applications require an alternate boot sector in the flash so that the main sector in flash can be erased and reprogrammed in the field.

When an alternate boot is needed, the user can create two bootable sectors. The low sector is the main boot sector and the high sector is the alternate boot sector. The alternate boot sector does not need to be consecutive to the main boot sector.

This ensures that even if one boot sector is erased still there will always be another active boot sector:

- Sector shall be activated (i.e., program a valid BOOT_ID instead of 0xFF as initially programmed).
- Sector shall be deactivated writing to 0 some of the bits BOOT_ID bit field (bit1 and/or bit3, and/or bit4, and/or bit6).

34.5.5 Boot through BAM

34.5.5.1 Executing BAM

Single chip boot mode is managed by hardware and BAM does not participate in it.

BAM is executed only on these two following cases:

- Serial boot mode has been selected by FAB pin
- Hardware has not found a valid Boot-ID in any Flash boot locations

If one of these conditions is true, the device fetches code at location 0xFFFF_C000 and BAM application starts.

34.5.5.2 BAM software flow

Figure 648 illustrates the BAM logic flow.

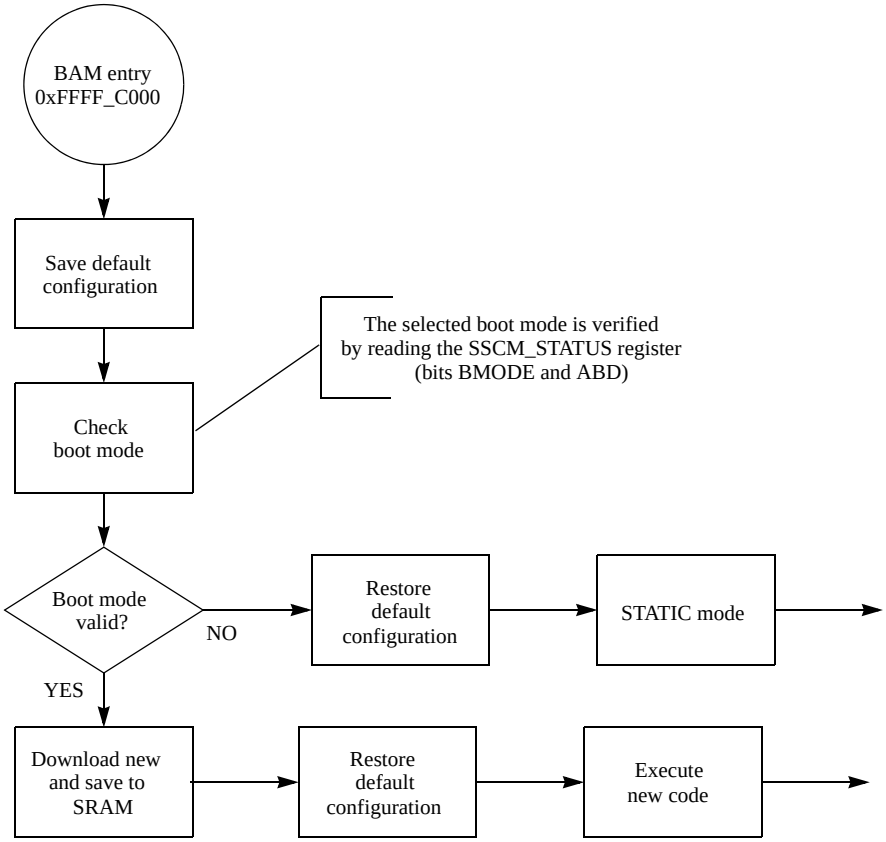


Figure 648. BAM logic flow

The first action is to save the initial device configuration. In this way is possible to restore the initial configuration after downloading the new code but before executing it. This allows the new code being executed as the device was just coming out of reset.

The BMODE and ABD fields of SSCM_STATUS register (see *Section 10.2.2.1: System Status register (STATUS)*) indicate which boot has to be executed (see *Table 574*).

If the BMODE field shows either a single chip value (011) or a reserved value, the boot mode is not considered valid and the BAM pushes the device into static mode.

In all other cases the code of the relative boot is called. Data is downloaded and saved into proper SRAM location.

Table 574. Fields of SSCM STATUS register used by BAM

Field	Description
BMODE	Device Boot Mode 000 FlexRay boot serial boot loader mode (future use) 001 FlexCAN serial boot loader mode 010 SCI serial boot loader mode 011 Single chip mode 100 Expanded chip mode Other values are reserved.
ABD	Autobaud select 0 Autobaud detection is not active. 1 Autobaud detection is active.

Then, the initial device configuration is restored and the code jumps to the address of downloaded code. At this point BAM has just finished its task.

If an error occurs, (e.g., communication error, wrong boot selected, etc.), the BAM restores the default configuration and puts the device into static mode. Static mode means the device enters the low power mode SAFE and the processor executes a wait instruction. This is needed if the device cannot boot in the selected mode. During BAM execution and after, the mode reported by the field S_CURRENT_MODE of the register ME_GS in the module ME Module is “DRUN”.

34.5.5.3 BAM resources

BAM uses/initializes the following MCU resources:

- ME and CGM modules to initialize mode and clock sources
- CAN_0, LINFlex_0, and their pads when performing serial boot mode
- SSCM to check the boot mode and during password check (see *Table 574* and *Figure 649*)
- External oscillator

The following hardware resources are used only when autobaud feature is selected:

- STM to measure the baud rate
- CMU to measure the external clock frequency related to the internal RC clock source
- FMPLL to work with system clock near the maximum allowed frequency (this to have higher resolution during baud rate measurement).

As already mentioned, the initial configuration is restored before executing the downloaded code.

When the autobaud feature is disabled, the system clock is selected directly from the external oscillator. Thus the oscillator frequency defines baud rates for serial interfaces used to download the user application (see *Table 575*).

Table 575. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
f_{extal}	f_{extal} / 833	f_{extal} / 40
8	9600	200 K

Table 575. Serial boot mode without autobaud—baud rates

Crystal frequency (MHz)	LINFlex baud rate (baud)	FlexCAN bit rate (bit/s)
12	14400	300 K
16	19200	400 K
20	24000	500 K
40	48000	1 M

34.5.5.4 Download and execute the new code

From a high level perspective, the download protocol follows these steps:

1. Send message and receive acknowledge message for autobaud or autobit rate selection. (optional step).
2. Send 64-bit password.
3. Send start address, size of downloaded code in bytes, and VLE bit[h].
4. Download data.
5. Execute code from start address.

Each step must be complete before the next step starts.

The step from 2 to 5 are correct if autobaud is disabled. Otherwise, to measure the baud rate, some data is sent from the host to the MCU before step 2 (see *Section 34.6.1: Autobaud feature*).

The communication is done in half duplex manner. Any transmission from the host is followed by the MCU transmission:

1. Host sends data to MCU and start waiting.
2. MCU echoes to host the data received.
3. MCU verifies if echo is correct.
   - If data is correct, the host can continue to send data.
   - If data is not correct, the host stops transmitting and the MCU needs to be reset.

All multi-byte data structures are sent MSB first.

A more detailed description of these steps follows.

34.5.5.5 Download 64-bit password and password check

The first 64 received bits represent the password. This password is sent to the Password Check procedure for verification.

Password check data flow is shown in *Figure 649* where:

- SSCM_STATUS[SEC] = 1 means flash secured
- SSCM_STATUS[PUB] = 1 means flash with public access.

h. Since the device supports only VLE code and does not support Book E code, this flag is used only for backward compatibility.

In case of flash with public access, the received password is compared with the public password 0xFEED_FACE_CAFE_BEEF.

If public access is not allowed but the flash is not secured, the received password is compared with the value saved on NVPWD0 and NVPWD1 registers.

In uncensored devices, it is possible to download code via LINFlex or FlexCAN (serial boot mode) into internal SRAM with any 64-bit private password stored in the flash and provided during the boot sequence.

In the previous cases, comparison is done by the BAM application. If it goes wrong, BAM pushes the device into static mode.

In case of public access not allowed and flash secured, the password is written into SSCM[PWCMPH/L] registers.

In case of flash secured with public access not allowed (user password configured in the NVPWD0 and NVPWD1 registers), the user must set the swapped password: NVPWD1 and NVPWD0 to access the device (refer to following examples).

Example 1

In devices with flash secured, registers are programmed:

NVPWD0= 0x87654321

NVPWD1= 0x12345678

NVSCI0= 0x55AA1111

NVSCI1= 0x55AA1111

To download the code via SLB, the provided password is 0x1234_5678_8765_4321 (swapped WITH respect to NVPWD0/NVPWD1 —> NVPWD1/NVPWD0).

Example 2

In devices with flash NOT secured, registers are programmed:

NVPWD0= 0x87654321

NVPWD1= 0x12345678

NVSCI0= 0x55AA55AA

NVSCI1= 0x55AA55AA

To download the code via SLB the provided password is 0x8765_4321_1234_5678 (as expected from NVPWD0/NVPWD1).

After a fixed time waiting, comparison is done by hardware. Then BAM again verifies the SEC flag in SSCM_STATUS:

- SEC = 0, flash is now unsecured and BAM continues its task
- SEC = 1, flash is still secured because password was wrong; BAM puts the device into static mode.

This fixed time depends on external crystal oscillator frequency (XOSC). With XOSC of 12 MHz, fixed time is 350 ms.

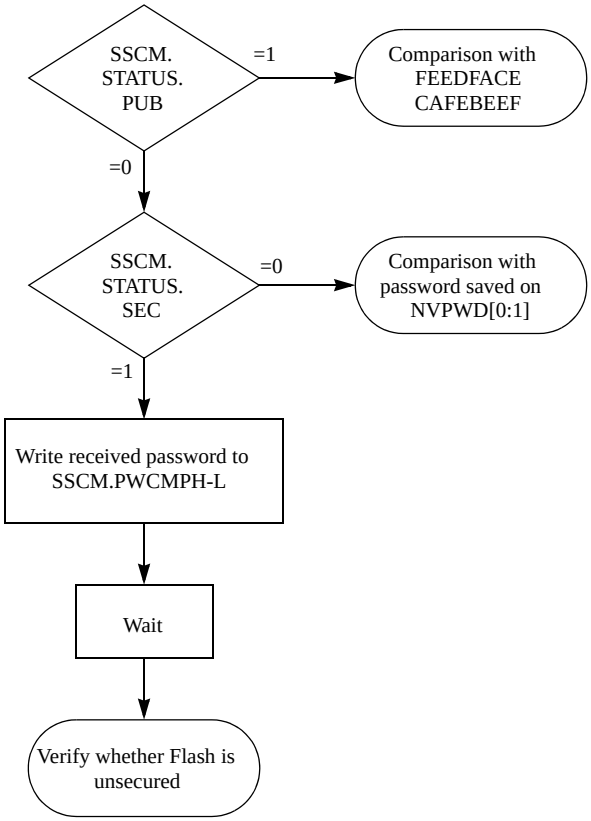


Figure 649. Password check flow

34.5.5.6 Download start address, VLE bit and code size

The next 8 bytes received by the MCU contain a 32-bit Start Address, the VLE mode bit and a 31-bit code Length as shown in *Figure 650*.

The VLE bit (Variable Length Instruction) indicates which instruction set for which the code has been compiled. This device family supports only VLE = 1. The bit is used for backward compatibility.

The Start Address defines where the received data will be stored and where the MCU will branch after the download is finished. The two LSB bits of the Start Address are ignored by the BAM program, such that the loaded code should be 32-bit word aligned.

The Length defines how many data bytes have to be loaded.

START_ADDRESS[31:16]

START_ADDRESS[15:0]

VLE	CODE_LENGTH[30:16]

CODE_LENGTH[15:0]

Figure 650. Start address, VLE bit and download size in bytes

34.5.5.7 Download data

Each byte of data received is stored into device's SRAM, starting from the address specified in the previous protocol step.

The address increments until the number of bytes of data received matches the number of bytes specified in the previous protocol step.

Since the SRAM is protected by 32-bit wide Error Correction Code (ECC), BAM always writes bytes into SRAM grouped into 32-bit words. If the last byte received does not fall onto a 32-bit boundary, BAM fills it with 0 bytes.

Then a "dummy" word (0x0000_0000) is written to avoid ECC error during core prefetch.

34.5.5.8 Execute code

The BAM program waits for the last echo message transmission being completed.

Then it restores the initial MCU configuration and jumps to the loaded code at Start Address that was received in step 3 of the protocol.

At this point BAM has finished its tasks and MCU is controlled by new code executing from SRAM.

34.5.6 Boot from UART—autobaud disabled

34.5.6.1 Configuration

Boot from UART protocol is implemented by LINFlex_0 module. Pins used are:

- LINFlex_TX corresponds to pin B[2]
- LINFlex_RX corresponds to pin B[3].

When autobaud feature is disabled, the system clock is driven by external oscillator.

LINFlex controller is configured to operate at a baud rate = system clock frequency/833 (see *Table 575* for baud rate example), using 8-bit data frame without parity bit and 1 stop bit.

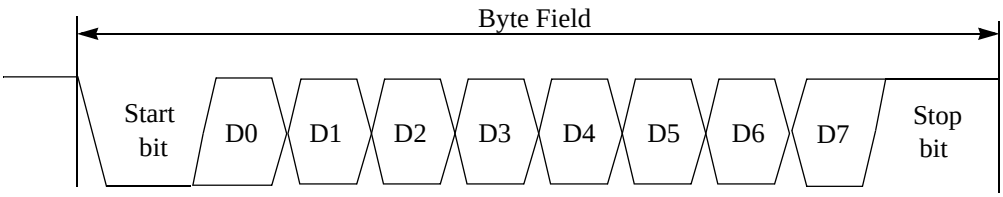


Figure 651. LINFlex bit timing in UART mode

34.5.6.2 UART boot mode download protocol

Table 576 summarizes the download protocol and BAM action during the UART boot mode.

Table 576. UART boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	64-bit password (MSB first)	64-bit password	Password checked for validity and compared against stored password.
2	32-bit store address	32-bit store address	Load address is stored for future use.
3	VLE bit + 31-bit number of bytes (MSB first)	VLE bit + 31-bit number of bytes (MSB first)	Size of download is stored for future use. Verify if VLE bit is set to 1.
4	8 bits of raw binary data	8 bits of raw binary data	8-bit data are packed into 32-bit word. This word is saved into SRAM starting from the “Load address”. “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
5	none	none	Branch to downloaded code.

34.5.7 Bootstrap with FlexCAN—autobaud disabled

34.5.7.1 Configuration

Boot from FlexCAN protocol is implemented by the FlexCAN_0 module. Pins used are:

- CAN_TX corresponds to pin B[0]
- CAN_RX corresponds to pin B[1].

Boot from FlexCAN with autobaud disabled uses system clock driven by the external oscillator.

The FlexCAN controller is configured to operate at a baud rate equal to the system clock frequency/40 (see *Table 575* for examples of baud rate).

It uses the standard 11-bit identifier format detailed in the FlexCAN 2.0A specification.

FlexCAN controller bit timing is programmed with 10 time quanta, and the sample point is 2 time quanta before the end, as shown in *Figure 652*.

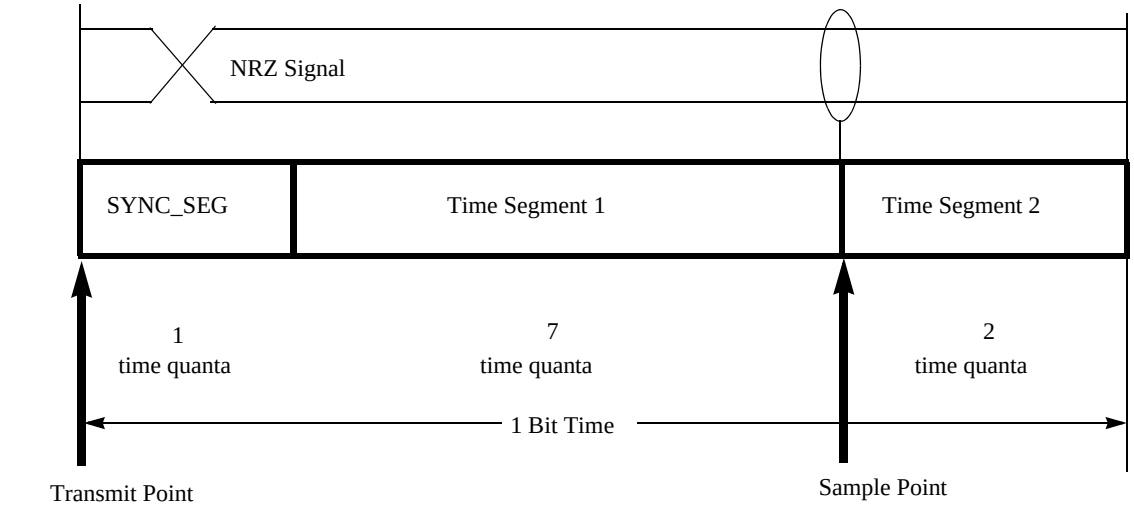


Figure 652. FlexCAN bit timing

34.6 FlexCAN boot mode download protocol

Table 577 summarizes the download protocol and BAM action during the FlexCAN boot mode. All data are transmitted bytewise.

Table 577. FlexCAN boot mode download protocol (autobaud disabled)

Protocol step	Host sent message	BAM response message	Action
1	FlexCAN ID 0x011 + 64-bit password	FlexCAN ID 0x001 + 64-bit password	Password checked for validity and compared against stored password.
2	FlexCAN ID 0x012 + 32-bit store address + VLE bit+ 31-bit number of bytes	FlexCAN ID 0x002 + 32-bit store address + VLE bit + 31-bit number of bytes	Load address is stored for future use. Size of download is stored for future use. Verify if VLE bit is set to 1.
3	FlexCAN ID 0x013 + 8 to 64 bits of raw binary data	FlexCAN ID 0x003 + 8 to 64 bits of raw binary data	8-bit data are packed into 32-bit words. These words are saved into SRAM starting from the “Load address”. “Load address” increments until the number of data received and stored matches the size as specified in the previous step.
4	none	none	Branch to downloaded code.

34.6.1 Autobaud feature

The autobaud feature allows boot operation with a wide range of baud rates independent of the external oscillator frequency.

34.6.1.1 Configuration

SPC560P44Lx, SPC560P50Lx devices implement the autobaud feature via FlexCAN or LINFlex selecting the active serial communication peripheral by means of an autoscan routine.

When autobaud configuration is selected by ABS and FAB pins, the autoscan routine starts and listens to the active bus protocol. Initially the LinFlex_0 RX pin and FlexCAN_0 RX pin are configured as GPIO inputs:

- for 144-pin, and 100-pin LQFP packages internal weak pull-up enabled for both RX pins

The autoscan routine waits in polling for the first LOW level to select which routine will be executed:

- FlexCAN Autobaud routine
- LinFlex Autobaud routine

Then the measurement baud rate is computed to configure the serial communication at the right rate. In the end of baud rate measurement, LinFlex_0 RX pin and FlexCAN_0 RX pin switches to work as dedicated pin.

Baud rate measurement is using the System Timer Module (STM) which is driven by the system clock. Measurement itself is performed by software polling the related inputs as general purpose IO's, resulting in a detection granularity that is directly related to the execution speed of the software.

One main difference of the autobaud feature is that the system clock is not driven directly by the external oscillator, but it is driven by the FMPLL output. The reason is that to have an optimum resolution for baud rate measurement, the system clock needs to be nearer to the maximum allowed device's frequency.

This is achieved with the following two steps:

1. using the Clock Monitor Unit (CMU) and the internal RC oscillator (IRC), the external frequency is measured using the IRC as reference to determine this frequency.
2. Based on the result of this measurement, the FMPLL is programmed to generate a system clock that is configured to be near, but lower, to the maximum allowed frequency.

The relation between system clock frequency and external clock frequency with FMPLL configuration value is shown in *Table 578*.

Table 578. System clock frequency related to external clock frequency

f_{osc} [MHz]	f_{rc}/f_{osc} (1)	f_{sys} [MHz]
4–8	4–2	16–32
8–12	2–4/3	32–48
12–16	4/3–1	36–48
16–24	1–2/3	32–48
> 24	< 2/3	> 24

1. These values and consequently the f_{sys} suffer from the precision of RC internal oscillator used to measure f_{osc} through CMU module.

After setting up the system clock, the BAM autoscan code configures the FlexCAN RX pin (B[1] on all packages) and LINFlex RX pin (B[3] on LQFP100 or B[7] on LQFP64) as GPIO inputs and searches for FlexCAN RX pin level to verify if CAN is connected or not.

Then continuously waits in polling on change of RX pins level.The FlexCAN RX pin level takes precedence. First signal found at low level selects the serial boot routine that will be executed.

In case a low level is detected on any input, the corresponding autobaud measurement functionality is started:

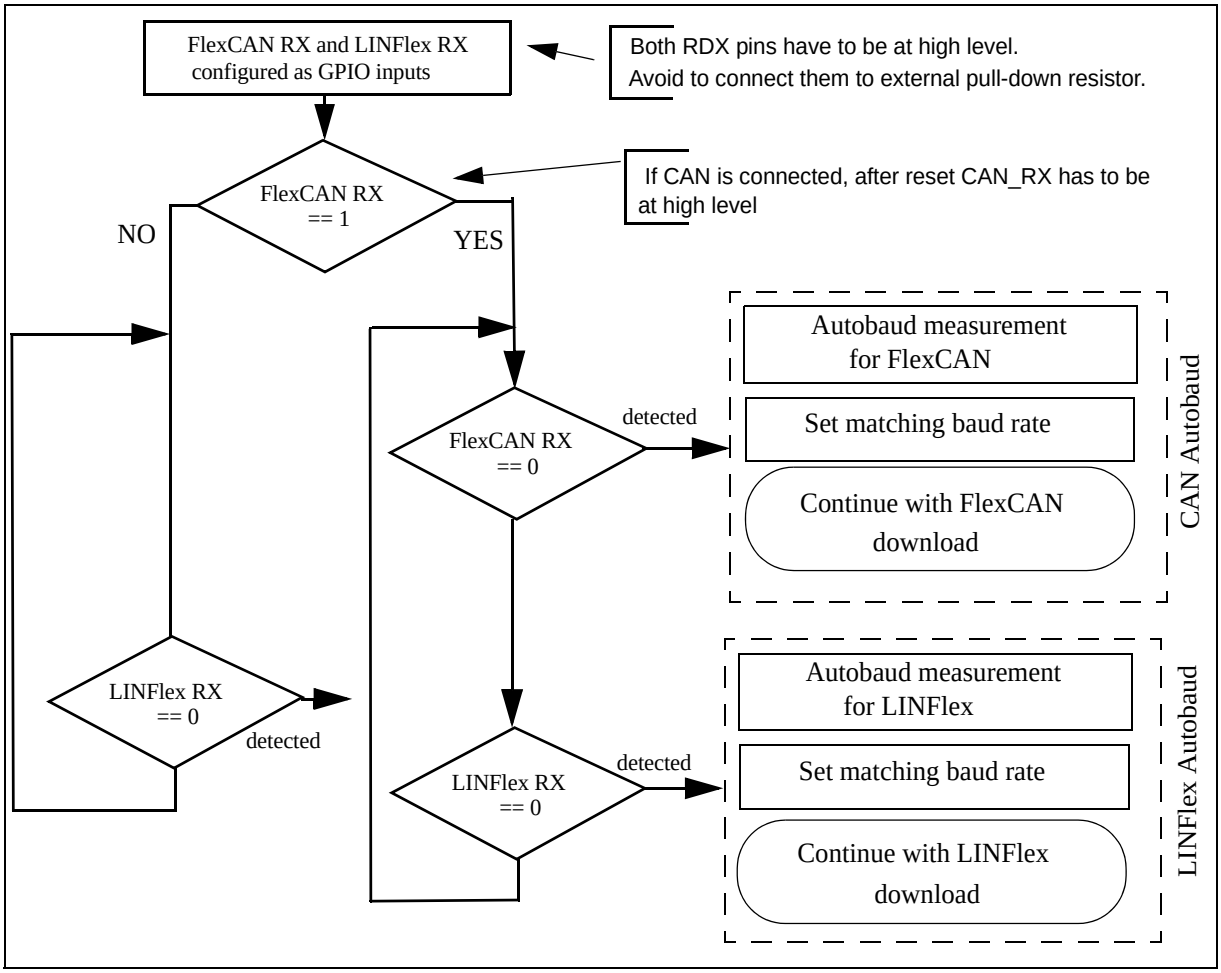
- when FlexCAN RX (corresponds to pin B[1]) level is low, the CAN autobaud measurement starts and then sets up the FlexCAN baud rate accordingly;
- when UART RX (corresponds to pin B[3] on LQFP100 or B[7] on LQFP64) level is low, the UART autobaud measurement starts and then sets up the LINFlex baud rate accordingly.

After performing the autobaud measurement and setting up the baud rate, the corresponding RX input is reconfigured and the related standard download process is started; in case of a detected CAN transmission a download using the CAN protocol as described in *Section 34.5.7: Bootstrap with FlexCAN—autobaud disabled*, and in case of a detected UART transmission a download using the UART protocol as described in *Section 34.5.6: Boot from UART—autobaud disabled*.

The following *Figure 653* identifies the corresponding flow and steps.

Note: When autobaud scan is selected, initially both LINFlex_0 RX pin and FlexCAN_0 RX pin should be at high level. No external circuity should pull-down them to allow right autoscan.

Figure 653. BAM Autoscan code flow



34.6.1.2 Boot from UART with autobaud enabled

The only difference between booting from UART with autobaud enabled and booting from UART with autobaud disabled is that a further byte is sent from the host to the MCU when autobaud is enabled. The value of that byte is 0x00.

This first byte measures the time from falling edge and rising edge. The baud rate can be calculated from this time.

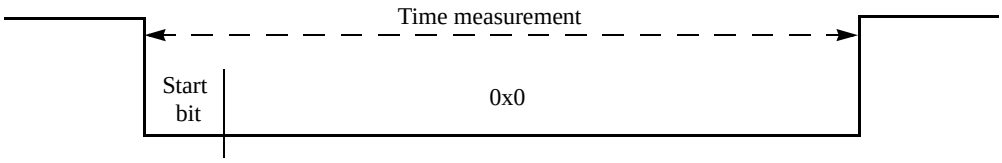


Figure 654. Baud measurement on UART boot

Initially the UART RX pin is configured as GPIO input and it waits in polling for the first falling edge, then STM starts. UART RX pin waits again for the first rising. Then STM stops and from its measurement baud rate is computed.

The LINFlex module is configured to work in UART mode with the calculated baud rate. Then an acknowledge byte (0x59, ASCII char “Y”) is sent.

From this point, the BAM follows the normal UART mode boot protocol (see *Figure 655*).

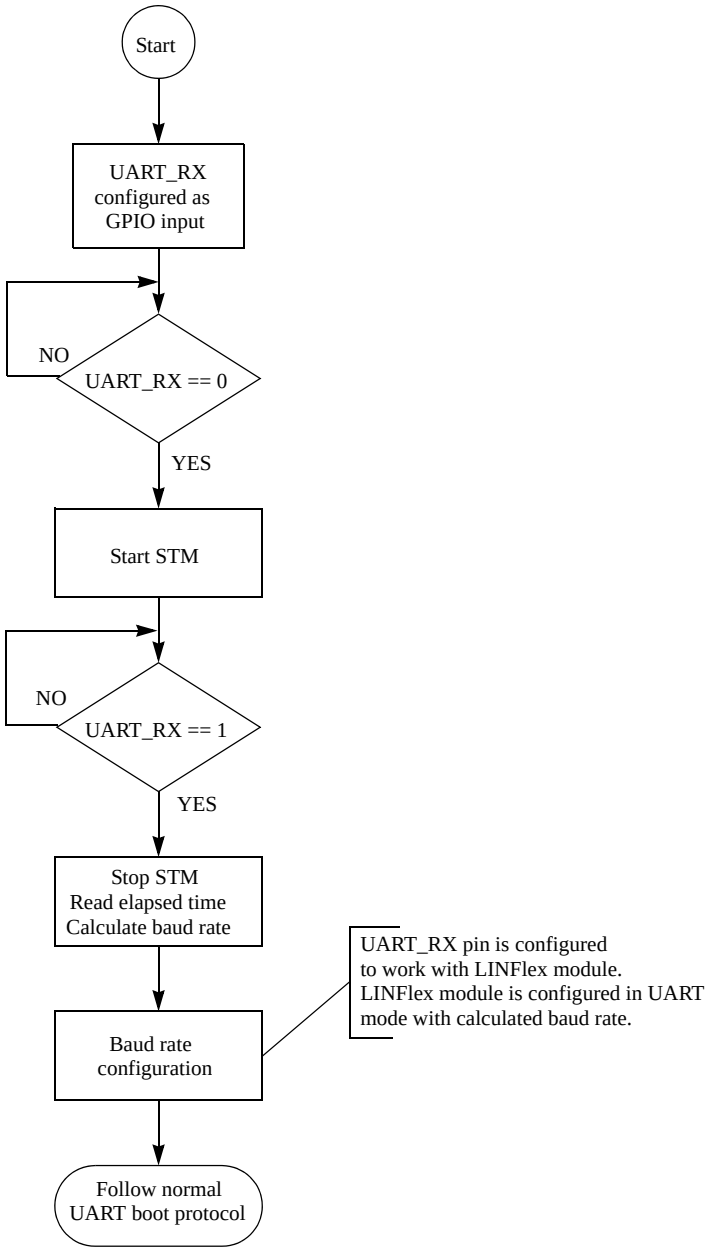


Figure 655. BAM rate measurement flow during UART boot

34.6.1.2.1 Choosing the host baud rate

The calculation of the UART baud rate from the length of the first 0 byte that is received, allows the operation of the boot loader with a wide range of baud rates. However, to ensure proper data transfer, the upper and lower limits have to be kept.

SCI autobaud rate feature operates by polling the LINFlex_RX pin for a low signal. The length of time until the next low to high transition is measured using the System Timer Module (STM) time base. This high-low-high transition is expected to be a zero byte: a start bit (low) followed by eight data bits (low) followed by a stop bit (high).

Upon reception of a zero byte and configuration of the baud rate, an acknowledge byte is returned to the host using the selected baud rate.

Time base is enabled at reception of first low bit, disabled and read at reception of next high bit. Error introduced due to polling will be small (typically < 6 cycles).

The following equation gives the relation between baud rate and LINFlex register configuration:

Equation 67

$$LDIV = \frac{F_{cpu}}{16 \cdot baudrate}$$

LDIV is an unsigned fixed point number and its mantissa is coded into 13 bits of the LINFlex's register LINIBRR.

From this equation and considering that a single UART transmission contains 9 bits, it is possible to obtain the connection between time base measured by STM and LINIBB register:

Equation 68

$$LINIBRR = \frac{timebase}{144}$$

To minimize errors in baud rate, a large external oscillator frequency value and low baud rate signal are preferred.

Example 3 Baud rate calculation for 24 kBaud signal

Considering a 24 kbaud signal and the device operating with 20 MHz external frequency.

Over 9 bits the STM will measure: (9 × 20 MHz)/24 kbaud = 7497 cycles.

Error expected to be approximately ± 6 cycles due to polling.

Thus, LINIBB will be set to 52 (rounding required). This results in a baud rate of 24.038 kbaud. Error of < 0.2%.

To maintain the maximum deviation between host and calculated baud rate, recommendations for the user are listed *Table 579*.

Table 579. Maximum and minimum recommended baud rates

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 2.5% deviation
4	13.4 Kbit/s (SBR = 19)	30 bit/s (SBR = 8192)
8	26.9 Kbit/s (SBR = 19)	60 bit/s (SBR = 8192)
12	38.4 Kbit/s (SBR = 20)	90 bit/s (SBR = 8192)

Table 579. Maximum and minimum recommended baud rates(Continued)

$f_{sys} = f_{xtal}$ (MHz)	Max baud rate for guaranteed < 2.5% deviation	Min baud rate for guaranteed < 2.5% deviation
16	51.2 Kbit/s (SBR = 20)	120 bit/s (SBR = 8192)
20	64.0 Kbit/s (SBR = 20)	150 bit/s (SBR = 8192)

Higher baud rates high may be used, but the user will be required to ensure they fall within an acceptable error range. This is illustrated in *Figure 656*, which shows the effect of quantization error on the baud rate selection.

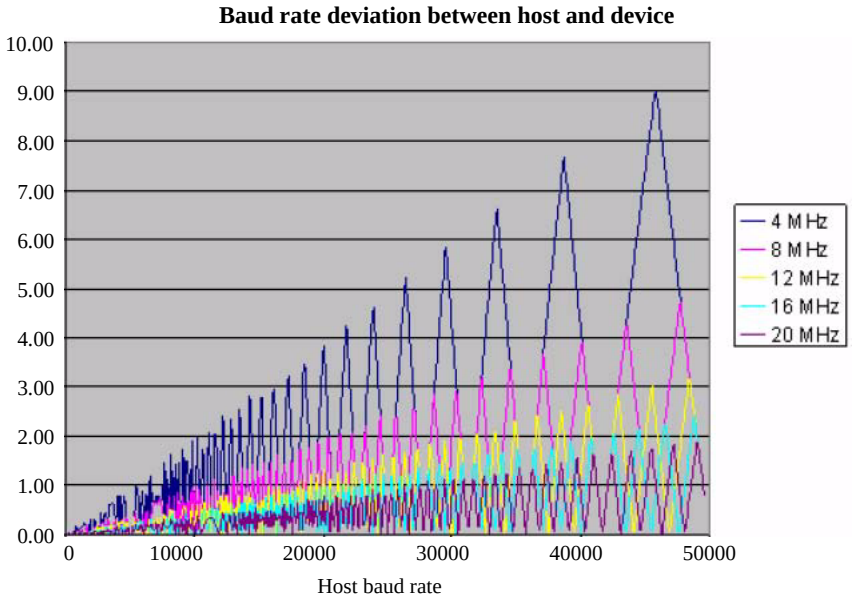


Figure 656. Baud rate deviation between host and SPC560P44Lx, SPC560P50Lx

Example 4 Baud rate calculation for 250 kBaud signal

Considering reception of a 250kBaud signal from the host and SPC560P44Lx, SPC560P50Lx operating with a 4 MHz oscillator. Over 9 bits the time base will measure: (9 × 4e6)/250e3 = 144 cycles.

Thus, LINIBB is set to 144/144 = 1. This results in a baud rate of exactly 250 kBd.

However, a slower 225 kBd signal operating with 4 MHz XTAL would again result in LINIBB = 1, but this time with an 11.1% deviation.

34.6.1.3 Boot from FlexCAN with autobaud enabled

The only difference between booting from FlexCAN with autobaud enabled and booting from FlexCAN with autobaud disabled is that the following initialization FlexCAN frame is sent for baud measurement purposes from the host to the MCU when autobaud is enabled:

- Standard identifier = 0x0,
- Data Length Code (DLC) = 0x0.

As all the bits to be transmitted are dominant bits, there is a succession of 5 dominant bits and 1 stuff bit on the FlexCAN network (see *Figure 657*).

From the duration of this frame, the MCU calculates the corresponding baud rate factor with respect to the current CPU clock and initializes the FlexCAN interface accordingly.

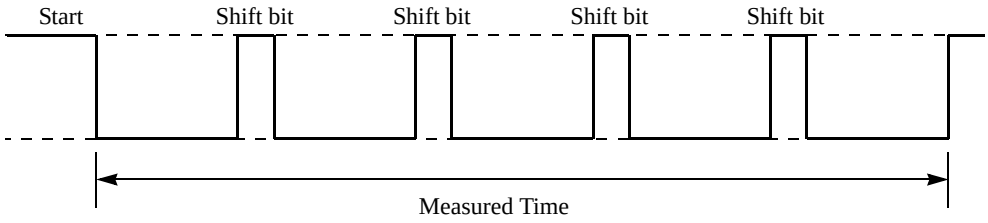


Figure 657. Bit time measure

In FlexCAN boot mode, the FlexCAN RX pin is first configured to work as a GPIO input. In the end of baud rate measurement, it switches to work with the FlexCAN module.

The baud rate measurement flow is detailed in *Figure 658*.

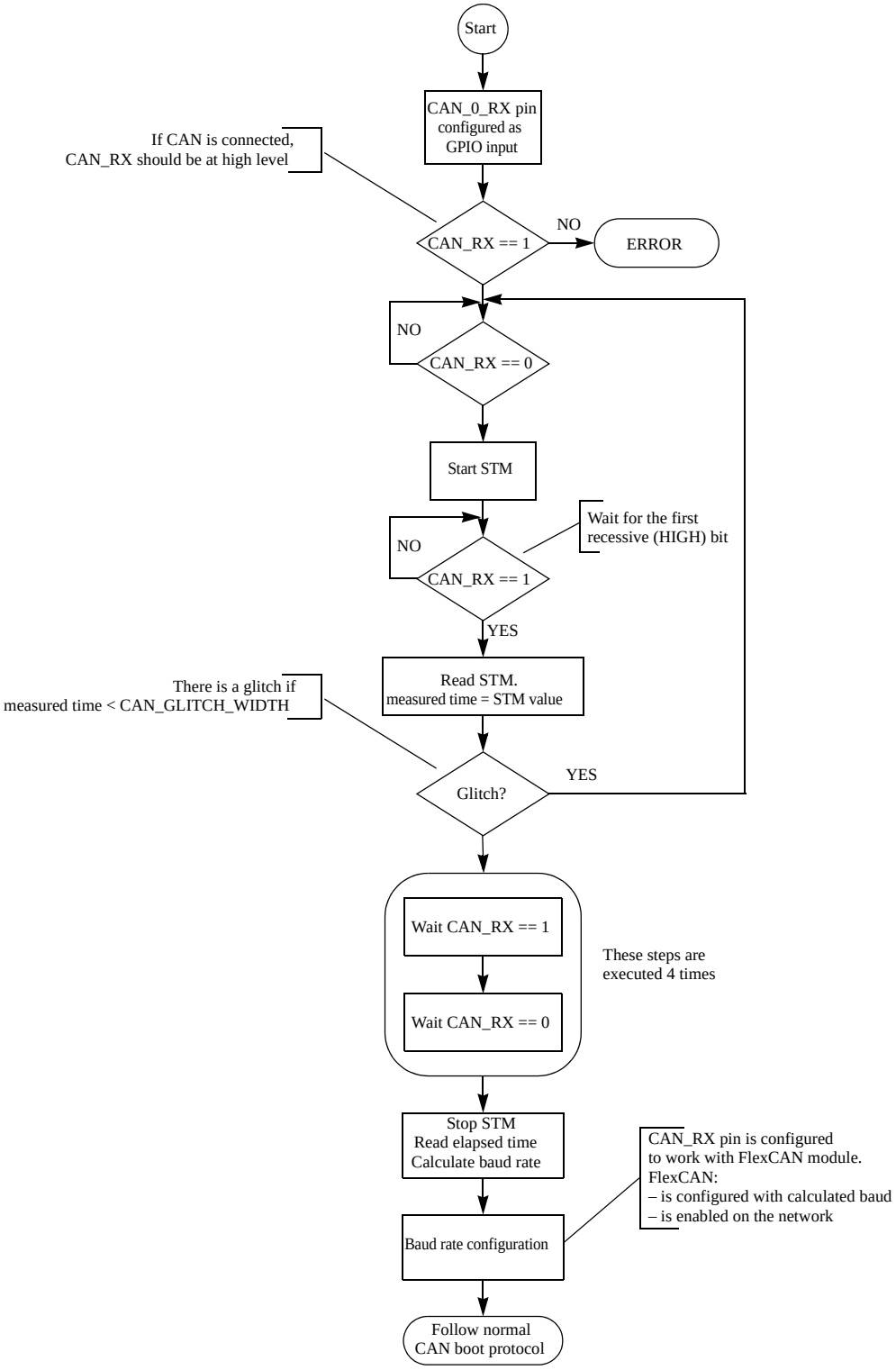


Figure 658. BAM rate measurement flow during FlexCAN boot

34.6.1.3.1 Choosing the host baud rate

The calculation of the FlexCAN baud rate allows the operation of the boot loader with a wide range of baud rates. However, to ensure proper data transfer, the upper and lower limits have to be kept.

