



User Manual

Document revision 1.7

Document release date April 2023

Document number BST-DHW-AN013

Technical reference code(s) n.a.

Notes Data and descriptions in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product appearance.

Contents

1	Introduction	5
2	Accessing the sensor on Application Board using C and SensorAPI	7
2.1	Introduction to COINES	7
2.2	Working principles	7
2.2.1	Running examples on PC side	7
2.2.2	Running examples directly on the MCU of the Application board	8
3	Accessing the sensors on Nicla Sense ME using C and SensorAPI	8
3.1	COINES on Nicla Sense ME	8
4	Installation	9
4.1	System requirements	9
4.2	Installation (Windows)	9
4.2.1	Installation of COINES	9
4.2.2	Installation of compiler environment	9
4.3	Installation (Linux/MacOS)	10
4.3.1	Installation of COINES	10
4.3.2	Installation of compiler environment	11
5	Quick start guide	12
5.1	Compiling and executing code (command line)	12
5.2	Cross compiling and downloading example to Application Board's microcontroller	12
5.3	Cross compiling and downloading example to Nicla Sense ME's microcontroller	13
5.4	Eclipse project for examples	14
6	coinesAPI description	16
6.1	Overview of PC side implementation of COINES	16
6.2	GPIO mapping of APP2.0 shuttle board pins	17
6.3	GPIO mapping of APP3.0 shuttle board pins	17
6.4	coinesAPI calls: Interface and board information	18
6.4.1	coines_open_comm_intf	18
6.4.2	coines_close_comm_intf	18
6.4.3	coines_get_board_info	18
6.5	coinesAPI calls: GPIO oriented calls	19
6.5.1	coines_set_pin_config	19
6.5.2	coines_get_pin_config	19
6.5.3	coines_set_shuttleboard_vdd_vddio_config	19
6.6	coinesAPI calls: Sensor communication	19
6.6.1	coines_config_i2c_bus	19
6.6.2	coines_config_spi_bus	20
6.6.3	coines_config_i2s_bus	20
6.6.4	coines_deconfig_spi_bus	20
6.6.5	coines_deconfig_i2c_bus	20
6.6.6	coines_deconfig_i2s_bus	20
6.6.7	coines_write_i2c	20
6.6.8	coines_read_i2c	21
6.6.9	coines_i2c_set	21

6.6.10	coines_i2c_get	21
6.6.11	coines_write_spi	22
6.6.12	coines_read_spi	22
6.6.13	coines_config_word_spi_bus	22
6.6.14	coines_write_16bit_spi	22
6.6.15	coines_read_16bit_spi	23
6.6.16	coines_delay_msec	23
6.6.17	coines_delay_usec	23
6.7	coinesAPI calls: Streaming feature	23
6.7.1	coines_config_streaming	23
6.7.2	coines_start_stop_streaming	24
6.7.3	coines_read_stream_sensor_data	25
6.7.4	coines_trigger_timer	25
6.8	coinesAPI calls: Other useful APIs	26
6.8.1	coines_get_millis	26
6.8.2	coines_get_micro_sec	26
6.8.3	coines_attach_interrupt	26
6.8.4	coines_detach_interrupt	26
6.8.5	coines_intf_available	27
6.8.6	coines_intf_connected	27
6.8.7	coines_flush_intf	27
6.8.8	coines_read_intf	27
6.8.9	coines_write_intf	27
6.8.10	coines_get_version	28
6.8.11	coines_soft_reset	28
6.8.12	coines_read_temp_data	28
6.8.13	coines_read_bat_status	28
6.8.14	coines_ble_config	28
6.8.15	coines_set_led	28
6.8.16	coines_timer_config	29
6.8.17	coines_timer_deconfig	29
6.8.18	coines_timer_start	29
6.8.19	coines_timer_stop	29
6.8.20	coines_get_realtime_usec	29
6.8.21	coines_delay_realtime_usec	30
6.8.22	coines_attach_timed_interrupt	30
6.8.23	coines_detach_timed_interrupt	30
6.8.24	coines_echo_test	30
6.8.25	coines_shuttle_eeprom_write	30
6.8.26	coines_shuttle_eeprom_read	31
6.8.27	coines_yield	31
6.8.28	coines_execute_critical_region	31
6.8.29	coines_scan_ble_devices	31
7	Extending the usage of the example files	32
7.1	Simple data logging	32
7.2	Data plotting and visualization	32
8	Media Transfer Protocol (MTP) firmware for Application Board 3.0	34
8.1	Switching to MTP mode	36
8.2	Copying the files using MTP	36

9	USB/BLE DFU bootloader	37
9.1	Key Features	37
9.1.1	USB DFU	37
9.1.2	BLE DFU	37
9.2	Invoking the Bootloader	37
9.3	Using the Bootloader via USB	38
9.4	Using the Bootloader via BLE	39
10	Updating Bootloader, DD firmware and MTP firmware using COINES	40
10.1	Updating bootloader	40
10.2	Updating DD firmware	40
10.3	Updating MTP firmware	40
10.4	Updating Coines Bridge firmware on Nicla Sense ME Board	40
11	Accessing the Application Board using Python	41
11.1	Introduction to coinespy library	41
11.2	Installation	41
11.3	coinespy API description	41
11.3.1	coinespy API calls: Interface and board information	42
11.3.2	coinespy API calls: GPIO oriented calls	43
11.3.3	coinespy API calls: Sensor communication	44
11.3.4	coinespy API calls: Streaming feature	46
11.3.5	coinespy API calls: Other useful APIs	49
11.3.6	Definiton of constants	49
11.3.7	Error Codes	53
12	FAQ	55
13	Legal disclaimer	56
13.1	Engineering samples	56
13.2	Product use	56
13.3	Application examples and hints	56
14	Document history and modifications	57

1 Introduction

Bosch Sensortec offers a toolkit for evaluation of its sensor products. The toolkit consisting of 3 elements:

1. A sensor specific shuttle board also known as breakout board. APP3.0 shuttle boards also known as mini shuttle boards has smaller form factor when compared with APP2.0 shuttle board.

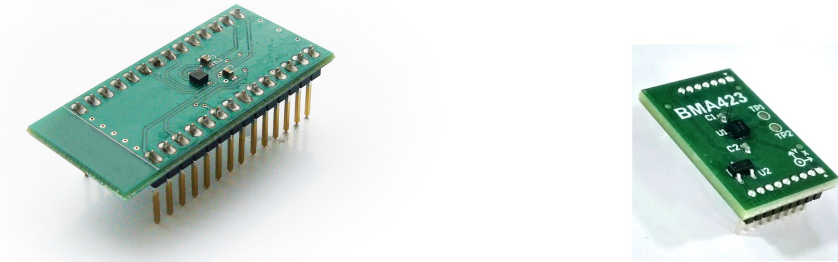


Fig. 1: APP2.0/3.0 sensor shuttle board

2. **Application Board** has a connector for the shuttle board and serves as interface translator from the sensor interface (I²C or SPI) to a USB interface, allowing PC software to communicate with the sensor on the shuttle.

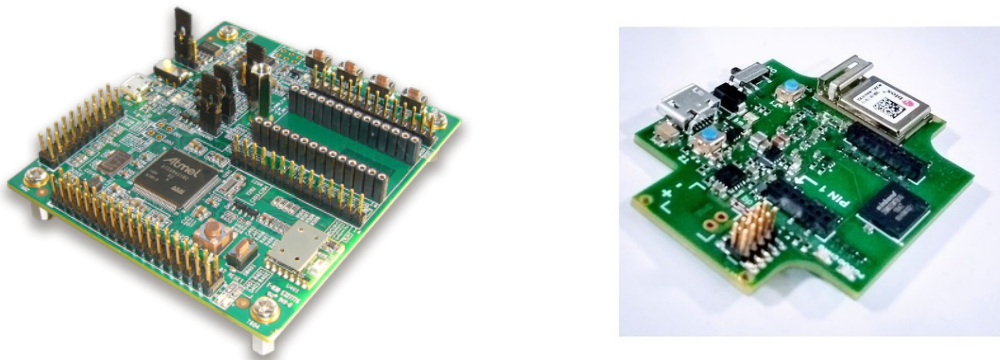


Fig. 2: Application Board 2.0/3.0

3. **Nicla Sense ME** board combines four state-of-the-art sensors from Bosch Sensortec (BHI260AP, BMP390, BMM150 and BME688) in the Arduino ecosystem.



Fig. 3: Nicla Sense ME

4. On the PC side, Bosch Sensortec provides the software packages Development Desktop 2.0 and COINES to connect to the sensor on the Application Board.
 - ▶ Development Desktop 2.0 provides a GUI which allows to configure the sensor, plot and export streamed sensor data.
 - ▶ COINES provides a C based interface, which allows to communicate with the sensor using the SensorAPI from Bosch Sensortec on the PC side.
 - ▶ Starting from COINES v2.0, user has an option to cross-compile the example and run it directly on the Application Board's microcontroller.

2 Accessing the sensor on Application Board using C and SensorAPI

2.1 Introduction to COINES

COINES ("COmmunication with INertial and Environmental Sensors") provides a low-level interface to Bosch Sensortec's Application Board. The user can access Bosch Sensortec's MEMS sensors through a C interface. COINES can be used with SensorAPI of the sensor. SensorAPI is available at <https://github.com/BoschSensortec>. The source code of example applications and SensorAPI are provided with the COINES library as a package. The user can modify, compile and run the sample applications.

COINES can be used to see how to use the SensorAPI in an embedded environment and allows convenient data logging.

The full working environment consists of:

- ▶ A Bosch Sensortec MEMS sensor on a shuttle board mounted on the socket of Bosch Sensortec's application board APP2.0/APP3.0
- ▶ Windows or Linux PC to which the Application Board is connected via USB
- ▶ COINES software release as found here: <http://www.bosch-sensortec.com>
- ▶ C compiler is also required (details see below)

2.2 Working principles

2.2.1 Running examples on PC side

When compiling the examples for PC side, the COINES layer provides an abstraction of the embedded environment on the host side. COINES library provides read and write functions for I²C and SPI on PC side. These functions receive the arguments of the user input (i.e. what register address to read from) and tunnel them through the USB connection to the Application Board, where they are fed into the embedded I²C and SPI functions and are executed to access the sensor. Any result or response from those functions is tunneled back to the PC side and provided to the example application.

This approach allows easy and flexible programming and offers the possibility to integrate the example code into other applications or add advanced logging options. The drawback is that in this mode the code is not executed in real time, as it runs on a multi-tasking operating system. To overcome this drawback, the examples can also be run on the MCU side (see next section).

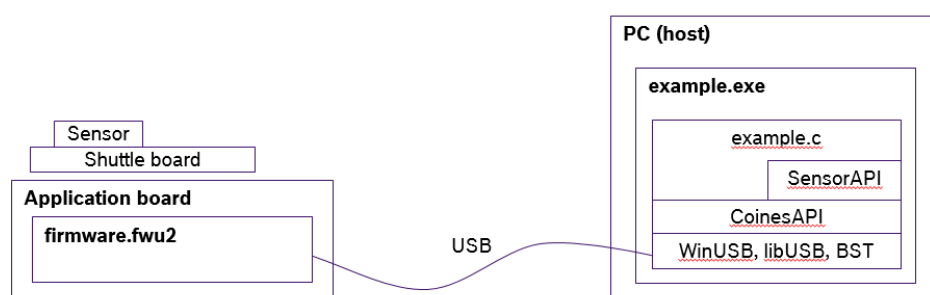


Fig. 4: Working principle: running example on PC side

2.2.2 Running examples directly on the MCU of the Application board

The examples can also be cross-compiled on PC side and downloaded into the memory of the Application board and executed there. The user can choose to download the created binary into the flash memory or into the RAM (if the binary is not too big).

Important is, that the example is placed in a location in the flash memory other than where the default firmware is stored. The example is executed with a specific command, allowing to jump to the start address of the compiled example from the default firmware. As the firmware itself is not overwritten, the board always returns to its default state after a power-off-power-on cycle.

In this configuration the COINES layer provides a simple abstraction on top of the MCU BSP (i.e. board level support layer of the microcontroller). Any printf command will now not output to the console, but rather to the USB connection, which appears as virtual COM port on PC side.

This mode allows to also perform many time-critical operations on the sensor, such as fast reading of FIFO content at high data rates.

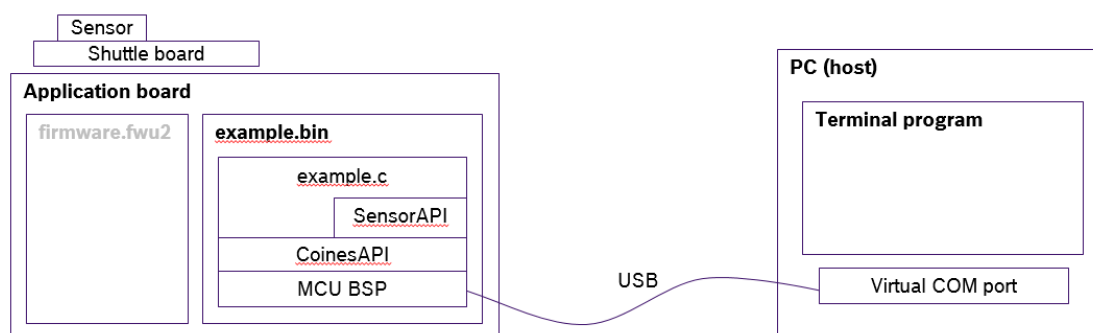


Fig. 5: Working principle: running example on the MCU of the Application board

3 Accessing the sensors on Nicla Sense ME using C and SensorAPI

3.1 COINES on Nicla Sense ME

COINES can be used to access the onboard Bosch sensors with the Nicla Sense ME. The working environment consists of windows or Linux PC to which the Nicla Sense ME Board is connected via USB. For details refer to section 2.

