Bosch Sensortec
COINES
COmmunication for INertial and Environmental Sensors

BOSCH

User Manual

Document revision	1.3
Document release date	Mar 2020
Document number	BST-COINES-SD001
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

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

1. A sensor specific shuttle board also known as breakout board.
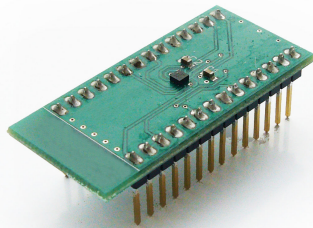
Fig. 1: APP2.0 sensor shuttle board

2. Application Board 2.0, which has a connector for the shuttle board and serves as interface translator from the sensor interface (I^2C or SPI) to a USB interface, allowing PC software to communicate with the sensor on the shuttle.
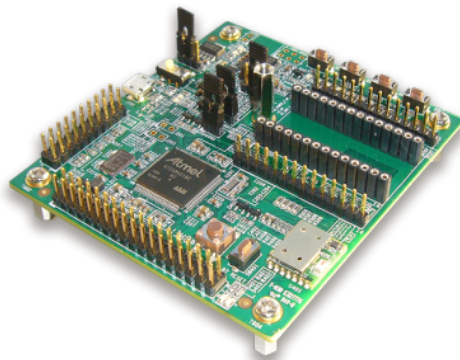
Fig. 2: Application Board 2.0

3. 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.

4

2 Accessing the sensor on Application Board using C and SensorAPI

2.1 Introduction to COINES

COINES ("**CO**mmunication with **IN**ertial and **E**nvironmental **S**ensors") 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 a an abstraction of the embedded environment on the host side. COINES library provides read and write functions for I^2C 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 I^2C 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 exected 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).
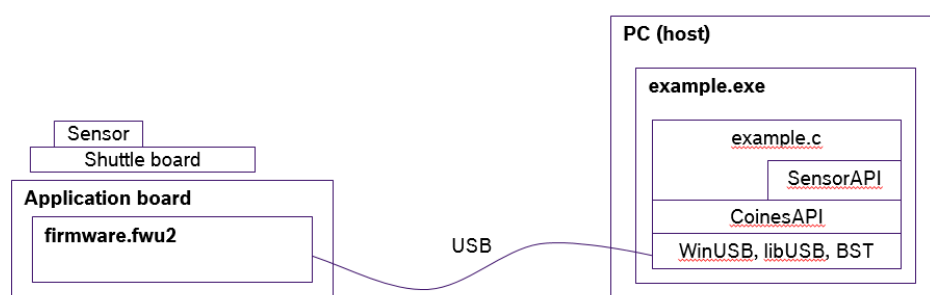
Fig. 3: Working princple: running example on PC side

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 complied 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 perfom many time-critical operations on the sensor, such as fast reading of FIFO content at high data rates.
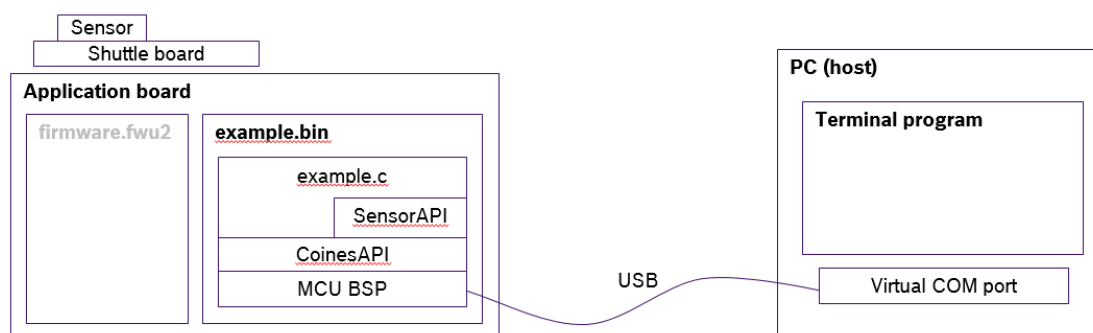
Fig. 4: Working princple: running example on the MCU of the Application board

6

3 Installation

3.1 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)

