User manual

DA14583 IoT sensor development kit

UM-B-063

Abstract

This document describes the architecture and the implementation details of the Dialog Bluetooth® IoT Sensor Kit application running on the DA14583 development kit and on Android or iOS.

Contents

Figures

Tables

1 Terms and definitions

AHRS	Attitude and Heading Reference System
API	Application Programming Interface
BD	Bluetooth Device (address)
BLE	Bluetooth Low Energy (Bluetooth Smart)
CIB	Communication Interface Board
IoT	Internet of Things
DK	Development Kit
DWS	Dialog Wearable Service
FSM	Finite State Machine
GAP	Generic Access Profile
GATT	Generic ATTribute profile
GUI	Graphical User Interface
MEMS	Micro Electro-Mechanical Systems
MTU	Maximum Transmission Unit
NED	North East Down (coordinate system)
NVDS	Non-Volatile Data Storage
PC	Personal Computer
RAM	Random Access Memory
Report	Notification of sensor data and control
SF	Sensor Fusion
SFL	Sensor Fusion Library

2 References

[1] UM-B-050, DA1458x Software development guide, User manual, Dialog Semiconductor.

[2] UM-B-051, DA1458x Software architecture, User manual, Dialog Semiconductor.

[3] DA14583 Datasheet, Dialog Semiconductor.

[4] BMI160 Datasheet, Bosch Sensortec.

[5] BMM150 Datasheet, Bosch Sensortec.

[6] BME280 Datasheet, Bosch Sensortec.

[7] GAP Assigned Numbers, Bluetooth SIG.

[8] Quaternion, Wikipedia.

[9] Quaternions and spatial rotation, Wikipedia.

3 Introduction

3.1 Internet of Things (IoT)

With yearly shipments of more than 10 billion microcontrollers that all can exchange information locally or through the Internet, a huge variety of so called ‘intelligent devices’ are enabled. These devices include motion sensors, pool pumps, gas/electric meters, street lamps and many more.

All these devices can be accessed over the Internet; thanks to the rapid increase in infrastructure coverage and Internet access. This evolution is often called the Internet of Things (IoT). Other names include Internet of Everything (IoE), Web of Things, Embedded Web and Industry 4.0. The goal is to establish an Internet connection for the small ‘things’ you carry with you or use in a factory, hospital, in a city or in a home. IoT industrial experts predict that there are going to be more than 50 billion connected devices within the next 10 years.