4 Installation

4.1 System requirements

COINES should be usable on any recent PC or laptop system which has at least a performance as an “office PC”. The hardware should provide a USB 2.0 interface.

COINES can run on recent versions of Windows and Linux.

Tested with following Operating Systems

- ▶ Windows 7,10
- ▶ Debian based - Ubuntu 14.04,16.04,18.04, Debian Jessie/Stretch
- ▶ Redhat based - CentOS 7 ,Fedora 27
- ▶ Raspbian (Raspberry Pi 3 hardware)

4.2 Installation (Windows)

4.2.1 Installation of COINES

1. Download the latest version of COINES from Bosch Sensortec’s website in the "Downloads" section
2. Run the Installer
3. Accept the End User License Agreement and click Next
4. Click Install to start Installation
5. Click Start → All programs → COINES → examples → respective sensors to view examples

4.2.2 Installation of compiler environment

COINES C examples can be built using GNU C compiler (GCC). There are various distributions of GCC. TDM-GCC is easy to install and hence preferred for COINES. TDM GCC is based on MinGW GCC.

If you have already installed GCC (MinGW/Cygwin/MSYS2 GCC) and added to 'PATH' environmental variable ,you can skip compiler installation.

1. Download the TDM32/TDM64 bundle ([link](#)). **Use TDM32 bundle if your Windows OS is 32-bit and TDM64 bundle if 64-bit.**
2. Start the Installer. Ensure that the option Check for updated files on the TDM GCC server is unchecked. Click Create and proceed with the installation
3. If you intend to do run the COINES example on Application Board’s microcontroller,install the latest version of [GNU Embedded Toolchain for ARM](#) for Windows. Make sure you have checked 'Add path to environmental variable'

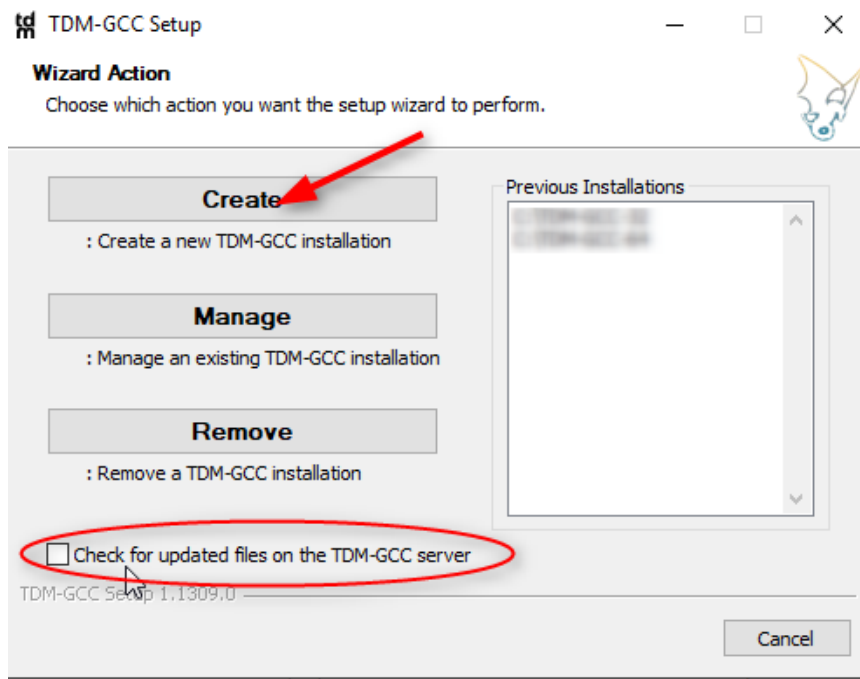


Fig. 6: TDM-GCC installation dialog

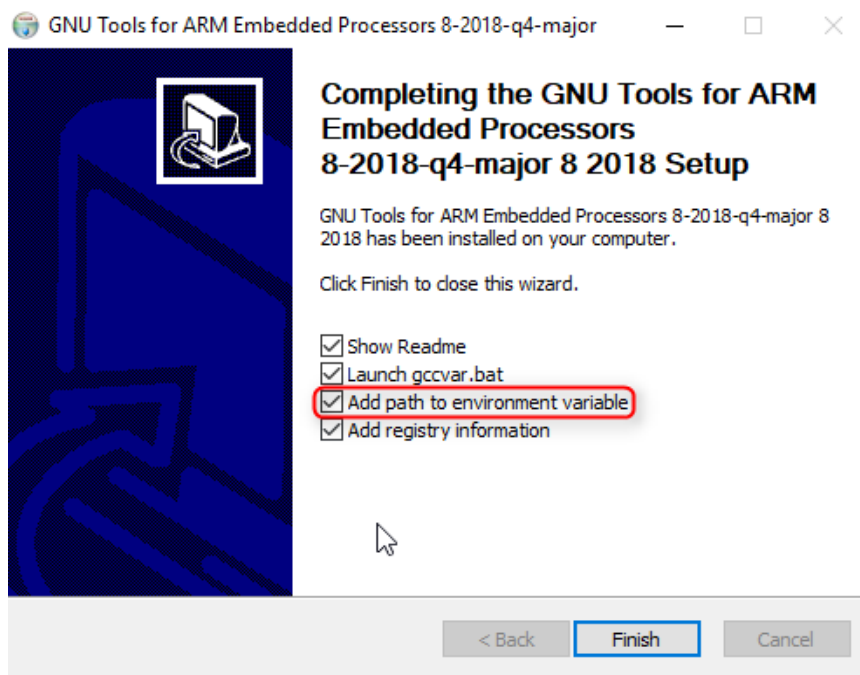


Fig. 7: GNU ARM Toolchain installation

4.3 Installation (Linux/MacOS)

4.3.1 Installation of COINES

1. Download the installer.
2. Use the command `cd` to go to the directory where the installer is located and make the installer executable:

```
$ chmod +x coines_vX.Y.sh
```

3. Ensure that you are connected to the Internet before running the installer, which is executed like this:

```
$ ./coines_vX.Y.sh
```

4. Accept the End User License agreement
5. The installer will prompt you if the required dependencies/packages are not installed. (This step requires root privileges.)

4.3.2 Installation of compiler environment

On a Debian or Redhat based Linux distro, the installer prompts for installation of missing dependencies, gcc, make and libusb-dev packages. If due to some reason installation fails, the user can manually install the dependencies.

- ▶ Debian based distros - gcc, make, libusb-1.0-0-dev, dfu-util
- ▶ Redhat based distros - gcc, make, libusb-devel, dfu-util
- ▶ MacOS - libusb, dfu-util

If you intend to run the COINES example on Application Board's microcontroller, download the latest version of [GNU Embedded Toolchain for ARM](#) for Linux and extract the package. Add the compiler to PATH variable by editing \$HOME/.bashrc or similar file like /etc/profile or /etc/environment

5 Quick start guide

5.1 Compiling and executing code (command line)

1. Connect the Application Board board via USB, with the sensor shuttle board mounted.
2. Open the command prompt or the terminal.
3. Use the command `cd` to go to the directory where the example that is to be built is located.
4. Type `'mingw32-make'` (TDM-GCC/MinGW) or `'make'` (Linux/Cygwin/MSYS2/MacOS)
5. Run the example and see the output.

5.2 Cross compiling and downloading example to Application Board's microcontroller

1. Make sure that [GNU Embedded Toolchain for ARM](#) is installed on your PC and added to environmental variable `PATH`
2. Connect the Application board via USB, with the sensor shuttle board mounted.
3. Open the command prompt or the terminal.
4. Use the command `cd` to go to the directory where the example that is to be built is located.
5. Type `mingw32-make TARGET=MCU_APP20 download`. Other available options are

Cross-compile for APP2.0 board	<code>mingw32-make TARGET=MCU_APP20</code>
Download example to APP2.0 MCU RAM	<code>mingw32-make LOCATION=RAM TARGET=MCU_APP20 download</code>
Download example to APP2.0 MCU FLASH	<code>mingw32-make LOCATION=FLASH TARGET=MCU_APP20 download</code>
Download example to APP3.0 MCU RAM	<code>mingw32-make LOCATION=RAM TARGET=MCU_APP30 download</code>
Download example to APP3.0 MCU FLASH *	<code>mingw32-make LOCATION=FLASH TARGET=MCU_APP30 download</code>
Compile for PC (Default)	<code>mingw32-make TARGET=PC</code>
Run an example already residing in APP2.0 Flash memory	<code>mingw32-make run</code>

Linux/MacOS/Cygwin/MSYS2 users can use `make`

NOTE: Downloading COINES example to APP3.0 Flash memory will overwrite default firmware.

6. Use a Serial Terminal application to view output.
 - ▶ Windows - PuTTY, HTerm, etc.,
 - ▶ Linux - `cat` command. Eg: `cat /dev/ttyACM0`
 - ▶ macOS - `screen` command. Eg: `screen /dev/tty.usbmodem9F31`
7. For bluetooth, use [Serial Bluetooth terminal](#).

Note:

- ▶ Some examples may not compile for both PC and MCU target. Please refer to the example documentation or simply the example name (e.g. examples that can only be compiled for the PC are named with a following `'_pc'`).
- ▶ The binary on the MCU will be executed once the serial port is opened. The port must be opened including DTR signal set, otherwise the binary will not be executed. Some terminal programs such as HTerm allow explicit setting of the DTR signal.
- ▶ For printing over APP3.0 bluetooth interface, use `fprintf(bt_w,...)`

5.3 Cross compiling and downloading example to Nicla Sense ME's microcontroller

1. Make sure that [GNU Embedded Toolchain for ARM](#) is installed on your PC and added to environmental variable PATH
2. Connect the Nicla Sense ME board via USB.
3. Open the command prompt or the terminal.
4. Use the command `cd` to go to the directory where the example that is to be built is located.
5. Type `mingw32-make TARGET=MCU_NICLA download` . Other available options are

Download example to NICLA MCU FLASH *	<code>mingw32-make LOCATION=FLASH TARGET=MCU_NICLA download</code>
Compile for PC (Default)	<code>mingw32-make TARGET=PC</code>

NOTE: Only Coins Bridge example supported in v2.8.8.

5.4 Eclipse project for examples

- Open Eclipse
- Click File → New → C/C++ Project
 - i. Input Project name → Uncheck use default location → Provide the location of the example folder

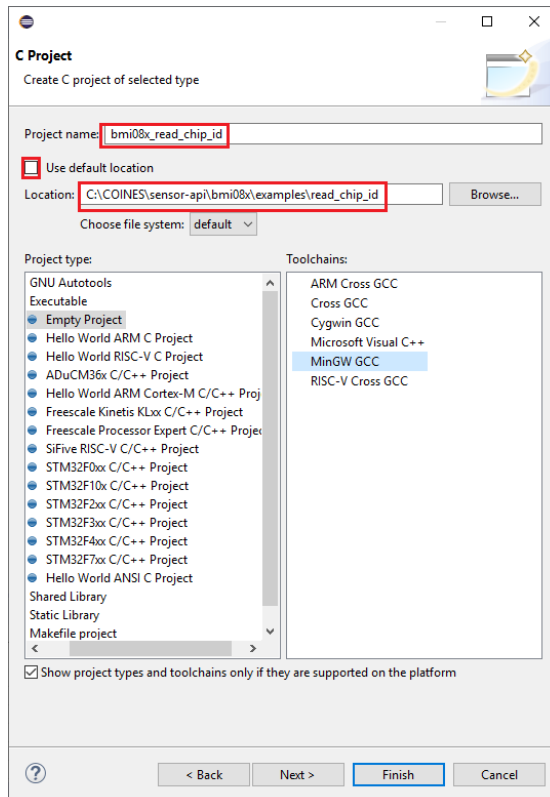


Fig. 8: Eclipse C Project for Windows

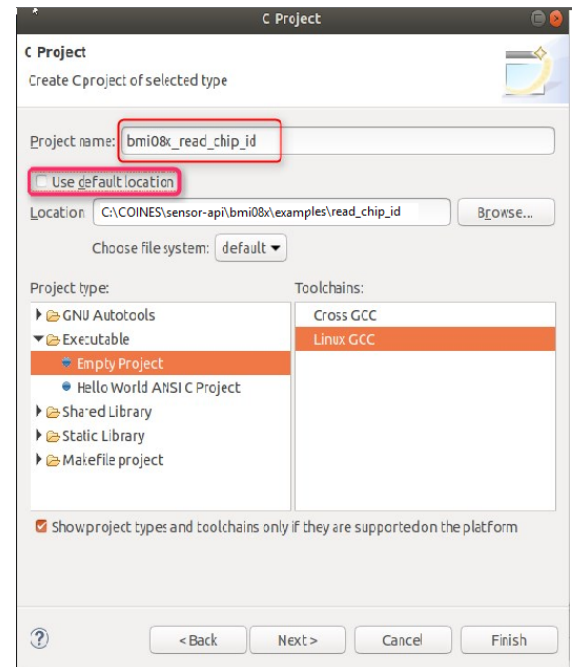


Fig. 9: Eclipse C Project for Linux

- ii. Select Executable → Empty project in Project type
- iii. For Windows, Select MinGW GCC as Toolchain
- iv. For Linux, Select Linux GCC as Toolchain
- In Project Explorer window, Right click on the project created → Click Properties → C/C++ Build → Tool Chain Editor → Select Current builder as Gnu Make Builder
- Again click on C/C++ Build
 - i. For Windows, Uncheck "Use default build command" and type build command as mingw32-make
 - ii. Uncheck generate Makefiles automatically
 - iii. Ensure Build location path is chosen from the workspace

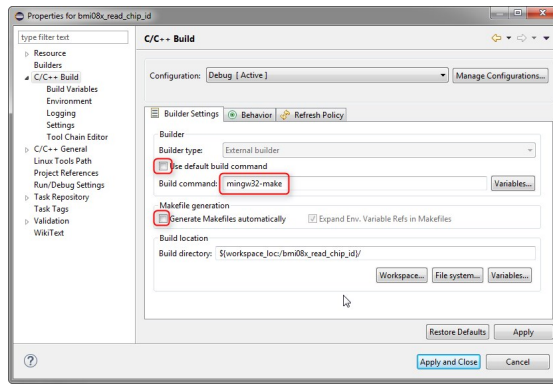


Fig. 10: Windows Eclipse Project Properties

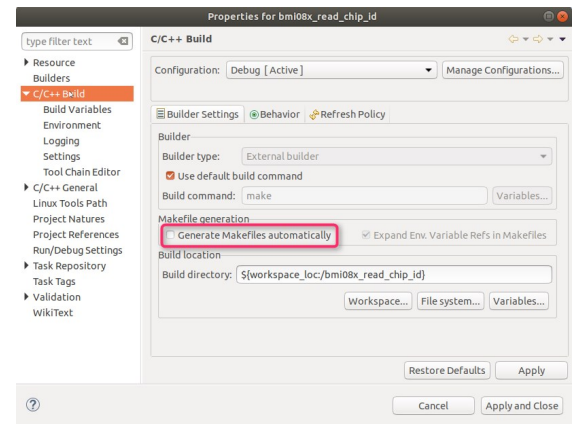


Fig. 11: Linux Eclipse Project Properties

- iv. Click Apply and Close button

Build project

In Project Explorer window, Right click on the project → Click Build Project. The executable file will be generated.