3.2 Installation (Windows)

3.2.1 Installation of COINES

1. Download the lastest 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 to view examples

3.2.2 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'
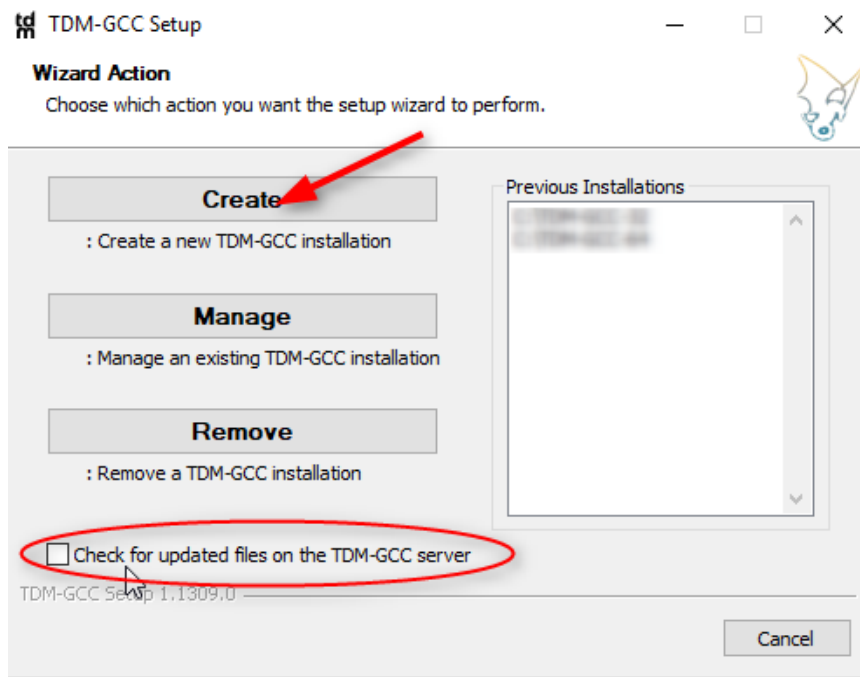
Fig. 5: TDM-GCC installation dialog
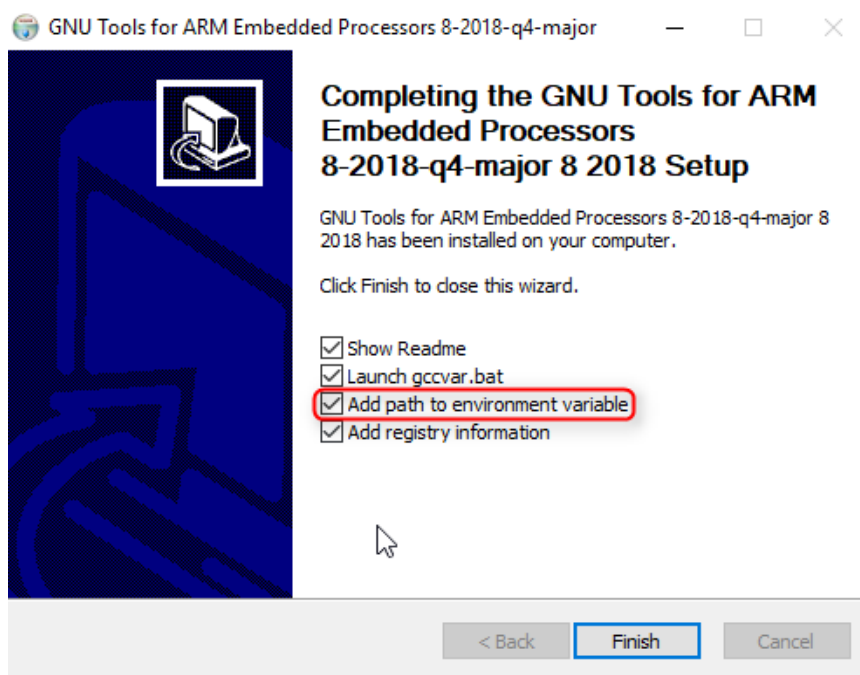
Fig. 6: GNU ARM Toolchain installation

3.3 Installation (Linux/MacOS)

3.3.1 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.)

6. Install the optional COINES Code Editor if desired.

3.3.2 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,libusbx-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`

4 Quick start guide

4.1 Compiling and executing code (COINES Code Editor)

The COINES installer is shipped with a simple graphical tool called COINES Code Editor that allows the user to write, build and execute example code for Bosch sensors. It can be found in the start menu (or start icon on the desktop) or directly in the COINES installation folder in the directory gui.

The purpose of the COINES Code Editor is to give the user, who is not familar with the command line and who does not want to invest into COINES integration into some IDE (see below chapter for integration into eclipse, for example) an easy possibility to start using the sensors and the example files.

