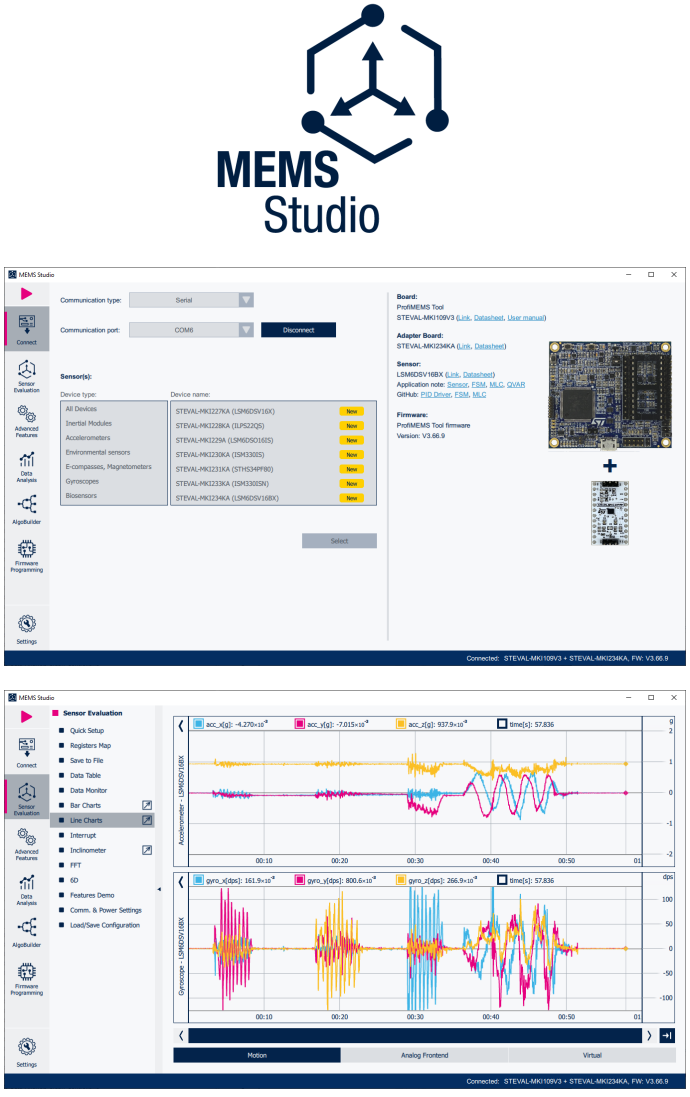
Getting started with MEMS Studio

Introduction

MEMS Studio is a complete desktop software solution designed to develop embedded AI features, evaluate embedded libraries, analyze data, and design no-code algorithms for the entire portfolio of MEMS sensors. This unique software solution offers a versatile development environment, enabling the evaluation and programming of all MEMS sensors, and launches a new generation of solutions to expand the functions of the well-established applications Unico-GUI, Unicleo-GUI, and AlgoBuilder.

MEMS Studio facilitates the process of implementing proof of concept using a graphical interface without writing code for STM32 microcontrollers. This solution allows configuring sensors and embedded AI (machine learning and neural networks), leveraging on a machine learning core (MLC), neural networks for the intelligent sensor processing unit (ISPU), and finite state machines (FSM). It reuses embedded software libraries, combines multiple functionalities in a single project, and visualizes data in real time using plot and display.

Figure 1. MEMS Studio application



1 Overview

MEMS Studio offers the following user experience:

- Sensor evaluation for motion, environmental, and infrared sensors in the MEMS portfolio
- Configuration and testing of in-sensor features such as the finite state machine (FSM), machine learning core (MLC), intelligent sensor processing unit (ISPU)
- Runtime and offline data analysis
- No-code graphical design of algorithms

The key features of the application include:

- Sensor configuration
  - Easy sensor configuration and evaluation
  - Access to the full sensor register map
  - Interrupt status monitoring
- Sensor data analysis
  - Runtime sensor data visualization charts (line charts, bar graphs, 3D plots, …)
  - Data logging to .csv file
  - Offline data visualization, data labeling, and editing
  - Fast Fourier transform (FFT) analysis of online and offline data
  - Spectrogram analysis of online and offline data
- Application development
  - Testing of the advanced embedded features (FIFO, pedometer, free fall, …) in the sensor
  - In-sensor AI design and programming for the finite state machine (FSM), machine learning core (MLC), and intelligent sensor processing unit (ISPU)
  - Embedded AutoML tool (automatic filter and feature selection) to simplify the MLC configuration process
  - Visualization and data logging of the output of the embedded software libraries
  - Development of no-code algorithms for data processing in STM32 microcontrollers
  - Analysis of pretrained neural network model, optimization, and conversion to c code
- Support for Windows, macOS, and Linux operating systems
- Network updates with automatic notification of new releases

2 Getting started

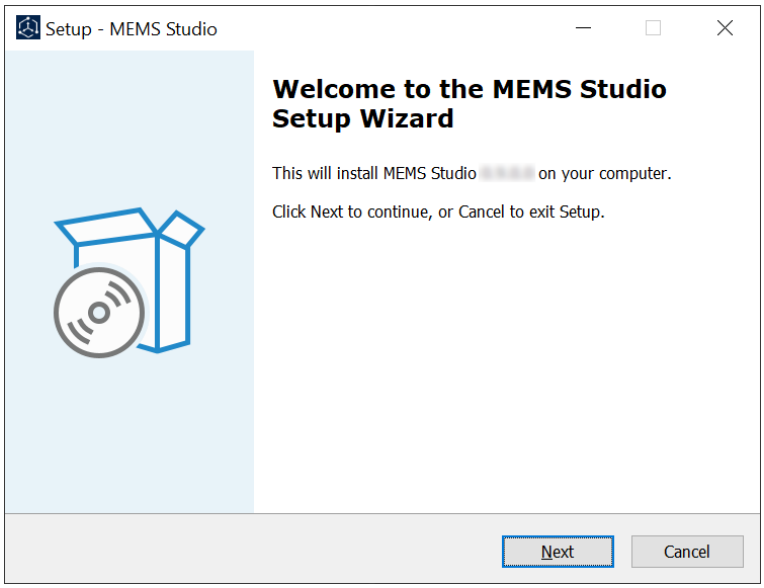
2.1 Installation

The MEMS Studio software has been designed to operate with Microsoft Windows platforms, Linux platforms, and macOS platforms.

Windows platforms

To install MEMS Studio, launch the [**mems-studio.exe**] installation file and follow the instructions that appear on the screen. When the software is installed, you can run it from: [**Start**]>[**STMicroelectronics**]>[**MEMS Studio**]

Figure 2. MEMS Studio Installer



Linux platforms

For Linux Debian-based distributions, a .deb package is provided.

On Ubuntu, you can install the .deb package by opening a terminal and writing the following command:

```
sudo dpkg -i mems-studio_VERSION_amd64.deb
```

If the installation fails due to a missing dependent library, it is necessary to install the missing library before proceeding with the MEMS Studio installation:

```
sudo apt-get install <missing_library_name>
```

```
sudo dpkg -i mems-studio_VERSION_amd64.deb
```

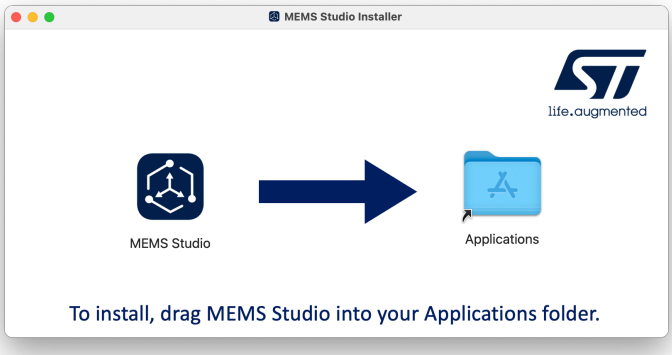
After the installation has been completed, the executable file (“mems-studio”) is stored in the /usr/bin folder. To run the MEMS Studio software, just click on the MEMS Studio icon in the Application launcher menu.

Note: The current version of MEMS Studio is designed to work on Ubuntu 18.04 LTS, although it should be possible to install and run it on newer versions of Ubuntu or other Debian-based distributions after having installed all the dependencies required.

macOS platforms

To install MEMS Studio, simply open the DMG package and drag MEMS Studio to your Applications folder.

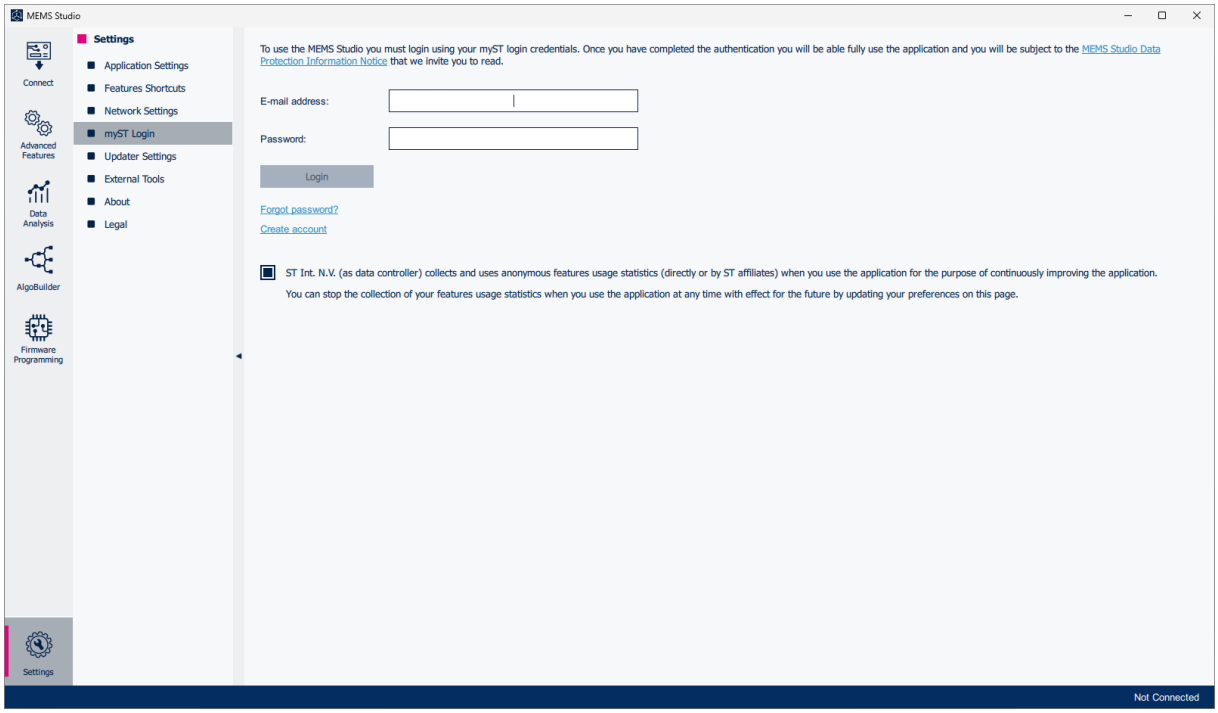
Figure 3. MEMS Studio Installer for macOS



2.2 myST Login

To use MEMS Studio, it is mandatory to enter your myST login credentials. By logging in with your myST credentials, you can fully utilize all the functionalities and services offered by MEMS Studio.

Figure 4. myST Login page



2.3 Application settings

Some application parameters can be adjusted in the [**Applications Settings**] in the [**Settings**] menu.

Application colors can be selected using [**Color theme**]. Available themes are Light, Dark, and Legacy.

[**Font size**] can be adjusted by the user in the application settings for good readability.

[**Plot grid in rolling mode**] can be set to be stationary or moving.

If [**Automatic registers reading**] is enabled, sensor registers are automatically read when entering [**Register map page**].

The user can select the units for graphs in the application for acceleration, angular rate, magnetic field, temperature, humidity, and pressure.

[**Allow multiple page undocking**] enables the possibility to undock more than one page at a time. This option requires more computer resources.

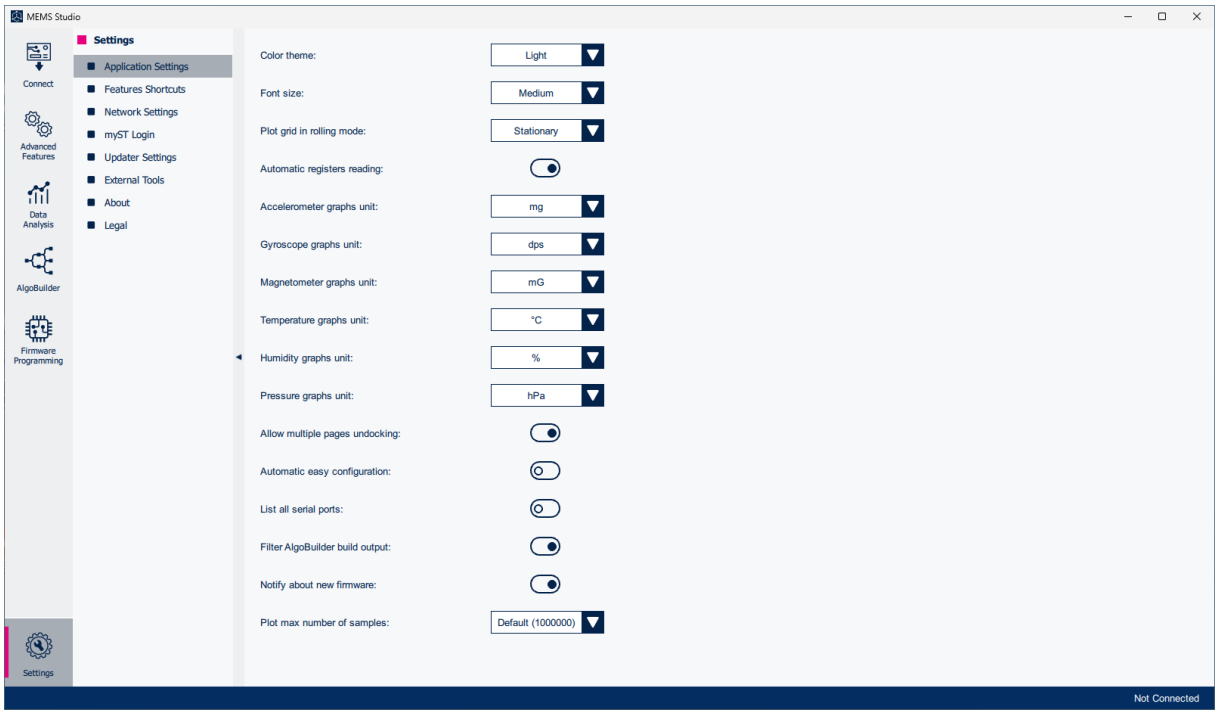
If [**Automatic easy configuration**] is enabled, the sensor is automatically configured after connection (when the sensor is not in power-down mode). This option is valid only for ProfiMEMS firmware.

If [**Filter AlgoBuilder build output**] is enabled, AlgoBuilder automatically filters outputs from the external compiler and makes them more readable in the console.

If [**Notify about new firmware**] is enabled, the application automatically checks if new firmware for the connected board is available and offers a firmware update.

The [**Plot max number of samples**] option enables setting the maximum number of samples that will be kept in the application buffers and displayed in the line charts. This setting also impacts the amount of RAM memory allocated by the application.

Figure 5. Application Settings



2.4 External Tools

MEMS Studio utilizes some external tools. The paths to these tools need to be set in [**External Tools**] in the [**Settings**] menu.

It is necessary to specify the path to at least one compiler in order to compile AlgoBuilder firmware. [**IAR embedded workbench path**], [**Keil uVision path**], or [**STM32CubeIDE path**] can be specified.

Firmware programing in the standalone page or programming from the AlgoBuilder page utilizes the STM32CubeProgramer CLI tool. [**STM32CubeProgrammer path**] must be specified in order to make the programming available.

[**ISPU Model Converter**] requires that the path to the [**ST Edge AI Core**] tool needs to be specified.

The path to the [**Netron**] tool needs to be specified in order to graphically visualize the neural network.

The path to [**ISPU toolchain**] needs to be specified in order to invoke ISPU sensor configuration generation directly from MEMS Studio.

The path to [**Make**] needs to be specified in order to invoke the compiler from the ISPU IDE.

Figure 6. External Tools

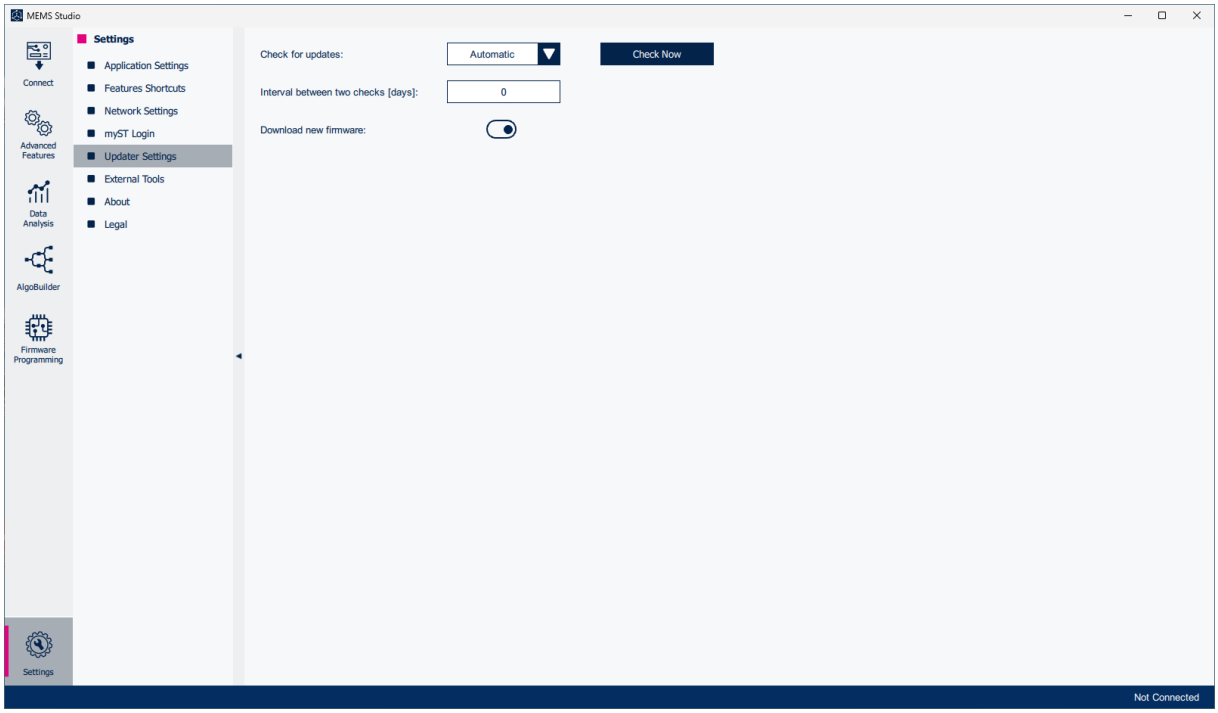
2.5 Updater Settings

The MEMS Studio application is able to check and notify if a new version is available. You can then decide whether to download and install the new version or not on the [**Updater Settings**] page. The network parameters have to be properly set in the [**Network Settings**] page. Both pages are in the [**Settings**] menu.

A check for a new version can be done manually by pressing the [**Check Now**] button or automatically during application startup. An interval for the automatic check can be set.

If the [**Download new firmware**] option is enabled, the application will check and download also new firmware for supported evaluation boards. The firmware package is downloaded separately and does not have any impact on the installed MEMS Studio version.

Figure 7. Updater Settings



2.6 Network Settings

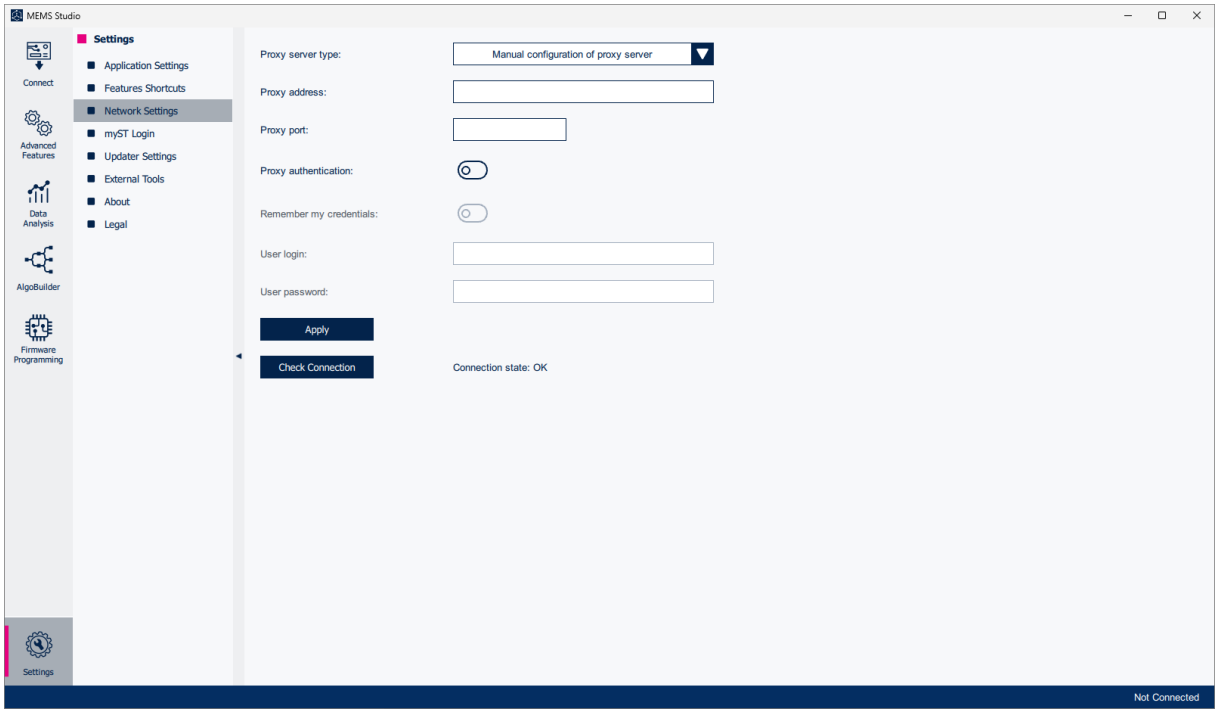
For proper MEMS Studio functionality, network settings must be correctly configured.

If a proxy server is used, corresponding parameters like the HTTP address and port must be set.

If the proxy server requires authentication, user credentials must be entered in the appropriate fields.

The [**Check Connection**] button can be used to check if the settings are done correctly and the server can be reached.

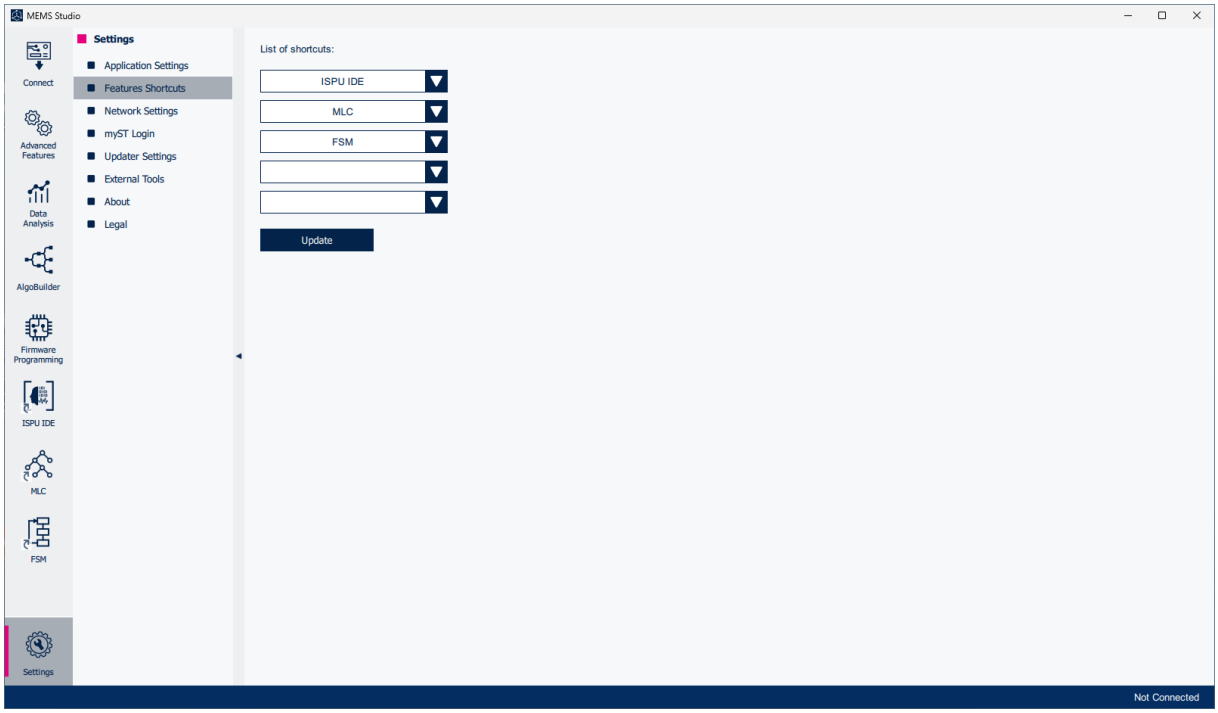
Figure 8. Network Settings



2.7 Features Shortcuts

MEMS Studio has shortcuts for advanced features, making the applications easier and faster to use. Up to five shortcuts can be configured to help you quickly access the most frequently used features from the side menu.

Figure 9. Features Shortcuts



2.8 Running the application for the first time

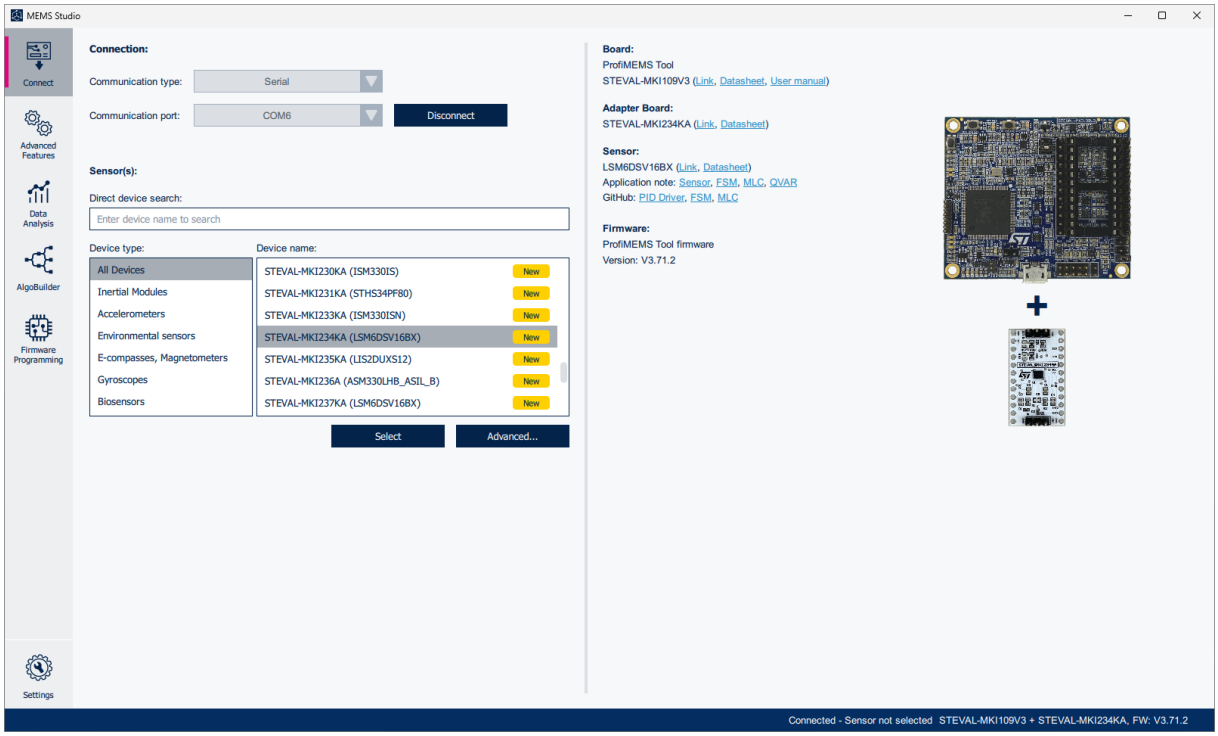
After the application startup, the [**Connect**] page is displayed. To evaluate the sensor and libraries, a compatible device with proper firmware must be connected. You can click on a particular board in the [**Supported Boards**] section to see the list of supported sensors and feature on the board or check Section 2.9: Hardware and firmware compatibility. Features that do not require a connected device like data analysis, MLC design, AlgoBuilder, and others are available all the time.

Figure 10. Connect page

In order to connect to a board, [**Communication type**] needs to be selected. Serial, USB, and BLE interfaces are supported. The USB is used for data acquisition using High Speed Datalog firmware. The BLE interface is used only for the AlgoBuilder firmware evaluation. According to the selected communication type, [**Communication port**], [**USB device**] or [**BLE device**] can then be selected. Communication with the device is initialized by pressing the [**Connect**] button.

After the connection is established, the firmware for the sensor evaluation allows selecting the sensor. In the case of ProfiMEMS firmware, the reference part number of DIL24 adapters or the sensor name is used for the selection. In the case of DatalogExtended firmware, sensor names are used.

Figure 11. **Sensor selection - ProfiMEMS firmware**



Note: As soon as the user selects the device, a validation check is performed at firmware level to verify if the selected device is the same as the connected one. The firmware checks the identifier register WHO_AM_I value in order to verify that it matches the expected value. If it does not match, an error is shown to the user.

The communication interface between the microcontroller and the sensor, as well as the I²C address when the I²C interface is selected and the power supply voltages can be configured before the connection is established by clicking on the [**Advanced...**] button.

Figure 12. **Advanced connection parameters**

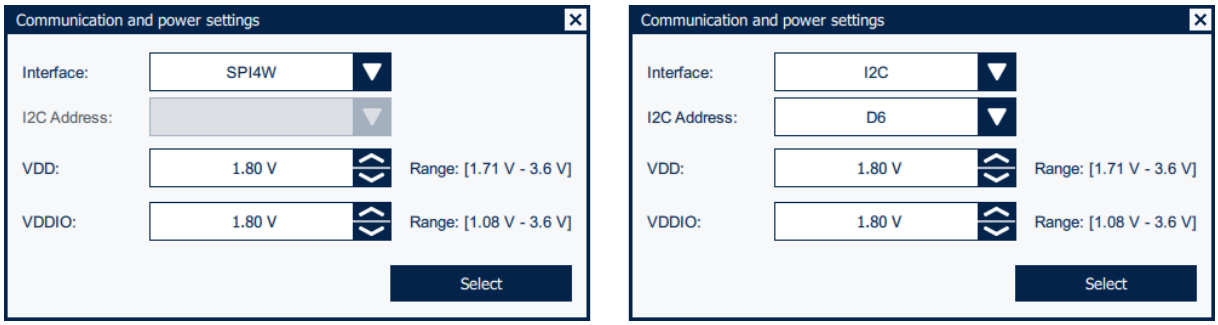
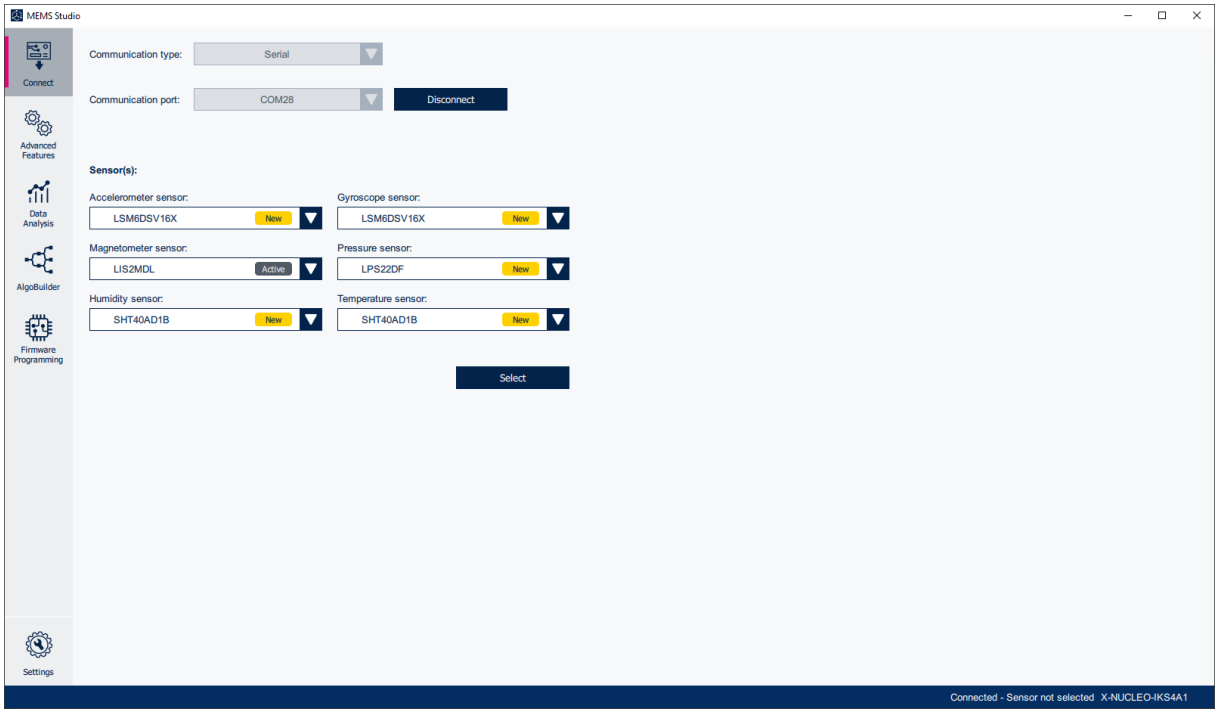


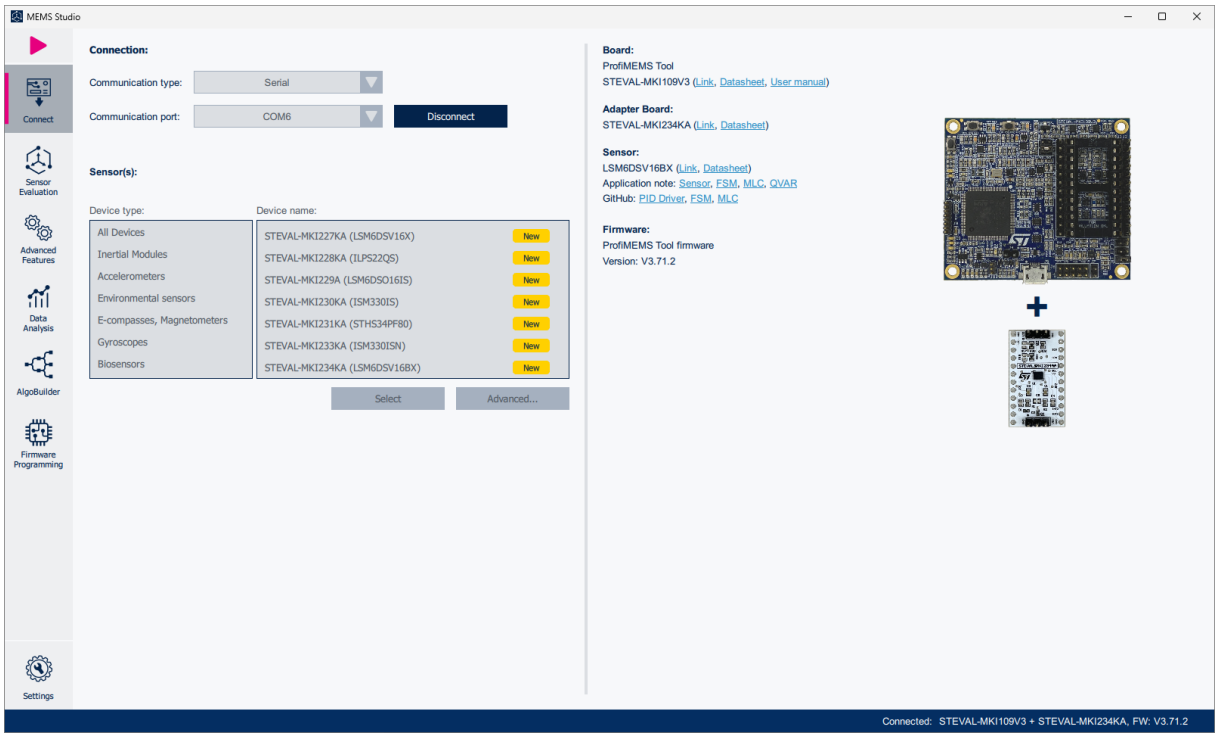
Figure 13. **Sensor selection - DatalogExtended firmware**



In the case of firmware that does not allow sensor selection (firmware for library evaluation, AlgoBuilder firmware, …), the list of sensors is automatically populated and selected.

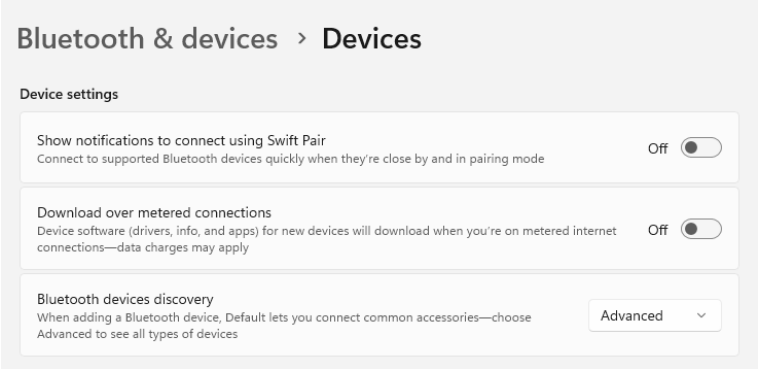
Details about connected boards, sensors, and firmware are displayed in the right panel including links to all available resources like datasheets, applications notes, websites, GitHub repositories, and others.

Figure 14. **Sensor and board references**



Development boards with appropriate AlgoBuilder or DatalogExtended firmware can be connected over a Bluetooth interface. In the Windows operating system, the device needs to be paired first. In order to successfully detect the device in Windows 11, the [**Advanced**] Bluetooth device discovery mode needs to be selected in the Bluetooth settings.

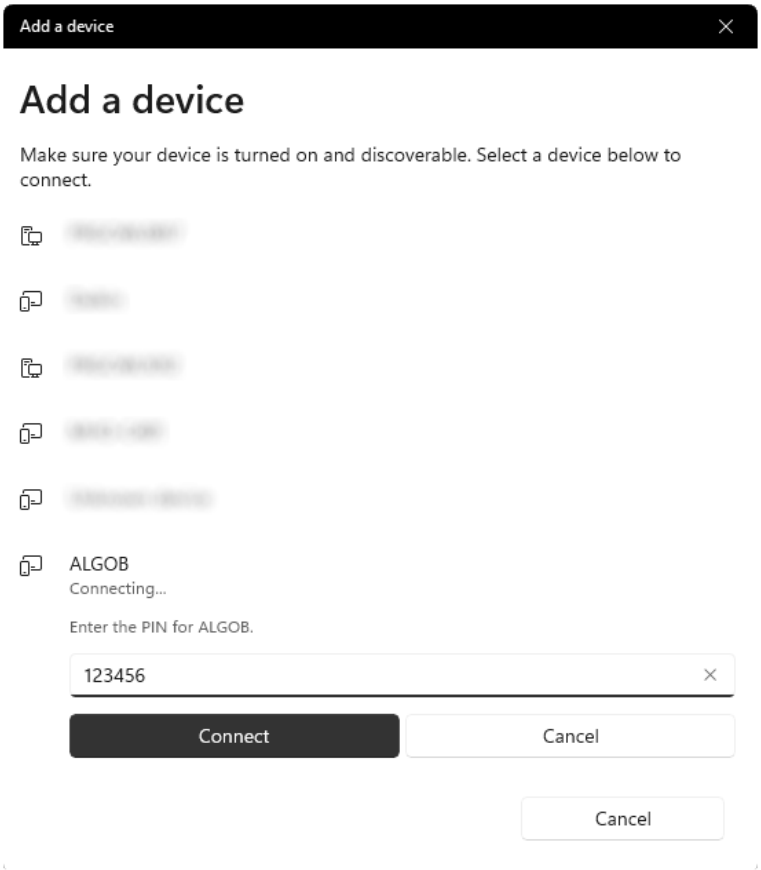
Figure 15. Bluetooth device discovery mode (Windows 11)



The device with AlgoBuilder firmware appears as **ALGOB**. The pin for pairing is always **123456**.

The device with AlgoBuilder firmware appears as **DATAE** followed by the last two bytes of the device MAC address. This firmware does not require a pairing code.

Figure 16. Device pairing



2.9 Hardware and firmware compatibility

The following tables list the combinations of hardware and firmware supported by MEMS Studio for MEMS sensor evaluation as well as those combinations that support library evaluation and the AlgoBuilder functionality.

For a complete evaluation of the sensor, the STEVAL-MKI109D or STEVAL-MKI109V3 (professional MEMS tool) evaluation platform should be used, which supports the adapter boards and sensors indicated in Table 1.

Refer to Table 2 for the X-NUCLEO expansion boards with STM32 ODE (open development environment) used for development and form factor boards dedicated to prototyping. The sensor evaluation firmware on these boards offers configuring and testing some embedded features, such as MLC, FSM, or interrupts.

Refer to Table 3 for supported boards intended for data logging using Datalog2 (High Speed Datalog) firmware.

For library evaluation, refer to Table 4, which indicates the X-NUCLEO expansion board and supported sensor, using the firmware from the X-CUBE-MEMS1 package, however other sensors are supported if the application is generated using STM32CubeMX.

Finally, Table 5 provides the list of hardware platforms and development kits, used in conjunction with AlgoBuilder, that support a limited number of sensors.