Debug project

- ▶ Click on Run -> Debug As -> Local C/C++ Application
- ▶ Once launching is completed, Click on
 - i. Resume button to run the application
 - ii. Terminate button to stop running the application

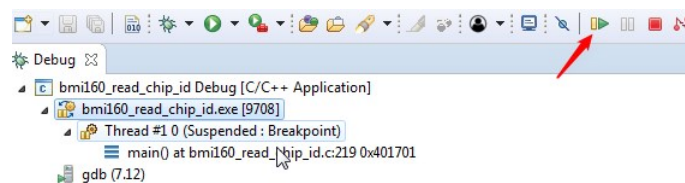


Fig. 12: Eclipse Debug Configuration

6 coinesAPI description

6.1 Overview of PC side implementation of COINES

Bosch Sensortec recommends using the SensorAPI in order to communicate with the sensors. The SensorAPI, an abstraction layer written in C makes it much more convenient for the user to access the register map of the sensor in order to configure certain functionality and obtain certain information from it.

For making use of the SensorAPI, three function pointers must be set to the appropriate read/write functions of the selected bus on the system (either I²C or SPI), as well as one function pointer to a system's function causing delays in milliseconds.

In order to execute C code using SensorAPI on a PC, the coinesAPI provides the mentioned read, write, delay functions. These functions are wrapper functions, embedding the actual SensorAPI payloads into a transport package, sending this via USB to the APP2.0, where the payload is translated into corresponding SPI or I²C messages and sent to the sensor on the shuttle board. The mapping would look similar to the one below.

```
#include "bst_sensor.h"

struct bst_sensor_dev sensordev;
....
....
sensordev.dev_id = I2C_ADDR; // SPI - CS PIN
sensordev.read = coines_read_i2c; // coines_read_spi
sensordev.write = coines_write_i2c; // coines_write_spi
sensordev.delay_ms = coines_delay_msec;
```

Using this method, the full functionality of the SensorAPI can be used on PC side, sample code can be modified and tested, and data can be logged in a convenient way.

This setup has the challenge of lacking the real-time capabilities known from a pure microcontroller environment. To overcome this, the coinesAPI offers streaming functions, which allow the user to schedule data readout directly on the microcontroller, either based on a data interrupt coming from the sensors or based on the timer of the microcontroller. The scheduler waits for the configured interrupt (sensor interrupt or timer interrupt) and reads out areas of the register map, which can be configured by the user.

As an example, the user could choose to read out the 6 bytes from the register map of a certain inertial sensor, containing the sensor data of three axis (2 bytes per axis). If the user would configure for example a readout once per milliseconds, the result would be a data stream of three-axis sensor data at a rate of 1 kHz.

6.2 GPIO mapping of APP2.0 shuttle board pins

The APP2.0 shuttle board has total 28 pins, of which some have a predefined functionality and some can be used as GPIO by the user.

The shuttle board connector details are given in the table below.

Pin number on shuttle board	Name / function	Pin number on shuttle board	Name / function
1	VDD (3.3V)	28	SHTLE_COD #4
2	VDDIO (3.3V)	27	SHTLE_COD #3
3	GND	26	SHTLE_COD #2
4	SPI MISO	25	SHTLE_COD #1
5	SPI: MOSI / I ² C: SDA	24	SHTLE_COD #0
6	SPI: SCK / I ² C: SCL	23	SHTLE_COD_GND
7	SPI: CS	22	IO_4 (GPIO #4)
8	IO_5 (GPIO #5)	21	IO_7 (GPIO #7)
9	IO_0 (GPIO #0)	20	IO_6 (GPIO #6)
10	SHTLE_COD #5	19	IO_8 (GPIO #8)
11	SHTLE_COD #6	18	SCL (see note)
12	SHTLE_COD #7	17	SDA (see note)
13	SHTLE_COD #8	16	IO_3 (GPIO #3)
14	IO_1 (GPIO #1)	15	IO_2 (GPIO #2)

Table 1: Overview of shuttle board pins and their function

Note:

- In coinesAPI the pins are addressed using the same numbers as on the shuttle board. For example, the GPIO #5 has the pin number 8.
- In some cases (depending on the sensor), the I²C lines are shuttle board pin 6 for the clock signal SCL and shuttle board pin 5 for the data line SDA. In such cases pins 17 and 18 may not be connected. Please carefully read the shuttle board documentation.

6.3 GPIO mapping of APP3.0 shuttle board pins

The APP3.0 shuttle board has a total of 16 pins, 7 on the left and 9 on the right. (with shuttle board pins facing downwards)

Note:

- In coinesAPI the pins are addressed as on the APP3.0 shuttle board. For example, the GPIO #5 is addressed as COINES_MINI_SHUTTLE_PIN_2_6.
- Supported VDD voltages on APP3.0 board are 0, 1.8V and 2.8V.
- Supported VDDIO voltage on APP3.0 board is 1.8V.

Pin number on shuttle board	Name / function	Pin number on shuttle board	Name / function
1_1	VDD (1.8/2.8V)	2_1	SPI_CS
1_2	VDDIO (1.8)	2_2	SPI: SCK / I ² C: SCL
1_3	GND	2_3	SPI: MISO
1_4	GPIO0	2_4	SPI: MOSI / I ² C: SDA
1_5	GPIO1	2_5	GPIO4*
1_6	GPIO2	2_6	GPIO5*
1_7	GPIO3	2_7	IOXP_INT*
		2_8	PlugDet*
		2_9	EEPROM_RW

*SPI pins for secondary interface - CS:GPIO4, SCK:GPIO5, MISO:IOXP_INT, MOSI:PlugDet

Table 2: Overview of APP3.0 shuttle board pins and their function

6.4 coinesAPI calls: Interface and board information

6.4.1 coines_open_comm_intf

Opens the communication interface. Currently only COINES_COMM_INTF_USB (USB Connection) interface is available. COINES_COMM_INTF_BLE is available for MCU_APP30 target.

In case of MCU Target, API waits indefinitely for serial port or BLE connection (MCU_APP30 target only).

In order to use `fprintf` and `fscanf` with BLE, `intf_type` should be COINES_COMM_INTF_BLE

```
int16_t coines_open_comm_intf(enum coines_comm_intf intf_type, void *arg);
```

6.4.2 coines_close_comm_intf

Closes the communication interface.

```
int16_t coines_close_comm_intf(enum coines_comm_intf intf_type, void *arg);
```

6.4.3 coines_get_board_info

Gets the board information.

```
int16_t coines_get_board_info(struct coines_board_info *data);
```

The data structure contains the following items

```
struct coines_board_info {
    /*Board hardware ID */
    uint16_t hardware_id;
    /*Board software ID */
};
```

```
uint16_t software_id;
/*Type of the board like APP2.0, Arduino Due*/
uint8_t board;
/*Shuttle ID of the sensor connected*/
uint16_t shuttle_id;
};
```

6.5 coinesAPI calls: GPIO oriented calls

6.5.1 coines_set_pin_config

Sets the pin direction and the state.

```
int16_t coines_set_pin_config(enum coines_multi_io_pin pin_number, enum
    coines_pin_direction direction, enum coines_pin_value pin_value);
```

6.5.2 coines_get_pin_config

Gets the pin configuration.

```
int16_t coines_get_pin_config(enum coines_multi_io_pin pin_number, enum
    coines_pin_direction *pin_direction, enum coines_pin_value *pin_value);
```

6.5.3 coines_set_shuttleboard_vdd_vddio_config

Configures the VDD and VDDIO of the sensor. For APP2.0, a voltage level of 0 or 3300 mV is supported. Any values above 0 will default to 3300 mV.

```
int16_t coines_set_shuttleboard_vdd_vddio_config(uint16_t vdd_millivolt, uint16_t
    vddio_millivolt);
```

6.6 coinesAPI calls: Sensor communication

6.6.1 coines_config_i2c_bus

Configures the I²C bus.

```
int16_t coines_config_i2c_bus(enum coines_i2c_bus bus, enum coines_i2c_mode i2c_mode);
```

The first argument refers to the bus on the board. Currently, on APP2.0, there is only one bus available, so the argument is always COINES_I2C_BUS_0.

The following I²C modes are available:

```
COINES_I2C_STANDARD_MODE
COINES_I2C_FAST_MODE
COINES_I2C_SPEED_3_4_MHZ
COINES_I2C_SPEED_1_7_MHZ
```

6.6.2 coines_config_spi_bus

Configures the SPI bus of the board. The argument `coines_spi_bus` refers to the bus on the board. On APP2.0, there is only one bus available, so the user should only use `COINES_SPI_BUS_0`. The SPI speed can be chosen in various discrete steps, as defined in enum `coines_spi_speed` in `coines.h`. (For example, `COINES_SPI_SPEED_2_MHZ` sets the SPI speed to 2 MHz.)

```
int16_t coines_config_spi_bus(enum coines_spi_bus bus, uint32_t spi_speed, enum
                             coines_spi_mode spi_mode);
```

6.6.3 coines_config_i2s_bus

This API is used to configure the I²S bus to match the TDM configuration

```
int16_t coines_config_i2s_bus(uint16_t data_words, coines_tdm_callback callback);
```

Arguments:

- ▶ `data_words`: number of words to use in the buffer. Max is set at `COINES_TDM_BUFFER_SIZE_WORDS`.
- ▶ `callback`: register a callback to be called to process and copy the data.

6.6.4 coines_deconfig_spi_bus

This API is used to de-configure the SPI bus

```
int16_t coines_deconfig_spi_bus(enum coines_spi_bus bus);
```

6.6.5 coines_deconfig_i2c_bus

This API is used to de-configure the I²C bus

```
int16_t coines_deconfig_i2c_bus(enum coines_i2c_bus bus);
```

6.6.6 coines_deconfig_i2s_bus

This API is used to stop the I²S/TDM interface from reading data from the sensor

```
void coines_deconfig_i2s_bus(void);
```

6.6.7 coines_write_i2c

Writes 8-bit register data to the I²C device at `COINES_I2C_BUS_0`.

```
int8_t coines_write_i2c(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t reg_addr,
                       uint8_t *reg_data, uint16_t count);
```

Arguments:

- ▶ `bus`: I²C bus to be used
- ▶ `dev_addr`: I²C device address.

- ▶ reg_addr: Starting address for writing the data.
- ▶ reg_data: Data to be written.
- ▶ count: Number of bytes to write.

6.6.8 coines_read_i2c

Reads 8-bit register data from the I²C device at COINES_I2C_BUS_0.

```
int8_t coines_read_i2c(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t reg_addr,  
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- ▶ bus: I²C bus to be used
- ▶ dev_addr: I²C device address.
- ▶ reg_addr: Starting address for reading the data.
- ▶ reg_data: Buffer to take up the read data.
- ▶ count: Number of bytes to read.

6.6.9 coines_i2c_set

This API is used to write the data in I2C communication.

```
int8_t coines_i2c_set(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t *data, uint8_t  
    count);
```

Arguments:

- ▶ bus: I²C bus to be used
- ▶ dev_addr: I²C device address.
- ▶ data: Data to be written.
- ▶ count: Number of bytes to write.

6.6.10 coines_i2c_get

This API is used to read the data in I2C communication.

```
int8_t coines_i2c_get(enum coines_i2c_bus bus, uint8_t dev_addr, uint8_t *data, uint8_t  
    count);
```

Arguments:

- ▶ bus: I²C bus to be used
- ▶ dev_addr: I²C device address.
- ▶ data: Data read from the sensor.
- ▶ count: Number of bytes to read.

6.6.11 coines_write_spi

Writes 8-bit register data to the SPI device at COINES_SPI_BUS_0.