Pins are measured until reception of the 5th recessive bit. Thus a total of 29 bit-times are measured (see *Figure 657*).

When calculating bit times and prescalers, to minimize any errors, ideally operate with the minimum system clock prescaler divider (CAN_CR[PRESDIV]) and maximum number of time quanta possible.

After measuring the 29 bit times, the results stored in the STM time base are used to select PRESDIV. The number of time quanta in a FlexCAN bit time is given by:

Note: *Bit_time = SYNCSEG + TSEG1 + TESG2*

SYNCSEG = Exactly one time quantum.

TSEG1 = PROGPSEG + PSEG1 + 2

TSEG2 = PSEG2 + 1

Time base result = 29 × (Presdiv+1) × (SYNCSEG + TSEG1 + TSEG2)

FlexCAN protocol specifies that the FlexCAN bit timing should comprise a minimum of 8 time quanta and a maximum of 25 time quanta. Therefore, the available range is:

$8 \leq 1 + TSEG1 + TESG2 \leq 25$

For 29 bit times, the possible range in which the result in the time base may lie, accounting for PRESDIV, is:

$(232 \times (1 + PRESDIV)) \leq time\ base \leq (725 \times (1 + PRESDIV))$

Therefore, the available values of the time base can be divided into windows of 725 counts.

Table 580. Prescaler/divider and time base values

PRESDIV	Time base Minimum	Time base Maximum
0	232	725
1	726	1450
2	1451	2175
3	2176	2900

In the BAM, the time base is divided by 726, the remainder is discarded. The result provides the CAN_CR[PRESDIV] to be selected.

To help compensate for any error in the calculated baud rate, the resynchronization jump width will be increased from its default value of 1 to a fixed value of 2 time quanta.This is the maximum value allowed that can accommodate all permissible can baud rates. See *Table 581*.

Table 581. FlexCAN standard compliant bit timing segment settings

Time Segment 1	Time Segment 2	RJW
5..10	2	1..2
4..11	3	1..3
5..12	4	1..4
6..13	5	1..4
7..14	6	1..4
8..15	7	1..4
9..16	8	1..4

Timing segment 2 is kept as large as possible to keep sample time within bit time.

Table 582. Lookup table for FlexCAN bit timings

Desired number of Time quanta (DTq)	Time Segment 2	Time segment 1	
	PSEG2+1	PSEG1+1	PROPSEG+1
8 to 13(1)	2	2	DTq-5
8 to 13(2)	3	2	DTq-6
14 to 15	3	3	DTq-6
16 to 17	4	4	DTq-7
18 to 19	5	5	DTq-8
20 to 21	6	6	DTq-9
22 to 23	7	7	DTq-10
24 to 25	8	8	DTq-11

1. PRESDIV+1 > 1
2. PRESDIV+1 = 1 (to accommodate information processing time IPT of 3 tq) Note: All TSEG1 and TSEG2 times have been chosen to preserve a sample time between 70% and 85% of the bit time.

Table 583. PRESDIV + 1 = 1

Desired number of time quanta	Register contents for CANA_CR
8	0x004A_2001
9	0x004A_2002
10	0x004A_2003
11	0x004A_2004
12	0x004A_2005
13	0x004A_2006

Table 584. PRESDIV + 1 > 1 (YY = PRESDIV)

Desired number of time quanta	Register contents for CANA_CR
8	0xYY49_2002
9	0xYY49_2003
10	0xYY49_2004
11	0xYY49_2005
12	0xYY49_2006
13	0xYY49_2007
14	0xYY52_2007
15	0xYY52_2008
16	0xYY5B_2008
17	0xYY5B_2009
18	0xYY64_2009
19	0xYY64_200A
20	0xYY6D_200A
21	0xYY6D_200B
22	0xYY76_200B
23	0xYY76_200C
24	0xYY7F_200C
25	0xYY7F_200D

Worked examples showing FlexCAN Autobaud rate:

Example 5 8 MHz crystal

Consider case where using an 8 MHz crystal, user attempts to send 1 MB (max permissible baud rate) FlexCAN message.

- Time base, clocking at crystal frequency, would measure:
- 1MB = 8 clocks/bit => 29 * 8 = 232 clocks
- To calculate PRESDIV = 232/725 =>PRESDIV = 0
- To calculate time quanta requirement:
- Time base result = 29 *(Presdiv+1) * (SYNCSEG + TSEG1 + TSEG2)
- 232 = 29 * 1 * (1 + TSEG1 + TSEG2)
- 1 + TSEG1 + TSEG2 = 8.
- From the lookup table, CANA_CR = 0x004A_2001.
- This give a baud rate of X. This give 0% error.

Example 6 20 MHz crystal

Consider case where using a 20 MHz crystal, user attempts to send 62.5 Kb/s FlexCAN message.

- Time base, clocking at crystal frequency, would measure:
- 62.5 Kb/s = 320 clocks/bit => 29 * 320 = 9280 clocks
- To calculate PRESDIV = 9280/725 = 12r580 => PRESDIV = 12
- To calculate time quanta requirement:
- Time base result = 29 x (Presdiv+1) x (SYNCSEG + TSEG1 + TSEG2)
- 9280 = 29 x 13 x (1 + TSEG1 + TSEG2)
- 1 + TSEG1 + TSEG2 = 24.6~ 25 (need to round up - S/W must track remainder)
- From the lookup table, CANA_CR = 0x0C7F_200D.
- This give a baud rate of 61.538k Baud
- This equates to an error of: ~1.6%

This excludes cycle count error introduced due to software polling likely to be ~6 system clocks.

34.6.2 Interrupt

No interrupts are generated by or are enabled by the BAM.

34.7 Censorship

Censorship can be enabled to protect the contents of the flash memory from being read or modified. In order to achieve this, the censorship mechanism controls access to the:

- JTAG / Nexus debug interface
- Serial boot mode (which could otherwise be used to download and execute code to query or modify the flash memory)

To re-gain access to the flash memory via JTAG or serial boot, a 64-bit password must be correctly entered.

Caution: When censorship has been enabled, the only way to regain access is with the password. If this is forgotten or not correctly configured, then there is no way back into the device.

There are two 64-bit values stored in the shadow flash which control the censorship (see *Table 147* for a full description):

- Nonvolatile Private Censorship Password registers, NVPWD0 and NVPWD1
- Nonvolatile System Censorship Control registers, NVSCI0 and NVSCI1

34.7.0.1 Censorship password registers (NVPWD0 and NVPWD1)

The two private password registers combine to form a 64-bit password that should be programmed to a value known only by you. After factory test these registers are programmed as shown below:

- NVPWD0 = 0xFEED_FACE
- NVPWD1 = 0xCAFE_BEEF

This means that even if censorship was inadvertently enabled by writing to the censorship control registers, there is an opportunity to get back into the microcontroller using the default private password of 0xFEED_FACE_CAFE_BEEF.

When configuring the private password, each half word (16-bit) must contain at least one "1" and one "0". Some examples of legal and illegal passwords are shown in *Table 585*:

Table 585. Examples of legal and illegal passwords

Legal (valid) passwords	Illegal (invalid) passwords
0x0001_0001_0001_0001 0xFFFE_FFFE_FFFE_FFFE 0x1XXX_X2XX_XX4X_XXX8	0x0000_XXXX_XXXX_XXXX 0xFFFF_XXXX_XXXX_XXXX

In uncensored devices it is possible to download code via LINFlex or FlexCAN (Serial Boot Mode) into internal SRAM even if the 64-bit private password stored in the flash and provided during the boot sequence is a password that does not conform to the password rules.

34.7.0.2 Nonvolatile System Censorship Control registers (NVSCI0 and NVSCI1)

These registers are used together to define the censorship configuration. After factory test these registers are programmed as shown below which disables censorship:

- NVSCI0 = 0x55AA_55AA
- NVSCI1 = 0x55AA_55AA

Each 32-bit register is split into an upper and lower 16-bit field. The upper 16 bits (the SC field) are used to control serial boot mode censorship. The lower 16 bits (the CW field) are used to control flash memory boot censorship.

Caution: If the contents of the shadow flash memory are erased and the NVSCI0,1 registers are not re-programmed to a valid value, the microcontroller will be permanently censored with no way for you to regain access. A microcontroller in this state cannot be debugged or re-flashed.

34.7.0.3 Censorship configuration

The steps to configuring censorship are:

1. Define a valid 64-bit password that conforms to the password rules.
2. Using the table and flow charts below, decide what level of censorship you require and configure the NVSCI0,1 values.
3. Re-program the shadow flash memory and NVPWD0,1 and NVSCI0,1 registers with your new values. A POR is required before these will take effect.

Caution: If
(NVSCI0 and NVSCI1 do not match)
or
(Either NVSCI0 or NVSCI1 is not set to 0x55AA)
then the microcontroller will be permanently censored with no way to get back in.

Table 586 shows all the possible modes of censorship. The red shaded areas are to be avoided as these show the configuration for a device that is permanently locked out. If you wish to enable censorship with a private password there is only one valid configuration — to

modify the CW field in both NVSCI0,1 registers so they match but do not equal 0x55AA. This will allow you to enter the private password in both serial and flash boot modes.

Table 586. Censorship configuration and truth table

Boot configuration		Serial censorship control word (NVSCI*n*[SC])	Censorship control word (NVSCI*n*[CW])	Internal flash memory state	Nexus state	Serial password	JTAG password
FAB pin state	Control options						
0 (flash memory boot)	Uncensored	0xXXXX AND NVSCI0 == NVSCI1	0x55AA AND NVSCI0 == NVSCI1	Enabled	Enabled		N/A
	Private flash memory password and censored	0x55AA AND NVSCI0 == NVSCI1	!0x55AA AND NVSCI0 == NVSCI1	Enabled	Enabled with password		NVPWD**1,0** (SSCM reads flash memory(1))
	Censored with no password access (lockout)	!0x55AA	!0X55AA	Enabled	Disabled		N/A
		OR NVSCI0 != NVSCI1					
1 (serial boot)	Private flash memory password and uncensored	0x55AA AND NVSCI0 == NVSCI1		Enabled	Enabled	NVPWD**0,1** (BAM reads flash memory(1))	
	Private flash memory password and censored	0x55AA AND NVSCI0 == NVSCI1	!0x55AA AND NVSCI0 == NVSCI1	Enabled	Disabled	NVPWD**1,0** (SSCM reads flash memory(1))	
	Public password and uncensored	!0x55AA AND NVSCI0 != NVSCI1	0X55AA AND NVSCI0 != NVSCI1	Enabled	Enabled	Public (0xFEED_FACE_CAFE_BEEF)	
	Public password and censored (lockout)	!0x55AA		Disabled	Disabled	Public (0xFEED_FACE_CAFE_BEEF)	
		OR NVSCI0 != NVSCI1					

= Microcontroller permanently locked out

= Not applicable

1. When the SSCM reads the passwords from flash memory, the NVPWD0 and NVPWD1 password order is swapped, so you have to submit the 64-bit password as {NVPWD1, NVPWD0}.

The flow charts in *Figure 659* and *Figure 660* provide a way to quickly check what will happen with different configurations of the NVSCI0,1 registers as well as detailing the correct way to enter the serial password. In the password examples, assume the 64-bit password has been programmed into the shadow flash memory in the order {NVPWD0, NWPWD1} and has a value of 0x01234567_89ABCDEF.

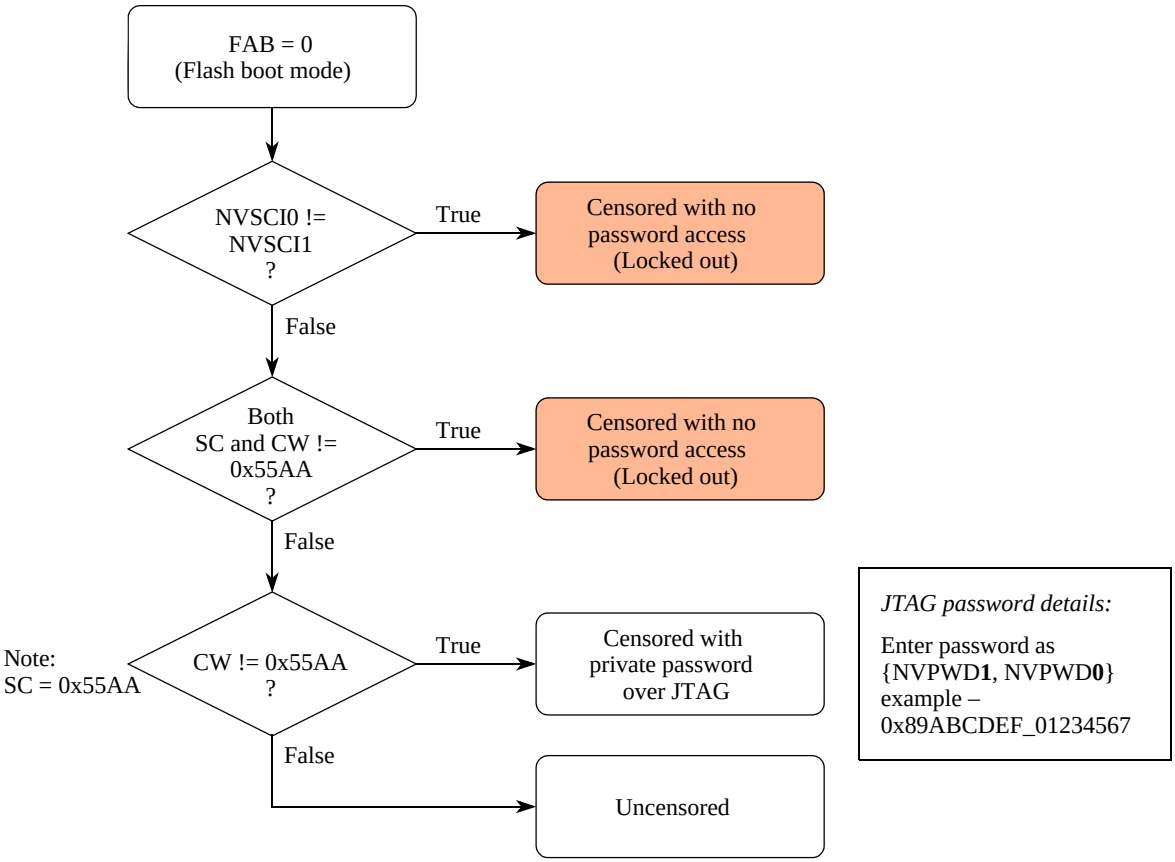


Figure 659. Censorship control in flash memory boot mode

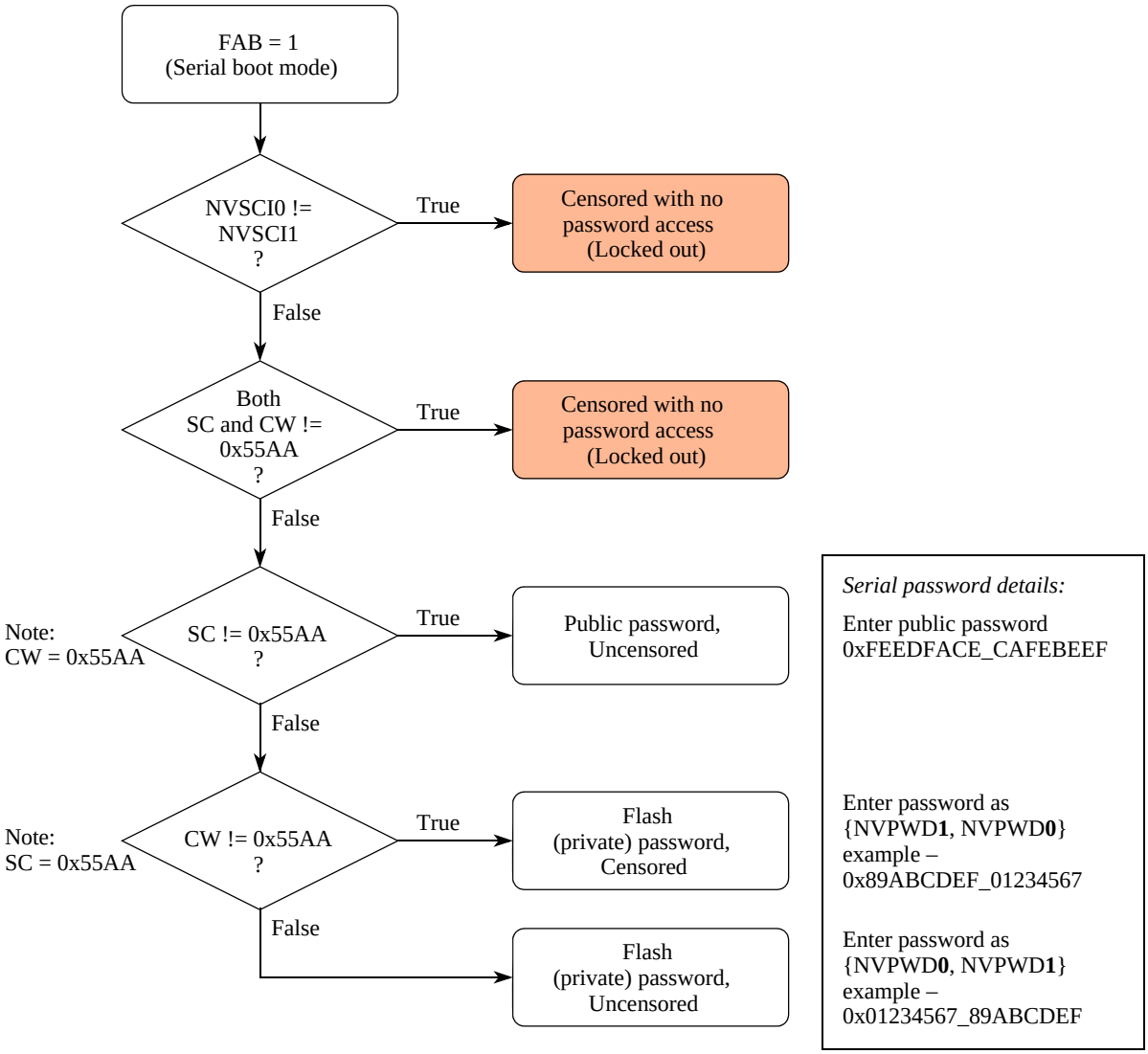


Figure 660. Censorship control in serial boot mode

35 Voltage Regulators and Power Supplies

35.1 Voltage regulator

The power blocks are used for providing 1.2 V digital supply to the internal logic of the device. The main/input supply is 3.3 V to 5.0 V ±10% and the digital/regulated output supply has a trim target voltage of 1.28 V. The voltage regulator used in SPC560P44Lx, SPC560P50Lx is the high power or main regulator (HPREG).

The internal voltage regulator requires an external ballast transistor and (external) capacitance (CREG) to be connected to the device in order to provide a stable low voltage digital supply to the device. Capacitances should be placed on the board as near as possible from the associated pins.The regulator has a digital domain called the High Power domain that has a low voltage detector for the 1.2 V output voltage. Additionally, there are two low voltage detectors for the main/input supply with different threshold, one at 3.3 V level and the other one at 5 V level.

35.1.1 High Power or Main Regulator (HPREG)

The HPREG converts the 3.3 V–5 V input supply to a 1.2 V digital supply. The nominal target output is 1.28 V. Due to all variations, the actual output will be in range of 1.08 V to 1.32 V in the full current load range (0–250 mA) after trimming.

The stabilization for HPREG is achieved using an external capacitance. The minimum recommended value is 3 × 10 µF with low ESR (refer to datasheet for details).

Note: In general an offset voltage must be avoided to pre-charge $V_{DD\_HV\_REG}$ through parasitic paths to allow a correct power up sequence.

The MCU supply must power on from GND to power supply with a voltage ramp for which minimum and maximum values are described in the data sheet (TV_{DD}).

35.1.2 Low Voltage Detectors (LVD) and Power On Reset (POR)

Five types of low voltage detectors are provided on the device:

- $V_{REGLVDMOK\_L}$ monitors the 3.3 V regulator supply
- $V_{FLLVDMOK\_L}$ monitors the 3.3 V flash supply
- $V_{IOLVDMOK\_L}$ monitors the 3.3 V I/O supply
- $V_{IOLVDM5OK\_L}$ monitors the 5 V I/O supply
- $V_{MLVDDOK\_L}$ monitors the 1.2 V digital logic supply

LVD_MAIN is the main voltage LVD with a threshold set around 2.7 V. LVD_MAIN5 is the main voltage LVD with a threshold set around 4 V. The LVD_MAIN and LVD_MAIN5 sense the 3.3 V–5 V power supply for CORE, shared with IO ring supply and indicate when the 3.3 V–5 V supply is stabilized. The threshold levels of LVD_MAIN5 are trimmable with the help of LVDM5[0:3] trim bits.

The LVD_MAIN and LVD_MAIN5 detectors sense the V_{DDIO} supply and provide $V_{IOLVDMOK\_H}$/$V_{IOLVDM5OK\_H}$ and $V_{IOLVDMOK\_L}$/$V_{IOLVDM5OK\_L}$ as active high signals at 3.3 V and 1.2 V supply levels, respectively.

Two more LVD_MAIN detectors are also used for sensing VDDREG and VDDFLASH.

An LVD_DIG in the regulator senses the HPREG output. It provides $V_{MLVDDOK\_H}$ and V_{MLVDOK_L} as active high signals.

The reference voltage used for all LVDs is trimmed for LVD_DIG using the bits LP[4:7]. Therefore, during the pre-trimming period, LVD_DIG exhibits higher thresholds, whereas post trimming, the thresholds come in the desired range. The trimming bits are provided by SSCM device option bits, which are updated during the reset phase (RGM reset phase 2) only. Power-down pins are provided for LVDs. When LVDs are powered down, their outputs are pulled high. LVDs are not controllable by the user. The only option is the possibility to mask LVD_MAIN5 using the mask bit (5V_LVD_MASK) in the VREG_CTL register.

POR is required to initialize the device during supply rise. POR works only on the rising edge of main supply. To ensure its functioning during the following rising edge of the supply, it is reset by the output of the LVD_MAIN block when main supply reaches below the lower voltage threshold of the LVD_MAIN.

POR is asserted on power-up when V_{DD} supply is above V_{PORUP} minimum (refer to datasheet for details). It is released only after V_{DD} supply is above V_{PORH} (refer to datasheet for details). V_{DD} above V_{PORH} ensures power management module including internal LVDs modules are fully functional.

35.1.3 VREG digital interface

The voltage regulator digital interface provides the temporization delay at initial power-up and at exit from low-power modes. A signal, indicating that Low Power domain is powered, is used at power-up to release reset to temporization counter. On completion of the delay counter, a end-of-count signal is released, it is gated with an other signal indicating main domain voltage fine in order to release the VREGOK signal. This is used by RGM to release the reset to the device.

The VREG digital interface also holds control register to mask 5 V LVD status coming from the voltage regulator at the power-up.

35.1.4 Registers Description

For more information about these registers, see *Chapter 7: Power Control Unit (PCU)*.

35.1.4.1 Voltage Regulator Control Register (VREG_CTL)

Address: Base + 0x0080 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_MASK
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Figure 661. Voltage Regulator Control register (VREG_CTL)

Table 587. VREG_CTL field descriptions

Field	Description
5V_LVD_MASK	Mask bit for 5 V LVD from regulator This is a read/write bit and must be unmasked by writing a 1 by software to generate LVD functional reset request to RGM for 5V trip. 0 5 V LVD not masked 1 5 V LVD masked

35.1.4.2 Voltage Regulator Status register (VREG_STATUS)

Address: Base + 0x0084 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	5V_LVD_STATUS
W																
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

Figure 662. Voltage Regulator Status register (VREG_STATUS)

Table 588. VREG_STATUS field descriptions

Field	Description
5V_LVD_STATUS	Status bit for 5 V LVD from regulator 0 5 V LVD not OK 1 5 V LVD OK

35.2 Power supply strategy

The SPC560P44Lx, SPC560P50Lx provides three dedicated supply domains at the package level:

- HV—High voltage external power supply for I/Os, voltage regulator module, and most analog modules

  This must be provided externally through V_{DD_HV}/V_{SS_HV} power pins. Voltage values should be aligned with V_{DD}/V_{SS}. Refer to the device datasheet for details.
- ADC—High voltage external power supply for ADC module

  This must be provided externally through $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$ power pins. Voltage values should be aligned with $V_{DD\_HV\_ADx}/V_{SS\_HV\_ADx}$. Refer to the device datasheet for details.
- LV—Low voltage internal power supply for core, PLL, and flash digital logic

  This is provided to the core, PLL and flash. Five V_{DD_LV}/V_{SS_LV} pins pairs are provided to connect the low voltage power supply. Refer to the device datasheet for details.

The three dedicated supply domains are further divided within the package in order to reduce EMI and noise as much as possible:

- HV_REG—High voltage regulator supply
- HV_IO*n*—High voltage PAD supply
- HV_FL—High voltage flash supply
- HV_OSC[i]—High voltage external oscillator and regulator supply
- HV_AD*n*—High voltage supply and reference for ADC module. Supplies are further star routed to reduce impact of ADC resistive reference on ADC capacitive reference accuracy.
- LV_COR*n*—Low voltage supply for the core. It is also used to provide supply for PLL and Flsah memory through double bonding.

i. Regulator ground is separated from oscillator ground and shorted to the LV ground through star routing.

36 IEEE 1149.1 Test Access Port Controller (JTAGC)

36.1 Introduction

The JTAG port of the device consists of three inputs and one output. These pins include test data input (TDI), test mode select (TMS), test clock input (TCK) and test data output (TDO). TDI, TMS, TCK and TDO are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

36.2 Block diagram

Figure 663 is a block diagram of the JTAG Controller (JTAGC).

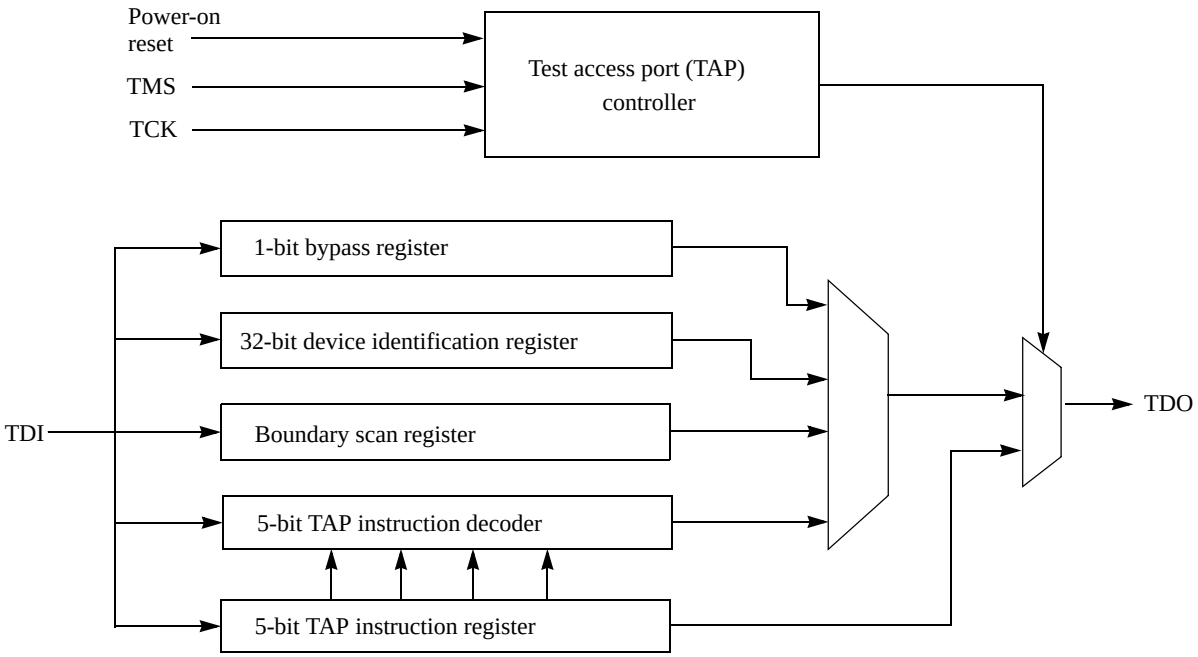


Figure 663. JTAG controller block diagram

36.3 Overview

The JTAGC provides the means to test device functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

36.4 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface
- Four pins (TDI, TMS, TCK, and TDO)—see *Section 36.6: External signal description*.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions.
- Test data registers: a bypass register, a boundary scan register, and a device identification register.
- A TAP controller state machine that controls the operation of the data registers, instruction register, and associated circuitry.

36.5 Modes of operation

The JTAGC uses a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

36.5.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

36.5.2 IEEE 1149.1-2001 defined test modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in *Section 36.8.4: JTAGC instructions*.

36.5.2.1 Bypass mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register provides a minimum-length serial path to shift data between TDI and TDO.

36.5.2.2 TAP sharing mode

There are four selectable auxiliary TAP controllers that share the TAP with the JTAGC. Selectable TAP controllers include the Nexus port controller (NPC), e200 OnCE, and eDMA Nexus. The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are ACCESS_AUX_TAP_ONCE and ACCESS_AUX_TAP_NPC. Instruction opcodes for each instruction are shown in *Table 591*.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers refer to *Chapter 37: Nexus Development Interface (NDI)*.

36.6 External signal description

The JTAGC consists of four signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in *Table 589*.

Table 589. JTAG signal properties(1)

Name	I/O	Function	Reset state	Pull(2)
TCK	I	Test clock	—	Down
TDI	I	Test data in	—	Up
TDO	O	Test data out	High Z(3)	Down(3)
TMS	I	Test mode select	—	Up

1. Test clock frequency must always be less than one fourth of system clock frequency.
2. The pull is not implemented in this module. Pullup/down devices are implemented in the pads.
3. TDO output buffer enable is negated when JTAGC is not in the Shift-IR or Shift-DR states. A weak pulldown can be implemented on TDO.

36.7 Memory map and registers description

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

36.7.1 Instruction register

The JTAGC uses a 5-bit instruction register as shown in *Figure 664*. The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

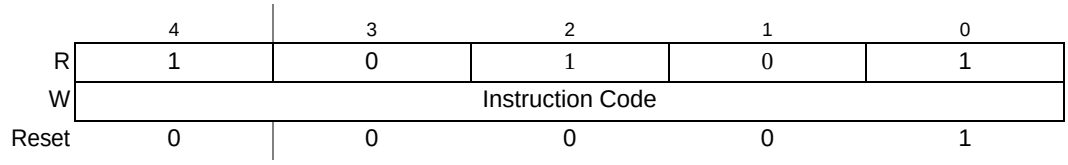


Figure 664. 5-bit Instruction register

36.7.2 Bypass register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

36.7.3 Device identification register

The device identification register, shown in *Figure 665*, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

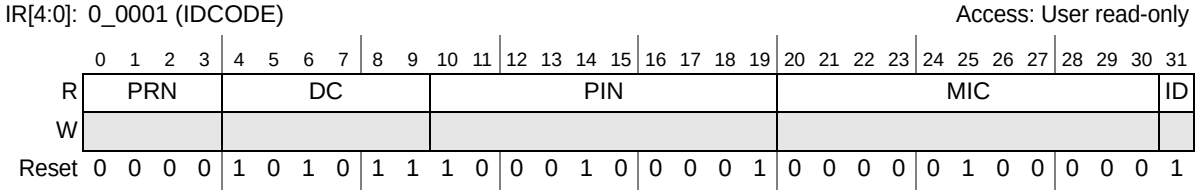


Figure 665. Device identification register

Table 590. Device identification register field descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. For the SPC560P44Lx, SPC560P50Lx this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC560P44Lx, SPC560P50Lx, this value is 0x221.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for STMicroelectronics, 0x20.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

36.7.4 Boundary scan register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in *Section 36.8.5: Boundary scan*. The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file.

36.8 Functional description

36.8.1 JTAGC reset configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

36.8.2 IEEE 1149.1-2001 (JTAG) Test Access Port (TAP)

The JTAGC uses the IEEE 1149.1-2001 Test Access Port (TAP) for accessing registers. This port can be shared with other TAP controllers on the MCU. For more detail on TAP sharing via JTAGC instructions refer to *Section 36.8.4.2: ACCESS_AUX_TAP_x instructions*.

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in *Figure 666*. This applies for the instruction register, test data registers, and the bypass register.

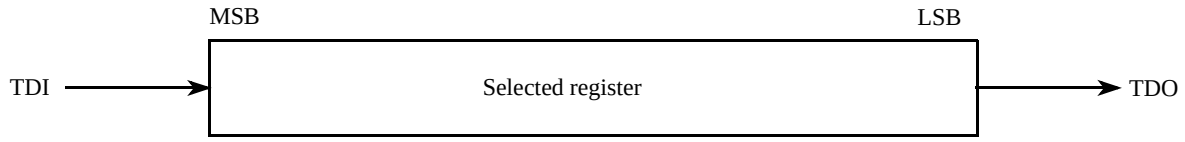
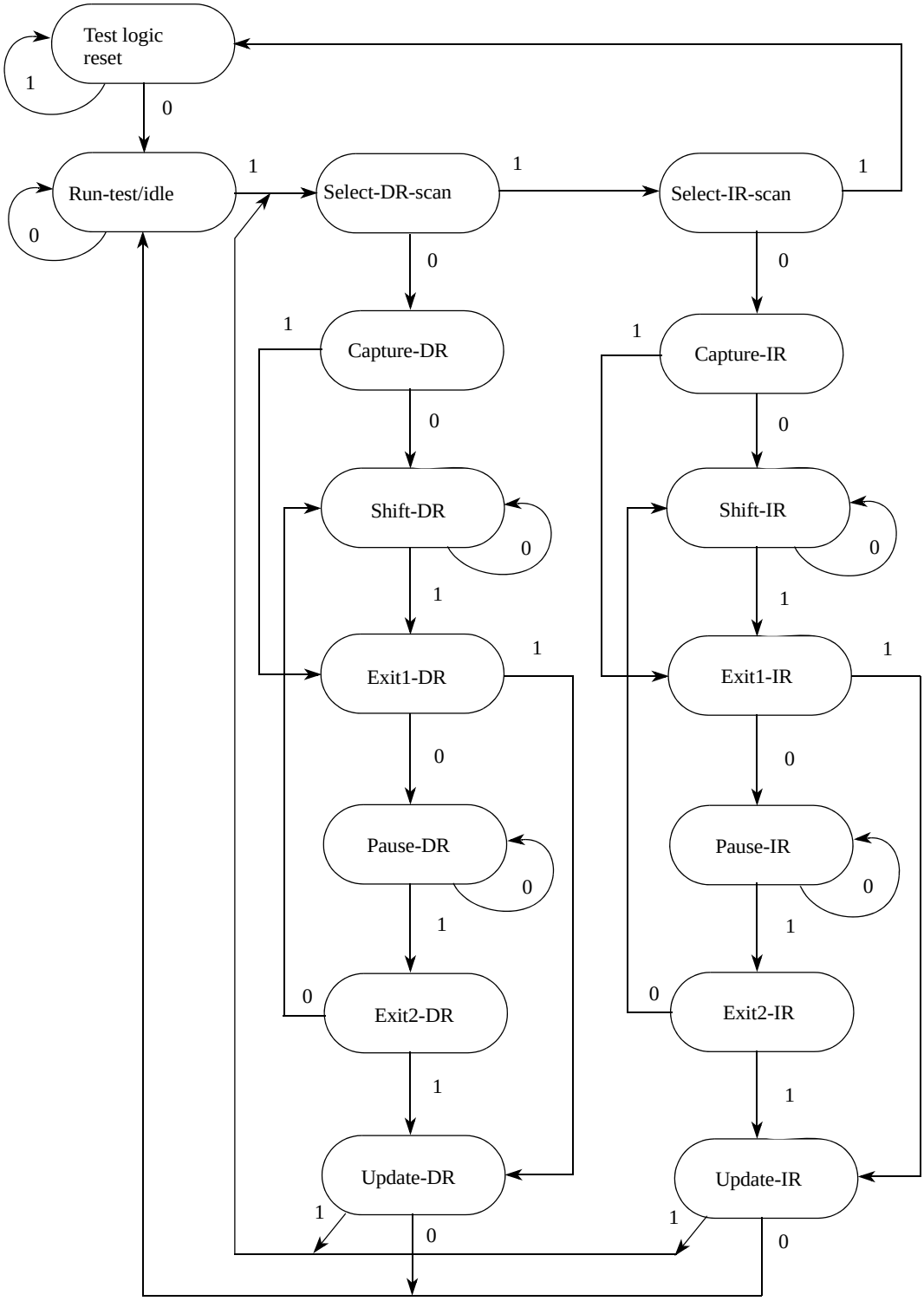


Figure 666. Shifting data through a register

36.8.3 TAP controller state machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. *Figure 667* shows the machine's states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As *Figure 667* shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 667. IEEE 1149.1-2001 TAP controller finite state machine

36.8.3.1 Selecting an IEEE 1149.1-2001 register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path reads or writes the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

36.8.4 JTAGC instructions

This section gives an overview of each instruction. Refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in *Table 591*.

Table 591. JTAG instructions

Instruction	Code[4:0]	Instruction summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
HIGHZ	01001	Selects bypass register while tristating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Grants the Nexus port controller (NPC) ownership of the TAP
ACCESS_AUX_TAP_ONCE	10001	Grants the Nexus e200z0 core interface ownership of the TAP
BYPASS	11111	Selects bypass register for data operations
Factory Debug Reserved(1)	00101 00110 01010	Intended for factory debug only
Reserved	All Other Codes	Decoded to select bypass register

1. Intended for factory debug, and not customer use.

36.8.4.1 BYPASS instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from

other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

36.8.4.2 ACCESS_AUX_TAP_*x* instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

36.8.4.3 CLAMP instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

36.8.4.4 EXTEST — external test instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

36.8.4.5 HIGHZ instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

36.8.4.6 IDCODE instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

36.8.4.7 SAMPLE instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the

update-DR state. Both the data capture and the shift operation are transparent to system operation.

36.8.4.8 SAMPLE/PRELOAD instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

36.8.5 Boundary scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

36.9 e200z0 OnCE controller

The e200z0 core OnCE controller supports a complete set of Nexus 1 debug. A complete discussion of the e200z0 OnCE debug features is available in the core reference manual.

36.9.1 e200z0 OnCE controller block diagram

Figure 668 is a block diagram of the e200z0 OnCE block.

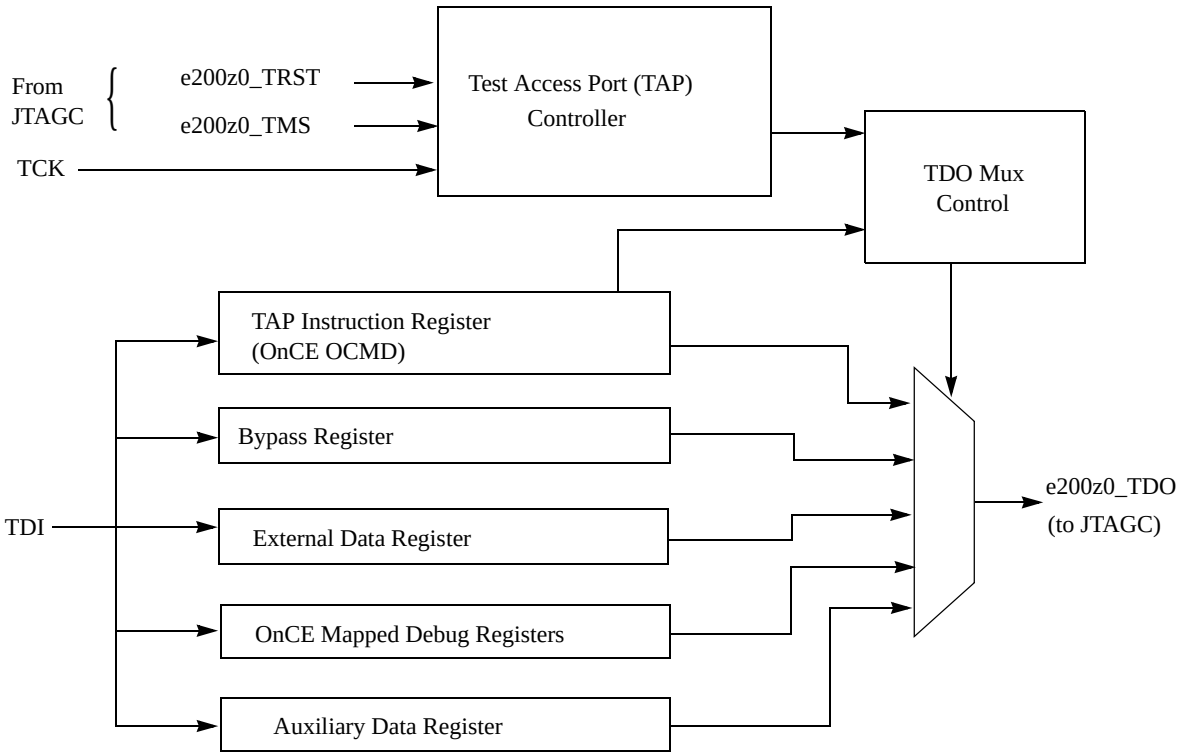


Figure 668. e200z0 OnCE block diagram

36.9.2 e200z0 OnCE controller functional description

The functional description for the e200z0 OnCE controller is the same as for the JTAGC, with the differences described as follows.

