

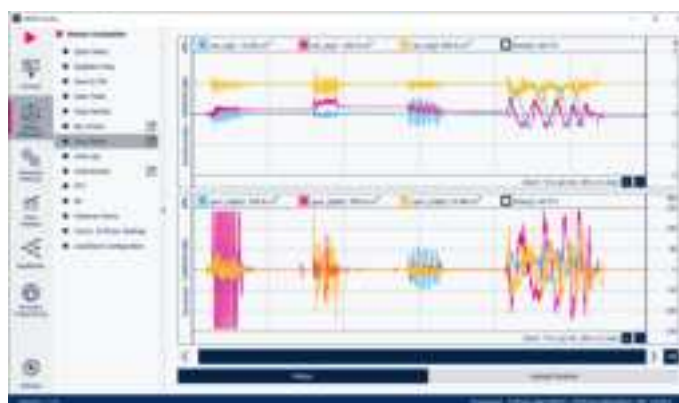
Getting started with MEMS-Studio

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 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
- Support for Windows, macOS, and Linux operating systems
- Network updates with automatic notification of new releases

Note: Firmware for sensor evaluation using STEVAL-MKI109V3 (Professional MEMS tool) and STEVAL-MKBOXPRO (SensorTile.box PRO) is distributed within MEMS-Studio.

2 Getting started

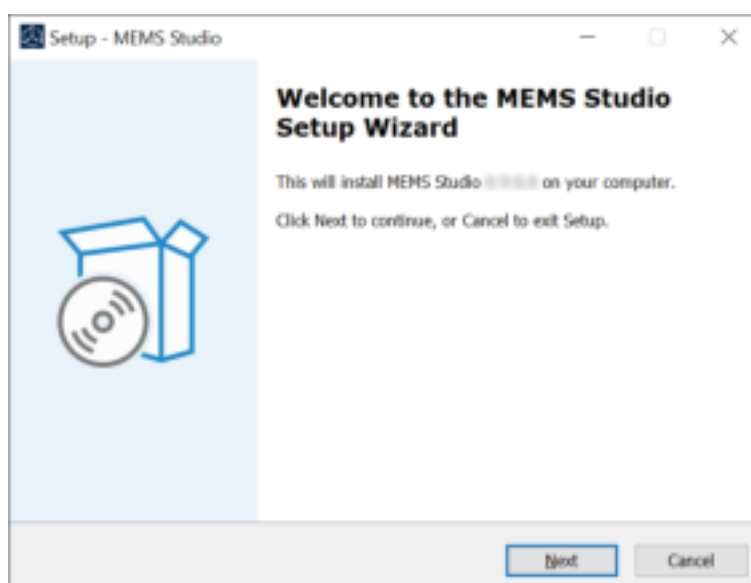
2.1 Installation

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 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.

[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 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.

Due to the limitations of the Qt framework used, **[Font Size]** and **[Application scaling]** might need to be adjusted by the user in the application settings for certain displays to match the user's requirement for application size and good readability. **[Application scaling]** is effective after application restart.

Figure 4. Application Settings



2.3 AlgoBuilder settings

Options related to **[AlgoBuilder Settings]** and **[Firmware Programming]** can be adjusted in the **[AlgoBuilder Settings]** 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 programming in the standalone page or programming from the AlgoBuilder page utilizes the STM32CubeProgrammer CLI tool. **[STM32CubeProgrammer path]** must be specified in order to make the programming available.

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

Figure 5. AlgoBuilder settings



2.4 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. Some functional network parameters have to be properly set in the **[Updater Settings]** 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 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 6. Updater Settings



Check for updates: Manual

Interval between two checks [days]: 0

Proxy server type: Manual configuration of proxy server

Proxy address: PROXY_SERVER_ADDRESS

Proxy port: 8080

Proxy authentication: ☒

Remember my credentials: ☒

User login: USER_NAME

User password: *****

Connection state: Unknown

2.5 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 (see [Section 2.6: 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.

In order to connect to a board, **[Communication type]** needs to be selected. Serial and BLE interfaces are supported. The BLE interface is used only for the AlgoBuilder firmware evaluation. According to the selected communication type, **[Communication port]** 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 7. Sensor selection - ProfiMEMS firmware

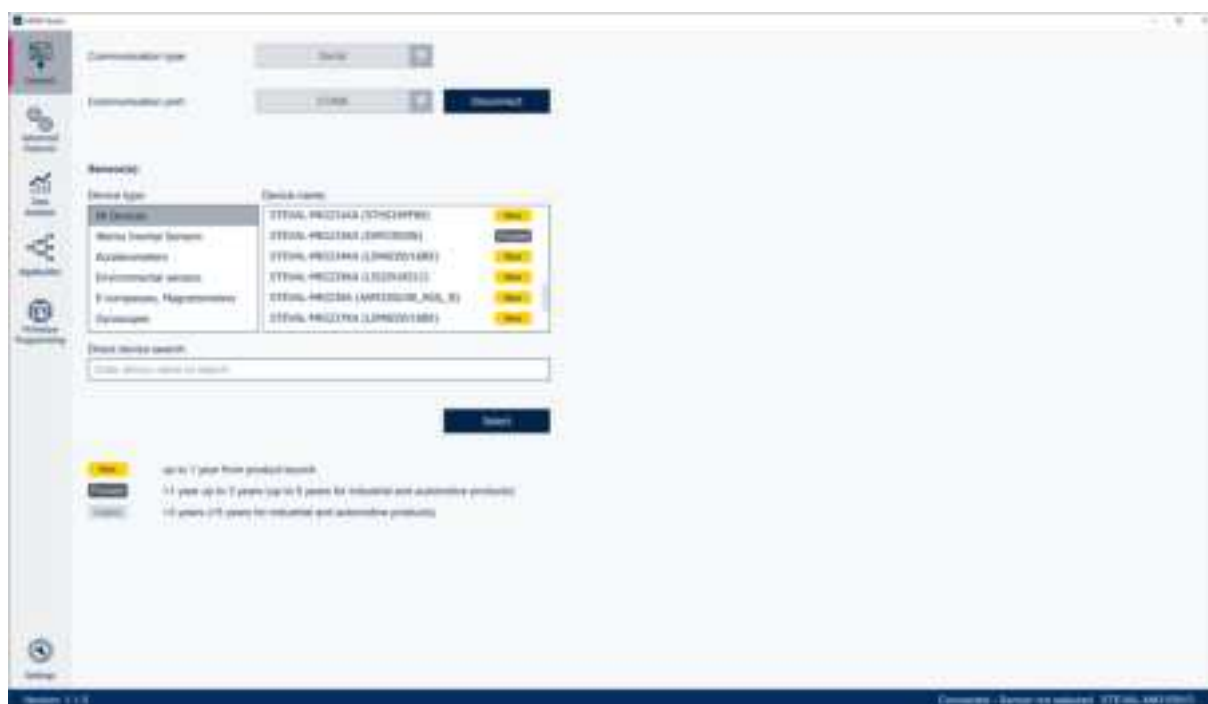


Figure 8. Sensor selection - DatalogExtended firmware


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.

After the sensor selection is done, details about connected boards, sensors, and firmware are displayed including links to all available resources like datasheets, applications notes, websites, GitHub repositories, and others.

Figure 9. Sensor and board references


2.6 Hardware and firmware compatibility

Table 1. Supported hardware and firmware for full sensor evaluation

Hardware	Firmware	Supported sensors
STEVAL-MKI109V3 (Professional MEMS tool)	ProfiMEMSToolVx.y.z.bin (from MEMS Studio)	STEVAL-MKI172V1 (LSM303AGR)
		STEVAL-MKI181V1 (LIS2MDL)
		STEVAL-MKI197V1 (LSM6DSOX)
		STEVAL-MKI200V1K (STTS22H)
		STEVAL-MKI208V1K (IIS3DWB)
		STEVAL-MKI212V1 (ASM330LHHX)
		STEVAL-MKI217V1 (LSM6DSOX, LIS2MDL)
		STEVAL-MKI222V1 (LIS2DU12)
		STEVAL-MKI223V1 (ILPS28QSW)
		STEVAL-MKI224V1 (LPS22DF)
		STEVAL-MKI225A (LPS28DFW)
		STEVAL-MKI227KA (LSM6DSV16X)
		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-MKI240A (LSM6DSV32X)
		STEVAL-MKI243A (ASM330LHHXG1)
		STEVAL-MKI244A (ASM330LHBG1)
		SENSEVAL-SHT4XV1 (SHT40-AD1B)

Hardware	Firmware	Supported sensors
		SENSEVAL-SCB4XV1 (SHT40-AD1B, SGP40-D-R4, LPS22DF)

Table 2. Supported hardware and firmware for reduced sensor evaluation

Hardware	Firmware	Supported sensors
X-NUCLEO-IKS02A1	DatalogExtended.bin (from the X-CUBE-MEMS1 package)	ISM330DHCX IIS2MDC In the DIL24 socket: STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI212V1 (ASM330LHHX)
X-NUCLEO-IKS01A3	DatalogExtended.bin (from the X-CUBE-MEMS1 package)	LSM6DSO LIS2DW12 LIS2MDL LPS22HH HTS221 In the DIL24 socket: STEVAL-MKI185V1 (IIS2MDC) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI222V1 (LIS2DU12) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI224V1 (LPS22DF) STEVAL-MKI225A (LPS28DFW) STEVAL-MKI227KA (LSM6DSV16X) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI235KA (LIS2DUXS12) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI240A (LSM6DSV32X) SENSEVAL-SHT4XV1 (SHT40-AD1B)
X-NUCLEO-IKS4A1	DatalogExtended.bin (from the X-CUBE-MEMS1 package)	LSM6DSV16X LSM6DSO16IS LIS2DUXS12 LIS2MDL LPS22DF STTS22H SHT40-AD1B In the DIL24 socket: STEVAL-MKI179V1 (LIS2DW12)

Hardware	Firmware	Supported sensors
		STEVAL-MKI185V1 (IIS2MDC) STEVAL-MKI192V1 (LPS22HH) STEVAL-MKI196V1 (LSM6DSO) STEVAL-MKI197V1 (LSM6DSOX) STEVAL-MKI207V1 (ISM330DHCX) STEVAL-MKI209V1K (IIS2ICLX) STEVAL-MKI212V1 (ASM330LHHX) STEVAL-MKI222V1 (LIS2DU12) STEVAL-MKI223V1 (ILPS28QSW) STEVAL-MKI225A (LPS28DFW) STEVAL-MKI234KA (LSM6DSV16BX) STEVAL-MKI238A (LIS2DUX12) STEVAL-MKI240A (LSM6DSV32X)
STEVAL-MKBOXPRO (SensorTile.box Pro)	DatalogExtended.bin (from MEMS Studio)	LSM6DSV16X LIS2DU12 LIS2MDL LPS22DF STTS22H

Table 3. Supported hardware and firmware for library evaluations

Hardware	Firmware	Supported sensors
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.

Table 4. Supported hardware and firmware for AlgoBuilder functionality

Hardware	Firmware	Supported sensors
NUCLEO-L476RG X-NUCLEO-IKS01A3	generated in AlgoBuilder	LSM6DSO LIS2MDL LPS22HH HTS221 STEVAL-MKI229KA (LSM6DSO16IS)
NUCLEO-L476RG X-NUCLEO-IKS4A1		LSM6DSV16X LSM6DSO16IS LIS2MDL LPS22DF SHT40-AD1B
NUCLEO-L476RG X-NUCLEO-IKS02A1 STEVAL-MKI230KA		STEVAL-MKI230KA (ISM330IS) IIS2MDC
NUCLEO-F401RE X-NUCLEO-IKS01A3		LSM6DSO LIS2MDL LPS22HH HTS221 STEVAL-MKI229KA (LSM6DSO16IS)
NUCLEO-F401RE X-NUCLEO-IKS4A1		LSM6DSV16X LSM6DSO16IS LIS2MDL LPS22DF SHT40-AD1B
NUCLEO-F401RE 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-STWINKT1 (STWIN)		ISM330DHCX IIS2MDC LPS22HH HTS221
STEVAL-STWINBX1 (STWIN.box)		ISM330DHCX IIS2MDC

Hardware	Firmware	Supported sensors
	generated in AlgoBuilder	ILPS22QS STTS22H STEVAL-MKI230KA (ISM330IS)

3 Sensor evaluation

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. If more evaluation modes are available, they can be set up 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.

Figure 10. 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.