```
int8_t coines_write_spi(enum coines_spi_bus bus, uint8_t dev_addr, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- ▶ bus: SPI bus to be used.
- ▶ dev_addr: Chip select pin number.
- ▶ reg_addr: Starting address for writing the data.
- ▶ reg_data: Data to be written.
- ▶ count: Number of bytes to write.

6.6.12 coines_read_spi

Reads 8-bit register data from the SPI device at COINES_SPI_BUS_0.

```
int8_t coines_read_spi(enum coines_spi_bus bus, uint8_t dev_addr, uint8_t reg_addr,
    uint8_t *reg_data, uint16_t count);
```

Arguments:

- ▶ bus: SPI bus to be used.
- ▶ dev_addr: Chip select pin number.
- ▶ reg_addr: Starting address for reading the data.
- ▶ reg_data: Buffer to take up the read data.
- ▶ count: Number of bytes to read.

6.6.13 coines_config_word_spi_bus

Configures the SPI bus parameters speed, mode, 8-bit/16-bit transfer (COINES_SPI_TRANSFER_8BIT / COINES_SPI_TRANSFER_16BIT).

```
int16_t coines_config_word_spi_bus(enum coines_spi_bus bus, enum coines_spi_speed
    spi_speed, enum coines_spi_mode spi_mode, enum coines_spi_transfer_bits
    spi_transfer_bits);
```

6.6.14 coines_write_16bit_spi

Writes 16-bit register data to the SPI device at COINES_SPI_BUS_0.

```
int8_t coines_write_16bit_spi(enum coines_spi_bus bus, uint8_t cs, uint16_t reg_addr,
    void *reg_data, uint16_t count);
```

Arguments:

- ▶ bus: SPI bus to be used.
- ▶ cs: Chip select pin number.
- ▶ reg_addr: Starting address for writing the data.

- ▶ `reg_data`: Data to be written.
- ▶ `count`: Number of bytes to write.

6.6.15 `coines_read_16bit_spi`

Reads 16-bit register data from the SPI device at COINES_SPI_BUS_0.

```
int8_t coines_read_16bit_spi(enum coines_spi_bus bus, uint8_t cs, uint16_t reg_addr,
    void *reg_data, uint16_t count);
```

Arguments:

- ▶ `bus`: SPI bus to be used.
- ▶ `cs`: Chip select pin number.
- ▶ `reg_addr`: Starting address for reading the data.
- ▶ `reg_data`: Buffer to take up the read data.
- ▶ `count`: Number of bytes to read.

6.6.16 `coines_delay_msec`

Introduces delay in millisecond.

```
void coines_delay_msec(uint32_t delay_ms);
```

6.6.17 `coines_delay_usec`

Introduces delay in microsecond.

```
void coines_delay_usec(uint32_t delay_us);
```

6.7 `coinesAPI` calls: Streaming feature

Note :

1. The below APIs are supported only on PC Target.
2. A simpler approach of using `coines_attach_interrupt()` API for is available for MCU.

6.7.1 `coines_config_streaming`

Sets the configuration for streaming sensor data.

```
int16_t coines_config_streaming(uint8_t channel_id, struct coines_streaming_config
    *stream_config, struct coines_streaming_blocks *data_blocks);
```

Arguments:

- ▶ `channel_id`: An integer number that can be used as identifier/index to the sensor data that will be streamed for this setting
- ▶ `stream_config`: Contains information regarding interface settings and streaming configuration.

- ▶ `coins_streaming_blocks`: Contains information regarding numbers of blocks to read, register address and size for each block.

Note:

The below parameters should always be set:

- ▶ `data_block.no_of_blocks`: number of blocks to stream (must at least be one)
- ▶ For each block `b`:
 - ▶ `data_block.reg_start_addr[b]`: start address of the block in the register map
 - ▶ `stream_block.no_of_data_bytes[b]`: number of bytes to read, starting from the start address

For reading data from I²C bus, then set the below parameters:

- ▶ `stream_config.intf = COINES_SENSOR_INTF_I2C`;
- ▶ `stream_config.i2c_bus`: I²C bus (in case of APP2.0, this is always `COINES_I2C_BUS_0`)
- ▶ `stream_config.dev_addr`: I²C address of the sensor

For reading data from SPI bus, then set the below parameters:

- ▶ `stream_config.intf = COINES_SENSOR_INTF_SPI`;
- ▶ `stream_config.spi_bus`: SPI bus (in case of APP2.0, this is always `COINES_SPI_BUS_0`)
- ▶ `stream_config.cs_pin`: CS pin of the sensor, information can be obtained from the shuttle board documentation for the sensor.

When polling mode is requested, set the below parameters:

- ▶ `stream_config.sampling_units`:
either milliseconds (`COINES_SAMPLING_TIME_IN_MILLI_SEC`)
or microseconds (`COINES_SAMPLING_TIME_IN_MICRO_SEC`)
- ▶ `stream_config.sampling_time`: sampling period in the unit as defined in `stream_config.sampling_units`

When interrupt mode is requested, set the below parameters:

- ▶ `stream_config.int_pin`: pin of the interrupt which shall trigger the sensor read-out. If the interrupt output of the sensor is used, the required information about the pin number can be obtained from the shuttle board documentation for the sensor.
- ▶ `stream_config.int_timestamp`: it can be configured if the sensor data is tagged with a timestamp (`COINES_TIMESTAMP_ENABLE`) or not (`COINES_TIMESTAMP_DISABLE`).

6.7.2 coins_start_stop_streaming

Starts or stops sensor data streaming.

```
int16_t coins_start_stop_streaming(enum coins_streaming_mode stream_mode, uint8_t start_stop);
```

Arguments:

- ▶ `stream_mode`: streaming mode (either `COINES_STREAMING_MODE_POLLING` or `COINES_STREAMING_MODE_INTERRUPT`)
- ▶ `start_stop`: flag to either start (`COINES_STREAMING_START`) or stop (`COINES_STREAMING_STOP`) the streaming

6.7.3 coines_read_stream_sensor_data

Reads the data streamed from the sensor.

```
int16_t coines_read_stream_sensor_data(uint8_t sensor_id, uint32_t number_of_samples,
    uint8_t *data, uint32_t *valid_samples_count);
```

Arguments:

- ▶ **sensor_id**: id of the sensor
- ▶ **number_of_samples**: number of samples the user wishes to read (not implemented)
- ▶ **data**: data buffer
 - ▶ Interrupt streaming - Packet counter + Register data + Timestamp
 - ▶ Polling streaming - Register data
- ▶ **valid_samples_count**: number of samples the user has actually received (may be less than **number_of_samples**)

Example of a packet:

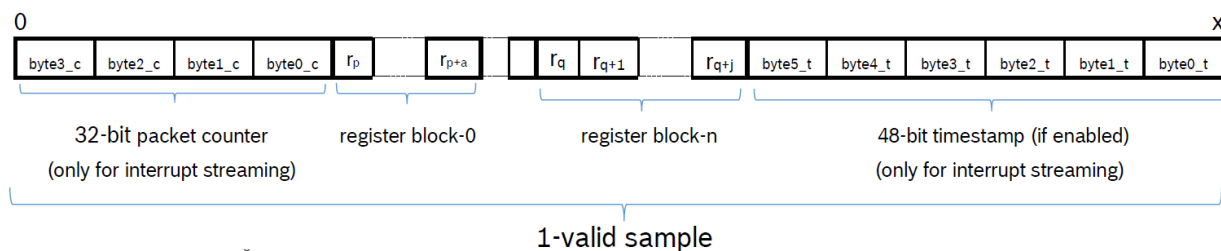


Fig. 13: Format of streaming packages

In the above figure, the following meaning apply to the mentioned abbreviations:

- ▶ **r_p**: Value at register address p
- ▶ **a**: Size of register block-0
- ▶ **r_{p+a}**: Value at register address p

Similarly is the case for **r_q**, **j** and **r_{q+j}**. See the **coines_streaming_blocks** structure for information regarding register blocks.

The packet counter and the timestamp can be obtained as follows:

```
packet_counter = (byte3_c << 24) | (byte2_c << 16) | (byte1_c << 8) | (byte0_c)
timestamp = (byte5_t << 40) | (byte4_t << 32) | (byte3_t << 24) | (byte2_t << 16) |
(byte1_t << 8) | (byte0_t)
```

The 48-bit timestamp is enabled by using

```
coines_trigger_timer(COINES_TIMER_START, COINES_TIMESTAMP_ENABLE);
```

Timestamp in microseconds can be obtained using below formula:

$$Timestamp (\mu s) = \frac{48bit_timestamp}{30}$$

6.7.4 coines_trigger_timer

Triggers the timer in firmware and also enables or disables the time stamp feature.

```
int16_t coines_trigger_timer(enum coines_timer_config tmr_cfg, enum
    coines_time_stamp_config ts_cfg);
```

Arguments:

- ▶ `tmr_cfg`: start, stop or reset the timer (COINES_TIMER_START, COINES_TIMER_STOP or COINES_TIMER_RESET)
- ▶ `ts_cfg`: Enables/disables microcontroller timestamp (COINES_TIMESTAMP_ENABLE or COINES_TIMESTAMP_DISABLE)

6.8 coinesAPI calls: Other useful APIs

6.8.1 coines_get_millis

Returns the number of milliseconds passed since the program started

```
uint32_t coines_get_millis();
```

6.8.2 coines_get_micro_sec

Returns the number of microseconds passed since the program started

```
uint64_t coines_get_micro_sec();
```

6.8.3 coines_attach_interrupt

Attaches an interrupt to a Multi-IO pin. Works only on MCU.

```
void coines_attach_interrupt(enum coines_multi_io_pin pin_number, void
    (*callback)(uint32_t, uint32_t), enum coines_pin_interrupt_mode int_mode);
```

Arguments:

- ▶ `pin_number`: Multi-IO pin
- ▶ `callback`: Name of the function to be called on detection of interrupt
- ▶ `int_mode`: Trigger modes - change (COINES_PIN_INTERRUPT_CHANGE), rising edge (COINES_PIN_INTERRUPT_RISING_EDGE), falling edge (COINES_PIN_INTERRUPT_FALLING_EDGE)

6.8.4 coines_detach_interrupt

Detaches interrupt from a Multi-IO pin. Works only on MCU.

```
void coines_detach_interrupt(enum coines_multi_io_pin pin_number);
```

Arguments:

- ▶ `pin_number`: Multi-IO pin.

6.8.5 coines_intf_available

Return the number of bytes available in the read buffer of the interface. Works only on APP3.0 MCU target.

```
uint16_t coines_intf_available(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

6.8.6 coines_intf_connected

Check if the interface is connected. Works only on APP3.0 MCU target.

```
bool coines_intf_connected(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

6.8.7 coines_flush_intf

Flush the write buffer. Works only on APP3.0 MCU target.

```
void coines_flush_intf(enum coines_comm_intf intf);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)

6.8.8 coines_read_intf

Read data over the specified interface. Works only on APP3.0 MCU target.

```
uint16_t coines_read_intf(enum coines_comm_intf intf, void *buffer, uint16_t len);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)
- `buffer`: Pointer to the buffer to store the data
- `len`: Length of the buffer

6.8.9 coines_write_intf

Write data over the specified interface. Works only on APP3.0 MCU target.

```
uint16_t coines_write_intf(enum coines_comm_intf intf, void *buffer, uint16_t len);
```

Arguments:

- `intf`: Type of interface (USB, COM, or BLE)
- `buffer`: Pointer to the buffer storing the data
- `len`: Length of the buffer

6.8.10 coines_get_version

Returns pointer to COINES version string

```
char* coines_get_version(void);
```

6.8.11 coines_soft_reset

Resets the device. After reset device jumps to the address specified in makefile(APP_START_ADDRESS).

```
void coines_soft_reset(void);
```

6.8.12 coines_read_temp_data

This API is used to read the temperature sensor data.

```
int16_t coines_read_temp_data(float *temp_data);
```

Arguments:

- ▶ temp_conv_data: Buffer to retrieve the sensor data in degree Celsius.

6.8.13 coines_read_bat_status

This API is used to read the battery status.

```
int16_t coines_read_bat_status(uint16_t *bat_status_mv, uint8_t *bat_status_percent);
```

Arguments:

- ▶ bat_status_mv: Buffer to retrieve the battery status in millivolt
- ▶ bat_status_percent: Buffer to retrieve the battery status in percentage

6.8.14 coines_ble_config

This API is used to configure BLE name and power. It should be called before calling coines_open_comm_intf API.

```
int16_t coines_ble_config(struct coines_ble_config *ble_config);
```

Arguments:

- ▶ ble_config: structure holding ble name and power details

6.8.15 coines_set_led

This API is used to set led state(on or off).

```
int16_t coines_set_led(enum coines_led led, enum coines_led_state led_state);
```

Arguments:

- ▶ led: led to which the state has to be set.

- ▶ `led_state`: state to be set to the given led.

6.8.16 `coines_timer_config`

This API is used to configure the hardware timer.

```
int16_t coines_timer_config(enum coines_timer_instance instance, void* handler);
```

Arguments:

- ▶ `instance`: timer instance.
- ▶ `handler`: callback to be called when timer expires.

6.8.17 `coines_timer_deconfig`

This API is used to de-configure the hardware timer.

```
int16_t coines_timer_deconfig(enum coines_timer_instance instance);
```

Arguments:

- ▶ `instance`: timer instance.

6.8.18 `coines_timer_start`

This API is used to start the configured hardware timer.

```
int16_t coines_timer_start(enum coines_timer_instance instance, uint32_t timeout);
```

Arguments:

- ▶ `instance`: timer instance.
- ▶ `timeout`: timeout in microseconds.