Table 1. Supported hardware and firmware for full sensor evaluation

Hardware platform	Firmware	Supported adapter board and sensor
STEVAL-MKI109D STEVAL-MKI109V3 (Professional MEMS tool)	ProfiMEMSToolVx.y.z.bin (from MEMS Studio)	STEVAL-MKI105V1 (LIS3DH) STEVAL-MKI110V1 (AIS328DQ) STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI153V1 (H3LIS331DL) STEVAL-MKI164V1 (LIS2HH12) STEVAL-MKI165V1 (LPS25HB) STEVAL-MKI166V1 (H3LIS100DL) STEVAL-MKI167V1 (H3LIS200DL) STEVAL-MKI168V1 (IIS2DH) STEVAL-MKI169V1 (I3G4250D) STEVAL-MKI170V1 (IIS328DQ) STEVAL-MKI172V1 (LSM303AGR) STEVAL-MKI174V1 (LIS2DS12) STEVAL-MKI175V1 (LIS2DE12) STEVAL-MKI178V2 (LSM6DSL) STEVAL-MKI179V1 (LIS2DW12) STEVAL-MKI180V1 (LIS3DHH) STEVAL-MKI181V1 (LIS2MDL) STEVAL-MKI182V2 (ISM330DLC) STEVAL-MKI184V1 (ISM303DAC) STEVAL-MKI185V1 (IIS2MDC) STEVAL-MKI186V1 (IIS3DHHC) STEVAL-MKI189V1 (LSM6DSM) STEVAL-MKI190V1 (LIS2DTW12) STEVAL-MKI191V1 (IIS2DLPC) STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI193V1 (ASM330LHH) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI200V1K (STTS22H) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI208V1K (IIS3DWB) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI210V2K (ISM330DHCX) STEVAL-MKI211V1K (LIS25BA) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI213V1 (LPS27HHW) STEVAL-MKI214V1 (LPS33K) STEVAL-MKI215V1 (LSM6DSO32) STEVAL-MKI216V1K (IIS3DHHC) STEVAL-MKI217V1 (LSM6DSOX, LIS2MDL) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI220V1 (LPS27HHTW) STEVAL-MKI221V1 (LSM6DSO32X)

Hardware platform	Firmware	Supported adapter board and sensor
		STEVAL-MKI222V1 (LIS2DU12) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI224V1 (LPS22DF) STEVAL-MKI225A (LPS28DFW) STEVAL-MKI226KA (AIS25BA) STEVAL-MKI227KA (LSM6DSV16X) STEVAL-MKI228KA (ILPS22QS) STEVAL-MKI229A (LSM6DSO16IS) STEVAL-MKI230KA (ISM330IS) STEVAL-MKI231KA (STHS34PF80) STEVAL-MKI233KA (ISM330ISN) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI235KA (LIS2DUXS12) STEVAL-MKI236A (ASM330LHB ASIL-B) STEVAL-MKI237KA (LSM6DSV16BX) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI239A (LSM6DSV) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI241KA (LSM6DSV16B) STEVAL-MKI242A (ST1VAFE6AX) STEVAL-MKI243A (ASM330LHHXG1) STEVAL-MKI244A (ASM330LHBG1) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI246KA (IIS2DULPX) STEVAL-MKI247A (LSM6DSV80X) STEVAL-MKI248KA (ISM6HG256X) STEVAL-MKI249KA (ST1VAFE6AX) STEVAL-MKI250KA (ST1VAFE3BX) STEVAL-MKI251A (LSM6DSV320X) STEVAL-MKI252KA (IIS3DWBG1) STEVAL-MKI253KA (IIS3DWB10IS) STEVAL-MKI254A (ASM330LHHG1) STEVAL-MKI255A (MIS2DU12) STEVAL-MET001V1 (LPS22HB) SENSEVAL-SHT4XV1 (SHT40-AD1B) SENSEVAL-SCB4XV1 (SHT40-AD1B, SGP40-D-R4, LPS22DF)

Table 2. Supported hardware and firmware with reduced sensor evaluation

Hardware platform	Firmware	Supported sensor
X-NUCLEO-IKS02A1	DatalogExtended.bin (from the X-CUBE-MEMS1 package)	ISM330DHCX IIS2MDC IIS2DLPC **In the DIL24 socket:** STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI246KA (IIS2DULPX)
X-NUCLEO-IKS01A3	DatalogExtended.bin (from the X-CUBE-MEMS1 package)	LSM6DSO LIS2DW12 LIS2MDL LPS22HH HTS221 **In the DIL24 socket:** STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI153V1 (H3LIS331DL) STEVAL-MKI185V1 (IIS2MDC) STEVAL-MKI191V1 (IIS2DLPC) STEVAL-MKI193V1 (ASM330LHH) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI210V2K (ISM330DHCX)

Hardware platform	Firmware	Supported sensor
		STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI215V1 (LSM6DSO32) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI221V1 (LSM6DSO32X) STEVAL-MKI222V1 (LIS2DU12) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI224V1 (LPS22DF) STEVAL-MKI225A (LPS28DFW) STEVAL-MKI227KA (LSM6DSV16X) STEVAL-MKI228KA (ILPS22QS) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI235KA (LIS2DUXS12) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI239A (LSM6DSV) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI241KA (LSM6DSV16B) STEVAL-MKI247A (LSM6DSV80X) SENSEVAL-SHT4XV1 (SHT40-AD1B)
X-NUCLEO-IKS4A1	DatalogExtended.bin (from MEMS Studio)	LSM6DSV16X LSM6DSO16IS LIS2DUXS12 LIS2MDL LPS22DF STTS22H SHT40-AD1B **In the DIL24 socket:** STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI153V1 (H3LIS331DL) STEVAL-MKI179V1 (LIS2DW12) STEVAL-MKI185V1 (IIS2MDC) STEVAL-MKI191V1 (IIS2DLPC) STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI193V1 (ASM330LHH) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI215V1 (LSM6DSO32) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI221V1 (LSM6DSO32X) STEVAL-MKI222V1 (LIS2DU12) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI225A (LPS28DFW) STEVAL-MKI228KA (ILPS22QS) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI239A (LSM6DSV) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI241KA (LSM6DSV16B) STEVAL-MKI242A (ST1VAFE6AX) STEVAL-MKI243A (ASM330LHHXG1) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI247A (LSM6DSV80X) STEVAL-MKI250KA (ST1VAFE3BX) STEVAL-MKI251A (LSM6DSV320X) STEVAL-MKI254A (ASM330LHHG1) STEVAL-MKI255A (MIS2DU12)
X-NUCLEO-IKS5A1	DatalogExtended.bin (from MEMS Studio)	ISM6HG256X ISM330IS IIS2DULPX IIS2MDC ILPS22QS

Hardware platform	Firmware	Supported sensor
		In the DIL24 socket: STEVAL-MKI191V1 (IIS2DLPC) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI210V2K (ISM330DHCX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI243A (ASM330LHHXG1) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI255A (MIS2DU12) SENSEVAL-SHT4XV1 (SHT40-AD1B)
STEVAL-MKBOXPRO (SensorTile.box Pro)	DatalogExtended.bin (from MEMS Studio)	LSM6DSV16X LIS2DU12 LIS2MDL LPS22DF STTS22H **In the DIL24 socket:** STEVAL-MKI110V1 (AIS328DQ) STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI153V1 (H3LIS331DL) STEVAL-MKI179V1 (LIS2DW12) STEVAL-MKI191V1 (IIS2DLPC) STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI193V1 (ASM330LHH) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI210V2K (ISM330DHCX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI215V1 (LSM6DSO32) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI220V1 (LPS27HHTW) STEVAL-MKI221V1 (LSM6DSO32X) STEVAL-MKI228KA (ILPS22QS) STEVAL-MKI229A (LSM6DSO16IS) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI235KA (LIS2DUXS12) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI239A (LSM6DSV) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI241KA (LSM6DSV16B) STEVAL-MKI242A (ST1VAFE6AX) STEVAL-MKI243A (ASM330LHHXG1) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI246KA (IIS2DULPX) STEVAL-MKI247A (LSM6DSV80X) STEVAL-MKI250KA (ST1VAFE3BX) STEVAL-MKI251A (LSM6DSV320X) STEVAL-MKI252KA (IIS3DWBG1) STEVAL-MKI253KA (IIS3DWB10IS) STEVAL-MKI254A (ASM330LHHG1) STEVAL-MKI255A (MIS2DU12)
STEVAL-STWINBX1 (STWIN.box)	DatalogExtended.bin (from MEMS Studio)	ISM330DHCX IIS2MDC ILPS22QS STTS22H IIS2ICLX IIS2DLPC IIS3DWB **In the DIL24 socket:** STEVAL-MKI125V1 (A3G4250D) STEVAL-MKI151V1 (LIS2DH12) STEVAL-MKI179V1 (LIS2DW12) STEVAL-MKI182V2 (ISM330DLC)

Hardware platform	Firmware	Supported sensor
		STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI193V1 (ASM330LHH) STEVAL-MKI194V1 (LSM6DSR) STEVAL-MKI195V1 (LSM6DSRX) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI215V1 (LSM6DSO32) STEVAL-MKI218V1 (AIS2IH) STEVAL-MKI220V1 (LPS27HHTW) STEVAL-MKI221V1 (LSM6DSO32X) STEVAL-MKI227KA (LSM6DSV16X) STEVAL-MKI229A (LSM6DSO16IS) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI239A (LSM6DSV) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI241KA (LSM6DSV16B) STEVAL-MKI242A (ST1VAFE6AX) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI247A (LSM6DSV80X) STEVAL-MKI251A (LSM6DSV320X) STEVAL-MKI252KA (IIS3DWBG1) STEVAL-MKI253KA (IIS3DWB10IS) STEVAL-MKI254A (ASM330LHHG1) STEVAL-MKI255A (MIS2DU12)

Note: Reduced sensor evaluation on selected boards, in comparison with the full sensor evaluation on the professional MEMS tool, offers only selected features due to hardware platform and firmware limitations.

Table 3. Supported hardware for datalogging using Datalog2 (High Speed Datalog) firmware

Hardware platform	Firmware	Supported sensor
X-NUCLEO-IKS02A1	DATALOG2_Release.bin (from the FP-SNS-DATALOG2 package)	ISM330DHCX IIS2MDC IIS2DLPC
STEVAL-MKBOXPRO (SensorTile.box Pro)	DATALOG2_Release.bin (from the FP-SNS-DATALOG2 package)	LSM6DSV16X LIS2DU12 LIS2MDL LPS22DF STTS22H **In the DIL24 socket:** STEVAL-MKI153V1 (H3LIS331DL) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI230KA (ISM330IS) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI240KA (LSM6DSV32X) STEVAL-MKI247A (LSM6DSV80X) STEVAL-MKI251A (LSM6DSV320X)
STEVAL-STWINKT1B (STWIN)	DATALOG2_Release.bin (from the FP-SNS-DATALOG2 package)	ISM330DHCX IIS2MDC LPS22HH HTS221
STEVAL-STWINBX1 (STWIN.box)	DATALOG2_Release.bin (from the FP-SNS-DATALOG2 package)	ISM330DHCX IIS2MDC ILPS22QS STTS22H IIS2ICLX IIS2DLPC IIS3DWB **In the DIL24 socket:** STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI230KA (ISM330IS) STEVAL-MKI245KA (ISM330BX) STEVAL-MKI246KA (IIS2DULPX) STEVAL-MKI248KA (ISM6HG256X)

Note: Other hardware platforms might be also supported by the Datalog2 firmware for more details check the latest version of the FP-SNS-DATALOG2 package.

Table 4. Supported hardware and firmware for library evaluation

Hardware platform	Firmware	Supported sensor
X-NUCLEO-IKS02A1	see Applications folder (from the X-CUBE-MEMS1 package)	ISM330DHCX IIS2MDC Other components can be used if the application is generated through STM32CubeMX.
X-NUCLEO-IKS01A3	see Applications folder (from the X-CUBE-MEMS1 package)	LSM6DSO LIS2MDL LPS22HH HTS221 Other components can be used if the application is generated through STM32CubeMX.
X-NUCLEO-IKS4A1	see Applications folder (from the X-CUBE-MEMS1 package)	LSM6DSV16X LIS2MDL LPS22DF STTS22H Other components can be used if the application is generated through STM32CubeMX.
X-NUCLEO-IKS5A1	see Applications folder (from the X-CUBE-MEMS1 package)	ISM6HG256X IIS2MDC ILPS22QS Other components can be used if the application is generated through STM32CubeMX.

Table 5. Supported hardware and firmware with AlgoBuilder functionality

Hardware platform	Firmware	Supported sensor
NUCLEO-F401RE or NUCLEO-U575ZI-Q or NUCLEO-U385RG-Q with X-NUCLEO-IKS5A1	generated in AlgoBuilder	ISM6HG256X ISM330IS IIS2MDC ILPS22QS
NUCLEO-F401RE or NUCLEO-U575ZI-Q or NUCLEO-U385RG-Q with X-NUCLEO-IKS4A1		LSM6DSV16X LSM6DSO16IS LIS2MDL LPS22DF SHT40-AD1B
NUCLEO-F401RE NUCLEO-U575ZI-Q with X-NUCLEO-IKS02A1 STEVAL-MKI230KA		STEVAL-MKI230KA (ISM330IS) IIS2MDC
STEVAL-MKSBOX1V1 (SensorTile.box)		LSM6DSOX LIS2MDL LPS22HH HTS221
STEVAL-MKBOXPRO (SensorTile.box Pro)		LSM6DSV16X LIS2MDL LPS22DF STTS22H STEVAL-MKI229KA (LSM6DSO16IS) STEVAL-MKI251A (LSM6DSV320X)
STEVAL-STWINKT1B (STWIN)		ISM330DHCX IIS2MDC LPS22HH HTS221
STEVAL-STWINBX1 (STWIN.box)		ISM330DHCX IIS2MDC ILPS22QS STTS22H STEVAL-MKI230KA (ISM330IS) STEVAL-MKI248KA (ISM6HG256X)

3 Sensor evaluation

The sensor evaluation features are dynamically adjusted according to the available functionalities of the connected evaluation board, sensor, and firmware.

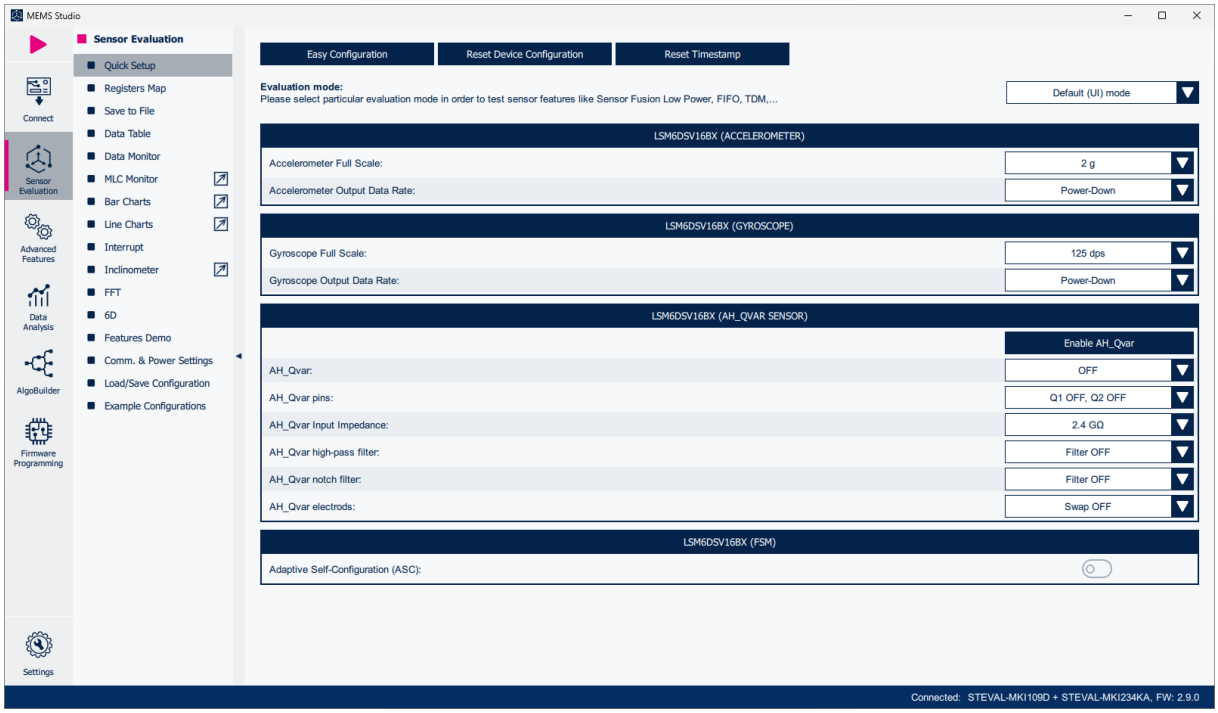
Quick Setup page

The [**Quick Setup**] page allows the user to configure basic sensor configurations such as the output data rate, full scale, and others according to the device datasheet. The content of the page is adjusted according to the connected evaluation board, sensors, and firmware.

If more evaluation modes are available like sensor fusion low-power, FIFO, TDM, and others, they can be selected on this page. The available sensor evaluation pages are adjusted according to the selected mode. If applicable, the control can align its status to the current register value.

The [**Easy Configuration**] button configures the sensor to a predefined configuration to stream sensor data. The [**Reset Device Configuration**] button restores the default sensor configuration. The [**Reset Timestamp**] button resets the timestamp value to 0 and clears all application data buffers, which means it will also clear all the charts.

Figure 17. Quick Setup page



Registers Map page

The [**Registers Map**] displays the list of device registers divided by register pages, if any.

Every register item has a [**Read**], [**Write**], and [**Default**] (restore default value) button enabled according to the datasheet and a bit view populated according to the register value.

Using the dedicated button [**?**], any register item can be expanded to explore the register value bit map description. Individual bit values can be modified by clicking on the value in the table or clicking on the related check box.

Figure 18. Registers Map page

[**Custom Register Action**] allows direct read and write transactions to a single register by specifying the register address, page and value.

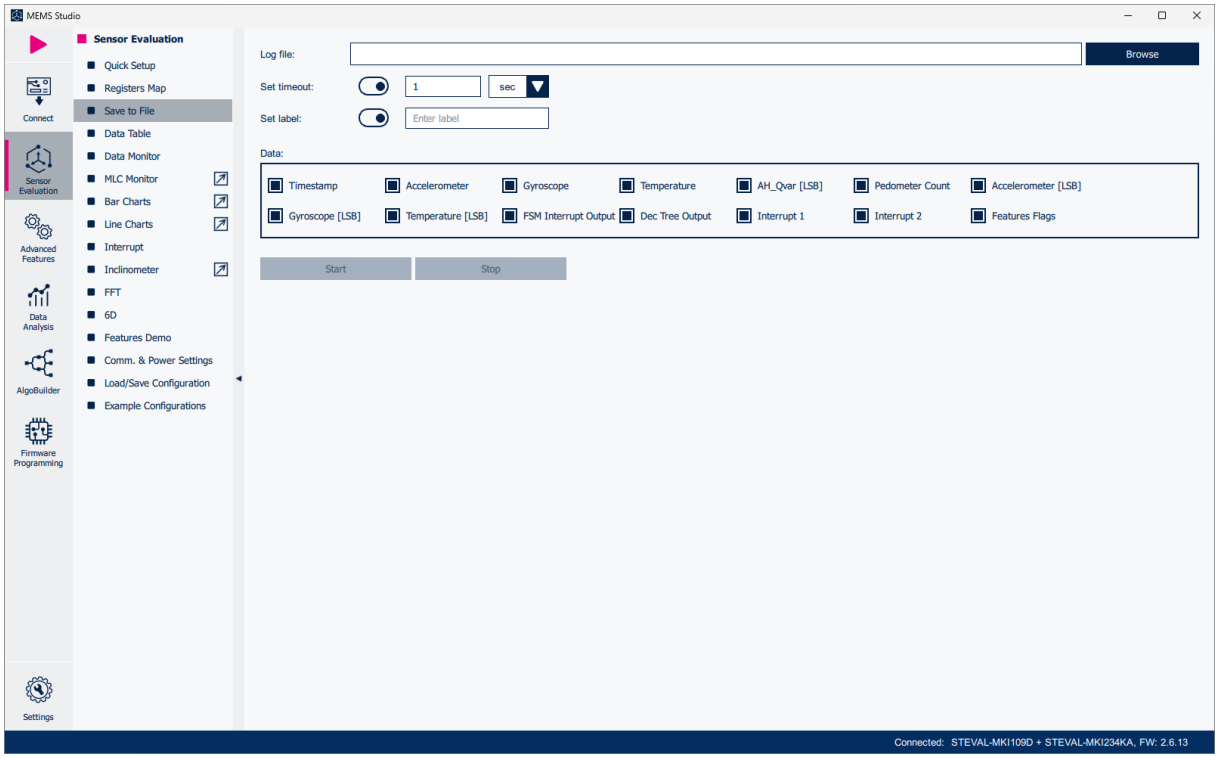
Figure 19. Custom register action

Save to File page

The user can use this page to save a stream of sensor output data in a .csv file for postprocessing. This can be done by selecting the data to be stored.

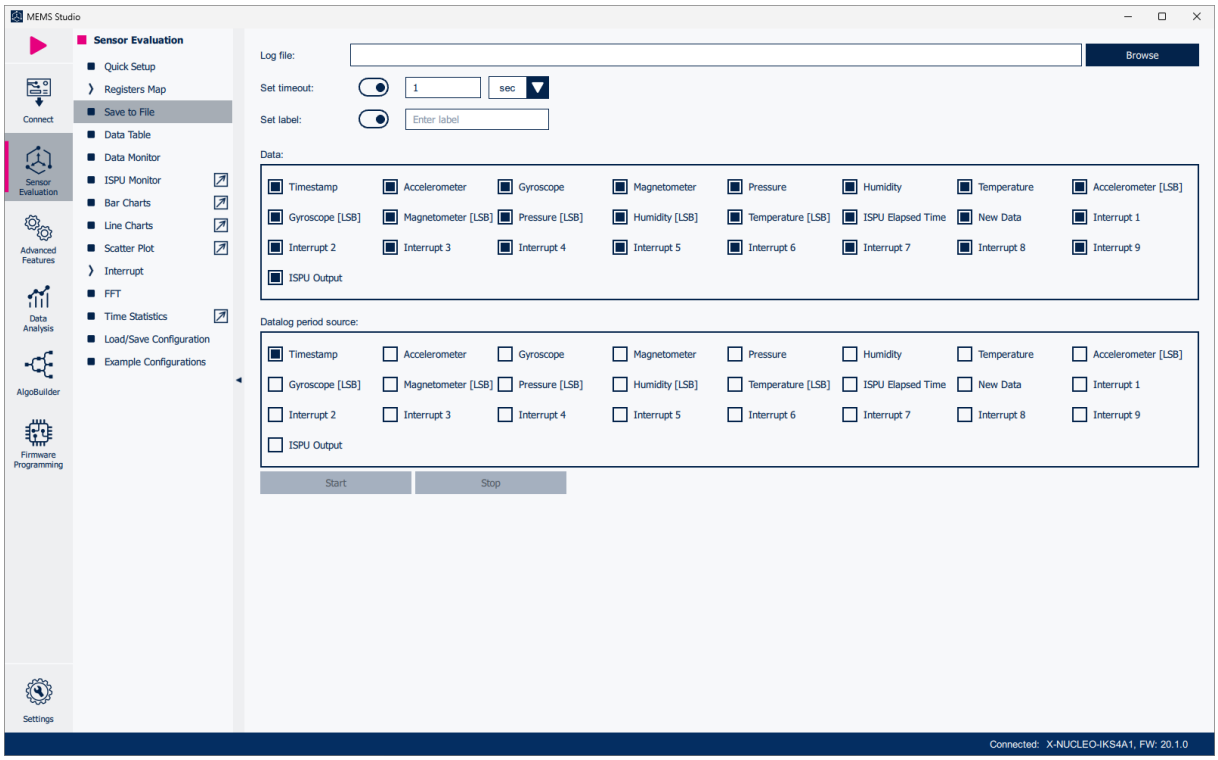
The [**Browse**] button is used to select the folder and insert the file name. The acquisition period can be set to a fixed time in [ms] if desired. Then data to be saved can be selected from the list. The acquisition can be controlled using the [**Start**] and [**Stop**] buttons. If the [**Logging timeout**] option is used, the acquisition is going to run as long as the timeout period. A label can be assigned to the log using the [**Set label**] option.

Figure 20. Save to File page



If the sensors are configured according to different output data rates, the user can select the [**Data log period source**] from among the available data types using a dedicated selection option. Refer to the following figure.

Figure 21. Data log period source



If the Timestamp is selected as the datalog period source, every new data from any sensor will create a new row in the log file. For other sensors that do not have new data at that moment, the row is filled with previous data.

In order to distinguish newly updated data in each row in the CSV log file, a**data_changed** bit field is used. This bit field indicates which column in a given row contains new data. It is not dependent on any sensor type, rather it is strictly related to the column number. The least significant bit corresponds to the first data column and the timestamp column is not counted. This field is used in the DatalogExtended firmware, as well as whenever data with different output data rates are logged in the ProfiMEMS firmware i.e. logging UI + TDM data. The same principle applies to the **data_changed** bit field in the Datalog2 firmware.

When using Datalog2 (High Speed Datalog) firmware, data are primarily stored in Datalog2 native binary format. Using the [**Convert to CSV**] switch, binary format can be automatically converted to CSV format at the end of the logging session. After conversion to a CSV file, an additional column called **data_changed** is added to indicate changes.

*Note: The **data_changed** bit field replaces the **new_data** flag from MEMS Studio v2.4.0. The **new_data** flag will no longer be saved in the CSV file, but it will be properly decoded by Data Analysis.*

A **new_data** flag was used to distinguish which data points are new in that particular row. The new data flag is a bit field with following meaning.

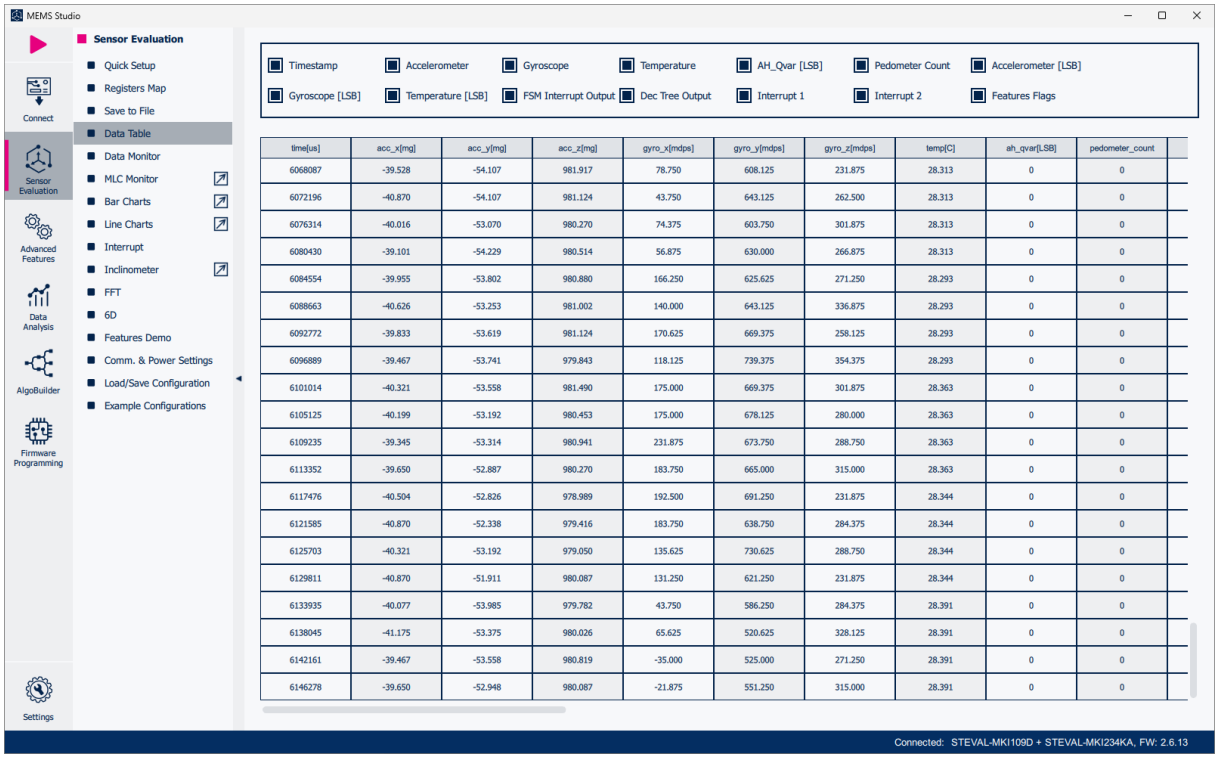
Table 6. New data flag values

Sensor	Bit	Value
Pressure Sensor	1	0x001
Temperature Sensor	2	0x002
Humidity Sensor	3	0x004
Accelerometer	4	0x008
Gyroscope	5	0x010
Magnetometer	6	0x020
Interrupt	7	0x040
MLC	8	0x080
FSM	9	0x100
ISPU	10	0x200
VAFE/QVAR	11	0x400
Accelerometer	12	0x800

Data Table tool

The [**Data Table**] tool displays the output values measured by the sensor connected to the evaluation board. This view provides an overview of all samples received with a table data update rate of 0.5 Hz to reduce overall CPU/GPU consumption. This functionality is disabled in the case of very high data rates.

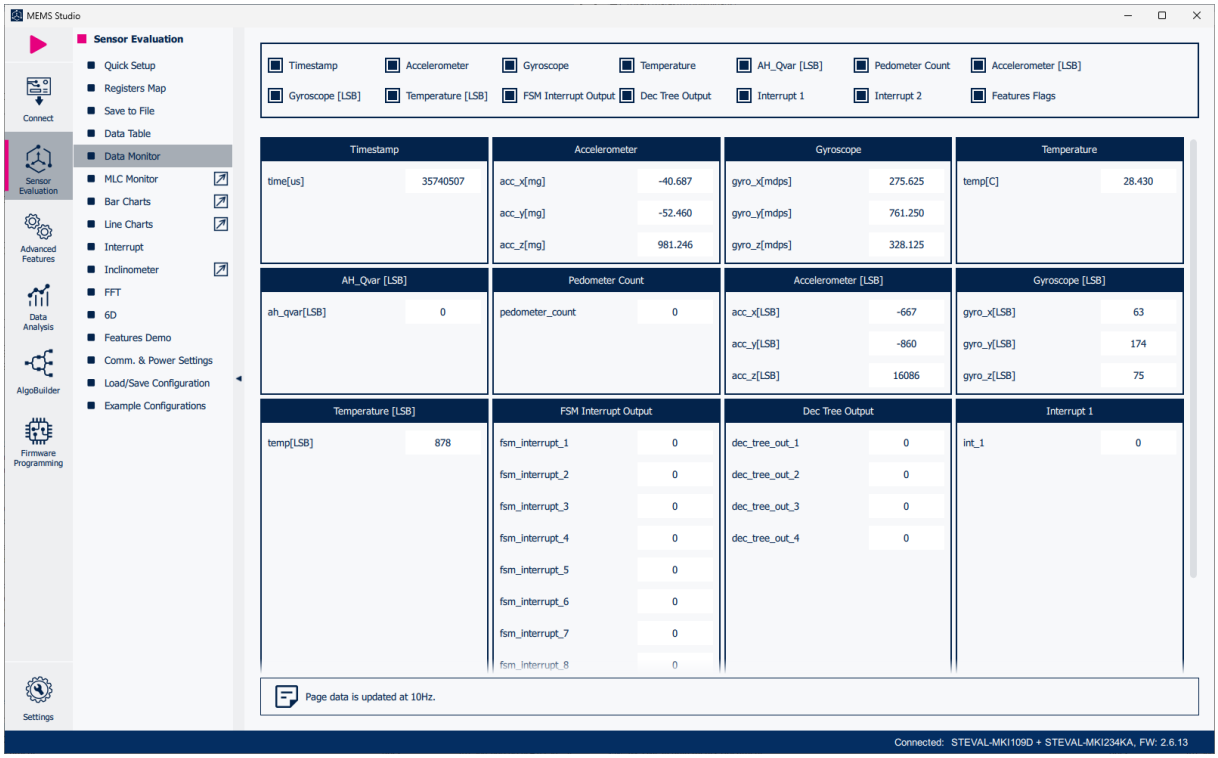
Figure 22. Data Table tool



Data Monitor tool

The [**Data Monitor**] tool displays the latest output values measured by the sensor connected to the evaluation board. This view provides an overview of the last samples received with an update rate of 10 Hz to reduce overall CPU/GPU consumption in the case of high data rates.

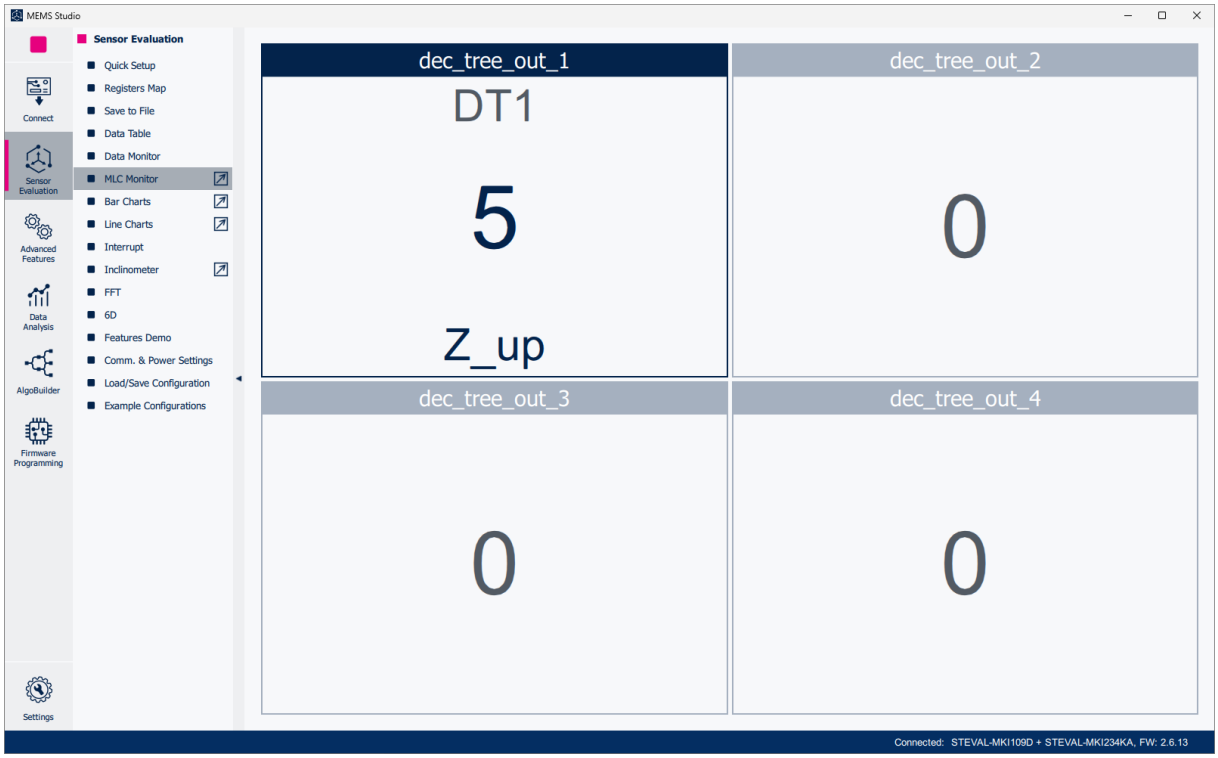
Figure 23. Data Monitor tool



MLC Monitor, FSM Monitor, ISPU Monitor

The [**MLC Monitor**], [**FSM Monitor**], and [**ISPU Monitor**] pages can be used to display output from the machine learning core (MLC) or finite state machine (FSM) including the label assigned to each output value and output from the intelligent sensor processing unit (ISPU). This feature provides a clear and user-friendly overview of the current states and classifications detected by the sensors, making it easier to monitor and interpret the results in real time. The textual values are taken from the JSON file with the sensor configuration, which is loaded from the [**Load/Save Configuration**] or [**Example Configurations**] pages.

Figure 24. MLC Monitor



Bar Charts tool

The [**Bar Charts**] tool displays the data measured by the sensor in a bar chart format.

The height of each bar is determined by the amplitude of the signal measured by the sensor along the corresponding axis. The full scale of the graph depends on the configuration and can be changed through either the [**Quick Setup**] or the [**Registers Map**] tab.

Figure 25. Bar Charts tool

Line Charts tool

The [**Line Charts**] tool shows the evolution of the output over time. This tool generates a time graph of data samples from the connected sensors.

Each waveform can be shown or hidden by clicking on the corresponding colored box in the top bar of the plot.

The user can click on the minus and plus at the bottom of the graph object to manually scale the plot horizontally. Furthermore, using a dedicated scroll bar at the bottom of the page, the user can display the older samples.

By mousing over the plot area, the top bar labels are updated to display the collected values of the samples.

Figure 26. Line Charts tool

Note: Individual signals can be disabled or enabled in the chart top bar. Using a right mouse click in this area, all channels can be selected or deselected.

Hovering over the right side of a graph object, available options and settings are displayed such as:

- [**Fit**]: updates the Y-axis scale to fit all waveforms
- [**Auto**]: automatic update of the Y-axis scale
- [**Full**]: sets the full-scale value according to the device configuration
- [**Save**]: exports a screenshot of the chart to an image file in .png format
- [**Copy**]: saves the chart as an image in the system clipboard
- [**Help**]: displays the available keyboard shortcuts for navigation in the chart

Figure 27. Line chart options and help

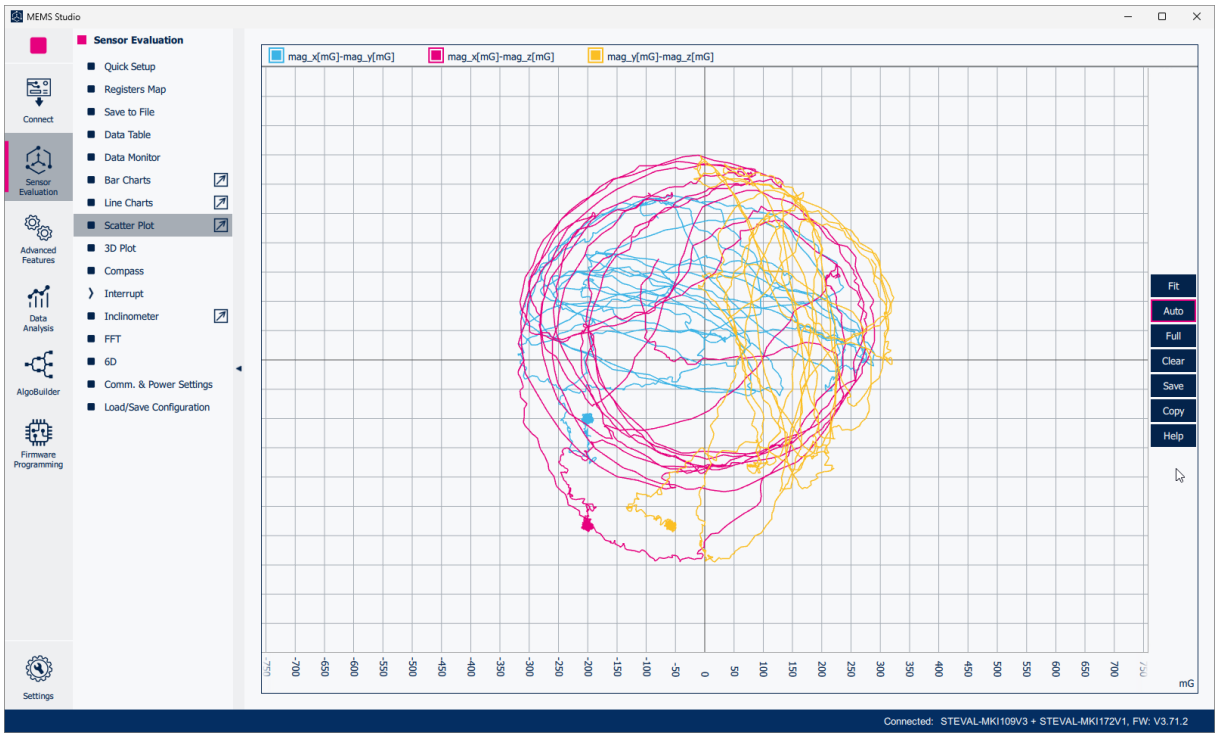
Scatter Plot tool

The [**Scatter Plot**] tool shows the graphical representation of the magnetometer data using Cartesian coordinates. In general, a scatter plot is used to display values for typically two variable sets of data.

The plot shows three lines according to the sensor axes (X-Y, Y-Z, X-Z). The three lines can be enabled and disabled independently by clicking on the corresponding colored box at the top of the graph. By clicking on the [**Clear**] button, the user can reset all data in the plot.

The plot component detects mouse wheel events in order to zoom in or out on data. [**Auto**] allows the user to set the automatic zoom in order to fit all data on the graph.

Figure 28. Scatter Plot tool



Compass tool

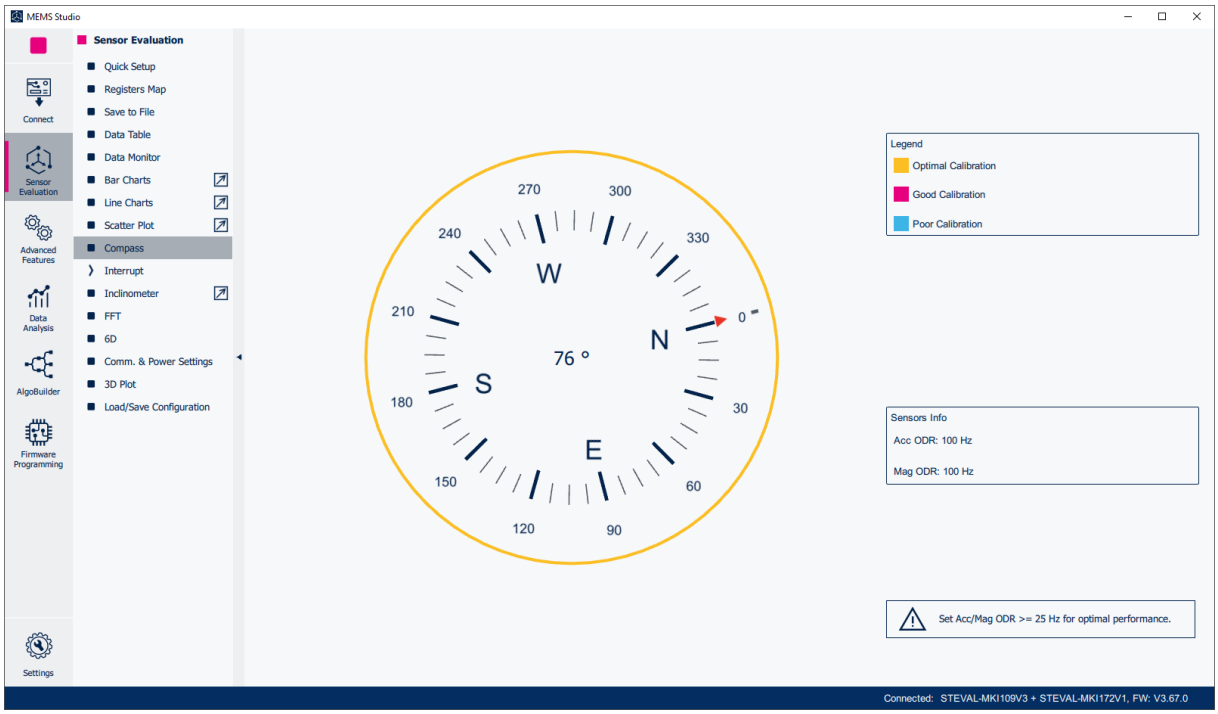
The [**Compass**] tool shows an example of a compass application, which can be implemented using a 6-axis module (3-axis accelerometer and 3-axis magnetometer).

The algorithm uses the magnetometer data to measure the Earth's magnetic field, and the accelerometer data to compensate for the inclination of the board. Rotating the board, the application shows the heading of the compass.

The performance of the compass is related to the configuration used. So, the application shows the current configuration and the recommended configuration (accelerometer and magnetometer ODR).

Before using the compass demo, the system must be calibrated by moving the board randomly for a few seconds; the quality of the calibration step is indicated by the compass object border color according to the legend on the right side.