36.9.2.1 Enabling the TAP controller

To access the e200z0 OnCE controller, the proper JTAGC instruction needs to be loaded in the JTAGC instruction register, as discussed in *Section 36.5.2.2: TAP sharing mode*. The e200z0 OnCE TAP controller may either be accessed independently or chained with the e200z1 OnCE TAP controller, such that the TDO output of the e200z1 TAP controller is fed into the TDI input of the e200z0 TAP controller. The chained configuration allows commands to be loaded into both core's OnCE registers in one shift operation, so that both cores can be sent a GO command at the same time for example.

36.9.3 e200z0 OnCE controller registers description

Most e200z0 OnCE debug registers are fully documented in the core reference manual.

36.9.3.1 OnCE Command register (OCMD)

The OnCE command register (OCMD) is a 10-bit shift register that receives its serial data from the TDI pin and serves as the instruction register (IR). It holds the 10-bit commands to be used as input for the e200z0 OnCE Decoder. The OCMD is shown in *Figure 669*. The OCMD is updated when the TAP controller enters the update-IR state. It contains fields for

controlling access to a resource, as well as controlling single-step operation and exit from OnCE mode.

Although the OCMD is updated during the update-IR TAP controller state, the corresponding resource is accessed in the DR scan sequence of the TAP controller, and as such, the update-DR state must be transitioned through in order for an access to occur. In addition, the update-DR state must also be transitioned through in order for the single-step and/or exit functionality to be performed, even though the command appears to have no data resource requirement associated with it.

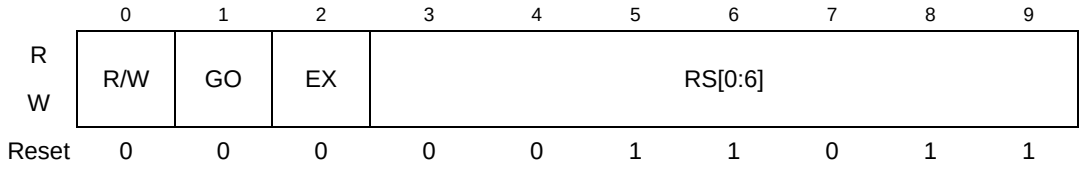


Figure 669. OnCE Command register (OCMD)

Table 592. e200z0 OnCE register addressing

RS[0:6]	Register selected
000 0000 000 0001	Reserved
000 0010	JTAG ID (read-only)
000 0011 – 000 1111	Reserved
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011 – 001 1111	Reserved
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)
010 0111	Data Value Compare 2 (DVC2)
010 1000 – 010 1111	Reserved
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug Control Register 1 (DBCR1)
011 0011	Debug Control Register 2 (DBCR2)
011 0100 – 101 1111	Reserved (do not access)

Table 592. e200z0 OnCE register addressing(Continued)

RS[0:6]	Register selected
110 1111	Shared Nexus Control Register (SNC) (only available on the e200z0 core)
111 0000 – 111 1001	General Purpose Register Selects [0:9]
111 1010 – 111 1011	Reserved
111 1100	Reserved
111 1101	LSRL Select (factory test use only)
111 1110	Enable_OnCE
111 1111	Bypass

36.10 Initialization/Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Place the JTAGC in reset through TAP controller state machine transitions controlled by TMS
2. Load the appropriate instruction for the test or action to be performed.

37 Nexus Development Interface (NDI)

37.1 Introduction

The Nexus Development Interface (NDI) block provides real-time development support capabilities for the SPC560P44Lx, SPC560P50Lx MCU in compliance with the IEEE-ISTO 5001-2003 standard. This development support is supplied for MCUs without requiring external address and data pins for internal visibility.

The NDI block is an integration of several individual Nexus blocks that are selected to provide the development support interface for SPC560P44Lx, SPC560P50Lx.

The NDI block interfaces to the e200z0, and internal buses to provide development support as per the IEEE-ISTO 5001-2003 standard. The development support provided includes program trace, watchpoint messaging, ownership trace, watchpoint triggering, processor overrun control, run-time access to the MCU's internal memory map, and access to the e200z0 internal registers during halt, via the JTAG port.

37.2 Block diagram

Figure 670 shows a functional block diagram of the NDI.

A simplified block diagram of the NDI illustrates the functionality and interdependence of major blocks (see *Figure 671*) and how the individual Nexus blocks are combined to form the NDI.

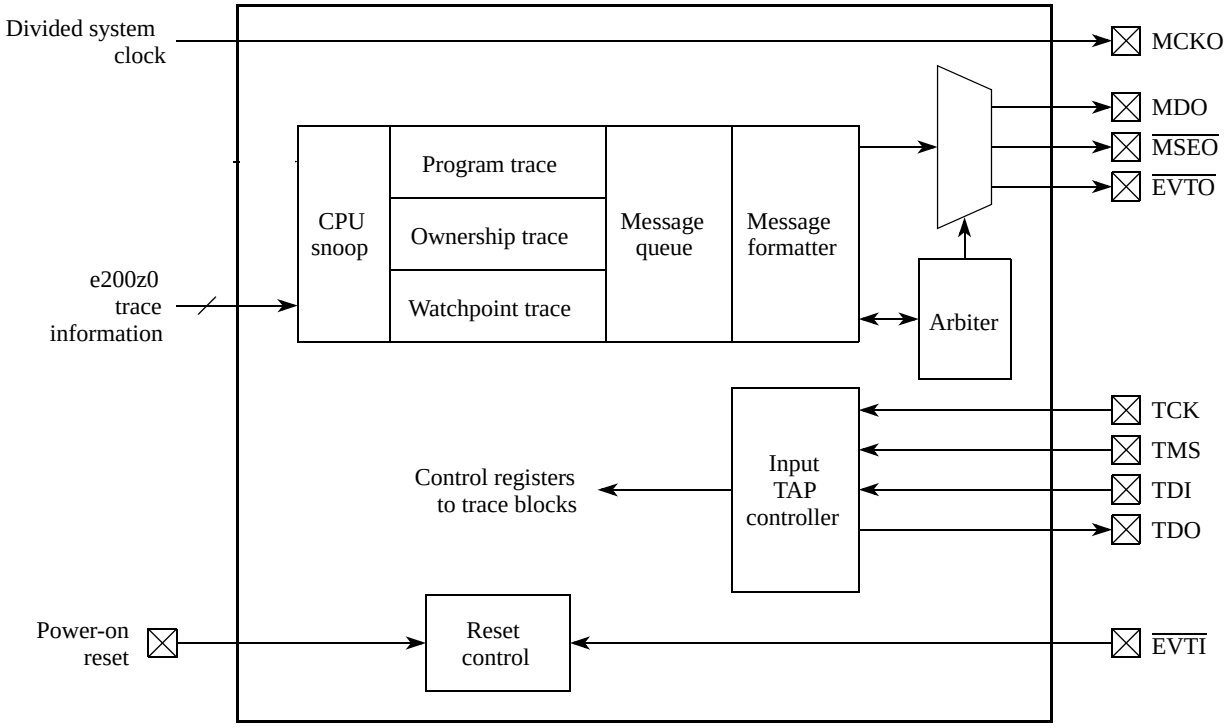


Figure 670. NDI functional block diagram

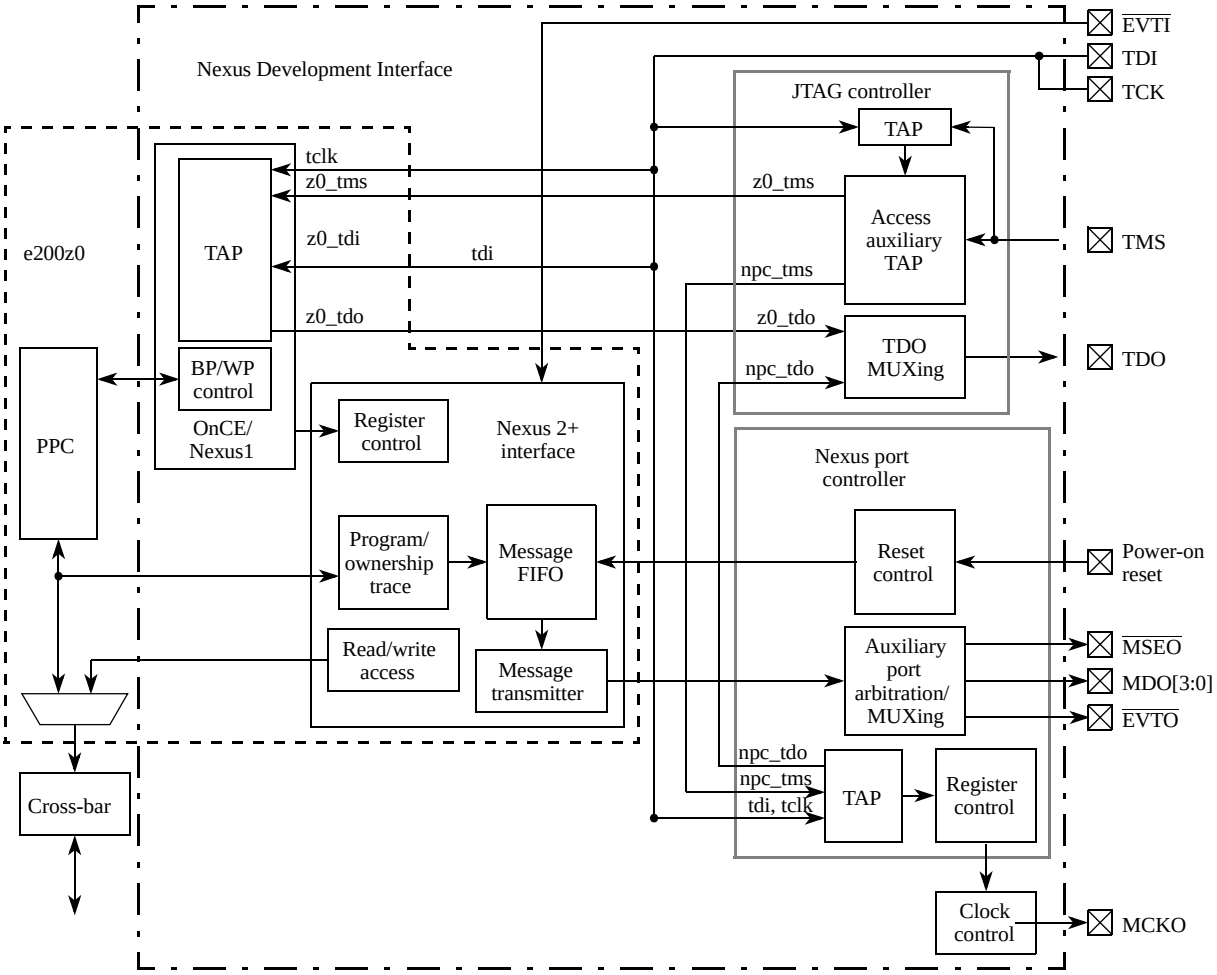


Figure 671. NDI implementation block diagram

37.3 Features

The NDI module of the SPC560P44Lx, SPC560P50Lx is compliant with Class 2 of the IEEE-ISTO 5001-2003 standard, with additional Class 3 and Class 4 features available. The following features are implemented:

- Program trace via branch trace messaging (BTM)—Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code may be traced.
- Ownership trace via ownership trace messaging (OTM)—OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated.

An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.

- Watchpoint messaging via the auxiliary pins
- Watchpoint trigger enable of program trace messaging
- Auxiliary interface for higher data input/output
  - 4 message data out (MDO) pins (2 MDO pins in reduced-port mode or 4 MDO pins in full-port mode)
  - 2 message start/end out pins ($\overline{\text{MSEO}}$[0:1])
  - 1 watchpoint event pin ($\overline{\text{EVTO}}$)
  - 1 event-in pin ($\overline{\text{EVTI}}$)
  - 1 message clock out pin (MCKO)
  - 4-pin JTAG port (TDI, TDO, TMS, and TCK)
- Registers for program trace, ownership trace, and watchpoint trigger
- All features controllable and configurable via the JTAG port
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This allows for enhanced download/upload capabilities.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- Support for internal censorship mode to prevent external access to flash memory contents when censorship is enabled

Note: If the e200z0 core has executed a wait instruction, then the Nexus 2+ controller clocks are gated off. While the core is in this state, it is not possible to perform Nexus read/write operations.

37.4 Modes of operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal or through state machine transitions controlled by TMS. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block.

The NPC transitions out of the reset state immediately following negation of power-on reset.

37.4.1 Nexus reset

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to tristate the output pins or use them for another function.

37.4.2 Full-Port mode

In full-port mode, all available MDO pins transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Four MDO pins are available in full-port mode.

37.4.2.1 Reduced-port mode

In reduced-port mode, a subset of the available MDO pins transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. Two MDO pins are available in reduced-port mode. Unused MDO pins can be used as GPIO.

37.4.2.2 Disabled-port mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages can not be used. The primary features available are Class 1 features and read/write access.

37.4.2.3 Censored mode

The NDI supports internal flash censorship mode by preventing the transmission of trace messages and Nexus access to memory-mapped resources when censorship is enabled.

37.4.2.4 Stop mode

Stop mode logic is implemented in the Nexus port controller (NPC). When a request is made to enter stop mode, the NDI block completes monitoring of any pending bus transaction, transmits all messages already queued, and acknowledges the stop request. After the acknowledgment, the system clock input are shut off by the clock driver on the device. While the clocks are shut off, the development tool cannot access NDI registers via the JTAG port.

37.5 External signal description

All signals are shared by the individual blocks that make up the NDI block. The Nexus port controller (NPC) block controls the signal sharing.

Refer to *Chapter 3: Signal Description* for detailed signal descriptions. Note that the NDI signals in *Table 593* are not present in the 100-pin package.

37.5.1 Nexus signal reset states

Table 593. NDI signal reset state

Name	Function	Nexus reset state	Pull
EVTI	Event-in pin	—	Up
EVTO	Event-out pin	0b1	—
MCKO	Message clock out pin	0b0	—

Table 593. NDI signal reset state

Name	Function	Nexus reset state	Pull
MDO[3:0] or MDO[1:0]	Message data out pins	0(1)	—
MSEO[1:0]	Message start/end out pins	0b11	—

1. MDO[0] reflects the state of the internal power on reset signal until $\overline{\text{RESET}}$ is deasserted.

37.6 Memory map and registers description

The NDI block contains no memory-mapped registers. Nexus registers are accessed by a development tool via the JTAG port using a client-select value and a register index. OnCE registers are accessed by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD) via the JTAG port.

37.6.1 Nexus debug interface registers

Table 594 shows the NDI registers by client select and index values. OnCE register addressing is documented in *Chapter 36: IEEE 1149.1 Test Access Port Controller (JTAGC)*.

Table 594. Nexus debug interface registers

Register type	Client select	Index	Register
Client-independent	0bxxxx	0	Nexus Device ID (DID) register(1)
	0bxxxx	127	Port Configuration Register (PCR)(1)
e200z0 Control/Status	0b0000	2	Development Control register 1 (DC1)
	0b0000	3	Development Control register 2 (DC2)
	0b0000	4	Development Status register (DS)
	0b0000	7	Read/Write Access Control/Status (RWCS)
	0b0000	9	Read/Write Access Address (RWA)
	0b0000	10	Read/Write Access Data register (RWD)
	0b0000	11	Watchpoint Trigger register (WT)

1. Implemented in NPC block. All other registers implemented in e200z0 Nexus 2+ block.

37.6.2 Registers description

This section lists the NDI registers and describes the registers and their bit fields.

37.6.2.1 Nexus Device ID (DID) register

The NPC device identification register allows the part revision number, design center, part identification number, and manufacturer identity code of the device to be determined through the auxiliary output port, and serially through TDO.

Reg Index: 0 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRN				DC				PIN[0:5]							
W																
Reset	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PIN[6:9]				MIC											ID
W																
Reset	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1

Figure 672. Nexus Device ID (DID) register

Table 595. DID field descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. For the SPC560P44Lx, SPC560P50Lx this value is 0x2B.
10–19 PIN	Part identification number. Contains the part number of the device. For the SPC560P44Lx, SPC560P50Lx, this value is 0x221.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for STMicroelectronics, 0x20.
31	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

37.6.2.2 Port Configuration Register (PCR)

The PCR selects the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register should be configured as soon as the NDI is enabled.

The PCR may be rewritten by the debug tool subsequent to the enabling of the NPC for low power debug support. In this case, the debug tool may set and clear the LP_DBG_EN, SLEEP_SYNC, and STOP_SYNC bits, but must preserve the original state of the remaining bits in the register.

Note: The mode or clock division must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Reg Index: 127 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			EVT_EN	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LP_DBG_EN	0	0	0	0	0	SLEEP_SYNC	STOP_SYNC	0	0	0	0	0	0	0	PSTAT_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 673. Port Configuration Register (PCR)

Table 596. PCR field descriptions

Field	Description
0 FPM	Full Port Mode The value of the FPM bit determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 A subset of MDO pins transmits messages. 1 All MDO pins transmit messages.
1 MCKO_GT	MCKO Clock Gating Control This bit enables or disables MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 MCKO gating is disabled. 1 MCKO gating is enabled.
2 MCKO_EN	MCKO Enable This bit enables the MCKO clock to run. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.

Table 596. PCR field descriptions(Continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>3–5
MCKO_DIV</td><td>MCKO Division Factor
The value of this signal determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. In this table, SYS_CLK represents the system clock frequency.

<table>
<tr><th>MCKO_DIV[2:0]</th><th>MCKO Frequency</th></tr>
<tr><td>0b000</td><td>SYSCLK÷1</td></tr>
<tr><td>0b001</td><td>SYSCLK÷2</td></tr>
<tr><td>0b010</td><td>Reserved</td></tr>
<tr><td>0b011</td><td>SYS_CLK÷4</td></tr>
<tr><td>0b100</td><td>Reserved</td></tr>
<tr><td>0b101</td><td>Reserved</td></tr>
<tr><td>0b110</td><td>Reserved</td></tr>
<tr><td>0b111</td><td>SYS_CLK÷8</td></tr>
</table>
</td></tr>
<tr><td>6
EVT_EN</td><td>EVTO/EVTI Enable
This bit enables the EVTO/EVTI port functions.
0 EVTO/EVTI port disabled
1 EVTO/EVTI port enabled</td></tr>
<tr><td>7–15</td><td>Reserved</td></tr>
<tr><td>16
LP_DBG_EN</td><td>Low Power Debug Enable
The LP_DBG_EN bit enables debug functionality to support entry and exit from low power sleep and stop modes.
0 Low power debug disabled
1 Low power debug enabled</td></tr>
<tr><td>17–21</td><td>Reserved</td></tr>
<tr><td>22
SLEEP_SYNC</td><td>Sleep Mode Synchronization
The SLEEP_SYNC bit synchronizes the entry into sleep mode between the device and debug tool. The device sets this bit before a pending entry into sleep mode. After reading SLEEP_SYNC as set, the debug tool then clears SLEEP_SYNC to acknowledge to the device that it may enter into sleep mode.
0 Sleep mode entry acknowledge
1 Sleep mode entry pending</td></tr>
<tr><td>23
STOP_SYNC</td><td>Stop Mode Synchronization
The STOP_SYNC bit synchronizes the entry into stop mode between the device and debug tool. The device sets this bit before a pending entry into stop mode. After reading STOP_SYNC as set, the debug tool then clears STOP_SYNC to acknowledge to the device that it may enter into stop mode.
0 Stop mode entry acknowledge
1 Stop mode entry pending</td></tr>
<tr><td>24–30</td><td>Reserved</td></tr>
<tr><td>31
PSTAT_EN</td><td>Processor Status Mode Enable
Note: SPC560P44Lx, SPC560P50Lx does not support the PSTAT mode. Setting PSTAT_EN will drive 0s to the MDO and MSEO pins.</td></tr>
</table>

37.6.2.3 Development Control register 1, 2 (DC1, DC2)

The development control registers DC1 and DC2 control the basic development features of the Nexus module. *Figure 674* shows DC1 and *Table 597* describes the register's fields.

Nexus Reg: 0x0002 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OPC	MCK_DIV [1:0]		EOC[1:0]		0	PTM	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	OVC[2:0]			EIC[1:0]		TM[2:0]		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 674. Development Control register 1 (DC1)

Table 597. DC1 field descriptions

Field	Description
0 OPC	Output Port Mode Control 0 Reduced-port mode configuration (2 MDO pins) 1 Full-port mode configuration (4 MDO pins) **Note:** The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.
1–2 MCK_DIV[1:0]	MCKO Clock Divide Ratio 00 MCKO is 1 × processor clock freq. 01 MCKO is 1/2 × processor clock freq. 10 MCKO is 1/4 × processor clock freq. 11 MCKO is 1/8 × processor clock freq. **Note:** The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR). These bits are writable in the PCR but have no effect.
3–4 EOC[1:0]	$\overline{\text{EVTO}}$ Control 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoints (configured in DC2) 01 $\overline{\text{EVTO}}$ upon entry into debug mode 10 $\overline{\text{EVTO}}$ upon timestamping event 11 Reserved
5	Reserved
6 PTM	Program Trace Method 0 Program trace uses traditional branch messages. 1 Program trace uses branch history messages.

Table 597. DC1 field descriptions(Continued)

Field	Description
7 WEN	Watchpoint Trace Enable 0 Watchpoint messaging disabled 1 Watchpoint messaging enabled
8–23	Reserved
24–26 OVC[2:0]	Overrun Control 000 Generate overrun messages 001–010 Reserved 011 Delay processor for BTM / DTM / OTM overruns 1xx Reserved
27–28 EIC[1:0]	$\overline{\text{EVTI}}$ Control 00 $\overline{\text{EVTI}}$ is used for synchronization (program trace/ data trace). 01 $\overline{\text{EVTI}}$ is used for debug request. 1x Reserved
29–31 TM[2:0]	Trace Mode Any or all of the TM bits may set, enabling one or more traces. 000 No trace 1xx Program trace enabled x1x Data trace enabled (not supported mode) xx1 Ownership trace enabled

DC2 is shown in *Figure 675* and its fields are described in *Table 598*.

Nexus Reg: 0x0003 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EWC[7:0]								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 675. Development Control register 2 (DC2)

Table 598. DC2 field descriptions

Field	Description
0–7 EWC[7:0]	$\overline{\text{EVTO}}$ Watchpoint Configuration Any or all of the bits in EWC may be set to configure the $\overline{\text{EVTO}}$ watchpoint. 00000000 No Watchpoints trigger $\overline{\text{EVTO}}$ 1xxxxxxx Watchpoint #0 (IAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ x1xxxxxx Watchpoint #1 (IAC2 from Nexus1) triggers $\overline{\text{EVTO}}$ xx1xxxxx Watchpoint #2 (IAC3 from Nexus1) triggers $\overline{\text{EVTO}}$ xxx1xxxx Watchpoint #3 (IAC4 from Nexus1) triggers $\overline{\text{EVTO}}$ xxxx1xxx Watchpoint #4 (DAC1 from Nexus1) triggers $\overline{\text{EVTO}}$ xxxxx1xx Watchpoint #5 (DAC2 from Nexus1) triggers $\overline{\text{EVTO}}$ xxxxxx1x Watchpoint #6 (DCNT1 from Nexus1) triggers $\overline{\text{EVTO}}$ xxxxxxx1 Watchpoint #7 (DCNT2 from Nexus1) triggers $\overline{\text{EVTO}}$
8–31	Reserved

Note: The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

37.6.2.4 Development Status register (DS)

The development status register reports system debug status. When debug mode is entered or exited, or a core-defined low-power mode is entered, a debug status message is transmitted with DS[31:24]. The external tool can read this register at any time.

Nexus Reg: 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DBG	0	0	0	LPC[1:0]		CHK	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 676. Development Status register (DS)

Table 599. DS field descriptions

Field	Description
0 DBG	CPU Debug Mode Status 0 CPU not in debug mode 1 CPU in debug mode
1–3	Reserved

Table 599. DS field descriptions(Continued)

Field	Description
4–5 LPC[1:0]	CPU Low-Power Mode Status 00 Normal (run) mode 01 CPU in halted state 10 CPU in stopped state 11 Reserved
6 CHK	CPU Checkstop Status 0 CPU not in checkstop state 1 CPU in checkstop state
7–31	Reserved

37.6.2.5 Read/Write Access Control/Status (RWCS)

The read write access control/status register provides control for read/write access. Read/write access provides DMA-like access to memory-mapped resources on the system bus while the processor is halted or during runtime. The RWCS register also provides read/write access status information as shown in *Figure 677*.

Nexus Reg: 0x0007 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	AC	RW	SZ[2:0]			MAP[2:0]			PR[1:0]		BST	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT[13:0]														ERR	DV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 677. Read/Write Access Control/Status register (RWCS)

Table 600. RWCS field description

Field	Description
0 AC	Access Control 0 End access 1 Start access
1 RW	Read/Write Select 0 Read access 1 Write access

Table 600. RWCS field description(Continued)

Field	Description
2–4 SZ[2:0]	Word Size 000 8-bit (byte) 001 16-bit (halfword) 010 32-bit (word) 011 64-bit (doubleword—only in burst mode) 100–111 Reserved (default to word)
5–7 MAP[2:0]	MAP Select 000 Primary memory map 001-111 Reserved
8–9 PR[1:0]	Read/Write Access Priority 00 Lowest access priority 01 Reserved (default to lowest priority) 10 Reserved (default to lowest priority) 11 Highest access priority
10 BST	Burst Control 0 Module accesses are single bus cycle at a time 1 Module accesses are performed as burst operation
11–15	Reserved
16–31 CNT[13:0]	Access Control Count Number of accesses of word size SZ.
30 ERR	Read/Write Access Error See *Table 601*.
31 DV	Read/Write Access Data Valid See *Table 601*.

Table 601 details the status bit encodings.

Table 601. Read/write access status bit encoding

Read Action	Write Action	ERR	DV
Read access has not completed	Write access completed without error	0	0
Read access error has occurred	Write access error has occurred	1	0
Read access completed without error	Write access has not completed	0	1
Not allowed	Not allowed	1	1

37.6.2.6 Read/Write Access Address (RWA)

The read/write access address register provides the system bus address to be accessed when initiating a read or a write access.

Nexus Reg: 0x0009 Access: User read/write

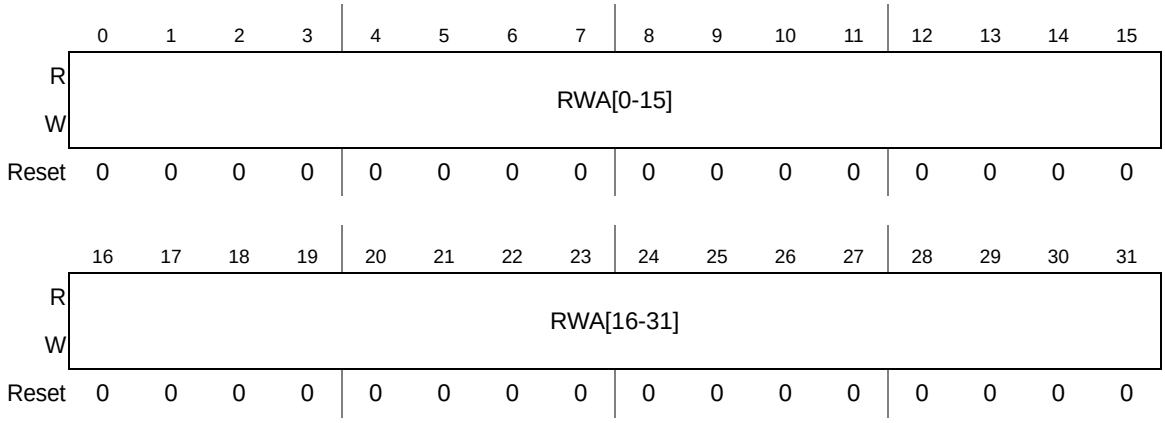


Figure 678. Read/Write Access Address register (RWA)

37.6.2.7 Read/Write Access Data register (RWD)

The read/write access data register provides the data to/from system bus memory-mapped locations when initiating a read or a write access.

Nexus Reg: 0x000A Access: User read/write

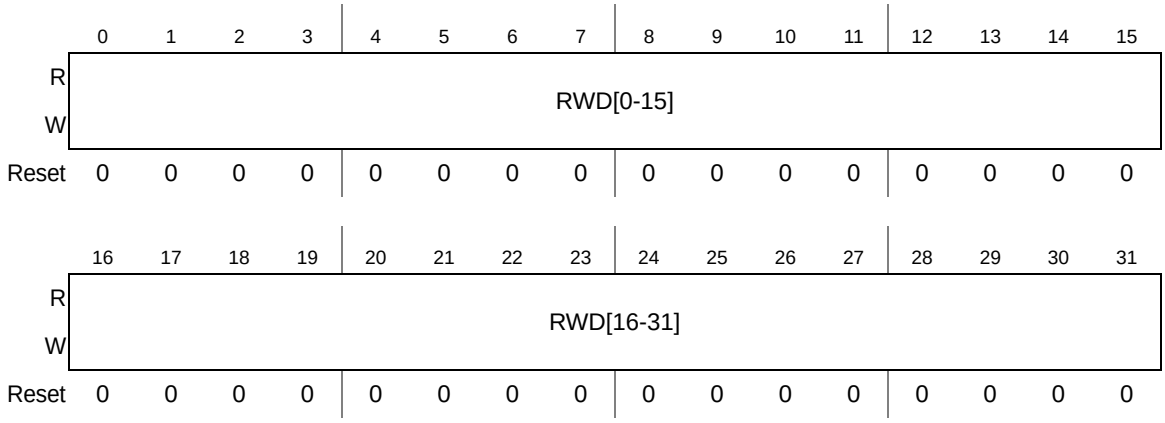


Figure 679. Read/Write Access Data register (RWD)

37.6.2.8 Watchpoint Trigger register (WT)

The watchpoint trigger register allows the watchpoints defined within the Nexus1 logic to trigger actions. These watchpoints can control program and/or data trace enable and disable. The WT bits can be used to produce an address-related window for triggering trace messages.

Nexus Reg: 0x000B Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PTS[2:0]			PTE[2:0]			0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 680. Watchpoint Trigger register (WT)

Table 602 details the watchpoint trigger register fields.

Table 602. WT field descriptions

Field	Description
0–2 PTS[2:0]	Program Trace Start Control 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
3–5 PTE[2:0]	Program Trace End Control 000 Trigger disabled 001 Use watchpoint #0 (IAC1 from Nexus1). 010 Use watchpoint #1 (IAC2 from Nexus1). 011 Use watchpoint #2 (IAC3 from Nexus1). 100 Use watchpoint #3 (IAC4 from Nexus1). 101 Use watchpoint #4 (DAC1 from Nexus1). 110 Use watchpoint #5 (DAC2 from Nexus1). 111 Use watchpoint #6 or #7 (DCNT1 or DCNT2 from Nexus1).
12–31	Reserved

37.7 Functional description

The NDI block is implemented by integrating the following blocks on the SPC560P44Lx, SPC560P50Lx:

- Nexus e200z0 development interface (OnCE and Nexus2p subblocks)
- Nexus port controller (NPC) Block

37.7.1 Enabling Nexus clients for TAP access

After the conditions have been met to bring the NDI out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in *Table 603*. After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in *Table 604*. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Table 603. JTAGC Instruction opcodes to enable Nexus clients

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_ONCE	10001	Enables access to the e200z0 TAP controller

Table 604. Nexus client JTAG instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus ENABLE instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z0 OnCE JTAG Instruction Opcodes(1)		
NEXUS2_ACCESS	Opcode for e200z0 OnCE Nexus ENABLE instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z0 OnCE BYPASS instruction (10-bits)	0x7F

1. Refer to the e200z0 reference manual for a complete list of available OnCE instructions.

37.7.2 Configuring the NDI for Nexus messaging

The NDI is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively. When writing to the PCR, the PCR LSB must be written to a logic zero. Setting the LSB of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 605 describes the NDI configuration options.

Table 605. NDI configuration options

MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
0	X	Disabled
1	1	Full-port mode
1	0	Reduced-port mode

37.7.3 Programmable MCKO frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include one-quarter and one-eighth system clock speed.

Table 606 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the system clock frequency. The default value selected if a reserved encoding is programmed is SYS_CLK÷2.

Note: On SPC560P44Lx, SPC560P50Lx, the pad type used for the Nexus 2+ signals will not support the default SYSCLK÷2 and SYSCLK÷1 setting, so the user must change the MCKO frequency to be not faster than SYSCLK÷4.

Table 606. MCKO_DIV values

MCKO_DIV[2:0]	MCKO frequency
0b000	SYSCLK
0b001	SYSCLK ÷ 2
0b010	Reserved
0b011	SYS_CLK ÷ 4
0b100	Reserved
0b101	Reserved
0b110	Reserved
0b111	SYS_CLK ÷ 8

37.7.4 Nexus messaging

Most of the messages transmitted by the NDI include a SRC field. This field identifies the source that generated the message. *Table 607* shows the values used for the SRC field by the different clients on the SPC560P44Lx, SPC560P50Lx. These values are specific to the SPC560P44Lx, SPC560P50Lx. The size of the SRC field in transmitted messages is 4 bits. This value is also specific to the SPC560P44Lx, SPC560P50Lx. The same values are used for the client select values written to the client select control register.

Table 607. SRC packet encodings

SRC[3:0]	SPC560P44Lx, SPC560P50Lx client
0b0000	e200z0
All other combinations	Reserved

37.7.5 $\overline{\text{EVTO}}$ sharing

The NPC block controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that generate an $\overline{\text{EVTO}}$ signal. The sharing mechanism is a logical AND of all incoming $\overline{\text{EVTO}}$ signals from Nexus blocks, thereby asserting $\overline{\text{EVTO}}$ whenever any block drives its $\overline{\text{EVTO}}$. When there is no active MCKO, such as in disabled mode, the NPC drives $\overline{\text{EVTO}}$ for two system clock periods. $\overline{\text{EVTO}}$ sharing is active as long as the NDI is not in reset.

37.7.6 Debug mode control

On SPC560P44Lx, SPC560P50Lx, program breaks can be requested either by using the $\overline{\text{EVTI}}$ pin as a break request, or when a Nexus event is triggered.

37.7.6.1 $\overline{\text{EVTI}}$ generated break request

To use the $\overline{\text{EVTI}}$ pin as a debug request, the EIC field in the e200z0 Nexus 2+ Development Control register 1 (DC1[4:3]) must be set to configure the $\overline{\text{EVTI}}$ input as a debug request.

Note: At least one TCK clock is necessary for the EVTI signal to be recognized by the MCU."

Appendix A Registers Under Protection

For SPC560P44Lx, SPC560P50Lx, the Register Protection module is operable on the registers listed in *Table 608*.