6.8.19 `coines_timer_stop`

This API is used to stop the hardware timer.

```
int16_t coines_timer_stop(enum coines_timer_instance instance);
```

Arguments:

- ▶ `instance`: timer instance.

6.8.20 `coines_get_realtime_usec`

This API is used to get the current counter(RTC) reference time in usec

```
uint32_t coines_get_realtime_usec(void);
```

6.8.21 coines_delay_realtime_usec

This API is used to introduce delay based on high precision RTC(LFCLK crystal) with the resolution of 30.517 usec.

```
void coines_delay_realtime_usec(uint32_t period);
```

Arguments:

- ▶ period: required delay in microseconds

6.8.22 coines_attach_timed_interrupt

Attaches a timed interrupt to a Multi-IO pin.

```
int16_t coines_attach_timed_interrupt(enum coines_multi_io_pin pin_number, void  
    (*timed_interrupt_cb)(uint64_t,uint32_t,uint32_t), enum coines_pin_interrupt_mode  
    int_mode);
```

Arguments:

- ▶ pin_number: Multi-IO pin.
- ▶ timed_interrupt_cb: Name of the function to be called on detection of interrupt.
- ▶ int_mode: Trigger modes - change, rising edge, falling edge.

6.8.23 coines_detach_timed_interrupt

Detaches a timed interrupt from a Multi-IO pin.

```
int16_t coines_detach_timed_interrupt(enum coines_multi_io_pin pin_number);
```

Arguments:

- ▶ pin_number: Multi-IO pin.

6.8.24 coines_echo_test

This API is used to test the communication.

```
int16_t coines_echo_test(uint8_t *data, uint16_t length);
```

Arguments:

- ▶ data: Data to be sent for testing.
- ▶ length: Length of the data.

6.8.25 coines_shuttle_eeprom_write

This API is used to write the content into shuttle eeprom.

```
int16_t coines_shuttle_eeprom_write(uint16_t start_addr, uint8_t *buffer, uint16_t  
    length);
```

Arguments:

- ▶ `start_addr`: EEPROM write address.
- ▶ `buffer`: Pointer to the buffer.
- ▶ `length`: Length of the buffer.

6.8.26 `coins_shuttle_eeprom_read`

This API is used to read the content from shuttle eeprom.

```
int16_t coins_shuttle_eeprom_read(uint16_t start_addr, uint8_t *buffer, uint16_t length);
```

Arguments:

- ▶ `start_addr`: EEPROM read address.
- ▶ `buffer`: Pointer to the buffer.
- ▶ `length`: Length of the buffer.

6.8.27 `coins_yield`

This API can be defined to perform a task when yielded from an ongoing blocking call.

```
void coins_yield(void);
```

6.8.28 `coins_execute_critical_region`

This API is used to execute the function inside critical region.

```
void coins_execute_critical_region(coins_critical_callback callback);
```

Arguments:

- ▶ `callback`: function to execute.

6.8.29 `coins_scan_ble_devices`

This API is used to connect to BLE Adapter and return list of BLE peripherals found during BLE scan.

```
int8_t coins_scan_ble_devices(struct ble_peripheral_info *ble_info, uint8_t *peripheral_count, size_t scan_timeout_ms)
```

Arguments:

- ▶ `ble_info`: array of struct containing found BLE peripheral information
- ▶ `peripheral_count`: number of BLE peripherals found
- ▶ `scan_timeout_ms`: timeout for BLE scan

7 Extending the usage of the example files

7.1 Simple data logging

The output data generated by the example files can easily be routed into log files for storing of the data. The following code snippet shows what the user would have to do in principle to generate a log file, stored in the current working directory, on each example execution. The name of the log file is derived from the current time stamp at the time of execution. The code snippet is valid for examples compiled for PC side (TARGET=PC, see above). If the example is run on the MCU, the data is provided via virtual COM port and the user can use any terminal program to access and store the data.

Note that the code snippet does not contain any exception handling, such as checking file overwrite or if fopen returns without error.

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *log_fd;
    char *logfile = malloc(28);
    time_t now;
    struct tm *tm;

    now = time(0);
    tm = localtime(&now);
    sprintf(logfile, "logfile_%04d%02d%02d_%02d%02d%02d.log",
        tm->tm_year+1900, tm->tm_mon+1, tm->tm_mday, tm->tm_hour, tm->tm_min, tm->tm_sec);
    log_fd = fopen(logfile, "w");

    ...

    while(CONDITION)
    {
        ...
        bmaXYZ_get_data(&data);
        fprintf(log_fd, "%d, %d, %d", data.x, data.y, data.z);
    }

    fclose(log_fd);
    return 0;
}
```

7.2 Data plotting and visualization

When compiling an example to run on MCU (for example TARGET=MCU_APP20, see above), the obtained sensor data can easily be plotted in the serial plotter of the Arduino IDE.

The example application must print the sensor data to be plotted in a text string, with a terminating new line character. Multiple sensor values per axis are possible. The printf command will stream the sensor data in an ASCII string via (virtual) COM port. Once the user connects to the COM port and opens the Arduino serial plotter, the data will be displayed in a graphical way.

Notes and hints:

- If the user wants to use an other plotting software, he must consider that the DTR signal line must be set, otherwise the flashed application on the application board will not start running.

The serial plotter and serial monitor of Arduino IDE set this signal automatically, other software (like HTerm) have the option to do this manually.

- The plotting window offers automatic re-sizing. If the user does not want this and needs fixed limits, he could plot the limits as additional lines.

Example: `printf("%d %d %d\n", lower_limit, sensor_data, upper_limit);`

- In case of sensor data with a high offset, such as the output of a barometric pressure sensor, which is usually around 100000 Pa, the user may want to subtract a certain offset, so see details of the signal.

Example: `printf("%d\n", (pressure - 99000));`

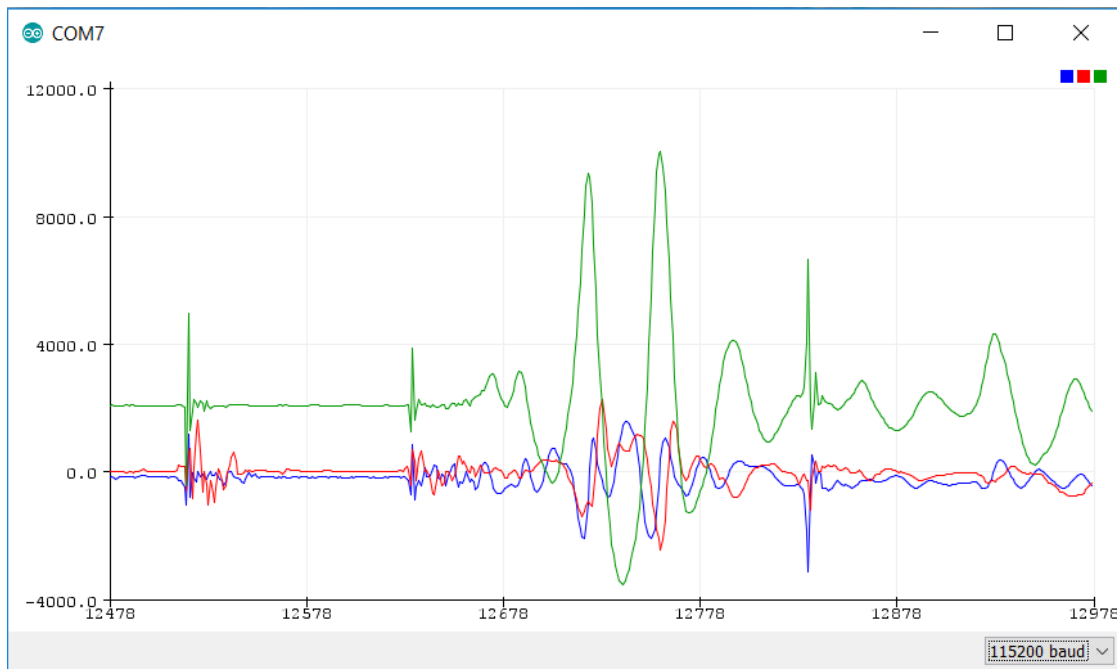


Fig. 14: Accelerometer sensor data on Arduino Serial Plotter

8 Media Transfer Protocol (MTP) firmware for Application Board 3.0

The external memory chip W25M02/W25N02 on APP3.0 is based on NAND flash.

FAT filesystem on NAND flash memory results in a complicated solution which uses a lot of RAM. Moreover use of FAT without Flash Translation Layer (to save RAM) wears out NAND flash with frequent usage. Hence the choice of [FlogFS](<https://github.com/conservify/FLogFS>), a filesystem optimized for use with NAND flash.

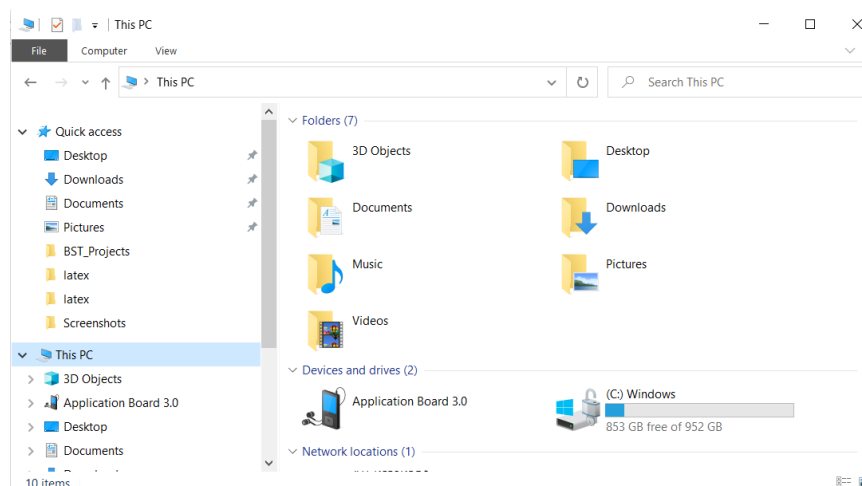
But the use of 'FlogFS', presents a new problem 'Filesystem access from PC via USB'. Use of 'FlogFS' with USB Mass Storage protocol is not possible because operating system can't recognize 'FlogFS' as a valid filesystem.

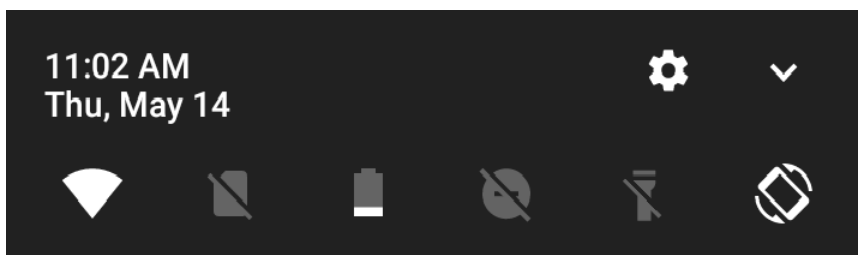
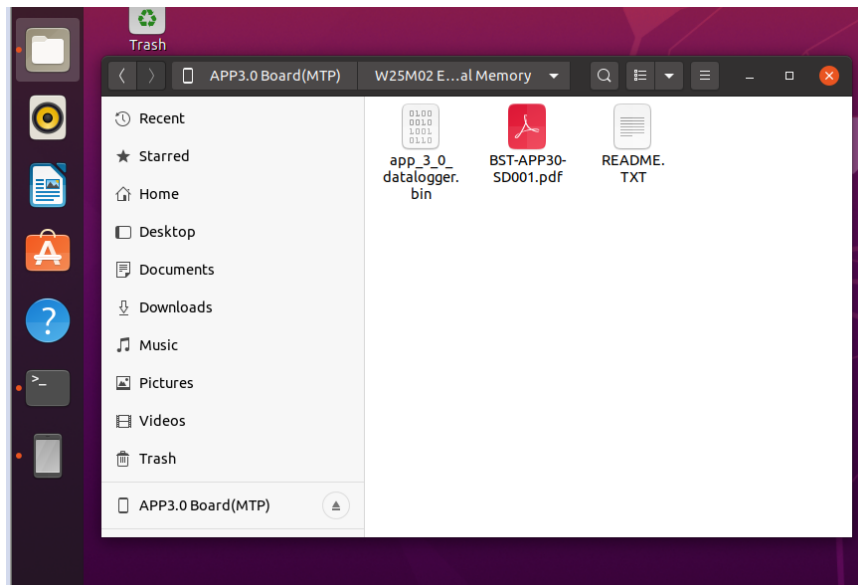
Use of custom protocol to do filesystem operations would mean re-inventing the wheel and a lot of effort. User also would not have the same experience as with USB Mass Storage.

Solution was to go with the "Media Transfer Protocol" developed initially by Microsoft for Portable Devices like MP3 players. Starting from Android Kitkat (v4.4), MTP is the only way to access files on an Android device since the whole flash memory (included user storage space) uses filesystems like ext4, YAFFS, F2FS, etc.,

Files in APP3.0 board's NAND flash memory can be viewed using the USB MTP firmware.

Supported on Windows, Linux, Android (via USB OTG) and macOS



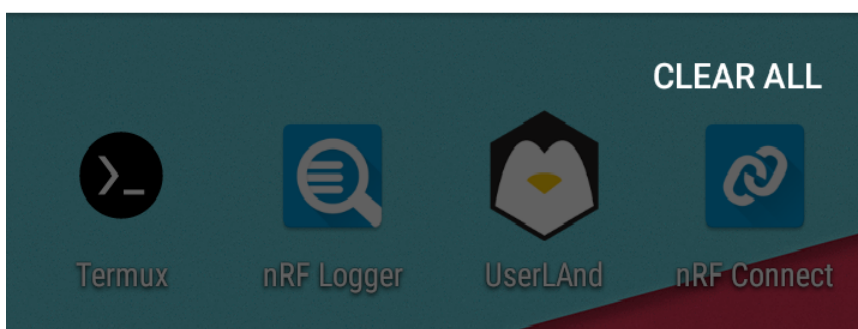


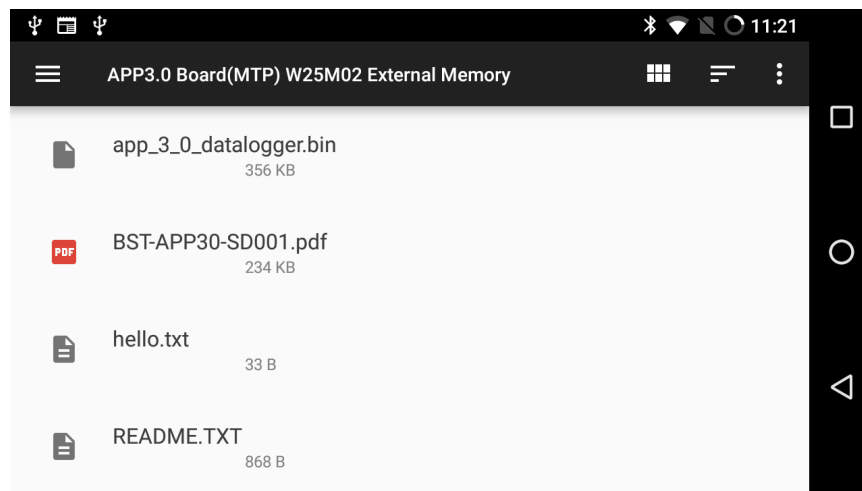
 Android System

Connected to APP3.0 Board(MTP)
Tap to view files

 MTP Host

Accessing files from APP3.0 Board(MTP)





8.1 Switching to MTP mode

- ▶ Connect the Application Board 3.0 using USB cable to PC.
- ▶ Application Board 3.0 comes with the preloaded MTP firmware update package.
- ▶ Turn OFF and turn ON the board with T1 pressed. Green LED glows on the board indicating that board switched to MTP mode.

For reference find the examples in following path COINES\v2.6.0\examples\c\file_handling and run using below command

- ▶ mingw32-make TARGET=MCU_APP30 download.

8.2 Copying the files using MTP

- ▶ Connect the Application Board 3.0 using USB cable to PC.
- ▶ Turn OFF and turn ON the board with T1 pressed.
- ▶ The device will enumerate as an MTP device with name "Application Board 3.0". Click on it and select the "W25M02 External Memory"
- ▶ The device will list all the available files and all required files can be copied.

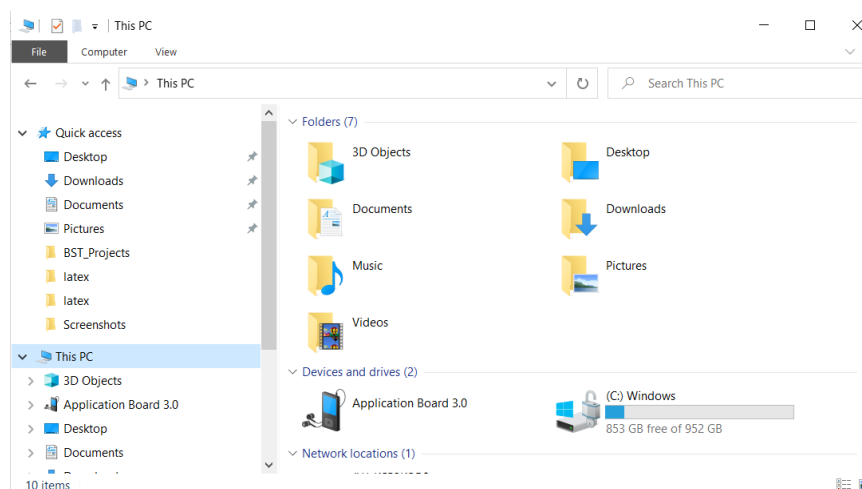


Fig. 15: Copy data log files to the PC over USB MTP

9 USB/BLE DFU bootloader

A USB/BLE Bootloader for APP3.0 Board/nRF52840 and Nicla Sense ME/nRF52832 chip complying with

- ▶ https://www.usb.org/sites/default/files/DFU_1.1.pdf
- ▶ https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v17.1.0%2Fble_sdk_app_dfu_bootloader.html&cp=9_1_4_2_3

APP3.0 Board bootloader can be found in the following path COINES\v2.6.0\firmware\app3.0\bootloader_update and Nicla Sense ME Board bootloader can be found in the following path COINES\v2.8.8\firmware\nicla\bootloader_update

9.1 Key Features

9.1.1 USB DFU

- ▶ Code download to RAM or FLASH
- ▶ Code read back (upload) from RAM or FLASH (Useful for taking firmware backups)
- ▶ Works with Windows, Linux, macOS and Android.

9.1.2 BLE DFU

- ▶ Code download to FLASH.
- ▶ Works with PC and mobile devices with iOS/Android.

Bootloader was written taking into account the following aspects

- ▶ Usability.
 - i. No special driver installation or admin rights should be required.
 - ii. The update process should be straight forward.
- ▶ Maintainability
 - i. Open source community takes care of PC side tools. For eg: dfu-util is a cross platform tool.
 - ii. Use Google Chrome's WebUSB to update firmware. Sample implementation <https://devanlai.github.io/webdfu/dfu-util/>
- ▶ Size
- ▶ COINES on MCU.

9.2 Invoking the Bootloader

APP3.0 Board

- ▶ Hardware.
 - i. Turn OFF and ON the board with T2 pressed, blue LED glows indicating that the board switched to bootloader mode.
- ▶ Software
 - i. Write 0x4E494F43 ('N','I','O','C') to MAGIC_LOCATION (0x2003FFF4)
 - ii. Write 0x0 or 0xF0000 to APP_START_ADDR (0x2003FFF8)

- iii. Call NVIC_SystemReset()
- iv. Invoke Bootloader from Software