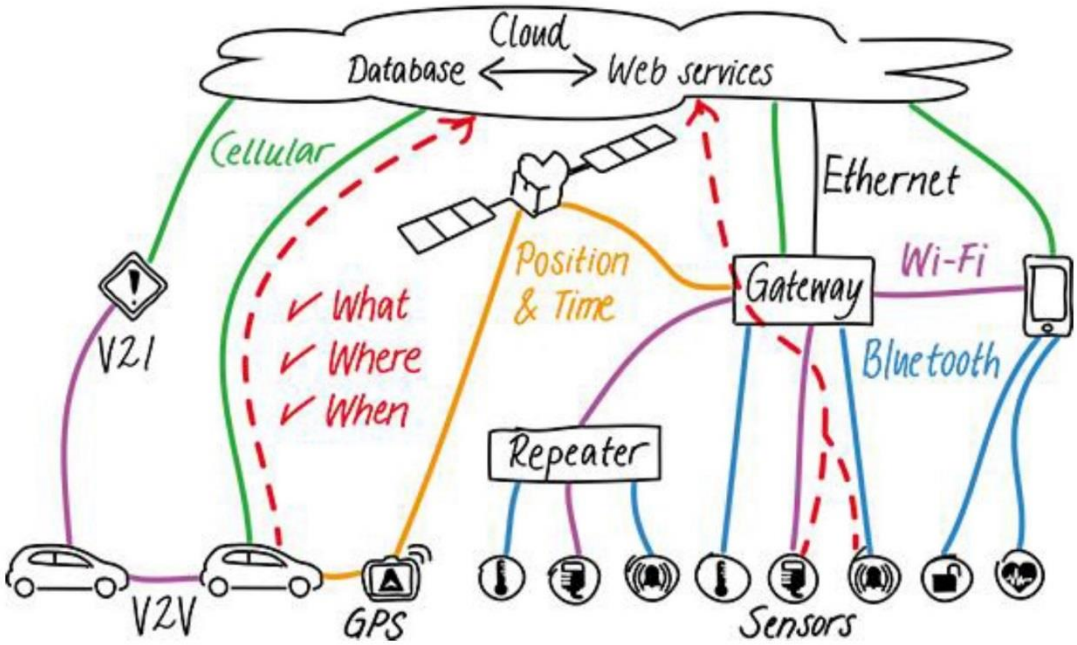

Figure 1: Internet of Things (IoT)

3.2 IoT Sensor board

The Dialog Semiconductor IoT Sensor board (Figure 2) provides the means to gather and process sensor data from six different sensors to develop an IoT application. It also includes the SmartFusion algorithm that extends the functionality of the device to advanced smart applications.

The IoT Sensor board allows developers to kick-start their projects with a proven hardware and software reference. This design will shorten the time to market for applications that require sensor data to be accessible from the Internet.

The application is built around the DA14583 Bluetooth Smart device. This device provides very low power consumption using an ARM M0 microcontroller and internal Flash memory. The integrated DC-DC converter and a single-ended RF port make this a very attractive device for use in battery supplied application requiring very small board space.