Figure 11. Registers Map page

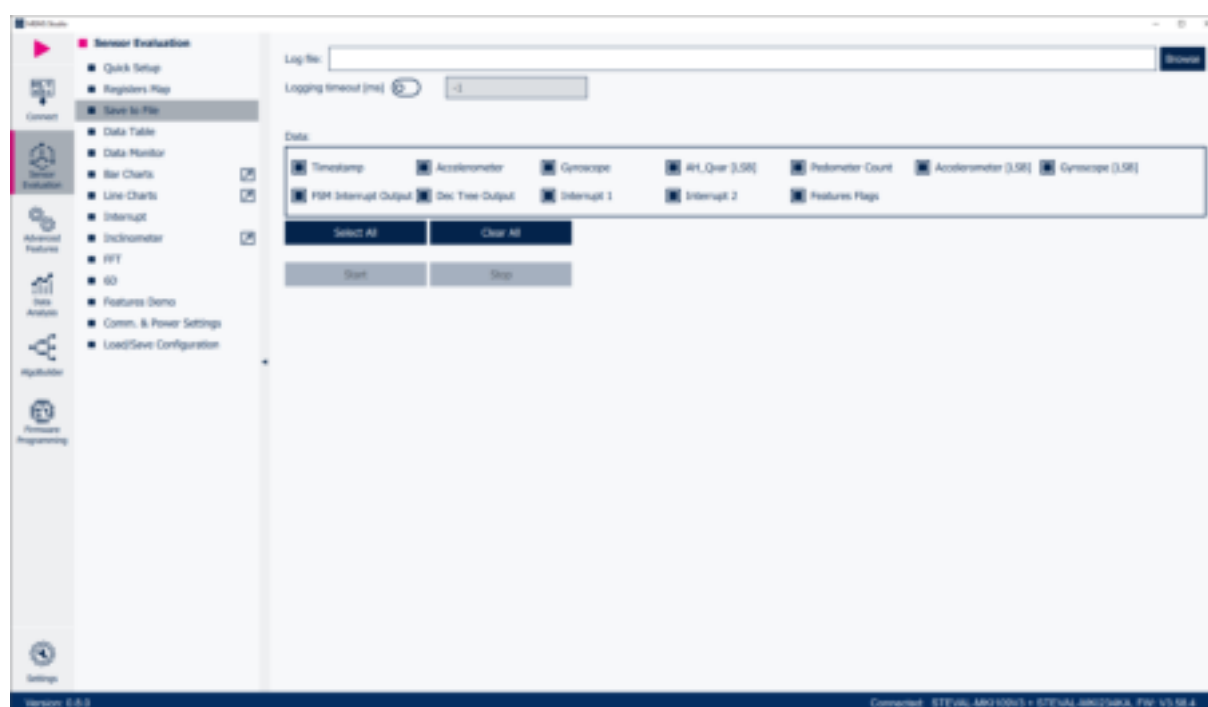
Address	Register Name	Value	Bit Map
0x01	FUNC_CFG_ACCESS	0x00	00000000
0x02	PIN_CTRL	0x22	00100010
0x03	IF_CFG	0x00	00000000
0x07	FIFO_CTRL1	0x00	00000000
0x08	FIFO_CTRL2	0x00	00000000
0x09	FIFO_CTRL3	0x00	00000000
0x0A	FIFO_CTRL4	0x00	00000000
0x0B	COUNTER_BOR_REG1	0x00	00000000
0x0C	COUNTER_BOR_REG2	0x00	00000000
0x0D	INT1_CTRL	0x00	00000000
0x0E	INT2_CTRL	0x00	00000000
0x0F	WHO_AM_I	0x71	01110001
0x10	CTRL1	0x07	00000111
[6:4] - OP_MODE_RL[2:0] Accelerometer operating mode selection. (000: high-performance mode (default); 001: reserved; 010: high-performance mode + TDR; 011: reserved; 100: low-power mode 1 (2 mean); 101: low-power mode 2 (4 mean); 110: low-power mode 3 (8 mean); 111: reserved) [3:0] - ODR_RL[3:0] Accelerometer ODR selection.			
0x11	CTRL2	0x07	00000111
0x12	CTRL3	0x44	01000100
0x13	CTRL4	0x00	00000000
0x14	CTRL5	0x00	00000000
0x15	CTRL6	0x00	00000000

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.

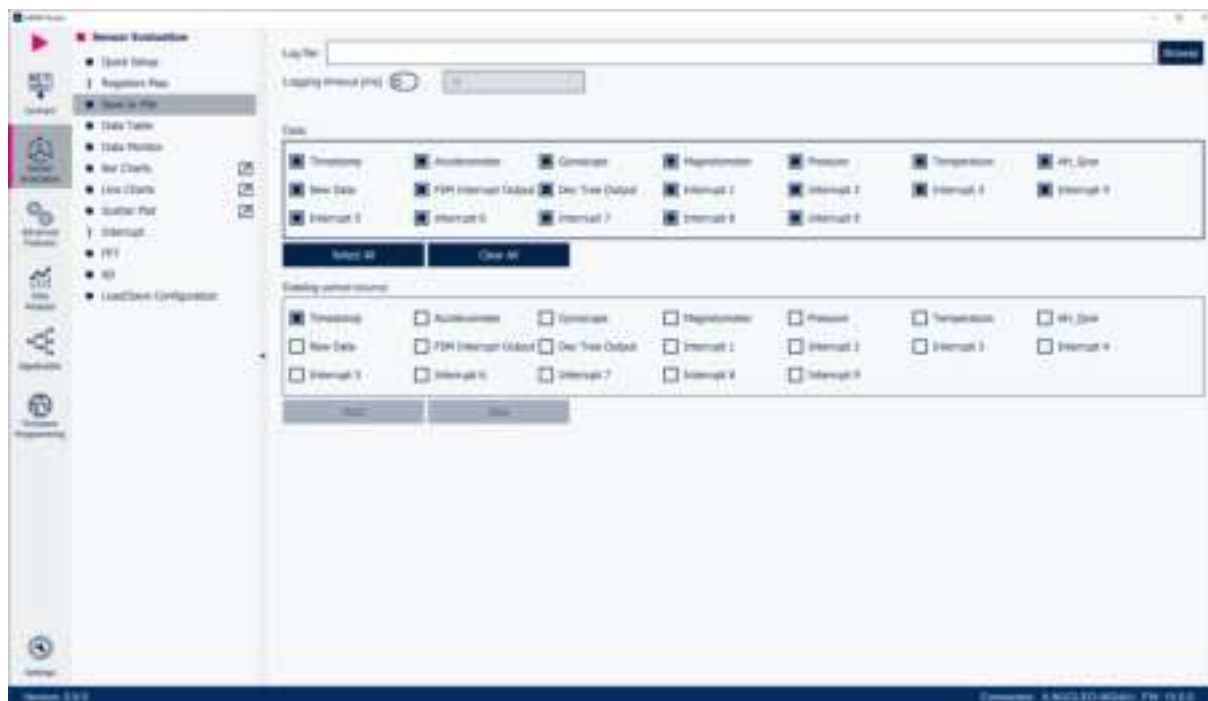
The **[Browse]** button is used to select the folder and insert the file name. The acquisition period can be set to a fixed time in [ms] if desired. Then data to be saved can be selected from the list. The acquisition can be controlled using the **[Start]** and **[Stop]** buttons. If the **[Logging timeout]**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

Figure 12. 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 13. Data log period source



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 14. Data Table tool

The screenshot shows the 'Data Table' tool interface. On the left is a sidebar with navigation options: Quick Setup, Registers Map, Save to File, Data Table (selected), Bar Charts, Line Charts, Interrupt, Inclinometer, FFT, G0, Features Demo, Comm. & Power Settings, and Load/Save Configuration. The main area displays a table of sensor data with columns for timestamp and various sensor outputs. The table is titled 'Sensor Evaluation' and includes checkboxes for 'Timestamp', 'Accelerometer', 'Gyroscope', 'AH_Qvar (3.5R)', 'Pedometer Count', 'Accelerometer (3.5R)', and 'Gyroscope (3.5R)'. Below the table, there are checkboxes for 'FPM Interrupt Output', 'Dec Tree Output', 'Interrupt 1', 'Interrupt 2', and 'Features Flags'.

timestamp	acc_x(mg)	acc_y(mg)	acc_z(mg)	gym_x(mdeg)	gym_y(mdeg)	gym_z(mdeg)	ah_qvar(5R)	pedometer_count	acc_xp(5R)	acc_yp(5R)	acc_zp(5R)
0.000000	952.088	-61.423	23.546	4.375	484.375	301.875	0	0	23628	-1323	386
0.000000	952.088	-61.423	23.546	36.375	484.375	301.875	0	0	23628	-1323	376
0.000000	952.084	-61.429	23.530	41.750	486.000	301.875	0	0	23624	-1323	380
0.000000	952.148	-61.147	23.875	76.750	553.625	325.125	0	0	23608	-1327	375
0.000000	952.074	-61.886	24.156	109.375	551.250	285.125	0	0	23604	-1306	386
0.000000	952.020	-61.623	23.526	36.375	525.750	245.000	0	0	23620	-1302	384
0.000000	952.148	-61.886	23.424	52.500	523.750	306.250	0	0	23608	-1306	384
0.000000	952.490	-62.120	24.627	36.375	553.625	315.000	0	0	23624	-1320	407
0.000000	952.020	-61.886	23.887	21.675	486.750	306.675	0	0	23620	-1306	377
0.000000	952.388	-61.623	23.576	17.000	473.500	321.500	0	0	23628	-1302	376
0.000000	952.476	-62.892	23.856	36.375	562.500	285.125	0	0	23608	-1322	376
0.000000	952.088	-61.327	24.889	36.375	584.375	286.675	0	0	23628	-1407	406
0.000000	952.020	-61.327	23.623	36.625	586.875	271.250	0	0	23620	-1407	371
0.000000	952.084	-62.120	23.280	109.375	558.125	245.625	0	0	23624	-1320	380
0.000000	952.027	-66.524	23.729	74.375	575.125	306.675	0	0	23627	-1409	389
0.000000	952.074	-61.623	23.509	75.000	551.250	321.500	0	0	23624	-1402	389
0.000000	952.247	-61.627	23.682	17.000	584.375	362.125	0	0	23627	-1307	373
0.000000	952.084	-61.623	23.720	15.000	546.875	271.250	0	0	23624	-1402	373
0.000000	951.980	-61.791	23.489	54.875	486.625	245.000	0	0	23620	-1321	383
0.000000	952.088	-61.288	23.812	15.250	562.125	341.250	0	0	23608	-1328	383
0.000000	952.750	-61.886	23.580	6.750	565.125	286.000	0	0	23629	-1308	380
0.000000	952.020	-66.629	23.580	41.750	577.500	306.675	0	0	23620	-1409	380
0.000000	952.720	-62.120	24.889	109.375	586.750	301.875	0	0	23620	-1320	385
0.000000	952.040	-61.886	23.448	87.500	526.625	354.375	0	0	23640	-1306	386
0.000000	952.980	-62.120	24.120	148.750	586.750	315.675	0	0	23621	-1328	376

Version: 0.0.0 Connected: STEVAL-MKI1002 + STEVAL-MKI220A - FW: V3.08.4

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 15. Data Monitor tool

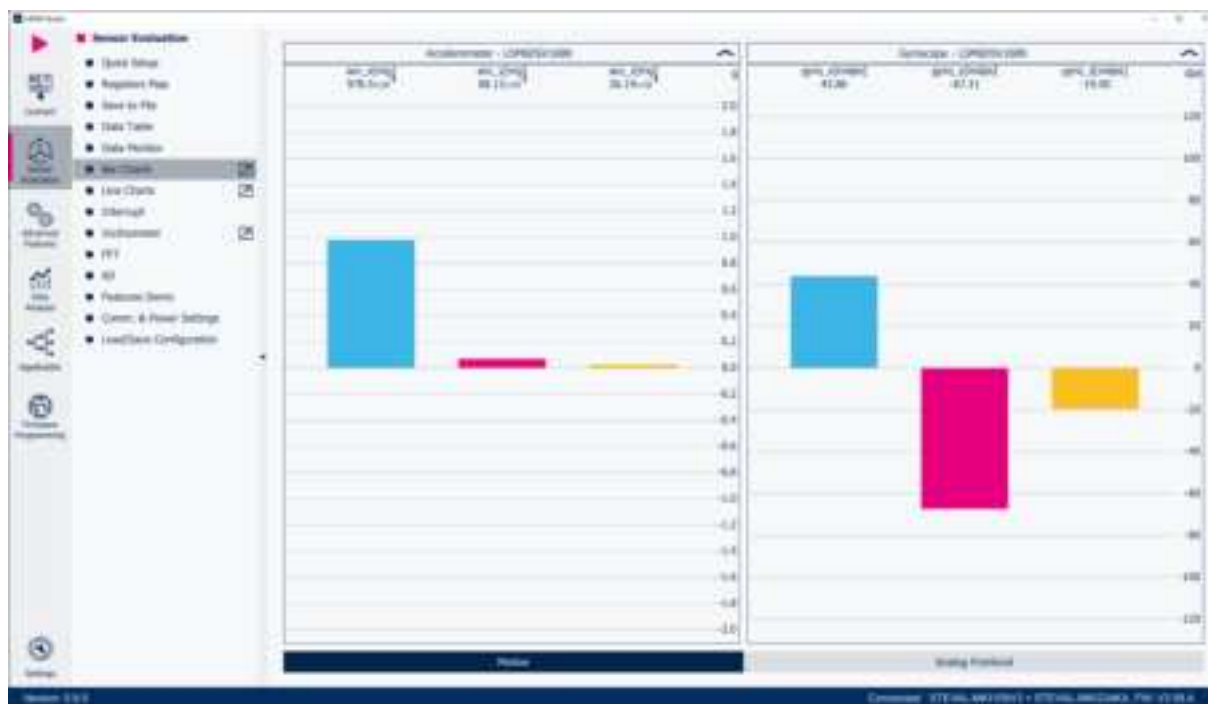


Bar Charts tool

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 16. 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.