```
#define MAGIC_LOCATION (0x2003FFF4)
#define APP_START_ADDR (*(uint32_t *) (MAGIC_LOCATION+4)

*((uint32_t *)MAGIC_LOCATION) == 0x4E494F43;
APP_START_ADDR = 0xF00000;
//APP_START_ADDR = 0x0;
NVIC_SystemReset();
```

- v. The same feature can also be used to perform application switch (2 or more applications can reside in the same flash memory at different address locations).Just write the application start address to APP_START_ADDR instead of bootloader address

Nicla Sense ME Board

► Hardware.

- i. Press three times reset button, blue LED glows indicating that the board switched to bootloader mode.

► Software

- i. Write 0x4E494F43 ('N','I','O','C') to MAGIC_LOCATION (0x2000F804)
- ii. Call NVIC_SystemReset()
- iii. Invoke Bootloader from Software

```
#define MAGIC_LOCATION (0x2000F804)
#define APP_START_ADDR (*(uint32_t *) (MAGIC_LOCATION+4)

*((uint32_t *)MAGIC_LOCATION) == 0x544F4F42;
NVIC_SystemReset();
```

- iv. The same feature can also be used to perform application switch (2 or more applications can reside in the same flash memory at different address locations).Just write the application start address to APP_START_ADDR instead of bootloader address

9.3 Using the Bootloader via USB

Write firmware to Flash memory using following command

► dfu-util -a FLASH -D fw.bin -R

Write firmware to RAM memory using following command

► dfu-util -a RAM -D fw.bin -R

Read firmware from Flash memory using following command

► dfu-util -a FLASH -U fw_bkup.bin

Read firmware from RAM memory using following command

► dfu-util -a RAM -U fw_bkup.bin

Read device serial number/ BLE MAC address

► dfu-util -l

Note: Not applicable for Nicla Sense ME board

9.4 Using the Bootloader via BLE

► PC (Windows/Linux and macOS)

Python script present in following path COINES\v2.6.0\tools\app30-ble-dfu can use the binary file directly.

i. Scan for devices to find BLE MAC address using below command

► `python app30-ble-dfu.py -l`

ii. Update firmware by using MAC address obtained in the previous step and firmware BIN file

► `python app30-ble-dfu.py -d D7:A3:CE:8E:36:14 -f firmware.bin`

► Android devices

i. Generate ZIP package using <https://pypi.org/project/adafruit-nrfutil/> before using nRF ToolBox for BLE or nRF connect for mobile.

► `adafruit-nrfutil dfu genpkg -dev-type 0x0052 -application firmware.bin dfu-package.zip`

Note: Not applicable for Nicla Sense ME board

10 Updating Bootloader, DD firmware and MTP firmware using COINES

10.1 Updating bootloader

App3.0 Board

- ▶ Connect the Application Board 3.0 using USB cable to PC.
- ▶ Application Board 3.0 comes preloaded bootloader update package.
- ▶ To update the bootloader run "update_bootloader.bat" file present in the following path COINES\ v2.6.0\firmware\app3.0\bootloader_update
- ▶ To go to bootloader mode turn OFF and ON the board with T2 pressed, blue LED glows indicating that the board switched to bootloader mode.

Nicla Sense ME Board

- ▶ Connect the Nicla Sense ME board using USB cable to PC.
- ▶ To prepare the board run "prepare_nicla.bat" file present in the following path COINES\v2.8.8\ firmware\nicla
- ▶ To update the bootloader run "update_bootloader.bat" file present in the following path COINES\ v2.8.8\firmware\nicla\bootloader_update
- ▶ To go to bootloader mode press three times reset button, blue LED glows indicating that the board switched to bootloader mode.

10.2 Updating DD firmware

- ▶ Connect the Application Board 3.0 using USB cable to PC.
- ▶ Make sure bootloader is flashed into Application board 3.0
- ▶ To update the DD firmware run "update_dd_fw.bat" file present in the following path COINES\v2.6.0\firmware\app3.0

10.3 Updating MTP firmware

- ▶ Connect the Application Board 3.0 using USB cable to PC.
- ▶ Make sure bootloader is flashed into Application board 3.0
- ▶ To update the MTP firmware run "update_mtp_fw.bat" file present in the following path C:\COINES\v2.6.0\firmware\app3.0\mtp_fw_update
- ▶ To switch to the MTP mode, turn OFF and ON the board with T1 pressed, green LED glows indicating that the board switched to MTP mode.

10.4 Updating Coines Bridge firmware on Nicla Sense ME Board

- ▶ Connect the Nicla Sense ME board using USB cable to PC.
- ▶ Make sure "prepare_nicla.bat" file is executed to flash softdevice and bootloader into Nicla Sense ME board.
- ▶ To update the Coines Bridge firmware run "update_coines_bridge_flash_fw.bat" file present in the following path C:\COINES\v2.8.8\firmware\nicla\coines_bridge
- ▶ To switch to the App mode from bootloader mode, press three times reset button.

11 Accessing the Application Board using Python

11.1 Introduction to coinespy library

The coinespy library allows users to access the Bosch Sensortec Application Board using Python.

- ▶ Control VDD and VDDIO of sensor
- ▶ Configure SPI and I²C bus parameters
- ▶ Read and write into registers of sensors from Bosch Sensortec via SPI and I²C
- ▶ Read and write digital pins of the Application Board

11.2 Installation

The coinespy module can be installed using pip:

```
pip install coinespy
```

Linux users may have to use the below commands due to co-existence of Python 2.7 and Python 3.x

```
pip3 install coinespy
python3 -m pip install coinespy
```

The module can be found on <https://pypi.org/project/coinespy/> and also in the COINES installation folder, precisely in the subfolder coines-api\pc\python, in which a python wheel package is placed.

It is highly recommended that the user is testing the following script (can be found as examples\python\coinespy_test.py in the COINES installation) to check if the installation was successful:

```
import coinespy as cpy
from coinespy import ErrorCodes

COM_INTF = cpy.CommInterface.USB

if __name__ == "__main__":
    board = cpy.CoinesBoard()
    print('coinespy version - %s' % cpy.__version__)
    board.open_comm_interface(COM_INTF)
    if board.error_code != ErrorCodes.COINES_SUCCESS:
        print(f'Could not connect to board: {board.error_code}')
    else:
        b_info = board.get_board_info()
        print(f"coines lib version: {board.lib_version}")
        print(f'BoardInfo: HW/SW ID: {hex(b_info.HardwareId)}/{hex(b_info.SoftwareId)}')
        board.close_comm_interface()
```

11.3 coinespy API description

As coinespy is only a wrapper on top of coinesAPI, the following API documentation is limited to the wrapper only. Details about meaning of variables and functionality can be found in the corresponding coinesAPI documentation in the chapter above. The following function calls are

defined within the class `coinsBoard`. Thus in order to access the functions, the user has to create an object of that class first.

```
import coinespy as cpy
coinsboard = cpy.CoinesBoard()
```

11.3.1 coinespy API calls: Interface and board information

11.3.1.1 open_comm_interface

Sets the communication interface between board and PC to USB, Serial or BLE.

```
coinsboard.open_comm_interface(interface=CommInterface.USB, arg=None) -> ErrorCodes
```

For the definition of `CommInterface`, refer to [11.3.6.3](#).

11.3.1.2 close_comm_interface

Disposes the resources used by the USB/serial/BLE communication.

```
coinsboard.close_comm_interface(arg=None) -> ErrorCodes
```

11.3.1.3 get_board_info

Obtains board specific information.