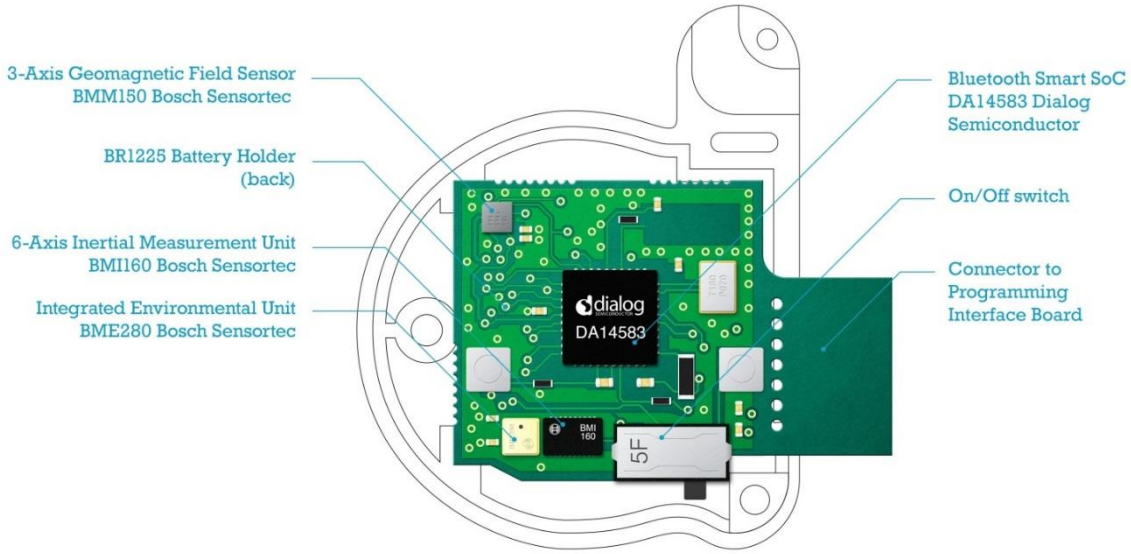

Figure 2: IoT Sensor board

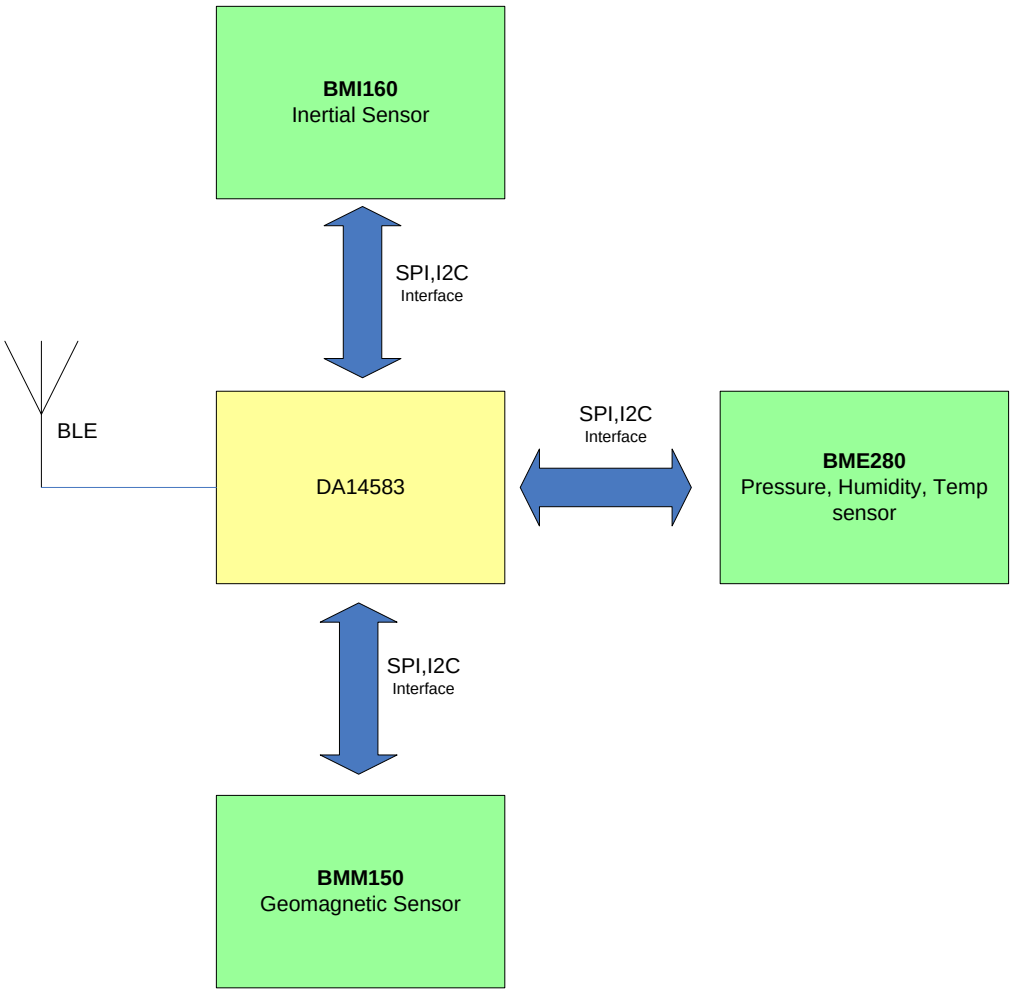

Figure 3: IoT Sensor board block diagram

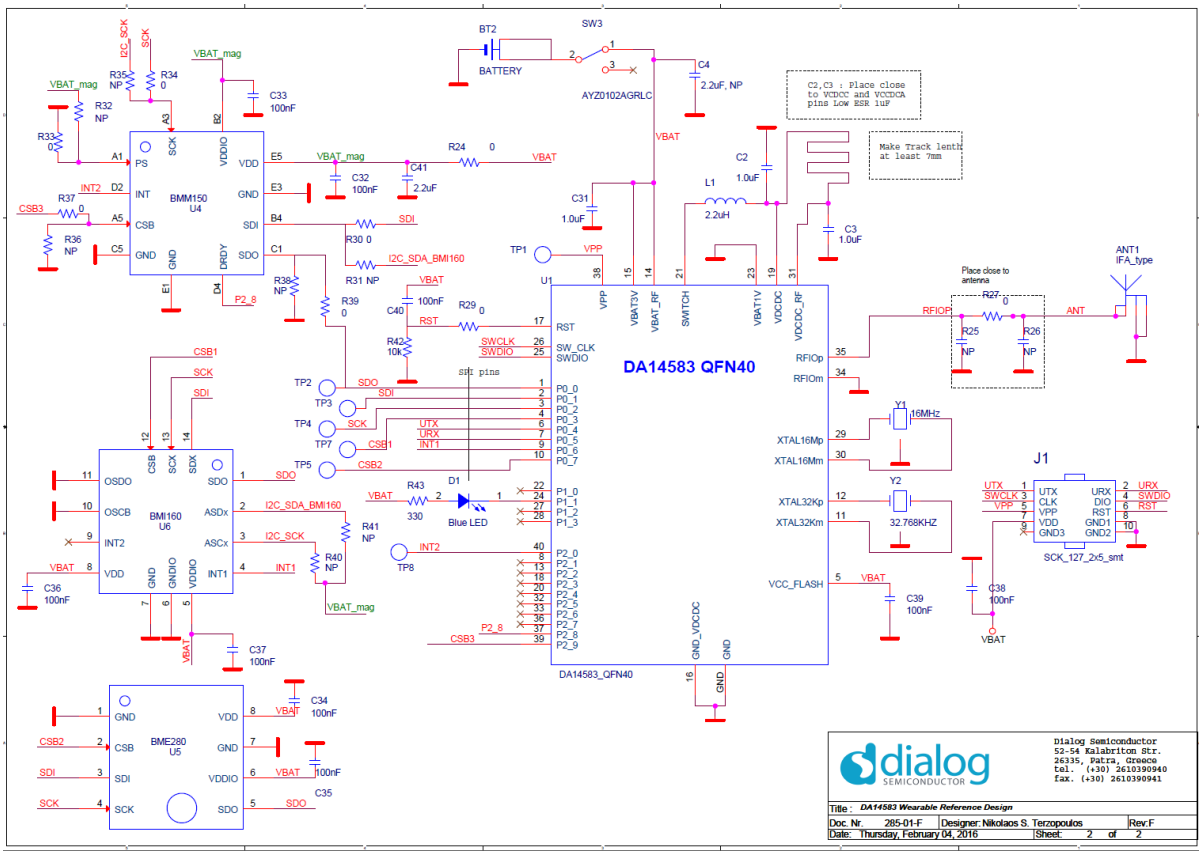

Figure 4: IoT Sensor board schematic

3.3 Sensor features

- BMI160: low-power, low-noise 3-axis 16 bit Inertial Measurement Unit (IMU) designed for use in mobile and indoor applications which require highly accurate, real-time sensor data.
- BMM150: low power and low noise 3 axis geomagnetic sensor to be used in compass applications.
- BME280: integrated environmental sensor (pressure, humidity and temperature) developed specifically for mobile applications.

3.4 Software features

This section explains the advanced software features of the Dialog IoT DK reference application.

- GAP Peripheral role.
- GATT Based bidirectional data and control transfers.
- Sensor fusion library with update rate from 10 Hz to 25 Hz and selection between three different sensors: accelerometer, gyroscope and magnetometer.
- Three environmental sensors: pressure, humidity and temperature.
- Selection between Sensor Fusion Library (SFL) and raw data (RAW) projects.
- Notifications for sensor data streaming.
- Sensor range and update rate control.
- Configuration set, save and restore.
- Three calibration modes for magnetometer.
- iOS and Android Central Application.
- Auto sleep and motion wakeup.

4 Firmware upgrade

This section describes the steps necessary to reprogram the IoT Sensor board with a new firmware version.