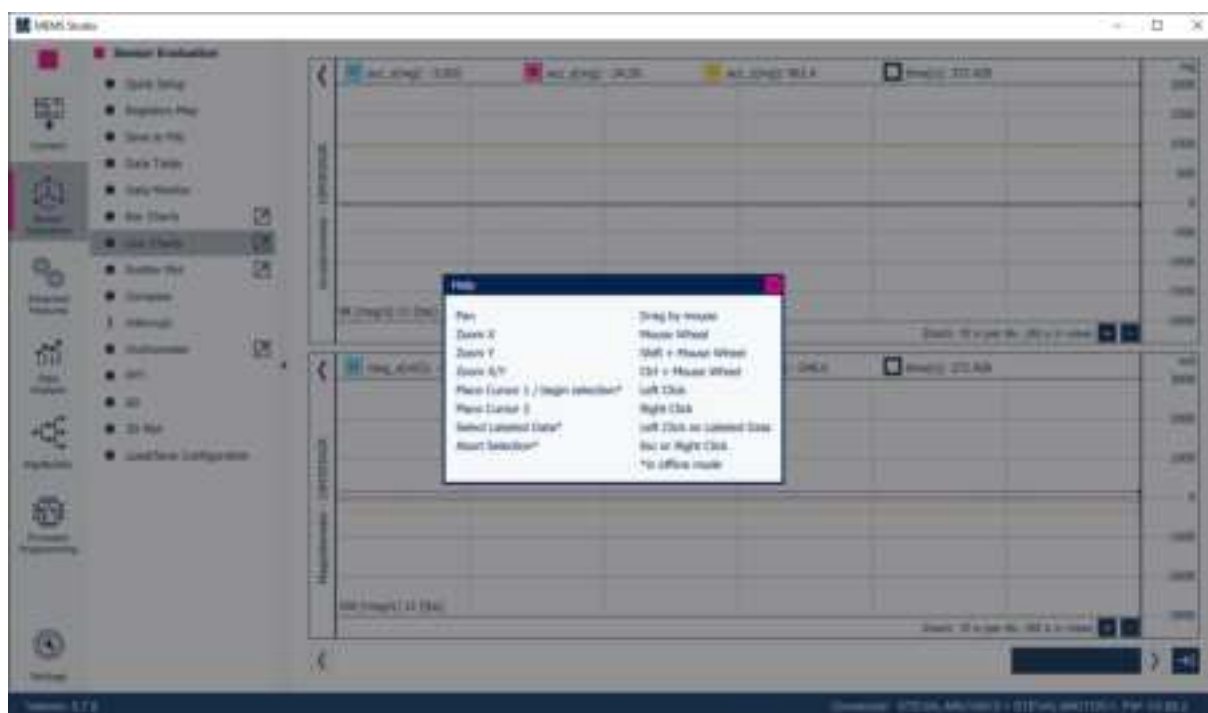
Figure 17. Line Charts tool



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
- **[Help]**: displays the available keyboard shortcuts for navigation in the chart

2.00
Fit
Auto
Full
Save
Help
-2.00



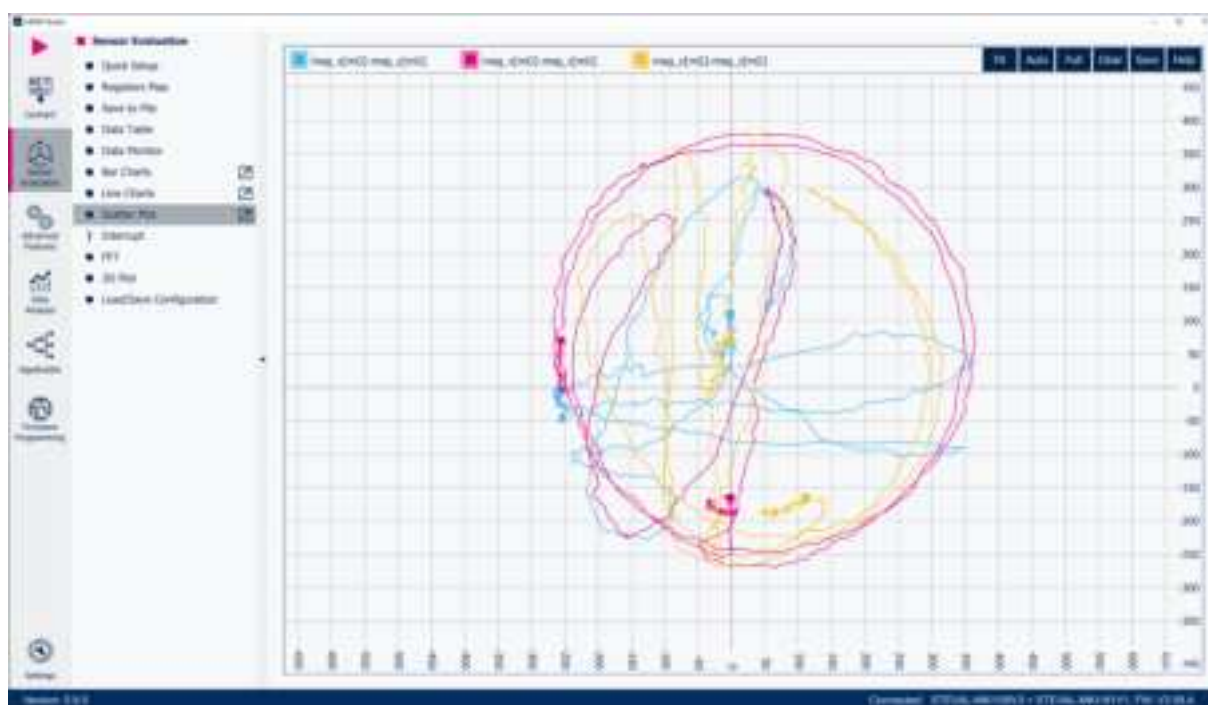
Scatter Plot tool

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 19. Scatter Plot tool



Compass tool

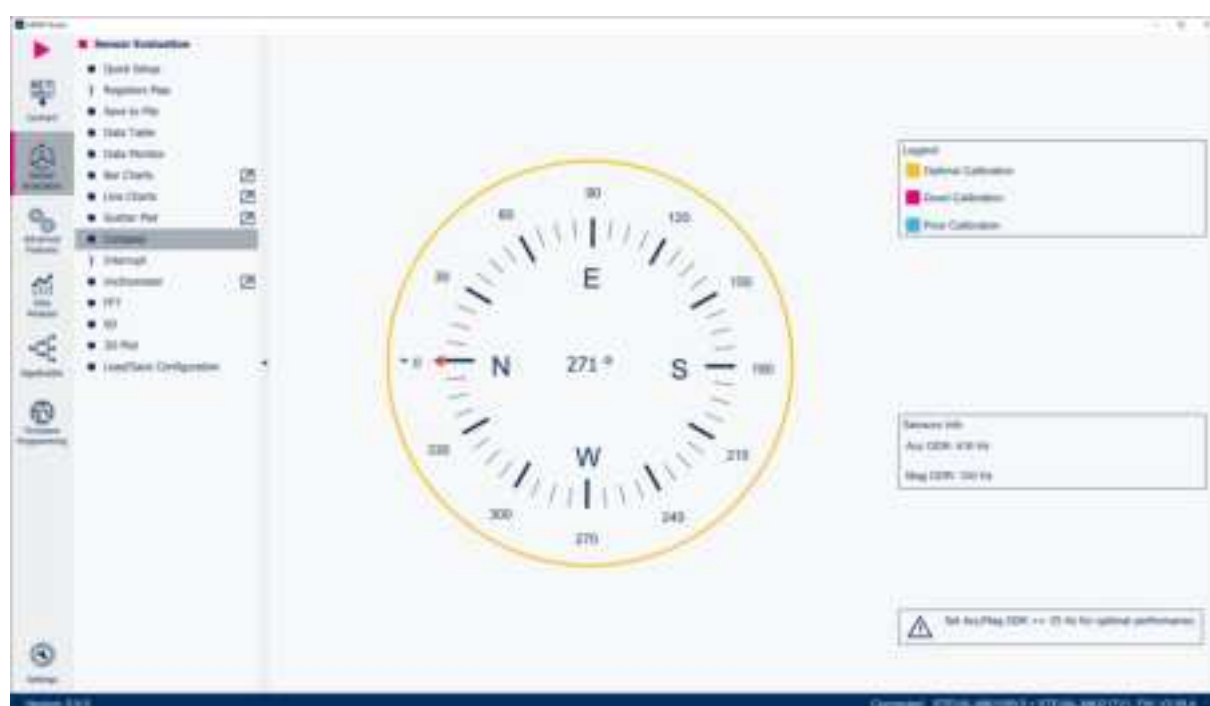
The [**Compass**] tool shows an example of a compass application, which can be implemented using a 6-axis module (3-axis accelerometer and 3-axis magnetometer).

The algorithm uses the magnetometer data to measure the Earth's magnetic field, and the accelerometer data to compensate 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 20. 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.

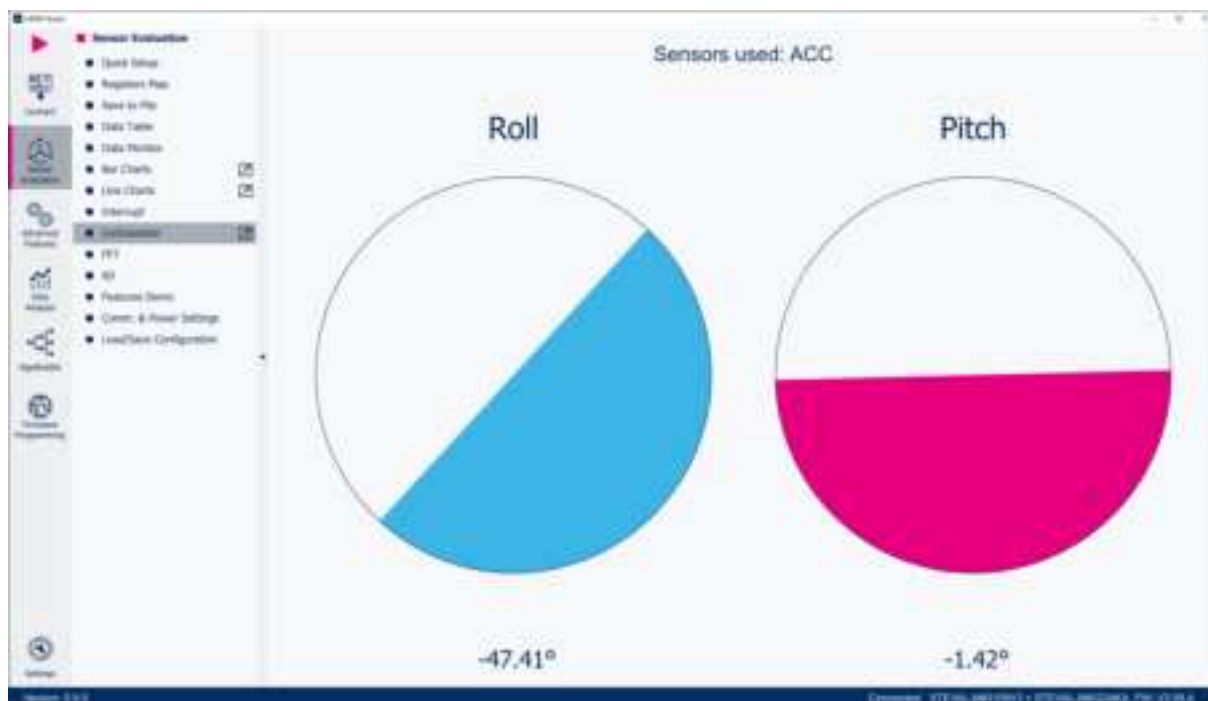
The user can configure the interrupts using the dedicated combo box in order to load the recommended configurations in the device.

Figure 21. Interrupt 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 22. 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 23. FFT tool



6D tool

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

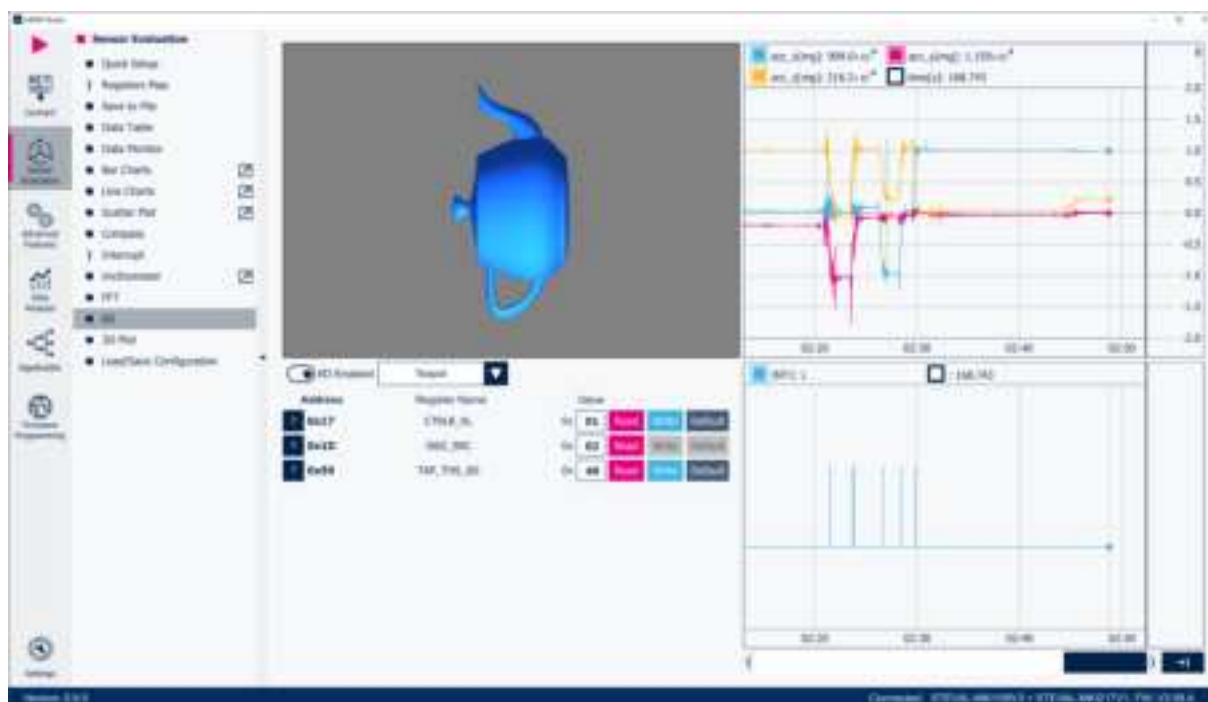
On this page, it is possible to configure the interrupt for 6D recognition by clicking on the [6D Enabled] toggle button.