Table 608. Registers under protection

Module	Register	Register size (bits)	Register offset	Protected bitfields
Code Flash—Base address: 0xC3F8_8000 4 registers to protect				
Code Flash	MCR	32	0x0000	bits[0:31]
Code Flash	PFCR0	32	0x001C	bits[0:31]
Code Flash	PFCR1	32	0x0020	bits[0:31]
Code Flash	PFAPR	32	0x0024	bits[0:31]
Data Flash—Base address: 0xC3F8_C000 1 register to protect				
Data Flash	MCR	32	0x0000	bits[0:31]
SIU lite—Base address: 0xC3F9_0000 154 registers to protect				
SIUL	IRER	32	0x0018	bits[0:31]
SIUL	IREER	32	0x0028	bits[0:31]
SIUL	IFEER	32	0x002C	bits[0:31]
SIUL	IFER	32	0x0030	bits[0:31]
SIUL	PCR0	16	0x0040	bits[0:15]
SIUL	PCR1	16	0x0042	bits[0:15]
SIUL	PCR2	16	0x0044	bits[0:15]
SIUL	PCR3	16	0x0046	bits[0:15]
SIUL	PCR4	16	0x0048	bits[0:15]
SIUL	PCR5	16	0x004A	bits[0:15]
SIUL	PCR6	16	0x004C	bits[0:15]
SIUL	PCR7	16	0x004E	bits[0:15]
SIUL	PCR8	16	0x0050	bits[0:15]
SIUL	PCR9	16	0x0052	bits[0:15]
SIUL	PCR10	16	0x0054	bits[0:15]
SIUL	PCR11	16	0x0056	bits[0:15]
SIUL	PCR12	16	0x0058	bits[0:15]
SIUL	PCR13	16	0x005A	bits[0:15]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR14	16	0x005C	bits[0:15]
SIUL	PCR15	16	0x005E	bits[0:15]
SIUL	PCR16	16	0x0060	bits[0:15]
SIUL	PCR17	16	0x0062	bits[0:15]
SIUL	PCR18	16	0x0064	bits[0:15]
SIUL	PCR19	16	0x0066	bits[0:15]
SIUL	PCR20	16	0x0068	bits[0:15]
SIUL	PCR21	16	0x006A	bits[0:15]
SIUL	PCR22	16	0x006C	bits[0:15]
SIUL	PCR23	16	0x006E	bits[0:15]
SIUL	PCR24	16	0x0070	bits[0:15]
SIUL	PCR25	16	0x0072	bits[0:15]
SIUL	PCR26	16	0x0074	bits[0:15]
SIUL	PCR27	16	0x0076	bits[0:15]
SIUL	PCR28	16	0x0078	bits[0:15]
SIUL	PCR29	16	0x007A	bits[0:15]
SIUL	PCR30	16	0x007C	bits[0:15]
SIUL	PCR31	16	0x007E	bits[0:15]
SIUL	PCR32	16	0x0080	bits[0:15]
SIUL	PCR33	16	0x0082	bits[0:15]
SIUL	PCR48	16	0x00A0	bits[0:15]
SIUL	PCR49	16	0x00A2	bits[0:15]
SIUL	PCR50	16	0x00A4	bits[0:15]
SIUL	PCR51	16	0x00A6	bits[0:15]
SIUL	PCR52	16	0x00A8	bits[0:15]
SIUL	PCR53	16	0x00AA	bits[0:15]
SIUL	PCR54	16	0x00AC	bits[0:15]
SIUL	PCR55	16	0x00AE	bits[0:15]
SIUL	PCR56	16	0x00B0	bits[0:15]
SIUL	PCR57	16	0x00B2	bits[0:15]
SIUL	PCR58	16	0x00B4	bits[0:15]
SIUL	PCR59	16	0x00B6	bits[0:15]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR60	16	0x00B8	bits[0:15]
SIUL	PCR61	16	0x00BA	bits[0:15]
SIUL	PCR62	16	0x00BC	bits[0:15]
SIUL	PCR63	16	0x00BE	bits[0:15]
SIUL	PCR64	16	0x00C0	bits[0:15]
SIUL	PCR65	16	0x00C2	bits[0:15]
SIUL	PCR66	16	0x00C4	bits[0:15]
SIUL	PCR67	16	0x00C6	bits[0:15]
SIUL	PCR68	16	0x00C8	bits[0:15]
SIUL	PCR69	16	0x00CA	bits[0:15]
SIUL	PCR70	16	0x00CC	bits[0:15]
SIUL	PCR71	16	0x00CE	bits[0:15]
SIUL	PCR72	16	0x00D0	bits[0:15]
SIUL	PCR73	16	0x00D2	bits[0:15]
SIUL	PCR74	16	0x00D4	bits[0:15]
SIUL	PCR75	16	0x00D6	bits[0:15]
SIUL	PCR76	16	0x00D8	bits[0:15]
SIUL	PCR77	16	0x00DA	bits[0:15]
SIUL	PCR78	16	0x00DC	bits[0:15]
SIUL	PCR79	16	0x00DE	bits[0:15]
SIUL	PCR80	16	0x00E0	bits[0:15]
SIUL	PCR81	16	0x00E2	bits[0:15]
SIUL	PCR82	16	0x00E4	bits[0:15]
SIUL	PCR83	16	0x00E6	bits[0:15]
SIUL	PCR84	16	0x00E8	bits[0:15]
SIUL	PCR85	16	0x00EA	bits[0:15]
SIUL	PCR86	16	0x00EC	bits[0:15]
SIUL	PCR87	16	0x00EE	bits[0:15]
SIUL	PCR88	16	0x00F0	bits[0:15]
SIUL	PCR89	16	0x00F2	bits[0:15]
SIUL	PCR90	16	0x00F4	bits[0:15]
SIUL	PCR91	16	0x00F6	bits[0:15]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	PCR92	16	0x00F8	bits[0:15]
SIUL	PCR93	16	0x00FA	bits[0:15]
SIUL	PCR94	16	0x00FC	bits[0:15]
SIUL	PCR95	16	0x00FE	bits[0:15]
SIUL	PCR96	16	0x0100	bits[0:15]
SIUL	PCR97	16	0x0102	bits[0:15]
SIUL	PCR98	16	0x0104	bits[0:15]
SIUL	PCR99	16	0x0106	bits[0:15]
SIUL	PCR100	16	0x0108	bits[0:15]
SIUL	PCR101	16	0x010A	bits[0:15]
SIUL	PCR102	16	0x010C	bits[0:15]
SIUL	PCR103	16	0x010E	bits[0:15]
SIUL	PCR104	16	0x0110	bits[0:15]
SIUL	PCR105	16	0x0112	bits[0:15]
SIUL	PCR106	16	0x0114	bits[0:15]
SIUL	PCR107	16	0x0116	bits[0:15]
SIUL	PSMI0_3	32	0x0500	bits[0:31]
SIUL	PSMI4_7	32	0x0504	bits[0:31]
SIUL	PSMI8_11	32	0x0508	bits[0:31]
SIUL	PSMI12_15	32	0x050C	bits[0:31]
SIUL	PSMI16_19	32	0x0510	bits[0:31]
SIUL	PSMI20_23	32	0x0514	bits[0:31]
SIUL	PSMI24_27	32	0x0518	bits[0:31]
SIUL	PSMI28_31	32	0x051C	bits[0:31]
SIUL	PSMI32_35	32	0x0520	bits[0:31]
SIUL	IFMC0	32	0x1000	bits[0:31]
SIUL	IFMC1	32	0x01004	bits[0:31]
SIUL	IFMC2	32	0x1008	bits[0:31]
SIUL	IFMC3	32	0x100C	bits[0:31]
SIUL	IFMC4	32	0x1010	bits[0:31]
SIUL	IFMC5	32	0x1014	bits[0:31]
SIUL	IFMC6	32	0x1018	bits[0:31]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
SIUL	IFMC7	32	0x101C	bits[0:31]
SIUL	IFMC8	32	0x1020	bits[0:31]
SIUL	IFMC9	32	0x1024	bits[0:31]
SIUL	IFMC10	32	0x1028	bits[0:31]
SIUL	IFMC11	32	0x102C	bits[0:31]
SIUL	IFMC12	32	0x1030	bits[0:31]
SIUL	IFMC13	32	0x1034	bits[0:31]
SIUL	IFMC14	32	0x1038	bits[0:31]
SIUL	IFMC15	32	0x103C	bits[0:31]
SIUL	IFMC16	32	0x1040	bits[0:31]
SIUL	IFMC17	32	0x1044	bits[0:31]
SIUL	IFMC18	32	0x1048	bits[0:31]
SIUL	IFMC19	32	0x104C	bits[0:31]
SIUL	IFMC20	32	0x1050	bits[0:31]
SIUL	IFMC21	32	0x1054	bits[0:31]
SIUL	IFMC22	32	0x1058	bits[0:31]
SIUL	IFMC23	32	0x105C	bits[0:31]
SIUL	IFMC24	32	0x1060	bits[0:31]
SIUL	IFMC25	32	0x1064	bits[0:31]
SIUL	IFMC26	32	0x1068	bits[0:31]
SIUL	IFMC27	32	0x106C	bits[0:31]
SIUL	IFMC28	32	0x1070	bits[0:31]
SIUL	IFMC29	32	0x1074	bits[0:31]
SIUL	IFMC30	32	0x1078	bits[0:31]
SIUL	IFMC31	32	0x107C	bits[0:31]
SIUL	IFCPR	32	0x1080	bits[0:31]
Voltage Regulator—Base address: 0xC3FE_8080 1 register to protect				
VREG	VREG_CTL	32	0x0000	bits[0:31]
MC Mode Entry—Base address: 0xC3FD_C000 36 registers to protect				
MC ME	ME_ME	32	0x0008	bits[0:31]
MC ME	ME_IM	32	0x0010	bits[0:31]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC ME	ME_TEST_MC	32	0x0024	bits[0:31]
MC ME	ME_SAFE_MC	32	0x0028	bits[0:31]
MC ME	ME_DRUN_MC	32	0x002C	bits[0:31]
MC ME	ME_RUN0_MC	32	0x0030	bits[0:31]
MC ME	ME_RUN1_MC	32	0x0034	bits[0:31]
MC ME	ME_RUN2_MC	32	0x0038	bits[0:31]
MC ME	ME_RUN3_MC	32	0x003C	bits[0:31]
MC ME	ME_HALT0_MC	32	0x0040	bits[0:31]
MC ME	ME_STOP0_MC	32	0x0048	bits[0:31]
MC ME	ME_RUN_PC0	32	0x0080	bits[0:31]
MC ME	ME_RUN_PC1	32	0x0084	bits[0:31]
MC ME	ME_RUN_PC2	32	0x0088	bits[0:31]
MC ME	ME_RUN_PC3	32	0x008C	bits[0:31]
MC ME	ME_RUN_PC4	32	0x0090	bits[0:31]
MC ME	ME_RUN_PC5	32	0x0094	bits[0:31]
MC ME	ME_RUN_PC6	32	0x0098	bits[0:31]
MC ME	ME_RUN_PC7	32	0x009C	bits[0:31]
MC ME	ME_LP_PC0	32	0x00A0	bits[0:31]
MC ME	ME_LP_PC1	32	0x00A4	bits[0:31]
MC ME	ME_LP_PC2	32	0x00A8	bits[0:31]
MC ME	ME_LP_PC3	32	0x00AC	bits[0:31]
MC ME	ME_LP_PC4	32	0x00B0	bits[0:31]
MC ME	ME_LP_PC5	32	0x00B4	bits[0:31]
MC ME	ME_LP_PC6	32	0x00B8	bits[0:31]
MC ME	ME_LP_PC7	32	0x00BC	bits[0:31]
MC ME	ME_PCTL[4..7]	32	0x00C4	bits[0:31]
MC ME	ME_PCTL[16..19]	32	0x00D0	bits[0:31]
MC ME	ME_PCTL[24..27]	32	0x00D8	bits[0:31]
MC ME	ME_PCTL[32..35]	32	0x00E0	bits[0:31]
MC ME	ME_PCTL[36..39]	32	0x00E4	bits[0:31]
MC ME	ME_PCTL[40..43]	32	0x00E8	bits[0:31]
MC ME	ME_PCTL[48..51]	32	0x00F0	bits[0:31]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC ME	ME_PCTL[92..95]	32	0x011C	bits[0:31]
Note: Only the following PCTL register numbers are implemented: 4,5,6,7, 16, 24, 26, 32, 33, 35, 38, 39, 41, 48, 49, 92.				
MC Clock Generation Module—Base address: 0xC3FE_0000 8 registers to protect				
MC CGM	CGM_OC_EN	32	0x0370	bits[0:31]
MC CGM	CGM_OCDS_SC	32	0x0374	bits[0:31]
MC CGM	CGM_AC0_SC	32	0x0380	bits[0:31]
MC CGM	CGM_AC0_DC0	32	0x0384	bits[0:31]
MC CGM	CGM_AC1_SC	32	0x0388	bits[0:31]
MC CGM	CGM_AC1_DC0	32	0x038C	bits[0:31]
MC CGM	CGM_AC2_SC	32	0x0390	bits[0:31]
MC CGM	CGM_AC2_DC0	32	0x0394	bits[0:31]
XOSC—Base address: 0xC3FE_0000 1 register to protect				
XOSC	OSC_CTL	32	0x0000	bits[0:31]
IRC_OSC—Base address: 0xC3FE_0060 1 register to protect				
IRC_OSC	RC_CTL	32	0x0000	bits[0:31]
FM PLL 0—Base address: 0xC3FE_00A0 2 registers to protect				
FMPLL 0	CR	32	0x0000	bits[0:31]
FMPLL 0	MR	32	0x0004	bits[0:31]
FM PLL 1—Base address: 0xC3FE_00C0 2 registers to protect				
FMPLL 1	CR	32	0x0000	bits[0:31]
FMPLL 1	MR	32	0x0004	bits[0:31]
CMU 0—Base address: 0xC3FE_0100 1 register to protect				
CMU 0	CMU_CSR	32	0x0000	bits[0:31]
CMU 1—Base address: 0xC3FE_0120 1 register to protect				
CMU 1	CMU_CSR	32	0x0000	bits[0:31]

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
MC Reset Generation Module—Base address: 0xC3FE_4000 6 registers to protect				
MC RGM	RGM_FERD	16	0x0004	bits[0:15]
MC RGM	RGM_DERD	16	0x0006	bits[0:15]
MC RGM	RGM_FEAR	16	0x0010	bits[0:15]
MC RGM	RGM_FESS	16	0x0018	bits[0:15]
MC RGM	RGM_FBRE	16	0x001C	bits[0:15]
PIT—Base address: 0xC3FF_0000 9 registers to protect				
PIT	PITMCR	32	0x0000	32-bit
PIT	LDVAL0	32	0x0100	32-bit
PIT	TCTRL0	32	0x0108	32-bit
PIT	LDVAL1	32	0x0110	32-bit
PIT	TCTRL1	32	0x0118	32-bit
PIT	LDVAL2	32	0x0120	32-bit
PIT	TCTRL2	32	0x0128	32-bit
PIT	LDVAL3	32	0x0130	32-bit
PIT	TCTRL3	32	0x0138	32-bit
ADC 0—Base address: 0xFFE0_0000 10 registers to protect				
ADC 0	MCR	32	0x0000	32-bit
ADC 0	TRC0	32	0x0050	32-bit
ADC 0	TRC1	32	0x0054	32-bit
ADC 0	TRC2	32	0x0058	32-bit
ADC 0	TRC3	32	0x005C	32-bit
ADC 1—Base address: 0xFFE0_4000 10 registers to protect				
ADC 1	MCR	32	0x0000	32-bit
ADC 1	TRC0	32	0x0050	32-bit
ADC 1	TRC1	32	0x0054	32-bit
ADC 1	TRC2	32	0x0058	32-bit
ADC 1	TRC3	32	0x005C	32-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
eTimer 0—Base address: 0xFFE1_8000 24 registers to protect				
eTimer 0	CH0_CTRL1	16	0x000E	16-bit
eTimer 0	CH0_CTRL2	16	0x0010	16-bit
eTimer 0	CH0_CTRL3	16	0x0012	16-bit
eTimer 0	CH0_CCCTRL	16	0x001C	16-bit
eTimer 0	CH1_CTRL1	16	0x002E	16-bit
eTimer 0	CH1_CTRL2	16	0x0030	16-bit
eTimer 0	CH1_CTRL3	16	0x0032	16-bit
eTimer 0	CH1_CCCTRL	16	0x003C	16-bit
eTimer 0	CH2_CTRL1	16	0x004E	16-bit
eTimer 0	CH2_CTRL2	16	0x0050	16-bit
eTimer 0	CH2_CTRL3	16	0x0052	16-bit
eTimer 0	CH2_CCCTRL	16	0x005C	16-bit
eTimer 0	CH3_CTRL1	16	0x006E	16-bit
eTimer 0	CH3_CTRL2	16	0x0070	16-bit
eTimer 0	CH3_CTRL3	16	0x0072	16-bit
eTimer 0	CH3_CCCTRL	16	0x007C	16-bit
eTimer 0	CH4_CTRL1	16	0x008E	16-bit
eTimer 0	CH4_CTRL2	16	0x0090	16-bit
eTimer 0	CH4_CTRL3	16	0x0092	16-bit
eTimer 0	CH4_CCCTRL	16	0x009C	16-bit
eTimer 0	CH5_CTRL1	16	0x00AE	16-bit
eTimer 0	CH5_CTRL2	16	0x00B0	16-bit
eTimer 0	CH5_CTRL3	16	0x00B2	16-bit
eTimer 0	CH5_CCCTRL	16	0x00BC	16-bit
eTimer 1—Base address: 0xFFE1_C000 24 registers to protect				
eTimer 1	CH0_CTRL1	16	0x000E	16-bit
eTimer 1	CH0_CTRL2	16	0x0010	16-bit
eTimer 1	CH0_CTRL3	16	0x0012	16-bit
eTimer 1	CH0_CCCTRL	16	0x001C	16-bit
eTimer 1	CH1_CTRL1	16	0x002E	16-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
eTimer 1	CH1_CTRL2	16	0x0030	16-bit
eTimer 1	CH1_CTRL3	16	0x0032	16-bit
eTimer 1	CH1_CCCTRL	16	0x003C	16-bit
eTimer 1	CH2_CTRL1	16	0x004E	16-bit
eTimer 1	CH2_CTRL2	16	0x0050	16-bit
eTimer 1	CH2_CTRL3	16	0x0052	16-bit
eTimer 1	CH2_CCCTRL	16	0x005C	16-bit
eTimer 1	CH3_CTRL1	16	0x00x006E	16-bit
eTimer 1	CH3_CTRL2	16	0x0070	16-bit
eTimer 1	CH3_CTRL3	16	0x0072	16-bit
eTimer 1	CH3_CCCTRL	16	0x007C	16-bit
eTimer 1	CH4_CTRL1	16	0x008E	16-bit
eTimer 1	CH4_CTRL2	16	0x0090	16-bit
eTimer 1	CH4_CTRL3	16	0x0092	16-bit
eTimer 1	CH4_CCCTRL	16	0x009C	16-bit
eTimer 1	CH5_CTRL1	16	0x00AE	16-bit
eTimer 1	CH5_CTRL2	16	0x00B0	16-bit
eTimer 1	CH5_CTRL3	16	0x00B2	16-bit
eTimer 1	CH5_CCCTRL	16	0x00BC	16-bit
FlexPWM—Base address: 0xFFE2_4000 53 registers to protect				
FlexPWM	SUB0_CTRL2	16	0x0004	16-bit
FlexPWM	SUB0_CTRL1	16	0x0006	16-bit
FlexPWM	SUB0_OCTRL	16	0x0018	16-bit
FlexPWM	SUB0_INTEN	16	0x001C	16-bit
FlexPWM	SUB0_DMAEN	16	0x001E	16-bit
FlexPWM	SUB0_TCTRL	16	0x0020	16-bit
FlexPWM	SUB0_DISMAP	16	0x0022	16-bit
FlexPWM	SUB0_DTCNT0	16	0x0024	16-bit
FlexPWM	SUB0_DTCNT1	16	0x0026	16-bit
FlexPWM	SUB1_CTRL2	16	0x0054	16-bit
FlexPWM	SUB1_CTRL1	16	0x0056	16-bit
FlexPWM	SUB1_OCTRL	16	0x0068	16-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
FlexPWM	SUB1_INTEN	16	0x006C	16-bit
FlexPWM	SUB1_DMAEN	16	0x006E	16-bit
FlexPWM	SUB1_TCTRL	16	0x0070	16-bit
FlexPWM	SUB1_DISMAP	16	0x0072	16-bit
FlexPWM	SUB1_DTCNT0	16	0x0074	16-bit
FlexPWM	SUB1_DTCNT1	16	0x0076	16-bit
FlexPWM	SUB2_CTRL2	16	0x00A4	16-bit
FlexPWM	SUB2_CTRL1	16	0x00A6	16-bit
FlexPWM	SUB2_OCTRL	16	0x00B8	16-bit
FlexPWM	SUB2_INTEN	16	0x00BC	16-bit
FlexPWM	SUB2_DMAEN	16	0x00BE	16-bit
FlexPWM	SUB2_TCTRL	16	0x00C0	16-bit
FlexPWM	SUB2_DISMAP	16	0x00C2	16-bit
FlexPWM	SUB2_DTCNT0	16	0x00C4	16-bit
FlexPWM	SUB2_DTCNT1	16	0x00C6	16-bit
FlexPWM	SUB3_CTRL2	16	0x00F4	16-bit
FlexPWM	SUB3_CTRL1	16	0x00F6	16-bit
FlexPWM	SUB3_OCTRL	16	0x0108	16-bit
FlexPWM	SUB3_INTEN	16	0x010C	16-bit
FlexPWM	SUB3_DMAEN	16	0x010E	16-bit
FlexPWM	SUB3_TCTRL	16	0x0110	16-bit
FlexPWM	SUB3_DISMAP	16	0x0112	16-bit
FlexPWM	SUB3_DTCNT0	16	0x0114	16-bit
FlexPWM	SUB3_DTCNT1	16	0x0116	16-bit
FlexPWM	MASK	16	0x0142	16-bit
FlexPWM	SWCOUT	16	0x0144	16-bit
FlexPWM	DTSRCSEL	16	0x0146	16-bit
FlexPWM	MCTRL	16	0x0148	16-bit
FlexPWM	FCTRL	16	0x014C	16-bit
DSPI 0—Base address: 0xFFF9_0000 11 registers to protect				
DSPI 0	DSPI_MCR	32	0x0000	32-bit
DSPI 0	DSPI_TCR	32	0x0008	32-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
DSPI 0	DSPI_CTAR0	32	0x000C	32-bit
DSPI 0	DSPI_CTAR1	32	0x0010	32-bit
DSPI 0	DSPI_CTAR2	32	0x0014	32-bit
DSPI 0	DSPI_CTAR3	32	0x0018	32-bit
DSPI 0	DSPI_CTAR4	32	0x001C	32-bit
DSPI 0	DSPI_CTAR5	32	0x0020	32-bit
DSPI 0	DSPI_CTAR6	32	0x0024	32-bit
DSPI 0	DSPI_CTAR7	32	0x0028	32-bit
DSPI 0	DSPI_RSER	32	0x0030	32-bit
DSPI 1—Base address: 0xFFF9_4000 11 registers to protect				
DSPI 1	DSPI_MCR	32	0x0000	32-bit
DSPI 1	DSPI_TCR	32	0x0008	32-bit
DSPI 1	DSPI_CTAR0	32	0x000C	32-bit
DSPI 1	DSPI_CTAR1	32	0x0010	32-bit
DSPI 1	DSPI_CTAR2	32	0x0014	32-bit
DSPI 1	DSPI_CTAR3	32	0x0018	32-bit
DSPI 1	DSPI_CTAR4	32	0x001C	32-bit
DSPI 1	DSPI_CTAR5	32	0x0020	32-bit
DSPI 1	DSPI_CTAR6	32	0x0024	32-bit
DSPI 1	DSPI_CTAR7	32	0x0028	32-bit
DSPI 1	DSPI_RSER	32	0x0030	32-bit
DSPI 2—Base address: 0xFFF9_8000 11 registers to protect				
DSPI 2	DSPI_MCR	32	0x0000	32-bit
DSPI 2	DSPI_TCR	32	0x0008	32-bit
DSPI 2	DSPI_CTAR0	32	0x000C	32-bit
DSPI 2	DSPI_CTAR1	32	0x0010	32-bit
DSPI 2	DSPI_CTAR2	32	0x0014	32-bit
DSPI 2	DSPI_CTAR3	32	0x0018	32-bit
DSPI 2	DSPI_CTAR4	32	0x001C	32-bit
DSPI 2	DSPI_CTAR5	32	0x0020	32-bit
DSPI 2	DSPI_CTAR6	32	0x0024	32-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
DSPI 2	DSPI_CTAR7	32	0x0028	32-bit
DSPI 2	DSPI_RSER	32	0x0030	32-bit
DSPI 3—Base address: 0xFFF9_C000 11 registers to protect				
DSPI 3	DSPI_MCR	32	0x0000	32-bit
DSPI 3	DSPI_TCR	32	0x0008	32-bit
DSPI 3	DSPI_CTAR0	32	0x000C	32-bit
DSPI 3	DSPI_CTAR1	32	0x0010	32-bit
DSPI 3	DSPI_CTAR2	32	0x0014	32-bit
DSPI 3	DSPI_CTAR3	32	0x0018	32-bit
DSPI 3	DSPI_CTAR4	32	0x001C	32-bit
DSPI 3	DSPI_CTAR5	32	0x0020	32-bit
DSPI 3	DSPI_CTAR6	32	0x0024	32-bit
DSPI 3	DSPI_CTAR7	32	0x0028	32-bit
DSPI 3	DSPI_RSER	32	0x0030	32-bit
FlexCAN—Base address: 0xFFFC_0000 7 registers to protect				
FlexCAN	CANx_MCR	32	0x0000	32-bit
FlexCAN	CANx_CTRL	32	0x0004	32-bit
FlexCAN	CANx_RXGMASK	32	0x0010	32-bit
FlexCAN	CANx_RX14MASK	32	0x0014	32-bit
FlexCAN	CANx_RX15MASK	32	0x0018	32-bit
FlexCAN	CANx_IMASK1	32	0x0028	32-bit
FlexRay—Base address: 0xFFFE_0000 1 register to protect				
FlexRay	MCR	16	0x0002	16-bit
Safety Port—Base address: 0xFFFE_8000 7 registers to protect				
Safety port	CANx_MCR	32	0x0000	32-bit
Safety port	CANx_CTRL	32	0x0004	32-bit
Safety port	CANx_RXGMASK	32	0x0010	32-bit
Safety port	CANx_RX14MASK	32	0x0014	32-bit

Table 608. Registers under protection(Continued)

Module	Register	Register size (bits)	Register offset	Protected bitfields
Safety port	CANx_RX15MASK	32	0x0018	32-bit
Safety port	CANx_IMASK1	32	0x0028	32-bit

Appendix B Memory Map

Table 609. Detailed Memory Map

Address	Register description	Register name	Size (bits)
0xC3F8_8000	**Code Flash Configuration** *Section 17.3.6: Registers description*		
Base + 0x0000	Module Configuration Register	MCR	32
Base + 0x0004	Low/Mid Address Space Block Locking Register	LML	32
Base + (0x0008 – 0x000B)	Reserved	—	—
Base + 0x000C	Secondary Low/Mid Address Space Block Locking Register	SLL	32
Base + 0x0010	Low/Mid Address Space Block Select Register	LMS	32
Base + (0x0014 – 0x0017)	Reserved	—	—
Base + 0x0018	Address Register	ADR	32
Base + 0x001C	Platform Flash Configuration Register 0	PFCR0	32
Base + 0x0020	Platform Flash Configuration Register 1	PFCR1	32
Base + 0x0024	Platform Flash Access Protection Register	PFAPR	32
Base + (0x0028 – 0x003B)	Reserved	—	—
Base + 0x003C	User Test Register 0	UT0	32
Base + 0x0040	User Test Register 1	UT1	32
Base + 0x0044	User Test Register 2	UT2	32
Base + 0x0048	User Multiple Input Signature Register 0	UMISR0	32
Base + 0x004C	User Multiple Input Signature Register 1	UMISR1	32
Base + 0x0050	User Multiple Input Signature Register 2	UMISR2	32
Base + 0x0054	User Multiple Input Signature Register 3	UMISR3	32
Base + 0x0058	User Multiple Input Signature Register 4	UMISR4	32
Base + (0x005C – 0x3FFF)	Reserved	—	—
0xC3F8_C000	**Data Flash Configuration** *Section 17.3.6: Registers description*		
Base + 0x0000	Module Configuration Register	MCR	32
Base + 0x0004	Low/Mid Address Space Block Locking Register	LML	32
Base + (0x0008 – 0x000B)	Reserved	—	—
Base + 0x000C	Secondary Low/Mid Address Space Block Locking Register	SLL	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0010	Low/Mid Address Space Block Select Register	LMS	32
Base + (0x0014 – 0x0017)	Reserved	—	—
Base + 0x0018	Address Register	ADR	32
Base + (0x001C – 0x003B)	Reserved	—	—
Base + 0x003C	User Test Register 0	UT0	32
Base + 0x0040	User Test Register 1	UT1	32
Base + 0x0044	User Test Register 2	UT2	32
Base + 0x0048	User Multiple Input Signature Register 0	UMISR0	32
Base + 0x004C	User Multiple Input Signature Register 1	UMISR1	32
Base + 0x0050	User Multiple Input Signature Register 2	UMISR2	32
Base + 0x0054	User Multiple Input Signature Register 3	UMISR3	32
Base + 0x0058	User Multiple Input Signature Register 4	UMISR4	32
Base + (0x005C – 0x3FFF)	Reserved	—	—
0xC3F9_0000	**System Integration Unit Lite (SIUL)** *Section 11.5: Memory map and register description*		
Base + (0x0000 – 0x0003)	Reserved	—	—
Base + 0x0004	MCU ID Register #1	MIDR1	32
Base + 0x0008	MCU ID Register #2	MIDR2	32
Base + (0x000C – 0x0013)	Reserved	—	—
Base + 0x0014	Interrupt Status Flag Register	ISR	32
Base + 0x0018	Interrupt Request Enable Register	IRER	32
Base + (0x001C – 0x0027)	Reserved	—	—
Base + 0x0028	Interrupt Rising Edge Event Enable Register	IREER	32
Base + 0x002C	Interrupt Falling-Edge Event Enable Register	IFEER	32
Base + 0x0030	IFER Interrupt Filter Enable Register	IFER	32
Base + (0x0034 – 0x003F)	Reserved	—	—
Base + 0x0040	Pad Configuration Register 0	PCR0	16
Base + 0x0042	Pad Configuration Register 1	PCR1	16
Base + 0x0044	Pad Configuration Register 2	PCR2	16
Base + 0x0046	Pad Configuration Register 3	PCR3	16
Base + 0x0048	Pad Configuration Register 4	PCR4	16
Base + 0x004A	Pad Configuration Register 5	PCR5	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x004C	Pad Configuration Register 6	PCR6	16
Base + 0x004E	Pad Configuration Register 7	PCR7	16
Base + 0x0050	Pad Configuration Register 8	PCR8	16
Base + 0x0052	Pad Configuration Register 9	PCR9	16
Base + 0x0054	Pad Configuration Register 10	PCR10	16
Base + 0x0056	Pad Configuration Register 11	PCR11	16
Base + 0x0058	Pad Configuration Register 12	PCR12	16
Base + 0x005A	Pad Configuration Register 13	PCR13	16
Base + 0x005C	Pad Configuration Register 14	PCR14	16
Base + 0x005E	Pad Configuration Register 15	PCR15	16
Base + 0x0060	Pad Configuration Register 16	PCR16	16
Base + 0x0062	Pad Configuration Register 17	PCR17	16
Base + 0x0064	Pad Configuration Register 18	PCR18	16
Base + 0x0066	Pad Configuration Register 19	PCR19	16
Base + 0x0068	Pad Configuration Register 20	PCR20	16
Base + 0x006A	Pad Configuration Register 21	PCR21	16
Base + 0x006C	Pad Configuration Register 22	PCR22	16
Base + 0x006E	Pad Configuration Register 23	PCR23	16
Base + 0x0070	Pad Configuration Register 24	PCR24	16
Base + 0x0072	Pad Configuration Register 25	PCR25	16
Base + 0x0074	Pad Configuration Register 26	PCR26	16
Base + 0x0076	Pad Configuration Register 27	PCR27	16
Base + 0x0078	Pad Configuration Register 28	PCR28	16
Base + 0x007A	Pad Configuration Register 29	PCR29	16
Base + 0x007C	Pad Configuration Register 30	PCR30	16
Base + 0x007E	Pad Configuration Register 31	PCR31	16
Base + 0x0080	Pad Configuration Register 32	PCR32	16
Base + 0x0082	Pad Configuration Register 33	PCR33	16
Base + 0x0084	Pad Configuration Register 34	PCR34	16
Base + 0x0086	Pad Configuration Register 35	PCR35	16
Base + 0x0088	Pad Configuration Register 36	PCR36	16
Base + 0x008A	Pad Configuration Register 37	PCR37	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x008C	Pad Configuration Register 38	PCR38	16
Base + 0x008E	Pad Configuration Register 39	PCR39	16
Base + 0x0090	Pad Configuration Register 40	PCR40	16
Base + 0x0092	Pad Configuration Register 41	PCR41	16
Base + 0x0094	Pad Configuration Register 42	PCR42	16
Base + 0x0096	Pad Configuration Register 43	PCR43	16
Base + 0x0098	Pad Configuration Register 44	PCR44	16
Base + 0x009A	Pad Configuration Register 45	PCR45	16
Base + 0x009C	Pad Configuration Register 46	PCR46	16
Base + 0x009E	Pad Configuration Register 47	PCR47	16
Base + 0x00A0	Pad Configuration Register 48	PCR48	16
Base + 0x00A2	Pad Configuration Register 49	PCR49	16
Base + 0x00A4	Pad Configuration Register 50	PCR50	16
Base + 0x00A6	Pad Configuration Register 51	PCR51	16
Base + 0x00A8	Pad Configuration Register 52	PCR52	16
Base + 0x00AA	Pad Configuration Register 53	PCR53	16
Base + 0x00AC	Pad Configuration Register 54	PCR54	16
Base + 0x00AE	Pad Configuration Register 55	PCR55	16
Base + 0x00B0	Pad Configuration Register 56	PCR56	16
Base + 0x00B2	Pad Configuration Register 57	PCR57	16
Base + 0x00B4	Pad Configuration Register 58	PCR58	16
Base + 0x00B6	Pad Configuration Register 59	PCR59	16
Base + 0x00B8	Pad Configuration Register 60	PCR60	16
Base + 0x00BA	Pad Configuration Register 61	PCR61	16
Base + 0x00BC	Pad Configuration Register 62	PCR62	16
Base + 0x00BE	Pad Configuration Register 63	PCR63	16
Base + 0x00C0	Pad Configuration Register 64	PCR64	16
Base + 0x00C2	Pad Configuration Register 65	PCR65	16
Base + 0x00C4	Pad Configuration Register 66	PCR66	16
Base + 0x00C6	Pad Configuration Register 67	PCR67	16
Base + 0x00C8	Pad Configuration Register 68	PCR68	16
Base + 0x00CA	Pad Configuration Register 69	PCR69	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00CC	Pad Configuration Register 70	PCR70	16
Base + 0x00CE	Pad Configuration Register 71	PCR71	16
Base + 0x00D0	Pad Configuration Register 72	PCR72	16
Base + 0x00D2	Pad Configuration Register 73	PCR73	16
Base + 0x00D4	Pad Configuration Register 74	PCR74	16
Base + 0x00D6	Pad Configuration Register 75	PCR75	16
Base + 0x00D8	Pad Configuration Register 76	PCR76	16
Base + 0x00DA	Pad Configuration Register 77	PCR77	16
Base + 0x00DC	Pad Configuration Register 78	PCR78	16
Base + 0x00DE	Pad Configuration Register 79	PCR79	16
Base + 0x00E0	Pad Configuration Register 80	PCR80	16
Base + 0x00E2	Pad Configuration Register 81	PCR81	16
Base + 0x00E4	Pad Configuration Register 82	PCR82	16
Base + 0x00E6	Pad Configuration Register 83	PCR83	16
Base + 0x00E8	Pad Configuration Register 84	PCR84	16
Base + 0x00EA	Pad Configuration Register 85	PCR85	16
Base + 0x00EC	Pad Configuration Register 86	PCR86	16
Base + 0x00EE	Pad Configuration Register 87	PCR87	16
Base + 0x00F0	Pad Configuration Register 88	PCR88	16
Base + 0x00F2	Pad Configuration Register 89	PCR89	16
Base + 0x00F4	Pad Configuration Register 90	PCR90	16
Base + 0x00F6	Pad Configuration Register 91	PCR91	16
Base + 0x00F8	Pad Configuration Register 92	PCR92	16
Base + 0x00FA	Pad Configuration Register 93	PCR93	16
Base + 0x00FC	Pad Configuration Register 94	PCR94	16
Base + 0x00FE	Pad Configuration Register 95	PCR95	16
Base + 0x0100	Pad Configuration Register 96	PCR96	16
Base + 0x0102	Pad Configuration Register 97	PCR97	16
Base + 0x0104	Pad Configuration Register 98	PCR98	16
Base + 0x0106	Pad Configuration Register 99	PCR99	16
Base + 0x0108	Pad Configuration Register 100	PCR100	16
Base + 0x010A	Pad Configuration Register 101	PCR101	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x010C	Pad Configuration Register 102	PCR102	16
Base + 0x010E	Pad Configuration Register 103	PCR103	16
Base + 0x0110	Pad Configuration Register 104	PCR104	16
Base + 0x0112	Pad Configuration Register 105	PCR105	16
Base + 0x0114	Pad Configuration Register 106	PCR106	16
Base + 0x0116	Pad Configuration Register 107	PCR107	16
Base + (0x0118 – 0x04FF)	Reserved	—	—
Base + 0x0500	Pad Selection for Multiplexed Inputs 0_3	PSMI0_3	32
Base + 0x0504	Pad Selection for Multiplexed Inputs 4_7	PSMI4_7	32
Base + 0x0508	Pad Selection for Multiplexed Inputs 8_11	PSMI8_11	32
Base + 0x050C	Pad Selection for Multiplexed Inputs 12_15	PSMI12_15	32
Base + 0x0510	Pad Selection for Multiplexed Inputs 16_19	PSMI16_19	32
Base + 0x0514	Pad Selection for Multiplexed Inputs 20_23	PSMI20_23	32
Base + 0x0518	Pad Selection for Multiplexed Inputs 24_27	PSMI24_27	32
Base + 0x051C	Pad Selection for Multiplexed Inputs 28_31	PSMI28_31	32
Base + 0x0520	Pad Selection for Multiplexed Inputs 32_35	PSMI32_35	32
Base + (0x0524 – 0x05FF)	Reserved	—	—
Base + 0x0600	GPIO Pad Data Output Register 0_3	GPDO0_3	32
Base + 0x0604	GPIO Pad Data Output Register 4_7	GPDO4_7	32
Base + 0x0608	GPIO Pad Data Output Register 8_11	GPDO8_11	32
Base + 0x060C	GPIO Pad Data Output Register 12_15	GPDO12_15	32
Base + 0x0610	GPIO Pad Data Output Register 16_19	GPDO16_19	32
Base + 0x0614	GPIO Pad Data Output Register 20_23	GPDO20_23	32
Base + 0x0618	GPIO Pad Data Output Register 24_27	GPDO24_27	32
Base + 0x061C	GPIO Pad Data Output Register 28_31	GPDO28_31	32
Base + 0x0620	GPIO Pad Data Output Register 32_35	GPDO32_35	32
Base + 0x0624	GPIO Pad Data Output Register 36_39	GPDO36_39	32
Base + 0x0628	GPIO Pad Data Output Register 40_43	GPDO40_43	32
Base + 0x062C	GPIO Pad Data Output Register 44_47	GPDO44_47	32
Base + 0x0630	GPIO Pad Data Output Register 48_51	GPDO48_51	32
Base + 0x0634	GPIO Pad Data Output Register 52_55	GPDO52_55	32
Base + 0x0638	GPIO Pad Data Output Register 56_59	GPDO56_59	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x063C	GPIO Pad Data Output Register 60_63	GPDO60_63	32
Base + 0x0640	GPIO Pad Data Output Register 64_67	GPDO64_67	32
Base + 0x0644	GPIO Pad Data Output Register 68_71	GPDO68_71	32
Base + 0x0648	GPIO Pad Data Output Register 72_75	GPDO72_75	32
Base + 0x064C	GPIO Pad Data Output Register 76_79	GPDO76_79	32
Base + 0x0650	GPIO Pad Data Output Register 80_83	GPDO80_83	32
Base + 0x0654	GPIO Pad Data Output Register 84_87	GPDO84_87	32
Base + 0x0658	GPIO Pad Data Output Register 88_91	GPDO88_91	32
Base + 0x065C	GPIO Pad Data Output Register 92_95	GPDO92_95	32
Base + 0x0660	GPIO Pad Data Output Register 96_99	GPDO96_99	32
Base + 0x0664	GPIO Pad Data Output Register 100_103	GPDO100_103	32
Base + 0x0668	GPIO Pad Data Output Register 104_107	GPDO104_107	32
Base + (0x066C – 0x07FF)	Reserved	—	—
Base + 0x0800	GPIO Pad Data Input Register 0_3	GPDI0_3	32
Base + 0x0804	GPIO Pad Data Input Register 4_7	GPDI4_7	32
Base + 0x0808	GPIO Pad Data Input Register 8_11	GPDI8_11	32
Base + 0x080C	GPIO Pad Data Input Register 12_15	GPDI12_15	32
Base + 0x0810	GPIO Pad Data Input Register 16_19	GPDI16_19	32
Base + 0x0814	GPIO Pad Data Input Register 20_23	GPDI20_23	32
Base + 0x0818	GPIO Pad Data Input Register 24_27	GPDI24_27	32
Base + 0x081C	GPIO Pad Data Input Register 28_31	GPDI28_31	32
Base + 0x0820	GPIO Pad Data Input Register 32_35	GPDI32_35	32
Base + 0x0824	GPIO Pad Data Input Register 36_39	GPDI36_39	32
Base + 0x0828	GPIO Pad Data Input Register 40_43	GPDI40_43	32
Base + 0x082C	GPIO Pad Data Input Register 44_47	GPDI44_47	32
Base + 0x0830	GPIO Pad Data Input Register 48_51	GPDI48_51	32
Base + 0x0834	GPIO Pad Data Input Register 52_55	GPDI52_55	32
Base + 0x0838	GPIO Pad Data Input Register 56_59	GPDI56_59	32
Base + 0x083C	GPIO Pad Data Input Register 60_63	GPDI60_63	32
Base + 0x0840	GPIO Pad Data Input Register 64_67	GPDI64_67	32
Base + 0x0844	GPIO Pad Data Input Register 68_71	GPDI68_71	32
Base + 0x0848	GPIO Pad Data Input Register 72_75	GPDI72_75	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x084C	GPIO Pad Data Input Register 76_79	GPDI76_79	32
Base + 0x0850	GPIO Pad Data Input Register 80_83	GPDI80_83	32
Base + 0x0854	GPIO Pad Data Input Register 84_87	GPDI84_87	32
Base + 0x0858	GPIO Pad Data Input Register 88_91	GPDI88_91	32
Base + 0x085C	GPIO Pad Data Input Register 92_95	GPDI92_95	32
Base + 0x0860	GPIO Pad Data Input Register 96_99	GPDI96_99	32
Base + 0x0864	GPIO Pad Data Input Register 100_103	GPDI100_103	32
Base + 0x0868	GPIO Pad Data Input Register 104_107	GPDI104_107	32
Base + (0x086C – 0x0BFF)	Reserved	—	—
Base + 0x0C00	Parallel GPIO Pad Data Out Register 0	PGPDO0	32
Base + 0x0C04	Parallel GPIO Pad Data Out Register 1	PGPDO1	32
Base + 0x0C08	Parallel GPIO Pad Data Out Register 2	PGPDO2	32
Base + 0x0C0C	Parallel GPIO Pad Data Out Register 3	PGPDO3	32
Base + (0x0C10 – 0x0C3F)	Reserved	—	—
Base + 0x0C40	Parallel GPIO Pad Data In Register 0	PGPDI0	32
Base + 0x0C44	Parallel GPIO Pad Data In Register 1	PGPDI1	32
Base + 0x0C48	Parallel GPIO Pad Data In Register 2	PGPDI2	32
Base + 0x0C4C	Parallel GPIO Pad Data In Register 3	PGPDI3	32
Base + (0x0C50 – 0x0C7F)	Reserved	—	—
Base + 0x0C80	Masked Parallel GPIO Pad Data Out Register 0	MPGPDO0	32
Base + 0x0C84	Masked Parallel GPIO Pad Data Out Register 1	MPGPDO1	32
Base + 0x0C88	Masked Parallel GPIO Pad Data Out Register 2	MPGPDO2	32
Base + 0x0C8C	Masked Parallel GPIO Pad Data Out Register 3	MPGPDO3	32
Base + 0x0C90	Masked Parallel GPIO Pad Data Out Register 4	MPGPDO4	32
Base + 0x0C94	Masked Parallel GPIO Pad Data Out Register 5	MPGPDO5	32
Base + 0x0C98	Masked Parallel GPIO Pad Data Out Register 6	MPGPDO6	32
Base + (0x0C9C – 0x0FFF)	Reserved	—	—
Base + 0x1000	Interrupt Filter Maximum Counter Register 0	IFMC0	32
Base + 0x1004	Interrupt Filter Maximum Counter Register 1	IFMC1	32
Base + 0x1008	Interrupt Filter Maximum Counter Register 2	IFMC2	32
Base + 0x100C	Interrupt Filter Maximum Counter Register 3	IFMC3	32
Base + 0x1010	Interrupt Filter Maximum Counter Register 4	IFMC4	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x1014	Interrupt Filter Maximum Counter Register 5	FMC5	32
Base + 0x1018	Interrupt Filter Maximum Counter Register 6	IFMC6	32
Base + 0x101C	Interrupt Filter Maximum Counter Register 7	IFMC7	32
Base + 0x1020	Interrupt Filter Maximum Counter Register 8	IFMC8	32
Base + 0x1024	Interrupt Filter Maximum Counter Register 9	IFMC9	32
Base + 0x1028	Interrupt Filter Maximum Counter Register 10	IFMC10	32
Base + 0x102C	Interrupt Filter Maximum Counter Register 11	IFMC11	32
Base + 0x1030	Interrupt Filter Maximum Counter Register 12	IFMC12	32
Base + 0x1034	Interrupt Filter Maximum Counter Register 13	IFMC13	32
Base + 0x1038	Interrupt Filter Maximum Counter Register 14	IFMC14	32
Base + 0x103C	Interrupt Filter Maximum Counter Register 15	IFMC15	32
Base + 0x1040	Interrupt Filter Maximum Counter Register 16	IFMC16	32
Base + 0x1044	Interrupt Filter Maximum Counter Register 17	IFMC17	32
Base + 0x1048	Interrupt Filter Maximum Counter Register 18	IFMC18	32
Base + 0x104C	Interrupt Filter Maximum Counter Register 19	IFMC19	32
Base + 0x1050	Interrupt Filter Maximum Counter Register 29	IFMC20	32
Base + 0x1054	Interrupt Filter Maximum Counter Register 21	IFMC21	32
Base + 0x1058	Interrupt Filter Maximum Counter Register 22	IFMC22	32
Base + 0x105C	Interrupt Filter Maximum Counter Register 23	IFMC23	32
Base + 0x1060	Interrupt Filter Maximum Counter Register 24	IFMC24	32
Base + 0x1064	Interrupt Filter Maximum Counter Register 25	IFMC25	32
Base + 0x1068	Interrupt Filter Maximum Counter Register 26	IFMC26	32
Base + 0x106C	Interrupt Filter Maximum Counter Register 27	IFMC27	32
Base + 0x1070	Interrupt Filter Maximum Counter Register 28	IFMC28	32
Base + 0x1074	Interrupt Filter Maximum Counter Register 29	IFMC29	32
Base + 0x1078	Interrupt Filter Maximum Counter Register 30	IFMC30	32
Base + 0x107C	Interrupt Filter Maximum Counter Register 31	IFMC31	32
Base + 0x1080	Interrupt Filter Clock Prescaler Register	IFCP	32
Base + (0x1084 – 0x3FFF)	Reserved	—	—
0xC3F9_4000	**WakeUp Unit (WKPU)** *Section 30.4: Memory map and registers description*		
Base + 0x0000	NMI Status Flag Register	NSR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	NMI Configuration Register	NCR	32
Base + (0x000C – 0x3FFF)	Reserved	—	—
0xC3FD_8000	**System Status and Configuration Module (SSCM)** *Section 10.2: Memory map and register description*		
Base + 0x0000	System Status Register	STATUS	16
Base + 0x0002	System Memory Configuration Register	MEMCONFIG	16
Base + (0x0004 – 0x0005)	Reserved	—	—
Base + 0x0006	Error Configuration Register	ERROR	16
Base + 0x0008	Debug Status Port Register	DEBUGPORT	16
Base + (0x000A – 0x000B)	Reserved	—	—
Base + 0x000C	Password Comparison Register High Word Register	PWCMPH	32
Base + 0x0010	Password Comparison Register Low Word Register	PWCMPL	32
Base + (0x0014 – 0x3FFF)	Reserved	—	—
0xC3FD_C000	**Mode Entry Module (ME)** *Section 6.3: Memory map and registers description*		
Base + 0x0000	Global Status Register	ME_GS	32
Base + 0x0004	Mode Control Register	ME_MCTL	32
Base + 0x0008	Mode Enable Register	ME_ME	32
Base + 0x000C	Interrupt Status Register	ME_IS	32
Base + 0x0010	Interrupt Mask Register	ME_IM	32
Base + 0x0014	Invalid Mode Transition Status Register	ME_IMTS	32
Base + 0x0018	Debug Mode Transition Status Register	ME_DMTS	32
Base + (0x001C – 0x001F)	Reserved	—	—
Base + 0x0020	RESET Mode Configuration Register	ME_RESET_MC	32
Base + 0x0024	TEST Mode Configuration Register	ME_TEST_MC	32
Base + 0x0028	Mode Configuration Register	ME_SAFE_MC	32
Base + 0x002C	DRUN Mode Configuration Register	ME_DRUN_MC	32
Base + 0x0030	Mode Configuration Register 0	ME_RUN0_MC	32
Base + 0x0034	Mode Configuration Register 1	ME_RUN1_MC	32
Base + 0x0038	Mode Configuration Register 2	ME_RUN2_MC	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x003C	Mode Configuration Register 3	ME_RUN3_MC	32
Base + 0x0040	HALT0 Mode Configuration Register	ME_HALT0_MC	32
Base + (0x0044 – 0x0047)	Reserved	—	—
Base + 0x0048	STOP0 Mode Configuration Register	ME_STOP0_MC	32
Base + (0x004C – 0x005F)	Reserved	—	—
Base + 0x0060	Peripheral Status Register 0	ME_PS0	32
Base + 0x0064	Peripheral Status Register 1	ME_PS1	32
Base + 0x0068	Peripheral Status Register 2	ME_PS2	32
Base + (0x006C – 0x007F)	Reserved	—	—
Base + 0x0080	RUN Peripheral Configuration Register 0	ME_RUN_PC0	32
Base + 0x0084	RUN Peripheral Configuration Register 1	ME_RUN_PC1	32
Base + 0x0088	RUN Peripheral Configuration Register 2	ME_RUN_PC2	32
Base + 0x008C	RUN Peripheral Configuration Register 3	ME_RUN_PC3	32
Base + 0x0090	RUN Peripheral Configuration Register 4	ME_RUN_PC4	32
Base + 0x0094	RUN Peripheral Configuration Register 5	ME_RUN_PC5	32
Base + 0x0098	RUN Peripheral Configuration Register 6	ME_RUN_PC6	32
Base + 0x009C	RUN Peripheral Configuration Register 7	ME_RUN_PC7	32
Base + 0x00A0	Low Power Peripheral Configuration Register 0	ME_LP_PC0	32
Base + 0x00A4	Low Power Peripheral Configuration Register 1	ME_LP_PC1	32
Base + 0x00A8	Low Power Peripheral Configuration Register 2	ME_LP_PC2	32
Base + 0x00AC	Low Power Peripheral Configuration Register 3	ME_LP_PC3	32
Base + 0x00B0	Low Power Peripheral Configuration Register 4	ME_LP_PC4	32
Base + 0x00B4	Low Power Peripheral Configuration Register 5	ME_LP_PC5	32
Base + 0x00B8	Low Power Peripheral Configuration Register 6	ME_LP_PC6	32
Base + 0x00BC	Low Power Peripheral Configuration Register 7	ME_LP_PC7	32
Base + (0x00C0 – 0x00C3)	Reserved	—	—
Base + 0x00C4	DSPI0 Control Register	ME_PCTL4	8
Base + 0x00C5	DSPI1 Control Register	ME_PCTL5	8
Base + 0x00C6	DSPI2 Control Register	ME_PCTL6	8
Base + 0x00C7	DSPI3 Control Register	ME_PCTL7	8
Base + (0x00C8 – 0x00CF)	Reserved	—	—
Base + 0x00D0	FlexCAN0 Control Register	ME_PCTL16	8