Figure 29. Compass tool



Interrupt tool

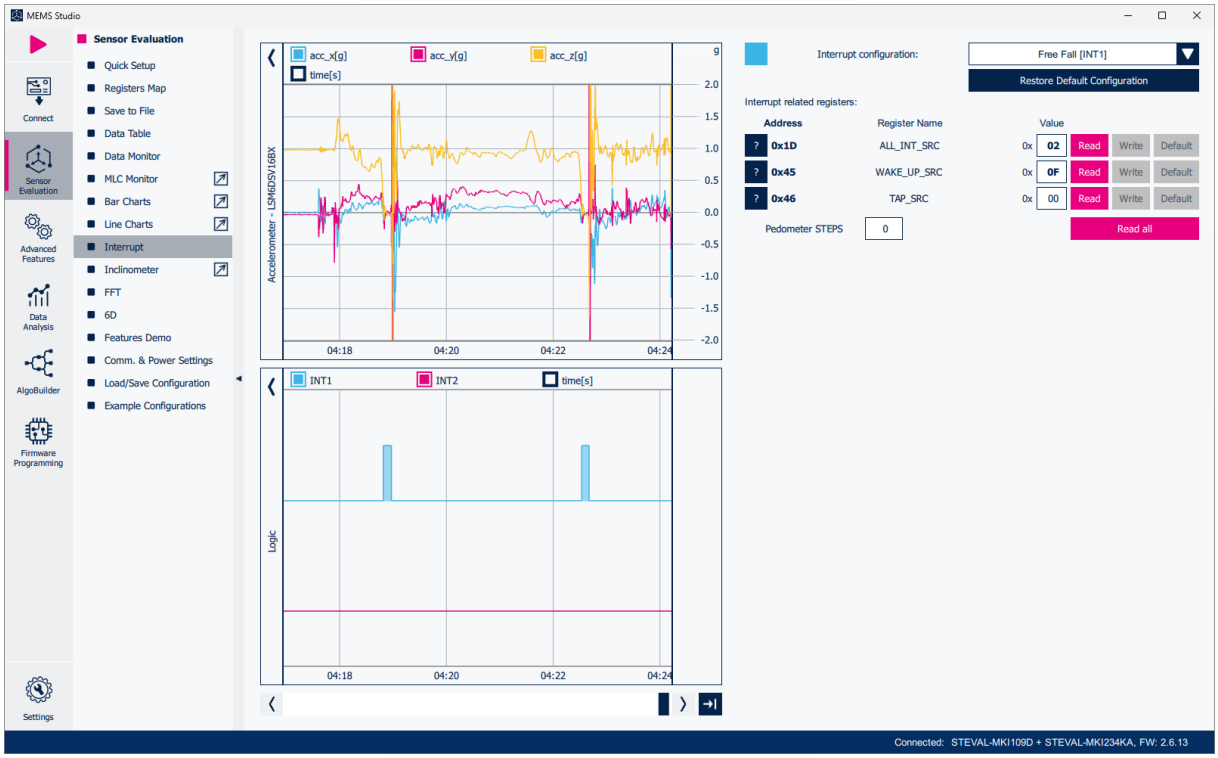
The [**Interrupt**] tool allows evaluating the interrupt generation features of the MEMS sensor.

On this page, it is possible to configure the characteristics of the inertial events that must be recognized by the device and to visualize (in real time) the evolution of the interrupt lines.

The application provides access to the interrupt registers, which allow the configuration of the interrupt sources of the device.

The user can configure the interrupts using the dedicated combo box in order to load the recommended configurations in the device.

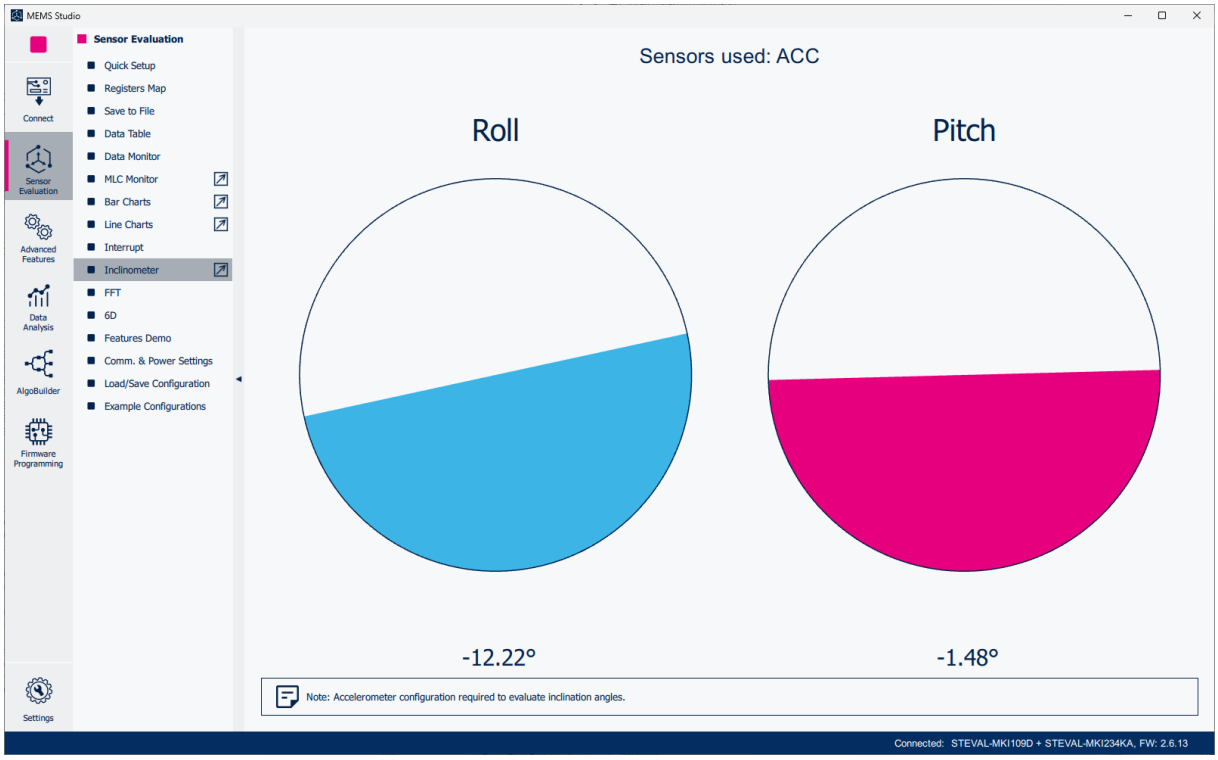
Figure 30. Interrupt tool



Inclinometer tool

The [**Inclinometer**] tool represents the angle between the accelerometer axis and the horizontal plane. This tool is available if the sensor in use integrates an accelerometer.

Figure 31. Inclinometer tool



FFT tool

The [**FFT**] tool displays the fast Fourier transform of the sensor data. The page shows the frequency-domain plot for each axis of the selected sensor.

The tool provides various options such as:

- [**DC nulling**]: removes the DC component from the signal
- [**Magnitude**]: calculates Euler's formula of the vector composed of the axis components
- [**Magnitude (scale)**]: Y-axis scale
- [**Frequency (scale)**]: X-axis scale
- [**Style**]: style of chart (waveform or bars)
- [**Window**]: window function applied to the signal before FFT evaluation
- [**Length**]: length of the window used to evaluate FFT (number of samples from which FFT is computed)
- [**Zero padding**]: number of zero samples added to the captured signal
- [**Show spectrogram**]: enables or disables spectrogram visualization
- [**Fixed refresh rate**]: updates the FFT chart without overlapping windows
- [**Fixed overlap**]: updates the FFT chart, overlapping windows according to the slider value of the window overlap
- [**Slider value**]: percentage of the samples saved for the next FFT computation

Figure 32. FFT tool

6D tool

The [**6D**] tool provides an example of how to use the [**6D position**] function.

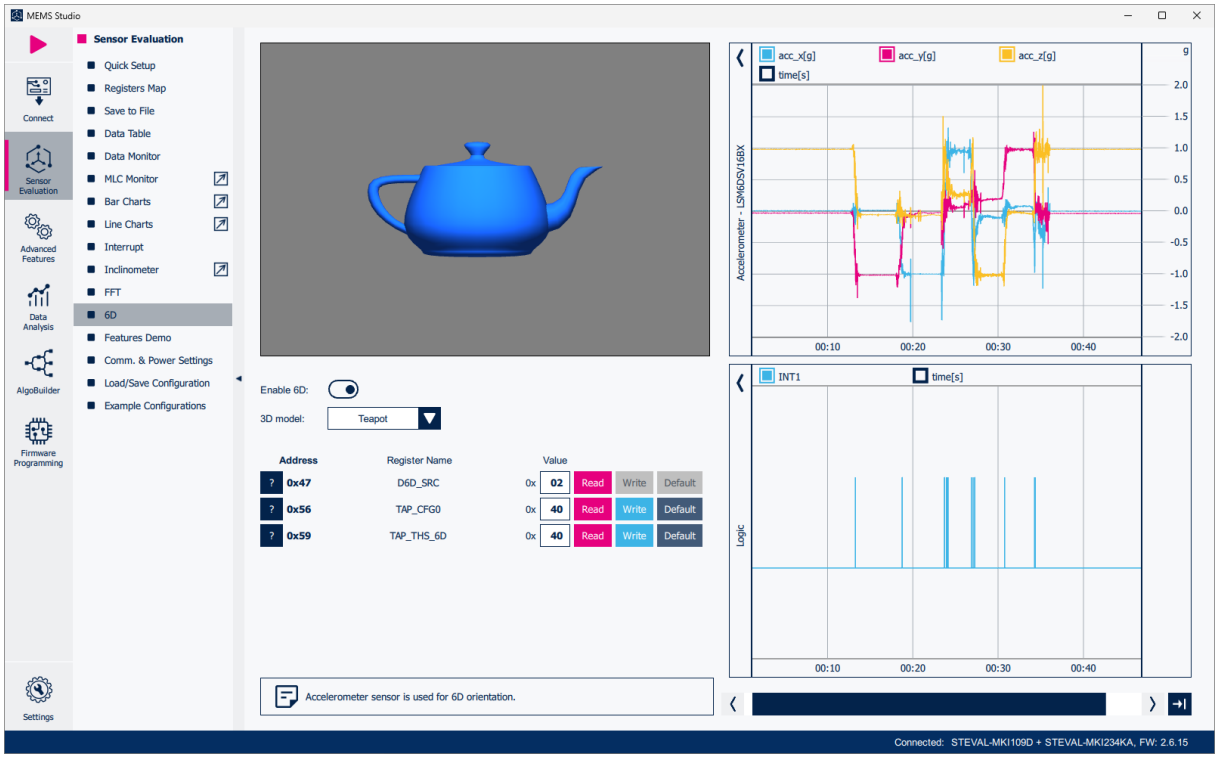
On this page, it is possible to configure the interrupt for 6D recognition by clicking on the [**6D Enabled**] toggle button.

After loading the configuration, the selected 3D model is oriented depending on the 6D position detected.

The window also shows (in real time) the evolution of the interrupt line.

The user can change the register configuration by setting different values in the interrupt registers as shown in the tool.

Figure 33. 6D tool

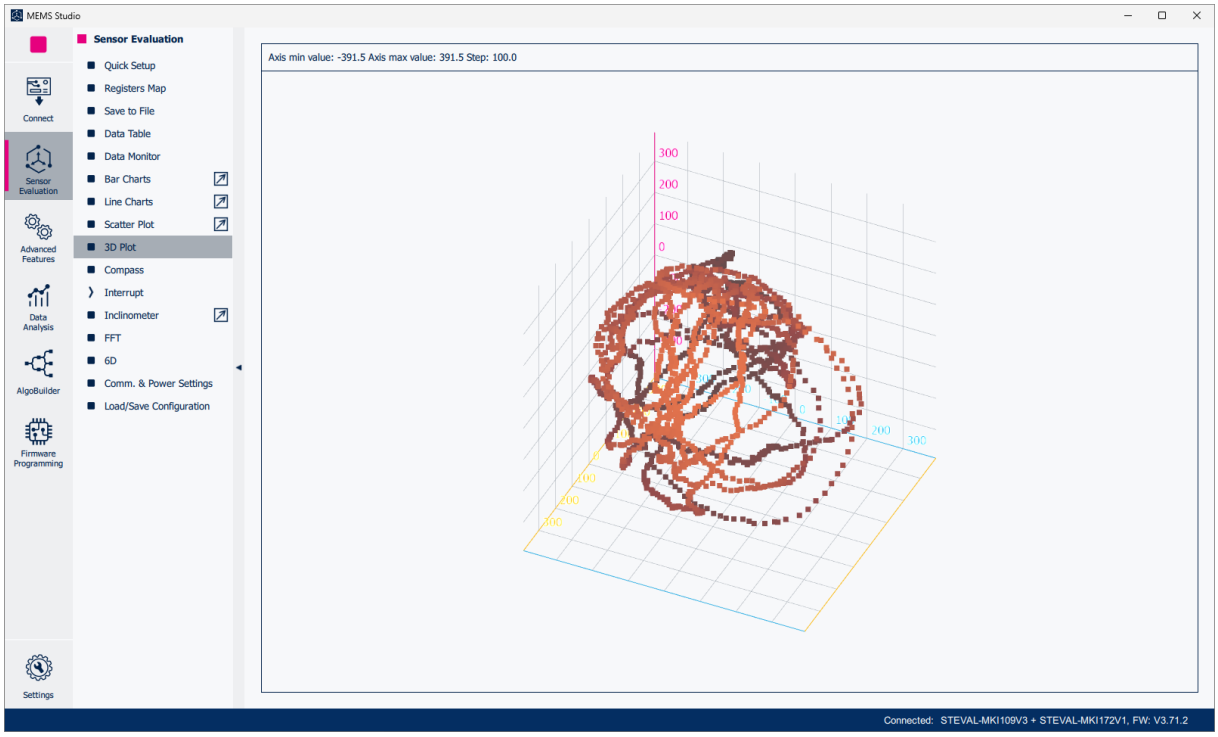


3D Plot tool

The [**3D Plot**] tool shows the graphical representation of the magnetometer data in 3D space.

Zooming in and out of the chart is possible using the mouse wheel. Changing the view angle is possible by clicking and panning in the chart.

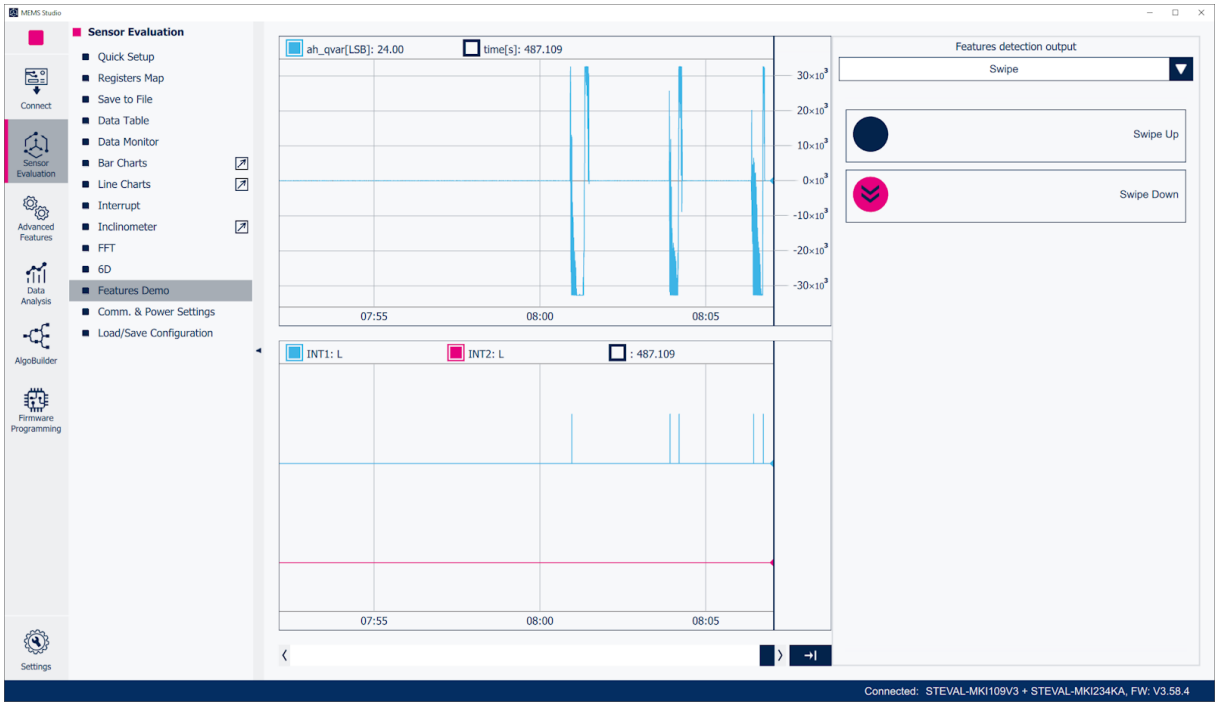
Figure 34. 3D Plot tool



Features Demo page

The [**Features Demo**] page allows the user to evaluate demo modes available for a connected sensor when this feature is supported. The side menu on the right displays indicators to notify the user about event detection and set parameters, if applicable.

Figure 35. Features Demo page



Communication and Power Settings tool

The [**Comm. & Power Settings**] tool allows the user to select a communication interface like I²C, SPI 3-wire, SPI 4-wire, I3C if available, read the current consumption, set the VDD and VDDIO values and select between Interrupt and Polling [**Data Reading Mode**]. This functionalities are available only on the Professional MEMS tool board (STEVAL-MKI109D, STEVAL-MKI109V3).

Figure 36. Comm. & Power Settings tool

FIFO tool

The [**FIFO**] tool can be used to test the first-in first-out data buffer embedded in the device when this feature is supported by the sensor (see the device datasheet for more details).

By using the combo box available on the page, the FIFO can be configured in all the supported modes (for example, bypass, FIFO, stream, stream-to-FIFO).

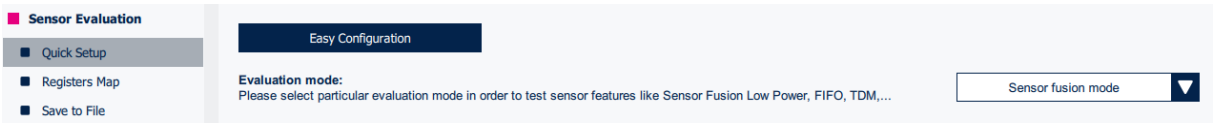
The application shows the values of the X, Y, Z data stored in the deep FIFO buffer, indicating both numerical data and the corresponding graph.

Figure 37. FIFO tool

Sensor fusion tool

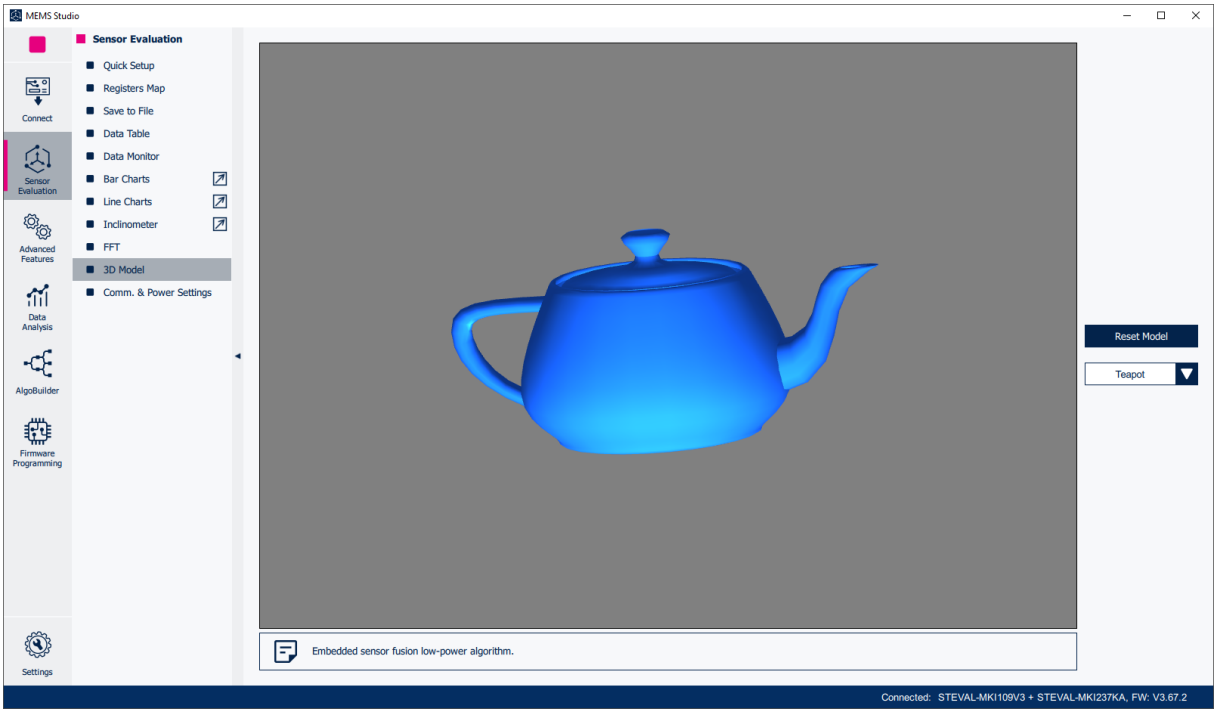
Some sensors can be equipped with the sensor fusion low-power (SFLP) feature for generating game rotation vector, gravity vector, and gyroscope bias data based on the accelerometer and gyroscope data processing. If this feature is available, the evaluation can be activated on the [**Quick Setup**] page.

Figure 38. Evaluation mode selection



If sensor fusion mode is selected, the available sensor evaluation pages are adjusted according to this mode. Game rotation vector, quaternions, gravity vector, and gyroscope bias signals are added in [**Data Table**], [**Data Monitor**], [**Bar Charts**], [**Line Charts**] and [**Save to File**]. The [**3D Model**] page is available in this mode. A 3D model is displayed on this page and the orientation of the model changes based on the game rotation vector data so the model follows the sensor movement. Different 3D models can be selected. The initial position of the model can be set by using the [**Reset Model**] button.

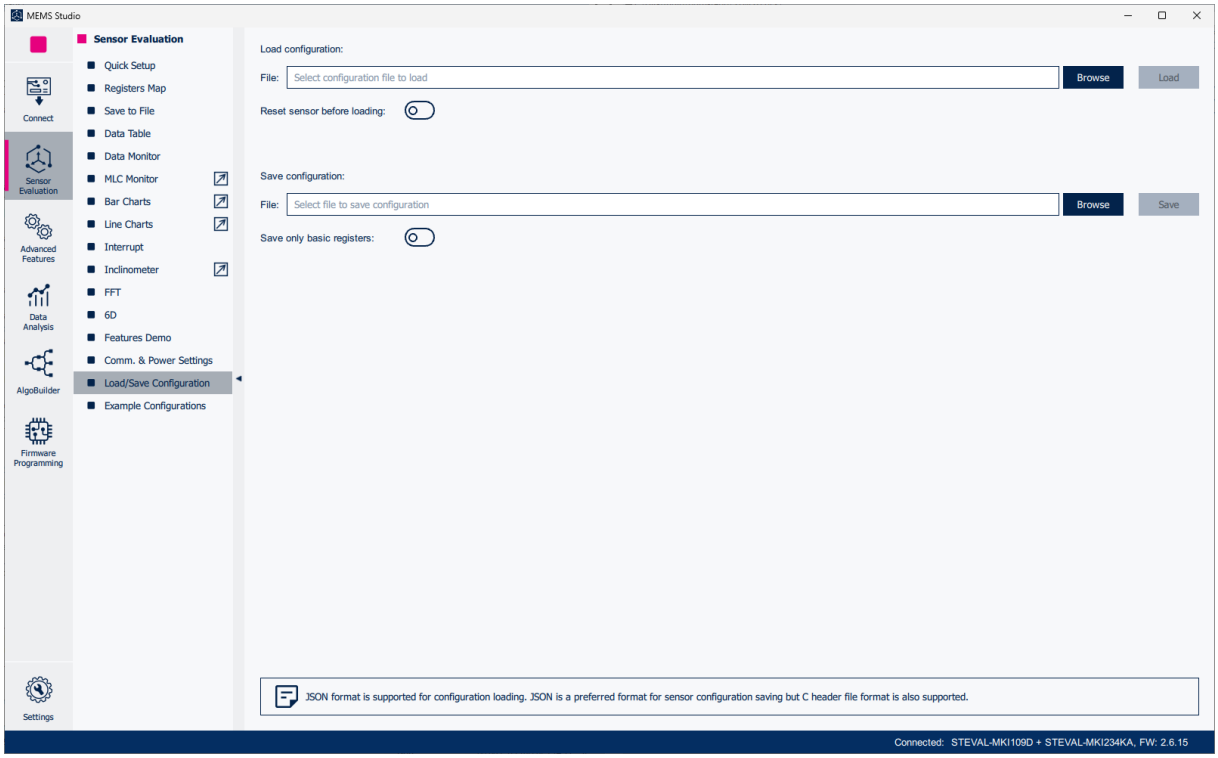
Figure 39. 3D Model page



Load/Save Configuration page

The [**Load/Save Configuration**] page allows the user to save and load the current register configuration. Clicking on the [**Save**] button stores the register values in a .json file. The saved configuration can be loaded at any time by clicking on the [**Load**] button.

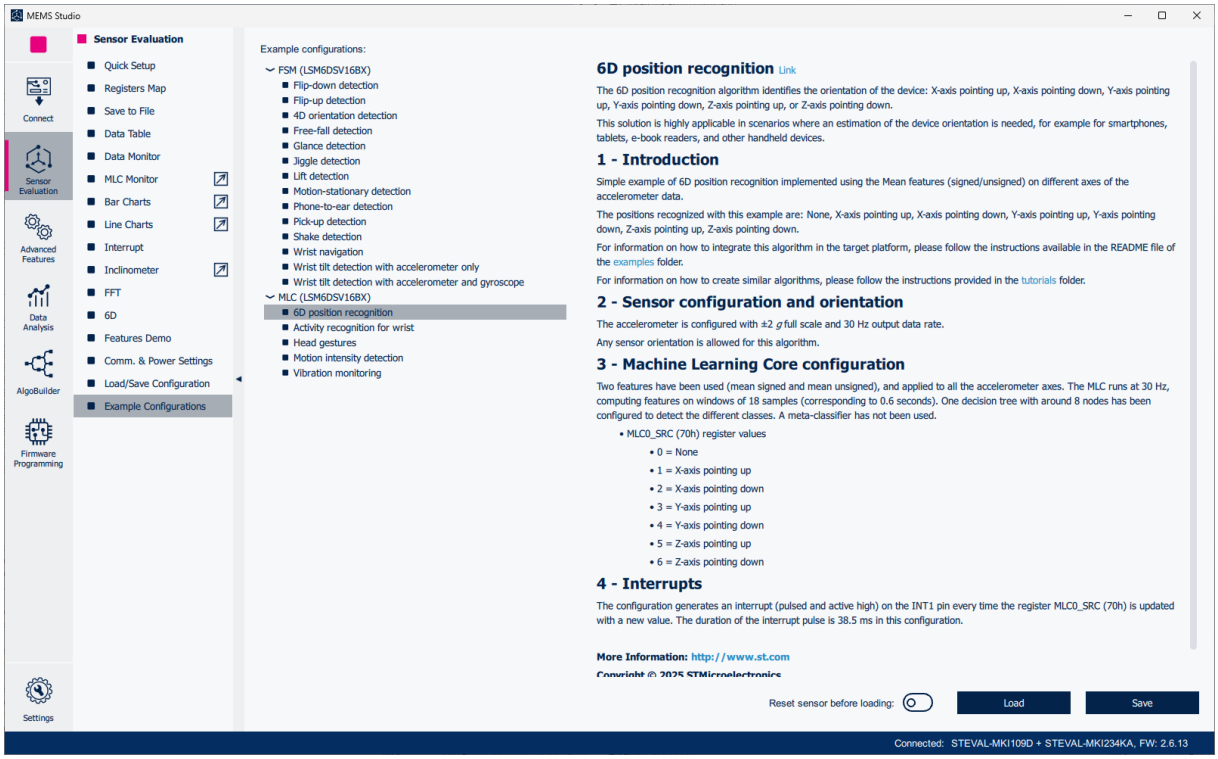
Figure 40. Load/Save Configuration page



Example Configurations page

The [**Example Configurations**] page is available for sensors equipped with FSM, MLC, or ISPU features. This page downloads content directly from the STMicroelectronics GitHub repository, providing an up-to-date list of example configurations for embedded features. Users can select a configuration and upload it directly to the sensor for immediate testing.

Figure 41. Example Configurations page

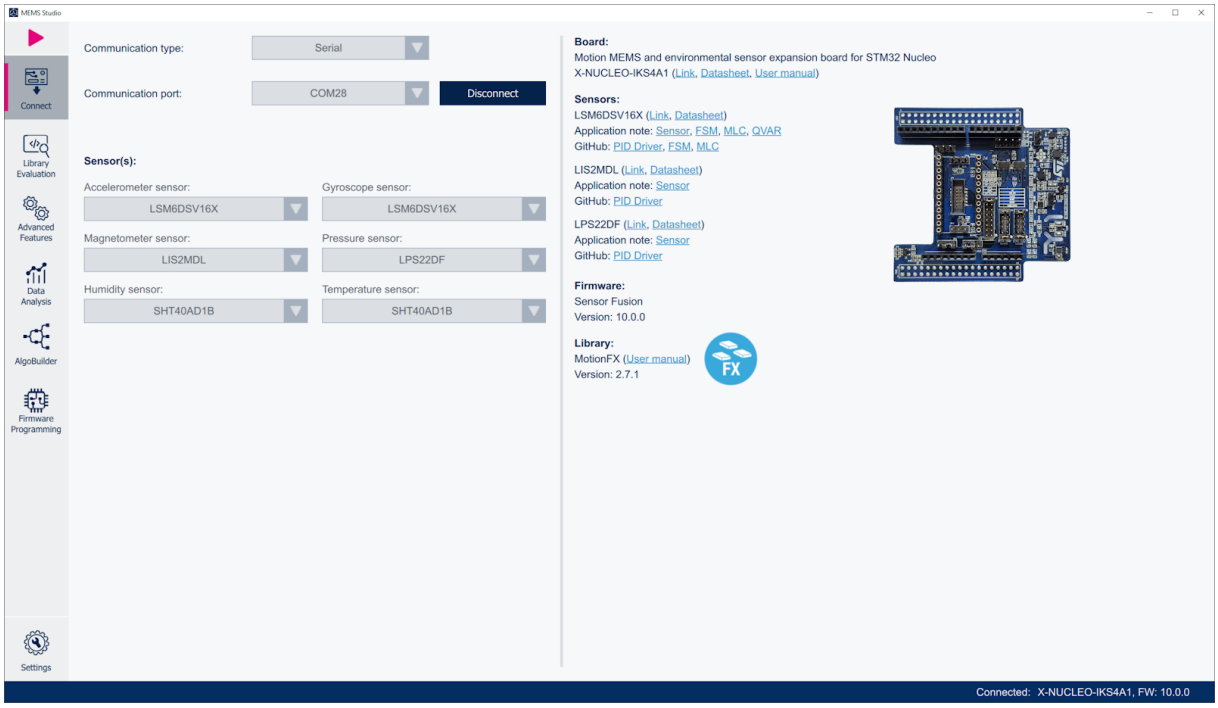


4 Library evaluation

Middleware libraries for ST sensors are provided in the X-CUBE-MEMS1 package. This package contains dedicated projects for each library, which can be used for library evaluation. Using MEMS Studio and the firmware for a particular library with the selected development board, a user can evaluate the functionality of the library.

After the development board is programmed with the selected firmware, the connection can be established by selecting [**Communication port**] and pressing the [**Connect**] button. Used sensors are indicated by the firmware, and they are listed on the [**Connect**] page with all details.

Figure 42. Connect page for library evaluation



After a successful connection, the [**Library Evaluation**] functionality is added in the menu.

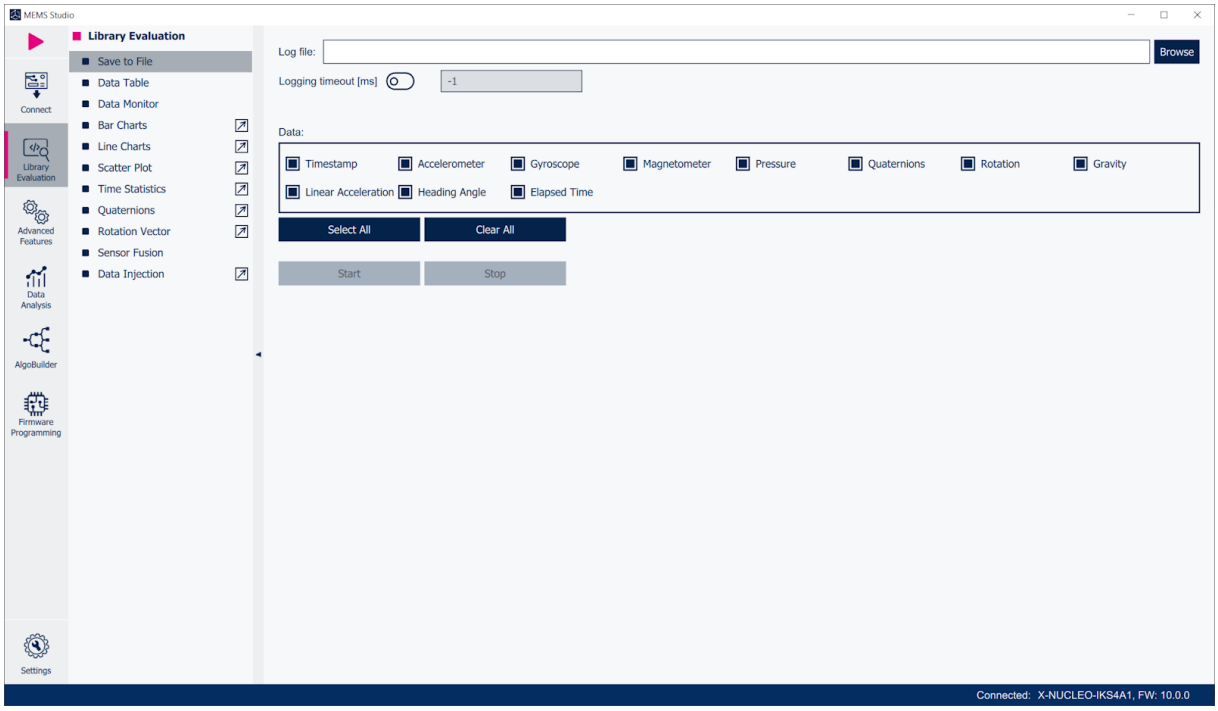
The library evaluation offers:

- Save To File
- Data Table
- Data Monitor
- Data visualization pages
- Time Statistics
- Data Injection

Save To File

The user can use this page to save a stream of sensor and library output data in a .csv file. This can be done by selecting the data to be stored. The [**Browse**] button is used to select the folder and insert the file name, then the [**Start**] and [**Stop**] buttons define the acquisition period. Additionally, the user can define a recording timeout using the dedicated toggle button and text field to interrupt data logging after a defined [ms] interval.

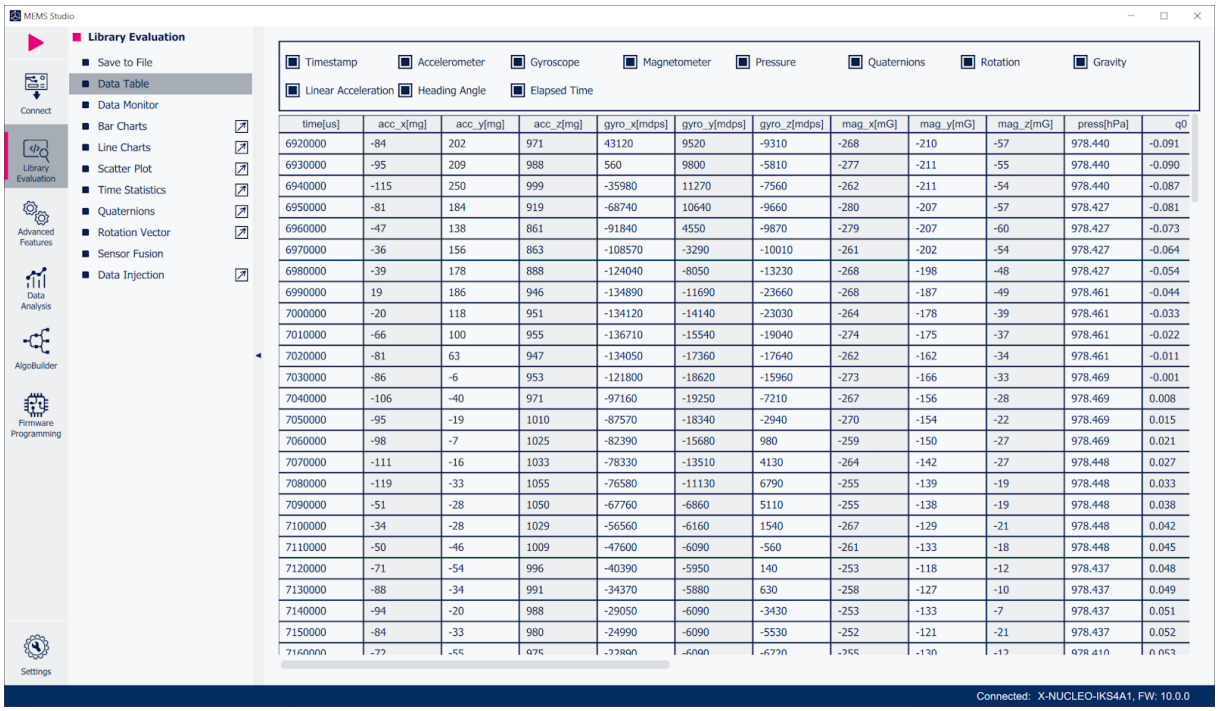
Figure 43. Save To File page



Data Table

The [**Data Table**] tool displays the output values from the sensors and libraries. This view provides an overview of all values received with a table data update rate of 0.5 Hz to reduce overall CPU/GPU consumption.

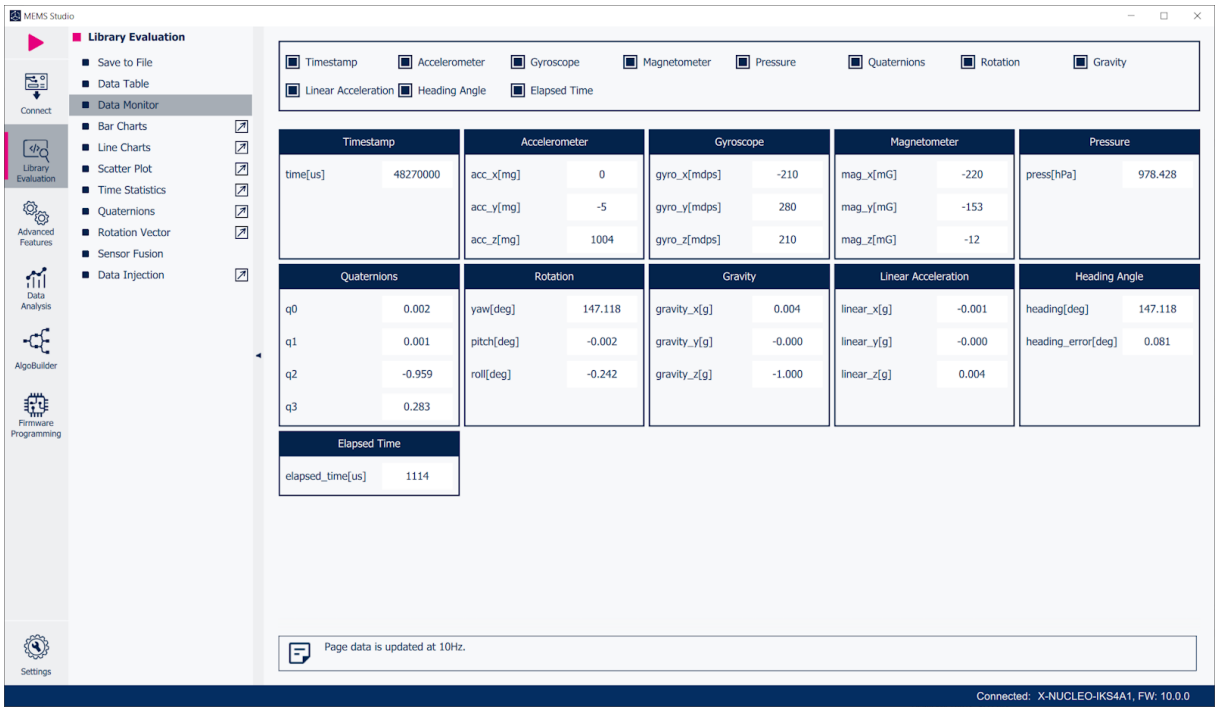
Figure 44. Data Table



Data Monitor

The [**Data Monitor**] tool displays the latest output values from the sensors and libraries. This view provides an overview of the last samples received with an update rate of 10 Hz to reduce overall CPU/ GPU consumption.

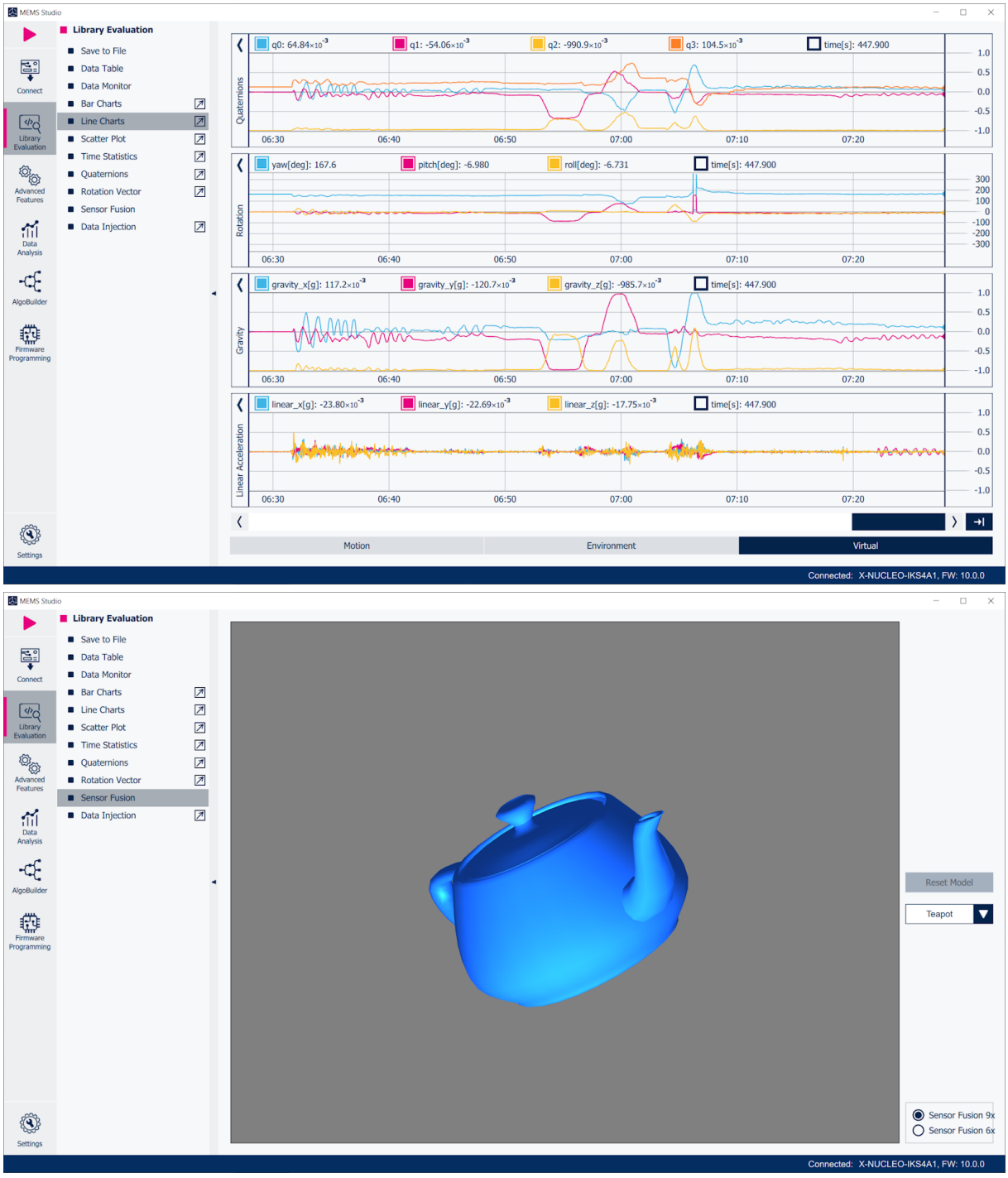
Figure 45. Data Monitor



Data visualization pages

The available data visualization pages depend on the selected library. Line charts, bar charts, and a scatter plot (for magnetometer data) are always available to visualize sensor data. Additional pages for visualization of library outputs like 3D plot, 3D model, status indicator and table, and others are displayed based on the library functionality.