With the COINES Code Editor, example files can be loaded, edited, compiled and the resulting executable can be executed. Compilation messages and any output from the examples are displayed in the console area (see figure 7).
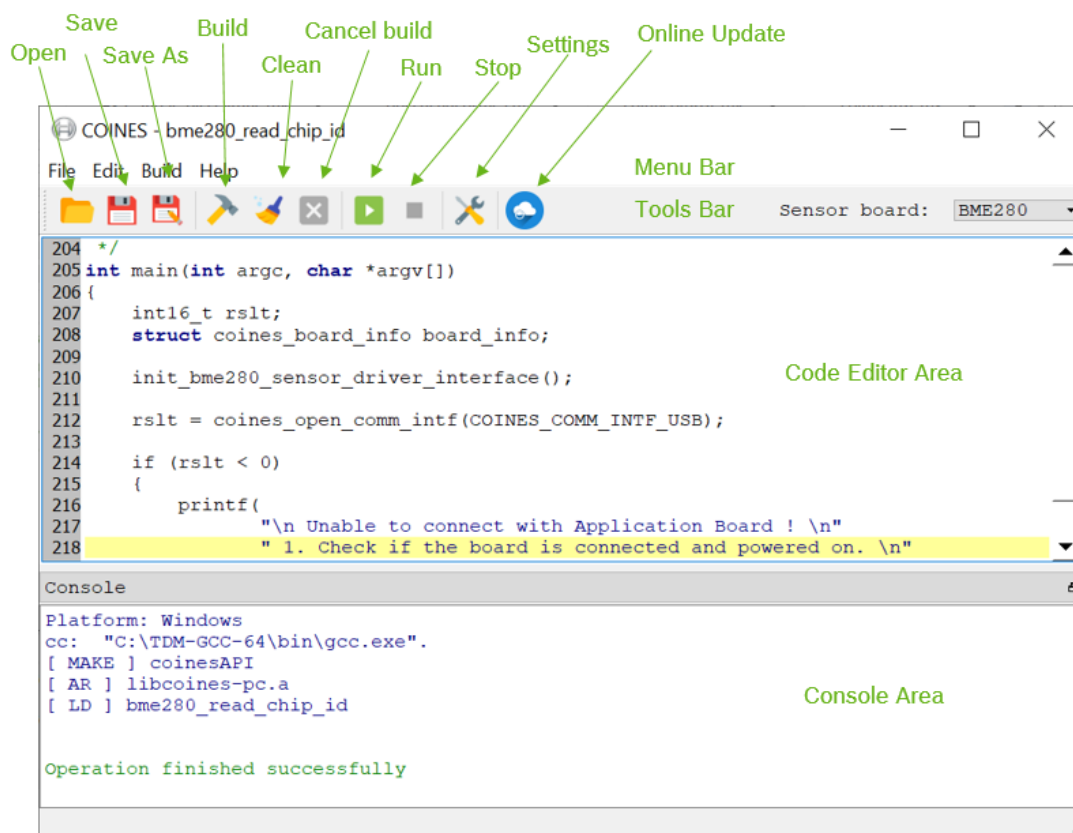
Fig. 7: COINES Code Editor

4.1.1 Using the COINES Code Editor

At start-up, the COINES Code Editor will try to load the previously edited source file. The open command will also point to the previously used work folder. In order to make any configuration adjustments, the user can access the Settings Dialog. The editor appearance can be changed slightly, and the 'make' tool can be manually selected, depending on what is installed on the machine. The path to this 'make' executable should be added to the environment path.

The build command uses the corresponding makefile found in the same folder as the source file, and the run command executes the build binary resulting from the build process. Any console output or error is displayed in the console area.

The build system supports "all" and "clean" targets, called for the corresponding makefile for the selected example.

4.1.2 Keyboard shortcuts

The following keyboard shortcuts can be used in the Code Editor:

▶ `Ctrl+O` - Open a new source file (example)

▶ `Ctrl+S` - Save currently opened file

▶ `Ctrl+Shift+S` - Save currently opened file with a new name

▶ `Ctrl+P` - Open the folder containing COINES examples

▶ `Ctrl+T` - Open Settings dialog

▶ `Ctrl+Q` - Exit Application

▶ `Ctrl+/` - Comment or uncomment the selected block of code

▶ `Ctrl+F` - Open Find feature (code search)

▶ `Ctrl+B` - Build currently opened example

▶ `Ctrl+L` - Clean currently opened example

▶ `Ctrl+Break` - Cancel build operation

▶ `Ctrl+R` - Run currently opened example (it automatically builds before running)

▶ `Ctrl+K` - Stop run operation

▶ `Ctrl+I` - Show About dialog

4.2 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.

4.3 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 evironmental 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.

11

5. Type `mingw32-make TARGET=MCU_APP20 download` . Other available options are

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

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`

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.

4.4 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
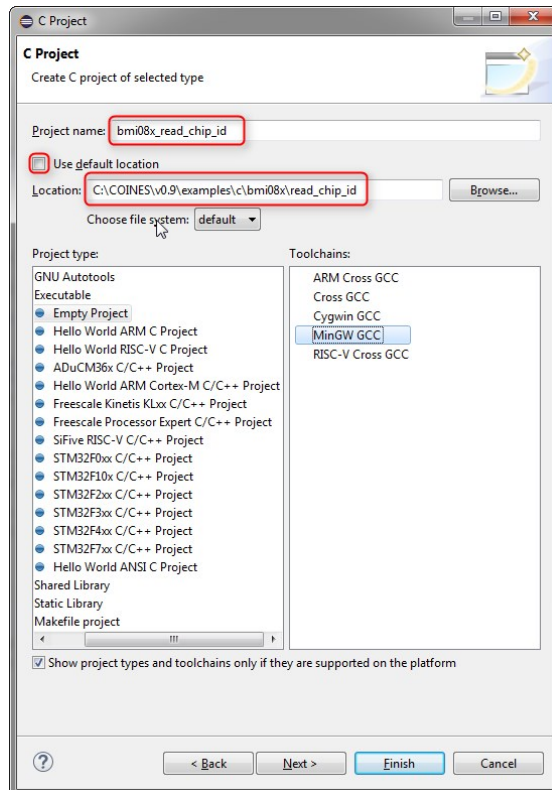
Fig. 8: Eclipse C Project for Windows
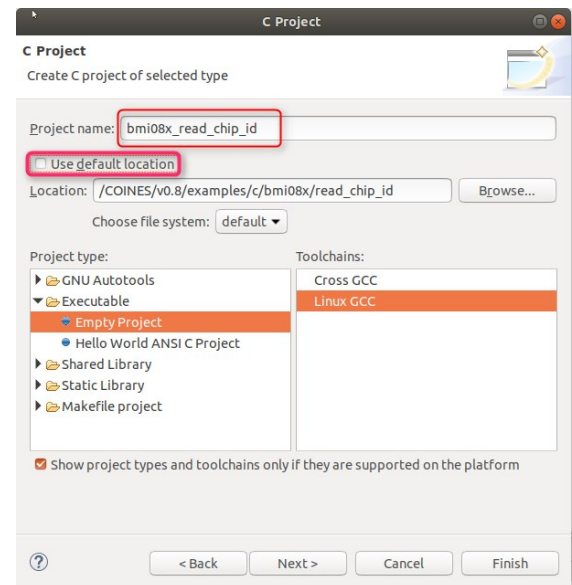
Fig. 9: Eclipse C Project for Linux