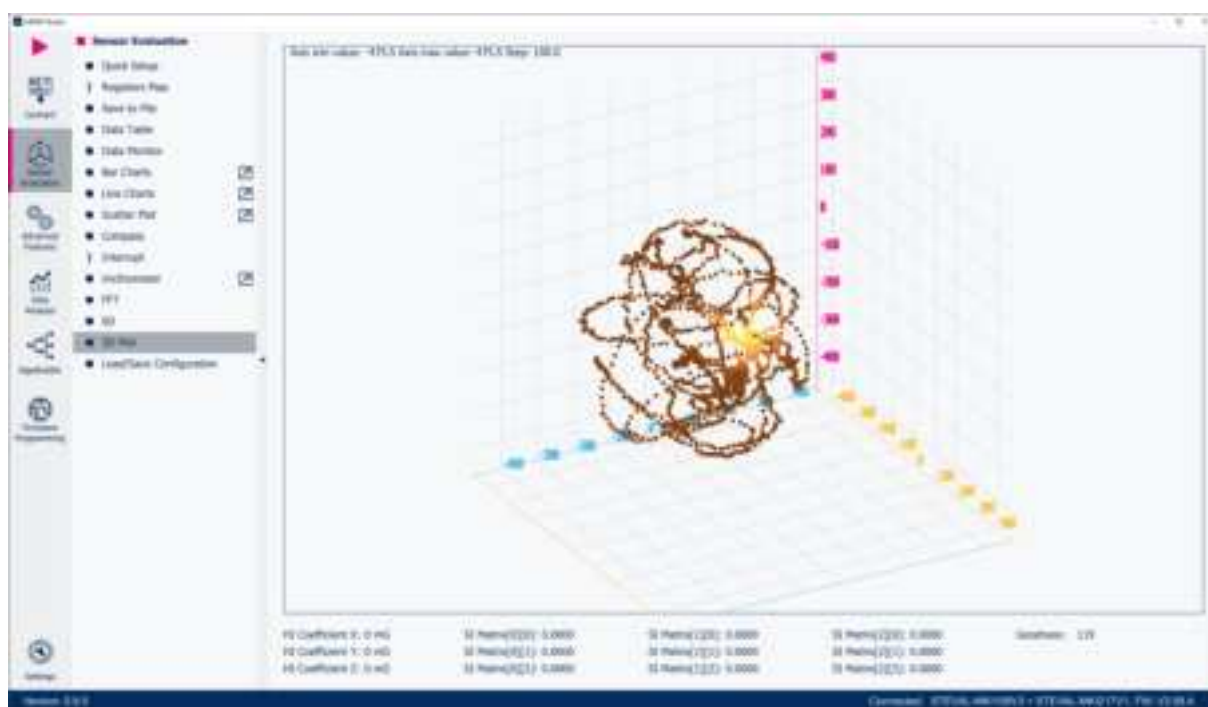
After loading the configuration, the selected 3D model is oriented depending on the 6D position detected.

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 24. 6D tool

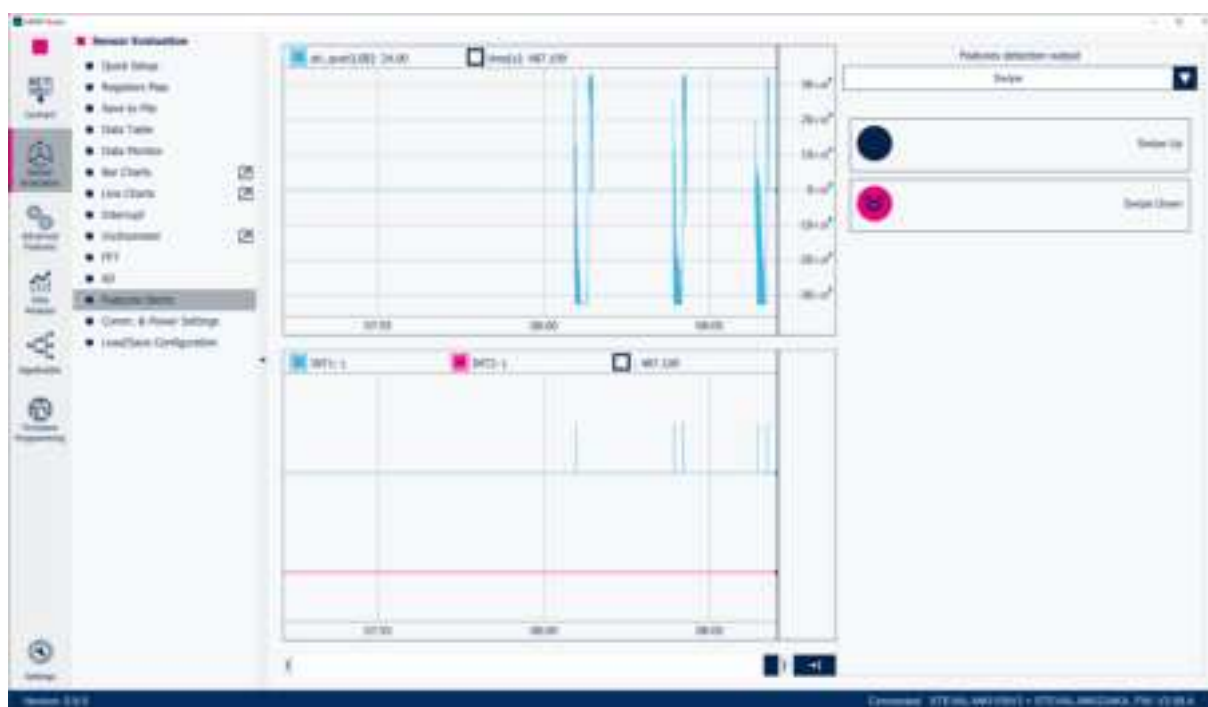




Features Demo page

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 26. Features Demo page



The **[Comm. & Power Settings]** tool allows the user to read the current consumption and update VDD and VDDIO values on the Professional MEMS tool board (STEVAL-MKI109V3).

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 28. FIFO tool



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 29. Load/Save Configuration page



4

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 30. Connect page for library evaluation



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

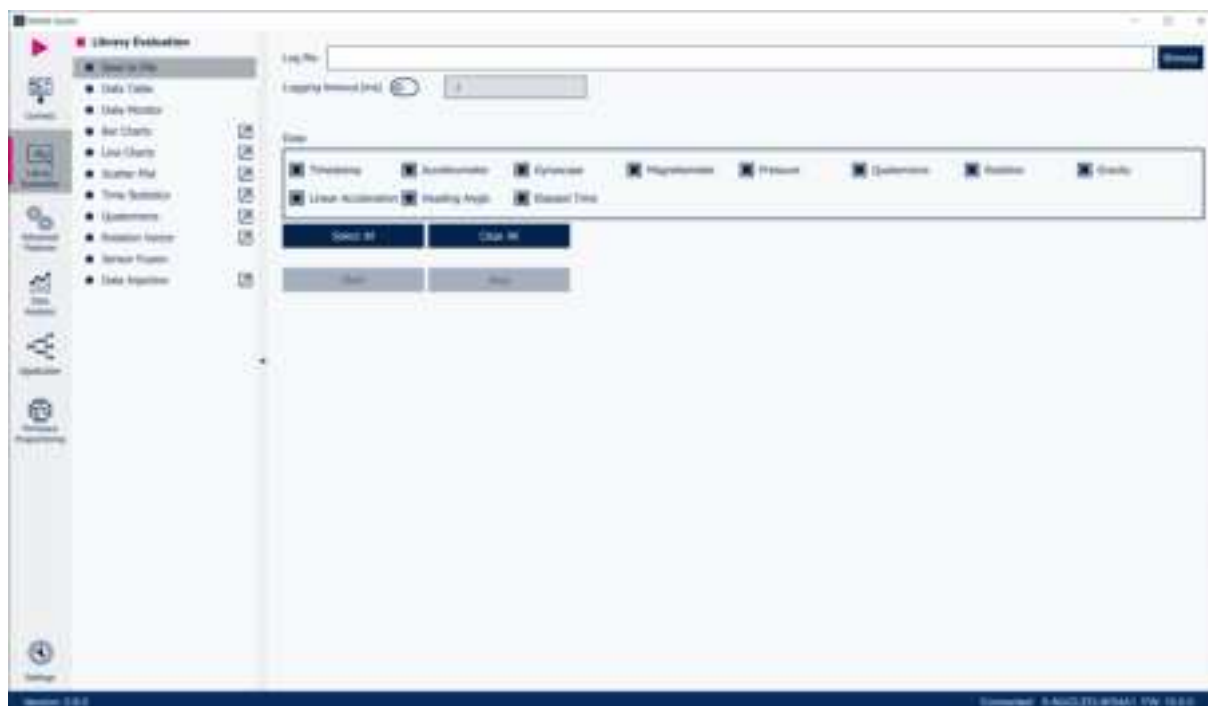
The library evaluation offers:

- 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 31. Save To File page



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.

Item Name	Size (MB)	Value (USD)
00000000	44	975
00000000	50	960
00000000	110	940
00000000	90	910
00000000	107	890
00000000	106	860
00000000	106	840
00000000	106	820
00000000	106	800
00000000	106	780
00000000	106	760
00000000	106	740
00000000	106	720
00000000	106	700
00000000	106	680
00000000	106	660
00000000	106	640
00000000	106	620
00000000	106	600
00000000	106	580
00000000	106	560
00000000	106	540
00000000	106	520
00000000	106	500
00000000	106	480
00000000	106	460
00000000	106	440
00000000	106	420
00000000	106	400
00000000	106	380
00000000	106	360
00000000	106	340
00000000	106	320
00000000	106	300
00000000	106	280
00000000	106	260
00000000	106	240
00000000	106	220
00000000	106	200
00000000	106	180
00000000	106	160
00000000	106	140
00000000	106	120
00000000	106	100
00000000	106	80
00000000	106	60
00000000	106	40
00000000	106	20
00000000	106	0

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.

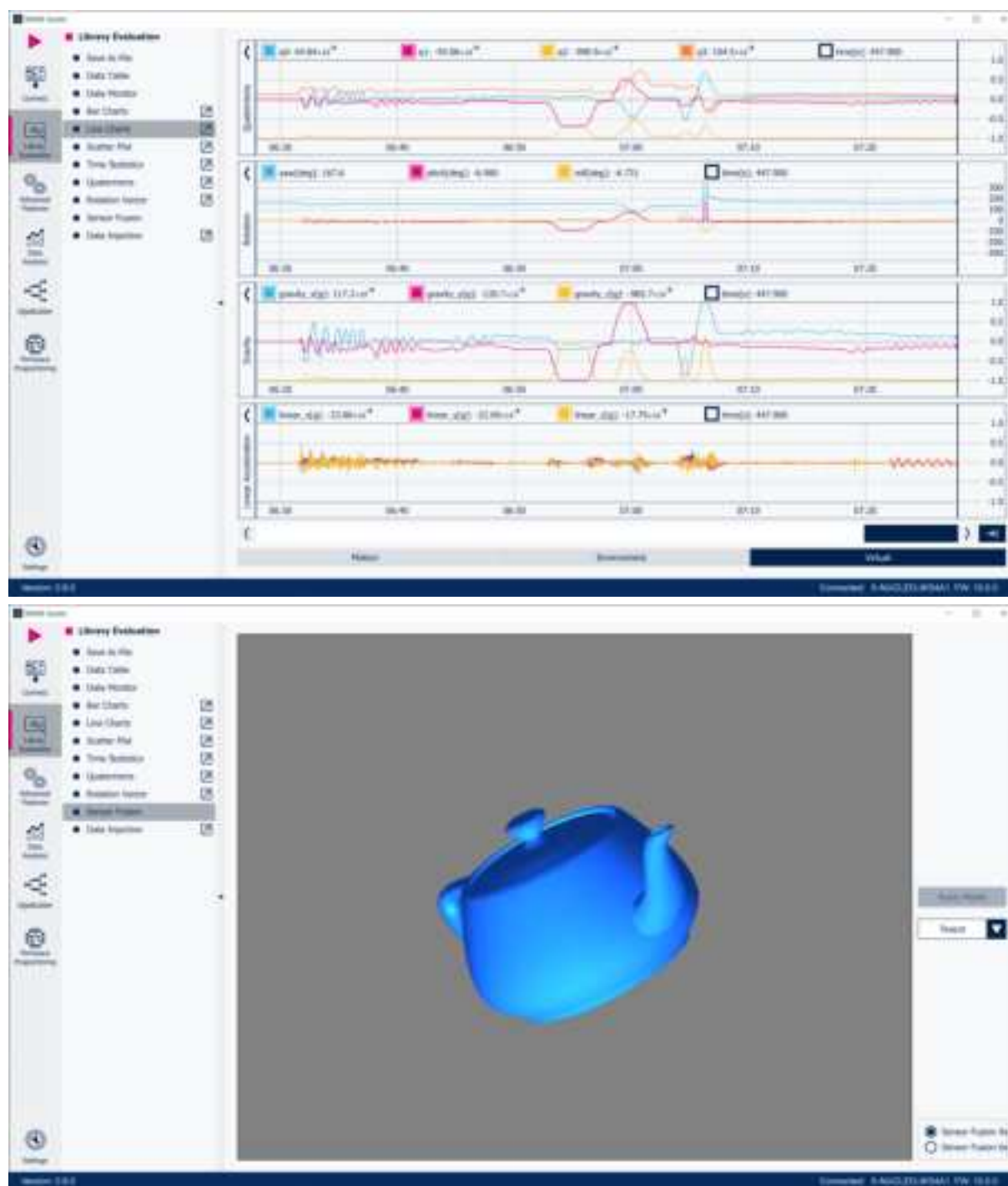
Figure 33. Data Monitor



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.