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x00D1 – 0x00D7)	Reserved	—	—
Base + 0x00D8	FlexRay Control Register	ME_PCTL24	8
Base + 0x00D9	Reserved	—	—
Base + 0x00DA	SafetyPort Control Register	ME_PCTL26	8
Base + (0x00DB – 0x00DF)	Reserved	—	—
Base + 0x00E0	ADC0 Control Register	ME_PCTL32	8
Base + 0x00E1	ADC1 Control Register	ME_PCTL33	8
Base + 0x00E2	Reserved	—	—
Base + 0x00E3	CTU0 Control Register	ME_PCTL35	8
Base + (0x00E4 – 0x00E5)	Reserved	—	—
Base + 0x00E6	eTimer0 Control Register	ME_PCTL38	8
Base + 0x00E7	eTimer1 Control Register	ME_PCTL39	8
Base + 0x00E8	Reserved	—	—
Base + 0x00E9	FlexPWM0 Control Register	ME_PCTL41	8
Base + (0x00EA – 0x00EF)	Reserved	—	—
Base + 0x00F0	LINFlex0 Control Register	ME_PCTL48	8
Base + 0x00F1	LINFlex1 Control Register	ME_PCTL49	8
Base + (0x00F2 – 0x010B)	Reserved	—	—
Base + 0x011C	PIT_RTI Control Register	ME_PCTL92	8
Base + (0x011D – 0x3FFF)	Reserved	—	—
0xC3FE_0000	**XOSCHS** ***Section 4.8.2: Register description***		
Base + 0x0000	Oscillator Control Register	OSC_CTL	32
Base + (0x0004 – 0x005F)	Reserved	—	—
0xC3FE_0060	**RC Digital Interface Registers** ***Section 4.7: IRC 16 MHz internal RC oscillator (RC_CTL)***		
Base + 0x0000	IRC 16 MHz Internal RC Oscillator Control Register	RC_CTL	32
Base + (0x0004 – 0x001F)	Reserved	—	—
0xC3FE_00A0	**FMPLL_0** ***Section 4.9.4: Memory map***		
Base + 0x0000	Control Register	CR	32
Base + 0x0004	Modulation Register	MR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0008 – 0x001F)	Reserved	—	—
0xC3FE_00C0	**FMPLL_1** *Section 4.9.4: Memory map*		
Base + 0x0000	Control Register	CR	32
Base + 0x0004	Modulation Register	MR	32
Base + (0x0008 – 0x003F)	Reserved	—	—
0xC3FE_0100	**Clock Monitor Unit (CMU)** *Section 4.10.4: Memory map and register description*		
Base + 0x0000	Control Status Register	CMU_0_CSR	32
Base + 0x0004	Frequency Display Register	CMU_0_FDR	32
Base + 0x0008	High Frequency Reference Register	CMU_0_HFREFR_A	32
Base + 0x000C	Low Frequency Reference Register	CMU_0_LFREFR_A	32
Base + 0x0010	Interrupt Status Register	CMU_0_ISR	32
Base + 0x0014	Reserved	—	—
Base + 0x0018	Measurement Duration Register	CMU_0_MDR	32
Base + 0x001C	Reserved	—	—
Base + 0x0020	Control Status Register	*CMU_1_CSR*	32
Base + 0x0024	Reserved	—	—
Base + 0x0028	High Frequency Reference Register FMPLL_1	CMU_1_HFREFR_A	32
Base + 0x002C	Low Frequency Reference Register FMPLL_1	CMU_1_LFREFR_A	32
Base + 0x0030	Interrupt Status Register (CMU_1_ISR	*CMU_1_ISR*	32
Base + (0x0034 – 0x026F)	Reserved	—	—
0xC3FE_0370	**Clock Generation Module (CGM)** *Section 5.5: Memory map and registers description*		
Base + 0x0000	Output Clock Enable Register	CGM_OC_EN	32
Base + 0x0004	Output Clock Division Select Control Register	CGM_OCDS_SC	32
Base + 0x0008	System Clock Select Status Register	CGM_SC_SS	32
Base + (0x000C 0x000F)	Reserved	—	—
Base + 0x0010	Auxiliary Clock 0 Select Control Register	CGM_AC0_SC	32
Base + 0x0014	Auxiliary Clock 0 Divider Configuration Register	CGM_AC0_DC0	32
Base + 0x0018	Auxiliary Clock 1 Select Control Register	CGM_AC1_SC	32
Base + 0x001C	Auxiliary Clock 1 Divider Configuration 0 Register	CGM_AC1_DC0	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0020	Auxiliary Clock 2 Select Control Register	CGM_AC2_SC	32
Base + 0x0024	Auxiliary Clock 2 Divider Configuration 0 Register	CGM_AC2_DC0	32
Base + 0x0028	Auxiliary Clock 3 Select Control Register	CGM_AC3_SC	32
Base + 0x002C	Auxiliary Clock 3 Divider Configuration 0 Register	CGM_AC3_DC0	32
Base + (0x0030 – 0x3C8F)	Reserved	—	—
0xC3FE_4000	**Reset Generation Module (RGM)** *Section 8.5: Memory map and registers description*		
Base + 0x0000	Functional Event Status Register	RGM_FES	16
Base + 0x0002	Destructive Event Status Register	RGM_DES	16
Base + 0x0004	Functional Event Reset Disable Register	RGM_FERD	16
Base + 0x0006	Destructive Event Reset Disable Register	RGM_DERD	16
Base + (0x0008 – 0x000F)	Reserved	—	—
Base + 0x0010	Functional Event Alternate Request Register	RGM_FEAR	16
Base + (0x0012 – 0x0017)	Reserved	—	—
Base + 0x0018	Functional Event Short Sequence Register	RGM_FESS	16
Base + (0x001A – 0x001B)	Reserved	—	—
Base + 0x001C	Functional Bidirectional Reset Enable Register	RGM_FBRE	16
Base + (0x001E – 0x3FFF)	Reserved	—	—
0xC3FE_8000	**Power Control Unit (PCU)** *Section 7.3: Memory map and register definition*		
Base + 0x0000	Power Domain #0 Configuration Register	PCU_PCONF0	32
Base + (0x0000 – 0x003F)	Reserved	—	—
Base + 0x0040	Power Domain Status Register	PCU_PSTAT	32
Base + (0x0044 – 0x007F)	Reserved	—	—
0xC3FE_8000	**Voltage Regulator (VREG)** *Section 35.1.4: Registers Description*		
Base + 0x0080	Voltage Regulator Control Register	VREG_CTL	32
Base + 0x0084	Voltage Regulator Status Register	VREG_STATUS	32
Base + (0x0088 – 0x7FFF)	Reserved	—	—
0xC3FF_0000	**Periodic Interrupt Timer (PIT)** *Section 31.4: Memory map and registers description*		
Base + 0x0000	PIT Module Control Register	PITMCR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0004 – 0x00FC)	Reserved	—	—
Base + 0x0100	Timer 0 Load Value Register	LDVAL0	32
Base + 0x0104	Timer 0 Current Value Register	CVAL0	32
Base + 0x0108	Timer 0 Control Register	TCTRL0	32
Base + 0x010C	Timer 0 Timer Flag Register	TFLG0	32
Base + 0x0110	Timer 1 Load Value Register	LDVAL1	32
Base + 0x0114	Timer 1 Current Value Register	CVAL1	32
Base + 0x0118	Timer 1 Control Register	TCTRL1	32
Base + 0x011C	Timer 1 Timer Flag Register	TFLG1	32
Base + 0x0120	Timer 2 Load Value Register	CLDVAL2	32
Base + 0x0124	Timer 2 Current Value Register	CVAL2	32
Base + 0x0128	Timer 2 Control Register	TCTRL2	32
Base + 0x012C	Timer 2 Timer Flag Register	TFLG2	32
Base + 0x0130	Timer 3 Load Value Register	LDVAL3	32
Base + 0x0134	Timer 3 Current Value Register	CVAL3	32
Base + 0x0138	Timer 3 Control Register	TCTRL3	32
Base + 0x013C	Timer 3 Timer Flag Register	TFLG3	32
Base + (0x0140 – 0x3FFF)	Reserved	—	—
0xFFE0_0000	**ADC_0** *Section 24.4: Register descriptions*		
Base + 0x0000	Main Configuration Register	MCR	32
Base + 0x0004	Main Status Register	MSR	32
Base + (0x0008 –0x000F)	Reserved	—	—
Base + 0x0010	Interrupt Status Register	ISR	32
Base + (0x0014 –0x001F)	Reserved	—	—
Base + 0x0020	Interrupt Mask Register	IMR	32
Base + (0x0024 –0x002F)	Reserved	—	—
Base + 0x0030	Watchdog Threshold Interrupt Status Register	WTISR	32
Base + 0x0034	Watchdog Threshold Interrupt Mask Register	WTIMR	32
Base + (0x0038 – 0x003F)	Reserved	—	—
Base + 0x0040	DMA Enable Register	DMAE	32
Base + 0x0044	DMA Channel Select Register 0	DMAR0	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0048 – 0x004F)	Reserved	—	—
Base + 0x0050	Threshold Control Register 0	TRC0	32
Base + 0x0054	Threshold Control Register 1	TRC1	32
Base + 0x0058	Threshold Control Register 2	TRC2	32
Base + 0x005C	Threshold Control Register 3	TRC3	32
Base + 0x0060	Threshold Register 0	THRHLR0	32
Base + 0x0064	Threshold Register 1	THRHLR1	32
Base + 0x0068	Threshold Register 2	THRHLR2	32
Base + 0x006C	Threshold Register 3	THRHLR3	32
Base + (0x0070–0x0093)	Reserved	—	—
Base + 0x0094	Conversion Timing Register 0	CTR0	32
Base + (0x0098–0x00A3)	Reserved	—	—
Base + 0x00A4	Normal Conversion Mask Register 0	NCMR0	32
Base + (0x00A8–0x00B3)	Reserved	—	—
Base + 0x00B4	Injected Conversion Mask Register 0	JCMR0	32
Base + (0x00B8–0x00C7)	Reserved	—	—
Base + 0x00C8	Power-down Exit Delay Register	PDEDR	32
Base + (0x00CC–0x00FF)	Reserved	—	—
Base + 0x0100	Channel 0 Data Register	CDR0	32
Base + 0x0104	Channel 1 Data Register	CDR1	32
Base + 0x0108	Channel 2 Data Register	CDR2	32
Base + 0x010C	Channel 3 Data Register	CDR3	32
Base + 0x0110	Channel 4 Data Register	CDR4	32
Base + 0x0114	Channel 5 Data Register	CDR5	32
Base + 0x0118	Channel 6 Data Register	CDR6	32
Base + 0x011C	Channel 7 Data Register	CDR7	32
Base + 0x0120	Channel 8 Data Register	CDR8	32
Base + 0x0124	Channel 9 Data Register	CDR9	32
Base + 0x0128	Channel 10 Data Register	CDR10	32
Base + 0x012C	Channel 11 Data Register	CDR11	32
Base + 0x0130	Channel 12 Data Register	CDR12	32
Base + 0x0134	Channel 13 Data Register	CDR13	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0138	Channel 14 Data Register	CDR14	32
Base + 0x013C	Channel 15 Data Register	CDR15	32
Base + (0x0140–0x3FFF)	Reserved	—	—
0xFFE0_4000	**ADC 1** *Section 24.4: Register descriptions*		
Base + 0x0000	Main Configuration Register	MCR	32
Base + 0x0004	Main Status Register	MSR	32
Base + (0x0008 –0x000F)	Reserved	—	—
Base + 0x0010	Interrupt Status Register	ISR	32
Base + (0x0014 –0x001F)	Reserved	—	—
Base + 0x0020	Interrupt Mask Register	IMR	32
Base + (0x0024 –0x002F)	Reserved	—	—
Base + 0x0030	Watchdog Threshold Interrupt Status Register	WTISR	32
Base + 0x0034	Watchdog Threshold Interrupt Mask Register	WTIMR	32
Base + (0x0038 – 0x003F)	Reserved	—	—
Base + 0x0040	DMA Enable Register	DMAE	32
Base + 0x0044	DMA Channel Select Register 0	DMAR0	32
Base + (0x0048 – 0x004F)	Reserved	—	—
Base + 0x0050	Threshold Control Register 0	TRC0	32
Base + 0x0054	Threshold Control Register 1	TRC1	32
Base + 0x0058	Threshold Control Register 2	TRC2	32
Base + 0x005C	Threshold Control Register 3	TRC3	32
Base + 0x0060	Threshold Register 0	THRHLR0	32
Base + 0x0064	Threshold Register 1	THRHLR1	32
Base + 0x0068	Threshold Register 2	THRHLR2	32
Base + 0x006C	Threshold Register 3	THRHLR3	32
Base + (0x0070–0x0093)	Reserved	—	—
Base + 0x0094	Conversion Timing Register 0	CTR0	32
Base + (0x0098–0x00A3)	Reserved	—	—
Base + 0x00A4	Normal Conversion Mask Register 0	NCMR0	32
Base + (0x00A8–0x00B3)	Reserved	—	—
Base + 0x00B4	Injected Conversion Mask Register 0	JCMR0	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x00B8–0x00C7)	Reserved	—	—
Base + 0x00C8	Power-down Exit Delay Register	PDEDR	32
Base + (0x00CC–0x00FF)	Reserved	—	—
Base + 0x0100	Channel 0 Data Register	CDR0	32
Base + 0x0104	Channel 1 Data Register	CDR1	32
Base + 0x0108	Channel 2 Data Register	CDR2	32
Base + 0x010C	Channel 3 Data Register	CDR3	32
Base + 0x0110	Channel 4 Data Register	CDR4	32
Base + 0x0114	Channel 5 Data Register	CDR5	32
Base + 0x0118	Channel 6 Data Register	CDR6	32
Base + 0x011C	Channel 7 Data Register	CDR7	32
Base + 0x0120	Channel 8 Data Register	CDR8	32
Base + 0x0124	Channel 9 Data Register	CDR9	32
Base + 0x0128	Channel 10 Data Register	CDR10	32
Base + 0x012C	Channel 11 Data Register	CDR11	32
Base + 0x0130	Channel 12 Data Register	CDR12	32
Base + 0x0134	Channel 13 Data Register	CDR13	32
Base + 0x0138	Channel 14 Data Register	CDR14	32
Base + (0x01–0x3FFF)	Reserved	—	—
0xFFE0_C000	**Cross Triggering Unit (CTU)** *Section 25.8: Memory map*		
Base + 0x0000	Trigger Generator Subunit Input Selection Register	TGSISR	32
Base + 0x0004	Trigger Generator Subunit Control Register	TGSCR	16
Base + 0x0006	Trigger 0 Compare Register	T0CR	16
Base + 0x0008	Trigger 1 Compare Register	T1CR	16
Base + 0x000A	Trigger 2 Compare Register	T2CR	16
Base + 0x000C	Trigger 3 Compare Register	T3CR	16
Base + 0x000E	Trigger 4 Compare Register	T4CR	16
Base + 0x0010	Trigger 5 Compare Register	T5CR	16
Base + 0x0012	Trigger 6 Compare Register	T6CR	16
Base + 0x0014	Trigger 7 Compare Register	T7CR	16
Base + 0x0016	TGS Counter Compare Register	TGSCCR	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0018	TGS Counter Reload Register	TGSCRR	16
Base + 0x001A	Reserved	—	—
Base + 0x001C	Commands List Control Register 1	CLCR1	32
Base + 0x0020	Commands List Control Register 2	CLCR2	32
Base + 0x0024	Trigger Handler Control Register 1	THCR1	32
Base + 0x0028	Trigger Handler Control Register 2	THCR2	32
Base + 0x002C	Commands List Register 1	CLR1	16
Base + 0x002E	Commands List Register 2	CLR2	16
Base + 0x0030	Commands List Register 3	CLR3	16
Base + 0x0032	Commands List Register 4	CLR4	16
Base + 0x0034	Commands List Register 5	CLR5	16
Base + 0x0036	Commands List Register 6	CLR6	16
Base + 0x0038	Commands List Register 7	CLR7	16
Base + 0x003A	Commands List Register 8	CLR8	16
Base + 0x003C	Commands List Register 9	CLR9	16
Base + 0x003E	Commands List Register 10	CLR10	16
Base + 0x0040	Commands List Register 11	CLR11	16
Base + 0x0042	Commands List Register 12	CLR12	16
Base + 0x0044	Commands List Register 13	CLR13	16
Base + 0x0046	Commands List Register 14	CLR14	16
Base + 0x0048	Commands List Register 15	CLR15	16
Base + 0x004A	Commands List Register 16	CLR16	16
Base + 0x004C	Commands List Register 17	CLR17	16
Base + 0x004E	Commands List Register 18	CLR18	16
Base + 0x0050	Commands List Register 19	CLR19	16
Base + 0x0052	Commands List Register 20	CLR20	16
Base + 0x0054	Commands List Register 21	CLR21	16
Base + 0x0056	Commands List Register 22	CLR22	16
Base + 0x0058	Commands List Register 23	CLR23	16
Base + 0x005A	Commands List Register 24	CLR24	16
Base + (0x005C – 0x006B)	Reserved	—	—
Base + 0x006C	FIFO Control Register	FDCR	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0070	FIFO Control Register FIFO	FCR	32
Base + 0x0074	FIFO Threshold Register	FTH	32
Base + (0x0078 – 0x007B)	Reserved	—	—
Base + 0x007C	Status Register	FST	16
Base + 0x0080	FIFO Right Aligned data 0	FR0	32
Base + 0x0084	FIFO Right Aligned data 1	FR1	32
Base + 0x0088	FIFO Right Aligned data 2	FR2	32
Base + 0x008C	FIFO Right Aligned data 3	FR3	32
Base + (0x0090 – 0x009F)	Reserved	—	—
Base + 0x00A0	FIFO Left Aligned data 0	FL0	32
Base + 0x00A4	FIFO Left Aligned data 1	FL1	32
Base + 0x00A8	FIFO Left Aligned data 2	FL2	32
Base + 0x00AC	FIFO Left Aligned data 3	FL3	32
Base + (0x00B0 – 0x00BF)	Reserved	—	—
Base + 0x00C0	CTU Error Flag Register	CTUEFR	16
Base + 0x00C2	CTU Interrupt Flag Register	CTUIFR	16
Base + 0x00C4	CTU Interrupt Register	CTUIR	16
Base + 0x00C6	Control On-Time Register	COTR	16
Base + 0x00C8	CTU Control Register	CTUCR	16
Base + 0x00CA	CTU Digital Filter Register	CTUDF	16
Base + 0x00CC	CTU Digital Filter Register	CTUPCR	16
Base + (0x00CE – 0x3FFF)	Reserved	—	—
0xFFE1_8000	**eTIMER0** *Section 27.6: Memory map and registers*		
eTimer0 Channel 0			
Base + 0x0000	Compare Register 1	COMP1	16
Base + 0x0002	Compare Register 2	COMP2	16
Base + 0x0004	Capture Register 1	CAPT1	16
Base + 0x0006	Capture Register 2	CAPT2	16
Base + 0x0008	Load Register	LOAD	16
Base + 0x000A	Hold Register	HOLD	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x000C	Counter Register	CNTR	16
Base + 0x000E	Control Register	CTRL1	16
Base + 0x0010	Control Register 2	CTRL2	16
Base + 0x0012	Control Register 3	CTRL3	16
Base + 0x0014	Status Register	STS	16
Base + 0x0016	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0018	Comparator Load Register 1	CMPLD1	16
Base + 0x001A	Comparator Load Register 2	CMPLD2	16
Base + 0x001C	Compare and Capture Control Register	CCCTRL	16
Base + 0x001E	Input Filter Register	FILT	16
eTimer0 Channel 1			
Base + 0x0020	Compare Register 1	COMP1	16
Base + 0x0022	Compare Register 2	COMP2	16
Base + 0x0024	Capture Register 1	CAPT1	16
Base + 0x0026	Capture Register 2	CAPT2	16
Base + 0x0028	Load Register	LOAD	16
Base + 0x002A	Hold Register	HOLD	16
Base + 0x002C	Counter Register	CNTR	16
Base + 0x002E	Control Register	CTRL1	16
Base + 0x0030	Control Register 2	CTRL2	16
Base + 0x0032	Control Register 3	CTRL3	16
Base + 0x0034	Status Register	STS	16
Base + 0x0036	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0038	Comparator Load Register 1	CMPLD1	16
Base + 0x003A	Comparator Load Register 2	CMPLD2	16
Base + 0x003C	Compare and Capture Control Register	CCCTRL	16
Base + 0x003E	Input Filter Register	FILT	16
eTimer0 Channel 2			
Base + 0x0040	Compare Register 1	COMP1	16
Base + 0x0042	Compare Register 2	COMP2	16
Base + 0x0044	Capture Register 1	CAPT1	16
Base + 0x0046	Capture Register 2	CAPT2	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0048	Load Register	LOAD	16
Base + 0x004A	Hold Register	HOLD	16
Base + 0x004C	Counter Register	CNTR	16
Base + 0x004E	Control Register	CTRL1	16
Base + 0x0050	Control Register 2	CTRL2	16
Base + 0x0052	Control Register 3	CTRL3	16
Base + 0x0054	Status Register	STS	16
Base + 0x0056	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0058	Comparator Load Register 1	CMPLD1	16
Base + 0x005A	Comparator Load Register 2	CMPLD2	16
Base + 0x005C	Compare and Capture Control Register	CCCTRL	16
Base + 0x005E	Input Filter Register	FILT	16
eTimer0 Channel 3			
Base + 0x0060	Compare Register 1	COMP1	16
Base + 0x0062	Compare Register 2	COMP2	16
Base + 0x0064	Capture Register 1	CAPT1	16
Base + 0x0066	Capture Register 2	CAPT2	16
Base + 0x0068	Load Register	LOAD	16
Base + 0x006A	Hold Register	HOLD	16
Base + 0x006C	Counter Register	CNTR	16
Base + 0x006E	Control Register	CTRL1	16
Base + 0x0070	Control Register 2	CTRL2	16
Base + 0x0072	Control Register 3	CTRL3	16
Base + 0x0074	Status Register	STS	16
Base + 0x0076	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0078	Comparator Load Register 1	CMPLD1	16
Base + 0x007A	Comparator Load Register 2	CMPLD2	16
Base + 0x007C	Compare and Capture Control Register	CCCTRL	16
Base + 0x007E	Input Filter Register	FILT	16
eTimer0 Channel 4			
Base + 0x0080	Compare Register 1	COMP1	16
Base + 0x0082	Compare Register 2	COMP2	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0084	Capture Register 1	CAPT1	16
Base + 0x0086	Capture Register 2	CAPT2	16
Base + 0x0088	Load Register	LOAD	16
Base + 0x008A	Hold Register	HOLD	16
Base + 0x008C	Counter Register	CNTR	16
Base + 0x008E	Control Register	CTRL1	16
Base + 0x0090	Control Register 2	CTRL2	16
Base + 0x0092	Control Register 3	CTRL3	16
Base + 0x0094	Status Register	STS	16
Base + 0x0096	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0098	Comparator Load Register 1	CMPLD1	16
Base + 0x009A	Comparator Load Register 2	CMPLD2	16
Base + 0x009C	Compare and Capture Control Register	CCCTRL	16
Base + 0x009E	Input Filter Register	FILT	16
eTimer0 Channel 5			
Base + 0x00A0	Compare Register 1	COMP1	16
Base + 0x00A2	Compare Register 2	COMP2	16
Base + 0x00A4	Capture Register 1	CAPT1	16
Base + 0x00A6	Capture Register 2	CAPT2	16
Base + 0x00A8	Load Register	LOAD	16
Base + 0x008AA	Hold Register	HOLD	16
Base + 0x00AC	Counter Register	CNTR	16
Base + 0x00AE	Control Register	CTRL1	16
Base + 0x00B0	Control Register 2	CTRL2	16
Base + 0x00B2	Control Register 3	CTRL3	16
Base + 0x00B4	Status Register	STS	16
Base + 0x00B6	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x00B8	Comparator Load Register 1	CMPLD1	16
Base + 0x009BA	Comparator Load Register 2	CMPLD2	16
Base + 0x00BC	Compare and Capture Control Register	CCCTRL	16
Base + 0x00BE	Input Filter Register	FILT	16
Base + (0x00C0 – 0x00FF)	Reserved	—	—