    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
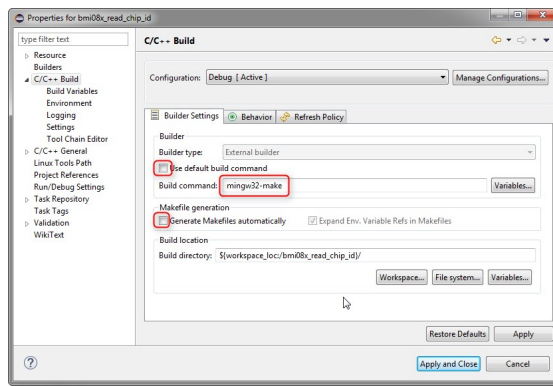
Fig. 10: Windows Eclipse Project Properties
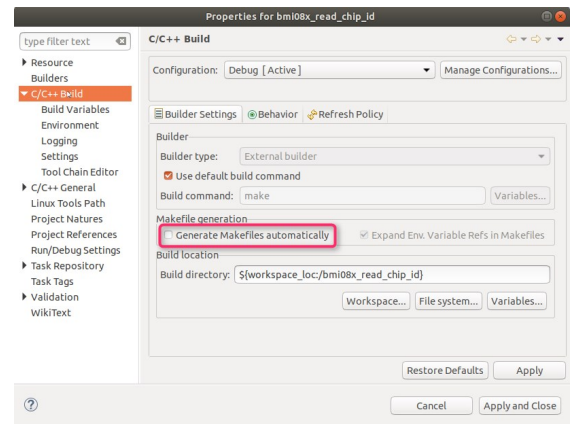
Fig. 11: Linux Eclipse Project Properties

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
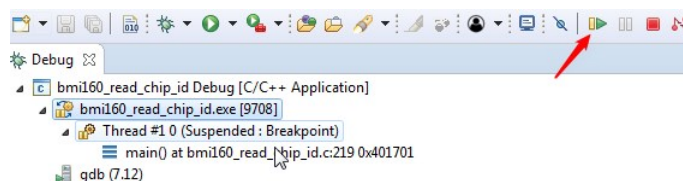
Fig. 12: Eclipse Debug Configuration

You can also use the COINES Code Editor from start menu to edit, build and run the example programs. Check Help -> Code Editor Quick Start Guide in COINES Code Editor to learn more.

4.5 Online update

The COINES installation provides a snapshot of the sensorAPI and example files taken on the day of the release of the COINES version. The code can be updated to the newest state by using the online updater, which can be launched either by clicking the appropriate update button of the COINES Code Editor or by executing the binary directly from the installation directory (here as example version 2.2 is taken):

`C:\COINES\v2.2\gui\External\coines_package_manager.exe`

From a functional point of view, the Online Updater checks for new versions of SensorAPI or example files on , downloading them and installing them in the appropriate folder of the COINES installation. The old SensorAPI and example files are zipped and the resulting zip archive is left in that very same folder, so that the old code can always be recovered (manually).

14

5 coinesAPI description

5.1 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 I^2C 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 I^2C 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.

5.2 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 I^{2}C 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.

5.3 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, VDDIO voltages on APP3.0 board are 0, 1.8V and 2.8V.

16

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 (1.8/2.8V)	2_2	SPI: SCK / I^{2}C: SCL
1_3	GND	2_3	SPI: MISO / I^{2}C: SDO
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

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

5.4 coinesAPI calls: Interface and board information

5.4.1 coines_open_comm_intf

Opens the communication interface.Currently only `COINES_COMM_INTF_USB` (USB Connection) interface is available.