Prerequisites:

- Firmware file in hex format.
- SmartSnippets tool from the Dialog support site.
- IoT DK Communication Interface Board (CIB).
- IoT sensor board.

4.1 Connecting the IoT sensor board to the Communication Interface Board

1. Connect the CIB to the PC with a mini-USB cable.
2. Turn OFF the CIB's power switch (SW2, Figure 5).
3. Turn OFF the IoT sensor board ON/OFF switch and remove the battery (Figure 6).
4. Plug the IoT sensor board onto the CIB at the J5 connector.
5. Turn ON the power switch (SW2). VDD LED (D3) must be illuminated.

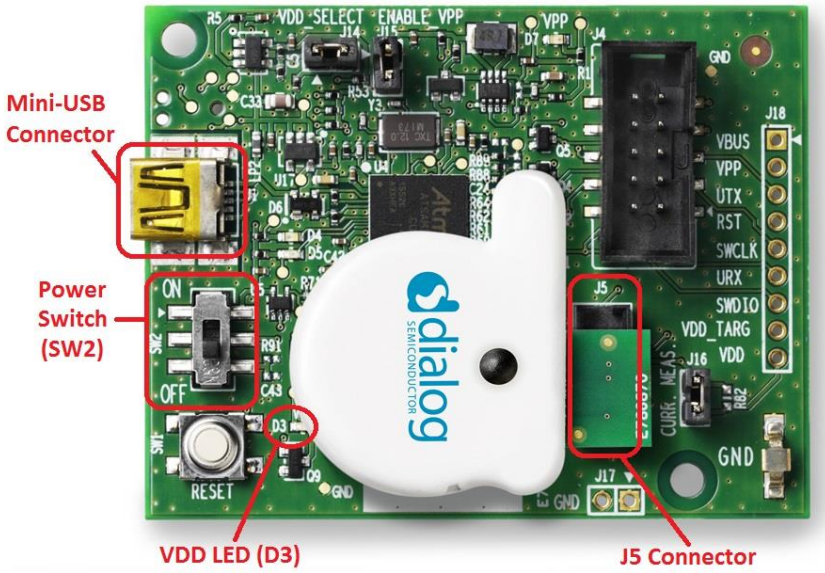

Figure 5: IoT Sensor attached to the interface board

⚠ CAUTION
Wrong connection of the IoT sensor board may damage both boards and render them unusable.

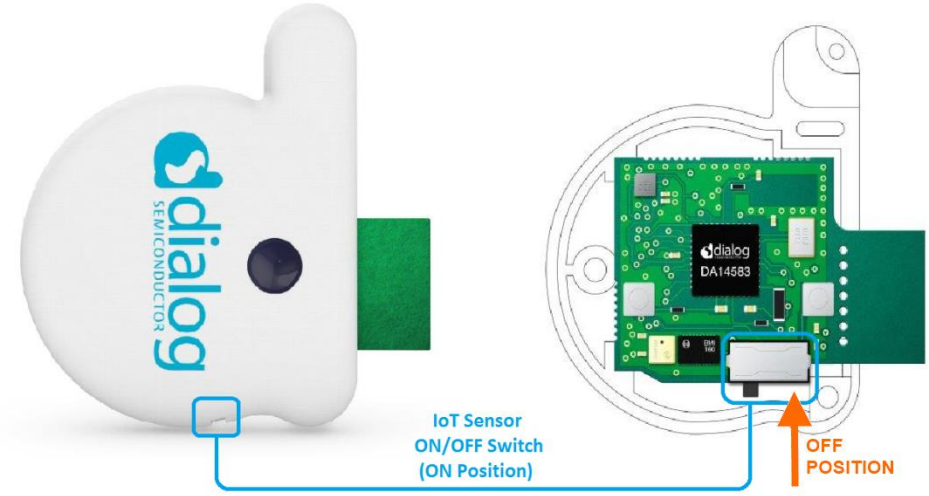

Figure 6: IoT Sensor ON/OFF switch

4.2 Programming using SmartSnippets

1. Start the SmartSnippets tool and perform the following actions:
   a. From the project screen select UART mode/UART COM port with DA14583.
   b. When using JTAG select JTAG/JTAG ID with DA14583 (Figure 7).
   c. Select the project type from the left column.