```
BoardInfo = coinsboard.get_board_info()
```

Return:

```
BoardInfo.HardwareId # Hardware ID
BoardInfo.SoftwareId # Firmware version information
BoardInfo.Board      # Board type
BoardInfo.ShuttleID  # ID of shuttle, in case a shuttle is detected
```

11.3.1.4 scan_ble_devices

This API is used to connect to BLE Adapter and return list of BLE peripherals found during BLE scan.

```
ble_info, peripheral_count = coinsboard.scan_ble_devices(scan_timeout_ms=0) ->
    Tuple[list, int]
```

For the definition of parameters, refer to [6.8.29](#).

11.3.1.5 echo_test

This API is used to test the communication.

```
coinsboard.echo_test(data: List[int]) -> ErrorCodes
```

Arguments:

- data: Data to be sent for testing.

11.3.2 coinespy API calls: GPIO oriented calls

11.3.2.1 set_pin_config

Configures the state, level and direction of a GPIO pin

```
coinesboard.set_pin_config(pin_number: MultiIOPin, direction: PinDirection,  
    output_state: PinValue) -> ErrorCodes
```

For the definition of MultiIOPin, refer to [11.3.6.8](#). For the definition of PinDirection, refer to [11.3.6.1](#). For PinValue, refer to [11.3.6.2](#).

11.3.2.2 get_pin_config

Obtains information regarding the Pin's state, level and direction.

```
PinConfigInfo = coinesboard.get_pin_config(pin_number: MultiIOPin)
```

Return:

```
PinConfigInfo.direction    # 0: INPUT, 1: OUTPUT  
PinConfigInfo.switch_state # 0: OFF, 1: ON  
PinConfigInfo.level        # 1: HIGH, 0: LOW
```

11.3.2.3 set_shuttleboard_vdd_vddio_config

Set the VDD and VDDIO voltage level.

```
coinesboard.set_shuttleboard_vdd_vddio_config(vdd_val: float = None, vddio_val: float =  
    None) -> ErrorCodes
```

```
# Example: coinesboard.set_shuttleboard_vdd_vddio_config(3.3, 3.3)
```

11.3.2.4 set_vdd

Set the VDD voltage level.

```
coinesboard.set_vdd(vdd_val: float = None) -> ErrorCodes
```

```
# Example: coinesboard.set_vdd(3.3)
```

11.3.2.5 set_vddio

Set the VDDIO voltage level.

```
coinesboard.set_vddio(vdd_val: float = None) -> ErrorCodes
```

```
# Example: coinesboard.set_vddio(3.3)
```

11.3.3 coinespy API calls: Sensor communication

For the definition of SPIBus, refer to [11.3.6.11](#). For the definition of I2CBus, refer to [11.3.6.10](#).

11.3.3.1 config_i2c_bus

Configures the I²C bus.

```
coinesboard.config_i2c_bus(bus: I2CBus, i2c_address: int, i2c_mode: I2CMode) ->
    ErrorCodes
```

For the definition of I2CMode, refer to [11.3.6.4](#).

11.3.3.2 config_spi_bus

Configures the SPI bus of the board.

```
coinesboard.config_spi_bus(bus: SPIBus, cs_pin: MultiIOPin, spi_speed=SPISpeed,
    spi_mode=SPIMode) -> ErrorCodes
```

For the definition of MultiIOPin, refer to [11.3.6.8](#). For the definition of SPISpeed, refer to [11.3.6.5](#).
For the definition of SPIMode, refer to [11.3.6.7](#).

11.3.3.3 deconfig_i2c_bus

This API is used to de-configure the I²C bus

```
coinesboard.deconfig_i2c_bus(bus: I2CBus) -> ErrorCodes
```

11.3.3.4 deconfig_spi_bus

This API is used to de-configure the SPI bus

```
coinesboard.deconfig_spi_bus(bus: SPIBus) -> ErrorCodes
```

11.3.3.5 write_i2c

Writes 8-bit register data to the I²C

```
coinesboard.write_i2c(bus: I2CBus, register_address: int, register_value: int,
    sensor_interface_detail: int = None) -> ErrorCodes
```

For the definition of parameters, refer to [6.6.7](#).

11.3.3.6 read_i2c

Reads 8-bit register data from the I²C

```
register_data = coinesboard.read_i2c(bus: I2Cbus, register_address: int,  
    number_of_reads=1, sensor_interface_detail: int = None)
```

For the definition of parameters, refer to [6.6.8](#).

11.3.3.7 write_spi

Writes 8-bit register data to the SPI device

```
coinesboard.write_spi(bus: SPIbus, register_address: int, register_value: int,  
    sensor_interface_detail: int = None) -> ErrorCodes
```

For the definition of parameters, refer to [6.6.11](#).

11.3.3.8 read_spi

Reads 8-bit register data from the SPI device.

```
register_data = coinesboard.read_spi(bus: SPIbus, register_address: int,  
    number_of_reads=1, sensor_interface_detail: int = None)
```

For the definition of parameters, refer to [6.6.12](#).

11.3.3.9 config_word_spi_bus

Configures the SPI bus parameters.

```
coinesboard.config_word_spi_bus(bus: SPIbus, cs_pin: MultiIOPin,  
    spi_speed=SPISpeed.SPI_1_MHZ, spi_mode=SPIMode.MODE0,  
    spi_bits=SPITransferBits.SPI16BIT) -> ErrorCodes
```

For the definition of MultiIOPin, refer to [11.3.6.8](#). For the definition of SPISpeed, refer to [11.3.6.5](#).
For the definition of SPITransferBits, refer to [11.3.6.6](#).

11.3.3.10 write_16bit_spi

Writes 16-bit register data to the SPI device.

```
coinesboard.write_16bit_spi(bus: SPIbus, register_address: int, register_value:  
    List[int], sensor_interface_detail: int = None) -> ErrorCodes
```

For the definition of parameters, refer to [6.6.14](#).

11.3.3.11 read_16bit_spi

Reads 16-bit register data from the SPI device.

```
register_data = coinesboard.read_16bit_spi(bus: SPIBus, register_address: int,
    number_of_reads=2, sensor_interface_detail: int = None)
```

For the definition of parameters, refer to [6.6.15](#).

11.3.3.12 delay_milli_sec

Introduces delay in millisecond.

```
coinesboard.delay_milli_sec(time_in_milli_sec=100)
```

11.3.3.13 delay_micro_sec

Introduces delay in microsecond.

```
coinesboard.delay_micro_sec(time_in_micro_sec=1)
```

11.3.4 coinespy API calls: Streaming feature

11.3.4.1 config_streaming

Sets the configuration for streaming sensor data.

```
coinesboard.config_streaming(sensor_id: int,
    stream_config: StreamingConfig, data_blocks: StreamingBlocks) -> ErrorCodes
```

Arguments:

- ▶ **sensor_id**: An integer number that can be used as identifier/index to the sensor data that will be streamed for this setting
- ▶ **stream_config**: Contains information regarding interface settings and streaming configuration.
- ▶ **data_blocks**: Contains information regarding numbers of blocks to read, register address and size for each block.

Note:

The below parameters should always be set:

- ▶ **data_blocks.NoOfBlocks**: number of blocks to stream (must at least be one)
- ▶ For each block **b**:
 - ▶ **data_blocks.RegStartAddr[b]**: start address of the block in the register map
 - ▶ **data_blocks.NoOfDataBytes[b]**: number of bytes to read, starting from the start address

For reading data from I²C bus, then set the below parameters:

- ▶ **stream_config.Intf** = **cpy.SensorInterface.I2C.value**
- ▶ **stream_config.I2CBus**: I²C bus (in case of APP2.0 and APP3.0, this is always **cpy.I2CBus.BUS_I2C_0.value**)

- ▶ `stream_config.DevAddr`: I²C address of the sensor

For reading data from SPI bus, then set the below parameters:

- ▶ `stream_config.Intf` = `cpy.SensorInterface.SPI.value`;
- ▶ `stream_config.SPIBus`: SPI bus (in case of APP2.0 and APP3.0, this is always `cpy.SPIBus.BUS_SPI_0.value`)
- ▶ `stream_config.CSPin`: CS pin of the sensor, information can be obtained from the shuttle board documentation for the sensor.
- ▶ `stream_config.SPIType`: 0 : 8-bit SPI; 1 : 16-bit SPI

When polling mode is requested, set the below parameters:

- ▶ `stream_config.SamplingUnits`: either milliseconds or microseconds. Refer to [11.3.6.15](#).
- ▶ `stream_config.SamplingTime`: sampling period in the unit as defined in `stream_config.SamplingUnits`

When interrupt mode is requested, set the below parameters:

- ▶ `stream_config.IntPin`: pin of the interrupt which shall trigger the sensor read-out. If the interrupt output of the sensor is used, the required information about the pin number can be obtained from the shuttle board documentation for the sensor.
- ▶ `stream_config.IntTimeStamp`: it can be configured if the sensor data is tagged with a timestamp - 1 or not - 0.
- ▶ `stream_config.HwPinState`: State of the hardware pin connected to the interrupt line - 0/1 : Low/high

Below parameters are common for both streaming types:

- ▶ `stream_config.IntlineCount`: Number of interrupt lines to be used for monitoring interrupts.
- ▶ `stream_config.IntlineInfo`: List of pin numbers that correspond to interrupt lines being used for interrupt monitoring.
- ▶ `stream_config.ClearOnWrite`: 0/1 : Disable/enable "clear on write" feature

The below parameters should be set only when `stream_config.ClearOnWrite` = 1:

- ▶ `stream_config.ClearOnWriteConfig.StartAddress`: Address of the sensor register at which the process of clearOnWrite should initiate.
- ▶ `stream_config.ClearOnWriteConfig.DummyByte`: Number of padding bytes that must be added before clearing the bytes starting from the designated address.
- ▶ `stream_config.ClearOnWriteConfig.NumBytesToClear`: Number of bytes that need to be cleared.

Below is the Python code snippet for interrupt streaming

```
# Store streaming settings in local variables
accel_stream_settings = dict(
    I2C_ADDR_PRIMARY=0x18,
    NO_OF_BLOCKS = 2,
    REG_X_LSB= [0x12, 0x00],
    NO_OF_DATA_BYTES= [6, 1],
    CHANNEL_ID=1,
```

```

        CS_PIN=cpy.MultiIOPin.SHUTTLE_PIN_8.value,
        INT_PIN=cpy.MultiIOPin.SHUTTLE_PIN_21.value,
        INT_TIME_STAMP=1,
    )
    gyro_stream_settings = dict(
        I2C_ADDR_PRIMARY=0x68,
        NO_OF_BLOCKS = 2,
        REG_X_LSB= [0x02,0x00],
        NO_OF_DATA_BYTES = [6, 1],
        CHANNEL_ID=2,
        CS_PIN=cpy.MultiIOPin.SHUTTLE_PIN_14.value,
        INT_PIN=cpy.MultiIOPin.SHUTTLE_PIN_22.value,
        INT_TIME_STAMP=1,
    )

    # set the config_streaming parameters
    stream_config = cpy.StreamingConfig()
    data_blocks = cpy.StreamingBlocks()
    if self.interface == cpy.SensorInterface.I2C:
        stream_config.Intf = cpy.SensorInterface.I2C.value
        stream_config.I2CBus = cpy.I2CBus.BUS_I2C_0.value
        stream_config.DevAddr = sensor["I2C_ADDR_PRIMARY"]

    elif self.interface == cpy.SensorInterface.SPI:
        stream_config.Intf = cpy.SensorInterface.SPI.value
        stream_config.SPIBus = cpy.SPIBus.BUS_SPI_0.value
        stream_config.CSPin = sensor["CS_PIN"]

    if sensor_type == bmi08x.SensorType.ACCEL and self.interface == cpy.SensorInterface.SPI:
        # extra dummy byte for SPI
        dummy_byte_offset = 1
    else:
        dummy_byte_offset = 0

    data_blocks.NoOfBlocks = sensor["NO_OF_BLOCKS"]
    for i in range(0, data_blocks.NoOfBlocks):
        data_blocks.RegStartAddr[i] = sensor["REG_X_LSB"][i]
        data_blocks.NoOfDataBytes[i] = sensor["NO_OF_DATA_BYTES"][i] + dummy_byte_offset

    stream_config.IntTimeStamp = sensor["INT_TIME_STAMP"]
    stream_config.IntPin = sensor["INT_PIN"]

    # call config_streaming API for each sensor to configure the streaming settings
    ret = coinesboard.config_streaming(
        accel_sensor_id, self.accel_stream_config, self.accel_data_blocks)
    ret = coinesboard.config_streaming(
        gyro_sensor_id, self.accel_stream_config, self.accel_data_blocks)

```

11.3.4.2 start_stop_streaming

Starts or stops sensor data streaming.

```

coinesboard.start_stop_streaming(stream_mode: StreamingMode, start_stop:
    StreamingState) -> ErrorCodes

```

For the definition of StreamingMode, refer to [11.3.6.13](#). For the definition of StreamingState, refer

to [11.3.6.14](#).

11.3.4.3 read_stream_sensor_data

Reads the data streamed from the sensor.

```
coinesboard.read_stream_sensor_data(sensor_id: int, number_of_samples: int,
    buffer_size=STREAM_RSP_BUF_SIZE) -> Tuple[ErrorCodes, list, int]
```

Return:

Tuple of ErrorCodes, data and valid_samples_count

For the detailed definition of parameters, refer to [6.7.3](#).

11.3.5 coinespy API calls: Other useful APIs

11.3.5.1 flush_interface

Flush the write buffer.