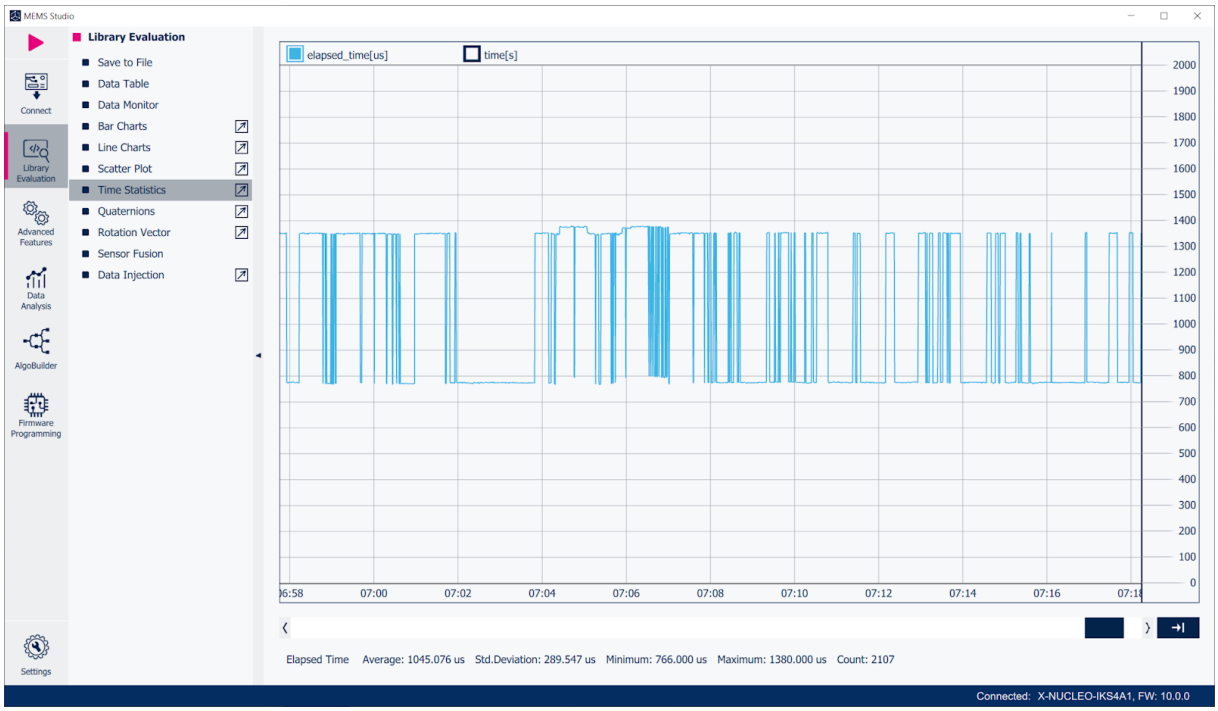
Figure 46. Example of data visualization pages



Time Statistics

The [**Time Statistics**] page displays the library elapsed time in line charts and statistical information line average, min, max values.

Figure 47. Time Statistics page



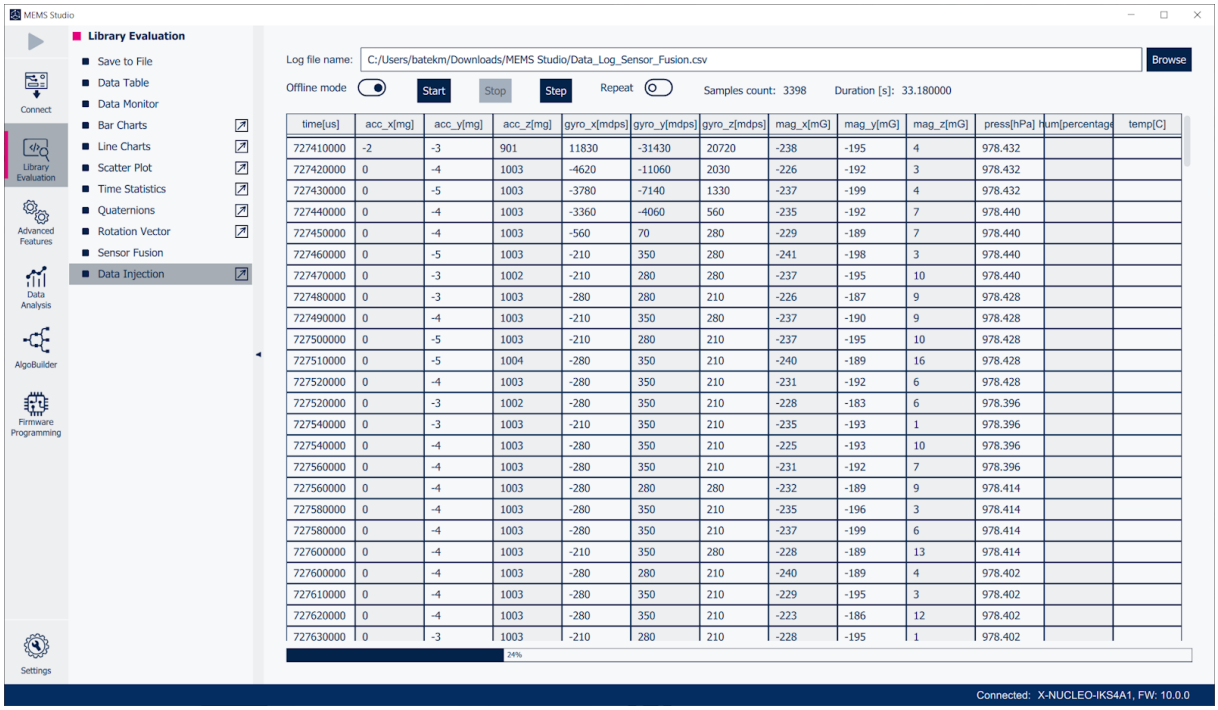
Data Injection

The [**Data Injection**] page allows injecting previously acquired sensor data into the library input instead of real-time sensor data. The library processes the data and sends the results in the same way as working with real-time sensor data. All the other features for the evaluation of the library are available and active.

The data log file with previously acquired data can be opened by clicking on the [**Browse**] button. The data must be acquired with the minimum sampling rate frequency given by the library and the library evaluation firmware. The data log with the higher sampling rate is automatically decimated. Input data are displayed in the table.

Data injection is enabled by switching the firmware to [**Offline mode**]. Data injection is controlled by the [**Start**] and [**Stop**] buttons. Injection line by line can be done using the [**Step**] button. The active line is highlighted in the table. A wraparound to move from the last sample to the first sample can be activated using the [**Repeat**] toggle switch.

Figure 48. Data injection page



5 Advanced features

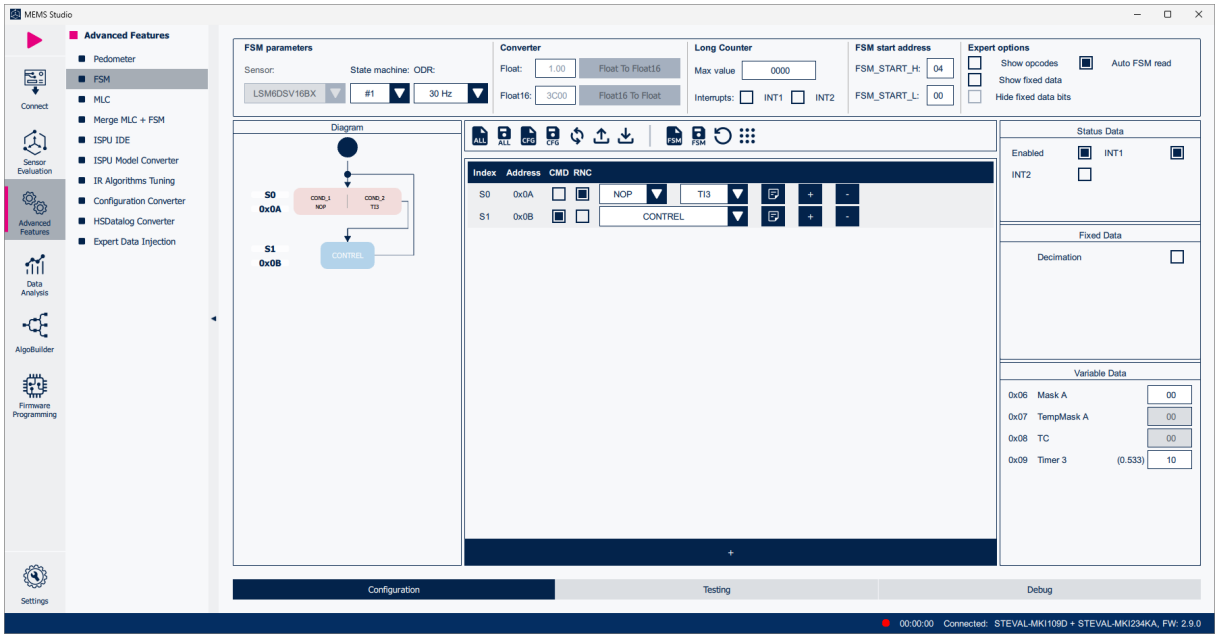
5.1 Finite state machine (FSM) tool

The finite state machine page allows the user to configure the state machines and test their functionality on the connected device in order to detect a sequence of events with specific timing (example: hand or head gesture).

FSM implementation consists of three tools:

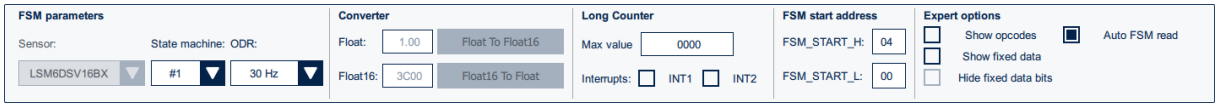
- [**Configuration**]: allows configuring the available state machines on the device
- [**Testing**]: shows plots with sensor data, interrupts, and state machine output register information
- [**Debug**]: (available only on the ProfiMEMS board). This page lets the user inject log file samples into the device in order to evaluate the number of interrupts detected and the overall FSM results based on a selected configuration.

Figure 49. FSM page



FSM parameters

Figure 50. Configuration of FSM parameters - menu bar

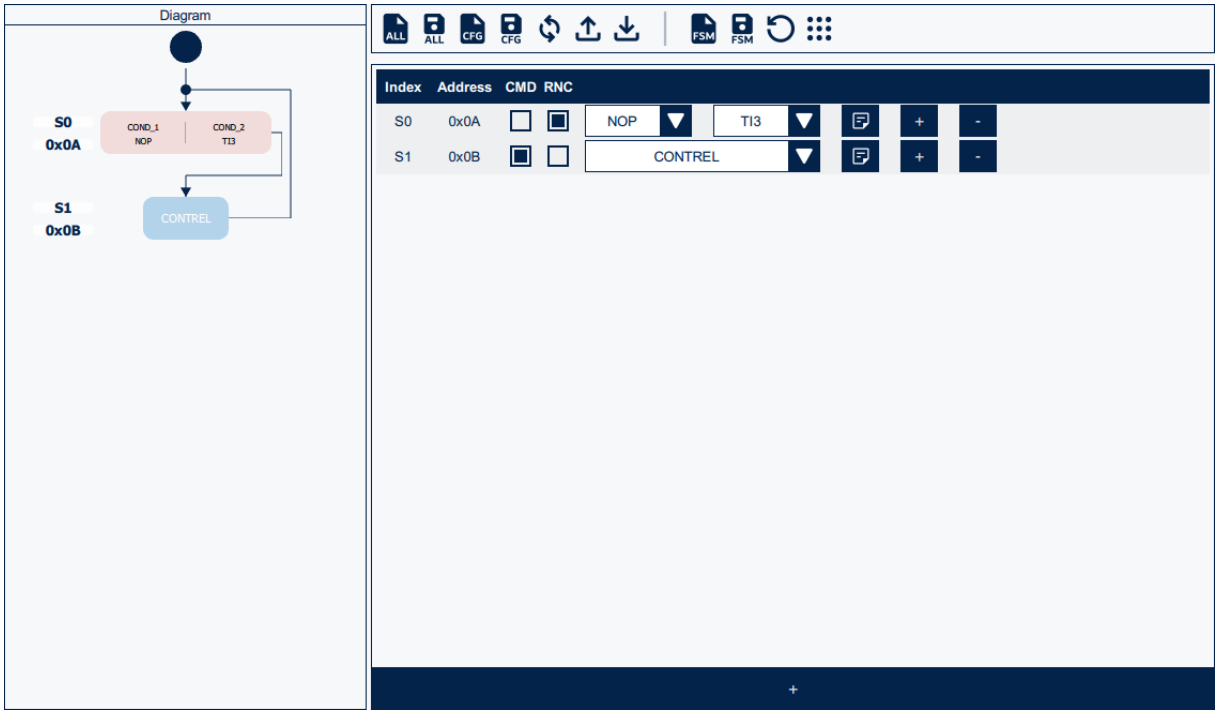


- [**Sensor selector**]: displays the selected device to configure
- [**State machine selector**]: displays the selected state machine to configure. This selection is applied to both the configuration page and debug page if available.
- [**FSM ODR**]: this setting determines the processing / output data rate of the FSM samples.
- [**Converter**]: this tool helps to generate the values to be set in the threshold resources in the [**Variable Data**] section. It converts a float32 value into a float16 format and vice versa.
- [**FSM Start Address**]: determines the first address written in the FSM registers while configuring the state machines
- **Show opcodes**: This allows the user to check the operation code corresponding to the selected command or Return Next Condition state.
- **Show fixed data**: This allows the user to enable the visualization of the FSM fixed data section parameters.
- **Hide fixed data bits**: This allows the user to expand/collapse the visualization of the FSM fixed data section parameters bit values.
- **Auto FSM read**: Automatically reads the FSM configuration from the sensor every time the user enters the FSM page.

Configuration

This tab lets the user add commands and conditions to the selected state machine and visualize diagrams while editing state parameters. A dedicated tool bar illustrates the actions that can be performed.

Figure 51. Configuration tab



The right side of the configuration tab shows the active parameters and provides the capability to edit them in order to update the values of the resources used in the [**Variable Data**] section. This section shows the active and enabled resources, depending on the list of instructions that have been set by the user in the FSM state diagram on the left side.

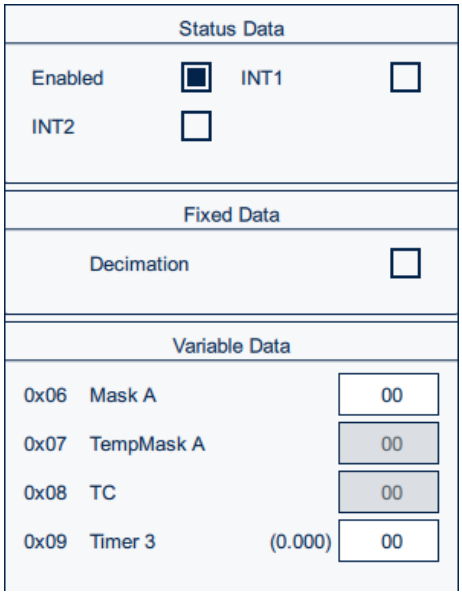


Figure 52. FSM configuration toolbar

The FSM configuration tool bar provides the capability to perform actions on all device state machines or on a single state machine. Some actions are available only if the target device is connected, such as writing or reading data to/from the connected device, while others might work in a different way such as loading an FSM configuration.

Loading an FSM configuration decodes configuration data on a host pc if no target device is connected or writes a configuration on the device and then decodes device data if the target device is connected to the MEMS Studio application in order to allow the user to inspect configuration data, ensuring the best reliability of data shown in the GUI.

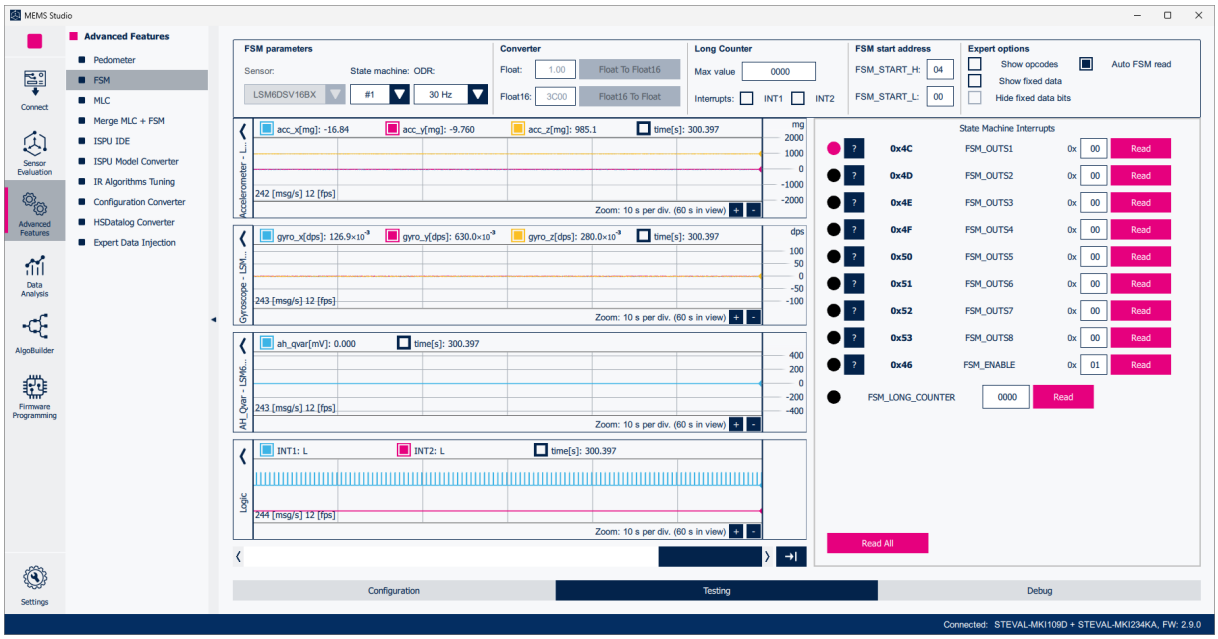
- **Load all state machines from file**: loads multiple state machines from .json file and populates the GUI
- **Save all state machine as**: exports state machines configuration to a dedicated .json file
- **Load device configuration from file**: imports all state machine configurations, writes the configuration to the device, and updates the GUI according to the written instructions
- **Save device configuration as**: exports the instructions of the device configuration for configuring all state machines to a .json file
- **Reset state machine:** clears the selected state machine GUI data without changing the device configuration
- **Read FSM configuration from sensor**: updates all state machine GUI data according to the current FSM device configuration
- **Write FSM configuration to sensor**: writes all current state machine GUI data to the device
- **Load state machine**: loads one state machine from .fsm (legacy) or .json file and populates the GUI
- **Save current FSM**: exports the current state machine configuration to a .json file
- **Reset all state machines**: clears all state machine GUI data
- **Read state machine from examples:** displays the available FSM examples for the selected device with their descriptions and allows the user to load a configuration to the GUI

Testing

This tab shows the available sensor data compatible with the FSM configuration, interrupt data plots, and FSM output register values. A dedicated blinking LED graphic is toggled every time an FSM interrupt is detected during the parsing of the FSM status register data and once it is activated, its state is not changed for ~300 msec.

This page helps the user to check the program functionalities after the FSM configuration is completed.

Figure 53. Testing tab



Debug

The **Debug** function allows debugging the selected state machine sample by sample after loading an input data pattern.

The main purpose of this tab is to check the state machine functionality in order to validate the program using recorded log files, assuming the device is configured as it was before starting to log data in the file.

During data injection, the [**Program Pointer**] and the [**Reset Pointer**] are updated according to the data that are read from the device.

The [**Program Pointer**] value contains the current address of the program and it is used to highlight the corresponding state in the diagram. The [**Reset Pointer**] value contains the address of the state or condition related to the CONTREL command or return condition of the program.

The tool provides the user with several data injection interactive controls to inject one or multiple samples at a time and write output data to a dedicated table.

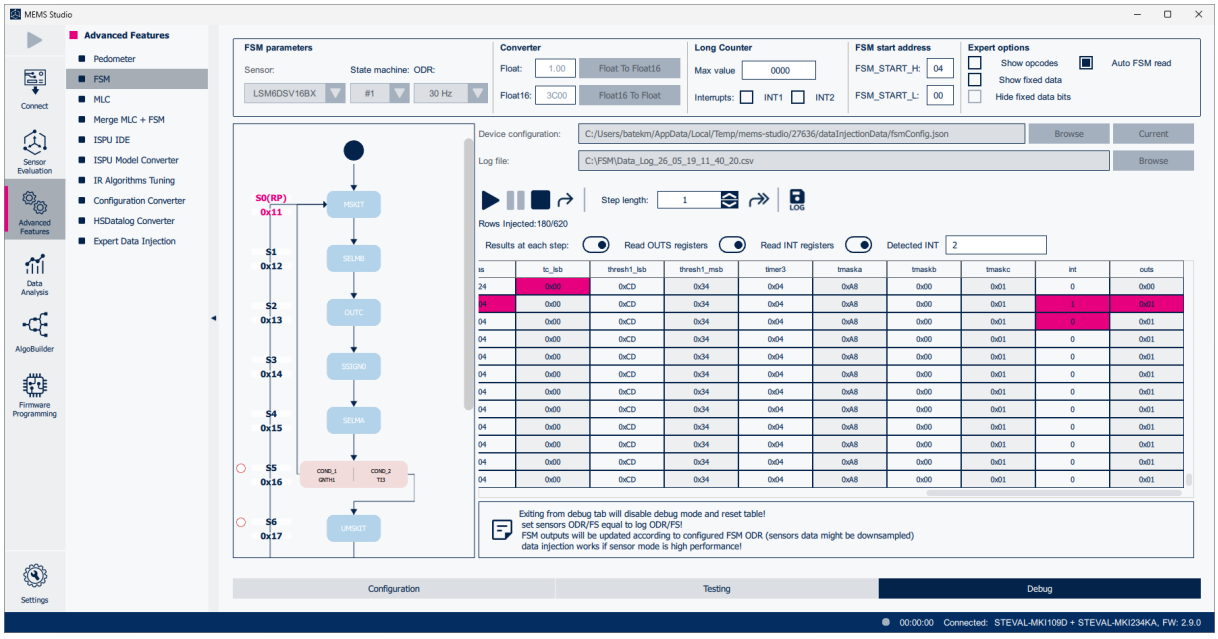
Every command before a condition and every condition has a radio button to configure a break point for pausing the data injection process if the program pointer address is equal to the address of the item with the enabled break point indicator. Whenever a break point is activated, the background color of the current state changes according to the selected color theme of the application.

Data injection controls

- [**Run**]: injects all the next samples
- [**Pause**]: pauses the data injection process
- [**Stop**]: interrupts the data injection process and exits debug mode.
- [**Next**]: injects the next sample and updates the table with output data
- [**Next step**]: injects a defined number of samples according to the [**Step length**]settings
- [**Export log**]: exports the content of the table with the results to a .csv file

The data table is updated at every step if the [**Results at each step**] switch is enabled, otherwise it is updated only when the data injection is paused or stopped. The [**Read OUTS registers**] switch enables or disables reading sensor output registers. The [**Read INT registers**] switch enables or disables reading interrupts status registers. [**Detect INT**] indicates the number of detected interrupts in the processed log file.

Figure 54. Debug tab



5.2 Machine learning core (MLC) tool

The machine learning core (MLC) tool allows the user to configure a machine learning core that is embedded in some devices and use it to perform features extraction, data classification and generate interrupts. For more details about the MLC, consult the specific application note for the sensor you would like to use. The configuration procedure is divided into several steps appearing in different tabs:

- [**Data patterns**]: allows managing the data patterns to be used and assigning a label to each data pattern loaded
- [**AFS tool**]: allows processing acquired data and recommends window length, filters, and features that can be used for decision tree generation
- [**ARFF generation**]: allows configuring inputs, filters, and features to generate an .arff file
- [**Decision tree generation**]: allows configuring the decision tree outputs and generating the decision trees
- [**Config generation**]: allows setting the metaclassifier and generating the configuration file
- [**Decision tree output viewer**]: allows live monitoring of MLC outputs in the chart when testing the generated MLC configuration
- [**Debug**]: allows testing the MLC configuration using previously acquired data

To start configuring the machine learning core, the workspace directory must be selected using the [**Browse**] button. After that a device must be selected. If a sensor using the MLC is connected, it is selected automatically. The entire MLC configuration is stored in the mlc_settings.json file inside the workspace directory. The changes are automatically saved to this file. The MLC configuration can be imported into the active workspace using the [**Import**] button. The [**Import**] dialog allows user to import both configuration file (*.json) and MLC project (*.zip) . While importing an existing project, ensure that files are stored according to following structure: "model" directory contains a configuration file named "configuration.json", a device configuration file named "sensor_name_mlc.json" and "sensor_name_mlc.h", if needed a "features.arff" file and decision trees .txt files. The "datalogs" folder should contain log files used to generate the .arff file.

All artifacts generated during the MLC configuration process can be deleted using the [**Clear Workspace**] button, and the mlc_settings.json is not deleted. All the settings can be deleted and restored to the default values using the [**Clear Settings**] button. The history of the performed actions and results are accessible in a dedicated window, which can be displayed using the [**Open Log**] button. The [**Export Project**] button allows user to export the entire MLC project according to the upper mentioned data structure.

MEMS Studio and ST AIoT Craft interoperability

ST AIoT Craft and MEMS Studio provide a two-way workflow for MLC projects. You can export an MLC project from one tool and import it into the other, making it easier to move between cloud-based project management and desktop-based development.

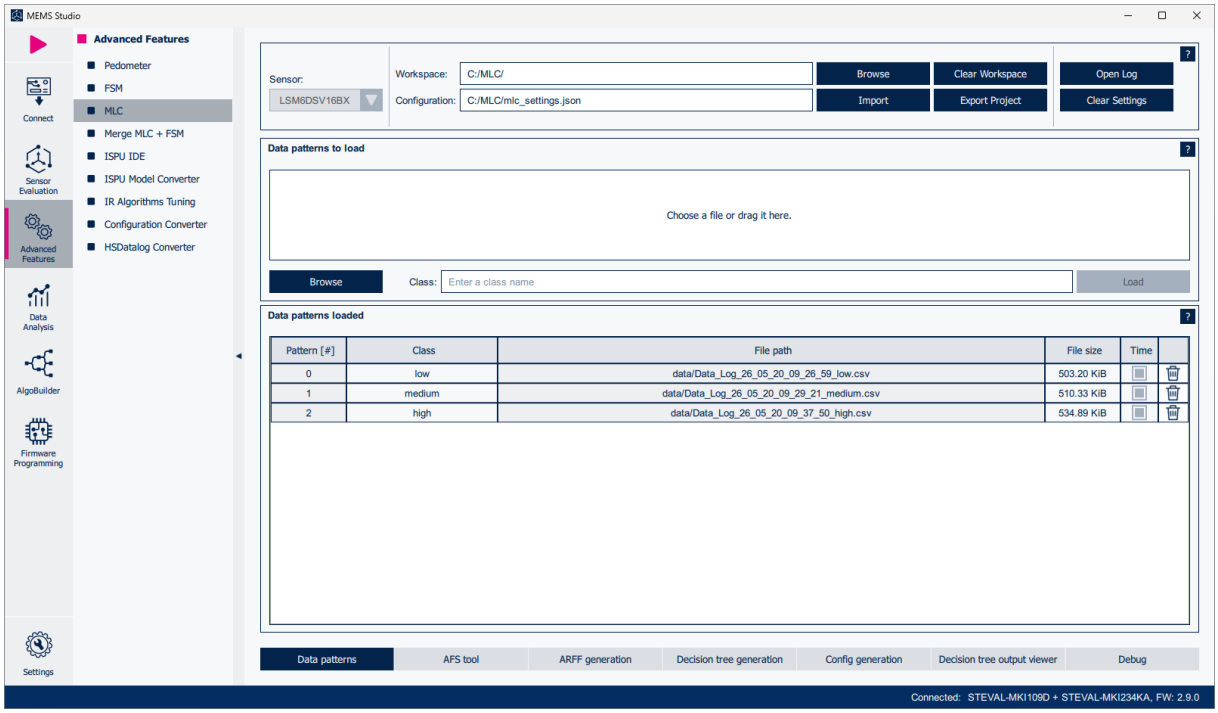
This interoperability helps you choose the most suitable environment for each stage of development. For example, you can manage projects and collaborate in ST AIoT Craft, then continue local validation or experimentation in MEMS Studio. You can also bring an existing MEMS Studio project into ST AIoT Craft and continue working with the cloud-based features available there.

Data patterns

The [**Data patterns**] tab allows managing the data patterns to be used for the machine learning processing. The data patterns that are possible to load must have the same data format as the log files generated by MEMS Studio, Unico-GUI, or Unicleo-GUI.

A data pattern(s) is selected using the [**Browse**] button. Multiple files can be selected. A class label must be assigned to each data pattern, which is then used for machine learning processing. The class label is confirmed by the [**Load**] button. Data patterns can be removed from the list using the [**trash bin**] button in the tables.

Figure 55. Data patterns tab



AFS tool

The AFS tool offers the possibility to estimate suitable window length, filters, and features for classification done by the decision tree.

Automatic window length calculation

The window length is calculated based on the lowest frequency present in the signal across all data classes.

Automatic filter selection

There are two methods that can be used for automatic filter selection. **Basic search** is a peak-based method that identifies the maxima of the signal in the frequency spectrum. This method is less computationally expensive, but it is not accurate on flat signals without significant peaks. **Exhaustive search** is an entropy-based method that searches all combinations of frequency bands (on a logarithmic scale) and selects the frequency of interest that minimizes the entropy against the other classes.

Automatic feature selection

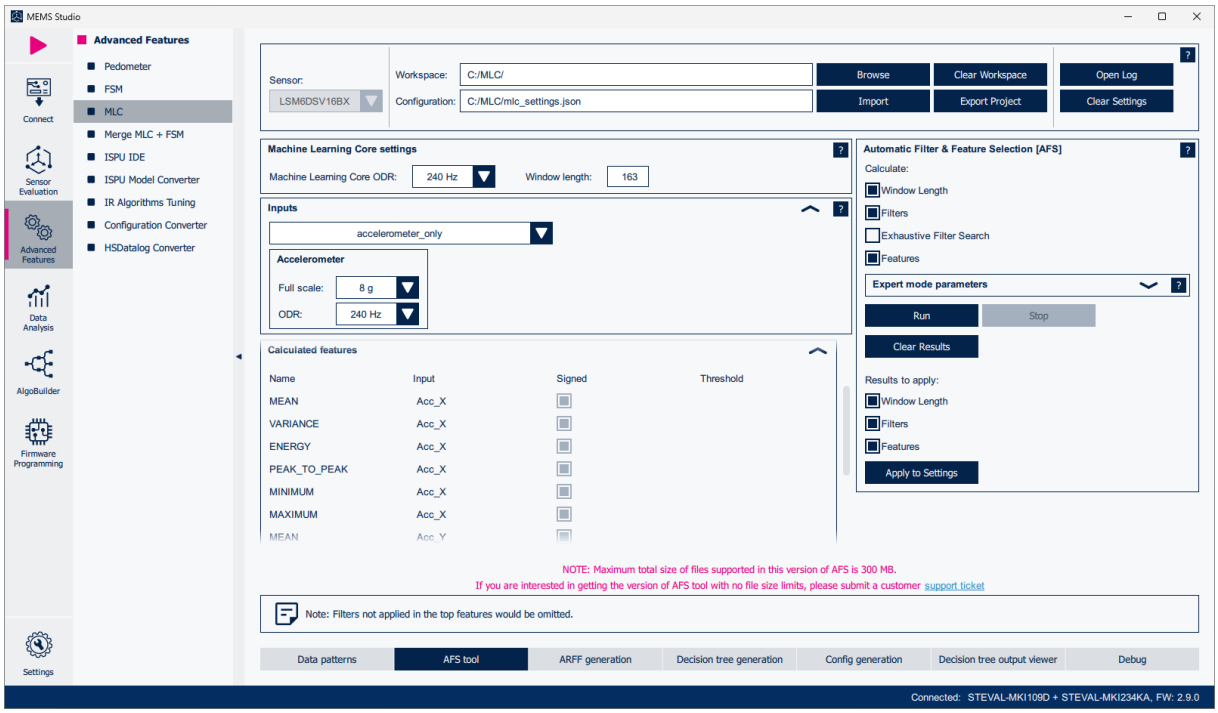
The automatic feature selection algorithm considers filters generated by the automatic filter selection and the features are computed for different filters and raw data. It uses an ensemble method for feature importance and cross correlation to remove highly correlated features. The following statistical features are supported for both single axis and norm: mean, variance, energy, peak to peak, minimum, maximum. The automatic feature selection algorithm selects the top features by computing the rank of each feature according to an ensemble method and averaging across all ensemble methods. There are five different ensemble methods available to determine feature importance: ANOVA, Ada Boost (ADA), Sequential Feature Selection (SFS), Random Forest (RF), Recursive Feature Elimination (RFE). By default, ANOVA, Ada Boost, Random Forest, and Recursive Feature Elimination are used as methods for ranking important features. Including Sequential Feature Selection may improve the overall performance but increases the overall runtime.

Data analysis takes into account the machine learning core ODR and input specification parameters from the [**ARFF generation**] tab. The window length is also taken from this tab in case the window length calculation is not enabled.

After the desired parameter selection, the data analysis is executed using the [**Run**] button.

The generated window length, filter, and features can be transferred to the MLC configuration in the [**ARFF generation**] tab using the [**Apply to Settings**] button.

Figure 56. AFS tool tab

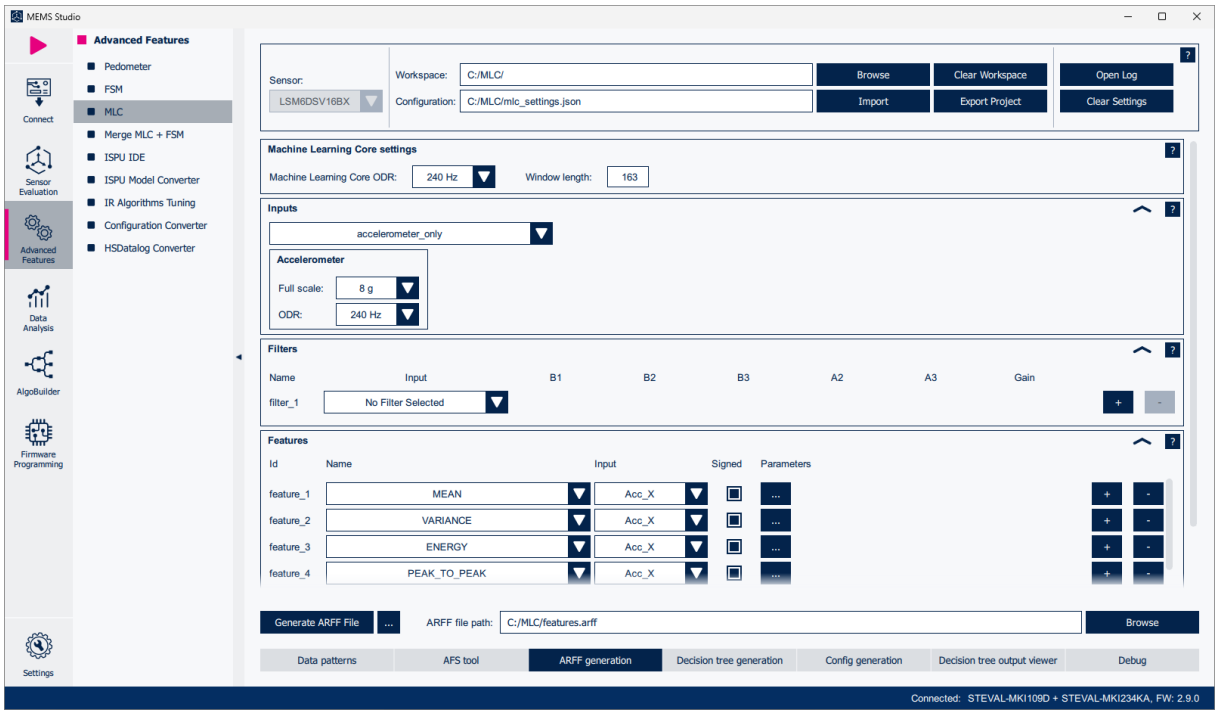


ARFF generation

Attribute-Relation File Format (ARFF) is a file format used for storing data in machine learning applications. It contains a list of instances, where each instance represents a single window processed by the MLC each row of this file contains the values of features evaluated for every instance and corresponding class assigned to the specific data processed.

The [**ARFF generation**] tab allows configuring sensor and MLC parameters. The MLC processes the data at the [**Machine Learning Core ODR**] rate in consequential data windows. The number of samples in each window can be configured by the [**Window length**] option. The MLC processes the sensor data specified in the [**Inputs**] section. The full scale and output data rate of the selected sensors can be defined. [**Filters**] offer preprocessing on the machine learning core input signals. The [**Features**] section contains a list of features that are calculated from the input data (filtered or not filtered). After setting all the parameters, the ARFF file can be generated using the [**Generate ARFF File**] button. Statistical parameters computed from the inputs are then used for classification in the decision tree.

Figure 57. ARFF generation tab



Decision tree generation

The [**Decision tree generation**] tab allows configuring a decision tree using selected classes to process and evaluating decision tree classification performance. The [**Number of decision trees**] can be specified.

The decision tree algorithm has both depth-first and best-first tree building implementations, implements Breiman's cost-complexity pruning, and identifies optimal tree size using cross validation.

The maximum depth of the tree ([**Max depth**]), minimum samples required in a leaf node ([**Min leaf samples**]), and [**Percentage of data to use for training**] can be configured.

A decision tree is generated using the [**Generate Decision Tree**] button. A file with an externally generated decision tree definition can be loaded using the [**Import**] button. After decision tree generation or loading, statistical parameters of the decision tree like accuracy, number of instances, confusion matrix, and others are displayed. The resulting decision tree can be analyzed using the [**Inspect tree info**] button for a textual content or using [**Show Decision Tree**] for a graphical representation of the same.

If the user generates a new decision tree selecting a reduced number of classes to be used compared to the one available into the selected .arff file, an intermediate .arff file will be generated by MEMS Studio into the workspace in order to exclude the classes not selected from the decision tree training process. This file is an intermediate file managed by MEMS Studio only and will be named as "features_DT(n).arff" where (n) represents the id of the selected decision tree.