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0100	Watchdog Time-out Low Register	WDTOL	16
Base + 0x0102	Watchdog Time-out High Register	WDTOH	16
Base + (0x0104 – 0x010B)	Reserved	—	—
Base + 0x010C	Channel Enable Register	ENBL	16
Base + 0x0110	DMA Request 0 Select Register	DREQ0	16
Base + 0x0112	DMA Request 1 Select Register	DREQ1	16
Base + (0x0114 – 0x3FFF)	Reserved	—	—
0xFFE1_C000	**eTIMER1** *Section 27.6: Memory map and registers*		
eTimer1 Channel 0			
Base + 0x0000	Compare Register 1	COMP1	16
Base + 0x0002	Compare Register 2	COMP2	16
Base + 0x0004	Capture Register 1	CAPT1	16
Base + 0x0006	Capture Register 2	CAPT2	16
Base + 0x0008	Load Register	LOAD	16
Base + 0x000A	Hold Register	HOLD	16
Base + 0x000C	Counter Register	CNTR	16
Base + 0x000E	Control Register	CTRL1	16
Base + 0x0010	Control Register 2	CTRL2	16
Base + 0x0012	Control Register 3	CTRL3	16
Base + 0x0014	Status Register	STS	16
Base + 0x0016	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0018	Comparator Load Register 1	CMPLD1	16
Base + 0x001A	Comparator Load Register 2	CMPLD2	16
Base + 0x001C	Compare and Capture Control Register	CCCTRL	16
Base + 0x001E	Input Filter Register	FILT	16
eTimer1 Channel 1			
Base + 0x0020	Compare Register 1	COMP1	16
Base + 0x0022	Compare Register 2	COMP2	16
Base + 0x0024	Capture Register 1	CAPT1	16
Base + 0x0026	Capture Register 2	CAPT2	16
Base + 0x0028	Load Register	LOAD	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x002A	Hold Register	HOLD	16
Base + 0x002C	Counter Register	CNTR	16
Base + 0x002E	Control Register	CTRL1	16
Base + 0x0030	Control Register 2	CTRL2	16
Base + 0x0032	Control Register 3	CTRL3	16
Base + 0x0034	Status Register	STS	16
Base + 0x0036	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0038	Comparator Load Register 1	CMPLD1	16
Base + 0x003A	Comparator Load Register 2	CMPLD2	16
Base + 0x003C	Compare and Capture Control Register	CCCTRL	16
Base + 0x003E	Input Filter Register	FILT	16
eTimer1 Channel 2			
Base + 0x0040	Compare Register 1	COMP1	16
Base + 0x0042	Compare Register 2	COMP2	16
Base + 0x0044	Capture Register 1	CAPT1	16
Base + 0x0046	Capture Register 2	CAPT2	16
Base + 0x0048	Load Register	LOAD	16
Base + 0x004A	Hold Register	HOLD	16
Base + 0x004C	Counter Register	CNTR	16
Base + 0x004E	Control Register	CTRL1	16
Base + 0x0050	Control Register 2	CTRL2	16
Base + 0x0052	Control Register 3	CTRL3	16
Base + 0x0054	Status Register	STS	16
Base + 0x0056	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0058	Comparator Load Register 1	CMPLD1	16
Base + 0x005A	Comparator Load Register 2	CMPLD2	16
Base + 0x005C	Compare and Capture Control Register	CCCTRL	16
Base + 0x005E	Input Filter Register	FILT	16
eTimer1 Channel 3			
Base + 0x0060	Compare Register 1	COMP1	16
Base + 0x0062	Compare Register 2	COMP2	16
Base + 0x0064	Capture Register 1	CAPT1	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0066	Capture Register 2	CAPT2	16
Base + 0x0068	Load Register	LOAD	16
Base + 0x006A	Hold Register	HOLD	16
Base + 0x006C	Counter Register	CNTR	16
Base + 0x006E	Control Register	CTRL1	16
Base + 0x0070	Control Register 2	CTRL2	16
Base + 0x0072	Control Register 3	CTRL3	16
Base + 0x0074	Status Register	STS	16
Base + 0x0076	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0078	Comparator Load Register 1	CMPLD1	16
Base + 0x007A	Comparator Load Register 2	CMPLD2	16
Base + 0x007C	Compare and Capture Control Register	CCCTRL	16
Base + 0x007E	Input Filter Register	FILT	16
eTimer1 Channel 4			
Base + 0x0080	Compare Register 1	COMP1	16
Base + 0x0082	Compare Register 2	COMP2	16
Base + 0x0084	Capture Register 1	CAPT1	16
Base + 0x0086	Capture Register 2	CAPT2	16
Base + 0x0088	Load Register	LOAD	16
Base + 0x008A	Hold Register	HOLD	16
Base + 0x008C	Counter Register	CNTR	16
Base + 0x008E	Control Register	CTRL1	16
Base + 0x0090	Control Register 2	CTRL2	16
Base + 0x0092	Control Register 3	CTRL3	16
Base + 0x0094	Status Register	STS	16
Base + 0x0096	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x0098	Comparator Load Register 1	CMPLD1	16
Base + 0x009A	Comparator Load Register 2	CMPLD2	16
Base + 0x009C	Compare and Capture Control Register	CCCTRL	16
Base + 0x009E	Input Filter Register	FILT	16
eTimer1 Channel 5			
Base + 0x00A0	Compare Register 1	COMP1	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00A2	Compare Register 2	COMP2	16
Base + 0x00A4	Capture Register 1	CAPT1	16
Base + 0x00A6	Capture Register 2	CAPT2	16
Base + 0x00A8	Load Register	LOAD	16
Base + 0x008AA	Hold Register	HOLD	16
Base + 0x00AC	Counter Register	CNTR	16
Base + 0x00AE	Control Register	CTRL1	16
Base + 0x00B0	Control Register 2	CTRL2	16
Base + 0x00B2	Control Register 3	CTRL3	16
Base + 0x00B4	Status Register	STS	16
Base + 0x00B6	Interrupt and DMA Enable Register	INTDMA	16
Base + 0x00B8	Comparator Load Register 1	CMPLD1	16
Base + 0x009BA	Comparator Load Register 2	CMPLD2	16
Base + 0x00BC	Compare and Capture Control Register	CCCTRL	16
Base + 0x00BE	Input Filter Register	FILT	16
Base + (0x00C0 – 0x00FF)	Reserved	—	—
Base + 0x0100	Watchdog Time-out Low Register	WDTOL	16
Base + 0x0102	Watchdog Time-out High Register	WDTOH	16
Base + (0x0104 – 0x010B)	Reserved	—	—
Base + 0x010C	Channel Enable Register	ENBL	16
Base + 0x0110	DMA Request 0 Select Register	DREQ0	16
Base + 0x0112	DMA Request 1 Select Register	DREQ1	16
Base + (0x0114 – 0x3FFF)	Reserved	—	—
0xFFE2_4000	**Motor Control Pulse Width Modulator Module (FlexPWM_0)** *Section 26.6.1: FlexPWM module memory map*		
FlexPWM_0 Submodule 0			
Base + 0x0000	Counter Register (Submodule 0)	CNT	16
Base + 0x0002	Initial Count Register (Submodule 0)	INIT	16
Base + 0x0004	Control 2 Register (Submodule 0)	CTRL2	16
Base + 0x0006	Control Register (Submodule 0)	CTRL1	16
Base + 0x0008	Value Register 0 (Submodule 0)	VAL0	16
Base + 0x000A	Value Register 1 (Submodule 0)	VAL1	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x000C	Value Register 2 (Submodule 0)	VAL2	16
Base + 0x000E	Value Register 3 (Submodule 0)	VAL3	16
Base + 0x0010	Value Register 4 (Submodule 0)	VAL4	16
Base + 0x0012	Value Register 5 (Submodule 0)	VAL5	16
Base + (0x0014 – 0x0017)	Reserved	—	—
Base + 0x0018	Output Control Register (Submodule 0)	OCTRL	16
Base + 0x001A	Status Register (Submodule 0)	STS	16
Base + 0x001C	Interrupt Enable Register (Submodule 0)	INTEN	16
Base + 0x001E	DMA Enable Register (Submodule 0)	DMAEN	16
Base + 0x0020	Output Trigger Control Register SUB0	TCTRL	16
Base + 0x0022	Fault Disable Mapping Register (Submodule 0)	DISMAP	16
Base + 0x0024	Deadtime Count Register 0 (Submodule 0)	DTCNT0	16
Base + 0x0026	Deadtime Count Register 1 (Submodule 0)	DTCNT1	16
Base + (0x0028 – 0x002F)	Reserved	—	—
Base + 0x0030	Capture Control Register X (Submodule 0)	CAPTCTRLX	16
Base + 0x0032	Capture Compare Register X (Submodule 0)	CAPTCOMPX	16
Base + 0x0034	Capture Value 0 Register (Submodule 0)	CVAL0	16
Base + 0x0036	Capture Value 0 Cycle Register (Submodule 0)	CVAL0C	16
Base + 0x0038	Capture Value 1 Register (Submodule 0)	CVAL1	16
Base + 0x003A	Capture Value 1 Cycle Register (Submodule 0)	VAL1C	16
Base + (0x003C – 0x004F)	Reserved	—	—
FlexPWM_0 Submodule 1			
Base + 0x0050	Counter Register (Submodule 1)	CNT	16
Base + 0x0052	Initial Count Register (Submodule 1)	INIT	16
Base + 0x0054	Control 2 Register (Submodule 1)	CTRL2	16
Base + 0x0056	Control Register (Submodule 1)	CTRL1	16
Base + 0x0058	Value Register 0 (Submodule 1)	VAL0	16
Base + 0x005A	Value Register 1 (Submodule 1)	VAL1	16
Base + 0x005C	Value Register 2 (Submodule 1)	VAL2	16
Base + 0x005E	Value Register 3 (Submodule 1)	VAL3	16
Base + 0x0060	Value Register 4 (Submodule 1)	VAL4	16
Base + 0x0062	Value Register 5 (Submodule 1)	VAL5	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0064 – 0x0067)	Reserved	—	—
Base + 0x0068	Output Control Register (Submodule 1)	OCTRL	16
Base + 0x006A	Status Register (Submodule 1)	STS	16
Base + 0x006C	Interrupt Enable Register (Submodule 1)	INTEN	16
Base + 0x006E	DMA Enable Register (Submodule 1)	DMAEN	16
Base + 0x0070	Output Trigger Control Register (Submodule 1)	TCTRL	16
Base + 0x0072	Fault Disable Mapping Register (Submodule 1)	DISMAP	16
Base + 0x0074	Deadtime Count Register 0 (Submodule 1)	DTCNT0	16
Base + 0x0076	Deadtime Count Register 1 (Submodule 1)	DTCNT1	16
Base + (0x0078 – 0x007F)	Reserved	—	—
Base + 0x0080	Capture Control Register X (Submodule 1)	CAPTCTRLX	16
Base + 0x0082	Capture Compare Register X (Submodule 1)	CAPTCOMPX	16
Base + 0x0084	Capture Value 0 Register (Submodule 1)	CVAL0	16
Base + 0x0086	Capture Value 0 Cycle Register (Submodule 1)	CVAL0C	16
Base + 0x0088	Capture Value 1 Register (Submodule 1)	CVAL1	16
Base + 0x008A	Capture Value 1 Cycle Register (Submodule 1)	VAL1C	16
Base + (0x008C – 0x009F)	Reserved	—	—
FlexPWM_0 Submodule 2			
Base + 0x00A0	Counter Register (Submodule 2)	CNT	16
Base + 0x000A2	Initial Count Register (Submodule 2)	INIT	16
Base + 0x00A4	Control 2 Register (Submodule 2)	CTRL2	16
Base + 0x00A6	Control Register (Submodule 2)	CTRL1	16
Base + 0x00A8	Value Register 0 (Submodule 2)	VAL0	16
Base + 0x00AA	Value Register 1 (Submodule 2)	VAL1	16
Base + 0x00AC	Value Register 2 (Submodule 2)	VAL2	16
Base + 0x00AE	Value Register 3 (Submodule 2)	VAL3	16
Base + 0x00B0	Value Register 4 (Submodule 2)	VAL4	16
Base + 0x00B2	Value Register 5 (Submodule 2)	VAL5	16
Base + (0x00B4 – 0x00B7)	Reserved	—	—
Base + 0x00B8	Output Control Register (Submodule 2)	OCTRL	16
Base + 0x00BA	Status Register (Submodule 2)	STS	16
Base + 0x00BC	Interrupt Enable Register (Submodule 2)	INTEN	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00BE	DMA Enable Register (Submodule 2)	DMAEN	16
Base + 0x00C0	Output Trigger Control Register (Submodule 2)	TCTRL	16
Base + 0x00C2	Fault Disable Mapping Register (Submodule 2)	DISMAP	16
Base + 0x00C4	Deadtime Count Register 0 (Submodule 2)	DTCNT0	16
Base + 0x00C6	Deadtime Count Register 1 (Submodule 2)	DTCNT1	16
Base + (0x00C8 – 0x00CF)	Reserved	—	—
Base + 0x00D0	Capture Control Register X (Submodule 2)	CAPTCTRLX	16
Base + 0x00D2	Capture Compare Register X (Submodule 2)	CAPTCOMPX	16
Base + 0x00D4	Capture Value 0 Register (Submodule 2)	CVAL0	16
Base + 0x00D6	Capture Value 0 Cycle Register (Submodule 2)	CVAL0C	16
Base + 0x00D8	Capture Value 1 Register (Submodule 2)	CVAL1	16
Base + 0x00DA	Capture Value 1 Cycle Register (Submodule 2)	VAL1C	16
Base + (0x00DC – 0x00EF)	Reserved	—	—
FlexPWM_0 Submodule 3			
Base + 0x00F0	Counter Register (Submodule 3)	CNT	16
Base + 0x000F2	Initial Count Register (Submodule 3)	INIT	16
Base + 0x00F4	Control 2 Register (Submodule 3)	CTRL2	16
Base + 0x00F6	Control Register (Submodule 3)	CTRL1	16
Base + 0x00F8	Value Register 0 (Submodule 3)	VAL0	16
Base + 0x00FA	Value Register 1 (Submodule 3)	VAL1	16
Base + 0x00FC	Value Register 2 (Submodule 3)	VAL2	16
Base + 0x00FE	Value Register 3 (Submodule 3)	VAL3	16
Base + 0x0100	Value Register 4 (Submodule 3)	VAL4	16
Base + 0x0102	Value Register 5 (Submodule 3)	VAL5	16
Base + (0x0104 – 0x00107)	Reserved	—	—
Base + 0x0108	Output Control Register (Submodule 3)	OCTRL	16
Base + 0x010A	Status Register (Submodule 3)	STS	16
Base + 0x010C	Interrupt Enable Register (Submodule 3)	INTEN	16
Base + 0x010E	DMA Enable Register (Submodule 3)	DMAEN	16
Base + 0x0110	Output Trigger Control Register (Submodule 3)	TCTRL	16
Base + 0x0112	Fault Disable Mapping Register (Submodule 3)	DISMAP	16
Base + 0x0114	Deadtime Count Register 0 (Submodule 3)	DTCNT0	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0116	Deadtime Count Register 1 (Submodule 3)	DTCNT1	16
Base + (0x0118 – 0x0011F)	Reserved	—	—
Base + 0x0120	Capture Control Register X (Submodule 3)	CAPTCTRLX	16
Base + 0x0122	Capture Compare Register X (Submodule 3)	CAPTCOMPX	16
Base + 0x0124	Capture Value 0 Register (Submodule 3)	CVAL0	16
Base + 0x0126	Capture Value 0 Cycle Register (Submodule 3)	CVAL0C	16
Base + 0x0128	Capture Value 1 Register (Submodule 3)	CVAL1	16
Base + 0x012A	Capture Value 1 Cycle Register (Submodule 3)	VAL1C	16
Base + (0x012C – 0x013F)	Reserved	—	—
Base + 0x0140	Output Enable Register	OUTEN	16
Base + 0x0142	Output Mask Register	MASK	16
Base + 0x0144	Software Controlled Output Register	SWCOUT	16
Base + 0x0146	Deadtime Source Select Register	DTSRCSEL	16
Base + 0x0148	Master Control Register	MCTRL	16
Base + (0x014A – 0x014B)	Reserved	—	—
Base + 0x014C	Fault Control Register	FCTRL	16
Base + 0x014E	Fault Status Register	FSTS	16
Base + 0x0150	Fault Filter Register	FFILT	16
Base + (0x0152 – 0x3FFF)	Reserved	—	—
0xFFE4_0000	**LINFlex 0** *Section 22.7: Memory map and registers description*		
Base + 0x0000	LIN Control Register	LINCR1	32
Base + 0x0004	LIN Interrupt Enable Register	LINIER	32
Base + 0x0008	LIN Status Register	LINSR	32
Base + 0x000C	LIN Error Status Register	LINESR	32
Base + 0x0010	UART Mode Control Register	UARTCR	32
Base + 0x0014	UART Mode Status Register	UARTSR	32
Base + 0x0018	LIN Time-Out Control Status Register	LINTCSR	32
Base + 0x001C	LIN Output Compare Register	LINOCR	32
Base + 0x0020	LIN Time-Out Control Register	LINTOCR	32
Base + 0x0024	LIN Fractional Baud Rate Register	LINFBRR	32
Base + 0x0028	LIN Integer Baud Rate Register	LINIBRR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x002C	LIN Checksum Field Register	LINCFR	32
Base + 0x0030	LIN Control Register 2	LINCR2	32
Base + 0x0034	Buffer Identifier Register	BIDR	32
Base + 0x0038	Buffer Data Register Least Significant	BDRL	32
Base + 0x003C	Buffer Data Register Most Significant	BDRM	32
Base + 0x0040	Identifier Filter Enable Register	IFER	32
Base + 0x0044	Identifier Filter Match Index	IFMI	32
Base + 0x0048	Identifier Filter Mode Register	IFMR	32
Base + 0x004C	Identifier Filter Control Register 0	IFCR0	32
Base + 0x0050	Identifier Filter Control Register 1	IFCR1	32
Base + 0x0054	Identifier Filter Control Register 2	IFCR2	32
Base + 0x0058	Identifier Filter Control Register 3	IFCR3	32
Base + 0x005C	Identifier Filter Control Register 4	IFCR4	32
Base + 0x0060	Identifier Filter Control Register 5	IFCR5	32
Base + 0x0064	Identifier Filter Control Register 6	IFCR6	32
Base + 0x0068	Identifier Filter Control Register 7	IFCR7	32
Base + 0x006C	Identifier Filter Control Register 8	IFCR8	32
Base + 0x0070	Identifier Filter Control Register 9	IFCR9	32
Base + 0x0074	Identifier Filter Control Register 10	IFCR10	32
Base + 0x0078	Identifier Filter Control Register 11	IFCR11	32
Base + 0x007C	Identifier Filter Control Register 12	IFCR12	32
Base + 0x0080	Identifier Filter Control Register 13	IFCR13	32
Base + 0x0084	Identifier Filter Control Register 14	IFCR14	32
Base + 0x0088	Identifier Filter Control Register 15	IFCR15	32
Base + (0x008C – 0x3FFF)	Reserved	—	—
0xFFE4_4000	**LINFlex 1** *Section 22.7: Memory map and registers description*		
Base + 0x0000	LIN Control Register	LINCR1	32
Base + 0x0004	LIN Interrupt Enable Register	LINIER	32
Base + 0x0008	LIN Status Register	LINSR	32
Base + 0x000C	LIN Error Status Register	LINESR	32
Base + 0x0010	UART Mode Control Register	UARTCR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0014	UART Mode Status Register	UARTSR	32
Base + 0x0018	LIN Time-Out Control Status Register	LINTCSR	32
Base + 0x001C	LIN Output Compare Register	LINOCR	32
Base + 0x0020	LIN Time-Out Control Register	LINTOCR	32
Base + 0x0024	LIN Fractional Baud Rate Register	LINFBRR	32
Base + 0x0028	LIN Integer Baud Rate Register	LINIBRR	32
Base + 0x002C	LIN Checksum Field Register	LINCFR	32
Base + 0x0030	LIN Control Register 2	LINCR2	32
Base + 0x0034	Buffer Identifier Register	BIDR	32
Base + 0x0038	Buffer Data Register Least Significant	BDRL	32
Base + 0x003C	Buffer Data Register Most Significant	BDRM	32
Base + 0x0040	Identifier Filter Enable Register	IFER	32
Base + 0x0044	Identifier Filter Match Index	IFMI	32
Base + 0x0048	Identifier Filter Mode Register	IFMR	32
Base + 0x004C	Identifier Filter Control Register 0	IFCR0	32
Base + 0x0050	Identifier Filter Control Register 1	IFCR1	32
Base + 0x0054	Identifier Filter Control Register 2	IFCR2	32
Base + 0x0058	Identifier Filter Control Register 3	IFCR3	32
Base + 0x005C	Identifier Filter Control Register 4	IFCR4	32
Base + 0x0060	Identifier Filter Control Register 5	IFCR5	32
Base + 0x0064	Identifier Filter Control Register 6	IFCR6	32
Base + 0x0068	Identifier Filter Control Register 7	IFCR7	32
Base + 0x006C	Identifier Filter Control Register 8	IFCR8	32
Base + 0x0070	Identifier Filter Control Register 9	IFCR9	32
Base + 0x0074	Identifier Filter Control Register 10	IFCR10	32
Base + 0x0078	Identifier Filter Control Register 11	IFCR11	32
Base + 0x007C	Identifier Filter Control Register 12	IFCR12	32
Base + 0x0080	Identifier Filter Control Register 13	IFCR13	32
Base + 0x0084	Identifier Filter Control Register 14	IFCR14	32
Base + 0x0088	Identifier Filter Control Register 15	IFCR15	32
Base + (0x008C – 0x3FFF)	Reserved	—	—

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
0xFFE6_8000	**Cyclic Redundancy Check (CRC)** *Section 33.5: Memory map and registers description*		
Base + 0x0000	CRC Configuration Register, Context 1	*CRC_CFG*	32
Base + 0x0004	CRC Input Register, Context 1	*CRC_INP*	32
Base + 0x0008	CRC Current Status Register, Context 1	*CRC_CSTAT*	32
Base + 0x000C	CRC Output Register, Context 1	CRC_OUTP	32
Base + 0x0010	CRC Configuration Register, Context 2	*CRC_CFG*	32
Base + 0x0014	CRC Input Register, Context 2	*CRC_INP*	32
Base + 0x0018	CRC Current Status Register, Context 2	*CRC_CSTAT*	32
Base + 0x001C	CRC Output Register, Context 2	CRC_OUTP	32
Base + (0x002A – 0x3FFF)	Reserved	—	—
0xFFE6_C000	**Fault Collection Unit (FCU)** *Section 29.2: Memory map and register definition*		
Base + 0x0000	Module Configuration Register	FCU_MCR	32
Base + 0x0004	Fault Flag Register	FCU_FFR	32
Base + 0x0008	Frozen Fault Flag Register	FCU_FFFR	32
Base + 0x000C	Fake Fault Generation Register	FCU_FFGR	32
Base + 0x0010	Fault Enable Register	FCU_FER	32
Base + 0x0014	Fault Collection Unit Key Register	FCU_KR	32
Base + 0x0018	Fault Collection Unit Timeout Register	FCU_TR	32
Base + 0x001C	Fault Collection Unit Timeout Enable Register	FCU_TER	32
Base + 0x0020	Module Status Register	FCU_MSR	32
Base + 0x0024	MC State Register	FCU_MCSR	32
Base + 0x0028	Frozen MC State Register	FCU_FMCSR	32
Base + (0x002C – 0x3FFF)	Reserved	—	—
0xFFF3_8000	**Software Watchdog (SWT)** *Section 28.3.5: SWT memory map and registers description*		
Base + 0x0000	Control Register	SWT_CR	32
Base + 0x0004	SWT Interrupt Register	SWT_IR	32
Base + 0x0008	SWT Time-Out Register	SWT_TO	32
Base + 0x000C	SWT Window Register	SWT_WN	32
Base + 0x0010	SWT Service Register	SWT_SR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0014	SWT Counter Output Register	SWT_CO	32
Base + 0x0018	SWT Service Key register	SWT_SK	32
Base + (0x0018 – 0x3FFF)	Reserved	—	—
0xFFF3_C000	**System Timer Module (STM)** *Section 32.5: Memory map and registers description*		
Base + 0x0000	Control Register	STM_CR	32
Base + 0x0004	STM Count Register	STM_CNT	32
Base + (0x0008 – 0x000F)	Reserved	—	—
Base + 0x00010	STM Channel 0 Control Register	STM_CCR0	32
Base + 0x00014	STM Channel 0 Interrupt Register	STM_CIR0	32
Base + 0x00018	STM Channel 0 Compare Register	STM_CMP0	32
Base + (0x001C – 0x001F)	Reserved	—	—
Base + 0x00020	STM Channel 1 Control Register	STM_CCR1	32
Base + 0x00024	STM Channel 1 Interrupt Register	STM_CIR1	32
Base + 0x00028	STM Channel 1 Compare Register	STM_CMP1	32
Base + (0x002C – 0x002F)	Reserved	—	—
Base + 0x00030	STM Channel 2 Control Register	STM_CCR2	32
Base + 0x00034	STM Channel 2 Interrupt Register	STM_CIR2	32
Base + 0x00038	STM Channel 2 Compare Register	STM_CMP2	32
Base + (0x003C – 0x003F)	Reserved	—	—
Base + 0x00040	STM Channel 3 Control Register	STM_CCR3	32
Base + 0x00044	STM Channel 3 Interrupt Register	STM_CIR3	32
Base + 0x00048	STM Channel 3 Compare Register	STM_CMP3	32
Base + (0x003C – 0x3FFF)	Reserved	—	—
0xFFF4_0000	**Error Correction Status Module (ECSM)** *Section 15.4: Memory map and registers description*		
Base + 0x0000	Processor Core Type	PCT	16
Base + 0x0002	SOC-Defined Platform Revision	REV	16
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	IPS Module Configuration	IMC	32
Base + (0x000C – 0x000E)	Reserved	—	—
Base + 0x000F	Miscellaneous Reset Status Register	MRSR	8

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x00010 – 0x001E)	Reserved	—	—
Base + 0x001F	Miscellaneous Interrupt Register	MIR	8
Base + (0x0020 – 0x0023)	Reserved	—	—
Base + 0x0024	Miscellaneous User Defined Control Register	MUDCR	32
Base + (0x0028 – 0x0042)	Reserved	—	—
Base + 0x0043	ECC Configuration Register	ECR	8
Base + (0x0044 – 0x0046)	Reserved	—	—
Base + 0x0047	ECC Status Register	ESR	8
Base + (0x0048 – 0x0049)	Reserved	—	—
Base + 0x004A	ECC Error Generation Register	EEGR	16
Base + (0x004C – 0x004F)	Reserved	—	—
Base + 0x0050	ECC Error Address Register	FEAR	32
Base + (0x0054 – 0x0055)	Reserved	—	—
Base + 0x0056	Flash ECC Master Number Register	FEMR	8
Base + 0x0057	Flash ECC Attributes Register	FEAT	8
Base + (0x0058 – 0x005B)	Reserved	—	—
Base + 0x005C	Flash ECC Data Register	FEDR	32
Base + 0x0060	RAM ECC Address Register	REAR	32
Base + 0x0064	Reserved	—	—
Base + 0x0065	RAM ECC Syndrome Register	RESR	8
Base + 0x0066	RAM ECC Master Number Register	REMR	8
Base + 0x0067	RAM ECC Attributes Register	REAT	8
Base + (0x0068 – 0x006B)	Reserved	—	—
Base + 0x006C	RAM ECC Data Register	REDR	32
Base + (0x0070 – 0x3FFF)	Reserved	—	—
0xFFF4_4000	**DMA2** *Section 18.5: Memory map and register definition*		
Base + 0x0000	eDMA Control Register	EDMA_CR	32
Base + 0x0004	eDMA Error Status Register	EDMA_ESR	32
Base + 0x0008	Reserved	—	—
Base + 0x000C	eDMA Enable Request Register	EDMA_ERQL	32
Base + 0x0010	Reserved	—	—

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0014	eDMA Enable Error Interrupt Register	EDMA_EEIRL	32
Base + 0x0018	eDMA Set Enable Request Register	EDMA_SERQR	8
Base + 0x0019	eDMA Clear Enable Request Register	EDMA_CERQR	8
Base + 0x001A	eDMA Set Enable Error Interrupt Register	EDMA_SEEI	8
Base + 0x001B	eDMA Clear Enable Error Interrupt Register	EDMA_CEEI	8
Base + 0x001C	eDMA Clear Interrupt Request Register	EDMA_CIRQR	8
Base + 0x001D	eDMA Clear Error Register	EDMA_CER	8
Base + 0x001E	eDMA Set START Bit Register	EDMA_SSBR	8
Base + 0x001F	eDMA Clear DONE Status Register	EDMA_CDSBR	8
Base + 0x0020	Reserved	—	—
Base + 0x0024	eDMA Interrupt Request Register	EDMA_IRQRL	32
Base + 0x0028	Reserved	—	—
Base + 0x002C	eDMA Error Register	EDMA_ERL	32
Base + 0x0030	Reserved	—	—
Base + 0x0034	eDMA Hardware Request Status Register	EDMA_HRSL	32
Base + (0x0038 – 0x00FF)	Reserved	—	—
Base + 0x0100	eDMA Channel 0 Priority Register	EDMA_CPR0	8
Base + 0x0101	eDMA Channel 1 Priority Register	EDMA_CPR1	8
Base + 0x0102	eDMA Channel 2 Priority Register	EDMA_CPR2	8
Base + 0x0103	eDMA Channel 3 Priority Register	EDMA_CPR3	8
Base + 0x0104	eDMA Channel 4 Priority Register	EDMA_CPR4	8
Base + 0x0105	eDMA Channel 5 Priority Register	EDMA_CPR5	8
Base + 0x0106	eDMA Channel 6 Priority Register	EDMA_CPR6	8
Base + 0x0107	eDMA Channel 7 Priority Register	EDMA_CPR7	8
Base + 0x0108	eDMA Channel 8 Priority Register	EDMA_CPR8	8
Base + 0x0109	eDMA Channel 9 Priority Register	EDMA_CPR9	8
Base + 0x010A	eDMA Channel 10 Priority Register	EDMA_CPR10	8
Base + 0x010B	eDMA Channel 11 Priority Register	EDMA_CPR11	8
Base + 0x010C	eDMA Channel 12 Priority Register	EDMA_CPR12	8
Base + 0x010D	eDMA Channel 13 Priority Register	EDMA_CPR13	8
Base + 0x010E	eDMA Channel 14 Priority Register	EDMA_CPR14	8
Base + 0x010F	eDMA Channel 15 Priority Register	EDMA_CPR15	8

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + (0x0110 – 0x0FFF)	Reserved	—	—
Base + 0x1000	Transfer Control Descriptor 0	TCD00	256
Base + 0x1020	Transfer Control Descriptor 1	TCD01	256
Base + 0x1040	Transfer Control Descriptor 2	TCD02	256
Base + 0x1060	Transfer Control Descriptor 3	TCD03	256
Base + 0x1080	Transfer Control Descriptor 4	TCD04	256
Base + 0x10A0	Transfer Control Descriptor 5	TCD05	256
Base + 0x10C0	Transfer Control Descriptor 6	TCD06	256
Base + 0x10E0	Transfer Control Descriptor 7	TCD07	256
Base + 0x1100	Transfer Control Descriptor 8	TCD08	256
Base + 0x1120	Transfer Control Descriptor 9	TCD09	256
Base + 0x1140	Transfer Control Descriptor 10	TCD10	256
Base + 0x1160	Transfer Control Descriptor 11	TCD11	256
Base + 0x1180	Transfer Control Descriptor 12	TCD12	256
Base + 0x11A0	Transfer Control Descriptor 13	TCD13	256
Base + 0x11C0	Transfer Control Descriptor 14	TCD14	256
Base + 0x11E0	Transfer Control Descriptor 15	TCD15	256
Base + (0x1200 – 0x3FFF)	Reserved	—	—
0xFFF4_8000	**Interrupt Controller (INTC)** *Section 9.5: Memory map and registers description*		
Base + 0x0000	Module Configuration Register	INTC_MCR	32
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	Current Priority Register	INTC_CPR	32
Base + (0x000C – 0x000F)	Reserved	—	—
Base + 0x0010	Interrupt Acknowledge Register	INTC_IACKR	32
Base + (0x0014 – 0x0017)	Reserved	—	—
Base + 0x0018	End of Interrupt Register	INTC_EOIR	32
Base + (0x001C – 0x001F)	Reserved	—	—
Base + 0x0020	Software Set/Clear Interrupt Register	INTC_SSCIR0_3	32
Base + 0x0024	Software Set/Clear Interrupt Register	INTC_SSCIR4_7	32
Base + (0x0028 – 0x003F)	Reserved	—	—
Base + 0x0040	Priority Select Register 0_3	INTC_PSR0_3	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0044	Priority Select Register 4_7	INTC_PSR4_7	32
Base + 0x0048	Priority Select Register 8_11	INTC_PSR8_11	32
Base + 0x004C	Priority Select Register 12_15	INTC_PSR12_15	32
Base + 0x0050	Priority Select Register 16_19	INTC_PSR16_19	32
Base + 0x0054	Priority Select Register 20_23	INTC_PSR20_23	32
Base + 0x0058	Priority Select Register 24_27	INTC_PSR24_27	32
Base + 0x005C	Priority Select Register 28_31	INTC_PSR28_31	32
Base + 0x0060	Priority Select Register 32_35	INTC_PSR32_35	32
Base + 0x0064	Priority Select Register 36_39	INTC_PSR36_39	32
Base + 0x0068	Priority Select Register 40_43	INTC_PSR40_43	32
Base + 0x006C	Priority Select Register 44_47	INTC_PSR44_47	32
Base + 0x0070	Priority Select Register 48_51	INTC_PSR48_51	32
Base + 0x0074	Priority Select Register 52_55	INTC_PSR52_55	32
Base + 0x0078	Priority Select Register 56_59	INTC_PSR56_59	32
Base + 0x007C	Priority Select Register 60_63	INTC_PSR60_63	32
Base + 0x0080	Priority Select Register 64_67	INTC_PSR64_67	32
Base + 0x0084	Priority Select Register 68_71	INTC_PSR68_71	32
Base + 0x0088	Priority Select Register 72_75	INTC_PSR72_75	32
Base + 0x008C	Priority Select Register 76_79	INTC_PSR76_79	32
Base + 0x0090	Priority Select Register 80_83	INTC_PSR80_83	32
Base + 0x0094	Priority Select Register 84_87	INTC_PSR84_87	32
Base + 0x0098	Priority Select Register 88_91	INTC_PSR88_91	32
Base + 0x009C	Priority Select Register 92_95	INTC_PSR92_95	32
Base + 0x00A0	Priority Select Register 96_99	INTC_PSR96_99	32
Base + 0x00A4	Priority Select Register 100_103	INTC_PSR100_103	32
Base + 0x00A8	Priority Select Register 104_107	INTC_PSR104_107	32
Base + 0x00AC	Priority Select Register 108_111	INTC_PSR108_111	32
Base + 0x00B0	Priority Select Register 112_115	INTC_PSR112_115	32
Base + 0x00B4	Priority Select Register 116_119	INTC_PSR116_119	32
Base + 0x00B8	Priority Select Register 120_123	INTC_PSR120_123	32
Base + 0x00BC	Priority Select Register 124_127	INTC_PSR124_127	32
Base + 0x00C0	Priority Select Register 128_131	INTC_PSR128_131	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00C4	Priority Select Register 132_135	INTC_PSR132_135	32
Base + 0x00C8	Priority Select Register 136_139	INTC_PSR136_139	32
Base + 0x00CC	Priority Select Register 140_143	INTC_PSR140_143	32
Base + 0x00D0	Priority Select Register 144_147	INTC_PSR144_147	32
Base + 0x00D4	Priority Select Register 148_151	INTC_PSR148_151	32
Base + 0x00D8	Priority Select Register 152_155	INTC_PSR152_155	32
Base + 0x00DC	Priority Select Register 156_159	INTC_PSR156_159	32
Base + 0x00E0	Priority Select Register 160_163	INTC_PSR160_163	32
Base + 0x00E4	Priority Select Register 164_167	INTC_PSR164_167	32
Base + 0x00E8	Priority Select Register 168_171	INTC_PSR168_171	32
Base + 0x00EC	Priority Select Register 172_175	INTC_PSR172_175	32
Base + 0x00F0	Priority Select Register 176_179	INTC_PSR176_179	32
Base + 0x00F4	Priority Select Register 180_183	INTC_PSR180_183	32
Base + 0x00F8	Priority Select Register 184_187	INTC_PSR184_187	32
Base + 0x00FC	Priority Select Register 188_191	INTC_PSR188_191	32
Base + 0x0100	Priority Select Register 192_195	INTC_PSR192_195	32
Base + 0x0104	Priority Select Register 196_199	INTC_PSR196_199	32
Base + 0x0108	Priority Select Register 200_203	INTC_PSR200_203	32
Base + 0x010C	Priority Select Register 204_207	INTC_PSR204_207	32
Base + 0x0110	Priority Select Register 208_211	INTC_PSR208_211	32
Base + 0x0114	Priority Select Register 212_215	INTC_PSR212_215	32
Base + 0x0118	Priority Select Register 216_219	INTC_PSR216_219	32
Base + 0x011C	Priority Select Register 220_221	INTC_PSR220_223	32
Base + (0x0120 – 0x3FFF)	Reserved	—	—
0xFFF9_0000	**DSPI 0** *Section 21.7: Memory map and registers description*		
Base + 0x0000	Module Configuration Register	DSPI_MCR	32
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	Transfer Count Register	DSPI_TCR	32
Base + 0x000C	Clock and Transfer Attribute Register 0	DSPI_CTAR0	32
Base + 0x0010	Clock and Transfer Attribute Register 1	DSPI_CTAR1	32
Base + 0x0014	Clock and Transfer Attribute Register 2	DSPI_CTAR2	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0018	Clock and Transfer Attribute Register 3	DSPI_CTAR3	32
Base + 0x001C	Clock and Transfer Attribute Register 4	DSPI_CTAR4	32
Base + 0x002C	Clock and Transfer Attribute Register 5	DSPI_CTAR5	32
Base + 0x0024	Clock and Transfer Attribute Register 6	DSPI_CTAR6	32
Base + 0x0028	Clock and Transfer Attribute Register 7	DSPI_CTAR7	32
Base + 0x002C	Status Register	DSPI_SR	32
Base + 0x0030	DMA/Interrupt Request Register	DSPI_RSER	32
Base + 0x0034	PUSH TX FIFO Register	DSPI_PUSHR	32
Base + 0x0038	POP RX FIFO Register	DSPI_POPR	32
Base + 0x003C	DSPI Transmit FIFO Register 0	DSPI_TXFR0	32
Base + 0x0040	DSPI Transmit FIFO Register 1	DSPI_TXFR1	32
Base + 0x0044	DSPI Transmit FIFO Register 2	DSPI_TXFR2	32
Base + 0x0048	DSPI Transmit FIFO Register 3	DSPI_TXFR3	32
Base + 0x004C	DSPI Transmit FIFO Register 4	DSPI_TXFR4	32
Base + (0x0050 – 0x0078)	Reserved	—	—
Base + 0x007C	Receive FIFO Register 0	DSPI_RXFR0	32
Base + 0x0080	Receive FIFO Register 1	DSPI_RXFR1	32
Base + 0x0084	Receive FIFO Register 2	DSPI_RXFR2	32
Base + 0x0088	Receive FIFO Register 3	DSPI_RXFR3	32
Base + 0x008C	Receive FIFO Register 4	DSPI_RXFR4	32
Base + (0x0090 – 0x3FFF)	Reserved	—	—
0xFFF9_4000	**DSPI 1** *Section 21.7: Memory map and registers description*		
Base + 0x0000	Module Configuration Register	DSPI_MCR	32
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	Transfer Count Register	DSPI_TCR	32
Base + 0x000C	Clock and Transfer Attribute Register 0	DSPI_CTAR0	32
Base + 0x0010	Clock and Transfer Attribute Register 1	DSPI_CTAR1	32
Base + 0x0014	Clock and Transfer Attribute Register 2	DSPI_CTAR2	32
Base + 0x0018	Clock and Transfer Attribute Register 3	DSPI_CTAR3	32
Base + 0x001C	Clock and Transfer Attribute Register 4	DSPI_CTAR4	32
Base + 0x002C	Clock and Transfer Attribute Register 5	DSPI_CTAR5	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0024	Clock and Transfer Attribute Register 6	DSPI_CTAR6	32
Base + 0x0028	Clock and Transfer Attribute Register 7	DSPI_CTAR7	32
Base + 0x002C	Status Register	DSPI_SR	32
Base + 0x0030	DMA/Interrupt Request Register	DSPI_RSER	32
Base + 0x0034	PUSH TX FIFO Register	DSPI_PUSHR	32
Base + 0x0038	POP RX FIFO Register	DSPI_POPR	32
Base + 0x003C	DSPI Transmit FIFO Register 0	DSPI_TXFR0	32
Base + 0x0040	DSPI Transmit FIFO Register 1	DSPI_TXFR1	32
Base + 0x0044	DSPI Transmit FIFO Register 2	DSPI_TXFR2	32
Base + 0x0048	DSPI Transmit FIFO Register 3	DSPI_TXFR3	32
Base + 0x004C	DSPI Transmit FIFO Register 4	DSPI_TXFR4	32
Base + (0x0050 – 0x0078)	Reserved	—	—
Base + 0x007C	Receive FIFO Register 0	DSPI_RXFR0	32
Base + 0x0080	Receive FIFO Register 1	DSPI_RXFR1	32
Base + 0x0084	Receive FIFO Register 2	DSPI_RXFR2	32
Base + 0x0088	Receive FIFO Register 3	DSPI_RXFR3	32
Base + 0x008C	Receive FIFO Register 4	DSPI_RXFR4	32
Base + (0x0090 – 0x3FFF)	Reserved	—	—
0xFFF9_8000	**DSPI 2** *Section 21.7: Memory map and registers description*		
Base + 0x0000	Module Configuration Register	DSPI_MCR	32
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	Transfer Count Register	DSPI_TCR	32
Base + 0x000C	Clock and Transfer Attribute Register 0	DSPI_CTAR0	32
Base + 0x0010	Clock and Transfer Attribute Register 1	DSPI_CTAR1	32
Base + 0x0014	Clock and Transfer Attribute Register 2	DSPI_CTAR2	32
Base + 0x0018	Clock and Transfer Attribute Register 3	DSPI_CTAR3	32
Base + 0x001C	Clock and Transfer Attribute Register 4	DSPI_CTAR4	32
Base + 0x002C	Clock and Transfer Attribute Register 5	DSPI_CTAR5	32
Base + 0x0024	Clock and Transfer Attribute Register 6	DSPI_CTAR6	32
Base + 0x0028	Clock and Transfer Attribute Register 7	DSPI_CTAR7	32
Base + 0x002C	Status Register	DSPI_SR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0030	DMA/Interrupt Request Register	DSPI_RSER	32
Base + 0x0034	PUSH TX FIFO Register	DSPI_PUSHR	32
Base + 0x0038	POP RX FIFO Register	DSPI_POPR	32
Base + 0x003C	DSPI Transmit FIFO Register 0	DSPI_TXFR0	32
Base + 0x0040	DSPI Transmit FIFO Register 1	DSPI_TXFR1	32
Base + 0x0044	DSPI Transmit FIFO Register 2	DSPI_TXFR2	32
Base + 0x0048	DSPI Transmit FIFO Register 3	DSPI_TXFR3	32
Base + 0x004C	DSPI Transmit FIFO Register 4	DSPI_TXFR4	32
Base + (0x0050 – 0x0078)	Reserved	—	—
Base + 0x007C	Receive FIFO Register 0	DSPI_RXFR0	32
Base + 0x0080	Receive FIFO Register 1	DSPI_RXFR1	32
Base + 0x0084	Receive FIFO Register 2	DSPI_RXFR2	32
Base + 0x0088	Receive FIFO Register 3	DSPI_RXFR3	32
Base + 0x008C	Receive FIFO Register 4	DSPI_RXFR4	32
Base + (0x0090 – 0x3FFF)	Reserved	—	—
0xFFF9_C000	**DSPI 3** *Section 21.7: Memory map and registers description*		
Base + 0x0000	Module Configuration Register	DSPI_MCR	32
Base + (0x0004 – 0x0007)	Reserved	—	—
Base + 0x0008	Transfer Count Register	DSPI_TCR	32
Base + 0x000C	Clock and Transfer Attribute Register 0	DSPI_CTAR0	32
Base + 0x0010	Clock and Transfer Attribute Register 1	DSPI_CTAR1	32
Base + 0x0014	Clock and Transfer Attribute Register 2	DSPI_CTAR2	32
Base + 0x0018	Clock and Transfer Attribute Register 3	DSPI_CTAR3	32
Base + 0x001C	Clock and Transfer Attribute Register 4	DSPI_CTAR4	32
Base + 0x002C	Clock and Transfer Attribute Register 5	DSPI_CTAR5	32
Base + 0x0024	Clock and Transfer Attribute Register 6	DSPI_CTAR6	32
Base + 0x0028	Clock and Transfer Attribute Register 7	DSPI_CTAR7	32
Base + 0x002C	Status Register	DSPI_SR	32
Base + 0x0030	DMA/Interrupt Request Register	DSPI_RSER	32
Base + 0x0034	PUSH TX FIFO Register	DSPI_PUSHR	32
Base + 0x0038	POP RX FIFO Register	DSPI_POPR	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x003C	DSPI Transmit FIFO Register 0	DSPI_TXFR0	32
Base + 0x0040	DSPI Transmit FIFO Register 1	DSPI_TXFR1	32
Base + 0x0044	DSPI Transmit FIFO Register 2	DSPI_TXFR2	32
Base + 0x0048	DSPI Transmit FIFO Register 3	DSPI_TXFR3	32
Base + 0x004C	DSPI Transmit FIFO Register 4	DSPI_TXFR4	32
Base + (0x0050 – 0x0078)	Reserved	—	—
Base + 0x007C	Receive FIFO Register 0	DSPI_RXFR0	32
Base + 0x0080	Receive FIFO Register 1	DSPI_RXFR1	32
Base + 0x0084	Receive FIFO Register 2	DSPI_RXFR2	32
Base + 0x0088	Receive FIFO Register 3	DSPI_RXFR3	32
Base + 0x008C	Receive FIFO Register 4	DSPI_RXFR4	32
Base + (0x0090 – 0x3FFF)	Reserved	—	—
0xFFFC_0000	**FlexCAN 0** *Section 23.3: Memory map and registers description*		
Base + 0x0000	Module Configuration	MCR	32
Base + 0x0004	Control Register	CTRL	32
Base + 0x0008	Free Running Timer	TIMER	32
Base + (0x000C – 0x000F)	Reserved	—	—
Base + 0x0010	Rx Global Mask Register	RXGMASK	32
Base + 0x0014	Rx 14 Mask Register	RX14MASK	32
Base + 0x0018	Rx 15 Mask Register	RX15MASK	32
Base + 0x001C	Error Counter Register	ECR	32
Base + 0x0020	Error and Status Register	ESR	32
Base + (0x0024 – 0x0027)	Reserved	—	—
Base + 0x0028	Interrupt Masks 1 Register	IMASK1	32
Base + (0x002C – 0x002F)	Reserved	—	—
Base + 0x0030	Interrupt Flags 1 Register	IFLAG1	32
Base + (0x0034 – 0x007F)	Reserved	—	—
Base + 0x0080	Message Buffer 0	MB0	128
Base + 0x0090	Message Buffer 1	MB1	128
Base + 0x00A0	Message Buffer 2	MB2	128
Base + 0x00B0	Message Buffer 3	MB3	128