In case of MCU Target, API waits till serial port is opened or DTR signal is asserted.

```
int16_t coines_open_comm_intf(enum coines_comm_intf intf_type);
```

5.4.2 coines_close_comm_intf

Closes the communication interface.

```
int16_t coines_close_comm_intf(enum coines_comm_intf intf_type);
```

5.4.3 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;
};
```

5.5 coinesAPI calls: GPIO oriented calls

5.5.1 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);
```

5.5.2 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);
```

5.5.3 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);
```

5.6 coinesAPI calls: Sensor communication

5.6.1 coines_config_i2c_bus

Configures the I^2C 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 I^2C modes are available:

```
COINES_I2C_STANDARD_MODE
COINES_I2C_FAST_MODE
COINES_I2C_SPEED_3_4_MHZ
COINES_I2C_SPEED_1_7_MHZ
```

5.6.2 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);
```

5.6.3 coines_write_i2c

Writes 8-bit register data to the I^{2}C device at `COINES_I2C_BUS_0`.

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

Arguments:

▶ dev_addr: I^{2}C device address.

▶ reg_addr: Starting address for writing the data.

▶ reg_data: Data to be written.

▶ count: Number of bytes to write.

5.6.4 coines_read_i2c

Reads 8-bit register data from the I^{2}C device at `COINES_I2C_BUS_0`.

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

Arguments:

▶ dev_addr: I^{2}C 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.

5.6.5 coines_write_spi

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

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

Arguments:

▶ 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.

5.6.6 coines_read_spi

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

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

Arguments:

▶ `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.

5.6.7 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);
```

5.6.8 coines_write_16bit_spi

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

```
int8_t coines_write_16bit_spi(uint8_t cs, uint16_t reg_addr, uint16_t *reg_data,
    uint16_t count);
```

Arguments:

▶ `cs`: Chip select pin number.

▶ `reg_addr`: Starting address for writing the data.

▶ `reg_data`: Data to be written.

▶ `count`: Number of words to write.

5.6.9 coines_read_16bit_spi

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

```
int8_t coines_read_16bit_spi(uint8_t cs, uint16_t reg_addr, uint16_t *reg_data,
    uint16_t count);
```

Arguments:

▶ `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 words to read.

5.6.10 coines_delay_msec

Introduces delay in millisecond.

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

5.6.11 coines_delay_usec

Introduces delay in microsecond.

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

5.7 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.

5.7.1 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.

▶ `coines_streaming_blocks`: Contains information regarding numbers of register blocks, range and size of 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 addresses to read, starting from the start address

For reading data from I^{2}C bus,then set the below parameters:

▶ `stream_config.intf = COINES_SENSOR_INTF_I2C;`

▶ `stream_config.i2c_bus`: I^{2}C bus (in case of APP2.0, this is always `COINES_I2C_BUS_0`)

▶ `stream_config.dev_addr`: I^{2}C 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`).

5.7.2 coines_start_stop_streaming

Starts or stops sensor data streaming.

```
int16_t coines_start_stop_streaming(enum coines_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

5.7.3 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:

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

▶ r_p: Value at register address p

BOSCH
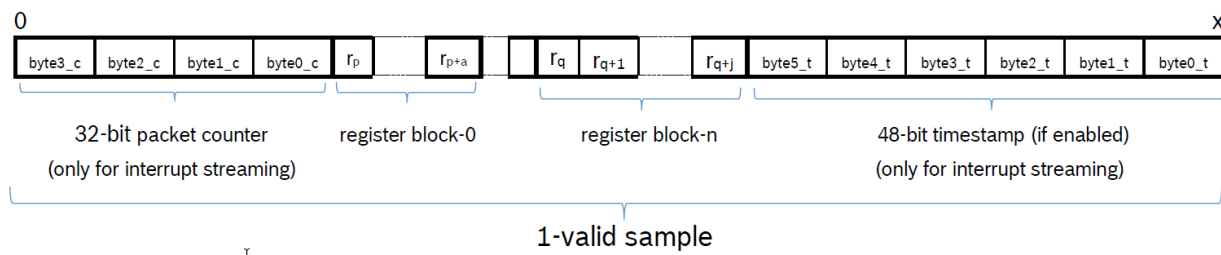
Fig. 13: Format of streaming packages

▶ 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:

$$Timestamp\,(\mu s) = \frac{48bit\_timestamp}{30}$$

5.7.4 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`)

5.8 coinesAPI calls: Other useful APIs

5.8.1 coines_get_millis

Returns the number of milliseconds passed since the program started

```
uint32_t coines_get_millis();
```

23

5.8.2 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)(void),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)

5.8.3 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 Extending the usage of the example files

6.1 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 sniplet 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 sniplet 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.