Figure 58. Decision tree generation tab

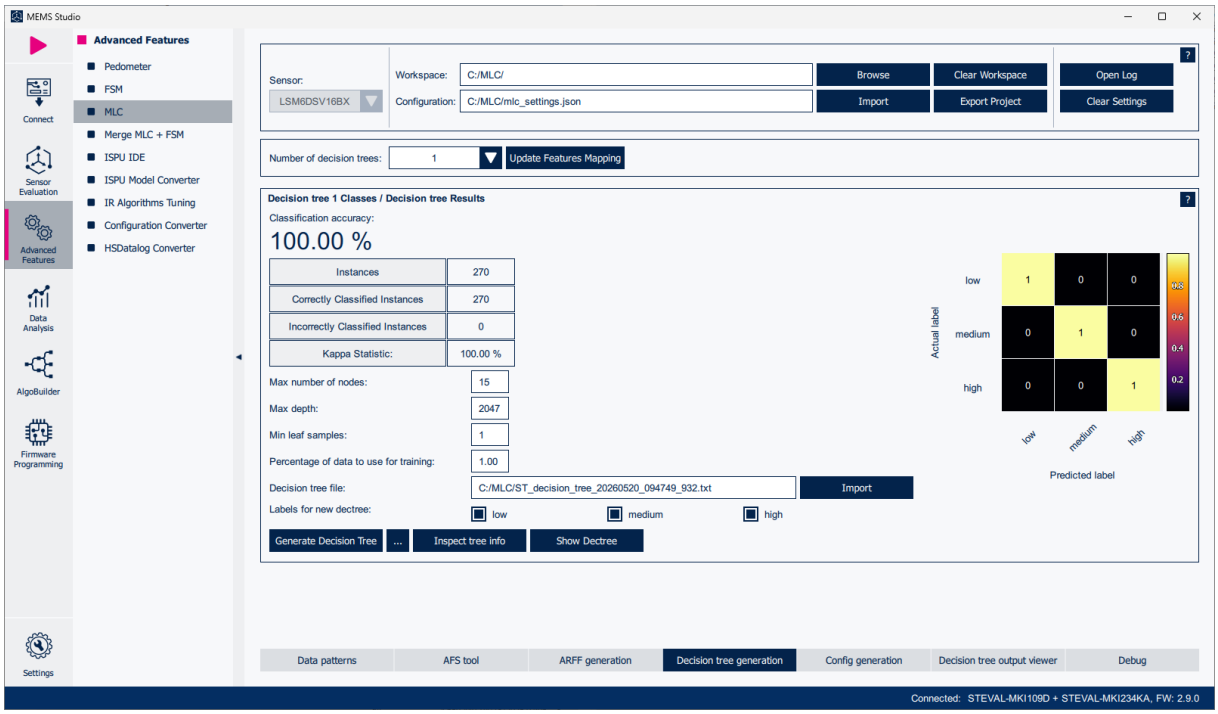


Figure 59. **Inspect tree info (textual content)**

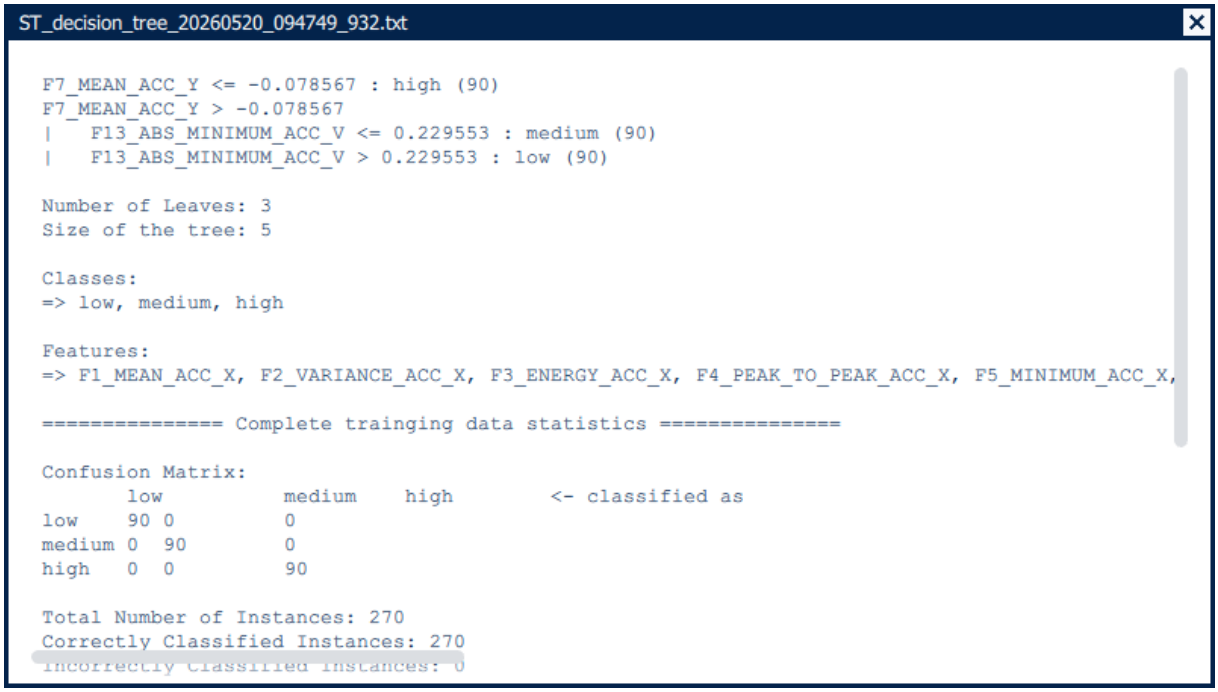
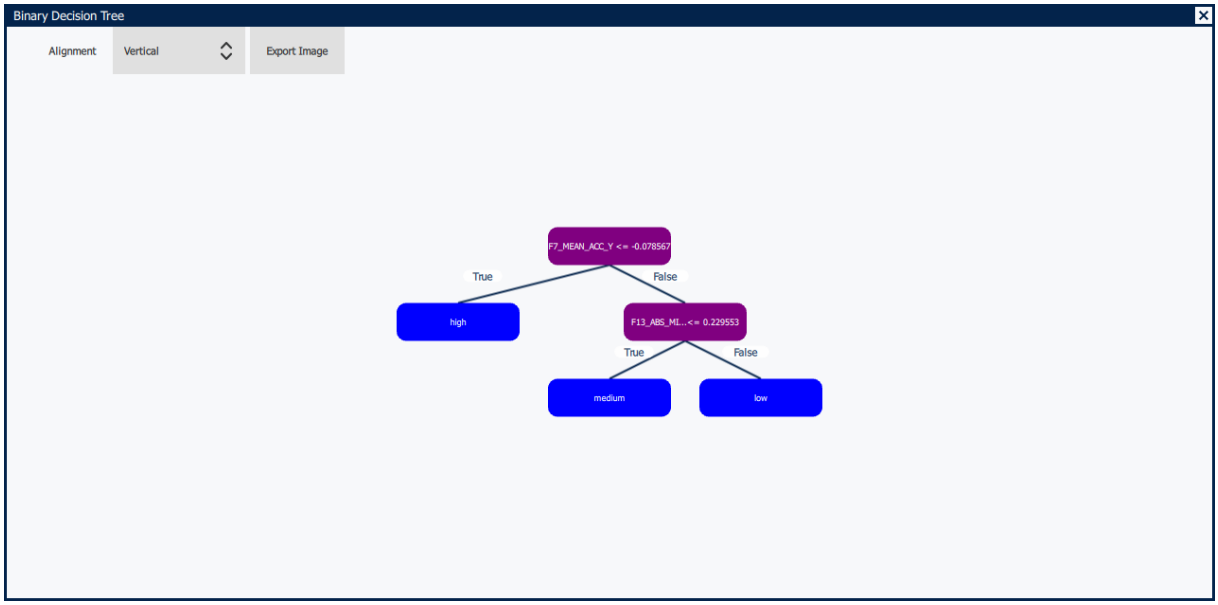
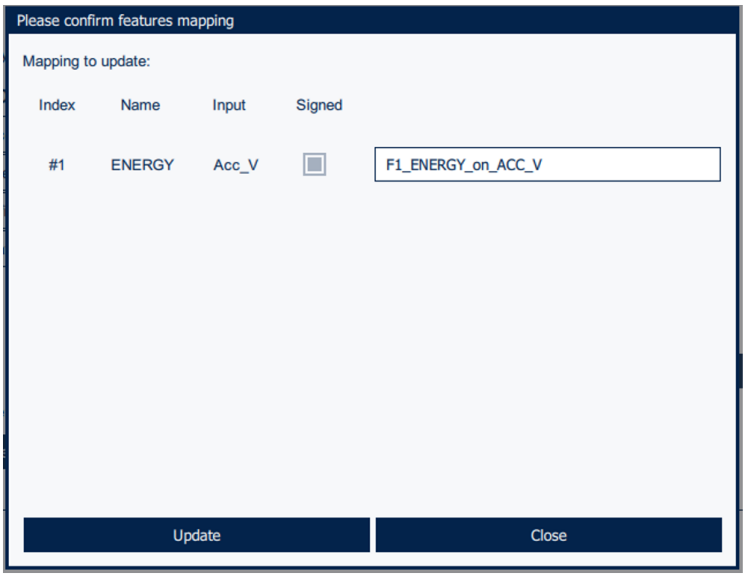


Figure 60. **Decision tree (graphical representation)**



If the user imports a decision tree, a feature mapping is needed to map every feature selected during arff generation with feature listed into decision tree file every feature selected during arff generation must be mapped, listing each feature in the decision tree file. Note that this mapping is applied for every decision tree imported or generated.

Figure 61. **Confirmation of features mapping**



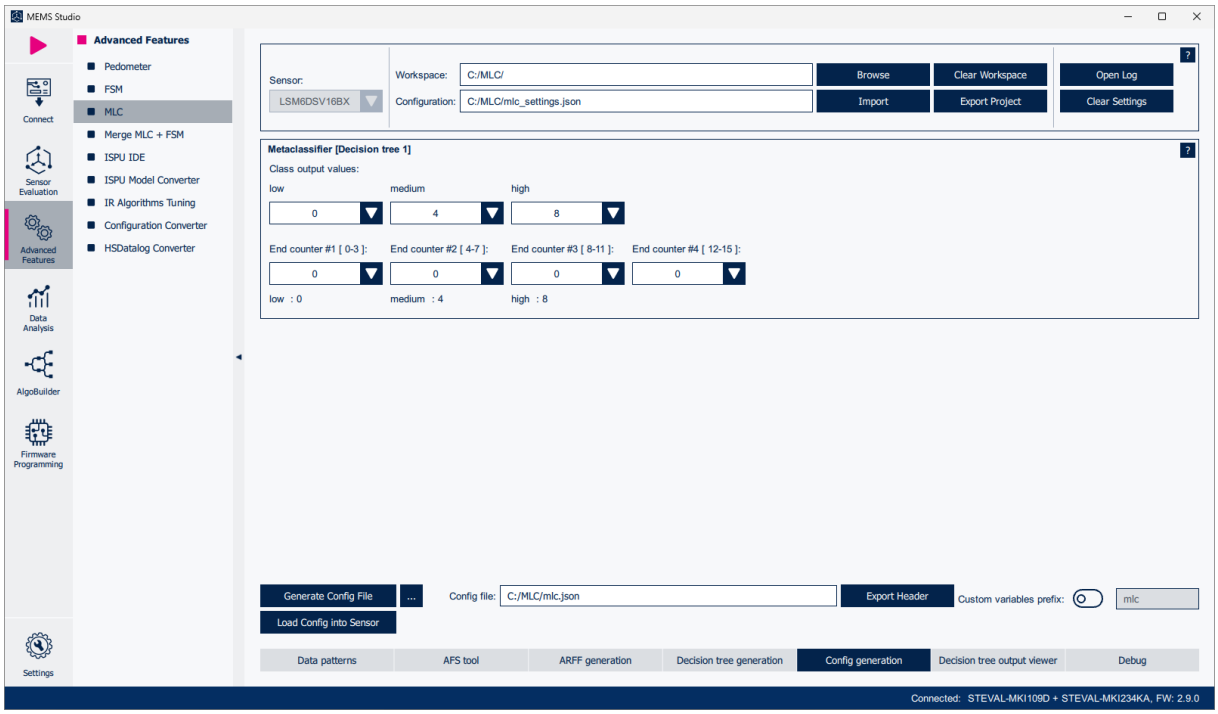
Config generation

The [**Config generation**] tab allows configuring the metaclassifiers and generating the .json file containing the machine learning core configuration.

The generated MLC configuration can be stored also in .h file using the [**Export As**] button.

In the case of a board with a selected sensor connected, the generated configuration can be uploaded to the sensor using the [**Load Config into Sensor**] button.

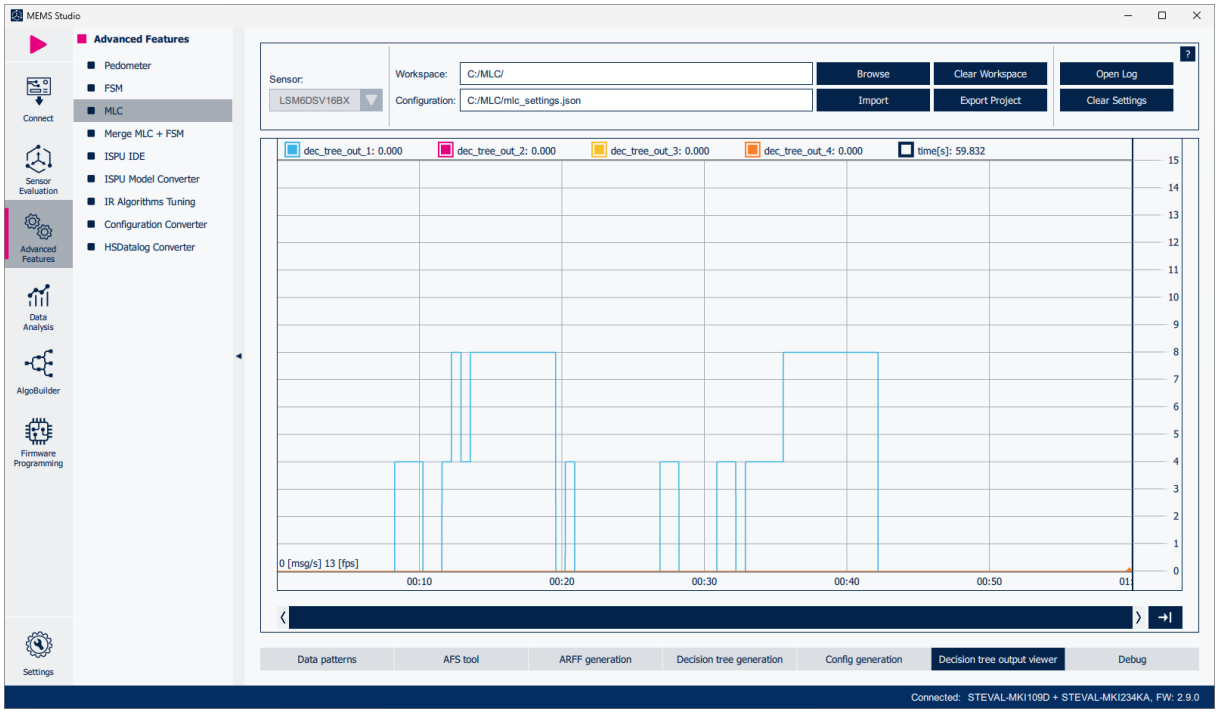
Figure 62. **Config generation tab**



Decision tree output viewer

The [**Decision tree output viewer**] tab allows testing the generated MLC configuration by monitoring individual MLC outputs in the chart.

Figure 63. Decision tree output viewer tab



Debug

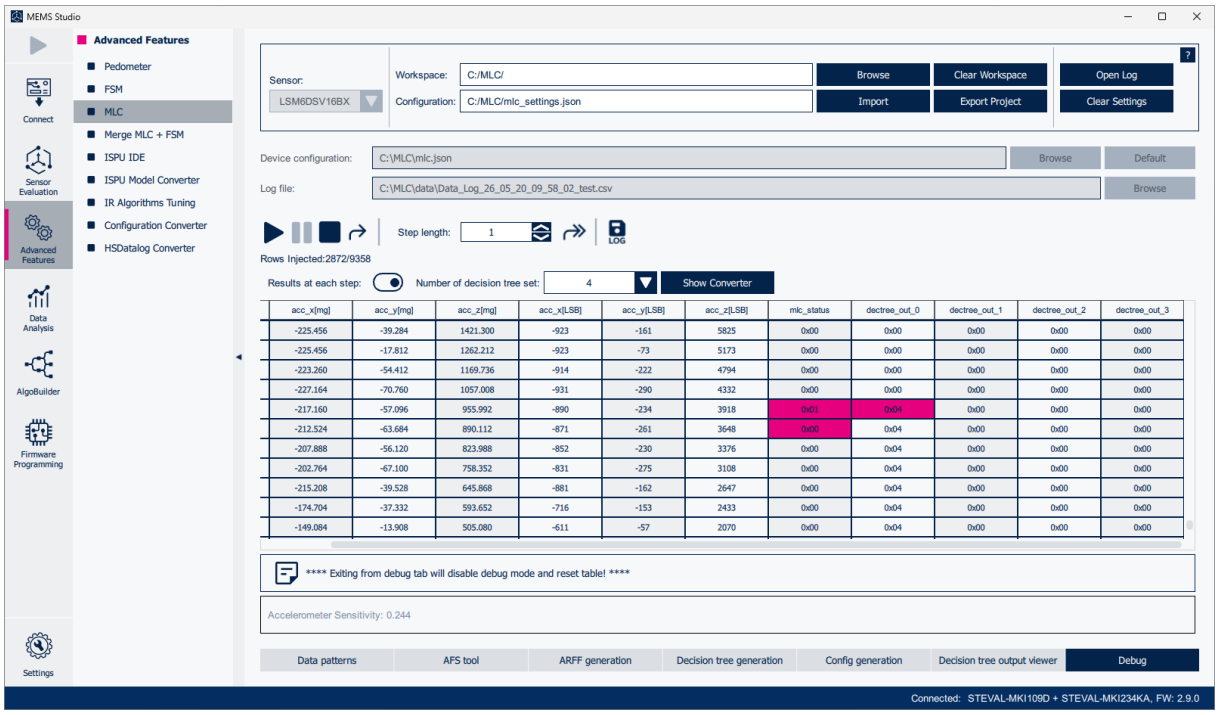
The **Debug** tab allows checking the MLC functionality in order to validate the configuration using previously recorded log files, assuming the device is configured as it was before starting to log data in the file.

Data injection control

- [**Run**]: injects all the next samples
- [**Pause**]: pauses the data injection process with the possibility to resume it
- [**Stop**]: interrupts the data injection process and exits debug mode.
- [**Next**]: injects the next sample and updates the table with output data
- [**Next step**]: injects a defined number of samples according to the [**Step length**]settings
- [**Export log**]: exports the content of the table with the results to a .csv file

The data table is updated at every step if the [**Results at each step**] switch is enabled, otherwise it is updated only when the data injection is paused or stopped. The [**Number of decision tree set**] settings specifies which MLC output registers will be monitored.

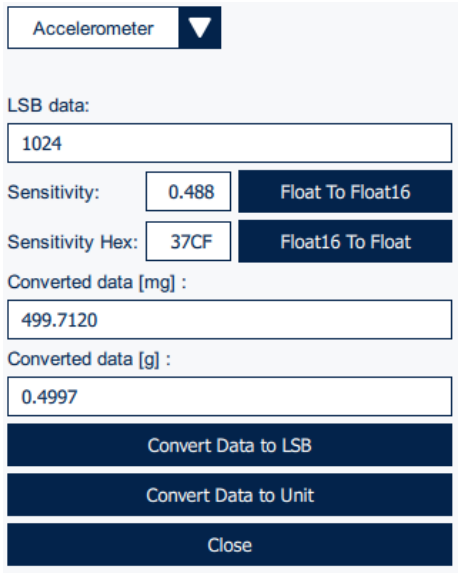
Figure 64. Data injection



The debug page provides the user with a dedicated converter tool to simplify data conversion during log data processing. The tool can be opened using the [**Show Converter**] button. The debug tool works with raw data in LSB and it converts sensor data internally into LSB using the sensitivity evaluated on the basis of a given configuration file if no LSB data was given.

The MLC in the hardware configures conditions in the units of measurement, which is why users might need a converter tool to easily understand if a certain condition of a decision tree is verified for a certain LSB data. For example if a decision tree condition has a threshold of 0.5 *g*, the corresponding LSB threshold value will be 1024 using a sensitivity of 0.488.

Figure 65. **Converter tool**



5.3 Merge MLC + FSM tool

Merging MLC and FSM configurations requires a special procedure. The [**Merge MLC + FSM**] tool can be used to perform combinations of separate MLC and FSM configuration files into one file.

Note: The tool acs on the MLC and FSM configuration only. The user must ensure the two configurations are compatible in terms of general sensor configuration like output data rate or full-scale settings.

Figure 66. Merge MLC + FSM page

5.4 Pedometer

The [**Pedometer**] tool can be used to configure and test the pedometer embedded in the device when this feature is supported by the sensor (see the device datasheet for more details).

There are three different tabs for this tool:

- [**Configuration**] tab: allows fast-configuring the pedometer with its default configuration
- [**Debug**] tab: used to load a data pattern into the device in order to test the pedometer on the pattern loaded
- [**Regression Tool**] tab: allows finding an optimal configuration based on a predefined dataset

*Note: Pedometer [**Configuration**] and [**Debug**] pages are available only on ProfiMEMS board if device with pedometer support is connected.*

Configuration

The [**Configuration**] tab allows fast-configuring the pedometer. In this window, it is possible to visualize in real time the evolution of both the accelerometer signal and the two interrupt lines and to read the pedometer step count output.

The user can enable and configure the pedometer with its default configuration using a dedicated button.

The GUI also allows directly accessing the pedometer-related registers in order to set a user-defined pedometer configuration.

Figure 67. Configuration tab

Debug

The [**Debug**] tab is used to load a data pattern into the device and run the pedometer logic on the pattern itself (offline postprocessing).

On the right side of the GUI, a three-button toolbar is available for loading a data pattern in the GUI. Pedometer configuration can be loaded from a .ucf or .json file. It is also possible to fast-configure the pedometer in its default configuration by clicking on the [**Pedometer Easy Configuration**] button.

The device enters debug mode right after the data pattern has been loaded.

On the top of the GUI, a toolbar is available to control the data injection, which can be applied with the maximum level of flexibility:

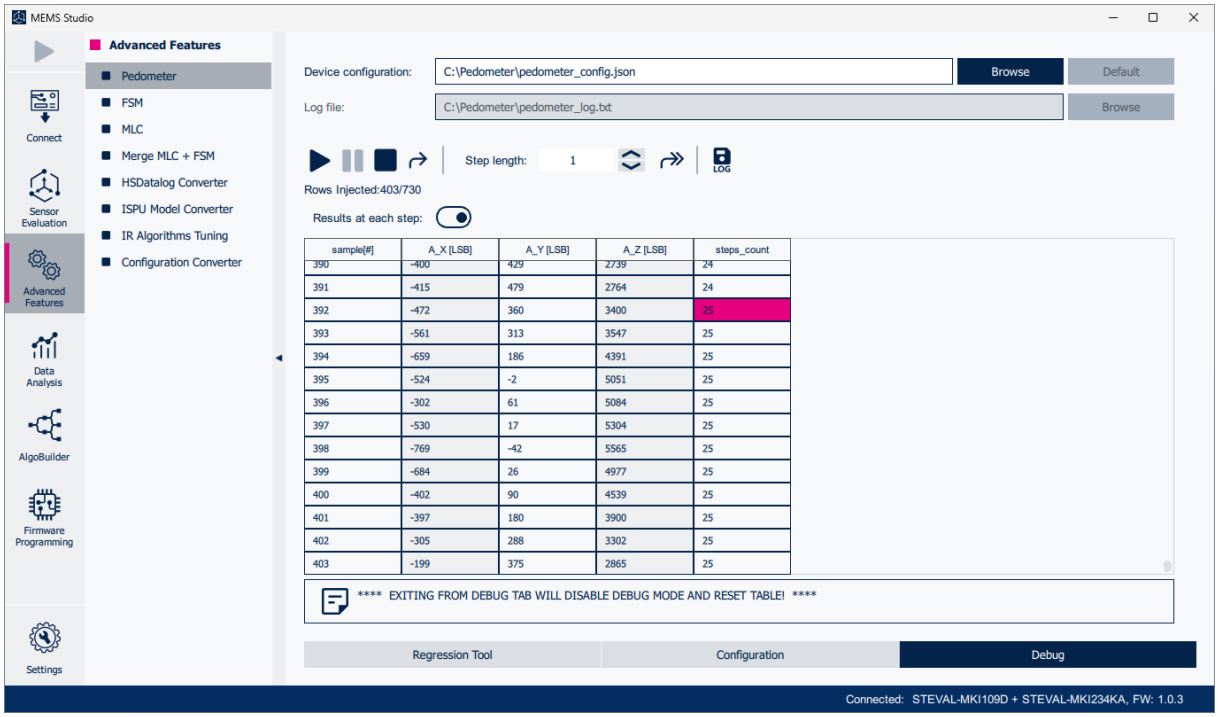
Data injection controls:

- [**Run**]: injects all samples
- [**Pause**]: pauses the data injection process with the possibility to resume it
- [**Stop**]: interrupts the data injection process and exits debug mode
- [**Next**]: injects the next sample and updates the table with the output data
- [**Next step**]: injects a defined number of samples according to the [**Step length**]settings
- [**Export log**]: exports the content of the table with the results to a .csv file

The data table is updated at every step if the [**Results at each step**] switch is enabled, otherwise it is updated only when the data injection is paused or stopped.

Once the debug session is completed, another debug session can be started (the data pattern must be reloaded).

Figure 68. Debug tab



Regression Tool

The [**Regression Tool**] tab allows finding an optimal configuration on the basis of a predefined dataset (in this context, a dataset is a collection of data patterns with a reference number of steps for each pattern).

On the right side of the GUI, the initialization view contains the two initialization parameters:

- [**Error type**]: the error to be minimized by the tool (that is, mean, mean plus standard deviation, mean plus two times the standard deviation)
- [**Complexity level**]: the level of depth (in terms of iterations) of the regression. The execution time of a low-complexity level regression is lower than a high-complexity level, but the parameter space is less deeply explored.

At the top of the GUI, the run view contains the textbox for the input data path and the output file path.

Using dedicated [**Start**] and [**Stop**] buttons, the user can run the regression. A progress bar is available in order to notify the user about the progress of the regression.

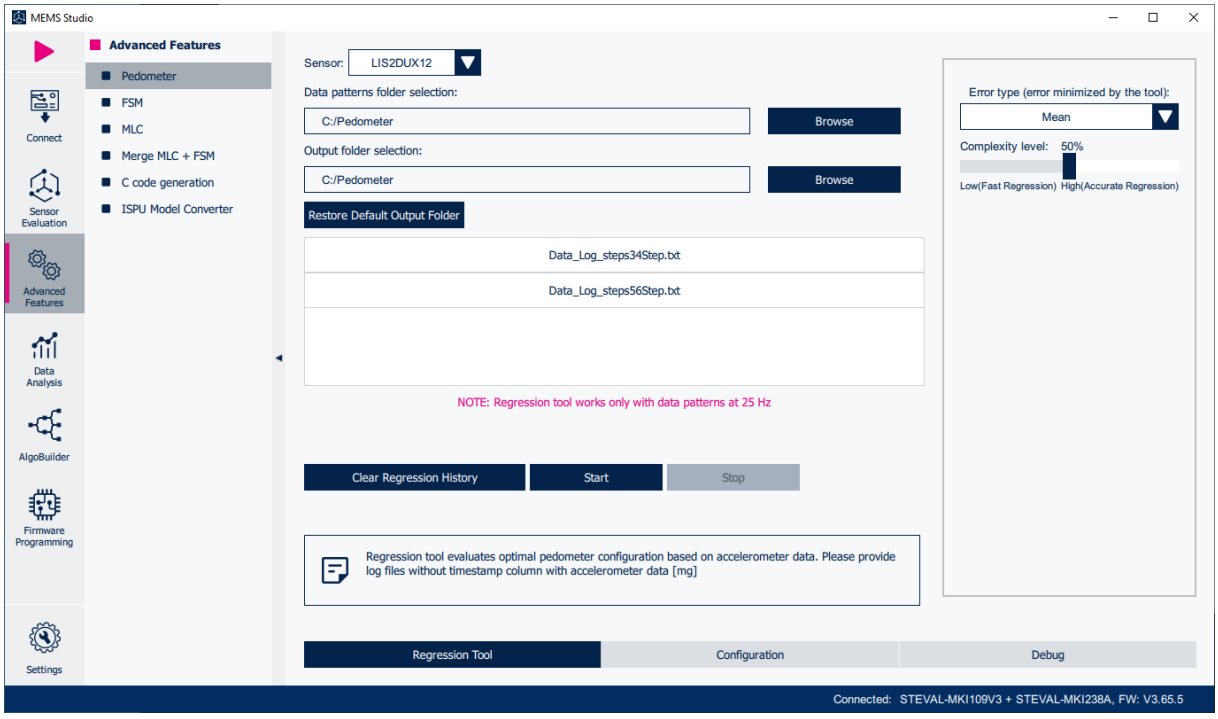
Note: The input data path to be specified in the textbox must be a folder containing only the pedometer data patterns to be applied to the regression session. Each file must contain accelerometer data in a tab-separated, space-separated, or comma-separated format in [mg] and collected at the proper output data rate according to the pedometer description in the datasheet. Each filename must contain the '_stepsXXX' string, where XXX is the effective reference number of steps (that is, test001_steps100.txt, data_steps54.txt, pattern_steps1043.txt, or any number of steps greater than 0). Log files with invalid file name or 0 steps will be discarded.

Once the regression has been completed, the tool generates a folder under the output folder, named [data]_[hour]_Pedometer, containing two files:

- pedometer.ucf, containing the optimal pedometer configuration generated by the tool
- regression_tool.config, containing some metainformation used by the [**Regression Tool**] itself and statistical data about the pedometer performance on the input dataset

Under the output folder, another regression_tool.config file is generated. This file is automatically used by the [**Regression Tool**] in order to start a new regression analysis from the optimal configuration found in the latest regression executed. If the user's intention is to run a new regression from scratch (starting from the default pedometer configuration), the user should click on the [**Clear Regression History**] button.

Figure 69. Regression Tool tab



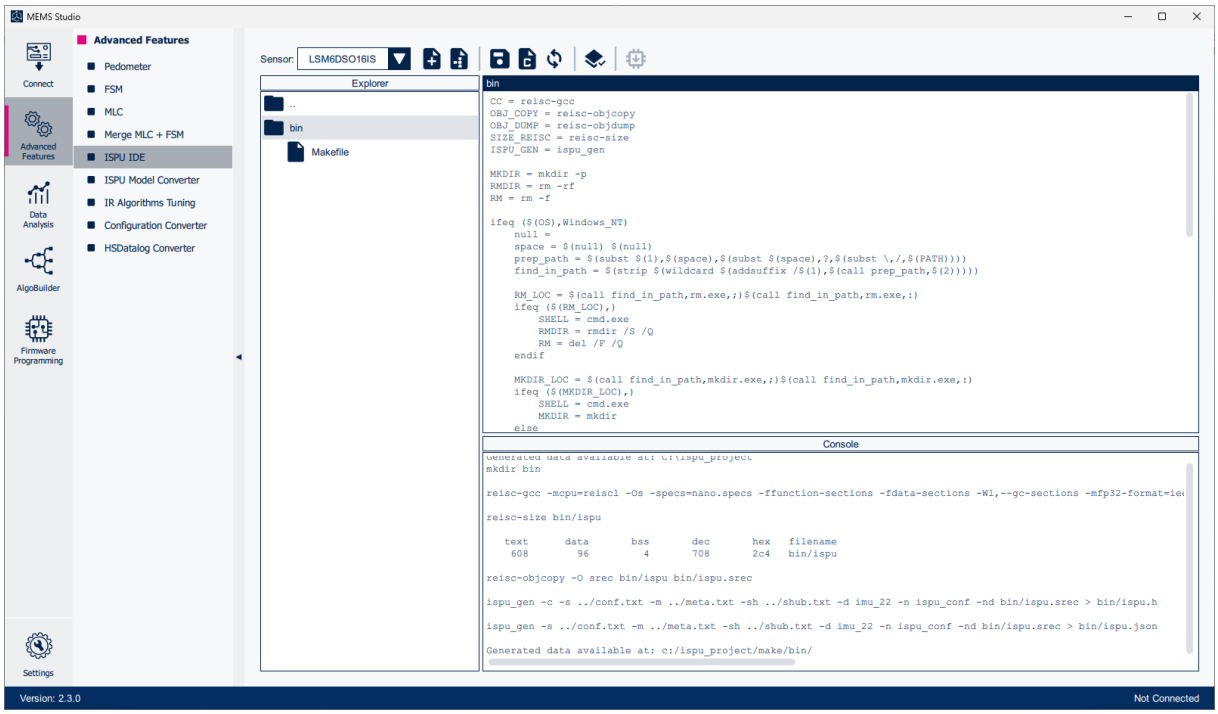
5.5 ISPU IDE

The ISPU IDE page is an easy-to-use development environment intended for creating ISPU projects, modifying source code, compiling projects, and programming the ISPU in the sensors.

*Important: To fully utilize this page, the paths to the ISPU toolchain and Make utility need to be set on the [**External Tools**] page in the application [**Settings**].*

The ISPU toolchain is a set of tools that compiles source code written in C into binary format, and then into a header file and a JSON file containing a sequence of operations required to program the ISPU in the sensor. The compilation and conversion process is invoked using the Make utility.

Figure 70. ISPU IDE page



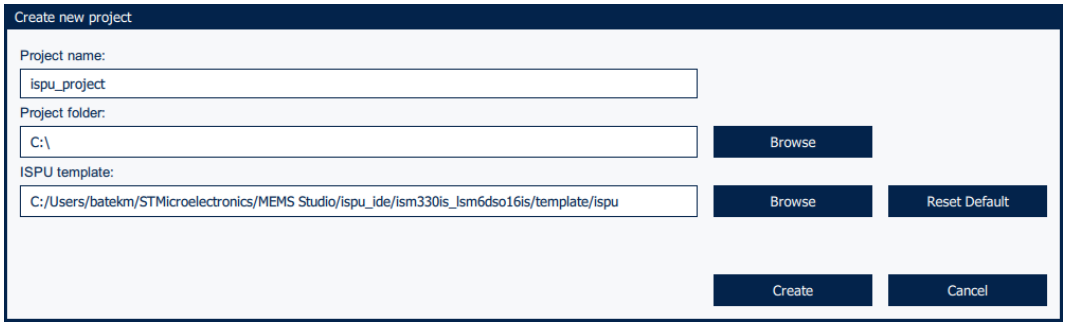
The ISPU IDE can be controlled using the toolbar.

Table 7. ISPU IDE toolbar functions

Toolbar icon	Function
	Create new ISPU project
	Open directory with ISPU project
	Save changes
	Open file in system default text editor
	Reload file
	Build ISPU project
	Program sensor

An ISPU project needs to be created starting from an ISPU template. The ISPU template is a set of files that can be modified by the user to program ISPU functionality and its content depends on the sensor selected using the dedicated combo box. The [**Create new Project**] dialog window opens by clicking on the  icon. In this dialog, the Project name and Project folder need to be defined, as well as the ISPU template location. The template location is predefined to the default folder where the template is distributed with the MEMS Studio installer.

Figure 71. Create new project



The ISPU IDE page is divided into three sections. The Explorer section allows navigation through the folder structure to open files, which can then be edited in the File section. The Console section displays output from the build process.

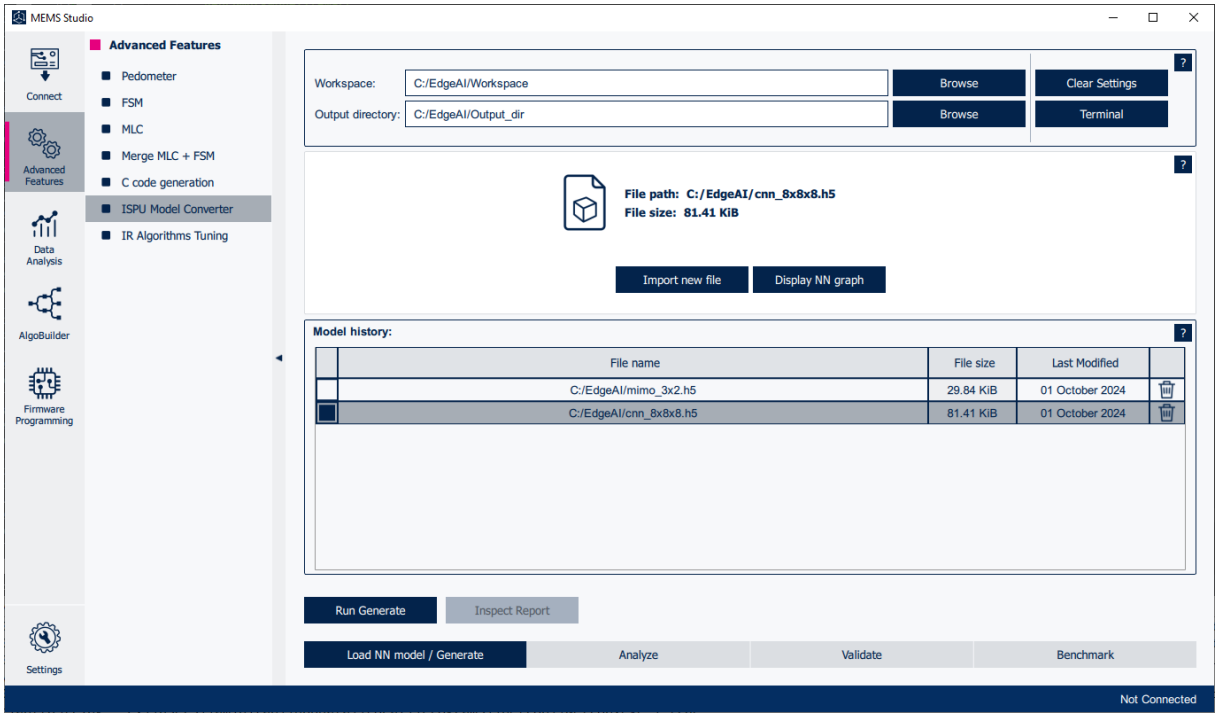
The build process can be executed only if a makefile is available in the project folder. Once the project is built, if the board and device are connected, the configuration file generated can be loaded by clicking on the  icon, selecting any file or folder in the project directory.

5.6 ISPU Model Converter

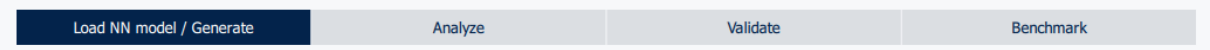
The [**ISPU Model Converter**] page allows importing pretrained neural network (NN) models from popular ML frameworks. Users can analyze these models in detail, optimize and convert them to .c / .h files, and finally validate the converted model (on the host PC or target using dedicated firmware).

The mentioned .c / .h files might then be included in a custom project (template available in the ISPU GitHub repository in order to build a binary to run the neural network on the ISPU-equipped sensor.

Figure 72. ISPU Model Converter page



There are four different tabs for this tool:



- [**Load NN Model / Generate**]: loads the pretrained neural network model in the application, generates the .c / .h files and a "Generate" textual report related to the NN model conversion process
- [**Analyze**]: provides information about the NN model architecture, its memory footprint, and computational complexity. It also generates an “Analyze” textual report from the NN model and represents data in table and radar charts.
- [**Validate**]: runs the original and the converted models and compares the results. It also generates a “Validate” textual report from the NN model and represents a summary of data in table, radar chart, and confusion matrix, if applicable.
- [**Benchmark**]: allows the user to compare different model results by numerical parameters and graphical representation using radar charts

Load NN Model / Generate page

To start configuring the ISPU Model Converter, the Workspace directory and the Output directory must be selected using the corresponding [**Browse**] buttons.

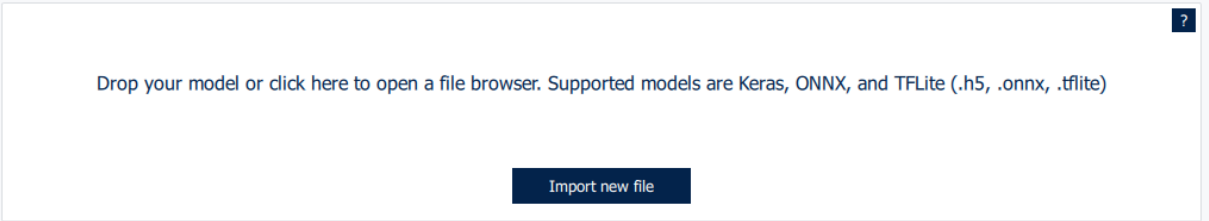
All GUI settings set during neural network processing can be restored to the default values by clicking on the [**Clear Settings**] button.

By clicking on the [**Terminal**] button, accessible from all pages, the user is able to inspect additional information related to the execution of commands requested in the available pages.

Figure 73. ISPU Model converter configuration bar

A drag and drop box allows importing NN files with supported formats such as Keras, ONNX, and TFLite (.h5, .onnx, .tflite).

Figure 74. ISPU Model converter drop area



By clicking on the [**Display NN graph**] button, the user is able to see visual graph of the chosen neural network model. To be able to access this function, the user needs to install the Netron application on the device and add it to the MEMS Studio tool path in the “External Tools” page.

Figure 75. Selected file options view

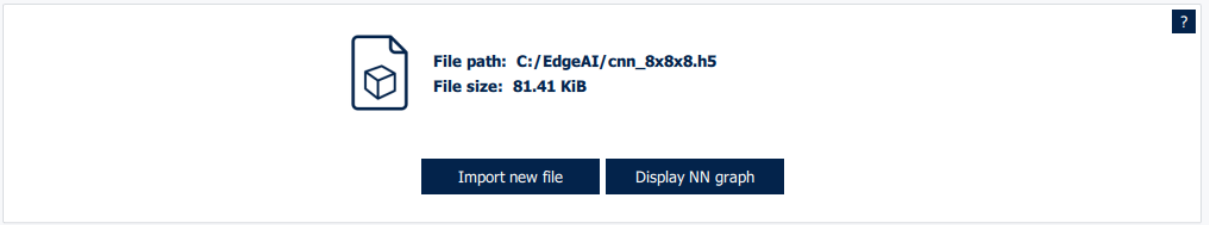
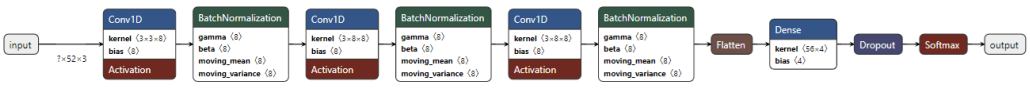
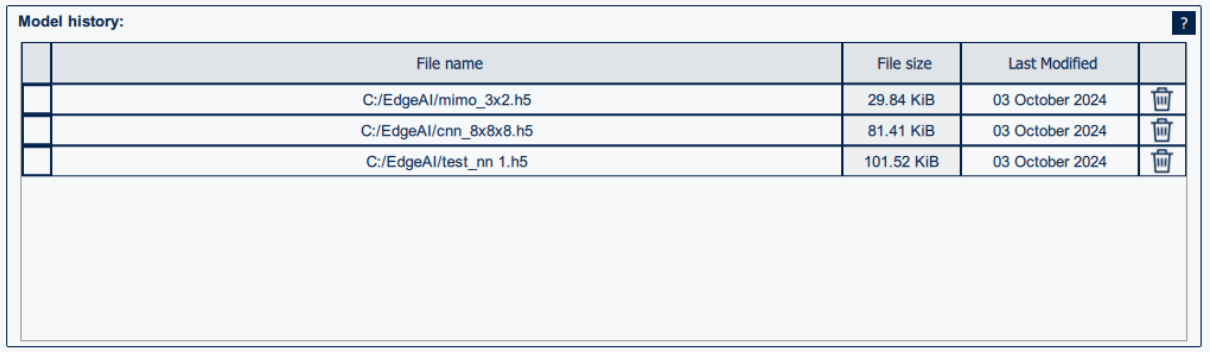


Figure 76. Neural network graph



The [**Model history**] section stores files previously uploaded by the user, displaying their directory path and the date of upload. Users can reselect files if they still exist in the same directory by clicking the [**Select**] button or delete them from the list by clicking on [**Delete**]. Deleting a file from the history only removes it from the graphical view, not from the container directory.

Figure 77. Loaded model history list



Once the workspace directory, output directory, and file to process are set, the [**Run Generate**] button becomes active. Users are able to start the generation of reports, .c and .h files by clicking on this button. Once the process is completed, the [**Inspect Report**] button becomes active, allowing users to view the textual report generated.

An additional button [**Open Output Directory**] is shown to navigate to the output directory.