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00C0	Message Buffer 4	MB4	128
Base + 0x00D0	Message Buffer 5	MB5	128
Base + 0x00E0	Message Buffer 6	MB6	128
Base + 0x00F0	Message Buffer 7	MB7	128
Base + 0x0100	Message Buffer 8	MB8	128
Base + 0x0110	Message Buffer 9	MB9	128
Base + 0x0120	Message Buffer 10	MB10	128
Base + 0x0130	Message Buffer 11	MB11	128
Base + 0x0140	Message Buffer 12	MB12	128
Base + 0x0150	Message Buffer 13	MB13	128
Base + 0x0160	Message Buffer 14	MB14	128
Base + 0x0170	Message Buffer 15	MB15	128
Base + 0x0180	Message Buffer 16	MB16	128
Base + 0x0190	Message Buffer 17	MB17	128
Base + 0x01A0	Message Buffer 18	MB18	128
Base + 0x01B0	Message Buffer 19	MB19	128
Base + 0x01C0	Message Buffer 20	MB20	128
Base + 0x01D0	Message Buffer 21	MB21	128
Base + 0x01E0	Message Buffer 22	MB22	128
Base + 0x01F0	Message Buffer 23	MB23	128
Base + 0x0200	Message Buffer 24	MB24	128
Base + 0x0210	Message Buffer 25	MB25	128
Base + 0x0220	Message Buffer 26	MB26	128
Base + 0x0230	Message Buffer 27	MB27	128
Base + 0x0240	Message Buffer 28	MB28	128
Base + 0x0250	Message Buffer 29	MB29	128
Base + 0x0260	Message Buffer 30	MB30	128
Base + 0x0270	Message Buffer 31	MB31	128
Base + (0x0280 – 0x087F)	Reserved	—	—
Base + 0x0880	RX Individual Mask Register 0	RXIMR0	32
Base + 0x0884	RX Individual Mask Register 1	RXIMR1	32
Base + 0x0888	RX Individual Mask Register 2	RXIMR2	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x088C	RX Individual Mask Register 3	RXIMR3	32
Base + 0x0890	RX Individual Mask Register 4	RXIMR4	32
Base + 0x0894	RX Individual Mask Register 5	RXIMR5	32
Base + 0x0898	RX Individual Mask Register 6	RXIMR6	32
Base + 0x089C	RX Individual Mask Register 7	RXIMR7	32
Base + 0x08A0	RX Individual Mask Register 8	RXIMR8	32
Base + 0x08A4	RX Individual Mask Register 9	RXIMR9	32
Base + 0x08A8	RX Individual Mask Register 10	RXIMR10	32
Base + 0x08AC	RX Individual Mask Register 11	RXIMR11	32
Base + 0x08B0	RX Individual Mask Register 12	RXIMR12	32
Base + 0x08B4	RX Individual Mask Register 13	RXIMR13	32
Base + 0x08B8	RX Individual Mask Register 14	RXIMR14	32
Base + 0x08BC	RX Individual Mask Register 15	RXIMR15	32
Base + 0x08C0	RX Individual Mask Register 16	RXIMR16	32
Base + 0x08C4	RX Individual Mask Register 17	RXIMR17	32
Base + 0x08C8	RX Individual Mask Register 18	RXIMR18	32
Base + 0x08CC	RX Individual Mask Register 19	RXIMR19	32
Base + 0x08D0	RX Individual Mask Register 20	RXIMR20	32
Base + 0x08D4	RX Individual Mask Register 21	RXIMR21	32
Base + 0x08D8	RX Individual Mask Register 22	RXIMR22	32
Base + 0x08DC	RX Individual Mask Register 23	RXIMR23	32
Base + 0x08E0	RX Individual Mask Register 24	RXIMR24	32
Base + 0x08E4	RX Individual Mask Register 25	RXIMR25	32
Base + 0x08E8	RX Individual Mask Register 26	RXIMR26	32
Base + 0x08EC	RX Individual Mask Register 27	RXIMR27	32
Base + 0x08F0	RX Individual Mask Register 28	RXIMR28	32
Base + 0x08F4	RX Individual Mask Register 29	RXIMR29	32
Base + 0x08F8	RX Individual Mask Register 30	RXIMR30	32
Base + 0x08FC	RX Individual Mask Register 31	RXIMR31	32
Base + (0x0900 – 0x3FFF)	Reserved	—	—
0xFFFD_C000	**DMA Channel Multiplexer (DMA_CH_MUX)** *Section 19.3.1: Memory map*		

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0000	Channel 0 Configuration Register 0	CHCONFIG0	8
Base + 0x0001	Channel 0 Configuration Register 1	CHCONFIG1	8
Base + 0x0002	Channel 0 Configuration Register 2	CHCONFIG2	8
Base + 0x0003	Channel 0 Configuration Register 3	CHCONFIG3	8
Base + 0x0004	Channel 0 Configuration Register 4	CHCONFIG4	8
Base + 0x0005	Channel 0 Configuration Register 5	CHCONFIG5	8
Base + 0x0006	Channel 0 Configuration Register 6	CHCONFIG6	8
Base + 0x0007	Channel 0 Configuration Register 7	CHCONFIG7	8
Base + 0x0008	Channel 0 Configuration Register 8	CHCONFIG8	8
Base + 0x0009	Channel 0 Configuration Register 9	CHCONFIG9	8
Base + 0x000A	Channel 0 Configuration Register 10	CHCONFIG10	8
Base + 0x000B	Channel 0 Configuration Register 11	CHCONFIG11	8
Base + 0x000C	Channel 0 Configuration Register 12	CHCONFIG12	8
Base + 0x000D	Channel 0 Configuration Register 13	CHCONFIG13	8
Base + 0x000E	Channel 0 Configuration Register 14	CHCONFIG14	8
Base + 0x000F	Channel 0 Configuration Register 15	CHCONFIG15	8
Base+ (0x0010 – 0x3FFF)	Reserved	—	—
0xFFFE_0000	**FlexRay Controller (FlexRay)** *Section 20.5.1: Memory map*		
Base + 0x0000	Module Version Register	MVR	16
Base + 0x0002	Module Configuration Register	MCR	16
Base + 0x0004	System Memory Base Address High Register	SYMBADHR	16
Base + 0x0006	System Memory Base Address Low Register	SYMBADLR	16
Base + 0x0008	Strobe Signal Control Register	STBSCR	16
Base + (0x000A – 0x000B)	Reserved	—	—
Base + 0x000C	Message Buffer Data Size Register	MBDSR	16
Base + 0x000E	Message Buffer Segment Size and Utilization Register	MBSSUTR	16
Base + (0x0010 – 0x0013)	Reserved	—	—
Base + 0x0014	Protocol Operation Control Register	POCR	16
Base + 0x0016	Global Interrupt Flag and Enable Register	GIFER	16
Base + 0x0018	Protocol Interrupt Flag Register 0	PIFR0	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x001A	Protocol Interrupt Flag Register 1	PIFR1	16
Base + 0x001C	Protocol Interrupt Enable Register 0	PIER0	16
Base + 0x001E	Protocol Interrupt Enable Register 1	PIER1	16
Base + 0x0020	CHI Error Flag Register	CHIERFR	16
Base + 0x0022	Message Buffer Interrupt Vector Register	MBIVEC	16
Base + 0x0024	Channel A Status Error Counter Register	CASERCR	16
Base + 0x0026	Channel A Status Error Counter Register	CBSERCR	16
Base + 0x0028	Protocol Status Register 0	PSR0	16
Base + 0x002A	Protocol Status Register 1	PSR1	16
Base + 0x002C	Protocol Status Register 2	PSR2	16
Base + 0x002E	Protocol Status Register 3	PSR3	16
Base + 0x0030	Macrotick Counter Register	MTCTR	16
Base + 0x0032	Cycle Counter Register	CYCTR	16
Base + 0x0034	Slot Counter Channel A Register	SLTCTAR	16
Base + 0x0036	Slot Counter Channel B Register	SLTCTBR	16
Base + 0x0038	Rate Correction Value Register	RTCORVR	16
Base + 0x003A	Offset Correction Value Register	OFCORVR	16
Base + 0x003C	Combined Interrupt Flag Register	CIFRR	16
Base + 0x003E	System Memory Access Time-Out Register	SYMATOR	16
Base + 0x0040	Sync Frame Counter Register	SFCNTR	16
Base + 0x0042	Sync Frame Table Offset Register	SFTOR	16
Base + 0x0044	Sync Frame Table Configuration, Control, Status Register	SFTCCSR	16
Base + 0x0046	Sync Frame ID Rejection Filter	SFIDRFR	16
Base + 0x0048	Sync Frame ID Acceptance Filter Value Register	SFIDAFVR	16
Base + 0x004A	Sync Frame ID Acceptance Filter Mask Register	SFIDAFMR	16
Base + 0x004C	Network Management Vector Register 0	NMVR0	16
Base + 0x004E	Network Management Vector Register 1	NMVR1	16
Base + 0x0050	Network Management Vector Register 2	NMVR2	16
Base + 0x0052	Network Management Vector Register 3	NMVR3	16
Base + 0x0054	Network Management Vector Register 4	NMVR4	16
Base + 0x0056	Network Management Vector Register 5	NMVR5	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0058	Network Management Vector Length Register	NMVLR	16
Base + 0x005A	Timer Configuration and Control Register	TICCR	16
Base + 0x005C	Timer 1 Cycle Set Register	TI1CYSR	16
Base + 0x005E	Timer 1 Macrotick Offset Register	TI1MTOR	16
Base + 0x0060	Timer 2 Configuration Register 0	TI2CR0	16
Base + 0x0062	Timer 2 Configuration Register 1	TI2CR1	16
Base + 0x0064	Slot Status Selection Register	SSSR	16
Base + 0x0066	Slot Status Counter Condition Register	SSCCR	16
Base + 0x0068	Slot Status Register 0	SSR0	16
Base + 0x006A	Slot Status Register 1	SSR1	16
Base + 0x006C	Slot Status Register 2	SSR2	16
Base + 0x006E	Slot Status Register 3	SSR4	16
Base + 0x0070	Slot Status Register 4	SSR4	16
Base + 0x0072	Slot Status Register 5	SSR5	16
Base + 0x0074	Slot Status Register 6	SSR6	16
Base + 0x0076	Slot Status Register 7	SSR7	16
Base + 0x0078	Slot Status Counter Register0	SSCR0	16
Base + 0x007A	Slot Status Counter Register 1	SSCR1	16
Base + 0x007C	Slot Status Counter Register 2	SSCR2	16
Base + 0x007E	Slot Status Counter Register 3	SSCR3	16
Base + 0x0080	MTS A Configuration Register	MTSACFR	16
Base + 0x0082	MTS B Configuration Register	MTSBCFR	16
Base + 0x0084	Receive Shadow Buffer Index Register	RSBIR	16
Base + 0x0086	Receive FIFO Selection Register	RFSR	16
Base + 0x0088	Receive FIFO Start Index Register	RFSIR	16
Base + 0x008A	Receive FIFO Depth and Size Register	RFDSR	16
Base + 0x008C	Receive FIFO A Read Index Register	RFARIR	16
Base + 0x008E	Receive FIFO B Read Index Register	RFBRIR	16
Base + 0x0090	Receive FIFO Message ID Acceptance Filter Value Register	RFMIDAFVR	16
Base + 0x0092	Receive FIFO Message ID Acceptance Filter Mask Register	RFMIAFMR	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0094	Receive FIFO Frame ID Rejection Filter Value Register	RFFIDRFVR	16
Base + 0x0096	Receive FIFO Frame ID Rejection Filter Mask Register	RFFIDRFMR	16
Base + 0x0098	Receive FIFO Range Filter Configuration Register	RFRFCFR	16
Base + 0x009A	Receive FIFO Range Filter Control Register	RFRFCTR	16
Base + 0x009C	Last Dynamic Transmit Slot Channel A Register	LDTXSLAR	16
Base + 0x009E	Last Dynamic Transmit Slot Channel B Register	LDTXSLBR	16
Base + 0x00A0	Protocol Configuration Register 0	PCR0	16
Base + 0x00A2	Protocol Configuration Register 1	PCR1	16
Base + 0x00A4	Protocol Configuration Register 2	PCR2	16
Base + 0x00A6	Protocol Configuration Register 3	PCR3	16
Base + 0x00A8	Protocol Configuration Register 4	PCR4	16
Base + 0x00AA	Protocol Configuration Register 5	PCR5	16
Base + 0x00AC	Protocol Configuration Register 6	PCR6	16
Base + 0x00AE	Protocol Configuration Register 7	PCR7	16
Base + 0x00B0	Protocol Configuration Register 8	PCR8	16
Base + 0x00B2	Protocol Configuration Register 9	PCR9	16
Base + 0x00B4	Protocol Configuration Register 10	PCR10	16
Base + 0x00B6	Protocol Configuration Register 11	PCR11	16
Base + 0x00B8	Protocol Configuration Register 12	PCR12	16
Base + 0x00BA	Protocol Configuration Register 13	PCR13	16
Base + 0x00BC	Protocol Configuration Register 14	PCR14	16
Base + 0x00BE	Protocol Configuration Register 15	PCR15	16
Base + 0x00C0	Protocol Configuration Register 16	PCR16	16
Base + 0x00C2	Protocol Configuration Register 17	PCR17	16
Base + 0x00C4	Protocol Configuration Register 18	PCR18	16
Base + 0x00C6	Protocol Configuration Register 19	PCR19	16
Base + 0x00C8	Protocol Configuration Register 20	PCR20	16
Base + 0x00CA	Protocol Configuration Register 21	PCR21	16
Base + 0x00CC	Protocol Configuration Register 22	PCR22	16
Base + 0x00CE	Protocol Configuration Register 23	PCR23	16
Base + 0x00D0	Protocol Configuration Register 24	PCR24	16

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00D2	Protocol Configuration Register 25	PCR25	16
Base + 0x00D4	Protocol Configuration Register 26	PCR26	16
Base + 0x00D6	Protocol Configuration Register 27	PCR27	16
Base + 0x00D8	Protocol Configuration Register 28	PCR28	16
Base + 0x00DA	Protocol Configuration Register 29	PCR29	16
Base + 0x00DC	Protocol Configuration Register 30	PCR30	16
Base + (0x00DE – 0x00FF)	Reserved	—	—
Base + 0x0100	Message Buffer 0	MB0	64
Base + 0x0108	Message Buffer 1	MB1	64
Base + 0x0110	Message Buffer 2	MB2	64
Base + 0x0118	Message Buffer 3	MB3	64
Base + 0x0120	Message Buffer 4	MB4	64
Base + 0x0128	Message Buffer 5	MB5	64
Base + 0x0130	Message Buffer 6	MB6	64
Base + 0x0138	Message Buffer 7	MB7	64
Base + 0x0140	Message Buffer 8	MB8	64
Base + 0x0148	Message Buffer 9	MB9	64
Base + 0x0150	Message Buffer 10	MB10	64
Base + 0x0158	Message Buffer 11	MB11	64
Base + 0x0160	Message Buffer 12	MB12	64
Base + 0x0168	Message Buffer 13	MB13	64
Base + 0x0170	Message Buffer 14	MB14	64
Base + 0x0178	Message Buffer 15	MB15	64
Base + 0x0180	Message Buffer 16	MB16	64
Base + 0x0188	Message Buffer 17	MB17	64
Base + 0x0190	Message Buffer 18	MB18	64
Base + 0x0198	Message Buffer 19	MB19	64
Base + 0x01A0	Message Buffer 20	MB20	64
Base + 0x01A8	Message Buffer 21	MB21	64
Base + 0x01B0	Message Buffer 22	MB22	64
Base + 0x01B8	Message Buffer 23	MB23	64
Base + 0x01C0	Message Buffer 24	MB24	64

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x01C8	Message Buffer 25	MB25	64
Base + 0x01D0	Message Buffer 26	MB26	64
Base + 0x01D8	Message Buffer 27	MB27	64
Base + 0x01E0	Message Buffer 28	MB28	64
Base + 0x01E8	Message Buffer 29	MB29	64
Base + 0x01F0	Message Buffer 30	MB30	64
Base + 0x01F8	Message Buffer 31	MB31	64
Base + (0x0200 – 0x7FFF)	Reserved	—	—
0xFFFE_8000	**Safety port** *Section 28.2.5.1: Register protection memory map*		
Base + 0x0000	Module Configuration	MCR	32
Base + 0x0004	Control Register	CTRL	32
Base + 0x0008	Free Running Timer	TIMER	32
Base + (0x000C – 0x000F)	Reserved	—	—
Base + 0x0010	Rx Global Mask Register	RXGMASK	32
Base + 0x0014	Rx 14 Mask Register	RX14MASK	32
Base + 0x0018	Rx 15 Mask Register	RX15MASK	32
Base + 0x001C	Error Counter Register	ECR	32
Base + 0x0020	Error and Status Register	ESR	32
Base + (0x0024 – 0x0027)	Reserved	—	—
Base + 0x0028	Interrupt Masks 1 Register	IMASK1	32
Base + (0x002C – 0x002F)	Reserved	—	—
Base + 0x0030	Interrupt Flags 1 Register	IFLAG1	32
Base + (0x0034 – 0x007F)	Reserved	—	—
Base + 0x0080	Message Buffer 0	MB0	128
Base + 0x0090	Message Buffer 1	MB1	128
Base + 0x00A0	Message Buffer 2	MB2	128
Base + 0x00B0	Message Buffer 3	MB3	128
Base + 0x00C0	Message Buffer 4	MB4	128
Base + 0x00D0	Message Buffer 5	MB5	128
Base + 0x00E0	Message Buffer 6	MB6	128

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x00F0	Message Buffer 7	MB7	128
Base + 0x0100	Message Buffer 8	MB8	128
Base + 0x0110	Message Buffer 9	MB9	128
Base + 0x0120	Message Buffer 10	MB10	128
Base + 0x0130	Message Buffer 11	MB11	128
Base + 0x0140	Message Buffer 12	MB12	128
Base + 0x0150	Message Buffer 13	MB13	128
Base + 0x0160	Message Buffer 14	MB14	128
Base + 0x0170	Message Buffer 15	MB15	128
Base + 0x0180	Message Buffer 16	MB16	128
Base + 0x0190	Message Buffer 17	MB17	128
Base + 0x01A0	Message Buffer 18	MB18	128
Base + 0x01B0	Message Buffer 19	MB19	128
Base + 0x01C0	Message Buffer 20	MB20	128
Base + 0x01D0	Message Buffer 21	MB21	128
Base + 0x01E0	Message Buffer 22	MB22	128
Base + 0x01F0	Message Buffer 23	MB23	128
Base + 0x0200	Message Buffer 24	MB24	128
Base + 0x0210	Message Buffer 25	MB25	128
Base + 0x0220	Message Buffer 26	MB26	128
Base + 0x0230	Message Buffer 27	MB27	128
Base + 0x0240	Message Buffer 28	MB28	128
Base + 0x0250	Message Buffer 29	MB29	128
Base + 0x0260	Message Buffer 30	MB30	128
Base + 0x0270	Message Buffer 31	MB31	128
Base + (0x0280 – 0x087F)	Reserved	—	—
Base + 0x0880	RX Individual Mask Register 0	RXIMR0	32
Base + 0x0884	RX Individual Mask Register 1	RXIMR1	32
Base + 0x0888	RX Individual Mask Register 2	RXIMR2	32
Base + 0x088C	RX Individual Mask Register 3	RXIMR3	32
Base + 0x0890	RX Individual Mask Register 4	RXIMR4	32
Base + 0x0894	RX Individual Mask Register 5	RXIMR5	32

Table 609. Detailed Memory Map(Continued)

Address	Register description	Register name	Size (bits)
Base + 0x0898	RX Individual Mask Register 6	RXIMR6	32
Base + 0x089C	RX Individual Mask Register 7	RXIMR7	32
Base + 0x08A0	RX Individual Mask Register 8	RXIMR8	32
Base + 0x08A4	RX Individual Mask Register 9	RXIMR9	32
Base + 0x08A8	RX Individual Mask Register 10	RXIMR10	32
Base + 0x08AC	RX Individual Mask Register 11	RXIMR11	32
Base + 0x08B0	RX Individual Mask Register 12	RXIMR12	32
Base + 0x08B4	RX Individual Mask Register 13	RXIMR13	32
Base + 0x08B8	RX Individual Mask Register 14	RXIMR14	32
Base + 0x08BC	RX Individual Mask Register 15	RXIMR15	32
Base + 0x08C0	RX Individual Mask Register 16	RXIMR16	32
Base + 0x08C4	RX Individual Mask Register 17	RXIMR17	32
Base + 0x08C8	RX Individual Mask Register 18	RXIMR18	32
Base + 0x08CC	RX Individual Mask Register 19	RXIMR19	32
Base + 0x08D0	RX Individual Mask Register 20	RXIMR20	32
Base + 0x08D4	RX Individual Mask Register 21	RXIMR21	32
Base + 0x08D8	RX Individual Mask Register 22	RXIMR22	32
Base + 0x08DC	RX Individual Mask Register 23	RXIMR23	32
Base + 0x08E0	RX Individual Mask Register 24	RXIMR24	32
Base + 0x08E4	RX Individual Mask Register 25	RXIMR25	32
Base + 0x08E8	RX Individual Mask Register 26	RXIMR26	32
Base + 0x08EC	RX Individual Mask Register 27	RXIMR27	32
Base + 0x08F0	RX Individual Mask Register 28	RXIMR28	32
Base + 0x08F4	RX Individual Mask Register 29	RXIMR29	32
Base + 0x08F8	RX Individual Mask Register 30	RXIMR30	32
Base + 0x08FC	RX Individual Mask Register 31	RXIMR31	32
Base + (0x0900 – 0x3FFF)	Reserved	—	—
0xFFFF_C000	**Boot Assist Module (BAM)** *Section 34.4: Memory map*		
Base + 0x0000	BAM Entry Point	—	—
Base + (0x2000 – 0x3FFF)	Reserved	—	—

Revision history

Table 610. Document revision history

Date	Revision	Changes
23-Jun-2008	1	Initial release.
24-Oct-2008	2	Section 1.1, Chipset overview; updated the contents. Section 1.2, Target applications: deleted a paragraph. Section 1.3, Features: Table 1: removed the note of “Execution Speed” Section 1.5, Critical performance parameters : removed a note Section 1.7.4, On-chip Flash with ECC: replaced “3 wait state for page buffer miss at 60MHz“ with “2 wait state for page buffer miss at 64MHz“ Section 1.7.19, Safety port (FlexCAN): “up to 7.5Mbit at 60MHz“ with “up to 8Mbit at 64MHz“ Section 1.10, SPC560P44Lx, SPC560P50Lx memory map; deleted first paragraph. Section 3.10, IRCOSC 16 MHz internal RC oscillator: Removed RCDIV field from RC_CTL register. Section 3.12.3, Features: Removed “1:1 Mode“ from features list Section 3.12.6.1, Normal mode: Replaced the denominator with “idf x odf” instead of ”ldf x odf”. Section 3.13.4.5, Interrupt Status Register (CMU_0_ISR) - Table 38 and Section 3.13.4.10, Interrupt Status Register (CMU_1_ISR) - Table 43: For the FLCI_A field the reference clock is FRC/4 and not FRC/16 Chapter 4, "Operating Modes": Added S_PLL1 field on Figure 19 (Global Status Register (ME_GS)) and removed S_SSCLK1 from table 66 (Global Status Register (ME_GS) field description) Figure 6: replaced “FMPLL_1_CLK_DIV“ and “SP_PLL_CLK“ frequency value from 60MHz to 64MHz Table 46: removed “ME_PCTL86 register“ Figure 131; replaced “XBAR“ with “Peripheral Bridge“ in the title. Section 5.3.1, Memory map:Showed PCONF0 register. Section 5.3.2.1, Power Domain #0 Configuration register (PCU_PCONF0): Showed. Section 9.5.2.8, Pad Configuration Registers (PCR[0:107]): Removed the field HYS. Section 7.4.1, Normal mode: inserted a note about “core register IVPR“ Chapter 13, Error Correction Status Module (ECSM): replaced the chapter title instead of “Miscellaneous Control Module (MCM)”. Chapter 20, "LIN controller (LINFlex)": updated offset and format of all registers. Section 20.7.1, Memory Map: Changed the size of BDRL and BDRM from 16-bit to 32-bit. Chapter 23, "ADC Digital Interface (ADCDig)": Replaced with a whole chapter Section 27.6.1, Overview: provide the eTimer_BASE = FFE1_8000. Table 156: showed again. Chapter 36, "Document Revision History": Hidden to customers the “Rev. 0” and “Rev. 1 Draft A” rows and added: Rev. 1 = Initial release Rev. 2 = (all modification applied) Cover page: replaced title by “32-bit MCU family built on Power Architecture for automotive Chassis&Safety application“ Fixed the numbering of the chapters from 21 onward. Throgh whole document:

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3	replaced “Error Correction Status Module (ECSM)“ instead of “Miscellaneous Control Module (MCM)”. Significant editorial enhancement and correction of technical content throughout the document Added Preface Chapter 1 Overview Editorial and formatting changes Updated features and device block diagram Replaced SPC560P44Lx, SPC560P50Lx product table with SPC560P44Lx, SPC560P50Lx device comparison table Section 1.8 Developer environment: Updated software support list Section 1.7.5 On-chip SRAM with ECC: Removed sentence referencing compatibility of device’s ECC handling to ECC handling in e200z6 core devices Section 1.7.7 System clocks and clock generation: Removed “(tbc)” from end of on-chip oscillator bullet Section 1.9 Package: Updated ECOPACK text Section 1.10 SPC560P44Lx, SPC560P50Lx memory map: Updated DMA_MUX and PIT region names; replaced DMA2x with eDMA Deserial serial peripheral interface (DSPI) module: Modified description of DSPI CS lines Changed value of *can0* variable from “FlexCan 0 (CAN0)” to “FlexCAN 0 (CAN0)” Chapter 2 Signal Description Editorial and formatting changes Updated LQFP 100- and 144-pin pinouts Removed section “Detailed Signal Description” (signal descriptions already provided in tables “System Pins” and “Pin Muxing”) Updated Pin muxing “Pin muxing“ table, changed port pin D[3] direction for function ALT3 from I/O to O Updated CTU / ADCs / FlexPWM / eTimers connections diagram Removed “Nexus Message Data Out (MDO[0])” section Remove refs to TCKC, TDOC, TDIC, TMSC, TCKC signals . This device does not support 1149.7 standard. It supports 1149.1 only. Chapter 3 Clock Description Editorial and formatting changes Changed FMPLL0 and FMPLL1 to FMPLL_0 and FMPLL_1 Replaced IRCOSC with IRC Replaced IRCOSC_CLK with IRC_CLK Removed MC_ prefixes Replaced PLL_*n* with FMPLL_*n* Replaced XOSC0 with XOSC Replaced occurrences of “OSC” with “XOSC” (to indicate external crystal oscillator) SPC560P44Lx, SPC560P50Lx system clock generation: Replaced “÷15” with “÷16” in dividers MC_PLL, CMU_PLL, SP_PLL and FR_PLL

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Section 3.4 Available clock domains: Changed AIPS-Lite to PBRIDGE Updated clock frequency symbols Crystal oscillator truth table: Replaced “Hiz” with “High Z” FMPLL Section 3.11.3 Features: Corrected number of available modes: was 5, is 4 Corrected and harmonized names of registers High Frequency Reference Register FMPLL_1 (CMU_1_HFREFR_A) and Low Frequency Reference Register FMPLL_1 (CMU_1_LFREFR_A) in memory map and register descriptions Added Progressive clock switching scheme CMU Section 3.12.1 Overview: Replaced RCOSC with IRC Updated and harmonized CMU register names and bitfield names Crystal clock monitor: Amended description of device behavior after failure event is signaled PLL clock monitor: Rephrased note about possibiltiy of XOSC or PLL monitors producing a false event; amended description of device behavior after failure events are signaled Chapter 4 Operating Modes Editorial and formatting changes Replaced all IRCOSC with IRC Replaced all IRCOSC_CLK with IRC_CLK Removed all MC_ prefixes Replaced all PLL_*n* with FMPLL_*n* Replaced all XOSC0 with XOSC Section “DRUN Mode Configuration register (ME_DRUN_MC)”: Removed second note at end of section (“The values of CFLAON and DFLAON are loaded from, respectively, on DRUN mode entry from the STANDBY0 mode.”) Chapter 5 Power Control Unit (PCU) Editorial and formatting changes Replaced all IRCOSC with IRC Replaced all IRCOSC_CLK with IRC_CLK Removed all MC_ prefixes Replaced all PLL_*n* with FMPLL_*n* Replaced all XOSC0 with XOSC Removed all references to STANDBY mode Updated PCU memory map for VREG registers Power Domain #1 Configuration register (PCU_PCONF1): Removed sentence referencing section “STANDBY0 mode transition” PCU registers: Updated register bit 23 in registers VREG_CTL and VREG_STATUS Chapter 6 Reset Generation Module (RGM) Editorial and formatting changes Replaced all IRCOSC with IRC Replaced all IRCOSC_CLK with IRC_CLK Removed all MC_ prefixes Replaced all PLL_n with FMPLL_n Replaced all XOSC0 with XOSC Updated Section 6.2, “Features Section “Destructive Event Status register (RGM_DES)”: Added two notes at end of section

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Chapter 7 Interrupt Controller (INTC) Editorial and formatting changes Table 85: Removed device column; filled out empty offset values INTC End-of-Interrupt Register (INTC_EOIR): Aligned bitmap cell shading to reflect write-only access Removed MC_ prefixes: – was MC_ME; is ME – was MC_RGM; is RGM Section 7.6 Functional description: Replaced “INTC_PSR211” with “INTC_PSR221” in note Chapter 8 System Status and Configuration Module (SSCM) Editorial and formatting changes Section 8.1.2 Features: Removed “Device identification information” bullet Section 8.2.2 Register description: Modified introduction and added a key to the register fields System Status register (STATUS): – Added fields PUB and SEC – Changed BMODE[0:2] to BMODE[2:0] in STATUS field descriptions System Memory Configuration register (MEMCONFIG): Updated register reset value and field descriptions MEMCONFIG field descriptions: Replaced “IFLASH” with “CFlash” in notes Error Configuration (ERROR) register: Modified description of field RAE; added note below field descriptions table ERROR field descriptions: Modified RAE description note to update bit names and replace AIPS with PBRIDGE DEBUGPORT field descriptions: Changed DEBUG_PORT[0:2] to DEBUG_PORT[2:0] Password comparison registers: Changed bit numbering for fields PWD_HI and PWD_LO from 0:31 to 31:0 Chapter 9 System Integration Unit Lite (SIUL) Editorial and formatting changes Updated System Integration Unit Lite block diagram System Integration Unit Lite block diagram: Replaced GPIO count 108 with 106 Reorganized and updated SIUL signal properties table Pad selection: – Moved column for 144 pins to right of column for 100 pins – Replaced “ECT” with “ETC” in port column Section 9.3 Features: – Replaced “GPIO function on 108 I/O pins” with “GPIO function on up to 106 I/O pins” – Replaced “4 system interrupt vectors for 32 interrupt sources” with “4 system interrupt vectors for up to 32 interrupt sources” Section 9.3.1 Register protection: Rephrased first sentence Section 9.4.1 Detailed signal descriptions: Replaced GPIO[0:107] with GPIO[0:105] Interrupt Filter Enable Register (IFER) bitmap: Corrected field name—was IFEE, is IFE General-purpose I/O pins (GPIO[0:105]): Removed paragraph referencing inputs IN1 and IN2 Section External interrupt request input pins (EIRQ[0:31]): Replaced SIU_ with SIUL_

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	SIUL memory map: – Updated address offsets – Replaced “0x0034–0x00FF” with “0x0034–0x003F” in ‘Reserved’ row between IFER and PCR rows Section 9.5.2 Register description: Added figure “Key to register fields” MCU ID Register #1 (MIDR1): Updated reset value for field PKG[4:0] in bitmap Section MCU ID Register #2 (MIDR2): – Changed reset value for field EE in bitmap to ‘x’ with footnote – Changed reset value for field FLASH_SIZE_2[3:0] in bitmap from ‘xxxx’ to ‘0000’ – MIDR2 field descriptions: Updated values for field SF Pad Selection for Multiplexed Inputs registers (PSMI[0_3:32_35]): Added [3:0] to PADSEL fields of bitmap MPGPDO[0:6] field descriptions: Replaced “MPPDO[x] register” with “MPPDO[x] field” in description of field MASK[x] Section 9.6.3 General purpose input and output pads (GPIO): Replaced “The SIUL manages 108 GPIO pads” with “The SIUL manages 106 GPIO pads; corrected cross-reference at end of section Chapter 10 e200z0 and e200z0h Core Editorial and formatting changes Section 10.1 Overview: Added sentence to specify which core version is implemented by the device Section 10.2 Features: Removed bullet “Power saving modes: doze, nap, sleep, and wait” Chapter 11 Peripheral Bridge (PBRIDGE) Editorial and formatting changes Chapter 12 Crossbar Switch (XBAR) Editorial and formatting changes Section 12.4 Features: Amended descriptions Device XBAR switch ports: Added row for Data Hardwired bus master priorities: Added row for Data Chapter 13 Error Correction Status Module (ECSM) Editorial and formatting changes Updated MUDCR register definition ECC Configuration Register (ECR): Updated content to clarify that single-bit memory corrections is always enabled ECC Error Generation Register (EEGR): Updated content to clarify that single-bit memory corrections is always enabled Chapter 14 Internal Static RAM (SRAM) Editorial and formatting changes Chapter 15 Flash Memory Substantial changes to and general restructuring and update of entire chapter Chapter 16 Enhanced Direct Memory Access (eDMA) Editorial and formatting changes

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Chapter 17 DMA Channel Mux (DMA_MUX) Editorial and formatting changes Section 17.3 Memory map and register definition: Removed redundant “Module memory map” table (DMA_MUX memory map table retained) Added heading Section 17.3.1 Memory map Section 17.3.2 Register descriptions: Removed introduction text (same information already present in Section 17.3 Memory map and register definition) DMA mux triggered channels diagram: – Changed figure title – Corrected trigger numbering: Was #1, #1, ...#4; is #1, #2, ...#4 – Added “Always #1” and “Always #4” arrow labels DMA mux channel 4–15 block diagram: Replaced “8–15” with “4–15” in figure title Chapter 18 FlexRay Communication Controller (FlexRay) Editorial and formatting changes Changed PLL1 to FMPLL_1 Made MCR[CLKSEL] visible Message buffer update: Replaced “determined by the following two items” with “determined by the following three items” Chapter 19 Deserial Serial Peripheral Interface (DSPI) Editorial and formatting changes Updated Table 375 “Baud rate values” for 100 MHz clock Updated DSPI block diagram Section 19.3 Overview: Replaced “DSPI_0 has two additional chip select (CS) lines” with “DSPI_0 has four additional chip select (CS) lines” Section 19.5 Modes of operation: Inserted sentence “All four modes are implemented on this device.” Section 19.5.4 Debug mode: Removed phrase “and remains dictated by the module-specific mode and configuration of the DSPI” from first paragraph Peripheral Chip Select 4 (CS4): Added statement saying that signal is not used in slave mode Section 19.7 Memory map and registers description: Added registers DSPI_TXFR4 and DSPI_RXFR4 DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx_CTARn): Replaced “QSPI module in the SPC560Pxx family of MCUs” with “QSPI module used in certain members of the SPC56x family of MCUs” DSPIx_CTARn field descriptions: –Modified description of field FMSZ to remove content about operating in TSB configuration –Modified description of field LSBFE to remove content about operating in TSB configuration –Modified description of field DT to remove content about enabling the TSB configuration Section 19.8.1 Modes of operation: Inserted sentence “All four modes are implemented on this device.” Section 19.8.6 Continuous Serial communications clock: Removed phrase “or the DCONT in the DSPIx_DSICR is set” Address calculation for first-in entry and last-in entry in TX FIFO: Replaced “implementation specific” with “(depth is 5)”

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Address calculation for first-in entry and last-in entry in RX FIFO: Replaced “implementation specific” with “(depth is 5)” Removed section “SPC560Pxx QSPI Compatibility with the DSPI” Removed all content describing CSI and DSI configurations LIN Controller (LINFlex): Substantial changes and major update to entire chapter FlexCAN Editorial and formatting changes Throughout chapter, made changes to reflect 32 Message Buffers, not 64 Removed content referencing Self Wake Up in Section 21.1.2 FlexCAN module features and Section 21.1.3 Modes of operation Section 21.3.1 FlexCAN memory mapping: Updated address offset ranges and RAM sizes Section Module Configuration Register (MCR): Updated bitmap and field descriptions Added RX14MASK field description Section Rx 15 Mask (RX15MASK): Replaced “RXG14MASK” with “RX15MASK” Added RX15MASK field description Section Rx Individual Mask Registers (RXIMR0–RXIMR31): Replaced “RXIM63” with “RXIMR31” in bitmap and field description titles Updated Section 21.4.9.1 Freeze mode Stop mode: Removed content referencing Self Wake Up Section 21.4.11 Bus interface: Removed sentence referencing Skyblue interface; minor editorial change in first sentence; updated address offset ranges Chapter 22 Analog-to-Digital Converter (ADC): Substantial changes, major restructuring and update of entire chapter Chapter 23 Cross Triggering Unit (CTU) Editorial and formatting changes throughout Removed MC_ prefixes from module names Section 23.4 Scheduler subunit (SU): Amended descriptions of eTimer1 pulse and eTimer2 pulse Scheduler subunit diagram: Reversed arrow direction for NEXT_CMD_1 Rewrote Section 23.4.2 ADC commands list format Amended Section 23.6 Power safety mode Section 23.7.2 CTU faults and errors: Replaced “trigger to the eTimer1” with “trigger to the eTimer0/1” Added bit descriptions for some bit tables (e.g., 0 Interrupt is disabled. 1 Interrupt is enabled.) In CLRx register figures and tables, changed bit LC (First Command) to FC (First Command) CLRx (CMS = 0) field descriptions: Updated FIFO description FRx field descriptions: Updated N_CH description FLx field descriptions: Updated N_CH description

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Chapter 24 FlexPWM Editorial and formatting changes Replaced instances of WAIT mode with WAIT/HALT mode Changed Control Register (CTRL) name to Control 1 Register (CTRL1) CTRL2 field descriptions: Replaced “FORCE signal” with “FORCE_OUT signal” in FRCEN field description STS field descriptions: – Removed reference to inexistent “FRACx” registers – Replaced “one of the INIT, VALx, or PRSC registers” with “one of the INIT, VALx, or PRSC fields” in SU field description DMAEN field descriptions: – Removed reference to inexistent “FRACx” registers – Removed duplicated 0 value definition from VALDE field description – Removed reference to bits CAxDE and CBxDE from FAND field description Deadtime Count registers (DTCNT0, DTCNT1): Inserted sentence “Reset sets the deadtime count registers to a default value of 0x0FFF, selecting a deadtime of 4095 peripheral clock cycles.” DTSRCSEL field descriptions: Added value ‘11’ as reserved to all fields Updated MCTRL field descriptions “Capture Value 0 Cycle register (CVAL0CYC)” and “Capture Value 1 Cycle register (CVAL1CYC)” use only bits [13:15] instead of bits[12:15] Section 24.7 Functional description: Removed subsection “Block diagram” Section 24.9.4 Reload errors: Removed reference to inexistent “FRACx” registers DMA summary: Deleted Submodule 2 read request Chapter 25 eTimer Editorial and formatting changes Section 25.2 Features: Removed faults input bullet eTimer channel block diagram: Removed fault input Added register bitfield tables Changed register name from CTRL to CTRL1 CTRL2 field descriptions: Removed reference to adjacent channels “6 and 7” from RDNT field description Control register 3 (CTRL3): Changed value from ‘0’ to ‘1’ in read cells of bits 7:10 DREQn field descriptions: Added bit values 01000 – 10111 to DREQn field description Section 25.7.1 General: Removed bullet “The response of the OFLAG output to a fault input is programmable.” Section ONE-SHOT mode: Changed phrase “OFLAG OUTMODE is set to 101” to “OFLAG OUTMODE is set to 0101” Section CASCADE-COUNT mode: Replaced phrase “and channels 6 and 5 cascaded separately” with “and channels 5 and 4 cascaded separately” in note at end of section Section PULSE-OUTPUT mode: Changed phrase “OFLAG OUTMODE is set to 111” to “OFLAG OUTMODE is set to 1111”

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Section FIXED-FREQUENCY PWM mode: Changed phrase “OFLAG OUTMODE is 110” to “OFLAG OUTMODE is 0111” VARIABLE-FREQUENCY PWM mode: Changed phrase “OFLAG OUTMODE is 100” to “OFLAG OUTMODE is 0100” Usage of compare registers: – Changed phrase “When the output mode is set to 100” to “When the output mode is set to 0100” – Amended content to specify that COMP1 is used when OFLAG == 0 and COMP2 is used when OFLAG == 1 – Replaced CMPx with COMPx STS field descriptions: Removed reference to channels 6 and 7 from RCF field description INTDMA field descriptions: Removed reference to channel 6 from RCFIE field description DMA summary: In DMA Requests column, replaced “Channels 0–1” with “Channels 0–5” Chapter 26 Functional Safety Editorial and formatting changes Changed AIPS to PBRIDGE Register protection memory map: Replaced “SSCM” with “Safety Port” in first sentence Chapter 27 Fault Collection Unit (FCU) Editorial, formatting and organizational changes; substantial changes to whole chapter Section 27.1.1 Overview: Replaced ECU with MCU Fault Collection Unit (FCU) block diagram: Replaced “CRPM state” with “Clock/Reset/Power/Mode state” Module Configuration Register (FCU_MCR): Added field numbers to multi-bit fields; replaced field name M with TM Frozen Fault Flag Register (FCU_FFFR): Changed field name ‘SRF’ to ‘FRSRF’ and ‘HRF’ to ‘FRHRF’ FCU_FFGR field description: – Replaced “FSRF2–SRF4” with “FSRF2–FSRF4” – Replaced “Fake Software Recoverable Fault[2:15]” with “Fake Software Recoverable Fault[2:4]” FCU_FER field descriptions: – Replaced “EHF15–ESF0” with “EHF15–EHF0” – Added field names in description cells FCU_TR field descriptions: Changed field name ‘FCUT’ to ‘TR’ FCU_TER field descriptions: Added field names in description cells Frozen MC State Register (FCU_FMCSR): Changed field name ‘MCPS’ to ‘FRMCPS’ and ‘MCAS’ to ‘FRMCAS’ FCU_MCSR field description: Changed description for value 0110 from ‘STANDBY’ to ‘Reserved’ in fields MCPS and MCAS Section 27.3 Functional description: Added description of output generation logic block Section 27.3.1 State machine: Changed “Optionally, the output pins can be disabled in Test mode (by setting the TM field to ‘10’)” to “Optionally, the output pins can be disabled in Test mode (by setting the TM field to ‘01’).”