```c
#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;
}
```

6.2 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.

25

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 substract a certain offset, so see details of the signal.
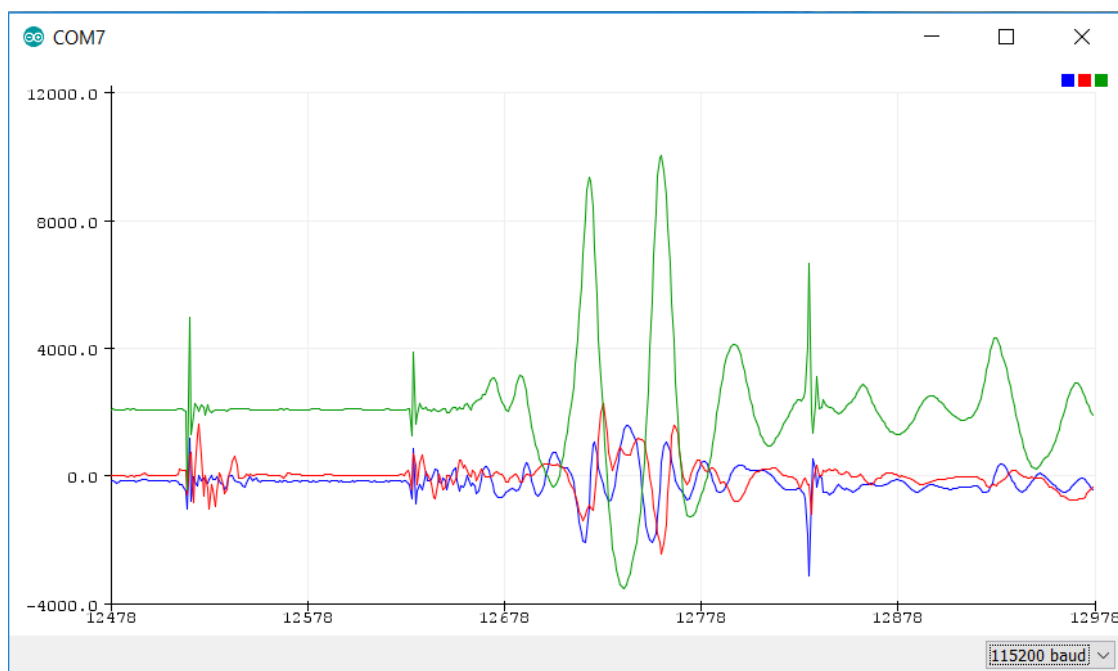Example: `printf("%d\n", (pressure - 99000));`

Fig. 14: Accelerometer sensor data on Arduino Serial Plotter

7 Accessing the Application Board using Python

7.1 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 I^2C bus parameters

▶ Read and write into registers of sensors from Bosch Sensortec via SPI and I^2C

▶ Read and write digital pins of the Application Board

7.2 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 `coinesAPI\pc\python`, in which a python wheel package is placed.

When running a python script and importing coinespy, the underlying ctypes module will try to load the shared library coineslib.dll (on Windows systems; on Linux systems the lib is called coineslib.so). The search stategy is as follows:

▶ One of the libraries included in the wheel package is checked. There are precompiled libraries available for a certain combination of operating systems and hardware configurations (e.g. Windows 64-bit, 32-bit, macOS, Linux ARMv7, x86, 64-bit). Depending on the user's configuration, the user may be lucky and the precompiled library works.

▶ If the user has compiled an own library using COINES, but has installed COINES into another folder than the default folder, the user can point to the right library inside the python code by initializing the `UserApplicationBoard` with the path to the library:
`coinespy.UserApplicationBoard(r'C:\PATH\TO\MY\libcoines.dll')`

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:

```python
import coinespy as BST

if __name__ == "__main__":

    board = BST.UserApplicationBoard()
    # If you get an error message on startup, that coineslib could not be loaded, then
    # intialize the UserApplicationBoard object with the path to the library, e.g.
    #board = BST.UserApplicationBoard(r'C:\COINES\v2.1\coinesAPI\libcoines.dll')

    board.PCInterfaceConfig(BST.PCINTERFACE.USB)
    if board.ERRORCODE != 0:
```

```
print('Could not connect to board: %d' % (board.ERRORCODE))
else:
b_info = board.GetBoardInfo()
print('BoardInfo: HW/SW ID: ' + str(b_info.HardwareId) + '/' +
    str(b_info.SoftwareId))
board.ClosePCInterface()
```

7.3 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.

Note: the streaming functionality (polling or interrupt streaming) is not available through the python interface.

7.3.1 coinespy API calls: Interface and board information

The following function calls are defined within the class `UserApplicationBoard`. Thus in order to access the functions, the user has to create an object of that class first.

7.3.1.1   PCInterfaceConfig

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

```
coinespy.PCInterfaceConfig(PCINTERFACE communicationChannel, string portName)
```

For the definition of `PCINTERFACE`, refer to 7.3.4.4.

7.3.1.2   ClosePCInterface

Disposes the resources used by the USB/serial communication.

```
coinespy.ClosePCInterface()
```

7.3.1.3   GetBoardInfo

Obtains board specific information.

```
BoardInfo = coinespy.BoardInfo = coinespy.GetBoardInfo()

# 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
```

7.3.2 coinespy API calls: GPIO oriented calls

7.3.2.1   PinConfig

Configures the state, level and direction of a GPIO pin

```
coinespy.PinConfig(int pinNumber, EONOFF switchState, PINMODE direction, PINLEVEL
    outputState)
```

For the definition of `EONOFF`, refer to 7.3.4.1. For the definition of `PINMODE`, refer to 7.3.4.2. For `PINLEVEL`, refer to 7.3.4.3.

7.3.2.2   GetPinConfig

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

```
PinConfigInfo = coinespy.GetPinConfig(ushort pinNumber)

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

7.3.2.3   SetVDD

Set the VDD voltage level.

```
coinespy.SetVDD(volts)

# Example: coinespy.SetVDD(3.3)
```

7.3.2.4   SetVDDIO

Set the VDDIO voltage level.

```
coinespy.SetVDDIO(volts)

# Example: coinespy.SetVDDIO(3.3)
```

7.3.3 coinespy API calls: Sensor communication

7.3.3.1   Read

Reads data from the sensor.

```
data = coinespy.Read(registerAddress, numberofReads=1, sensorInterfaceDetail=None)

# sensorInterfaceDetail:
# * SPI configuration: CSB pin (class ShuttleBoardPin)
# * I2C configuration: i2c address
# * Can be left empty if only one sensor is on the bus. configured through
    SensorSPIConfig or SensorI2CConfig
```

```
# Return: list of retrieved values as int type.
```

The CSB pin shall be given as item from 7.3.4.9.