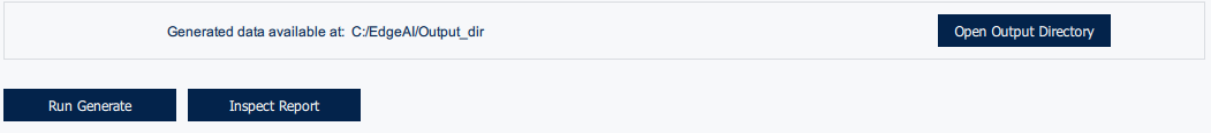
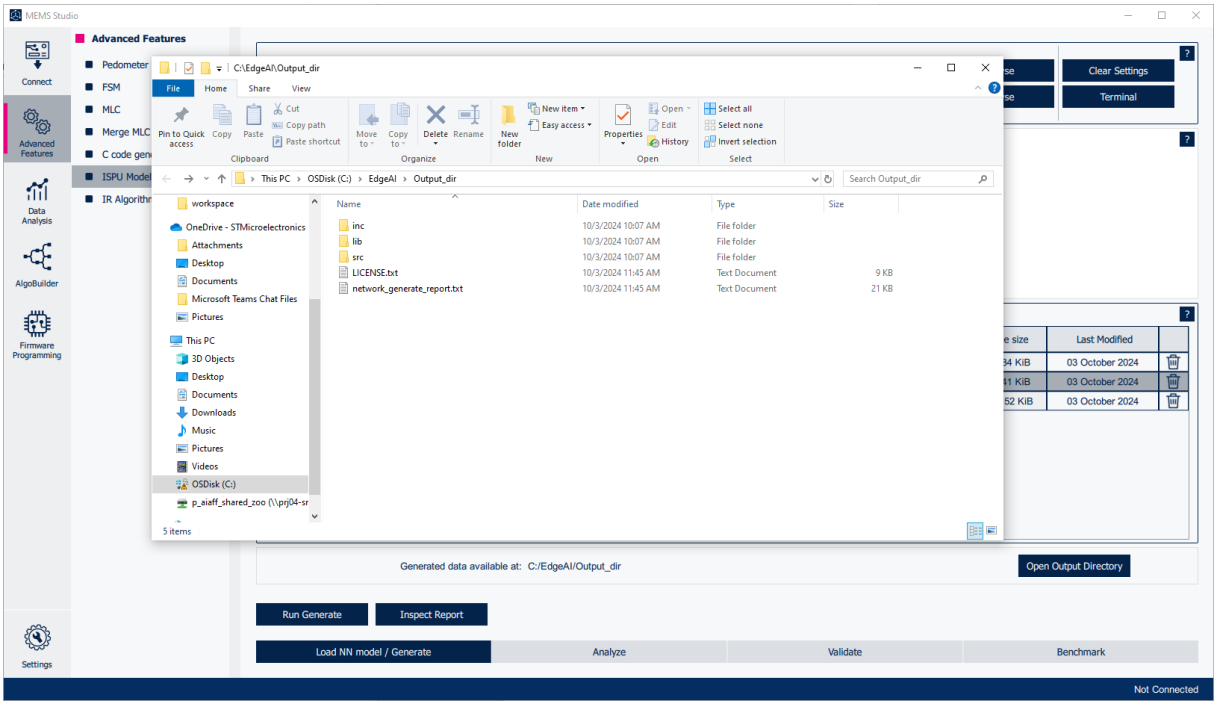
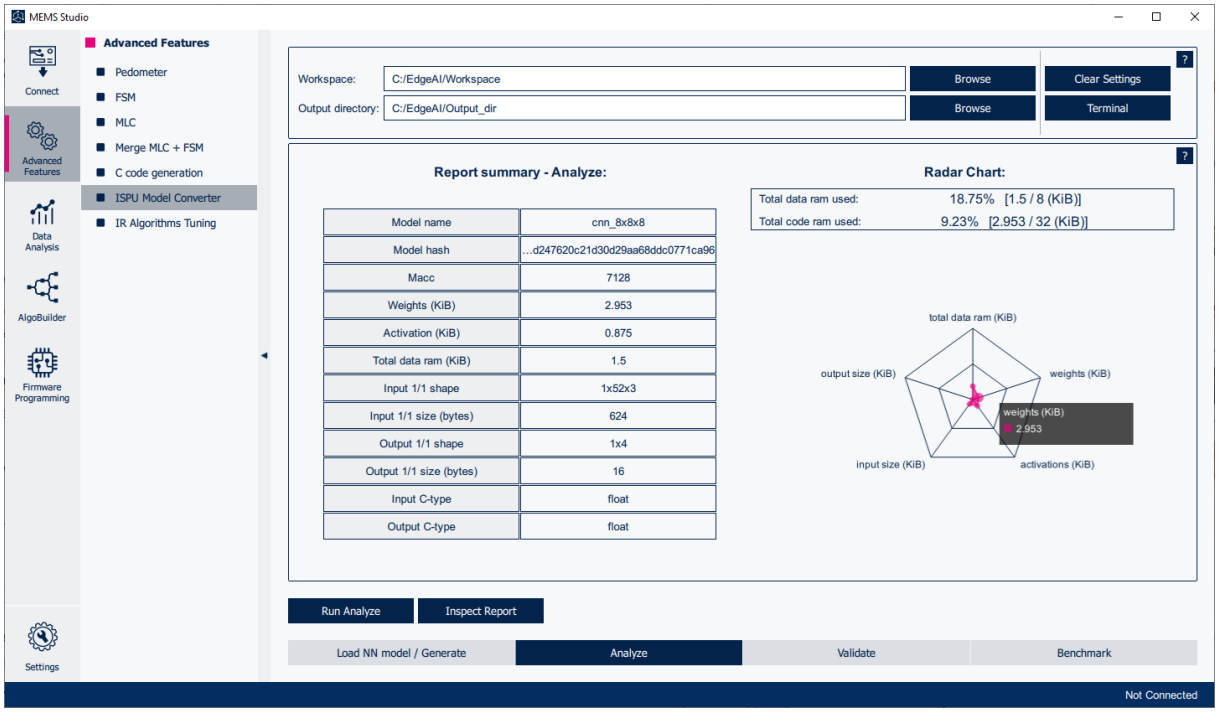


Figure 78. Output directory



Analyze page

Figure 79. Analyze page



Once the workspace directory, output directory, and file to process are set, the [**Run Analyze**] button becomes active and the user is able to start the analysis of the neural network model. Once the process is completed, the [**Inspect Report**] button becomes active, allowing users to view the textual report generated.

Users can view a summary of the report with the following data representation:

• **Table**: displays structured data extracted from the Analyze report, providing a detailed overview of key information

• **Radar Chart**: shows data metrics in a radar chart format, offering a graphical representation of different parameters.

Note: All the parameters in the radar chart are normalized in order to present data in a range [0, 1]. They are not in the same scale, so, to be able to show them in a radar chart, they are normalized according to their maximum value.

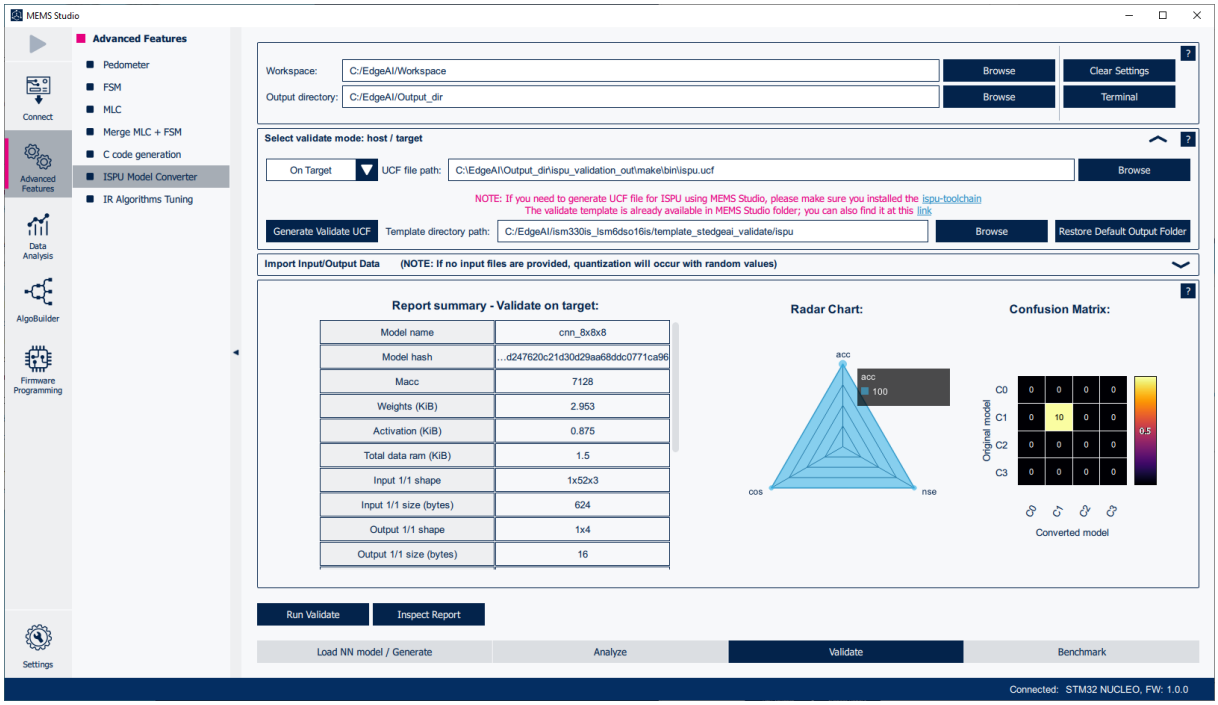
Considering (as an example) the LSM6DSO16IS sensor, the list of maximum values for radar chart parameters is as follows:

- Total data ram: 8 KiB (activation + input + output size)
- Activation: 8 KiB
- Input: 8 KiB
- Output: 8 KiB
- Total code ram: 32 KiB (weight + code)
- Weight: 32 KiB

The percentage of total ram used by the model is divided into two parts: data ram (8 KiB), which is the sum of activation, input and output size, and code ram (32 KiB), which includes code size and weight size.

Validate page

Figure 80. Validate page



On the Validate page, the user is able to run the original and converted models and compare the results by clicking on the [**Run Validate**] button. The converted model runs on the host PC and on the Target by connecting the sensor and choosing the mode to run [**On Host/ On Target**]. Once the process is completed, the [**Inspect Report**] button becomes active, allowing users to view the textual report generated. The processed data results are available in three different representations:

- **Table**: displays real values extracted from the Validate report, providing detailed information about various neural network metrics and parameters
- **Radar Chart**: displays normalized values in a radar chart format, offering a graphical representation of the key parameter.

Note: All the parameters in the radar chart are normalized in order to present data in a range [0, 1]. They are not on the same scale, so, to be able to show them in a radar chart, they are normalized according to their maximum value.

The maximum scale for each parameter is:

- Classification accuracy: 100 percent
- Cosine: 1
- Nash-Sutcliffe efficiency: 1

Converter models have more similar results to the original model if the parameters are close to the maximum.

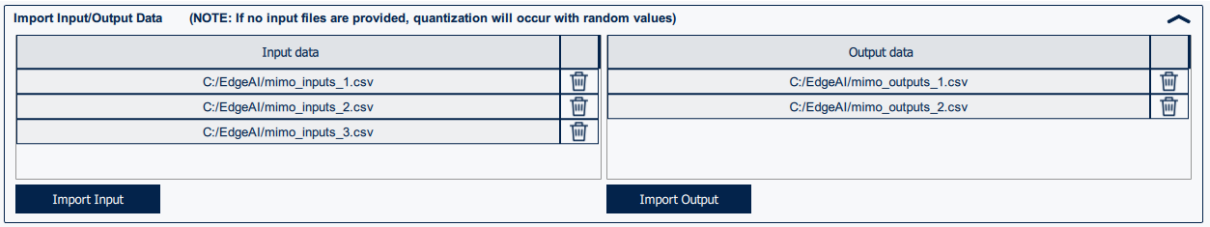
- **Confusion Matrix**: if available in the report, displays the performance of a classification model, showing true positive, true negative, false positive, and false negative values, assuming the original model as the reference
- **Select the output**: this combo box appears if the model contains multiple outputs, then the user is able see the result in a table based on the selected output

To be able to "Run Validate" on the Target, after selecting the mode [**On Target**], the user needs to import the UCF file. In order to generate the ISPU UCF file using MEMS Studio, the ISPU-toolchain path should be set in the “External Tools” page and a valid validation template should be selected. MEMS Studio embeds a template and link to download it. Once the [**Generate Validate UCF**] request is completed, a new .ucf file is created in the output directory in the subdirectory *ispu_validation_out /make/bin/ispu.ucf*

Figure 81. Validate options bar

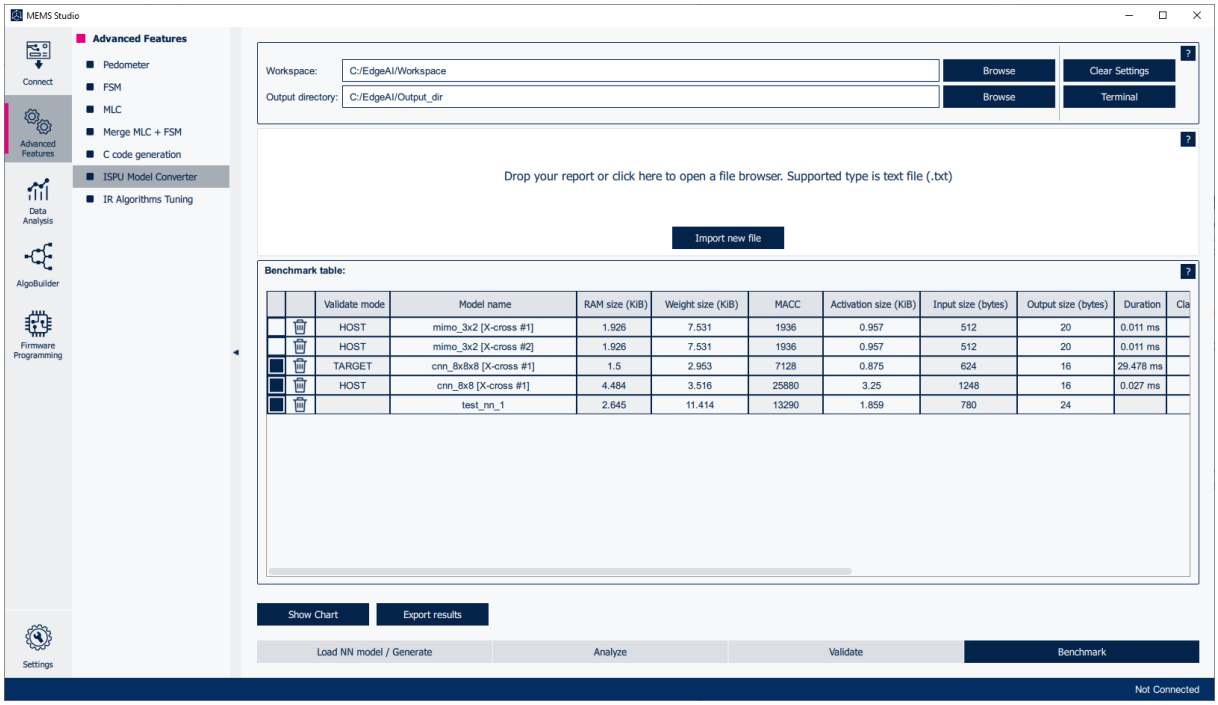
By default, the validation of the selected neural network is executed using random data. The user can select custom input and output to quantize using custom data.

Figure 82. Validate input/output selection bar



Benchmark page

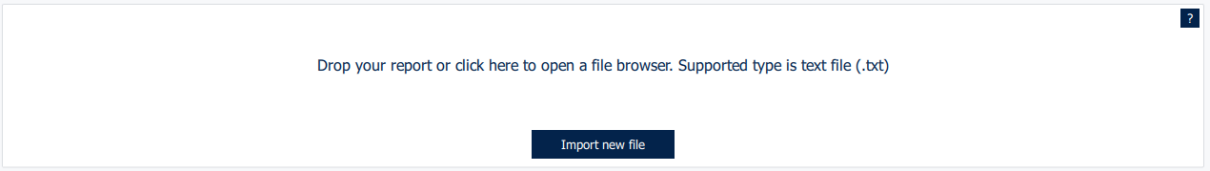
Figure 83. Benchmark page



On the Benchmark page, the user can compare different neural network models through numerical and visual results. This functionality allows for the comparison of the same model in different validation modes (On Host / On Target).

A drag and drop box allows importing neural network report files with the supported format as text (.txt).

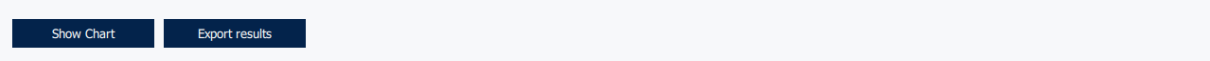
Figure 84. Benchmark drop area



By clicking on the [**Export results**] button, the user is able to extract the results of the entire benchmark table as a .csv file.

By clicking on the [**Show Chart**] button, the user is able to see multiple layers and compare the parameter visually.

Figure 85. Benchmark result options



The radar chart viewer displays normalized values in a radar chart format, offering a graphical representation of the key parameter in multiple layers of the selected model.

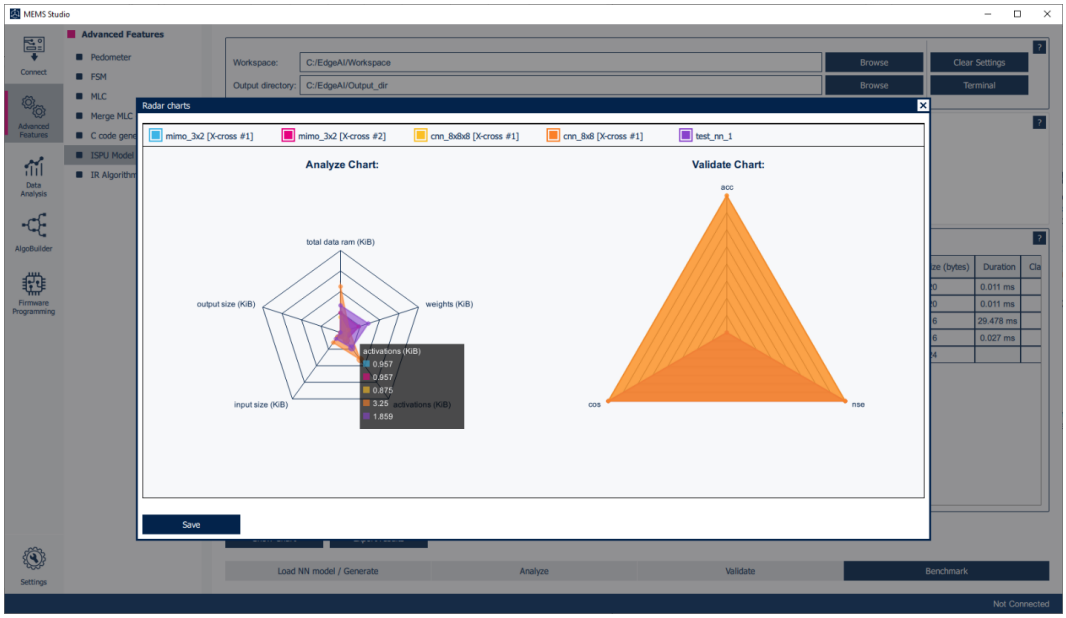
Note: All the parameters in the radar chart are normalized in order to present data in a range [0, 1]. They are not on the same scale, so, to be able to show them in a radar chart, they are normalized according to their maximum value.

The maximum scale for each parameter is:

- Classification accuracy: 100 percent
- Cosine: 1
- Nash-Sutcliffe efficiency: 1
- Total data ram: 8 KiB (activation + input + output size)
- Activation: 8 KiB
- Input: 8 KiB
- Output: 8 KiB
- Total code ram: 32 KiB (weight + code)
- Weight: 32 KiB

The converter models have more similar results to the original model if the parameters are close to the maximum.
By clicking on the [**Save**] button, the user is able to store the chart viewer results as a .png file.

Figure 86. Benchmark result charts



5.7 IR Algorithms Tuning

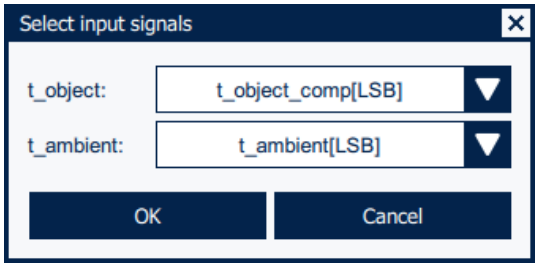
The [**IR Algorithms Tuning**] tool is designed to fine-tune infrared sensor settings to accurately detect presence, motion, and ambient shock.

The tool is divided in four tabs. The first tab is for presence detection, the second for motion detection, the third one for ambient shock detection, and the fourth for sequential processing of multiple files.

To begin using the tool, you need to load previously acquired data. Follow these steps:

1. Click on the [**Browse**] button.
2. Select the file containing the data.
3. Choose the columns corresponding to `t_object` and `t_ambient` data.
4. The data is displayed in charts for further analysis.

Figure 87. Columns selection



In the Presence Detection section, you can configure the following parameters:

- [**T_Presence Abs Mode**]: enables or disables absolute mode.
- [**Threshold [LBS]**]: sets the threshold value for presence detection.
- [**Hysteresis [LBS]**]: sets the hysteresis value for presence detection.
- [**LPF cut-off (Presence)**]: sets the cutoff frequency for the low-pass filter specific to presence detection.
- [**LPF cut-off (Presence & Motion)**]: sets the cutoff frequency for the low-pass filter that applies to both presence and motion detection.

Figure 88. Presence detection

Figure 89. **Sensor parameter settings**

In the Motion Detection section, you can configure the following parameters:

- [**Threshold [LBS]**]: sets the threshold value for motion detection.
- [**Hysteresis [LBS]**]: sets the hysteresis value for motion detection.
- [**LPF cut-off (Motion)**]: sets the cutoff frequency for the low-pass filter specific to motion detection.
- [**LPF cut-off (Presence & Motion)**]: sets the cutoff frequency for the low-pass filter that applies to both presence and motion detection.

Figure 90. **Motion detection**

In the Ambient Shock Detection section, you can configure the following parameters:

- [**Threshold [LBS]**]: sets the threshold value for ambient shock detection.
- [**Hysteresis [LBS]**]: sets the hysteresis value to prevent false detections.
- [**LPF cut-off (Motion)**]: sets the cutoff frequency for the low-pass filter specific to motion detection.
- [**LPF cut-off (Ambient Shock)**]: sets the cutoff frequency for the low-pass filter specific to ambient shock detection.

Figure 91. **Ambient temperature shock detection**

The toolbar provides several essential functions for managing your configurations and simulations:

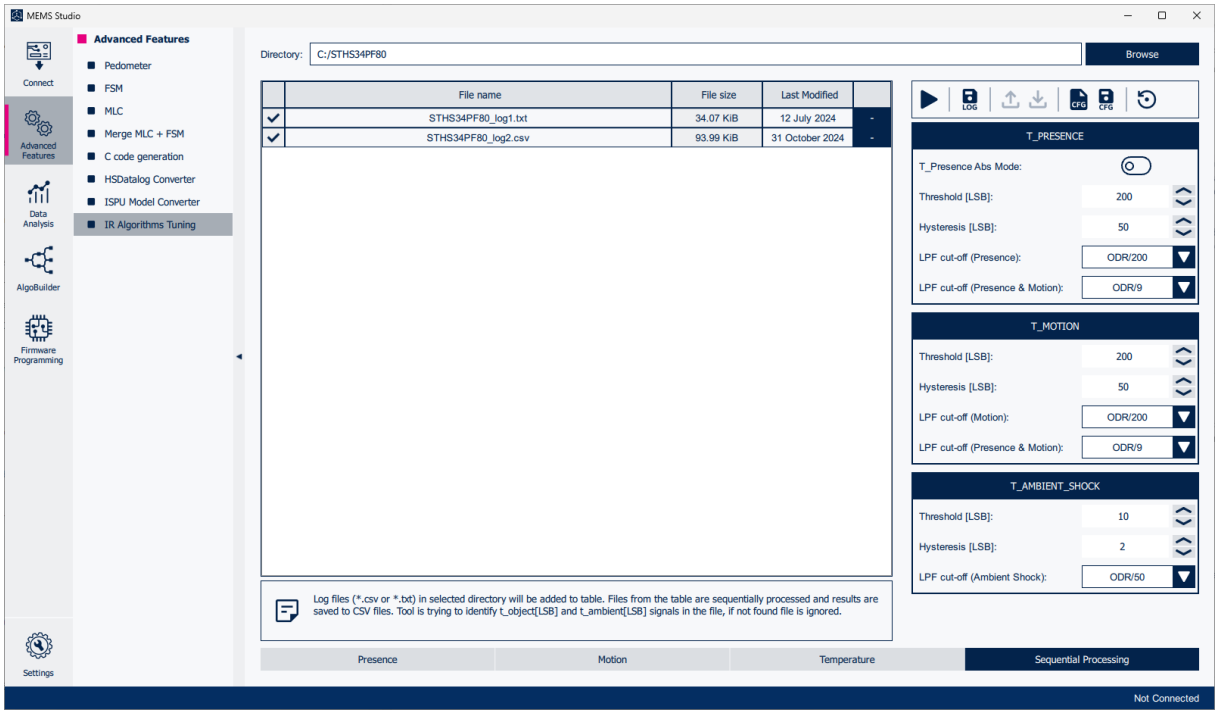
- [**Run processing**]: executes the simulation based on the current settings.
- [**Save log**]: saves the results of the simulation to a file.
- [**Read configuration from sensor**]: loads the current configuration settings from the sensor.
- [**Write configuration from sensor**]: writes the current configuration settings to the sensor.
- [**Load configuration from file**]: reads the configuration settings from a file.
- [**Save configuration from file**]: writes the current configuration settings to a file.
- [**Restore defaults**]: resets all settings to their default values.

Figure 92. IR Algorithms Tool toolbar

The Sequential Processing tab is designed to process multiple files sequentially. Follow these steps:

1. Click on the [**Browse**] button.
2. Select a folder containing the files to be processed.
3. The tool processes each file one by one.
4. The results are saved to the respective files.

Figure 93. Sequential Processing



Note: The sequential processing expects `t_object[LSB]` and `t_ambient[LSB]` columns in all files. If the columns are not found, the particular file is ignored.

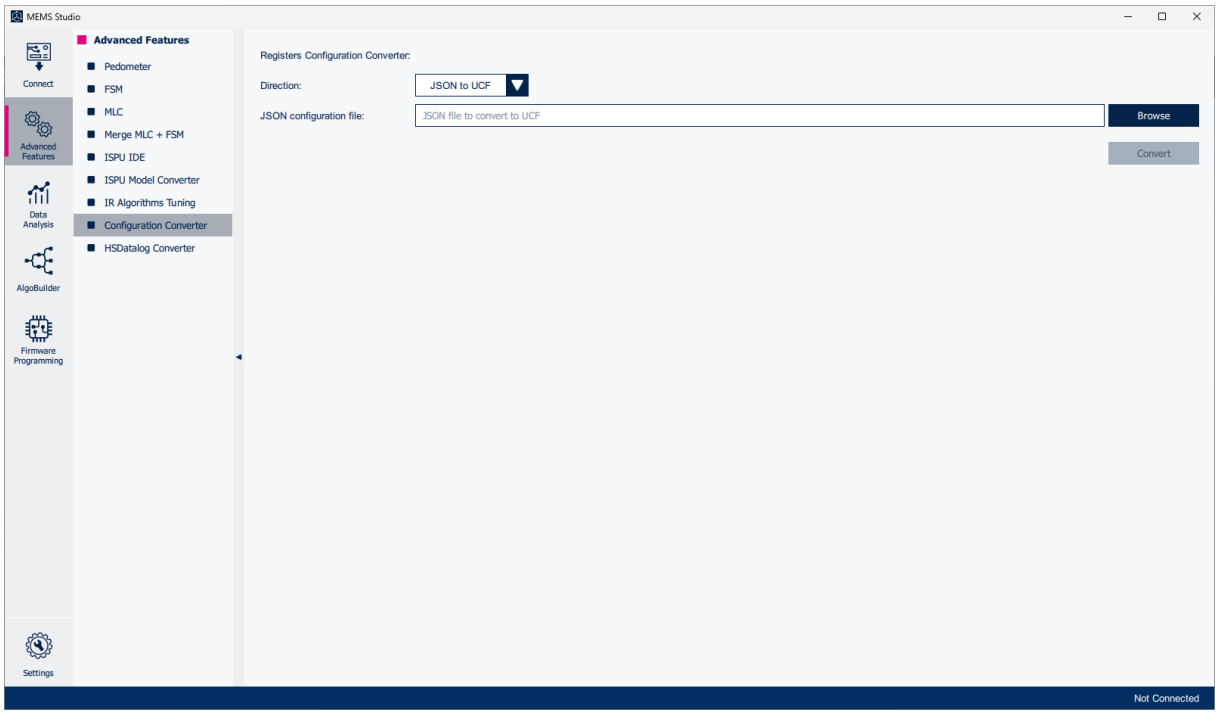
5.8 Configuration Converter

The [**Configuration Converter**] allows the conversion of sensor configuration from one file to another. The following options are supported:

- [**JSON to UCF**]: converts JSON configuration file to legacy UCF format. Please be aware that information and structures unsupported in UCF format will be omitted.
- [**UCF to JSON**]: converts legacy UCF format to JSON configuration file. The sensor name needs to be specified. There is an option to include also metadata from the UCF file.
- [**JSON to HEADER**]: converts JSON configuration file to .h header file intended for firmware projects written in C language. There is an option to include also metadata from the UCF file.
- [**HEADER to JSON**] : converts .h header file to JSON configuration files. There is an option to include also metadata from the header file.

Attention: *In MEMS Studio 2.0.0 the UCF (Unico Configuration Format) configuration file format has been replaced with JSON (JavaScript Object Notation) format throughout the application. This change brings several significant benefits and improvements to the user experience and the overall functionality of the application.*

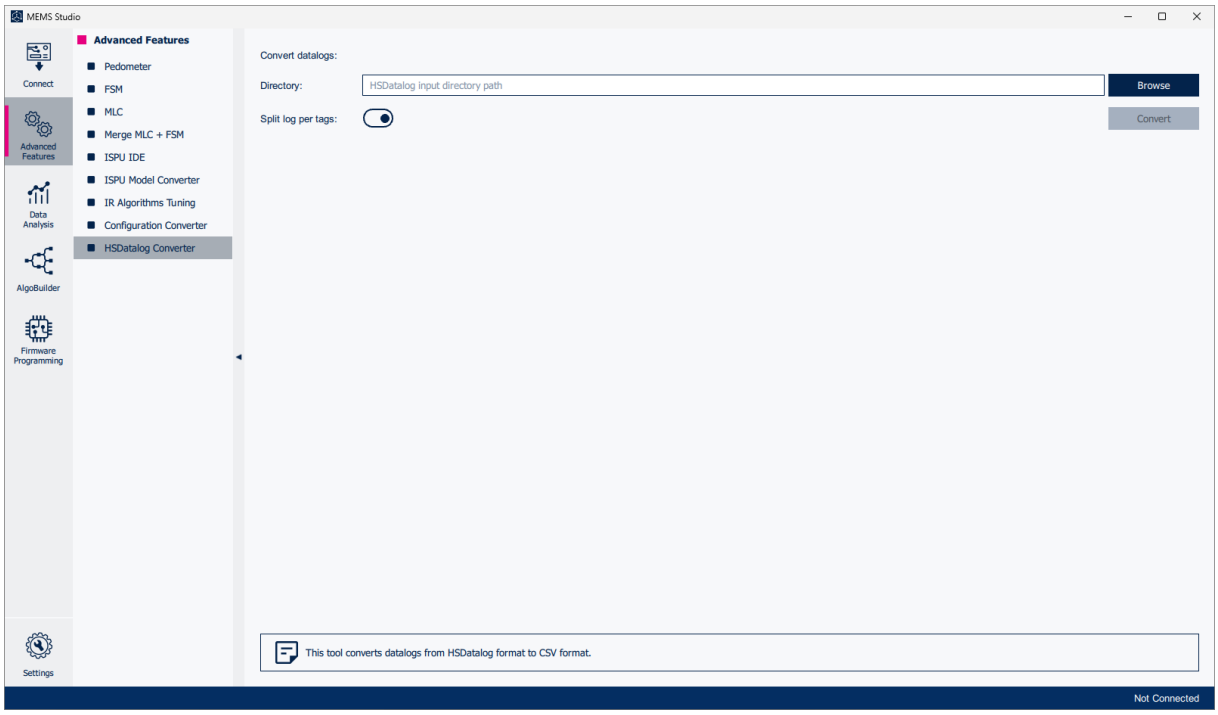
Figure 94. Configuration Converter



5.9 HSDatalog Converter

The [**HSDatalog Convertor**] allows the conversion of previously acquired data logs in binary format into CSV format. The binary data log files are generated by the Datalog2 (High Speed Datalog) firmware. There is a possibility to add a tag during the data logging process. The converted logs can be separated based on this tag when the [**Split log per tags**] option is enabled.

Figure 95. HSDatalog Convertor



6 Data analysis

The data analysis feature allows visualizing and analyzing data in the time and frequency domains. In the frequency domain, the frequency spectrum from part of or the whole data log can be calculated using fast Fourier transform (FFT). Another option of frequency domain analysis is to use a spectrogram, which visualizes frequency spectrum progress over time.

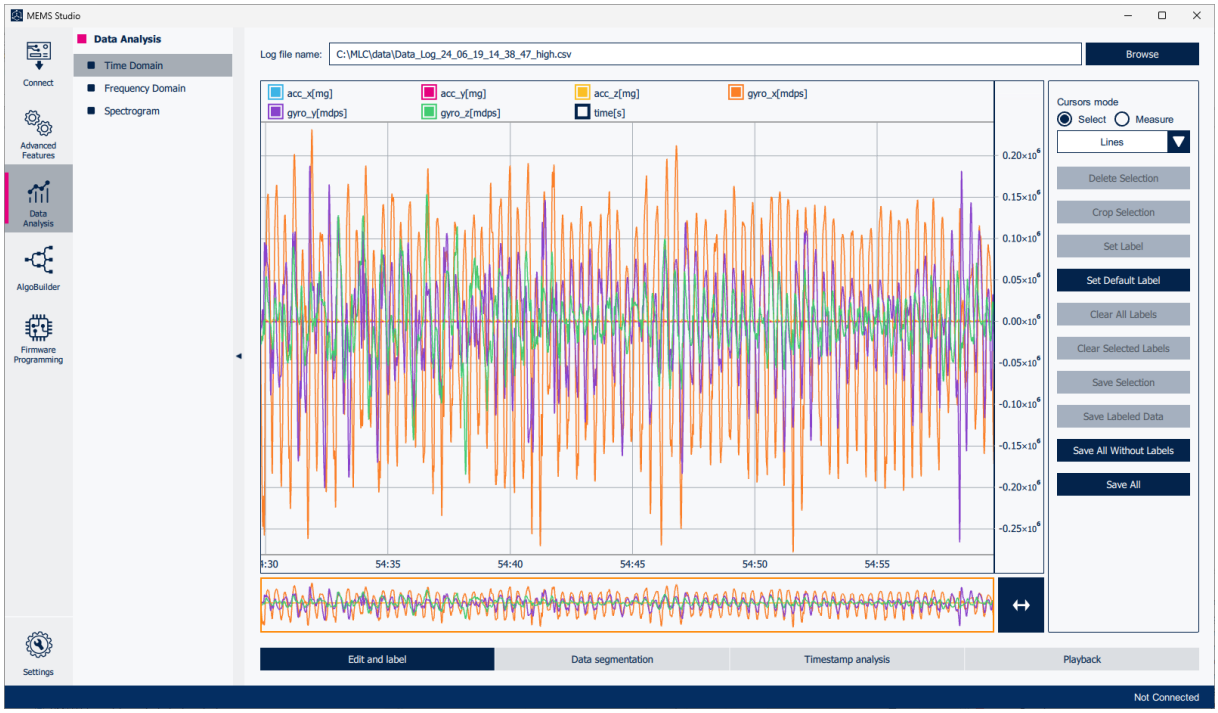
6.1 Time domain

Time domain analysis is intended for data visualization, measuring, editing, data segmentation and labelling, timestamp analysis and audio playback. The functionalities are divided into several pages.

Edit and label

Previously acquired data logs can be opened by clicking on the [**Browse**] button.

Figure 96. Time domain - Edit and label page



After successfully loading the data log, data are visualized in the line chart. The user can navigate through the chart in the same way as other charts in the application. Moving the cursor to the Y-axis offers a context menu for the chart.

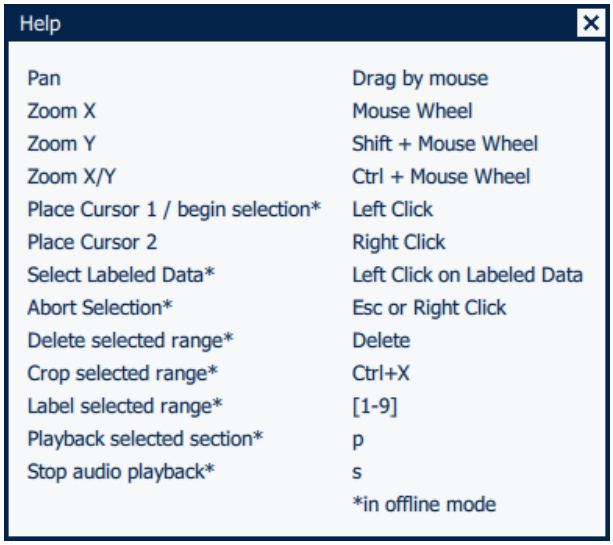
There are two options for data visualization. Data can be visualized as individual points or as lines connecting all the measured values.

Figure 97. **Control options for the line chart**

Available options are:

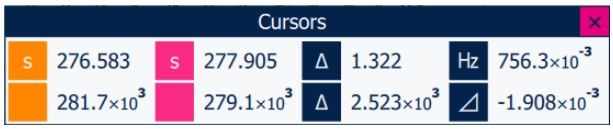
- [**Fit**]: adjusts the Y-axis range to display the whole signal
- [**Auto**]: enables/disables the option to automatically adjust the Y-axis range according to the visible data in the chart
- [**Full**]: adjusts the Y-axis range according to the min and max values, which can be modified by the user
- [**Save**]: saves the chart as an image in .png format
- [**Copy**]: saves the chart as an image in the system clipboard
- [**Help**]: displays help for working with the chart and using shortcuts

Figure 98. **Control shortcuts**



The chart offers the possibility to measure various values. This can be enabled by switching from Cursor mode to [**Measure mode**]. Two cursors are displayed in the chart, their X and Y positions can be adjusted using the mouse. The corresponding X and Y values and their differences are displayed in the table bellow. Cursors can be hidden by closing the cursor table in the chart area.

Figure 99. **Cursor value table**



The data log can be edited and labels assigned if the Cursor mode is switched to [**Select mode**]. The start of the area is marked by clicking at the desired position in the graph, the start cursor is placed at this position, then the signal is selected by mousing over it, the next click defines the end of the selected area, and the end cursor is placed at this position. The following operations are available with the selected area:

- [**Delete Selection**]: deletes the selected part of the signal
- [**Crop Selection**]: deletes all parts except the selected area
- [**Set Label**]: assigns a label to the selected part, the existing label name can be selected, or a new label created. If some labels are already defined, the existing labels can be assigned to the selected part of the signal using keys [**1–9**], according to the previously defined label order.
- [**Set Default Label**]: sets a label to all unlabeled parts
- [**Clear All Labels**]: removes all labels
- [**Clear Selected Labels**]: removes the label from the selected part
- [**Save Selection**]: saves the selected part to a new file

There are also other operations for data labeling:

- [**Save Labeled Data**]: saves each data for each label to a separate file
- [**Save All Without Labels**]: saves all changes without assigned labels to one file
- [**Save All**]: saves all changes including assigned labels to one file

Data segmentation

The data segmentation tool automatically identifies breakpoints within a selected data log by analyzing the statistical parameters of the time-series signal. It segments the data into meaningful sections based on these detected points. The algorithm is iterative, and the user provides feedback by adjusting breakpoints. The algorithm then re-estimates the breakpoints, incorporating the user feedback.

Figure 100. Time domain - Data segmentation page

The typical work flow is as follows:

1. **Load Input Data** - The user loads the input data from a file. The tool automatically groups signals into ODR (output data rate) groups, as the segmentation algorithm requires consistent ODR for the input signals.
2. **Select Signals** - The tool displays a list of signals for each ODR group. By default, all signals in the ODR group are selected for data segmentation, but the user can deselect any signal.

3. **Run Segmentation** - The user initiates segmentation by clicking the [**Run**] button. The selected signals are analyzed by the data segmentation algorithm, and the estimated breakpoints are displayed on the plot. Data segments are visualized with alternating colors.
4. **Adjust Breakpoints** - The breakpoints are interactive and can be moved using the mouse or removed by clicking the [**X**] button that appears when hovering over a breakpoint line.
5. **Update Algorithm** - Once the breakpoint adjustments are complete, the user provides feedback to the algorithm by clicking the [**Update**] button. The algorithm recalculates the breakpoints based on the user feedback. Breakpoints modified by the user are treated as fixed by the algorithm and are used as input for the new estimation. These fixed breakpoints are displayed as solid lines on the plot, while calculated breakpoints are shown as dashed lines.

The number of feedback iterations is unlimited. After each update, the user can modify the breakpoints and trigger a new calculation by clicking the [**Update**] button.

Additional Controls

- [**Reset**] - Resets the algorithm, allowing the user to select new input signals.
- [**Add Labels**] - Automatically adds labels to the identified segments. The labels are generated as S#, where # is a number starting from zero (e.g., S0, S1, S2, etc.). Automatically assigned labels can be modified on the [**Edit and label**] page.
- [**Load Weights**] - Loads the data segmentation algorithm state from a file. Use this button to apply a previously saved algorithm state to the current input signals.
- [**Save Weights**] - Saves the data segmentation algorithm state to a file. Use this button to store the algorithm state after training it on the current input signals.

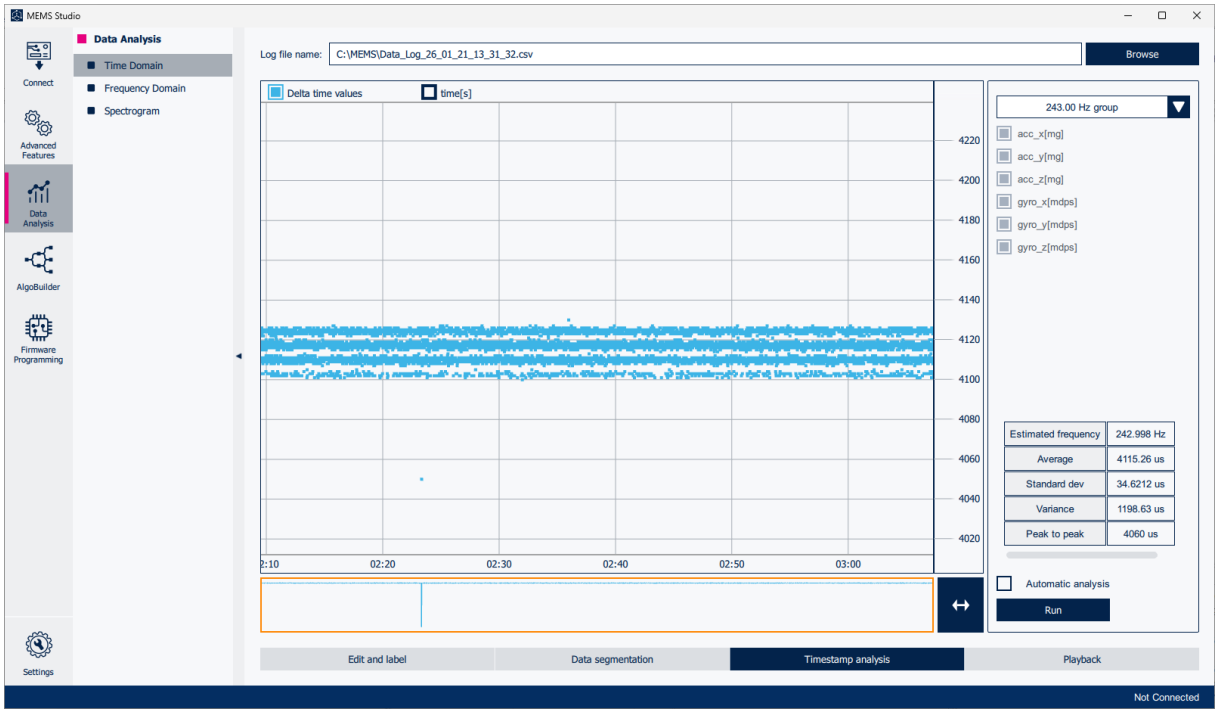
Timestamp analysis

The Timestamp analysis tool processes entire data log files to analyze the differences between consecutive timestamps. It visualizes these time intervals in chart, enabling quick identification of timing patterns or irregularities. Additionally, the tool calculates key statistical parameters such as average, standard deviation variance and peek to peek value.