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Chapter 28 Wakeup Unit (WKPU) Editorial and formatting changes Chapter 29 Periodic Interrupt Timer (PIT) Editorial and formatting changes PIT memory map: – Added page number links for Timer Channel registers – Changed end address in last row: Was 0x01FF; is 0x3FFF PIT Module Control Register (PITMCR): Added field MDIS (bit 30) Chapter 30 System Timer Module (STM) Editorial and formatting changes Corrected description and table name for STM Channel Interrupt Register (STM_CIRn) Chapter 31 Cyclic Redundancy Check (CRC) Editorial and formatting changes CRC computation flow: Replaced “context = CRC_CNTX_NUM” with “context = n” Improved the readability of “DMA-CRC Transmission Sequence” and “DMA-CRC Reception Sequence” figures Chapter 32 Boot Assist Module (BAM) Editorial and formatting changes Updated password validation procedure Removed MC_ prefixes from module names BAM memory organization: Added “Parameter” as header to lefthand column Updated Section 32.5.1 Entering boot modes to remove references to FlexRay and to improve description of autobaud scan Section 32.5.2 Reset Configuration Half Word Source (RCHW): Updated bitmap and description SPC560P44Lx, SPC560P50Lx Flash partitioning and RCHW search: Updated application boot information BAM resources: Replaced “CAN_A, LINFlex_A” with “CAN_0, LINFlex_0” Section Download 64-bit password and password check: Corrected register names, edited content and provided examples to improve understanding of password swapping Chapter 33 Voltage Regulators and Power Supplies Editorial and formatting changes Removed MC_ prefix from module names Updated base addresses for registers Section 33.1 Voltage regulator: Replaced “The internal voltage regulator requires an external capacitance (CREG)” with “The internal voltage regulator requires an external ballast transistor and (external) capacitance (CREG)” Section 33.1.2 Low Voltage Detectors (LVD) and Power On Reset (POR): – Replaced “Three types” with “Five types” in first sentence – Amended description of main LVDs and description of trimming bits – Removed sentence “The other LVD_DIG is placed in the standby domain and senses the standby 1.2 V supply level notifying that the 1.2 V output is stable.”

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Section 33.2 Power supply strategy: – Removed footnote concerning configuring regulator in bypass mode to provide VDD_LV externally during production test – Updated description of bullet “LV—Low voltage internal power supply” – Removed bullet “BV—High voltage supply for voltage regulator ballast” and associated description Section Voltage Regulator Control Register (VREG_CTL): – Changed reset values of register bits 23 and 27 – Removed field ‘MONITOR_MASK’ Section 33.1.4.2 Voltage Regulator Status register (VREG_STATUS): – Changed reset values of register bits 23 and 27 – Removed field ‘MONITOR_STATUS’ Chapter 34 IEEE 1149.1 Test Access Port Controller (JTAGC) Editorial and formatting changes Removed all references to JCOMP (which is not implemented on this device) Replaced instances of “e200z0 Reference Manual” with “core reference manual” Section 34.4 Features: Removed sentence ‘The boundary scan register length is 464 bits’ Section 34.6 External signal description: Replaced “The JTAGC consists of five signals” with “The JTAGC consists of four signals” Section 34.8.4 JTAGC instructions: Removed table “JTAG Instructions for silicon cut1” and associated introduction Chapter 35 Nexus Development Interface (NDI) Editorial and formatting changes Removed references to JCOMP Removed redundant sentence in Reduced-port mode Section 35.5 External signal description: Added note that NDI signals not available in 100-pin package Section 35.6.1 Nexus debug interface registers: – Harmonized register names – Removed CSC register EVTI generated break request: Removed sentence referencing Shared Nexus Control Register Changed MDO widths to FPM=4 MDO, RPM=2 MDO Changed MSEO bus direction from [0:1] to [1:0] Appendix A: Registers Under Protection Editorial changes Changed address format from XXXXXXXX to 0xXXXX_XXXX Registers under protection table: – Changed three Code Flash register names: — was BIU0; is PFCR0 — was BIU1; is PFCR1 — was BIU2; is PFAPR – Removed all rows DSPI_DSICR and DSPI_DSICR1 – Removed Module Base column – Updated number of registers to protect per module PMU module: Changed register name from CTL to VREG_CTL

Table 610. Document revision history(Continued)

Date	Revision	Changes
22-Feb-2010	3 (cont.)	Appendix B: Memory Map Editorial changes Updated register data Removed MC_ prefix from module names Detailed Memory Map: – Added missing size for THRHLR3 – Replaced “Base + 0x064” with “Base + 0x0064” in ECSM registers reserved row Throughout whole document: Replaced all IRCOSC with IRC. Replaced all IRCOSC_CLK with IRC_CLK. Removed all MC_ prefix from whole document. Replaced all PLL_*n* with FMPLL_*n* . Replaced all XOSC0 with XOSC. Removed MC_ prefixes: – was MC_ME; is ME. – was MC_RGM; is RGM. – was MC_CGM; is CGM. – was MC_PCU; is PCU.
20-Apr-2011	4	Added a chapter for the temperature sensor (TSENS) “Preface” chapter, entirely rewrote “Introduction” chapter Changed the title of this chapter from “Overview” to “Introduction” Renamed “Chipset overview” section to “The SPC560P44Lx, SPC560P50Lx microcontroller family” In the “The SPC560P44Lx, SPC560P50Lx microcontroller family”: – Rewrote the paragraph regarding the safaty requirements. – Replaced “electrical power steering” with “electric power steering” “SPC560P44Lx, SPC560P50Lx device comparison” table: removed SPC560P40 column “SPC560P44Lx, SPC560P50Lx device configuration differences” table: added “SRAM“, “FlexCAN”, and “DSPI” rows “SPC560P44Lx, SPC560P50Lx block diagram”: – Added the following blocks: “Nexus 2+“, “CRC“, “SSCM“, “MC_RGM“, “MC_CGM“, “MC_ME“, and “WKPU“ – “Flash memory” block split into 2 blocks: code flash memory, and data flash memory – Updated “ADC” block – Chip-level features” section: — Updated information about core and memory organization — Changed “64 MHz, single issue“ to “Up to 64 MHz, single issue“ – Made arrow going from peripheral bridge to crossbar switch bidirectional “Flash memory” section: – Updated section title (was: On-chip flash memory with ECC) – Updated information about “Read page sizes” – Updated information about ECC – Updated information about “Typical flash memory access time” – Changed “incurring 3 wait-states“ to “incurring 2 wait-states“

Table 610. Document revision history(Continued)

Date	Revision	Changes
20-Apr-2011	4 cont'd	"Static random access memory (SRAM)" section: – Updated section title (was: On-chip SRAM with ECC) – Changed "40 KB general purpose RAM" to "Up to 40 KB general purpose RAM" "Boot and censorship" and "Boot Assist Module (BAM)" sections: removed FlexRay from serial link boot options "Interrupt controller (INTC)" section: added information about the number of interrupt sources the INTC can handle Added "System status and configuration module (SSCM)" section "Frequency-modulated phase-locked loop (FMPLL)" section: – Updated section title (was "Frequency modulated PLL (FMPLL)") – Added information about "Maximum output frequency" "Periodic interrupt timer (PIT)" section: changed "As many as 4 general purpose" to "4 general purpose" "System timer module (STM)" section: changed "32-bit up counter" to "One 32-bit up counter" Updated "Error correction status module (ECSM)" section Added "Peripheral bridge (PBRIDGE)" section "Controller area network (FlexCAN)" section: updated section title (was: "CAN (FlexCAN)") "Pulse width modulator (FlexPWM)" section: – Updated section title (was: "FlexPWM") – Added information about: 2 falt inputs, and capture capability for PWMA/B/X Added "Cyclic redundancy check (CRC)" section Updated the "Developer environment" section. "Signal Description" chapter Pinout figures: – Renamed "$V_{DD\_LV\_PLL}$" and "$V_{SS\_LV\_PLL}$" supply pins with respectively "$V_{DD\_LV\_COR3}$" and "$V_{SS\_LV\_COR3}$". – Removed alternate functions from pinouts In the "Pin muxing" section, added, B[4] and B[5] rows In the "144-pin LQFP pinout – Full featured configuration (top view)" figure: – Pin 89: B[4] is now TDO – Pin 86: B[5] is now TDI In the "100-pin LQFP pinout – Full featured configuration (top view)" figure: – Pin 61: B[4] is now TDO – Pin 58: B[5] is now TDI In the "System pins" table: – Added MDO[0] row; – Removed TDI and TDO rows; – Updated peripheral nomenclature; – was $V_{DD\_LV\_PLL}$; is $V_{DD\_LV\_COR3}$ – was $V_{SS\_LV\_PLL}$; is $V_{SS\_LV\_COR3}$ – Updated descriptions of EXTAL and XTAL In the "Clock Description" section, replaced all occurences of XTALOUT with EXTAL.

Table 610. Document revision history(Continued)

Date	Revision	Changes
20-Apr-2011	4 cont'd	In the “Interrupt controller” section: – Added back MC_ prefixes: — was ME; is MC_ME — was RGM; is MC_RGM – In the “Interrupt vector table”, added the missing values in the Offset column. – In the “INTC memory map” table, removed access and reset columns. “System Integration Unit Lite (SIUL)” chapter In the "MCU ID Register # (MIDR2)" section: – replaced reset value of EE:x is now 0 – renamed the bit field "FR" with "FF" In the "MIDR2 field description" table, renamed the bit field "FR" with "FF" and replaced its description. In the “MCU ID Register #2 (MIDR2)” section, deleted “0011: 128 KB” in the FLASH_SIZE_1 description “e200z0 and e200z0h Core” chapter In the “Core registers and programmer’s model” section, removed note concerning the register numbering. “Error Correction Status Module (ECSM)” chapter Replaced all ocurences of “AXBS” with “XBAR” In the “Spp_Ips_Reg_Protection block diagram” figure, replaced “AIPS_LITE” with “PBRIDGE” “Flash Memory” chapter In the “Memory map” section, added a “caution” and a “note” concerning the register management. “DMA Channel Mux (DMA_MUX)” chapter In the “DMA_MUX memory map” table, removed access and reset columns. In the “DMA Mux block diagram”, added always-on slot labels “Always #1” and “Always #9” In the “DMA mux triggered channels diagram” figure, replaced “Always #4” with “Always #9” In the “DMA mux channel 4–15 block diagram” figure, added “Always #1” and “Always #9” arrow labels “Deserial Serial Peripheral Interface (DSPI)” chapter In the “DSPI memory map” table, removed access and reset columns. In the “DSPI block diagram” figure, replace arrow labels from “1” to “3”. In the DSPIx_MCR.CONT_SCKE filed description, added a note. In the “Continuous Serial Communications clock” section, added a note. In the “Continuous Selection Format” section, added a note. In the “Functional description” section, replaced “Each DSPI has six peripheral chip select” with “Each DSPI has four peripheral chip select”

Table 610. Document revision history(Continued)

Date	Revision	Changes
20-Apr-2011	4 cont'd	"LIN Controller (LINFlex)" chapter In the "IFER field descriptions" table, switched "activated" and "deactivated" in order to match with "IFER[FACT] configuration" table. In the "UART mode" section, in the "9-bit frames" subsection, changed "sum of the 7 data bits" to "sum of the 8 data bits". "FLexCAN" chapter In the "Module Configuration Register (MCR)" section, changed DOZE field to reserved bit In the "Freeze mode", replaced "(Disable, Doze, Stop)" with "(Halt, Stop)" in first paragraph In the "Module Configuration Register (MCR)" figure, changed reset value for field MAXMB[5:0] from 000000 to 001111 In the "Freeze mode" section: Deleted "(Halt, Stop)" in first paragraph In the "FlexCAN module memory map" table, removed access and reset columns. In the "Message Buffer lock mechanism" section, updated a footnote "Analog-to-Digital Converter (ADC)" chapter Editorial changes. Replaced ADCDig with ADC, rewriting content as necessary. In the PDEDR[PDED] field description, added "The delay is to allow time for the ADC power supply to settle before commencing conversions.". Replaced "ipg_clk" and "system clock" with "MC_PLL_CLK" Updated MCR[WLSIDE] bit description. Updated CDR register Changed the access type of DSDR in "read/write" Updated the DSD description in the DSDR field description table Updated following registers: CEOCFR, CIMR, WTISR, DMAR, PSR, NCMR, JCMR, CDR, CWSEL, CWENR, and AWORR Inserted "CTU triggered conversion" in the conversion list of "Functional description" section Replaced generic "system clock" with "peripheral set 3 clock" "ADC sampling and conversion timing" section, added information about "ADC_1" Moved CWSEL, CWENR and AWORR register to "Watchdog register" section CTR register: Inserted a footnote about OFFSHIFT field stating: "available only for CTR0" In the "Conversion timing registers CTR" section, removed footnote. in the Device-specific features section, removed "internal standard channels" from "Sampling and conversion time register" bullet In the "Mask registers" section changed the number of channel from 96 to 16. In the "Channel Data Register (CDR)" section changed the number of channel from 95 to 15

Table 610. Document revision history(Continued)

Date	Revision	Changes
20-Apr-2011	4 cont'd	"FlexPWM" chapter "Capture Value 0 Cycle register (CVAL0CYC)" and "Capture Value 1 Cycle register (CVAL1CYC)" use only bits [13:15] instead of bits[12:15] Replaced instances of WAIT mode with WAIT/HALT mode SPC560P44Lx, SPC560P50Lx device-specific changes: DMAEN field descriptions: Removed reference to bits CAxDE and CBxDE from FAND field description In the "Deadtime Count registers (DTCNT0, DTCNT1)" section, changed the reset value from 0x0FFF to 0x07FF. "Functional Safety" chapter: In the " Register protection memory map" table, removed access and reset columns. In the "SWT memory map" table, removed access and reset columns per new agreement. In the "SWT Control Register (SWT_CR)" table, changed the reset value from 0x4000_011B to 0xFF00_011B. In the "SWT memory map" table, inserted the SWT_1 offset value "Fault Collection Unit (FCU)" chapter: In the "FCU memory map" table, removed access and reset columns per new agreement. "Test mode" section: Removed sentence referencing software-triggered faults not being supported by the FCU_FFGR Register summary: In FCU_FFR, changed field SRF1 to read-only with value 0 In FCU_FFFR, changed field FRSRF1 to read-only with value 0 "Fault Flag Register (FCU_FFR)" section: Removed sentence referencing clearing the software fault flag SRF1; changed field SRF1 to read-only with value 0 Hardware/software fault description: Marked SRF1 as "Not used" "Frozen Fault Flag Register (FCU_FFFR)" section: Changed field FRSRF1 to read-only with value 0 FCU_FER field descriptions: Added note that field ESF1 not implemented FCU_TER field descriptions: Added note that field TESF1 not implemented "Microcontroller State Register (FCU_MCSR)" section: Updated bit values "Frozen MC State Register (FCU_FMCSR)" section: Updated bit values SRF1: was "not used", is now "FCU Software-triggered error". Switched the module of SFR2 with that of SFR4. Updated all registers according the "Hardware/software fault description" table. "Wakeup Unit (WKPU)" chapter In the "External Signal description" section, added note about Wakeup pin termination.

Table 610. Document revision history(Continued)

Date	Revision	Changes
20-Apr-2011	4 cont'd	"Boot Assist Module (BAM)" chapter: "Boot modes": Minor editorial changes Boot mode selection: Added Autobaud Scan boot mode "Reset Configuration Half Word (RCHW)" section: Removed the word "Source" from title System clock frequency related to external clock frequency: Minor editorial changes In the "Hardware configuration to select boot mode" table: rewrote the column title from "ABS[1:0]" to ABS[2,0] added footnote stating: ABS[1] is "don't care" "Voltage Regulators and Power Supplies" chapter In the "Power supply strategy" section: - Removed the BV domain - Update the "supply domains" "IEEE 1149.1 Test Access Port Controller (JTAGC)" chapter: In the "TAP sharing mode" section: - Removed "ACCESS_AUX_TAP_eTPU". - Replaced ACCESS_AUX_TAP_DMA with ACCESS_AUX_TAP_NPC. In the "Device identification register" section: - DC Design Center was 2D, now is 2B. - PIN Part Identification Number was 0x220, now is 0x221. "Nexus Development Interface (NDI)" chapter: Replaced all occurences of "Nexus2+" with "Nexus 2+" Appendix B Removed both DSDR registers.

Table 610. Document revision history(Continued)

Date	Revision	Changes
13-May-2013	5	Chapter 2, "SPC560P44Lx, SPC560P50Lx memory map" Added link to Appendix B, "Memory Map". Table 4 (Memory map), added footnote about “On-chip peripherals” group. Chapter 3, "Signal Description" Table 7 (Pin muxing), changed the description in the column "I/O direction" from "I/O" to "O" for the following port pins: A[10] with function B[0] A[11] with function A[0] A[11] with function A[2] A[12] with function A[2] A[12] with function B[2] A[13] with function B[2] C[7] with function A[1] C[10] with function A[3] C[15] with function A[1] D[0] with function B[1] D[10] with function A[0] D[11] with function B[0] D[13] with function A[1] D[14] with function B[1] Chapter 4, Clock Description Table 11 (Crystal oscillator truth table), removed in normal mode with XOSC enabled the configuration of oscillator providing external clock. Previous errata ERR003324 integrated into the reference manual: Section 4.7, IRC 16 MHz internal RC oscillator (RC_CTL), reworded register description Figure 9 (SPC560P44Lx, SPC560P50Lx system clock generation), replaced “FMPLL_1D1_CLK” with “FMPLL_1_D1_CLK”. Table 19 (CMU memory map), replaced “(CMU_0_FDISP)” with “(CMU_0_FDR)” Chapter 5, Clock Generation Module (CGM) Through whole chapter: Replaced all occurrences of “FMPLL_1 – 120 MHz” with “FMPLL_1” Replaced all occurrences of “FMPLL_1 – 80 MHz” with “FMPLL_1_D1” Chapter 6, "Mode Entry Module (ME)" Renamed Chapter 6, "Mode Entry Module (ME)" (was “Operating Modes”) Renamed Section 6.1, Introduction (was “Mode Entry Module (ME)”) Chapter 7, Power Control Unit (PCU) Table 60 (PCU memory map), updated offset value for Reserved areas Removed PCU_PCONF1 and PCU_PCONF2...15 registers Chapter 8, Reset Generation Module (RGM) Removed all references to RGM_DEAR register. Table 66 (RGM memory map), updated offset value for Reserved areas Table 70 (RGM_DERD field descriptions), removed D_COMP bit field Section 8.5.1.3, Functional Event Reset Disable register (RGM_FERD), replaced a cross-reference to RGM_DEAR with one to RGM_FEAR. changed D_FLASH as read-only

Table 610. Document revision history(Continued)

Date	Revision	Changes
13-May-2013	5 cont'd	Chapter 11, "System Integration Unit Lite (SIUL)" Added newTable 111 (PCR bit implementation by pad type) . Figure 107 (System Integration Unit Lite block diagram), replaced 106 wth 108. Updated Table 113 (Pad selection) Table 101 (MIDR1 field descriptions), updated MIDR flash size to 384K. Chapter 12, e200z0 and e200z0h Core Added Figure 131 (e200z0 Supervisor mode programmer's model) Chapter 15, "Error Correction Status Module (ECSM)" Table 123 (ECSM registers) Added PLAMC and PLASC registers Added "Size (bits)" column. Section 15.1, Introduction, removed " It also provides with register access protection for the following slave modules: INTC, ECSM, STM, and SWT" sentence. Section 15.3, Features, removed "Capability to restrict register access to supervisor mode to selected on-platform slave devices: INTC, ECSM, STM, and SWT" bullet. Updated MUDCR register, Removed MUDCR[31] and added MUDCR[30] Updated contets of Table 163 (Platform RAM syndrome mapping for single-bit correctable errors). Removed "ECSM_reg_protection" section. Chapter 16, Internal Static RAM (SRAM) Table 146 (SRAM memory map), added a column Chapter 17, "Flash Memory" Table 153 (544 KB code Flash module sectorization), added footnote in the B0F7 entry Table 167 (PFCR0 field descriptions), updated reset value. Figure 174 (Platform Flash Configuration Register 1 (PFCR1)) updated reset value Table 168 (PFCR1 field descriptions), updated reset value. Table 167 (PFCR0 field descriptions), in the BK0_RWSC, updated frequency range. Table 168 (PFCR1 field descriptions), in the BK0_RWSC, updated frequency range. Figure 188 (Non-Volatile User Options register (NVUSRO)), removed OSCILLATOR_MARGIN bit. Table 182 (NVUSRO field descriptions), removed OSCILLATOR_MARGIN field and updated UOx bit fields Chapter 18, Enhanced Direct Memory Access (eDMA) Through whole chapter, replace all occurrences of EDMA_CERR to EDMA_CER Section 18.5.2.2, eDMA Error Status Register (EDMA_ESR), changed GPE bit field to "reserved". Renamed Section 18.5.2.10, eDMA Clear Error Register (EDMA_CER) (was eDMA Clear Error Register (EDMA_CERR) Through whole chapter, replace all occurrences of EDMA_CERR to EDMA_CER Updated Section 18.7.2, DMA programming errors in accordance with GPE is no longer available Chapter 19, DMA Channel Mux (DMA_MUX) Table 211 (DMA_MUX memory map), removed rows concerning CHCONFIG16 to CHCONFIG31

Table 610. Document revision history(Continued)

Date	Revision	Changes
13-May-2013	5 cont'd	Chapter 20, FlexRay Communication Controller (FlexRay) Previous errata ERR002423 integrated into the reference manual: Table 292 (Channel assignment description), removed the text and added 'Reserved' instead. Previous errata ERR002421 integrated into the reference manual: Section 20.6.6.2.3.1, Application transitions and Section 20.6.6.3.1.1, Application transitions, added paragraph: "If the communication controller is started as a non-coldstart node..." Chapter 22, LIN Controller (LINFlex) Figure 398 (LIN status register (LINSR)), changed LINS access from read/write to write only Figure 401 (UART mode status register (UARTSR)), made RPS bit as read-only Figure 403 (LIN output compare register (LINOCR)), changed note from LINTCSR[LTOM] = 1 to LINTCSR[LTOM] = 0 Figure 415 (Identifier filter control register (IFCR2n)), changed ID access from read/write "w1c' to read/write only in initialization mode Figure 416 (Identifier filter control register (IFCR2n + 1)), changed ID access from read/write "w1c' to read/write only in initialization mode Updated Section 22.7.1.17, Identifier filter enable register (IFER) , Figure 412 (Identifier filter enable register (IFER)), changed FACT field from 8 bit to 16 bit Updated Table 384 (IFER field descriptions) Removed "IFER[FACT] configuration" table Section 22.7.1.20, Identifier filter control register (IFCR2n), replaced note "This register can be written in Initialization mode only." with "Register bit can be read in any mode, written only in Initialization mode" Section 22.7.1.21, Identifier filter control register (IFCR2n + 1), replaced note "This register can be written in Initialization mode only." with "Register bit can be read in any mode, written only in Initialization mode" Added Section 22.8.2.1.7, Overrun Section 22.8.2.2.1, Data transmission (transceiver as publisher), changed BDAR register with BDR register Reworded Section 22.8.2.2.8, Overrun Section 22.8.2.3.1, Filter mode, changed sentence "eight IFCR registers" with "sixteen IFCR registers" Section 22.8.2.3.2, Identifier filter mode configuration, changed sentence "the filter must first be deactivated by programming IFER[FACT] = 0" with "the filter must first be activated by programming IFER[FACT] = 1" Section 22.8.2.4.1, Automatic resynchronization method now is a section Section 22.8.3.1, LIN timeout mode, changed sentence "Setting the LTOM bit" with "Resetting the LTOM bit" Section 22.8.3.2, Output compare mode, changed sentence "Programming LINTCSR[LTOM] = 0 enables the output compare mode" with "Programming LINTCSR[LTOM] = 1 enables the output compare mode" Chapter 23, FlexCAN Previous errata ERR002685 integrated into the reference manual: Made editorial changes in Section 23.1.3, Modes of operation descriptions ("Modes of Operation" and "Modes of Operation Details" sections). Previous errata ERR002656 integrated into the reference manual: Made the following changes: Editorial changes in Section 23.4.2, Transmit process. Added a note to Section 23.4.6.1, Transmission abort mechanism.

Table 610. Document revision history(Continued)

Date	Revision	Changes
13-May-2013	5 cont'd	Previous errata ERR002360 integrated into the reference manual: Added Section 23.4.7.1, Precautions when using Global Mask and Individual Mask registers. Updated Section 23.3.3, Rx FIFO structure Table 405 (CTRL field descriptions), fixed the right sequence of: LPB, TWRN_MSK and RWRN_MSK. Chapter 24, Analog-to-Digital Converter (ADC) Section 24.1.1, Device-specific features, removed "ADC_1: channel 15 dedicated for the temperature sensor" feature Section 24.3.3, ADC sampling and conversion timing, removed "ADC_1" section. Renamed section "Analog watchdog pulse width modulation bus" with Section 24.3.5.2, Analog watchdog functionality, and reworded the section Section 24.3.7, Interrupts, removed sentence "Interrupts can be individually enabled on a channel by channel basis by programming the CIMR (Channel Interrupt Mask Register)." Section 24.3.8, Power-down mode, replaced sentence: "If the CTU is enabled and the CSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set." with "If the CTU is enabled and the MSR[CTUSTART] bit is '1', then the MCR[PWDN] bit cannot be set." Previous errata ERR003032 integrated into the reference manual: Added Note in Section 24.3.9, Auto-clock-off mode. Table 425 (ADC digital registers), removed CIM0 register, set address as reserved and added footnote concerning CDR15. Removed "Channel Interrupt Mask Register (CIMR[0])" section Figure 9 (SPC560P44Lx, SPC560P50Lx system clock generation),updated note to "FlexRay protocol clock does not support IRC and OSC as clock source". Chapter 25, Cross Triggering Unit (CTU) Figure 486 (Cross triggering unit error flag register (CTUEFR)), changed the access permission of all bit from "R/W" ro "W1C" Figure 488 (Cross triggering unit interrupt/DMA register (CTUIR)), changed the access permission of all bit from "R" ro "R/W" Figure 490 (Cross triggering unit control register (CTUCR)), renamed FE to DFE. Table 468 (CTUCR field descriptions), added notes for DFE, CGRE, GRE, and TGSISR_RE bit fileds. Updated Section 25.4.2, ADC commands list format Chapter 26, "FlexPWM" Updated Figure 494 (PWM submodule block diagram) Added Table 483 (DTCNT0 field descriptions) and Table 484 (DTCNT1 field descriptions) Figure 549 (Output logic section), added MASKA and MASKB bits Chapter 27, eTimer Table 506 (CTRL1 field descriptions), updated description of ONCE field. Section 27.6.2.13, Comparator Load register 1 (CMPLD1), modified the description of CMPLD field to read, "Specifies the preload value for the COMP1 register" instead of "Specifies the preload value for the COMP2 register".

Table 610. Document revision history(Continued)

Date	Revision	Changes
13-May-2013	5 cont'd	Chapter 28, Functional Safety Table 525 (SWT memory map), removed SWT_1 Figure 602 (SWT Counter Output register (SWT_CO)), changed the access permission from read/write to read only Table 528 (SWT_TO field descriptions), updated fied description, was (SWT_CR.WENSWT_CR.=0) is (SWT_CR[WEN]=0). Section 28.3.5.1, SWT Control Register (SWT_CR), replaced "MAP1 = 1 (only data bus access allowed)" with "MAPn = 1 (all master access allowed)" Chapter 33, Cyclic Redundancy Check (CRC) Table 564 (CRC_CFG field descriptions), adde note for POLYG bit field. Chapter 34, Boot Assist Module (BAM) Table 569 (Hardware configuration to select boot mode), Rewrote the column title from "ABS[2,0]" to ABS[1:0] removed footnote stating: ABS[1] is "don't care" Added footnote, "During reset the boot configuration pins are weak pull down." Section 34.5.1, Entering boot modes, Added PAD A[2] note Added the Boot Configuration Pins. Added Section 34.5.2, SPC560P44Lx, SPC560P50Lx boot pins. Updated Section 34.5.6.1, Configuration with new Figure 656 (BAM Autoscan code flow). Section 34.6.1.3, Boot from FlexCAN with autobaud enabled , Updated "UART boot mode" with "FlexCAN boot mode". Added Section 34.7, Censorship Chapter 35, Voltage Regulators and Power Supplies Updated Section 35.1.1, High Power or Main Regulator (HPREG) Chapter 36, IEEE 1149.1 Test Access Port Controller (JTAGC) Updated Section 36.9, e200z0 OnCE controller to remove reference of Nexus2+ configuration registers. Updated Table 591, e200z0 OnCE register addressing, Nexus 2+ Access was replaced with "Reserved". Section 36.7.4, Boundary scan register replaced " with "The size of the boundary scan register is 464 bits." with "The size of the boundary scan register and bit ordering is device-dependent and can be found in the device BSDL file." Chapter 37, Nexus Development Interface (NDI) Added Figure 673 (NDI functional block diagram). Section 37.6.2.1, Nexus Device ID (DID) register, – DC Design Center was 2D, now is 2B. – PIN Part Identification Number was 0x220, now is 0x221. Section 37.7.6.1, EVTI generated break request, added note. Appendix A, "Registers Under Protection" Deleted RGM_DEAR entry. Appendix B, "Memory Map" Aligned contents to current versions of memory tables contained on each chapter.

Table 610. Document revision history(Continued)

Date	Revision	Changes
06-Jun-2013	6	FlexCAN chapter: Section 23.1, “Introduction”, removed paragraph referring to both FlexCAN_0 and FlexCAN_1. Table 436 (FlexCAN module memory map), fixed offset values removing 0xFBEC_0000 and 0xFBEC_0000 (CAN_1) Table 447 (CTRL field descriptions) deleted note about AUX CLK SEL1 in the CLK_SRC description
18-Sep-2013	7	Updated Disclaimer.
10-Jul-2015	8	*Chapter 3: Signal Description*: In *Table 7: System pins*, changed the direction of TMS pin from “Bidirectional” to “Input only” and added footnote for F[10] pin. In *Table 8: Pin muxing*, changed GPIO[92] to GPIO[93] in column “Functions” for F[13] port pin. *Chapter 4: Clock Description*: Changed PMU to VREG in *Figure 10: SPC560P44Lx, SPC560P50Lx system clock distribution Part B*. *Chapter 9: Interrupt Controller (INTC)*: In *Table 85: Interrupt vector table* setted IRQ # 67 and IRQ # 210 as Reserved *Chapter 10: System Status and Configuration Module (SSCM)*: Updated the reset value of MEMCONFIG register in *Figure 101: System memory configuration (MEMCONFIG) register*. Updated *Table 90: MEMCONFIG field descriptions*. *Chapter 11: System Integration Unit Lite (SIUL)*: Updated the description of MAXCNTx field in *Table 118: IFMC[0:31] field descriptions*. *Chapter 12: e200z0 and e200z0h Core*: Removed “Testability” feature from *Section 12.2: Features*. *Chapter 15: Error Correction Status Module (ECSM)*: Updated the reset value of IPS register in *Figure 137: IPS Module Configuration (IMC) register* to 0003_E000. *Chapter 17: Flash Memory*: Added *Section 17.3.4.2.1.1: Unique Serial Number*. In *Table 168: PFCR1 field descriptions*, added a note in B1_P1_BFE field of PFCR1 register. In *Section 17.3.7.8: User Test 0 register (UT0)*, updated the access of UTE field from w1c to R/W in *Figure 173: User Test 0 register (UT0)* and updated the description of UTE field in *Table 170: UT0 field descriptions*. *Chapter 19: DMA Channel Mux (DMA_MUX)*: Changed CTU FIFO [1:4] to CTU FIFO [0:3] in column "DMA requesting module" of *Table 214: DMA channel mapping*.

Table 610. Document revision history(Continued)

Date	Revision	Changes
10-Jul-2015	8 (cont.)	*Chapter 23: FlexCAN* Updated description of MAXMB field in *Table 402: MCR field descriptions*. In *Section 23.1.2: FlexCAN module features* updated the sentence “Low power modes, with programmable wake up” with the sentence “Low power modes”. In *Figure 423: Module Configuration Register (MCR)* setted the bit 5 as only read 0 bit. In *Table 402: MCR field descriptions* removed the row related to field 5 and updated note into the description of MAXMB field. In *Figure 430: Error and Status Register (ESR)* setted the bit 31 as only read 0 bit. In *Table 409: Error and Status Register (ESR) field description* removed the row related to field 31. *Chapter 24: Analog-to-Digital Converter (ADC)*: Updated *Section 24.3.3: ADC sampling and conversion timing*, *Table 422: Max/Min ADC_clk frequency and related configuration settings at 5 V / 3.3 V*, and *Table 423: ADC sampling and conversion timing at 5 V / 3.3 V*. *Chapter 26: FlexPWM*: Updated the reset values of CTRL1 register in *Figure 495: Control 1 Register (CTRL1)* to 0400. Updated the reset values of FSTS register in *Figure 522: Fault Status Register (FSTS)* to 0F0F. *Chapter 27: eTimer*: Updated the description of ICF1 and ICF2 fields in *Table 511: STS field descriptions*. *Chapter 29: Fault Collection Unit (FCU)*: In *Table 535: Register summary*, changed field SRF1 to read-only with value 0 for FCU_FFR register and changed field FRSRF1 to read-only with value 0 for FCU_FFFR register. In *Section 29.2.3.2: Fault Flag Register (FCU_FFR)*, changed field SRF1 to read-only with value 0 and updated *Table 537: FCU_FFR field descriptions* accordingly. In *Table 538: Hardware/software fault description*, marked SRF1 as “Not used” and replaced PMU module with VREG module. In *Section 29.2.3.3: Frozen Fault Flag Register (FCU_FFFR)*, changed field SRF1 to read-only with value 0 and updated *Table 539: FCU_FFFR field descriptions* accordingly. In *Section 29.2.3.4: Fake Fault Generation Register (FCU_FFGR)*, removed SRF1 from the last sentence of register description paragraph. *Chapter 33: Cyclic Redundancy Check (CRC)*: Updated the reset value of CRC Configuration register in *Figure 639: CRC Configuration Register (CRC_CFG)* to 0000_0004. *Appendix A: Registers Under Protection*: In *Table 608: Registers under protection* replaced occurrences of ADC with ADC 0 and ADC 1 and replaced CANx_IMASK_1 with CANx_IMASK1.

Table 610. Document revision history(Continued)

Date	Revision	Changes
10-Jul-2015	8 (cont.)	*Appendix A: Registers Under Protection*: In *Table 608: Registers under protection*: – Removed PCONF2, CANx_MASK2, CFGR and TRIMR registers. – Removed “PIT_RTI” from the name of all PIT registers. – Changed the name of following registers for eTimer0 and eTimer 1: Changed CH0_CTRL to CH0_CTRL1 – Changed CH1_CTRL to CH1_CTRL1 – Changed CH2_CTRL to CH2_CTRL1 – Changed CH3_CTRL to CH3_CTRL1 – Changed CH4_CTRL to CH4_CTRL1 – Changed CH5_CTRL to CH5_CTRL1 Changed the name of following registers: – Changed SUB0_CTRL to SUB0_CTRL1 – Changed SUB1_CTRL to SUB1_CTRL1 – Changed SUB2_CTRL to SUB2_CTRL1 – Changed SUB3_CTRL to SUB3_CTRL1 – Changed IFCP to IFCPR – Changed CANx_IMASK to CANx_IMASK_1, for FlexCAN and Safety port. – Updated entries for ADC 0 and ADC 1 modules. – Updated entries from MC_CGM module. – Changed PMU module to VREG module for VREG_CTL register. – Added a note for PCTL registers and removed entry for “ME_PCTL[84.87]” registers. From FlexPWM, removed SUBx_CAPTCTRLA and SUBx_CAPTCTRLB register