```
coinesboard.flush_interface()
```

11.3.5.2 soft_reset

Resets the device.

```
coinesboard.soft_reset()
```

11.3.6 Definiton of constants

11.3.6.1 PinDirection

Pin mode definitions

```
class PinDirection:
    INPUT = 0 # COINES_PIN_DIRECTION_IN = 0
    OUTPUT = 1
```

11.3.6.2 PinValue

Pin level definitions

```
class PinValue:
    LOW = 0 # COINES_PIN_VALUE_LOW = 0
    HIGH = 1
```

11.3.6.3 CommInterface

Definition of Communication interface

```
class CommInterface:
```

```
    USB = 0
    SERIAL = 1
    BLE = 2
```

11.3.6.4 I2CMode

Definition of the speed of I2C bus.

```
class I2CMode:
```

```
    STANDARD_MODE = 0 # Standard mode - 100kHz
    FAST_MODE = 1 # Fast mode - 400kHz
    SPEED_3_4_MHZ = 2 # High Speed mode - 3.4 MHz
    SPEED_1_7_MHZ = 3 # High Speed mode 2 - 1.7 MHz
```

11.3.6.5 SPISpeed

Definition of the speed of SPI bus.

```
class SPISpeed:
```

```
    SPI_10_MHZ = 6
    SPI_7_5_MHZ = 8
    SPI_6_MHZ = 10
    SPI_5_MHZ = 12
    SPI_3_75_MHZ = 16
    SPI_3_MHZ = 20
    SPI_2_5_MHZ = 24
    SPI_2_MHZ = 30
    SPI_1_5_MHZ = 40
    SPI_1_25_MHZ = 48
    SPI_1_2_MHZ = 50
    SPI_1_MHZ = 60
    SPI_750_KHZ = 80
    SPI_600_KHZ = 100
    SPI_500_KHZ = 120
    SPI_400_KHZ = 150
    SPI_300_KHZ = 200
    SPI_250_KHZ = 240
```

11.3.6.6 SPITransferBits

Definition of the SPI bits.

```
class SPITransferBits:
```

```
    SPI8BIT = 8 # 8 bit register read/write
    SPI16BIT = 16 # 16 bit register read/write
```

11.3.6.7 SPIMode

Definition of the SPI mode.

```
class SPIMode:
    MODE0 = 0x00 # SPI Mode 0: CPOL=0; CPHA=0
    MODE1 = 0x01 # SPI Mode 1: CPOL=0; CPHA=1
    MODE2 = 0x02 # SPI Mode 2: CPOL=1; CPHA=0
    MODE3 = 0x03 # SPI Mode 3: CPOL=1; CPHA=1
```

11.3.6.8 MultiOPin

Definition of the shuttle board pin(s)

```
class MultiOPin(Enum):
    SHUTTLE_PIN_7 = 0x09 # CS pin
    SHUTTLE_PIN_8 = 0x05 # Multi-IO 5
    SHUTTLE_PIN_9 = 0x00 # Multi-IO 0
    SHUTTLE_PIN_14 = 0x01 # Multi-IO 1
    SHUTTLE_PIN_15 = 0x02 # Multi-IO 2
    SHUTTLE_PIN_16 = 0x03 # Multi-IO 3
    SHUTTLE_PIN_19 = 0x08 # Multi-IO 8
    SHUTTLE_PIN_20 = 0x06 # Multi-IO 6
    SHUTTLE_PIN_21 = 0x07 # Multi-IO 7
    SHUTTLE_PIN_22 = 0x04 # Multi-IO 4
    SHUTTLE_PIN_SDO = 0x1F

    # APP3.0 pins
    MINI_SHUTTLE_PIN_1_4 = 0x10 # GPIO0
    MINI_SHUTTLE_PIN_1_5 = 0x11 # GPIO1
    MINI_SHUTTLE_PIN_1_6 = 0x12 # GPIO2/INT1
    MINI_SHUTTLE_PIN_1_7 = 0x13 # GPIO3/INT2
    MINI_SHUTTLE_PIN_2_5 = 0x14 # GPIO4
    MINI_SHUTTLE_PIN_2_6 = 0x15 # GPIO5
    MINI_SHUTTLE_PIN_2_1 = 0x16 # CS
    MINI_SHUTTLE_PIN_2_3 = 0x17 # SDO
    MINI_SHUTTLE_PIN_2_7 = 0x1D # GPIO6
    MINI_SHUTTLE_PIN_2_8 = 0x1E # GPIO7
```

11.3.6.9 SensorInterface

To define Sensor interface.

```
class SensorInterface(Enum):
    SPI = 0
    I2C = 1
```

11.3.6.10 I2CBus

Used to define the I2C type.

```
class I2CBus(Enum):
    BUS_I2C_0 = 0
```

```
BUS_I2C_1 = 1
BUS_I2C_MAX = 2
```

11.3.6.11 SPIBus

Used to define the SPI type.

```
class SPIBus(Enum):
    BUS_SPI_0 = 0
    BUS_SPI_1 = 1
    BUS_SPI_MAX = 2
```

11.3.6.12 PinInterruptMode

Defines Pin interrupt modes.

```
class PinInterruptMode(Enum):
    # Trigger interrupt on pin state change
    PIN_INTERRUPT_CHANGE = 0
    # Trigger interrupt when pin changes from low to high
    PIN_INTERRUPT_RISING_EDGE = 1
    # Trigger interrupt when pin changes from high to low
    PIN_INTERRUPT_FALLING_EDGE = 2
    PIN_INTERRUPT_MODE_MAXIMUM = 4
```

11.3.6.13 StreamingMode

Streaming mode definitions

```
class StreamingMode:
    STREAMING_MODE_POLLING = 0 # Polling mode streaming
    STREAMING_MODE_INTERRUPT = 1 # Interrupt mode streaming
```

11.3.6.14 StreamingState

Streaming state definitions

```
class StreamingState:
    STREAMING_START = 1
    STREAMING_STOP = 0
```

11.3.6.15 SamplingUnits

Sampling Unit definitions

```
class SamplingUnits:
    SAMPLING_TIME_IN_MICRO_SEC = 0x01 # sampling unit in micro second
    SAMPLING_TIME_IN_MILLI_SEC = 0x02 # sampling unit in milli second
```

11.3.7 Error Codes

Error codes are not (always) returned by the different function calls. Internally, a `error_code` variable is maintained which is updated after the function call. It can be read out and checked by the user afterwards. Example:

```
import coinespy as cpy
board = cpy.CoinesBoard()
try:
    board.open_comm_interface(cpy.CommInterface.USB)
    board.close_comm_interface()
except:
    print(f'Could not connect to board: {board.error_code}')
    exit(board.error_code)
```

11.3.7.1 General Error Codes

Error code definitions

```
class ErrorCodes(Enum):
    COINES_SUCCESS = 0
    COINES_E_FAILURE = -1
    COINES_E_COMM_IO_ERROR = -2
    COINES_E_COMM_INIT_FAILED = -3
    COINES_E_UNABLE_OPEN_DEVICE = -4
    COINES_E_DEVICE_NOT_FOUND = -5
    COINES_E_UNABLE_CLAIM_INTERFACE = -6
    COINES_E_MEMORY_ALLOCATION = -7
    COINES_E_NOT_SUPPORTED = -8
    COINES_E_NULL_PTR = -9
    COINES_E_COMM_WRONG_RESPONSE = -10
    COINES_E_SPI16BIT_NOT_CONFIGURED = -11
    COINES_E_SPI_INVALID_BUS_INTERFACE = -12
    COINES_E_SPI_CONFIG_EXIST = -13
    COINES_E_SPI_BUS_NOT_ENABLED = -14
    COINES_E_SPI_CONFIG_FAILED = -15
    COINES_E_I2C_INVALID_BUS_INTERFACE = -16
    COINES_E_I2C_BUS_NOT_ENABLED = -17
    COINES_E_I2C_CONFIG_FAILED = -18
    COINES_E_I2C_CONFIG_EXIST = -19
    COINES_E_TIMER_INIT_FAILED = -20
    COINES_E_TIMER_INVALID_INSTANCE = -21
    COINES_E_TIMER_CC_CHANNEL_NOT_AVAILABLE = -22
    COINES_E_EEPROM_RESET_FAILED = -23
    COINES_E_EEPROM_READ_FAILED = -24
    COINES_E_INIT_FAILED = -25
    COINES_E_STREAM_NOT_CONFIGURED = -26
    COINES_E_STREAM_INVALID_BLOCK_SIZE = -27
    COINES_E_STREAM_SENSOR_ALREADY_CONFIGURED = -28
    COINES_E_STREAM_CONFIG_MEMORY_FULL = -29
    COINES_E_INVALID_PAYLOAD_LEN = -30
    COINES_E_CHANNEL_ALLOCATION_FAILED = -31
    COINES_E_CHANNEL_DE_ALLOCATION_FAILED = -32
    COINES_E_CHANNEL_ASSIGN_FAILED = -33
    COINES_E_CHANNEL_ENABLE_FAILED = -34
    COINES_E_CHANNEL_DISABLE_FAILED = -35
    COINES_E_INVALID_PIN_NUMBER = -36
```

```
COINES_E_MAX_SENSOR_COUNT_REACHED = -37  
COINES_E_EEPROM_WRITE_FAILED = -38  
COINES_E_INVALID_EEPROM_RW_LENGTH = -39
```

12 FAQ

1. I want to upgrade APP2.0/APP3.0 firmware.

- ▶ Use app20-flash tool (or) Development Desktop to upgrade APP2.0 firmware.
- ▶ Use dfu-util tool to upgrade APP3.0 firmware.

2. Why GCC is chosen as the compiler?

GCC is widely used and available in both Linux and Windows environments. However, if the user uses a different compiler, it should be easy to migrate the code, since no compiler-specific tweaks are needed.

3. Why do you use TDM-GCC in Windows?

It is a complete toolchain in a single installer, but does not come with too much overhead the COINES user most likely does not need. The installation procedures for other toolchains are more complicated and especially for in-experienced users difficult to handle.

4. Why do you use mingw32-make in Windows?

It comes as a part of TDM-GCC package and can handle Windows path names better compared e.g. with MSYS make. The usage of spaces in path names can be overcome using 8.3 naming format.

5. What to do in case of any communication or initialization failure while running examples?

Resetting or rebooting the board will help solving this

6. What does 'app_switch' tool do?

'app_switch' tool can command the Application Board to jump to a specified address on RAM or FLASH. It works only with APP2.0 firmware v3.1 or later. COINES uses this feature to jump to USB DFU Bootloader or example application.

7. Are libraries provided by microcontroller vendor used for COINES on MCU implementation ?

Yes ! ASF v3.42 (Advanced Software Framework) and nRF5 SDK v15.2 is being used for APP2.0 and APP3.0. One can download the latest version of libraries from the below links

- ▶ <https://www.microchip.com/mplab/avr-support/advanced-software-framework>
- ▶ https://developer.nordicsemi.com/nRF5_SDK/

8. How is the binary file from PC downloaded to RAM or Flash memory of MCU?

USB DFU protocol and open-source 'dfu-util' is used.

- ▶ USB DFU Specification - https://www.usb.org/sites/default/files/DFU_1.1.pdf
- ▶ dfu-util Homepage - <http://dfu-util.sourceforge.net/>

9. Why is there no output in my terminal application not stream data after cross-compiling and downloading an example on the MCU?

The code example on the MCU waits until the serial port of the board is opened. However, opening the port is not enough, the user has to ensure that also the DTR signal is set (this is required due to have higher compatibility among different terminal applications).

10. Why some examples can only be compiled for either PC or MCU target?

- ▶ Examples which make use of APIs like `coins_config_streaming`, `coins_read_stream_sensor_data` etc., are meant to work only on PC.
- ▶ Use of APIs like `coins_attach_interrupt` in example will make it only compatible with MCU.
- ▶ Constraints can also be introduced by the use of POSIX C library. Eg: Functions from `time.h`, `pthread.h`, etc.,

13 Legal disclaimer

13.1 Engineering samples

Engineering Samples are marked with an asterisk (*), (E) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

13.2 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or safety-critical systems. Safety-critical systems are those for which a malfunction is expected to lead to bodily harm, death or severe property damage. In addition, they shall not be used directly or indirectly for military purposes (including but not limited to nuclear, chemical or biological proliferation of weapons or development of missile technology), nuclear power, deep sea or space applications (including but not limited to satellite technology).

The resale and/or use of Bosch Sensortec products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.


The purchaser accepts the responsibility to monitor the market for the purchased products, particularly with regard to product safety, and to inform Bosch Sensortec without delay of all safety-critical incidents.

13.3 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

14 Document history and modifications

Rev. no.	Chapter	Description of modification/changes	Date
1.0		Initial release	Dec 2
1.1	All	Running COINES on MCU	Mar 2
1.2	All, ??	Added COINES Editer description, minor changes	Feb 2
1.3	6.6.14 , 6.6.15 11	Added 16-bit SPI functions Added Python interface description	Mar 2
1.4	All	Added info. on APP3.0 BLE support and new APIs <code>coines_get_version()</code> , <code>coines_read_intf()</code> , <code>coines_write_intf()</code> , <code>coines_intf_available()</code>	Octob
1.5	All	Added info. on new APIs	May 2
1.6	All	Added info. on new APIs, MTP and USB/BLE bootloader documentation	April 2



Bosch Sensortec GmbH
Gerhard-Kindler-Straße 9
72770 Reutlingen / Germany

www.bosch-sensortec.com

Modifications reserved | Printed in Germany

Preliminary - specifications subject to change without notice

Document number: BST-DHW-AN013

Revision 1.7