7.3.3.2 Write

Writes data to a partcular register.

```
coinespy.Write(registerAddress, registerValue, sensorInterfaceDetail=None)

# registerValue: either a single value of type int or an array of values to be written
    to the sensor. The function supports burst write (i.e. in case of more than one
    byte to write, register address is incremented for each following byte).
    Recommendation is to only write one byte at a time otherwise a long delay after the
    write command has to be considered.

# sensorInterfaceDetail:
# * SPI configuration: CSB pin (class ShuttleBoardPin)
# * I2C configuration: i2c address
# * Can be left empty if only one sensor is on the bus. configured through
    SensorSPIConfig or SensorI2CConfig
```

The CSB pin shall be given as item from 7.3.4.9.

7.3.3.3 SensorI2CConfig

Sets the interface to I2C and sets the I2C speed.

```
coinespy.SensorI2CConfig(i2cAddress, I2CSPEED speed)
```

For the definition of `I2CSPEED`, refer to 7.3.4.5.

7.3.3.4 SensorSPIConfig

Set the interface to SPI and sets the SPI speed and mode.

```
coinespy.SensorSPIConfig(chipSelectPin, SPISPEED spiSpeed=SPISPEED.SPI1000KBIT, SPIMODE
    spiMode=SPIMODE.MODE0)
```

The CSB pin shall be given as item from 7.3.4.9.

For the definition of `SPISPEED`, refer to 7.3.4.6. For the definition of `SPIMODE`, refer to 7.3.4.8.

7.3.3.5 CustomSPIConfig

Configures the SPI and sets the speed and mode. Difference to the `SensorSPIConfig` is, that in this function the SPI speed can be given in a free format. Only available for backward-compatibility with GenericAPI.

```
coinespy.CustomSPIConfig(sensorId, chipSelectPin, spiSpeed, SPIMODE
    spiMode=SPIMODE.MODE0)
```

The CSB pin shall be given as item from 7.3.4.9.

For the definition of SPISPEED, refer to 7.3.4.6. For the definition of SPIMODE, refer to 7.3.4.8.

7.3.3.6   Sensor16bitSPIConfig

Configures SPI for 16-bit read and write.

```
coinespy.Sensor16bitSPIConfig(chipSelectPin, spiSpeed=60, spiMode=SPIMODE.MODE0,
    spiBits=SPIBITS.SPI16BIT)
```

7.3.4 Definiton of constants

7.3.4.1   EONOFF

Defintion of value for ON and OFF.

```
class EONOFF:
   OFF = 0
   ON = 1
```

7.3.4.2   PINMODE

Definition of value for direction state of the pin. Sets to output or input.

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

7.3.4.3   PINLEVEL

Definition of value for pin level status. Either high or low.

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

7.3.4.4   PCINTERFACE

Definition to activate the communication channel.

```
class PCINTERFACE:
   USB = 0     # COINES_COMM_INTF_USB
   SERIAL = 1 # COINES_COMM_INTF_VCOM
```

7.3.4.5   I2CSPEED

Definition of the I2C speed.

```
class I2CSPEED:
    STANDARDMODE = 0 # Standard mode - 100kHz
    FASTMODE = 1     # Fast mode - 400kHz
    HSMODE = 2       # High Speed mode - 3.4 MHz
    HSMODE2 = 3      # High Speed mode 2 - 1.7 MHz
```

7.3.4.6   SPISPEED

SPI speed definition.

```
class SPISPEED:
    SPI250KBIT = 240     # COINES_SPI_SPEED_250_KHZ = 240 - 250 kHz */
    SPI300KBIT = 200
    SPI400KBIT = 150
    SPI500KBIT = 120
    SPI600KBIT = 100
    SPI750KBIT = 80
    SPI1000KBIT = 60
    SPI1200KBIT = 50
    SPI1250KBIT = 48
    SPI1500KBIT = 40
    SPI2000KBIT = 30
    SPI2500KBIT = 24
    SPI3000KBIT = 20
    SPI3750KBIT = 16
    SPI5000KBIT = 12
    SPI6000KBIT = 10
    SPI7500KBIT = 8
    SPI10000KBIT = 6
```

7.3.4.7   SPIBITS

SPI register access width.