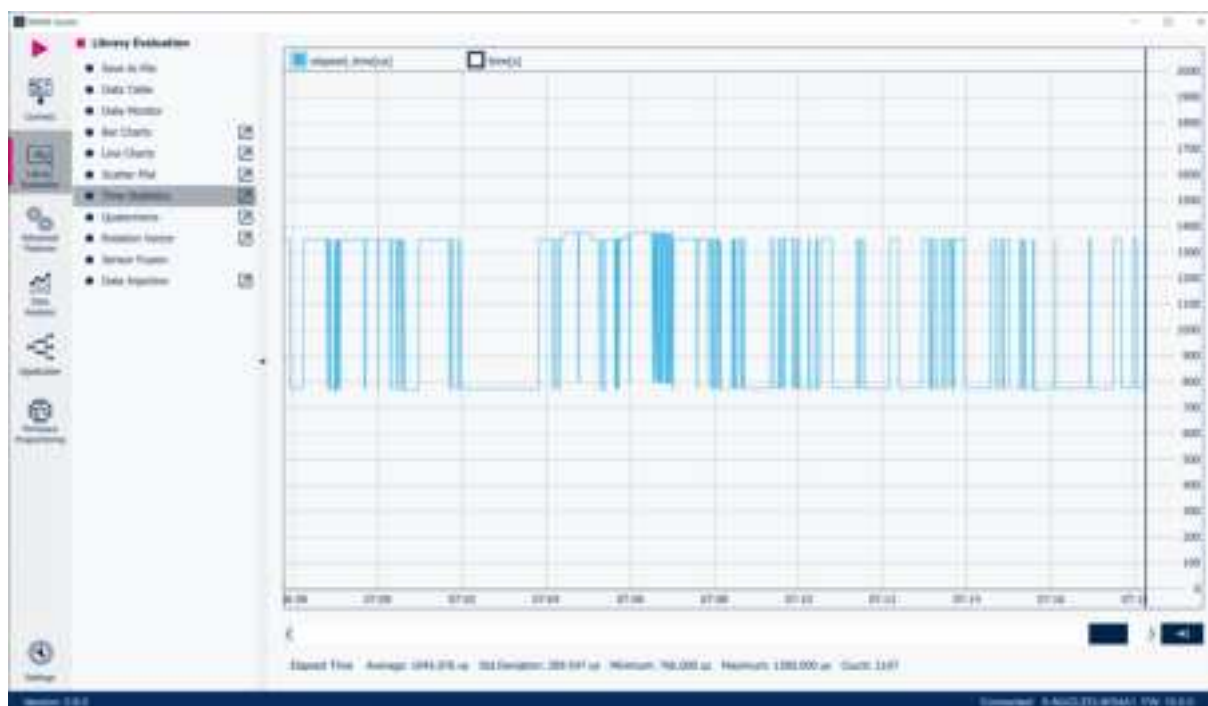
Figure 34. Example of data visualization pages



Time Statistics

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

Figure 35. Time Statistics page



5 Advanced features

5.1 Finite state machine (FSM)

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 37. FSM page



FSM parameters

Figure 38. Configuration of FSM parameters - menu bar

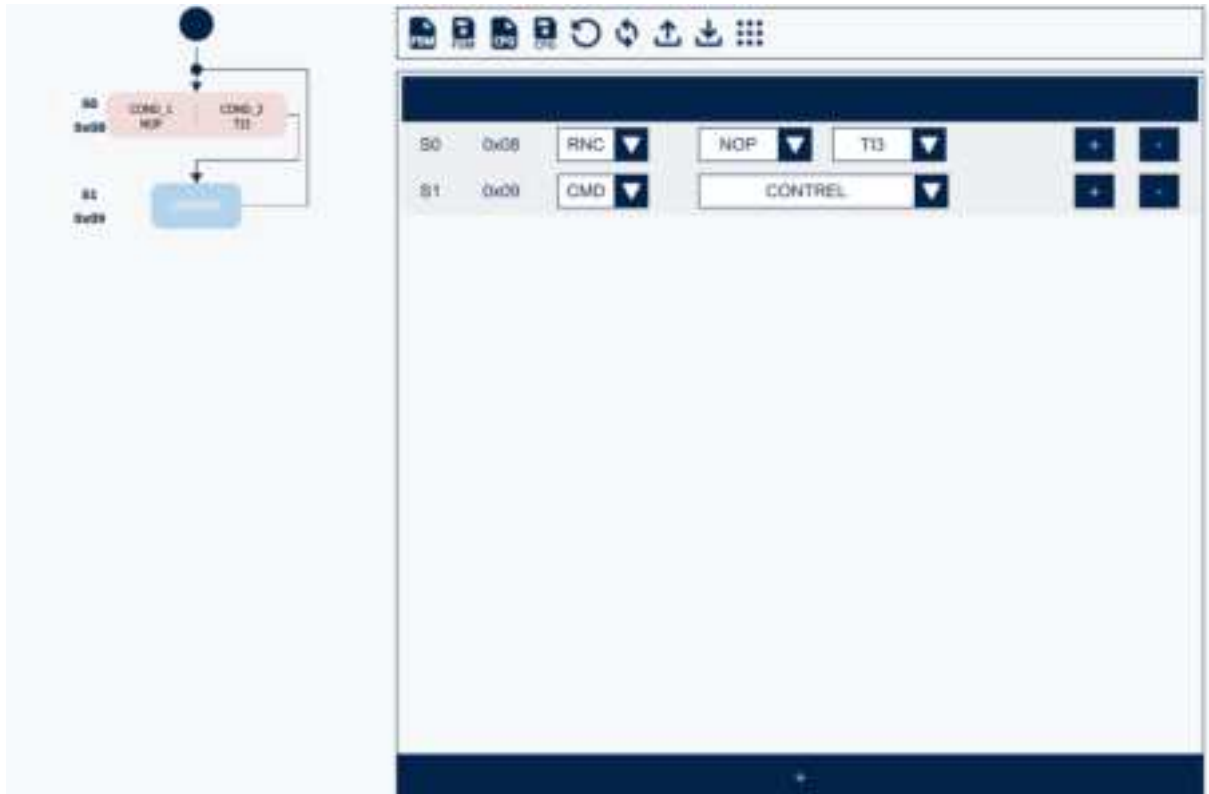


- **[Sensor selector]:** displays the selected device to configure
- **[State machine selector]:** displays the selected state machine to configure. This selection is applied to both the configuration page and debug page if available.
- **[FSM ODR]:** this setting determines the processing / output data 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

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 39. 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.

Status Data

Enabled	<input checked="" type="checkbox"/>
INT1	<input checked="" type="checkbox"/>
INT2	<input type="checkbox"/>

Fixed Data

Decimation	<input type="checkbox"/>
------------	--------------------------

Variable Data

0x06	TC	00
0x07	Tim... (0.533)	10

Figure 40. FSM configuration toolbar



- **Open state machine from file (.fsm or .json file):** populates the GUI with .fsm (legacy) or .json file content data
- **Save state machine as (.json):** exports the selected state machine configuration to a dedicated .json file
- **Open device configuration from file (.ucf or .json 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 (.json file):** 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
- **Reset all state machines:** clears all state machine GUI data
- **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
- **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 41. Testing tab



Debug / Data injection

The **Debug / Data injection** 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

- **[Next]**: injects the next sample and updates the table with output data
- **[All Next]**: injects all the next samples one by one, updating the table content at every sample injected
- **[Run]**: injects all the next samples, updating the table content at the end of the data injection process
- **[N Next]**: injects a defined number of samples, updating the table at every sample injected
- **[N Run]**: injects a defined number of samples, updating the table content once the given number of samples have been injected
- **[Pause]**: pauses the data injection process
- **[Stop]**: interrupts the data injection process and exits debug mode.

The content of the table with these results can be exported to a .csv file using the button **[Export Result]**.

Figure 42. Debug / Data injection 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. 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
- **[ARFF generation]**: allows configuring inputs, filters, and features to generate an .arff file
- **[Data analysis]**: allows processing acquired data and recommends window length, filters, and features that can be used for decision tree generation
- **[Decision tree generation]**: allows configuring the decision tree outputs and generating the decision trees
- **[UCF generation]**: allows setting the metaclassifier and generating the configuration as a .ucf file

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_configuration.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.

All artifacts generated during the MLC configuration process can be deleted using the **[Clear Workspace]** button, and the MLC_Configuration file.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.

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. The unit of measurement for the data is 'mg' for the accelerometer and 'dps' for the gyroscope.

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 [-] button in the tables.

Figure 43. Data patterns 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.

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 code 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.

The ARFF file can be generated by an external tool and loaded into MEMS Studio.

Figure 44. ARFF generation tab



Data analysis

Data analysis tools offer 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 45. Data analysis



UCF generation

The [UCF generation] tab allows configuring the metaclassifiers and generating the .ucf file containing the machine learning core configuration.

UCF (Unico Configuration File) is a text file in which we store sensor configuration. It is a sequence of pairs of register addresses and values.

The generated MLC configuration can be stored also in the JSON or .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 configuration to sensor] button.

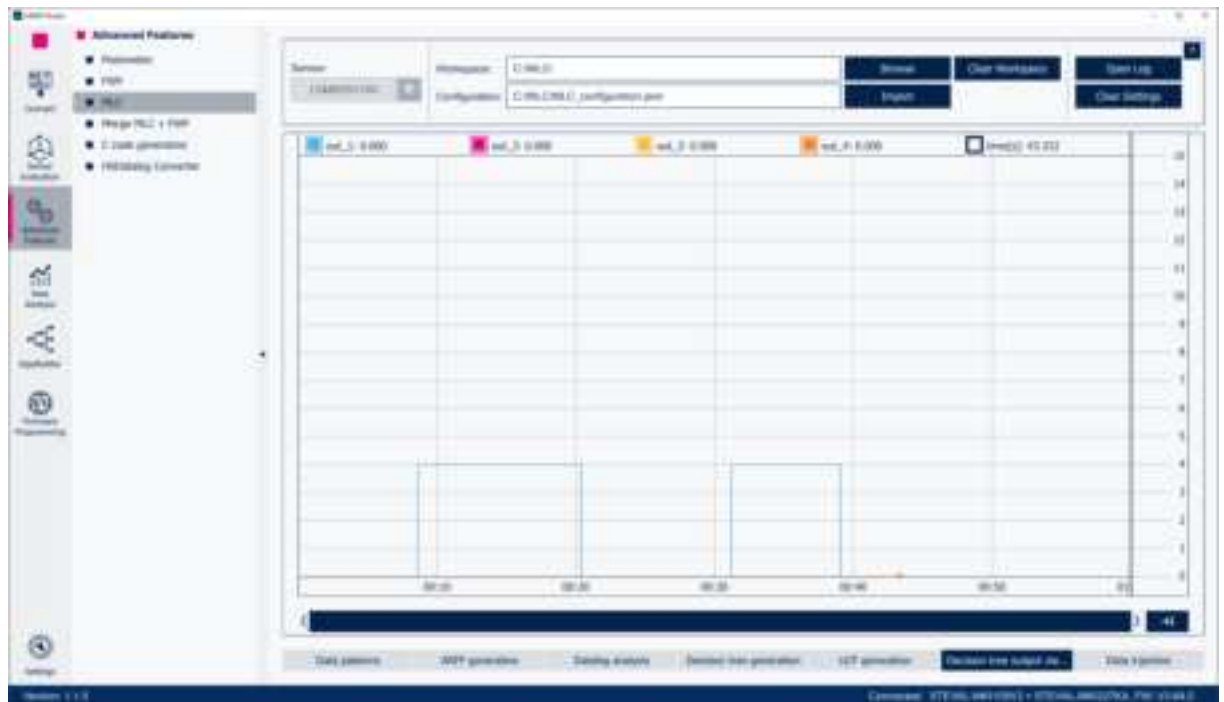
Figure 47. UCF generation tab



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 48. Decision tree output viewer tab



Data injection

The **Data injection** 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

- **[Next]**: injects the next sample and updates the table with output data
- **[All Next]**: injects all the next samples one by one, updating the table content at every sample injected
- **[Run]**: injects all the next samples, updating the table content at the end of the data injection process
- **[N Next]**: injects a defined number of samples, updating the table at every sample injected
- **[N Run]**: injects a defined number of samples, updating the table content once the given number of samples has been injected
- **[Pause]**: pauses the data injection process
- **[Stop]**: interrupts the data injection process and exits debug mode.

The content of the table with these results can be exported to a .csv file using the button **[Export Result]**.

Figure 49. Data injection



5.3 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

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 50. 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:

- **[Next]**: injects the next sample and updates the table with the output data
- **[All Next]**: injects all next samples one by one, updating the content of the table with every sample
- **[Run]**: injects all samples, updating the content of the table at the end of the data injection process
- **[N Next]**: injects a defined number of samples, updating the table with every sample injected
- **[N Run]**: injects a defined number of samples, updating the content of the table once a given number of samples is injected
- **[Pause]**: pauses the data injection process with the possibility to resume it
- **[Stop]**: interrupts the data injection process and exits debug mode

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

During a debug session, the current status (sample loaded and output number of steps) is displayed in the central part of the GUI. Clicking on a dedicated button, the user can export the results of the debug session in .csv format.

Figure 51. 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 or space-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).

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 52. Regression Tool tab



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, and assigning labels to specific parts of the data.

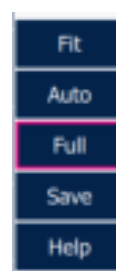
Previously acquired data logs can be opened by clicking on the **[Browse]** button.

Figure 53. Time domain 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.

Figure 54. 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
- **[Help]**: displays help for working with the chart

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 below. Cursors can be hidden by closing the cursor table in the chart area.

Figure 55. Cursor value table

Cursors					
	276.583		277.905		1.322
	281.7×10^3		279.1×10^3		2.523×10^3
					756.3×10^{-3}
					-1.908×10^{-3}

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
- **[Unlabel]**: 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:

- **[Set Default Label]**: sets a label to all unlabeled parts
- **[Clear Labels]**: remove all labels
- **[Save Labeled Data]**: saves each data for each label to a separate file
- **[Save All]**: saves all changes including assigned labels to one file

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.