The tool automatically groups signals into ODR (output data rate) groups.

The analysis can be executed by clicking the [**Run**] button. If the [**Automatic analysis**] option is checked, the analysis is automatically triggered with every ODR group change.

Figure 101. Time domain - Timestamp analysis page



Playback

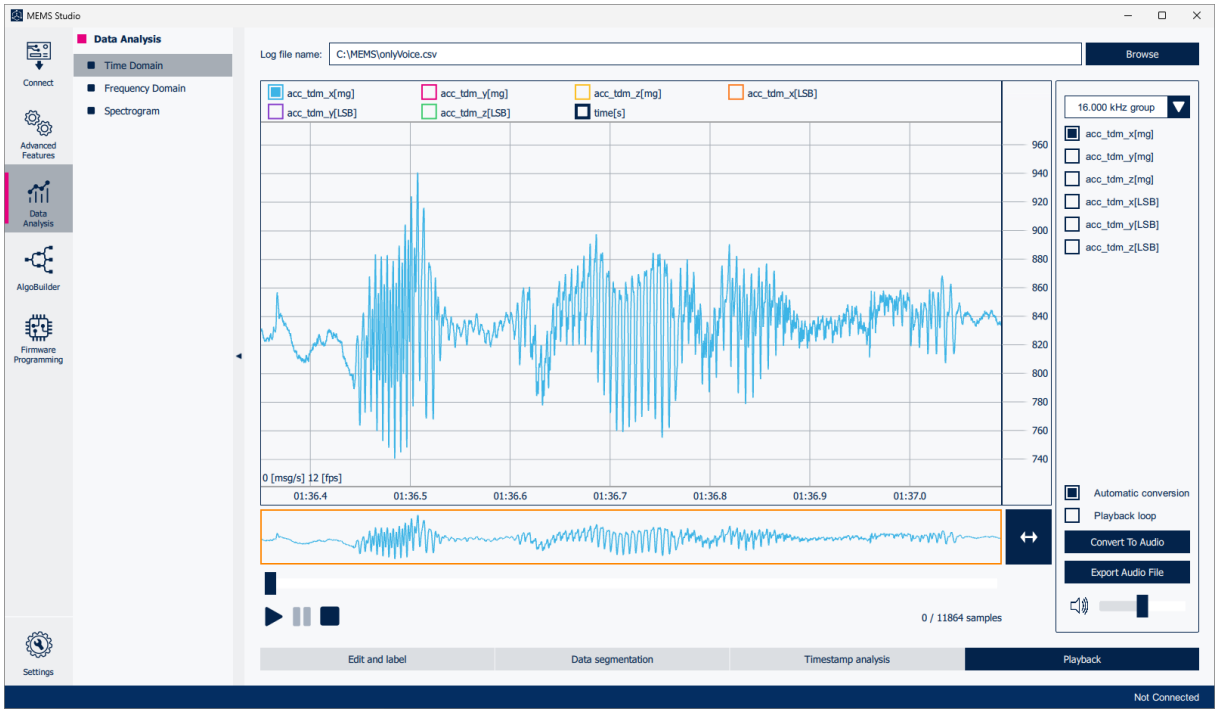
The playback functionality processes the entire data log in order to convert sensor sample data into audio to be streamed on a user laptop. The tool includes automatic sample frequency detection if the timestamp column is provided in the log file to be processed. If multiple signal groups with different sampling frequencies are detected, a dedicated combo box will be shown to select the sampling frequency group and signal to be processed. The audio conversion tool processes only one channel, therefore the user will not be able to select more than one channel.

If [**Automatic conversion**] checkbox is not checked user will need to click on [**Convert To Audio**] button in order to execute conversion action.

If [**Playback loop**] checkbox is checked, playback will start again on completion .

The [**Export Audio File**] button will allow user to store the .wav file generated into a given directory path.

Figure 102. Time domain - Playback page



6.2 Frequency domain

Frequency domain analysis is used to analyze a frequency spectrum using fast Fourier transformation (FFT). If a data log was already selected on the [**Time Domain**] page, it is automatically used on the [**Frequency Domain**] page. Other data logs can be selected using the [**Browse**] button.

The application offers two types of [**Analysis**]:

- [**FFT**]: Fast Fourier Transform algorithm to compute the frequency spectrum of a signal
- [**PSD**]: Power Spectral Density, which represents how signal power is distributed over frequencies

Figure 103. Frequency domain page

It is very important to know the sampling rate of the data for frequency domain analysis. If there is a timestamp in the data log, the sampling rate is calculated from this value. In the case of several sampling rates used in one data log, the signals are grouped according to each sampling rate and are analyzed separately. The active group can be selected by the user. If the timestamp is not available, the sampling rate must be entered by the user.

The frequency spectrum can be calculated from the entire signal or from a selected area. If [**Automatic evaluation**] is enabled, these two modes are automatically switched with respect to the part of the signal that is selected. Otherwise, the following buttons can be used:

- [**Compute FFT / Compute PSD**]: calculates the frequency spectrum from the entire signal
- [**Selected Data FFT / Select Data PSD**]: calculates the frequency spectrum from the selected area

If the data log contains labels, a labeled part of the signal can be quickly selected by clicking on the label.

A graph with the frequency spectrum can be configured using the following options:

- [**Magnitude**] [Linear, dB]: specifies the scale on the Y-axis
- [**Frequency**] [Linear, Logarithmic]: specifies the scale on the X-axis
- [**Style**] [Bar, Lines]: specifies visualization of the values

A frequency domain analysis can be configured using the following options:

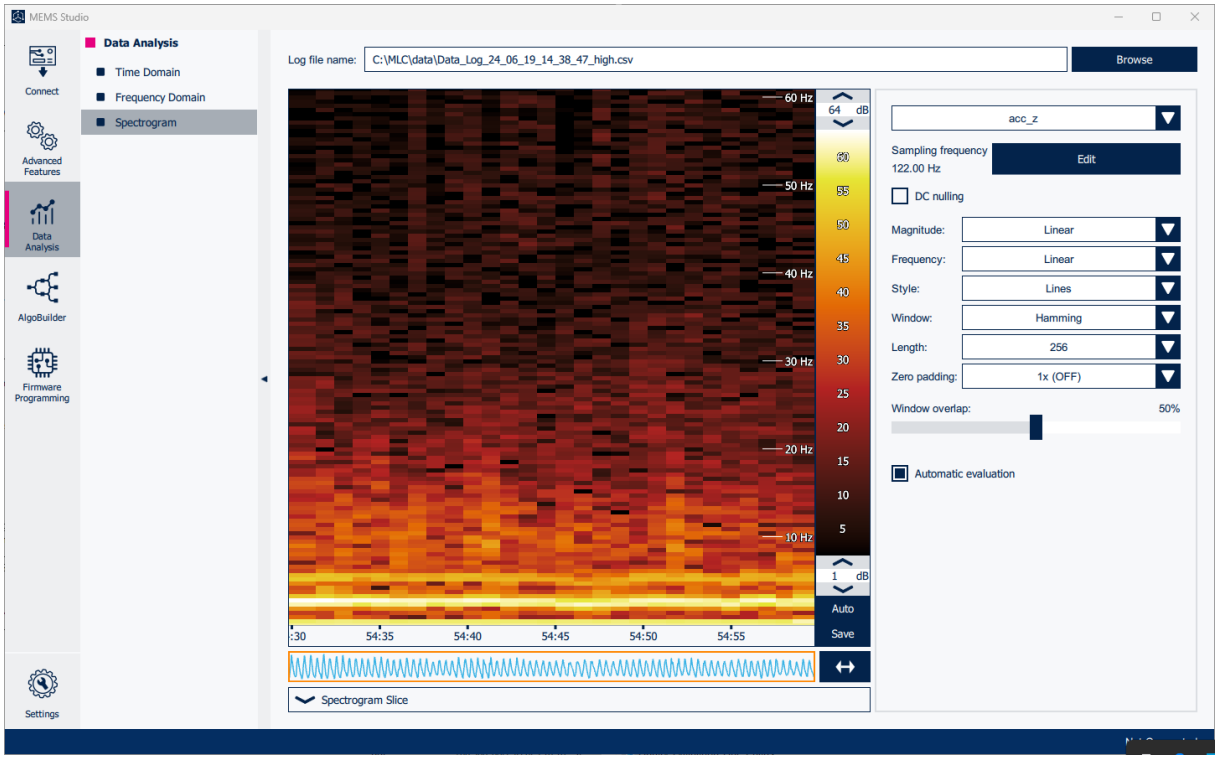
- [**DC nulling**]: removes the DC component from the signal
- [**Window**]: window function multiplied by the signal before frequency spectrum calculation in order to reduce the amplitude of the discontinuities at the boundaries of the signal part
- [**Length**]: number of samples used for the frequency spectrum calculation
- [**Zero padding**]: number of zeros added to the signal in order to increase resolution
- [**Window overlap**]: percentage of the previous window reused for the evaluation of the next window

The calculated frequency spectrum can be stored in a file using the [**Save All**] button.

6.3 Spectrogram

A spectrogram analysis is used to analyze the evaluation of the frequency spectrum over time. If a data log was already selected on the [**Time Domain**] or [**Frequency Domain**] page, it is automatically used on the [**Spectrogram**] page. Other data logs can be selected using the [**Browse**] button.

Figure 104. Spectrogram page



As the spectrogram utilizes the same fast Fourier transform as the frequency domain analysis, the same parameters can be configured also on the [**Spectrogram**] page.

7 AlgoBuilder

AlgoBuilder is a tool for the graphical design of algorithms.

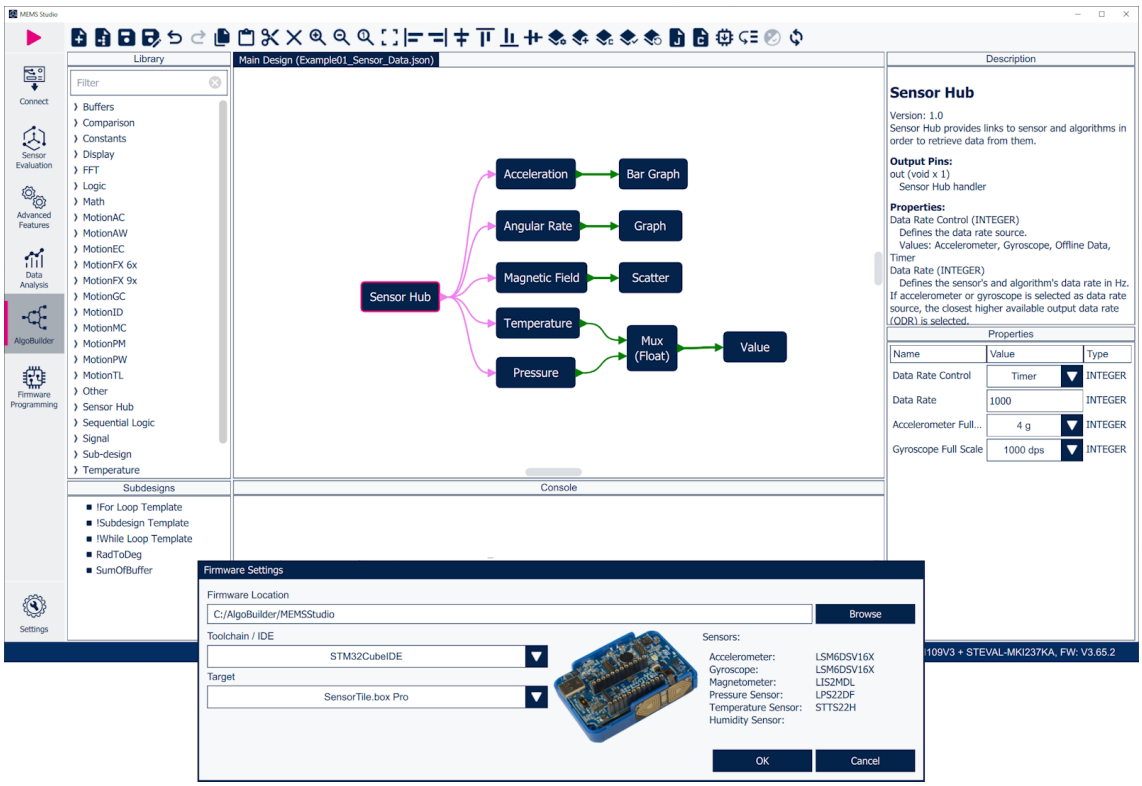
This tool quickly creates prototypes of applications for STM32 microcontrollers and MEMS sensors, including already existing algorithms (in the example of sensor fusion), user-defined data processing blocks, and additional functionalities.

The tool facilitates the process of implementing a proof of concept using a graphical interface without writing the code.

The key features of the application include:

- Simple graphical design of algorithms (drag and drop, connect, set properties, build, upload)
- Optional multilevel design
- Wide range of function blocks available in libraries, including motion sensor algorithms (for example, sensor fusion, gyroscope, magnetometer calibration, pedometer, and so forth)
- Integrated function blocks for FFT analysis
- Automatic validation of design rules
- C code generation from the graphical design
- Use of external compilers (STM32CubeIDE, IAR EWARM, Arm® Keil® µVision®)
- Possibility to integrate FSM, MLC, and ISPU in the design
- Open .json format for function blocks and design storage

Figure 105. AlgoBuilder



7.1 Principle of operation

The workflow starts from the graphical design of the desired functionality by using a simple "drag and drop" approach.

The flow diagram is composed of the predefined function blocks provided in the libraries. Custom function blocks can be created as well. Some function blocks have properties that can be adjusted.

Then, function blocks can be interconnected. AlgoBuilder automatically checks the compatibility between input and output and allows connecting only terminals with the same type and dimension.

When the design is finished, AlgoBuilder generates the C code from the defined graphical design. The final firmware project is created from the C code generator, combined with preprepared firmware templates and binary libraries.

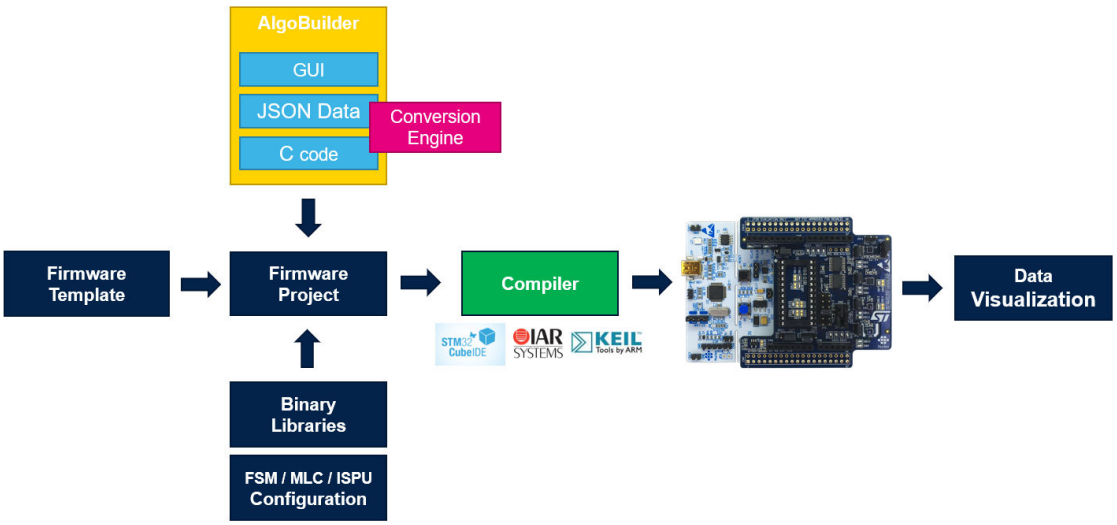
The project can be compiled using an external compiler tool and the most common compilers are supported (STM32CubeIDE, Arm® Keil® µVision®, IAR Embedded Workbench).

A development board is then programmed by the generated binary file.

When the firmware is executed, it starts reading data from the selected sensor, processes the data via the algorithm, and sends the results to the MEMS Studio application. During the graphical design, you can select how to see the results. Many options like graphs, logical analyzers, bar charts, 3D plots, scatter plots, and others are supported. During the startup, the firmware configures the UI to display the data in the desired format.

The graphical designs as well as the libraries are stored as .json files.

Figure 106. AlgoBuilder principle of operation



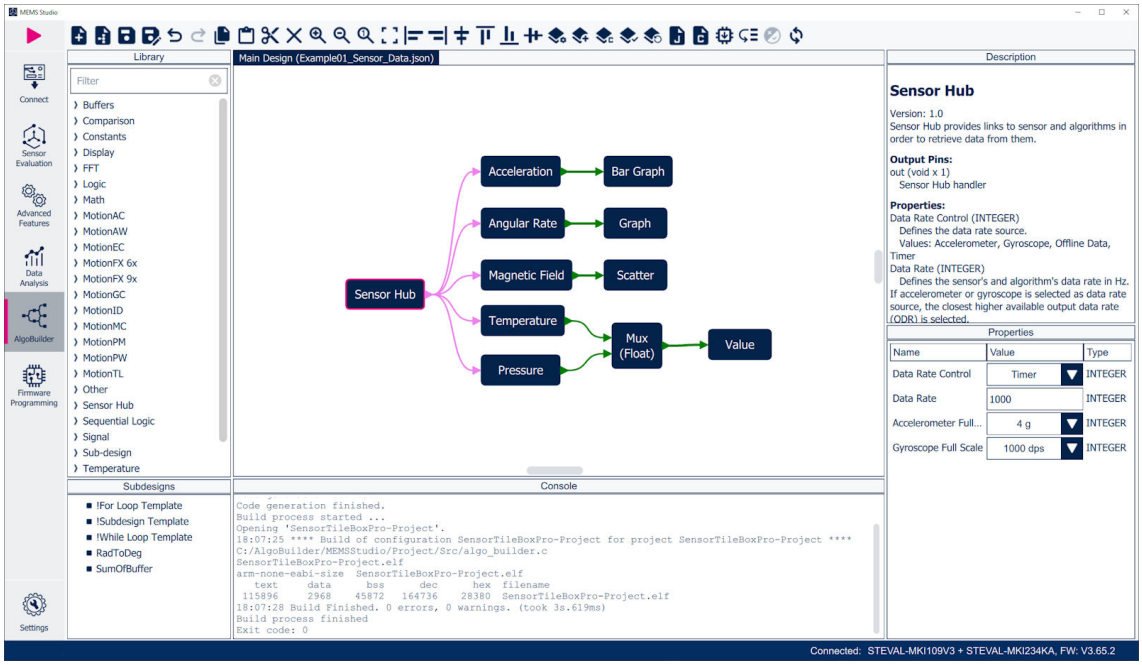
7.2 Overview

The AlgoBuilder tool contains the following:

- Central workspace where the algorithm is designed using function blocks
- [**Library**] dock with a list of available libraries and their function blocks, which can be dragged and dropped to the workspace window
- [**Subdesigns**] dock with a list of available subdesigns and subdesign templates
- [**Description**] dock that displays information about the selected component (function block, connection, and so forth)
- [**Properties**] dock that displays all available properties of the selected function block
- [**Console**] dock that displays messages from AlgoBuilder or an external compiler

The AlgoBuilder tool is controlled from a toolbar that offers all the necessary functions.

Figure 107. AlgoBuilder main page



7.3 Workspace

The algorithm design under development is created in the workspace area.

7.4 Library dock

The [**Library**] dock gives you access to all the available libraries and function blocks located in a particular library.

AlgoBuilder scans [Install path]\release\Library\ and the user's home directory \STMicroelectronics\MEMS Studio\algobuilder\library during startup and loads all valid libraries located there.

7.5 Subdesigns dock

The [**Subdesigns**] dock gives you access to all the available subdesigns. AlgoBuilder scans [Install path] \release\Subdesigns\ and the user's home directory\STMicroelectronics\MEMS Studio\algobuilder\subdesigns during startup and loads all valid subdesigns located there. The subdesign templates are located in the [Install path]\release\Subdesigns\ directory.

7.6 Description dock

The [**Description**] dock provides information about the component selected in the workspace or in the [**Library**] dock.

If you select a function block, the following information is shown:

- Name
- Version
- Description of the function block functionality
- Type, size, and functionality of all inputs and outputs
- Description of all function block properties

7.7 Properties dock

If a function block has a property or properties, they are displayed in the [**Properties**] dock.

Each property has [**Name**], [**Value**], and [**Type**] fields.

The values can be modified.

AlgoBuilder automatically checks if the value is valid and does not allow setting an invalid value (for example, a value out of an available range).

For the STRING type, the % character is forbidden and is automatically deleted.

7.8 Toolbar

The toolbar provides quick access to all needed functions.

Table 8. AlgoBuilder toolbar functions

Toolbar icon	Function
	Create new design
	Open existing design
	Save design (subdesign)
	Save design (subdesign) as a different file
	Undo
	Redo
	Copy
	Paste
	Cut
	Delete
	Zoom in
	Zoom out
	Zoom to default
	Fit whole design into the window
	Align blocks to the left
	Align blocks to the right
	Center blocks horizontally
	Align blocks to the top

Toolbar icon	Function
	Align blocks to the bottom
	Center blocks vertically
	Project configuration
	Initialize project directory
	Generate code
	Build project
	Rebuild project
	View design source file
	View generated source C file
	Program device by automatic selected method
	Display order configuration
	Add or remove conditional input
	Custom Block Creator

7.9 Libraries

The following libraries are available after the installation of MEMS Studio.

Table 9. AlgoBuilder libraries

Library	Content
Buffers	Function blocks for operations with data buffers
Comparison	Function blocks for two value comparisons (for example, >, <, =, and so forth)
Display	Function blocks for data visualization
FFT	Function blocks related to FFT analysis
Logic	Function blocks for logic operations (for example, And, Or, …, and so forth)
Math	Function blocks for various mathematical operations (for example, +, -, /, and so forth)
Other	Auxiliary function blocks (for example, mux, demux, type conversion, and so forth)
Sensor Hub	Sensor hub function block, which provides access to the sensors and prebuild algorithm, and function blocks that provide data from connected sensors
Sequential Logic	Function block for sequential logic (flip flops)
Signal	Function blocks for signal processing (for example, filters, and so forth)
Sub-design	Input and output terminals for subdesigns
Temperature	Function blocks for temperature unit conversions
User Input	Function blocks, which allow the user to send arbitrary data to the running firmware in real time
Vector	Function blocks for vector operation (for example, calculate magnitude, and so forth)

Already prepared algorithms in binary form can also be integrated in the user design.

Table 10. Motion libraries supported in AlgoBuilder

Library	Functionality
MotionAC	Accelerometer calibration algorithm
MotionAW	Activity recognition algorithm for wrist-worn devices
MotionEC	ecompass algorithm
MotionFX 6x	Sensor fusion algorithm using accelerometer and gyroscope
MotionFX 9x	Sensor fusion algorithm using accelerometer, gyroscope, and magnetometer
MotionGC	Real-time gyroscope calibration algorithm
MotionID	Motion intensity detection algorithm
MotionMC	Real-time magnetometer calibration algorithm
MotionPM	Pedometer algorithm for mobile devices
MotionPW	Pedometer algorithm for wrist-worn devices
MotionTL	Tilt sensing algorithm

Binary libraries are included in the firmware only if at least one function block is used in the design. This reduces occupied flash memory and RAM memory.

7.10 Data types

AlgoBuilder works with four data types:

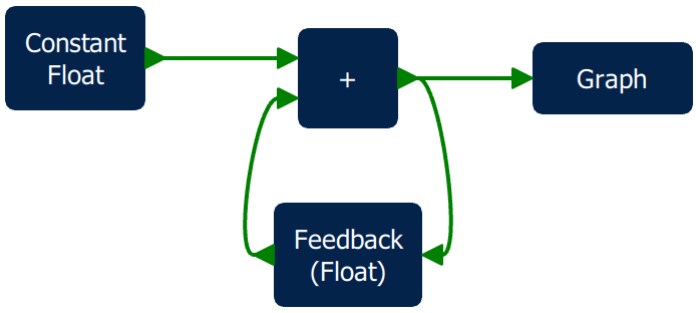
- **FLOAT** represents real numbers and is used for floating-point arithmetic (for example, in the acceleration function block output). In C code, the representation float variable is used. The size is 4 bytes.
- **INT** represents integer numbers (for example, in the counter function block). In C code, the int32_t variable is used. The size is 4 bytes.
- **VARIANT** is used for inputs of a set of function blocks; the variant changes its type on the basis of the type of output connected to this input (for example, the variant type is used for inputs of comparison function blocks).
- **VOID** is used exclusively for the connection between the sensor hub and its data outputs. This type cannot be visualized.

Each input or output is characterized by its type and size, the color of the connection line indicates the type.

Important: Only input and output with the same type and size can be connected together. The only exception is the VARIANT input, which gets the type of the connected output.

It is not possible to connect the input and output of the same function block. If this is desired, the [**Feedback**] function block for the particular data type needs to be used. The [**Feedback**] function block has an Init value, which defines the output value of the block for the first run.

Figure 108. Using the Feedback function block



7.11 Data visualization

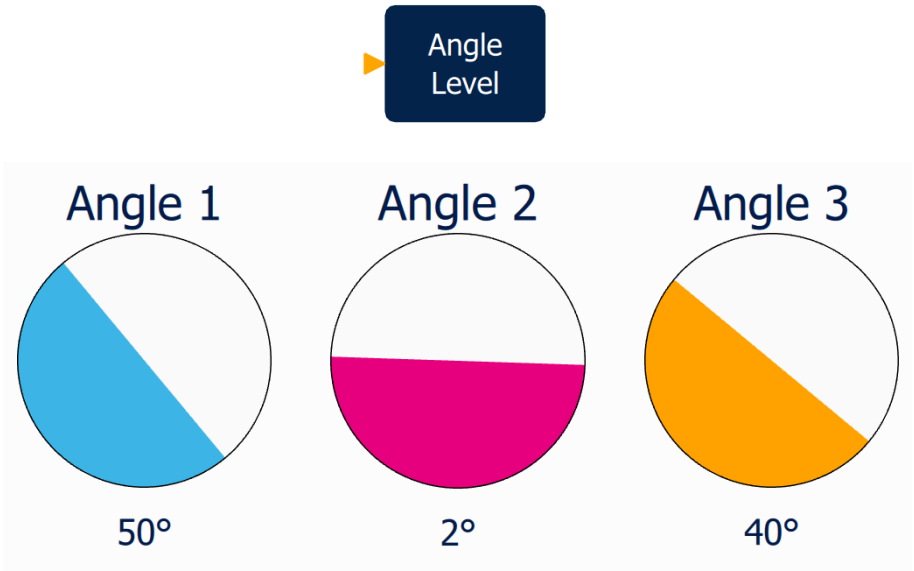
The application offers nine types of data visualization in the [**Display**] library:

- [**Angle Level**]
- [**Bar Graph**]
- [**FFT Plot**]
- [**Fusion**]
- [**Graph**]
- [**Logic Analyzer**]
- [**Plot3D**]
- [**Scatter Plot**]
- [**Value**]

Use the appropriate function block according to your requirements for data visualization.

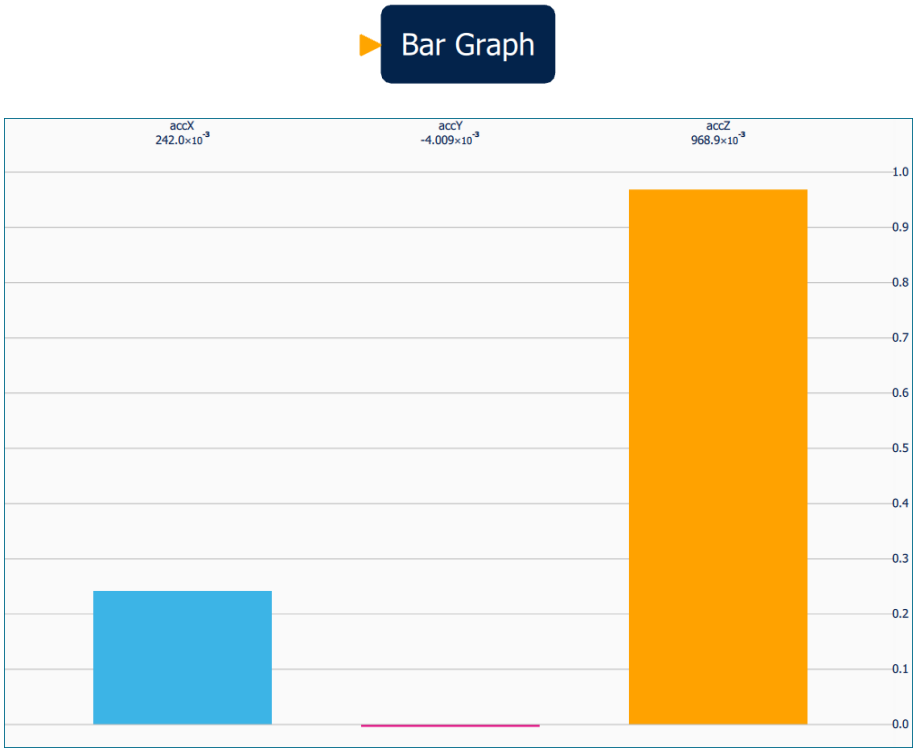
Use the [**Angle Level**] function block in the design to display data as an angle level indicator. This graph works with any floating or integer value and each of them can have up to three indicators. In the [**Properties**] field, you can set the name of the graph as well as the name of each angle level indicator with unit. Predefined configurations are available in the [**Preset Values**] property.

Figure 109. Angle Level function block and example of data visualization



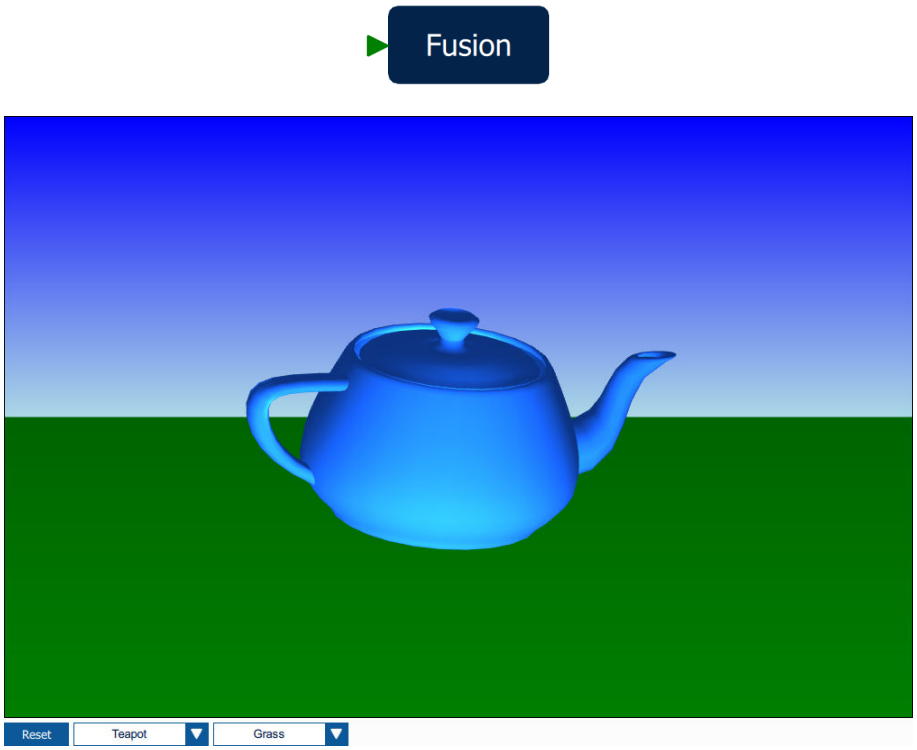
Use the [**Bar Graph**] function block in the design to display data as a bar graph. A bar graph is suitable for a quick check of an actual value without needing to see the history. This graph works with any floating or integer value and each of them can have up to six bars. In the [**Properties**] field, you can set the name of the graph as well as the name of each bar, unit on the Y-axis, position of 0 on the Y-axis, full scale, and enable or disable autoscale. Predefined configurations are available in the [**Preset Values**] property.

Figure 110. Bar Graph function block and example of data visualization



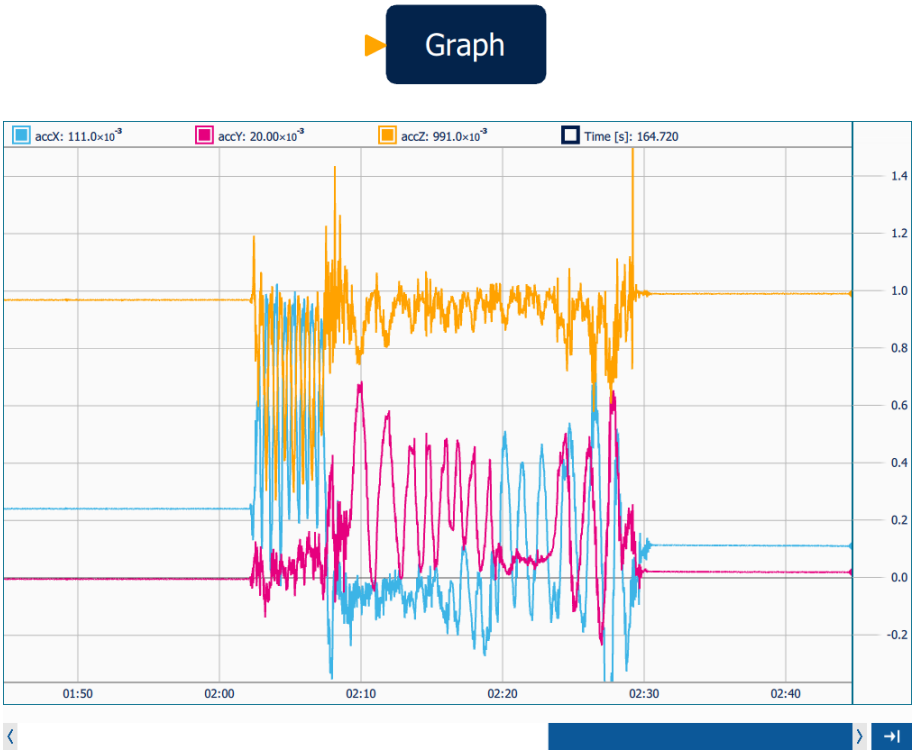
Use the [**Fusion**] function block in the design to display quaternion data on a 3D model. The 3D model (teapot, Nucleo board, car, head) is usually used to check device orientation in 3D space. This graph requires quaternions, which can be obtained, for example, from the sensor fusion (MotionFX) algorithm.

Figure 111. Fusion function block and example of data visualization



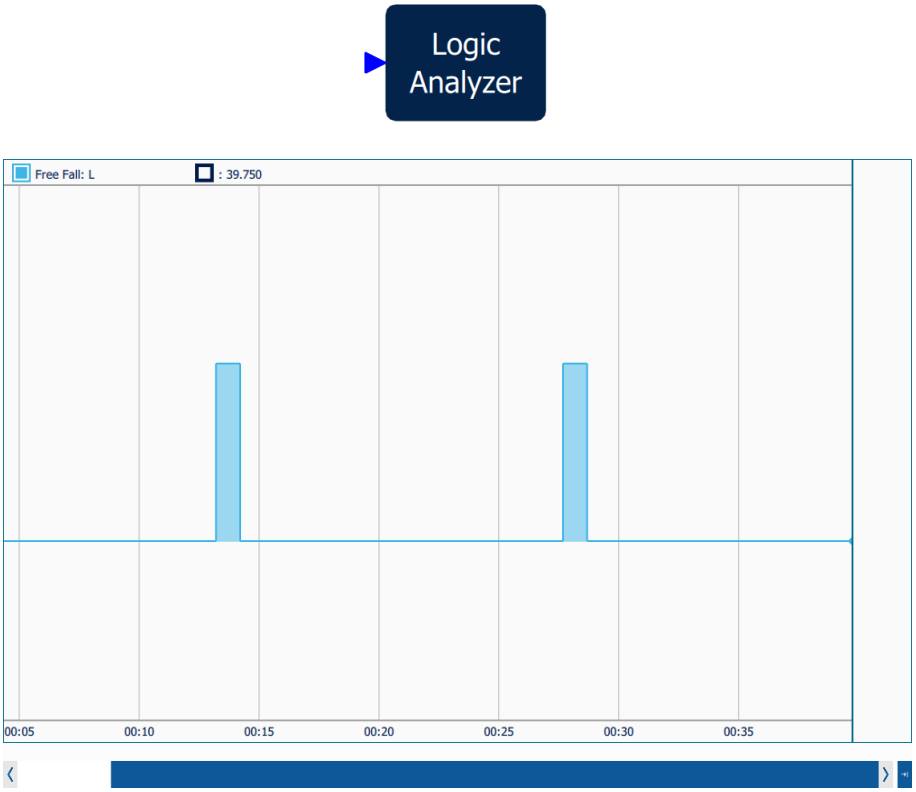
Use the [**Graph**] function block in the design to display data as a time graph. This graph works with any floating or integer value and each of them can have up to six waveforms. In the [**Properties**] field, you can set the name of the graph as well as the name of each waveform, unit on the Y-axis, position of the 0 on the Y-axis, full scale, and enable or disable autoscale. Predefined configurations are available in the [**Preset Values**] property.

Figure 112. Graph function block and example of data visualization



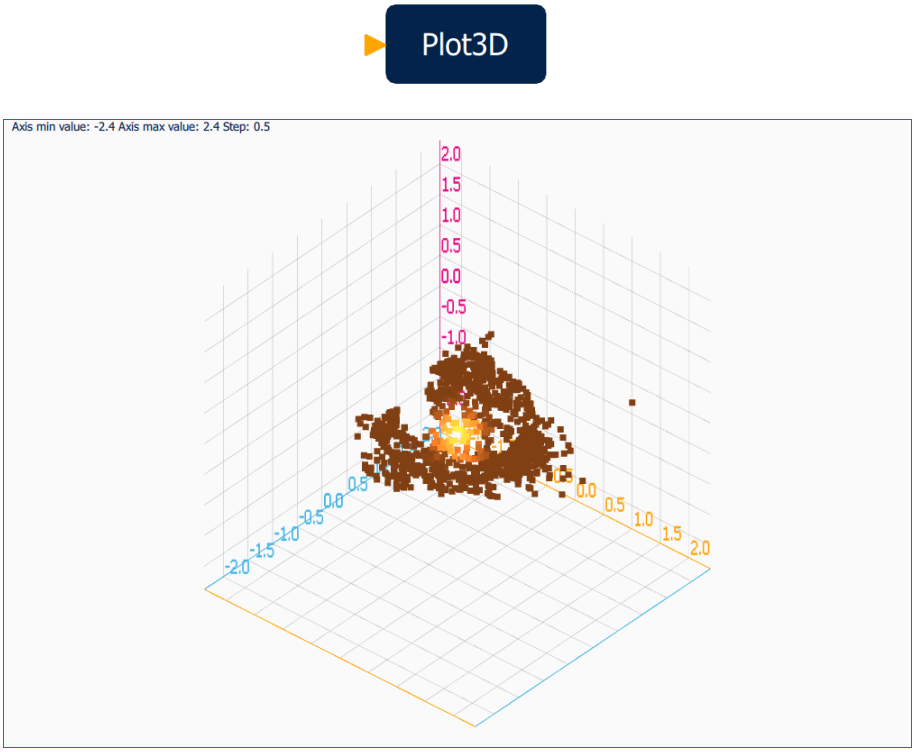
Use the [**Logic Analyzer**] function block in the design to display logic signals, which can have values of only 0 or 1. The logic analyzer can have up to 16 channels. In [**Properties**], you can change the name of each channel and the logic analyzer.

Figure 113. Logic Analyzer function block and example of data visualization



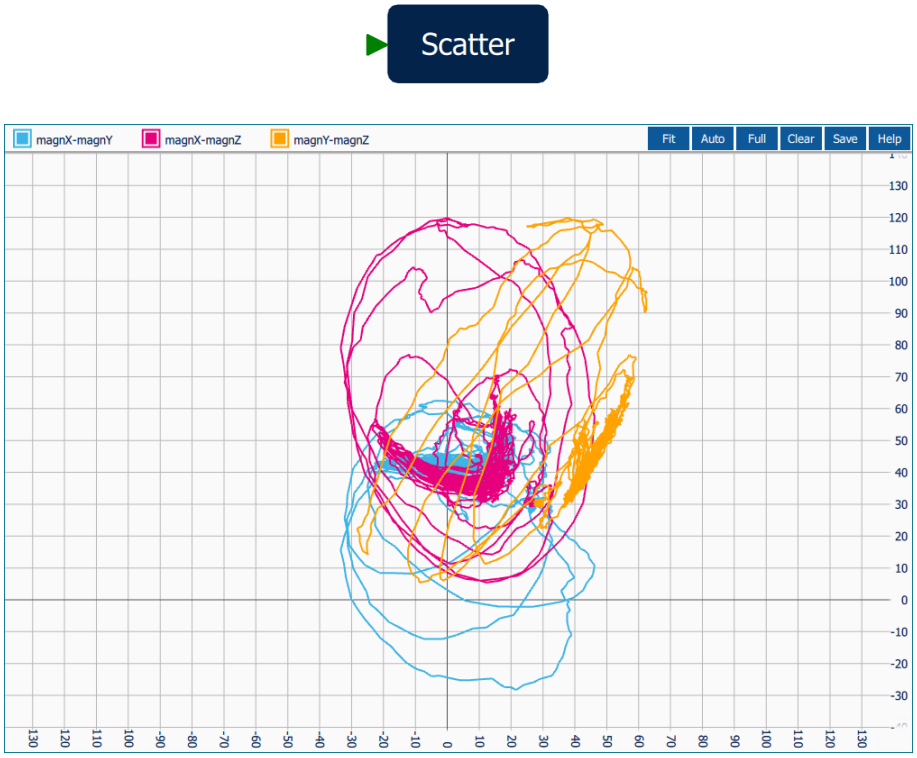
Use the [**Plot3D**] function block in the design to display X, Y, Z data as points in 3D space. This graph works with any floating or integer value. In the [**Properties**] field, you can set the name of the graph as well as the name of each value, unit, full scale, and enable or disable autoscale. Predefined configurations are available in the [**Preset Values**] property.

Figure 114. Plot3D function block and example of data visualization



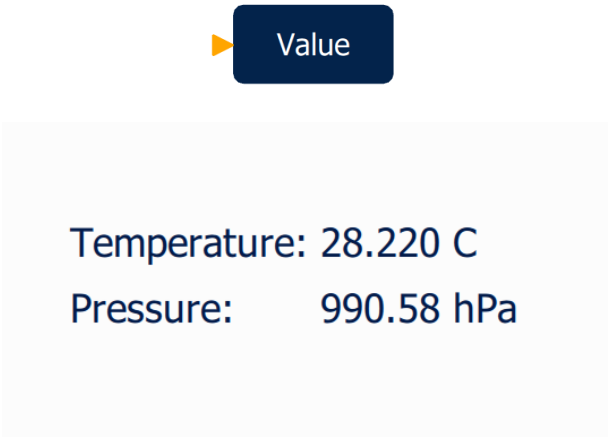
Use the [**Scatter**] function block in the design to display X, Y, Z data in 2D space (X-Y, X-Z, Y-Z charts). In the [**Properties**] field, you can set the name of the graph as well as the name of each value, unit, full scale, and enable or disable autoscale. Predefined configurations are available in the [**Preset Values**] property.