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

7.3.4.8   SPIMODE

SPI MODE definition.

```
class SPIMODE:
    MODE0 = 0     # SPI Mode 0: CPOL=0; CPHA=0
    MODE1 = 1     # SPI Mode 1: CPOL=0; CPHA=1
    MODE2 = 2     # SPI Mode 2: CPOL=1; CPHA=0
    MODE3 = 3     # SPI Mode 3: CPOL=1; CPHA=1
```

7.3.4.9 ShuttleBoardPin

Definiton of pins on the shuttle board which can be used as general purpose input/output pins.

```
class ShuttleBoardPin:
    COINES_SHUTTLE_PIN_7 = 9 # CS pin
    COINES_SHUTTLE_PIN_8 = 5 # Multi-IO 5
    COINES_SHUTTLE_PIN_9 = 0 # Multi-IO 0
    COINES_SHUTTLE_PIN_14 = 1 # Multi-IO 1
    COINES_SHUTTLE_PIN_15 = 2 # Multi-IO 2
    COINES_SHUTTLE_PIN_16 = 3 # Multi-IO 3
    COINES_SHUTTLE_PIN_19 = 8 # Multi-IO 8
    COINES_SHUTTLE_PIN_20 = 6 # Multi-IO 6
    COINES_SHUTTLE_PIN_21 = 7 # Multi-IO 7
    COINES_SHUTTLE_PIN_22 = 4 # Multi-IO 4

# APP3.0 pins
    COINES_MINI_SHUTTLE_PIN_1_4 = 0x10 # GPIO0
    COINES_MINI_SHUTTLE_PIN_1_5 = 0x11 # GPIO1
    COINES_MINI_SHUTTLE_PIN_1_6 = 0x12 # GPIO2/INT1
    COINES_MINI_SHUTTLE_PIN_1_7 = 0x13 # GPIO3/INT2
    COINES_MINI_SHUTTLE_PIN_2_5 = 0x14 # GPIO4
    COINES_MINI_SHUTTLE_PIN_2_6 = 0x15 # GPIO5
    COINES_MINI_SHUTTLE_PIN_2_1 = 0x16 # CS
    COINES_MINI_SHUTTLE_PIN_2_3 = 0x17 # SDO
```

7.3.4.10 MULTIIO

Definiton of pins on the shuttle board which can be used as general purpose input/output pins (these definitions are only for backward compatibilty, please use those definitions as stated in 7.3.4.9).

```
class MULTIIO:
    MULTIIO_0 = 0
    MULTIIO_1 = 1
    MULTIIO_2 = 2
    MULTIIO_3 = 3
    MULTIIO_4 = 4
    MULTIIO_5 = 5
    MULTIIO_6 = 6
    MULTIIO_7 = 7
    MULTIIO_8 = 8

# Old style naming for APP3.0 pins
class GPIO:
    GPIO_0 = 0x10
    GPIO_1 = 0x11
    GPIO_2 = 0x12
    GPIO_3 = 0x13
    GPIO_4 = 0x14
    GPIO_5 = 0x15
```

7.3.5 Error Codes

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

```python
BOARD = coinespy.UserApplicationBoard()
try:
    BOARD.PCInterfaceConfig(PCINTERFACE.USB)
except:
    print('Board initialization failed: ' + str(BOARD.ERRORCODE))
    exit(BOARD.ERRORCODE)
```

7.3.5.1 General Error Codes

Error Values	Description
0	No Error response/Success
-1	Failure
-2	Length Error
-4	Configuration is Unsuccessful
-5	Invalid Instruction
-6	Memory Error
-100	Timeout

7.3.5.2 Pinconfig Specific Error Codes

Error Values	Description
1	Analog Switch is turned ON/OFF
-10	Invalid Pin
-19	Invalid ADC Pin

7.3.5.3 Read/Write Specific Error Codes

Error Values	Description
2	Default read of 128 bytes is done. Requested bytes of read not supported. For APP2.0 board read more than 128 bytes is possible and up to 1204 bytes
-3	The number of bytes that shall be read is 2kB. If this exceeds, error code is updated.
-18	1. For APP2.0 the maximum number of bytes that shall be written for burst operation is 2kB based on RAM requirements. 2. For AB/DB, due to RAM size the maximum number of bytes that shall be written for burst operation is 46 bytes.

7.4 Migration from 'GenericAPI' to coinespy

The attempt was undertaken to keep the names of functions, constants and variables as close as possible to the GenericAPI. To migrate to coinespy, the user should only need to remove the .NET related library (i.e. `import clr`) and import coinespy as BST, as shown in the example files.

Old code may still contain some type conversions which are not necessary anymore. To avoid errors, the user could either modify the code or simply add these lines to his code (at the top of the file):

```python
def Byte(value):
    return value

def UInt16(value):
    return value

def Array(value):
    return value
```

8 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 compatibiliy 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 `coines_config_streaming`, `coines_read_stream_sensor_data` etc., are meant to work only on PC.
   - ▶ Use of APIs like `coines_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 .,

9 Legal disclaimer

9.1 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.

9.2 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.

9.3 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.

10 Document history and modifications

Rev. no.	Chapter	Description of modification/changes	Date
1.0		Initial release	Dec 2018
1.1	All	Running COINES on MCU	Mar 2019
1.2	All, 4.1.1	Added COINES Editer description, minor changes	Feb 2020
1.3	5.6.8, 5.6.9 7	Added 16-bit SPI functions Added Python interface description	Mar 2020

38