Figure 56. 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]: calculates the frequency spectrum from the entire signal
- [Selected Data FFT]: 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:

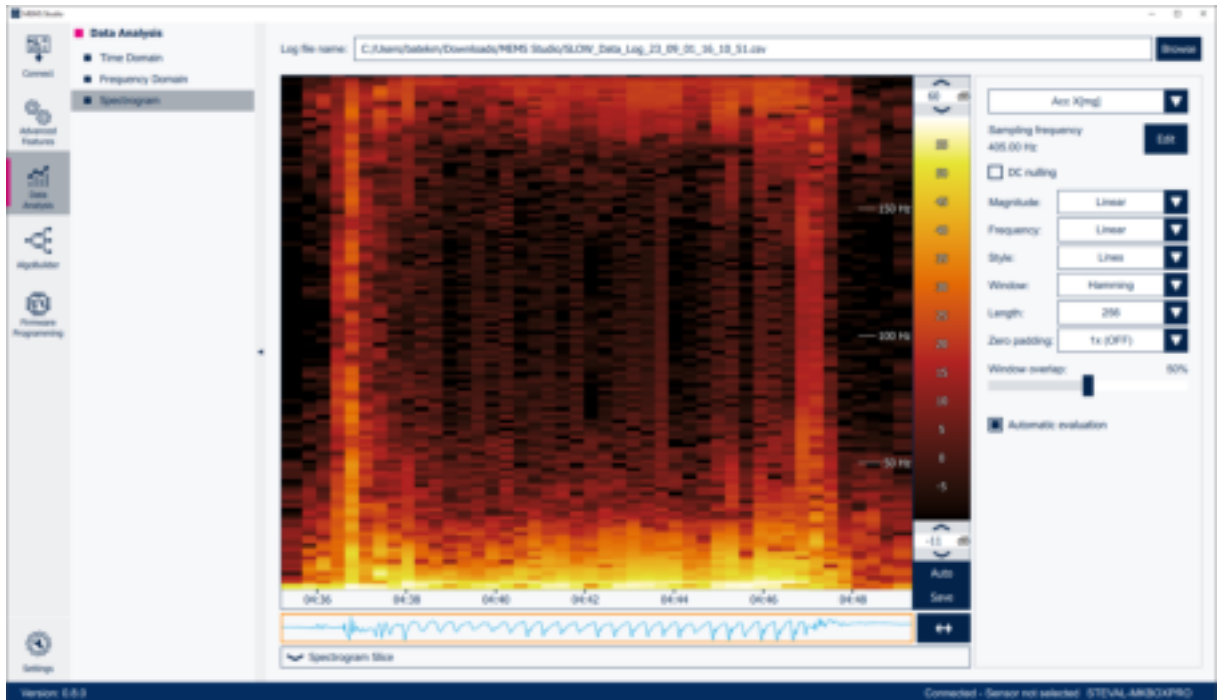
- [DC nulling]: removes the DC component from the signal
- [Window]: window function multiplied by the signal before frequency spectrum calculation in order to reduce the amplitude of the discontinuities at the boundaries of the signal part
- [Length]: number of samples used for the frequency spectrum 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 57. 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.

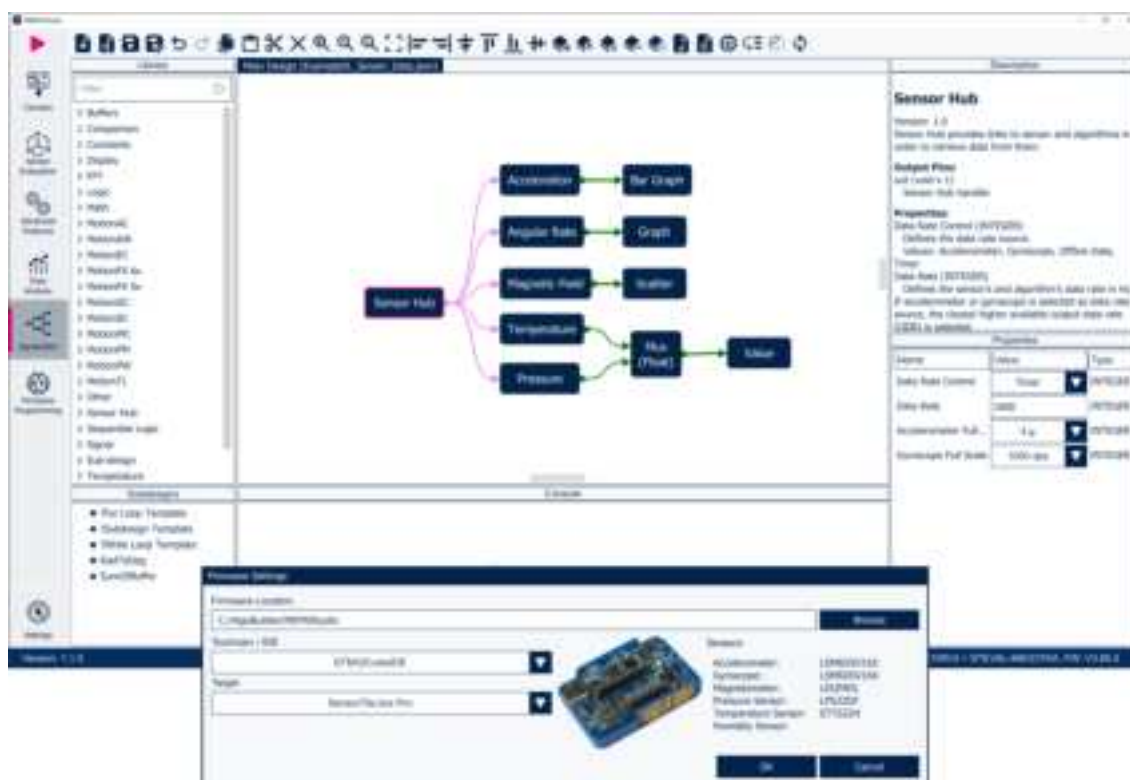
This tool quickly creates prototypes of applications for STM32 microcontrollers and MEMS sensors, including already existing algorithms (in the example of sensor fusion), user-defined data processing blocks, and additional functionalities.

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 58. 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.

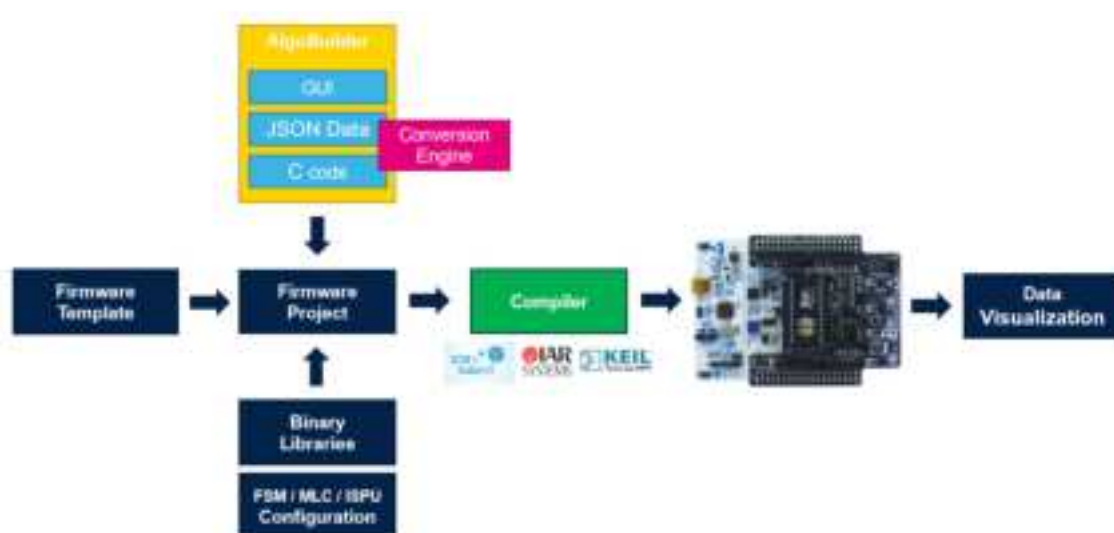
The project can be compiled using an external compiler tool and the most common compilers are supported (STM32CubeIDE, Arm® Keil® µVision®, IAR Embedded Workbench).

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 59. AlgoBuilder principle of operation



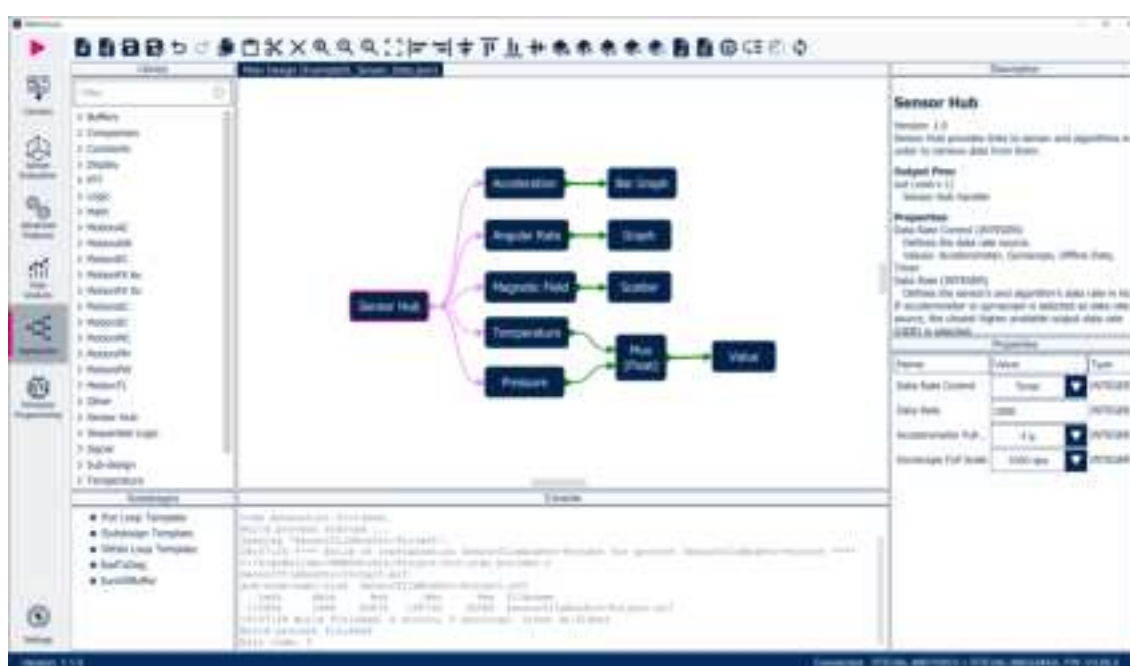
7.2 Overview

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 60. 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\ST_MEMS_Studio\algebuidler\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\ST_MEMS_Studio\algebuidler\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 5. 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

7.9 Libraries

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

Table 6. 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 7. 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:

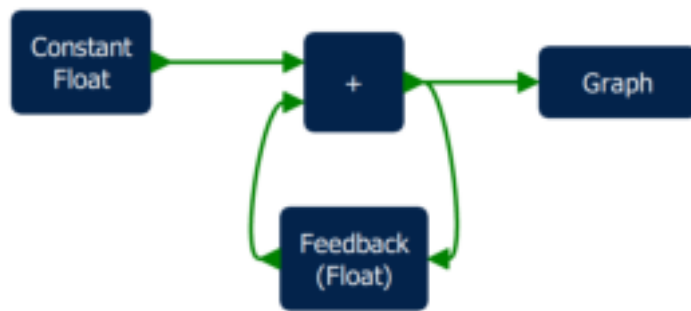
- **FLOAT** represents real numbers and is used for floating-point arithmetic (for example, in the acceleration function block output). In C code, the representation float variable is used. The size is 4 bytes.
- **INT** represents integer numbers (for example, in the counter function block). In C code, 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 61. Using the Feedback function block



7.11 Data visualization

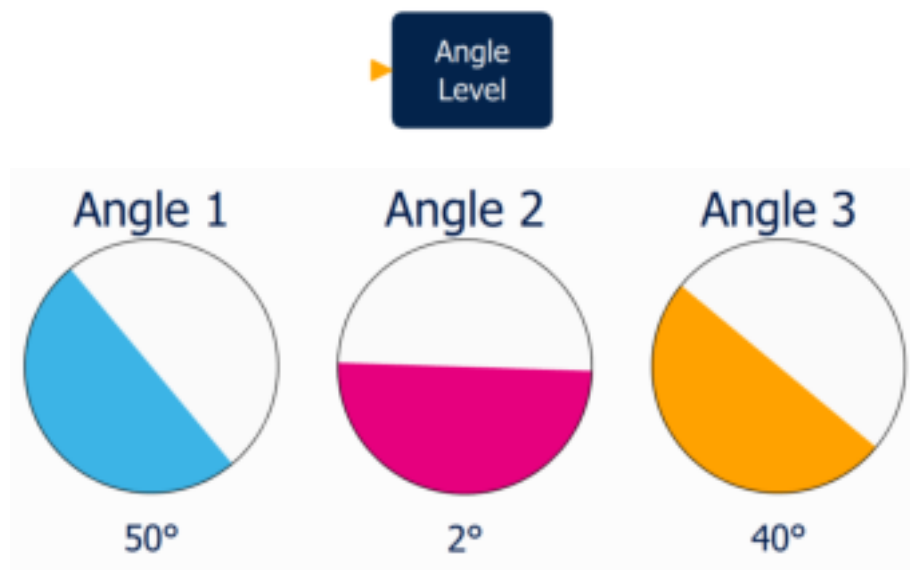
The application offers nine types of data visualization in the **[Display]** library:

- **[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.