Figure 115. Scatter function block and example of data visualization



Use the [**Value**] function block in the design to display the exact float or integer value as text. Each function block can display up to eight values. In [**Properties**], you can change the name of the value and unit for each item.

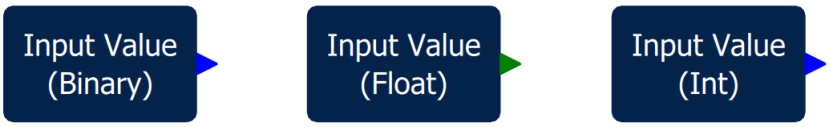
Figure 116. Value function block and example of data visualization



7.12 Data input

Three types of data can be sent from the MEMS Studio application to run the firmware: binary, integer, and float. This can be done by adding the [**Input Value**] function block (with the appropriate type) from the [**User Input**] library to the design. Each [**Input Value**] function block can represent up to four values. In the [**Properties**], it is possible to set the name of each value and the default value. The [**Input Value**] function block output size is defined by the [**Number of Values**] property.

Figure 117. Input Value function blocks



7.13 Conditional execution

In some cases, it is needed to execute the function block operation only if a certain condition is valid. For this case, it is possible to add [**Conditional Execution Input**] to the selected function block. This input then defines if the function block code is executed or not. To add or remove the conditional execution input to the selected function block, click on the (Add or Remove Conditional Input) icon in the toolbar.

7.14 Fast Fourier transform (FFT)

AlgoBuilder offers a function block for the frequency analysis of the sensor's output signal using FFT (fast Fourier transform). Fourier analysis converts a signal from the time domain to a representation in the frequency domain.

The [**FFT**] function block offers frequency analysis from 32, 64, 128, 256, 512 and 1024 samples. It is also possible to enable window usage to eliminate spectrum leakage. Hanning, Hamming, and Flat Top windows can be used.

Output from the [**FFT**] function block can be connected to the [**FFT plot**] function block, which visualizes the data as a frequency spectrum. To plot data only when the FFT calculation is finished, conditional execution input must be added to the [**FFT plot**] and connected to the full output of the [**FFT**] function block.

Figure 118. FFT and FFT Plot connections

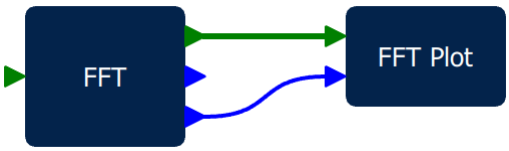
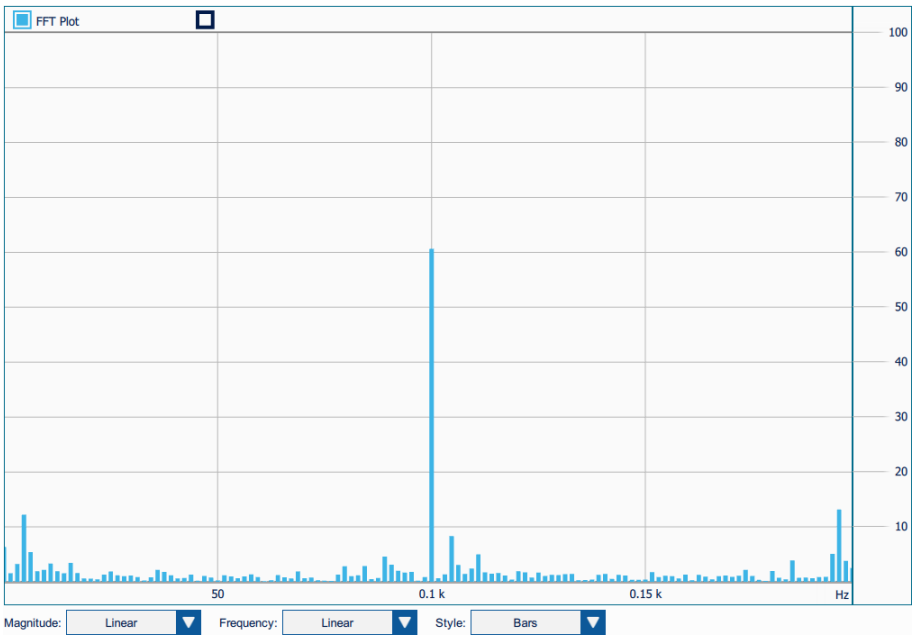


Figure 119. FFT Plot visualization



*Note: To get the correct frequency values, it is necessary to select the sensor whose data are analyzed (accelerometer or gyroscope) in the [**Data Rate Control**] property in the [**Sensor Hub**]. Real sensor ODR (output data rate) is measured during firmware initialization.*

7.15 Finite state machine (FSM) and machine learning core (MLC)

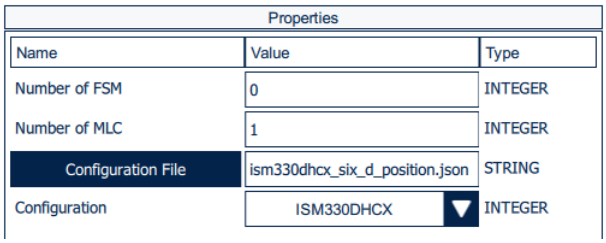
AlgoBuilder allows using outputs from the finite state machine (FSM) and/or machine learning core (MLC) in the design. The [**FSM/MLC**] function block is available in the [**Sensor Hub**] library.

Figure 120. Example of a design with FSM/MLC

The configuration of the FSM and MLC is stored in the .json file. The .json file can be created in the [**Advanced Features**] page.

It is mandatory to specify the number of FSM and MLC outputs used and the sensor name in the .json file in case the .json contains more configurations. The size of the FSM/MLC output vector is adjusted accordingly.

Figure 121. FSM/MLC function block properties



Warning:

The .json file usually contains settings of the sensor full scale (FS) and output data rate (ODR). These settings might be different than the settings required by the sensor hub. The firmware generated by AlgoBuilder contains a function to check if the FS and ODR set by the .ucf file is compatible with the sensor hub settings. If not, an error message is generated during connection to the device.

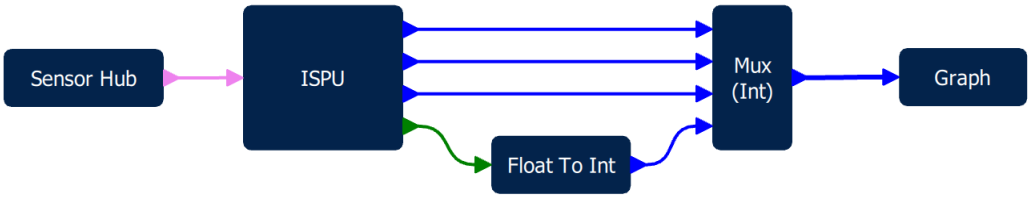
The interrupt pin INT1 is usually configured by the MLC tool in which case it is not possible to use the accelerometer or gyroscope as [**Data Rate Control**] in the sensor hub.

The [**FSM/MLC**] function block can be used only with sensors that are equipped with this functionality.

7.16 Intelligent sensor processing unit (ISPU)

AlgoBuilder also allows using outputs from the intelligent sensor processing unit (ISPU) in the design. The [**ISPU**] function block is available in the [**Sensor Hub**] library.

Figure 122. Example of a design with ISPU



The configuration of the ISPU is stored in the .json file. The path to the .json file is the [**Property**] of the [**ISPU**] function block. The .json file can be created using the ISPU-Toolchain software. An output description needs to be present in the same file with the ISPU configuration. The outputs of the [**ISPU**] function block are automatically adjusted according to the output description structure in the .json file.

Supported types of outputs are the following:

- int8_t, int16_t, int32_t, uint8_t, uint16_t, uint32_t
- float

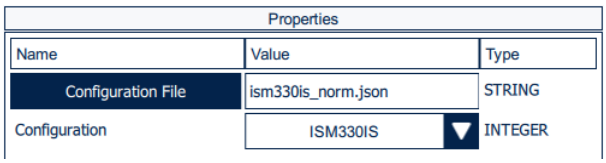
Arrays might be defined using a size key (that is, “size”: 3).

An example of the output description file is as follows:

```
"outputs": [
  {
    "name": "Acc x [LSB]",
    "core": "ISPU",
    "type": "int16_t",
    "len": "1",
    "reg_addr": "0x10",
    "reg_name": "ISPU_DOUT_00_L",
    "results": [
    ]
  },
  {
    "name": "Acc y [LSB]",
    "core": "ISPU",
    "type": "int16_t",
    "len": "1",
    "reg_addr": "0x12",
    "reg_name": "ISPU_DOUT_01_L",
    "results": [
    ]
  },
  {
    "name": "Acc z [LSB]",
    "core": "ISPU",
    "type": "int16_t",
    "len": "1",
    "reg_addr": "0x14",
    "reg_name": "ISPU_DOUT_02_L",
    "results": [
    ]
  },
  {
    "name": "Acc norm [LSB]",
    "core": "ISPU",
    "type": "float",
    "len": "1",
    "reg_addr": "0x16",
    "reg_name": "ISPU_DOUT_03_L",
    "results": [
```

```
      ]
   }
]
```

Figure 123. ISPU function block Properties



During firmware startup, the ISPU is configured according to the .json file and the output values are then read by the MCU and used in the AlgoBuilder flow.

Warnings:

The .json file also usually contains settings of the sensor full scale (FS) and output data rate (ODR). These settings might be different than the settings required by the sensor hub. The firmware generated by AlgoBuilder contains a function to check if the FS and ODR set by the .ucf file is compatible with the sensor hub settings. If not, an error message is generated.

If the interrupt pin INT1 is used by the ISPU algorithm, it is not possible to use the accelerometer or gyroscope as [**Data Rate Control**] in the [**Sensor Hub**].

The [**ISPU**] function block can be used only with sensors that are equipped with this functionality.

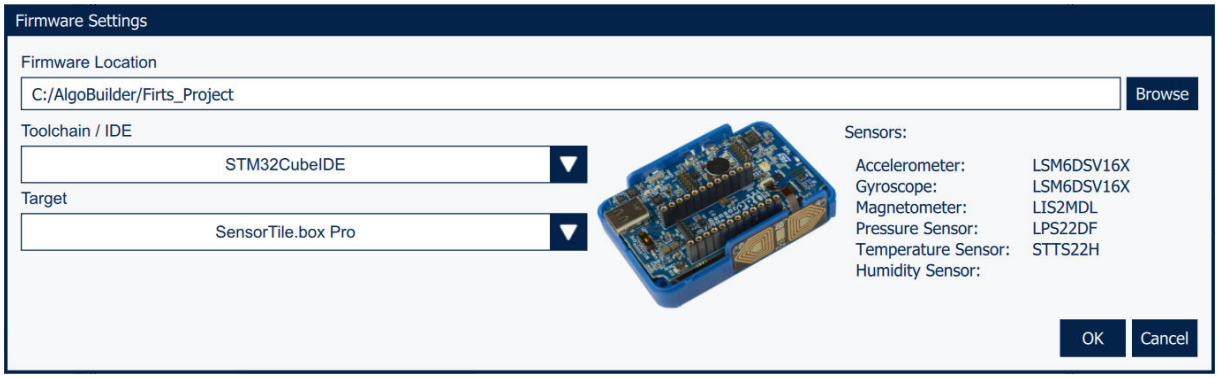
7.17 Creating your first design

As a first example, you can create a simple design to read acceleration data from the accelerometer sensor at a selected data rate and visualize the data in a line chart.

Step 1. Start with a new design by clicking on the icon (New Design) in the toolbar. The [**Firmware Settings**] window is opened automatically or you can open it by clicking on the icon (Firmware settings) in the toolbar.

Step 2. Set the path to the directory where the output firmware will be located, the toolchain to be used to compile the firmware, and the target to be used for testing.

Figure 124. AlgoBuilder Firmware Settings window



Step 3. [**Sensor Hub**] function block from the [**Sensor Hub**] library is automatically added to the workspace.

*Note: Each design must start with the [**Sensor Hub**] function block, which provides access to the sensors and prebuild algorithm.*

Step 4. Adjust the [**Sensor Hub**] properties. Select [**Timer**] as the source for [**Data Rate Control**], set the [**Data Rate**] to 50 Hz and [**Accelerometer Full Scale**] to 2 *g*.

Step 5. Add the [**Acceleration**] [g] function block from the [**Sensor Hub**] library and the [**Graph**] function block from the [**Display**] library to the workspace.

Figure 125. Sensor Hub, Acceleration [g], Graph function blocks

Step 6. Adjust the [**Graph**] properties. The [**Number of Curves**] in [**Graph**] defines the size of its input. The value needs to be changed to 3 to match the [**Acceleration**] [*g*] output size. The graph, waveforms, and unit names can also be changed. You can also quickly configure the [**Graph**] by selecting [**Accelerometer**] in [**Preset Values**].

Figure 126. Graph Properties

Properties

Name	Value	Type
Preset Values	Accelerometer	INTEGER
Number of Curves	3	INTEGER
Name	Acceleration	STRING
Waveform 1 Name	accX	STRING
Waveform 2 Name	accY	STRING
Waveform 3 Name	accZ	STRING
Unit Name	g	STRING
Zero axis position	Middle	INTEGER
Auto-scale	ON	INTEGER
Full Scale	1	STRING

Step 7. Connect the function blocks by clicking on the [**Sensor Hub**] output, holding and mousing over the input of the [**Acceleration**] [*g*] block.

Step 8. Repeat the previous step to connect [**Acceleration**] [*g*] to the [**Graph**] block.

Figure 127. Connected Sensor Hub, Acceleration, Graph function blocks

Step 9. Save the design by clicking on the icon (Save Design) in the toolbar.

Step 10. Your design is ready and you can generate C code from it. Click on the icon (Generate C Code) in the toolbar.

This function validates the flow diagram and creates the algo_builder.c file, which is the C code representation of the graphical design. You can check the generated C code by clicking on the icon (Show C Code) in the toolbar or [**Firmware**] menu.

Figure 128. Generated code in algo_builder.c file

algo_builder.c

```
#include "algo_builder.h"

const char Identification_String[] = "ID_STRING:First_Design.json,On-line";

/* GENERATED CODE START - Variables */
void *Sensor_Hub_1_out;
float Acceleration_1_data[3];

/* GENERATED CODE END - Variables */

/* GENERATED CODE START - Functions */
/* GENERATED CODE END - Functions */

/* GENERATED CODE START - Display Info */
sDISPLAY_INFO display_info_list[] = {
{INFO_TYPE_GRAPH,1,3,VAR_TYPE_FLOAT,0,"graph|Acceleration|g|1|19|accX|accY|accZ||||1",0},
{0,0,0,0,0,0,0}};
/* GENERATED CODE END - Display Info */

/* GENERATED CODE START - Init */
void AB_Init(void)
{
  Sensor_Hub_Init(0, 50, 0, 1);
  Accelero_Init();
  Message_Length = 12;
}
/* GENERATED CODE END - Init */

/* GENERATED CODE START - Main loop */
void AB_Handler(void)
{
  Sensor_Hub_Handler(&Sensor_Hub_1_out);
  Accelero_Sensor_GetData(Sensor_Hub_1_out, Acceleration_1_data);
  Display_Update(Acceleration_1_data, &display_info_list[0]);
}
/* GENERATED CODE END - Main loop */
```

Note: In case the firmware template in the target directory is accidentally broken or deleted, you can invoke reinitialization of the firmware template by clicking on the icon (Initialize Project Directory).

Step 11. Click on the icon (Build) to call the external compiler to build the firmware project and generate a binary file for the STM32 microcontroller. The console shows an output from the compiler. Once the compilation finishes, a “Build Process Finished” message appears. If there is no error message from the compiler, the firmware is ready to be programmed in the target board.

Figure 129. Compiler output

Console

```
C:/AlgoBuilder/Firts_Project/Project/Src/usbd_storage_if.c
C:/AlgoBuilder/Firts_Project/Project/Src/vcom.c
Application/STM32CubeIDE/startup_stm32u585xx.o
C:/AlgoBuilder/Firts_Project/Project/CubeIDE/STM32U585AI-SensorTileBoxPro/syscalls.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_buffers.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_display.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_fft.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_sensor_hub.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_sequential_logic.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_signal.c
C:/AlgoBuilder/Firts_Project/Project/Src/ab_user_input.c
SensorTileBoxPro-Project.elf
arm-none-eabi-size   SensorTileBoxPro-Project.elf
   text      data       bss       dec       hex   filename
 239112      3024     64672    306808     4ae78   SensorTileBoxPro-Project.elf
11:03:15 Build Finished. 0 errors, 0 warnings. (took 1m:17s.501ms)
```

7.18 Programming the target

The connected target can be programmed directly from the AlgoBuilder interface. After a successful build of the firmware, the target can be programmed by clicking on the icon (Program target by automatically selected method).

The following programming methods are supported:

- STLINK interface
- DFU (device firmware upgrade)
- Copy to flash drive

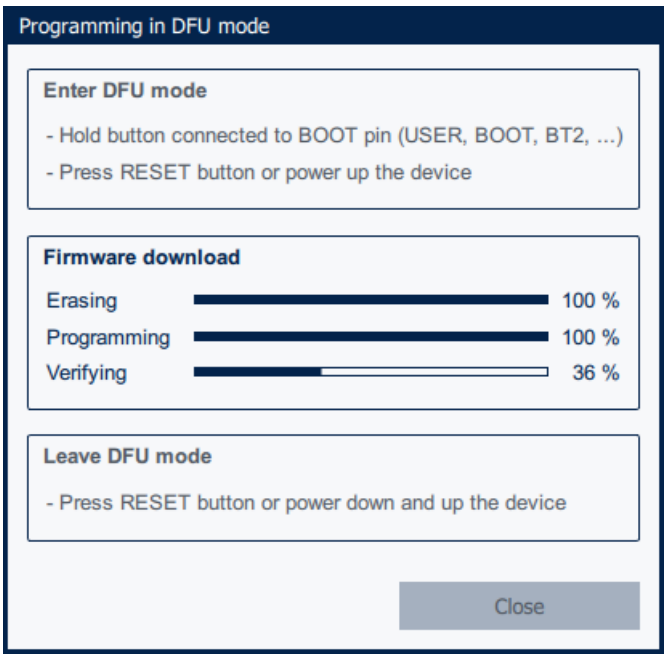
The STM32CubeProgrammer CLI application is used to program the STM32 microcontroller with an STLINK or DFU interface. The path to the STM32CubeProgrammer CLI application must be set in the MEMS Studio settings in the [**External Tools**] page. Copying to the flash drive is available for STM32 Nucleo boards and does not require an STM32CubeProgrammer CLI application.

The programming method is automatically selected with the following priority:

1. STLINK interface
2. Copy to flash drive
3. DFU (device firmware upgrade) interface

DFU mode allows programming the target without requiring a programmer. DFU mode is available for devices with a USB interface, for example SensorTile.box Pro. By pressing the [**Program**] button, a dedicated window for DFU is opened. There are two options for switching the device to DFU mode. The procedures are described in the window. After the device is switched to DFU mode, the memory is automatically erased and programmed by the new firmware.

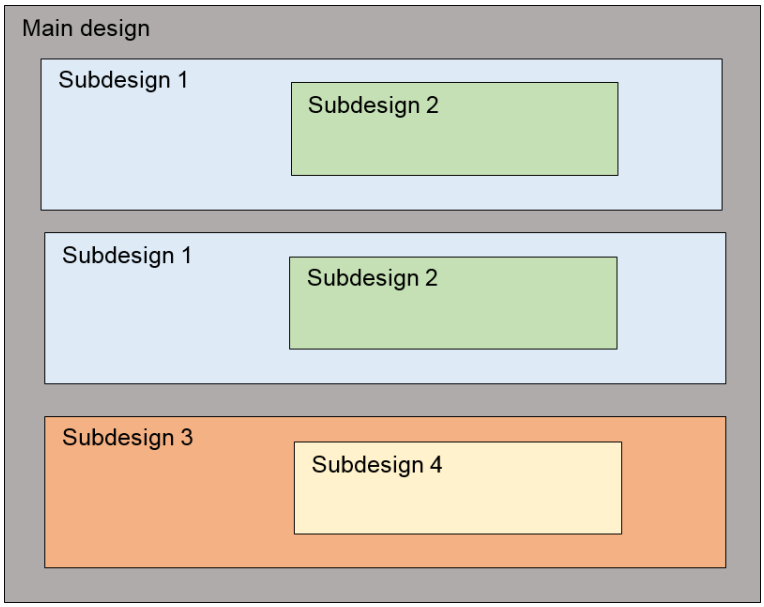
Figure 130. Programming target with DFU interface



7.19 Subdesign

AlgoBuilder offers the possibility to create a multilevel design. This feature allows encapsulating part of the design into a standalone part called subdesign. This subdesign is represented by a single function block and can be used multiple times in the main design and in all other designs. The subdesign can be shared between users. The subdesigns significantly increase the clarity of a highly complex project.

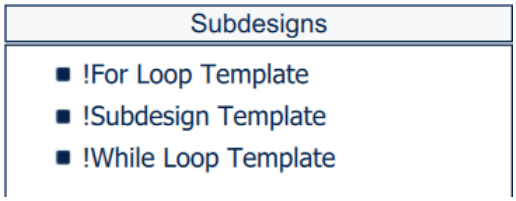
Figure 131. Principle of multilevel design



The subdesign allows implementing loops. By default AlgoBuilder offers three templates for subdesign creation:

- [**!Subdesign Template**]: a generic empty subdesign
- [**!For Loop Template**]: a subdesign intended for a loop, which is used if the number of iterations is known before entering the loop
- [**!While Loop Template**]: a subdesign intended for a loop, which is executed until a defined condition is valid

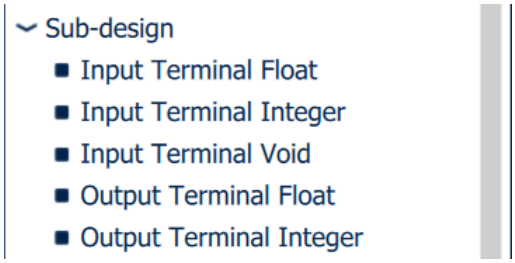
Figure 132. Subdesigns dock



Subdesigns can be created by double-clicking on the particular template. Existing subdesigns, listed in the [**Subdesigns**] dock or used in the design, can be opened in the same way. If a new or an existing subdesign is open, AlgoBuilder creates a dedicated tab in the workspace.

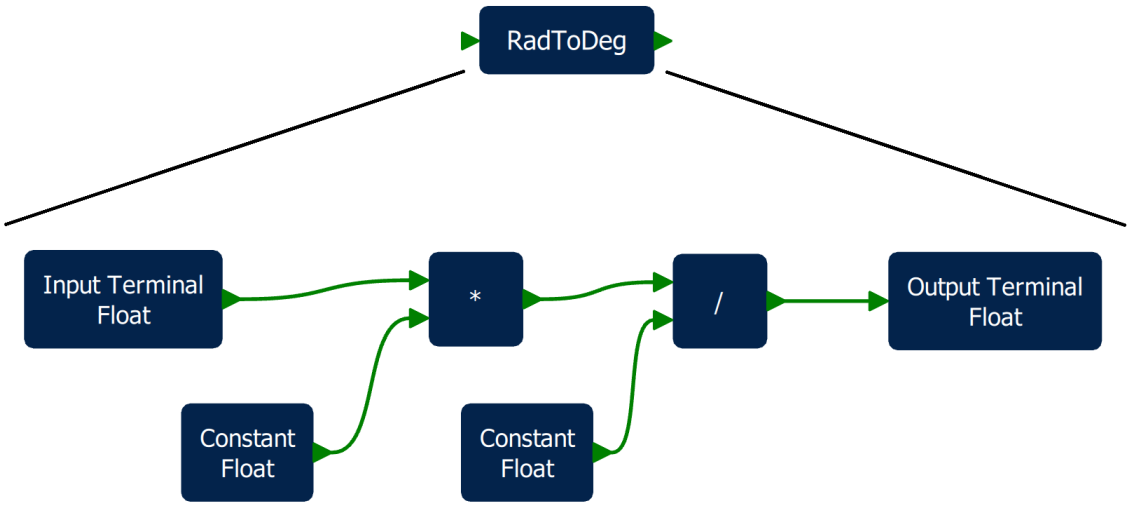
The subdesign can be saved in the same way as the main design by clicking on the icon (Save Design) or (Save Design As). The subdesign can then be used in the main design or other subdesign. To use the subdesign, drag and drop the particular item from the [**Subdesigns**] dock to the workspace. To be able to connect a subdesign with another function block, input and output terminals must be defined for each subdesign. The terminals can be added from the [**Subdesign**] library, which is active only if a subdesign is being edited.

Figure 133. **Subdesign terminals**



Each terminal has three properties: size, name, and description. The size and name must be defined, moreover the name must be unique in the subdesign. The description is optional. In the graphical design, the subdesign is represented by the same rectangle with inputs and outputs as function blocks.

Figure 134. **Example of a subdesign block and its content**



Note: The following two cases of a subdesign are not allowed:

- *A subdesign shall not be embedded within itself as this would create recursive calls and an infinite loop. This is prevented by checking the subdesign name.*
- *A subdesign 2 is permitted to be embedded within a subdesign 1 (refer to Figure 131. Principle of multilevel design), but subdesign 1 shall not be further embedded in the subroutine.*

The templates for the [**for**] and [**while**] loop contain several function blocks that are mandatory. These blocks cannot be deleted. Mandatory function blocks consist of the following:

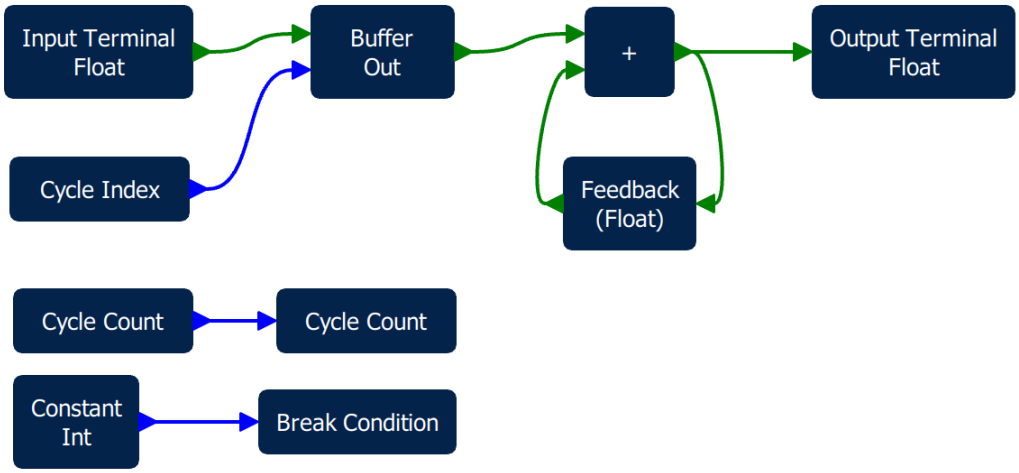
[**for**] loop:

- [**Cycle Count**] (input): defines the number of iterations that are executed
- [**Cycle Index**] (output): returns the number of current iterations
- [**Break Condition**] (input): allows terminating the loop execution before reaching the cycle count

[**while**] loop:

- [**Cycle Index**] (output): returns the number of current iterations
- [**Break Condition**] (input): terminates the loop execution

Figure 135. **Example of loop (calculates the sum of all items in the input buffer)**



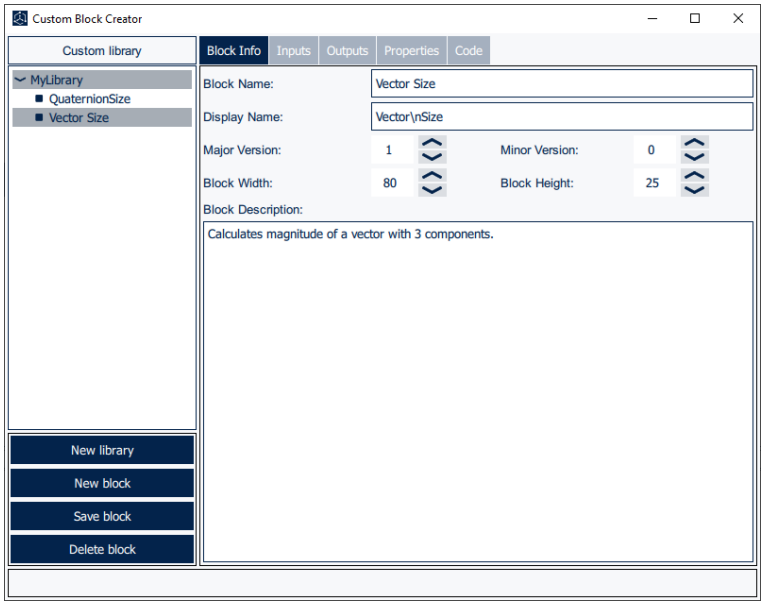
7.20 Custom Block Creator

AlgoBuilder offers a Custom Block Creator that allows creating user-defined function blocks. The [**Custom Block Creator**] can be opened from the toolbar by pressing the button. User-defined function blocks are stored in the user libraries, which are located in the user's home directory \STMicroelectronics\MEMS Studio\algobuilder\library. The Custom Block Creator also offers the possibility to edit or delete an already existing custom function block and custom libraries.

The custom block configuration is divided into several sections. In the [**Block Info**] section, [**Block Name**], [**Display Name**], [**Major Version**], [**Minor Version**], and [**Block Description**] can be defined.

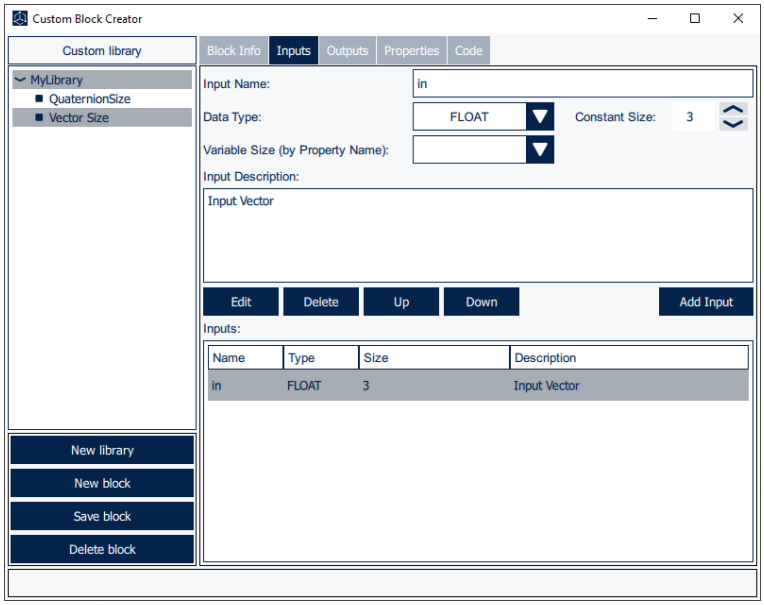
*Note: **[Display Name]** can be redivided into several lines using **\n** separator.*

Figure 136. **Custom Block Creator - Block Info**



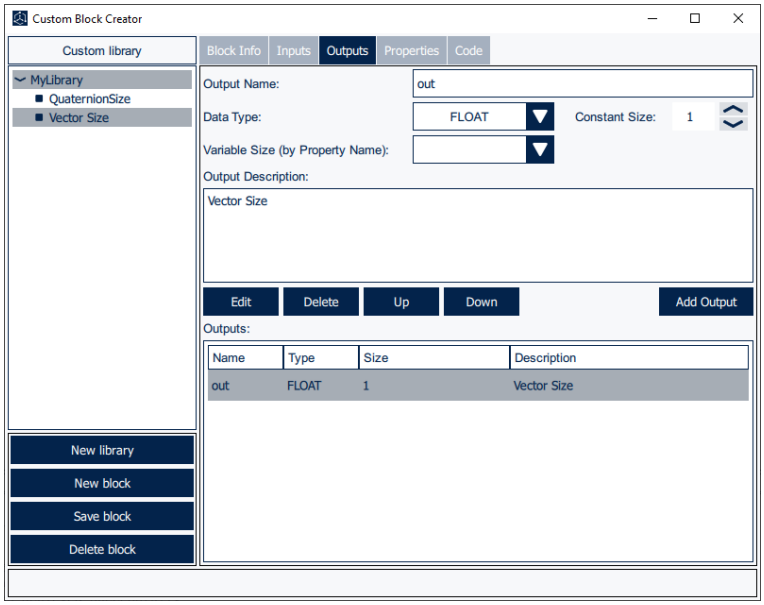
In the [**Inputs**] section, [**Input Name**], [**Data Type**], [**Constant Size**], [**Variable Size**], and [**Input Description**] can be defined. The input parameters as well as the input order can be modified.

Figure 137. Custom Block Creator - Inputs



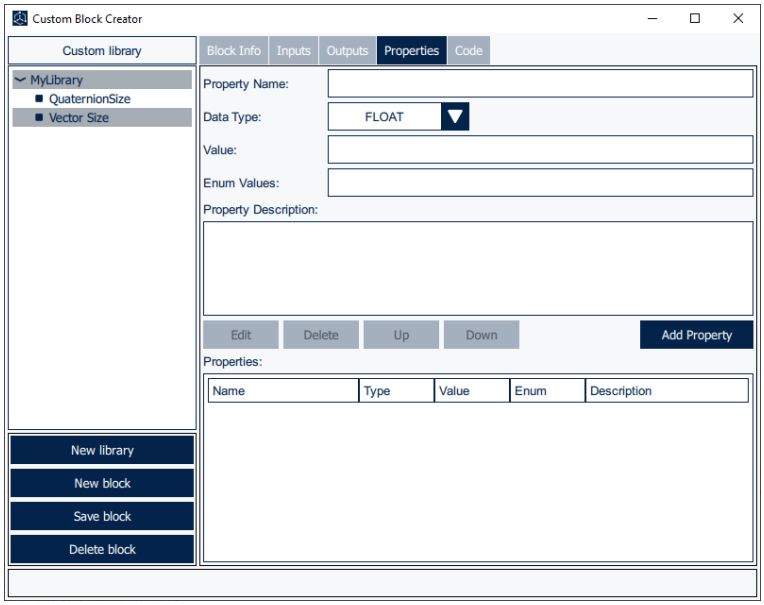
In the [**Outputs**] section, [**Output Name**], [**Data Type**], [**Constant Size**], [**Variable Size**], and [**Output Description**] can be defined. The output parameters as well as the output order can be modified.

Figure 138. Custom Block Creator - Outputs



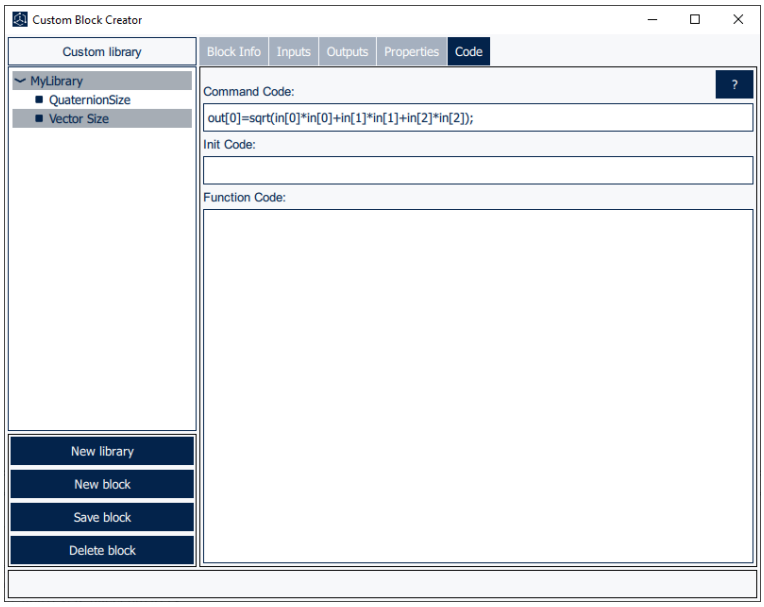
In the [**Properties**] section, [**Property Name**], [**Data Type**], [**Value**] (or [**Enum Values**]), and [**Property Description**] can be defined. The property parameters as well as the property order can be modified.

Figure 139. Custom Block Creator - Properties



In the [**Code**] section, a C code that represents the function block functionality can be entered. The input and output are referenced in C code using NAME[index] where NAME is the name of the input or output and index is the number of the item in the array. All inputs and outputs are represented by array so the index is mandatory even if only one item is in the array. Properties are scalar so the index is not used and only the property name is used as the reference.

Figure 140. Custom Block Creator - Code



8 Firmware programming

The [**Firmware Programming**] page offers quick development board (STM32 microcontroller) programming in the example to update firmware in the ProfiMEMS board. The following programming methods are supported:

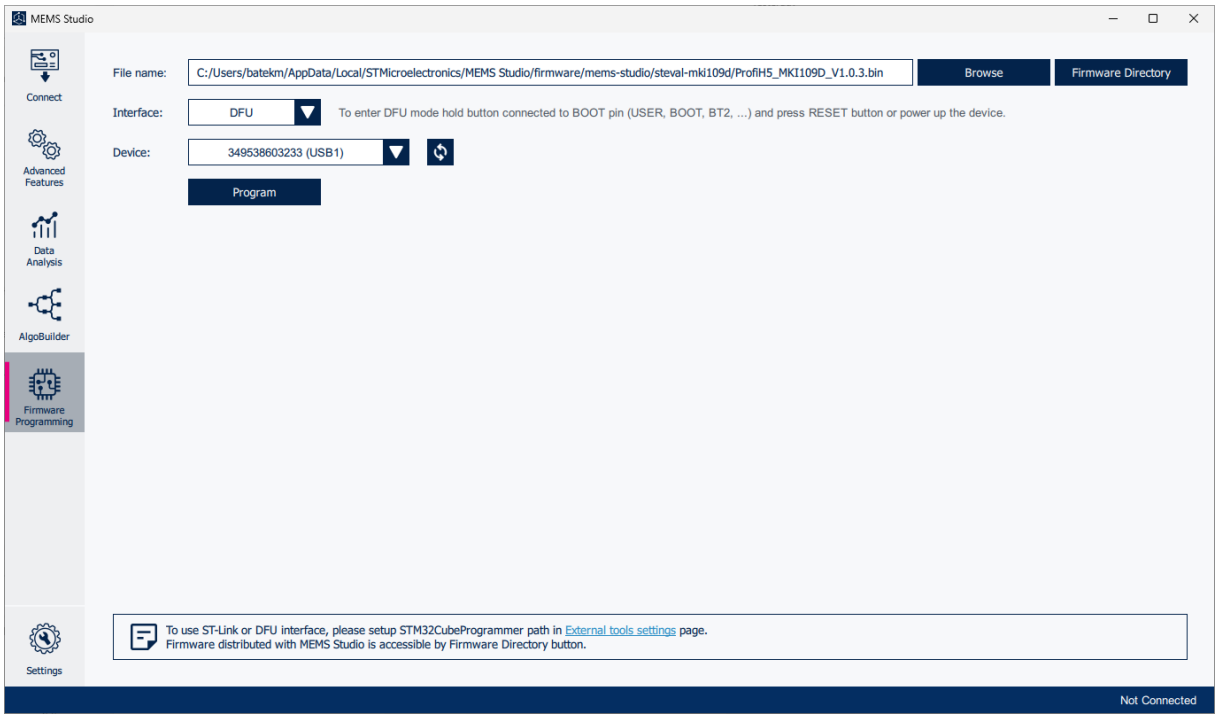
- STLINK interface
- DFU (device firmware upgrade)
- Mass storage

The STM32CubeProgrammer CLI application is used to program the STM32 microcontroller with an STLINK or DFU interface. The path to the STM32CubeProgrammer CLI application must be set in the MEMS Studio settings in the [**External Tools**] page. Copying the binary file with the firmware to the mass storage drive is available for the STM32 Nucleo boards and does not require the STM32CubeProgrammer CLI application.

The user first selects the binary file with the firmware for the STM32 microcontroller using the [**Browse**] button. Then the user selects the programming interface. Based on the selected interface, the list of devices is populated. The list can be updated by pressing the [**Refresh**] button. Programming is executed by clicking on the [**Program**] button.

The [**Firmware Directory**] button offers quick access to the folder where the firmware distributed together with MEMS Studio is located.

Figure 141. Firmware programming page



DFU mode allows programming the target without requiring a programmer. DFU mode is available for devices with a USB interface, for example the ProfiMEMS board (STEVAL-MKI109D, STEVAL-MKI109V3).

Revision history

Table 11. **Document revision history**

Date	Version	Changes
16-Nov-2023	1	Initial release
26-Mar-2024	2	Updated list of supported devices in Section 2.6: Hardware and firmware compatibility Updated Section 5.2: Machine learning core (MLC) tool
23-May-2024	3	Updated list of supported devices in Section 2.6: Hardware and firmware compatibility Added Section 7.20: Custom Block Creator
23-Jul-2024	4	Updated Section 2.3: External Tools Updated list of supported devices in Section 2.6: Hardware and firmware compatibility Updated Section 3: Sensor evaluation Updated Section 5: Advanced features Added Section 5.4: ISPU Model Converter Updated most of the figures to reflect the latest MEMS Studio application
18-Nov-2024	5	Updated Section 1: Overview Updated Section 2.4: External Tools Updated Section 2.7: Running the application for the first time Updated Section 2.8: Hardware and firmware compatibility Updated Section 3: Sensor evaluation Updated Section 5.2: Machine learning core (MLC) tool Updated Section 5.5: ISPU Model Converter Added Section 5.6: IR Algorithms Tuning
10-Apr-2025	6	Added Section 2.2: myST Login Updated Section 2.3: Application settings Updated Section 2.5: Updater settings Added Section 2.6: Network Settings Updated Section 2.7: Running the application for the first time Updated Section 2.8: Hardware and firmware compatibility Updated Section 3: Sensor evaluation Updated Section 5.1: Finite state machine (FSM) tool Updated Section 5.2: Machine learning core (MLC) tool Added Section 5.3: Merge MLC + FSM tool Updated Section 5.4: Pedometer Updated Section 5.5: ISPU Model Converter Added Section 5.7: Configuration Convertor Updated Section 6.1: Time domain Updated Section 6.2: Frequency domain Updated Section 7.15: Finite state machine (FSM) and machine learning core (MLC) Updated Section 7.16: Intelligent sensor processing unit (ISPU)
14-May-2025	7	Updated Section 2.7: Running the application for the first time Updated Section 2.8: Hardware and firmware compatibility Updated Section 3: Sensor evaluation

Date	Version	Changes
		Updated Section 6.1: Time domain
31-Oct-2025	8	Updated Section 2.8: Hardware and firmware compatibility
18-Feb-2026	9	Updated: Section 2.3: Application settings Section 2.4: External Tools Section 2.8: Hardware and firmware compatibility Section 3: Sensor evaluation Section 5.8: Configuration Converter Section 6.1: Time domain Added: Section 5.5: ISPU IDE Section 5.9: HSDatalog Converter
11-Jun-2026	10	Updated: Section 2.8: Running the application for the first time Section 2.9: Hardware and firmware compatibility Section 3: Sensor evaluation Section 5.1: Finite state machine (FSM) tool Section 5.2: Machine learning core (MLC) tool Section 5.4: Pedometer Section 5.5: ISPU IDE Added: Section 2.7: Features Shortcuts

Contents

List of tables

List of figures