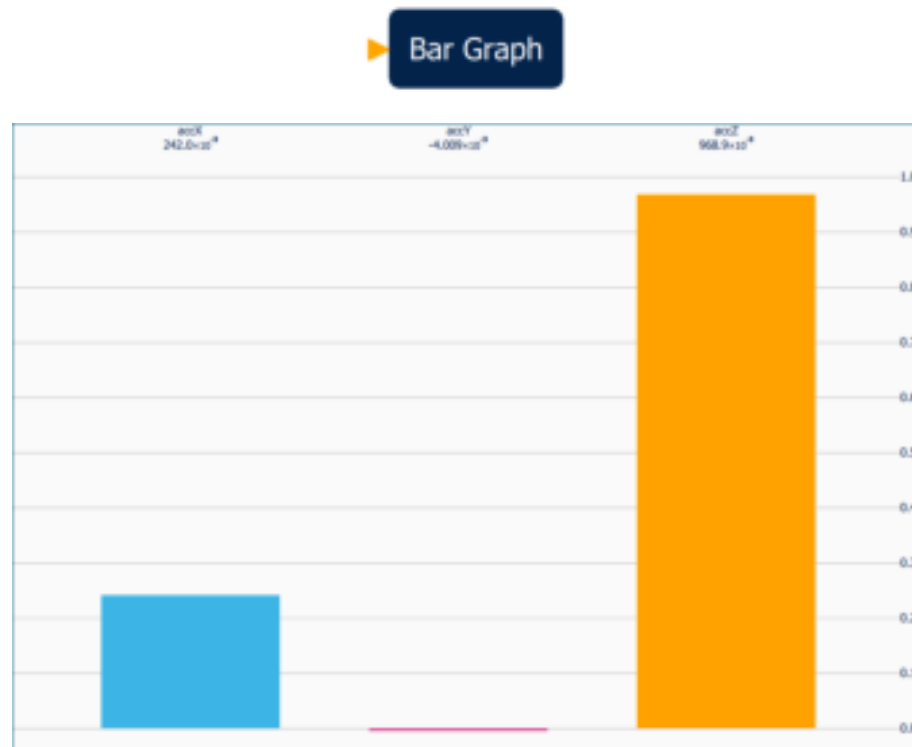
Use the **[Angle Level]** function block in the design to display data as an angle level indicator. This graph works with any floating or integer value and each of them can have up to three indicators. In the **[Properties]** field, you can set the name of the graph as well as the name of each angle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

Figure 62. Angle Level function block and example of data visualization



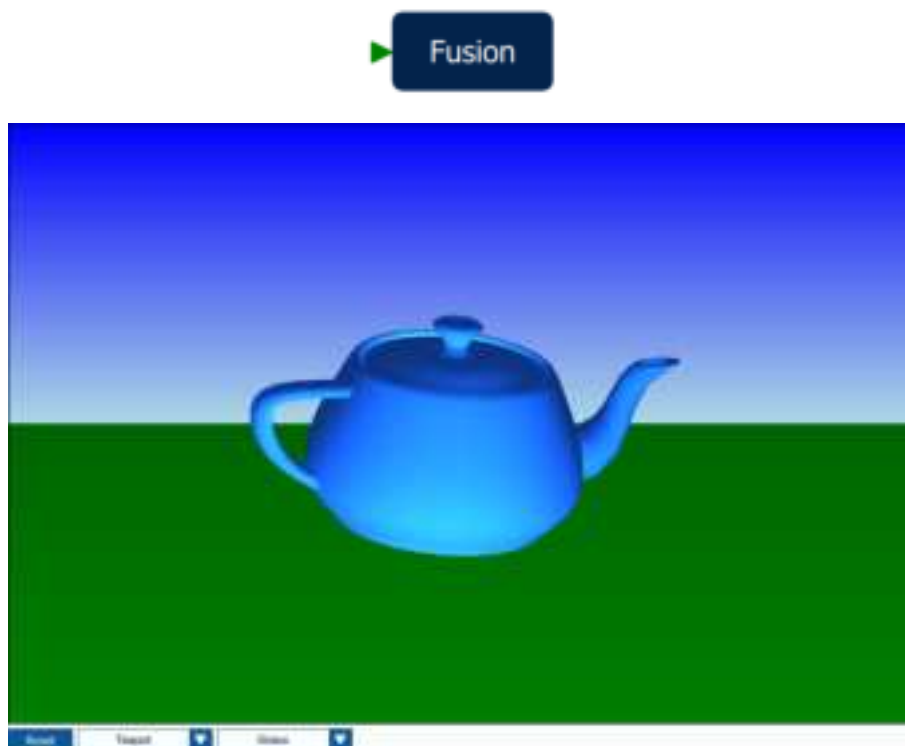
Use the **[Bar Graph]** function block in the design to display data as a bar graph. A bar graph is suitable for a quick check of an actual value without needing to see the history. This graph works with any floating or integer value and each of them can have up to six bars. In the **[Properties]** field, 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 63. Bar Graph function block and example of data visualization



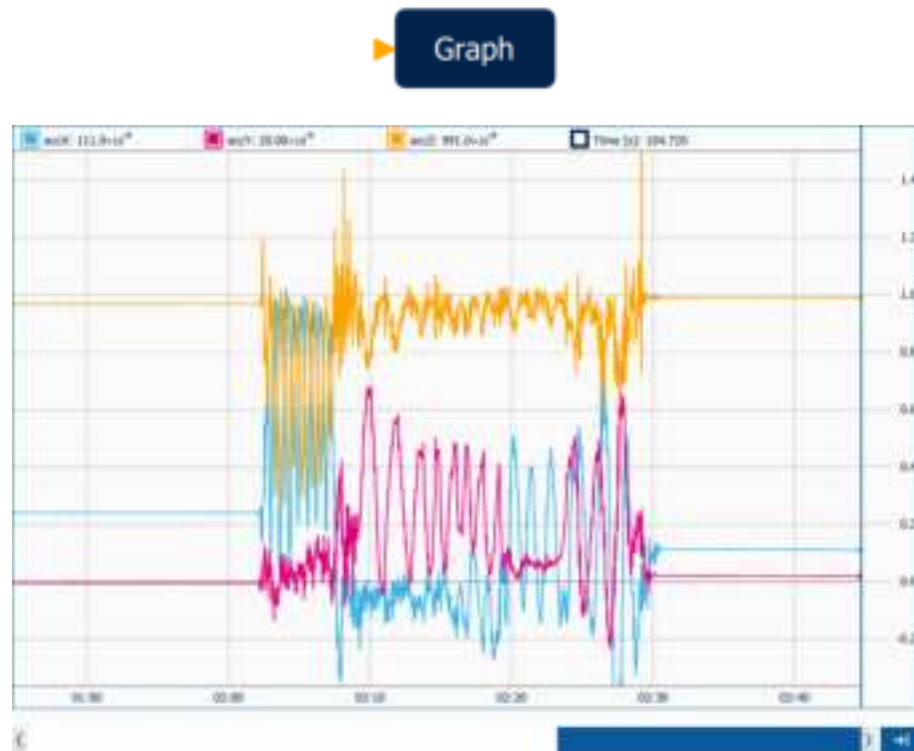
Use the **[Fusion]** function block in the design to display quaternion data on a 3D model. The 3D model (teapot, Nucleo board, car, head) is usually used to check device orientation in 3D space. This graph requires quaternions, which can be obtained, for example, from the sensor fusion (MotionFX) algorithm.

Figure 64. 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 65. Graph function block and example of data visualization



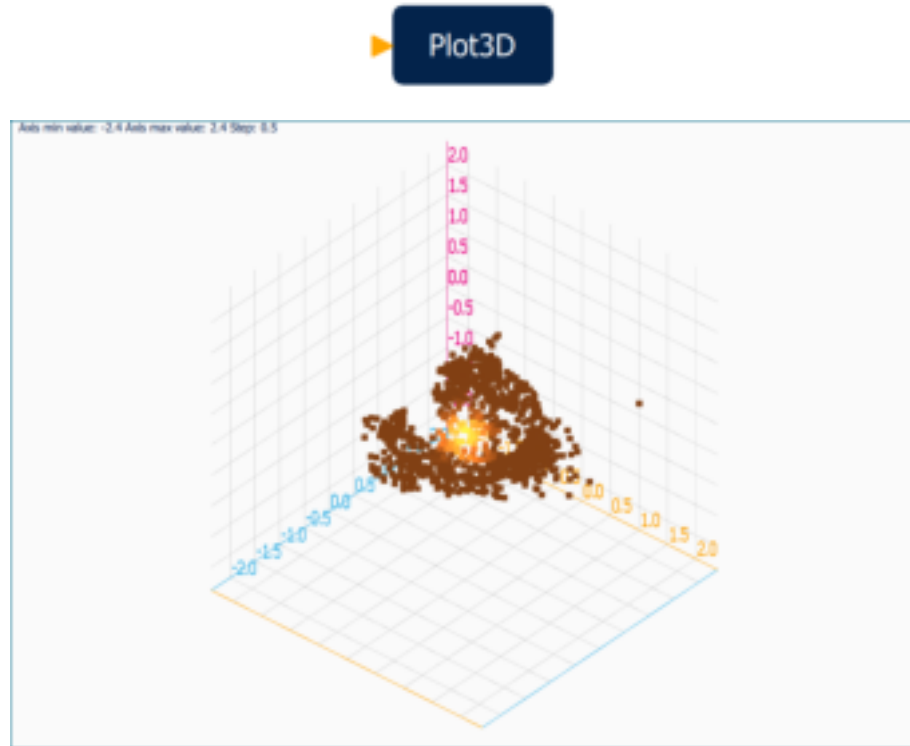
Use the **[Logic Analyzer]** function block in the design to display logic signals, which can have values of only 0 or 1. The logic analyzer can have up to 16 channels. In **[Properties]**, you can change the name of each channel and the logic analyzer.

Figure 66. 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 67. 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 68. 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.

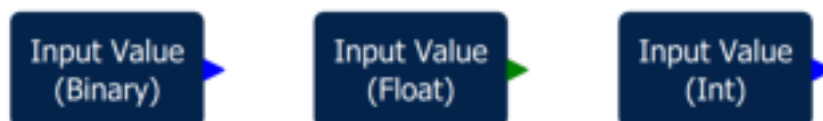
Figure 69. Value function block and example of data visualization



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 70. 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 71. FFT and FFT Plot connections



Figure 72. 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 73. Example of a design with FSM/MLC



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

It is mandatory to specify the number of used FSM and MLC outputs in the .ucf file. The size of the FSM/MLC output vector is adjusted accordingly.

Figure 74. FSM/MLC function block properties

Properties		
Name	Value	Type
Number of FSM	0	INTEGER
Number of MLC	1	INTEGER
Configuration File	ism330dhcx_six_d_position.ucf	STRING

Warning:

The UCF 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.

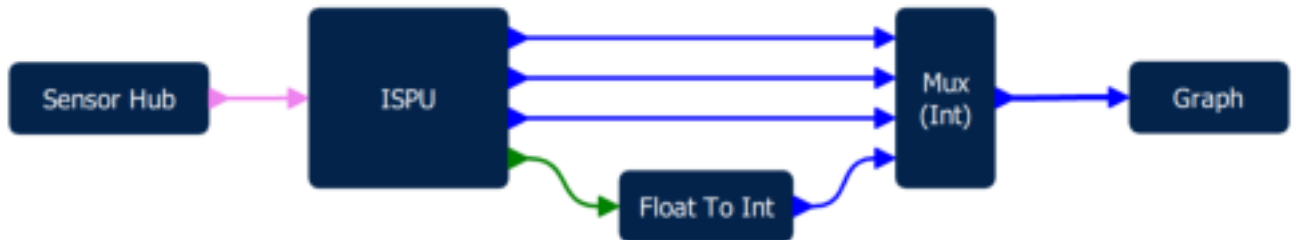
The interrupt pin INT1 is usually configured by the MLC tool in which case it is not possible to use the accelerometer or gyroscope as **[Data Rate Control]** in the sensor hub.

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 75. Example of a design with ISPU



The configuration of the ISPU is stored in the .ucf file. The path to the .ucf file is the [Property] of the [ISPU] function block. The .ucf file can be created using the ISPU-Toolchain software. An output description file needs to be present in the same directory as the .ucf file with the ISPU configuration and must have the same name (only the extension is .json). This output description file is a .json file, which contains the description of the ISPU output values. The outputs of the [ISPU] function block are automatically adjusted according to this .json structure.

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:

```
{
  "output":
  [
    {
      "name": "Acc x [LSB]",
      "type": "int16_t"
    },
    {
      "name": "Acc y [LSB]",
      "type": "int16_t"
    },
    {
      "name": "Acc z [LSB]",
      "type": "int16_t"
    },
    {
      "name": "Acc norm [LSB]",
      "type": "float"
    }
  ]
}
```

Figure 76. ISPU function block Properties

Properties		
Name	Value	Type
Configuration File	ism330is_norm.ucf	STRING

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

Warnings:

The UCF 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 77. 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 78. Sensor Hub, Acceleration [g], Graph function blocks



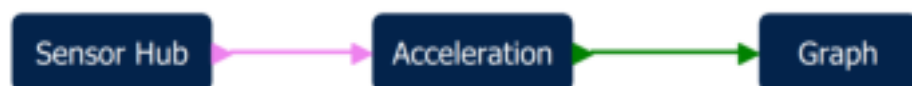
- Step 6.** Adjust the **[Graph]** properties. The **[Number of Curves]** in **[Graph]** defines the size of its input. The value needs to be changed to 3 to match the **[Acceleration]** [g] output size. The graph, waveforms, and unit names can also be changed. You can also quickly configure the **[Graph]**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

Figure 79. 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 80. Connected Sensor Hub, Acceleration, Graph function blocks



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



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 81. Generated code in algo_builder.c file

```

algo_builder.c

/* Generated by MEMS Studio version 0.8.0 */

#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][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 82. Compiler output



```

Console
C:/AlgoBuilder/Firmware_Project/Project/Src/usbd_storage_if.c
C:/AlgoBuilder/Firmware_Project/Project/Src/vcom.c
Application/STM32CubeMX/startup_stm32h745xx.s
C:/AlgoBuilder/Firmware_Project/Project/CubeMX/STM32USB-AI-SensorTileBoxPro/syscalls.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_buffers.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_display.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_fft.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_sensor_hub.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_sequential_logic.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_signal.c
C:/AlgoBuilder/Firmware_Project/Project/Src/ah_user_input.c
SensorTileBoxPro-Project.elf
arm-none-eabi-size SensorTileBoxPro-Project.elf
  text    data    bss     dec     hex    filename
  27812    3024    6452    36288    8a478    SensorTileBoxPro-Project.elf
11:03:13 Build Finished, 0 errors, 0 warnings, (took 1m:17s,301ms)
  
```

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

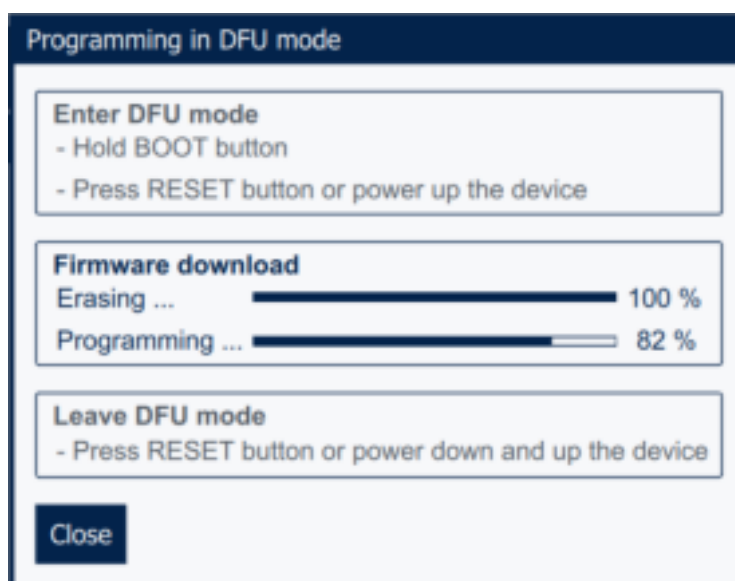
The STM32CubeProgrammer CLI application is used to program the STM32 microcontroller with an STLINK or DFU interface. The path to the STM32CubeProgrammer CLI application must be set in the MEMS Studio settings in the AlgoBuilder **[Settings]** page. Copying to the flash drive is available for STM32 Nucleo 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.

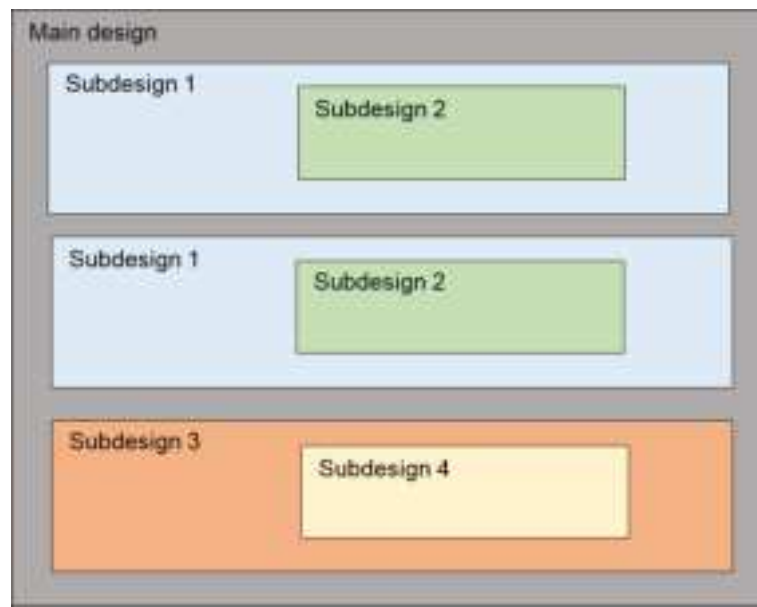
Figure 83. Programming target with DFU interface



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 84. 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 85. 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.



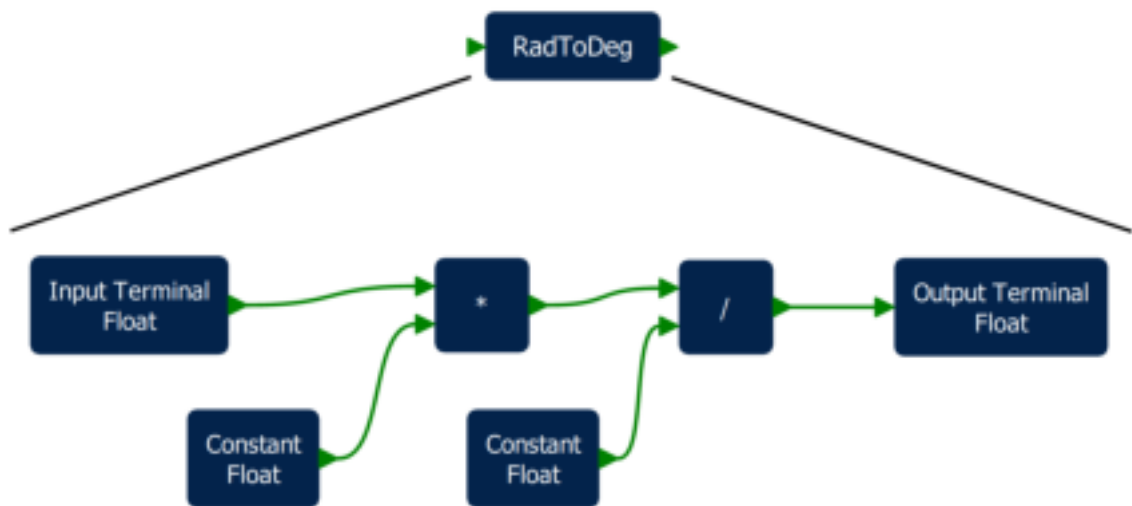
The subdesign can be saved in the same way as the main design by clicking on the icon  (Save Design) or  (Save Design As). The subdesign can then be used in the main design or other subdesign. To use the subdesign, drag and drop the particular item from the **[Subdesigns]** dock to the workspace. To 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 86. 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 87. 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 84. 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:

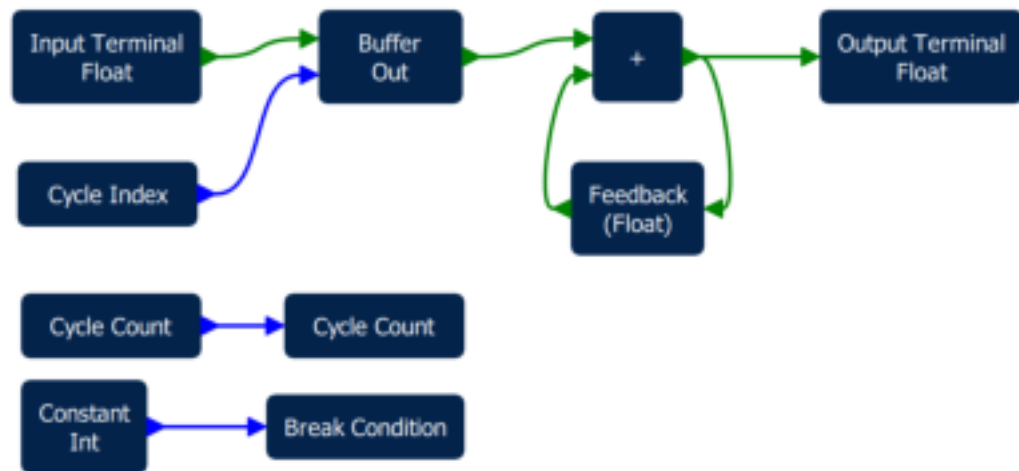
[for] loop:

- **[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 88. Example of loop (calculates the sum of all items in the input buffer)



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 **[AlgoBuilder Settings]** 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.

Figure 89. 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-MKI109V3).

Revision history

Table 8. 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

Contents

1	Overview	2
2	Getting started	3
2.1	Installation	3
2.2	Application settings	5
2.3	AlgoBuilder settings	6
2.4	Updater settings	7
2.5	Running the application for the first time	8
2.6	Hardware and firmware compatibility	10
3	Sensor evaluation	17
4	Library evaluation	37
5	Advanced features	44
5.1	Finite state machine (FSM)	44
5.2	Machine learning core (MLC) tool	48
5.3	Pedometer	55
6	Data analysis	58
6.1	Time domain	58
6.2	Frequency domain	60
6.3	Spectrogram	61
7	AlgoBuilder	62
7.1	Principle of operation	63
7.2	Overview	64
7.3	Workspace	64
7.4	Library dock	64
7.5	Subdesigns dock	64
7.6	Description dock	65
7.7	Properties dock	65
7.8	Toolbar	66
7.9	Libraries	68
7.10	Data types	69
7.11	Data visualization	70
7.12	Data input	77
7.13	Conditional execution	77
7.14	Fast Fourier transform (FFT)	78
7.15	Finite state machine (FSM) and machine learning core (MLC)	79

7.16	Intelligent sensor processing unit (ISPU)	80
7.17	Creating your first design	82
7.18	Programming the target	86
7.19	Subdesign	87
8	Firmware programming	90
	Revision history	91
	List of tables	94
	List of figures	95

List of tables

Table 1.	Supported hardware and firmware for full sensor evaluation	10
Table 2.	Supported hardware and firmware for reduced sensor evaluation	12
Table 3.	Supported hardware and firmware for library evaluations	14
Table 4.	Supported hardware and firmware for AlgoBuilder functionality	15
Table 5.	AlgoBuilder toolbar functions	66
Table 6.	AlgoBuilder libraries	68
Table 7.	Motion libraries supported in AlgoBuilder	68
Table 8.	Document revision history	91

List of figures

Figure 1.	MEMS Studio application	1
Figure 2.	MEMS Studio Installer	3
Figure 3.	MEMS Studio Installer for macOS	4
Figure 4.	Application Settings.	5
Figure 5.	AlgoBuilder settings	6
Figure 6.	Updater Settings.	7
Figure 7.	Sensor selection - ProfiMEMS firmware	8
Figure 8.	Sensor selection - DatalogExtended firmware.	9
Figure 9.	Sensor and board references	9
Figure 10.	Quick Setup page	17
Figure 11.	Registers Map page	18
Figure 12.	Save to File page	19
Figure 13.	Data log period source.	20
Figure 14.	Data Table tool	21
Figure 15.	Data Monitor tool	22
Figure 16.	Bar Charts tool	23
Figure 17.	Line Charts tool	24
Figure 18.	Line chart options and help	25
Figure 19.	Scatter Plot tool	26
Figure 20.	Compass tool	27
Figure 21.	Interrupt tool.	28
Figure 22.	Inclinometer tool	29
Figure 23.	FFT tool.	30
Figure 24.	6D tool	31
Figure 25.	3D Plot tool	32
Figure 26.	Features Demo page.	33
Figure 27.	Comm. & Power Settings tool	34
Figure 28.	FIFO tool	35
Figure 29.	Load/Save Configuration page	36
Figure 30.	Connect page for library evaluation	37
Figure 31.	Save To File page	38
Figure 32.	Data Table	39
Figure 33.	Data Monitor	40
Figure 34.	Example of data visualization pages	41
Figure 35.	Time Statistics page	42
Figure 36.	Data injection page	43
Figure 37.	FSM page	44
Figure 38.	Configuration of FSM parameters - menu bar	44
Figure 39.	Configuration tab	45
Figure 40.	FSM configuration toolbar	46
Figure 41.	Testing tab	46
Figure 42.	Debug / Data injection tab	47
Figure 43.	Data patterns tab	48
Figure 44.	ARFF generation tab	49
Figure 45.	Data analysis	50
Figure 46.	Decision tree generation tab	51
Figure 47.	UCF generation tab	52
Figure 48.	Decision tree output viewer tab	53
Figure 49.	Data injection	54
Figure 50.	Configuration tab	55
Figure 51.	Debug tab	56
Figure 52.	Regression Tool tab	57
Figure 53.	Time domain page	58

Figure 54.	Control options for the line chart	58
Figure 55.	Cursor value table.	59
Figure 56.	Frequency domain page	60
Figure 57.	Spectrogram page	61
Figure 58.	AlgoBuilder	62
Figure 59.	AlgoBuilder principle of operation	63
Figure 60.	AlgoBuilder main page.	64
Figure 61.	Using the Feedback function block	69
Figure 62.	Angle Level function block and example of data visualization	70
Figure 63.	Bar Graph function block and example of data visualization	71
Figure 64.	Fusion function block and example of data visualization	72
Figure 65.	Graph function block and example of data visualization	73
Figure 66.	Logic Analyzer function block and example of data visualization	74
Figure 67.	Plot3D function block and example of data visualization	75
Figure 68.	Scatter function block and example of data visualization	76
Figure 69.	Value function block and example of data visualization	76
Figure 70.	Input Value function blocks.	77
Figure 71.	FFT and FFT Plot connections	78
Figure 72.	FFT Plot visualization	78
Figure 73.	Example of a design with FSM/MLC	79
Figure 74.	FSM/MLC function block properties	79
Figure 75.	Example of a design with ISPU	80
Figure 76.	ISPU function block Properties	80
Figure 77.	AlgoBuilder Firmware Settings window	82
Figure 78.	Sensor Hub, Acceleration [g], Graph function blocks	82
Figure 79.	Graph Properties	83
Figure 80.	Connected Sensor Hub, Acceleration, Graph function blocks	83
Figure 81.	Generated code in algo_builder.c file.	84
Figure 82.	Compiler output	85
Figure 83.	Programming target with DFU interface	86
Figure 84.	Principle of multilevel design	87
Figure 85.	Subdesigns dock	87
Figure 86.	Subdesign terminals	88
Figure 87.	Example of a subdesign block and its content.	88
Figure 88.	Example of loop (calculates the sum of all items in the input buffer).	89
Figure 89.	Firmware programming page	90

IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2024 STMicroelectronics – All rights reserved