      For more information on how to create projects with various communication methods consult the SmartSnippets user guide that accompanies the SmartSnippets installation package.
   d. Select the SPI Flash Programmer (Figure 8).
2. In the SPI Flash Programmer proceed as follows:
   a. Select the new firmware hex file from the upper left window (Figure 8).
   b. Press the reset switch (SW1) on the interface board.
   c. Press the *Connect* button (Figure 8). A confirmation message will appear in the SmartSnippets log window. The user may be asked to press the reset button on the interface board.
   d. Press the *Erase* button (Figure 8) to erase the entire Flash memory.
   e. Press the *Burn* button (Figure 8) to write the new firmware into the Flash memory. When asked via a popup window whether to make the Flash image bootable, select Yes.
   f. After successful programming, reset the interface board using SW1 or by power cycling.
   g. If for any reason the programming fails, repeat the programming sequence from step b.

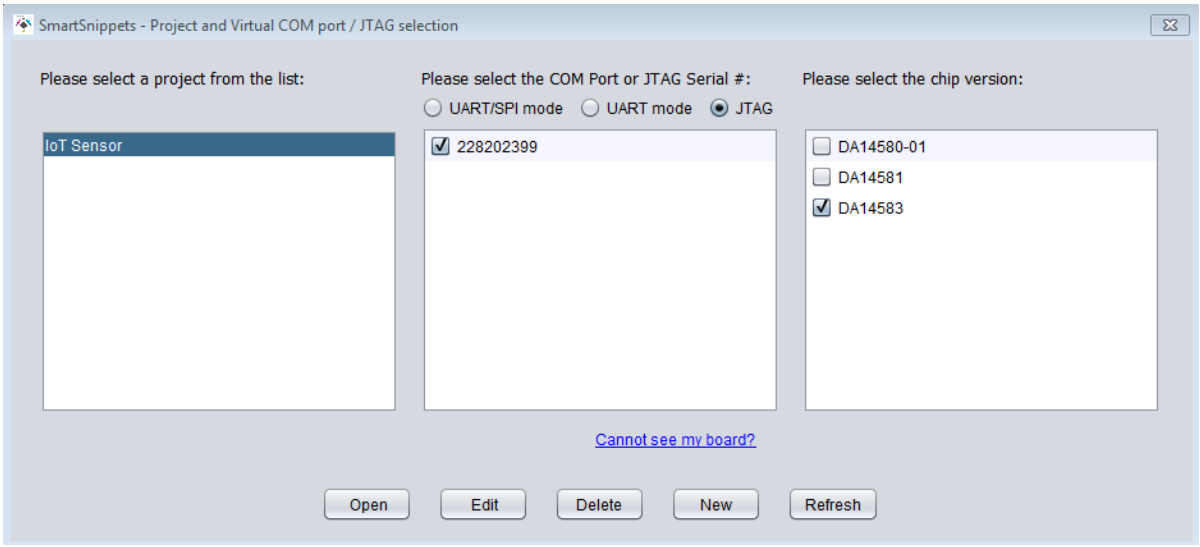

Figure 7: SmartSnippets project selection

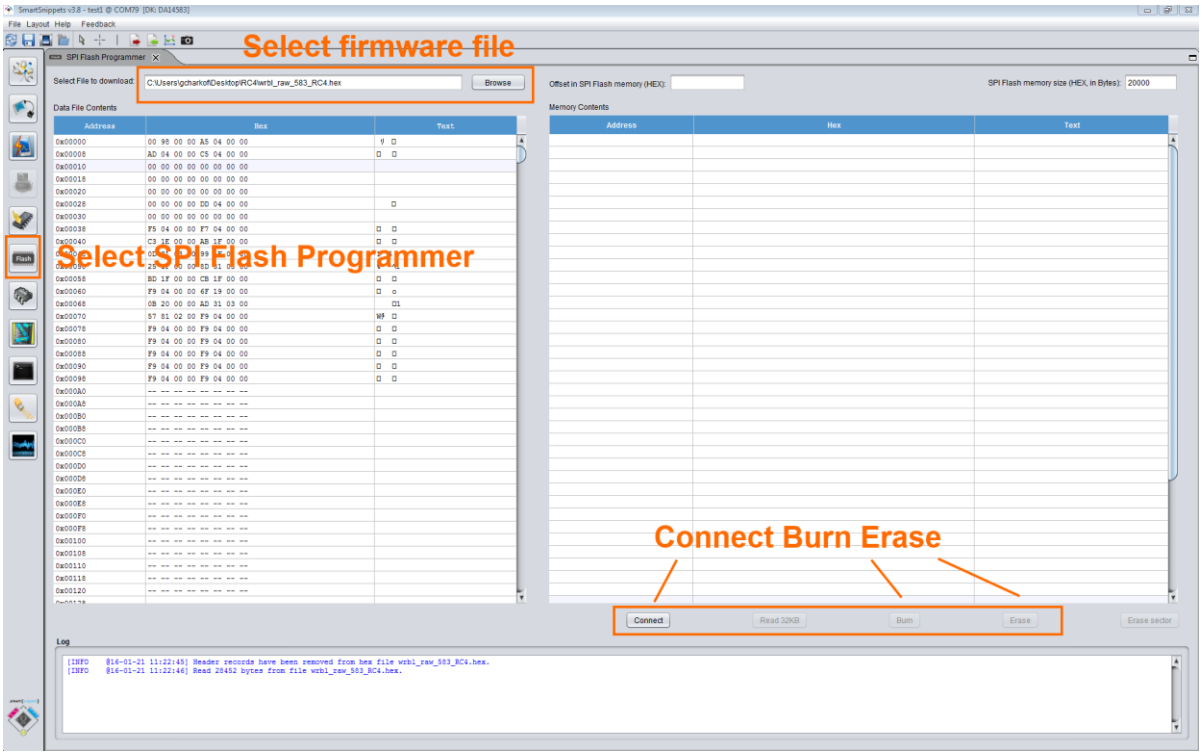

Figure 8: SPI Flash erase/burn

5 Software architecture

5.1 Project files

The IoT DK contains two projects based on DA14580 SDK version 5.0.3 including a ROM code patch for fixing an issue for an invalid LLC state.

- **Sensor Fusion Library (SFL) Keil project:**

  Resides in `projects\target_apps\wrbl\wrbl_sfl\Keil_5\wrbl_sfl.uvprojx`. This project uses the full featured interrupt/FIFO mechanism of BMI160 in order to acquire samples at the highest possible data rate. The samples are processed by the Sensor Fusion Library to produce a 3D vector of the board orientation. On demand, only a part of the raw data can be sent over to the central device at the same rate of the Sensor Fusion Rate.

- **Raw Data (RAW) Keil project:**

  Resides in `projects\target_apps\wrbl\wrbl_raw\Keil_5\wrbl_raw.uvprojx`. This project does not use the Sensor Fusion Library, but all the raw data gathered by the sensors can be transferred to the central device over BLE. The FIFO mechanism is not used and the code size is considerably smaller than the SFL project.

Both projects share a common and a separate folder structure:

- Common files: `projects\target_apps\wrbl\common`
- SFL specific files: `projects\target_apps\wrbl\wrbl_sfl`
- RAW specific files: `projects\target_apps\wrbl\wrbl_raw`
- Common SDK folder: `sdk\`

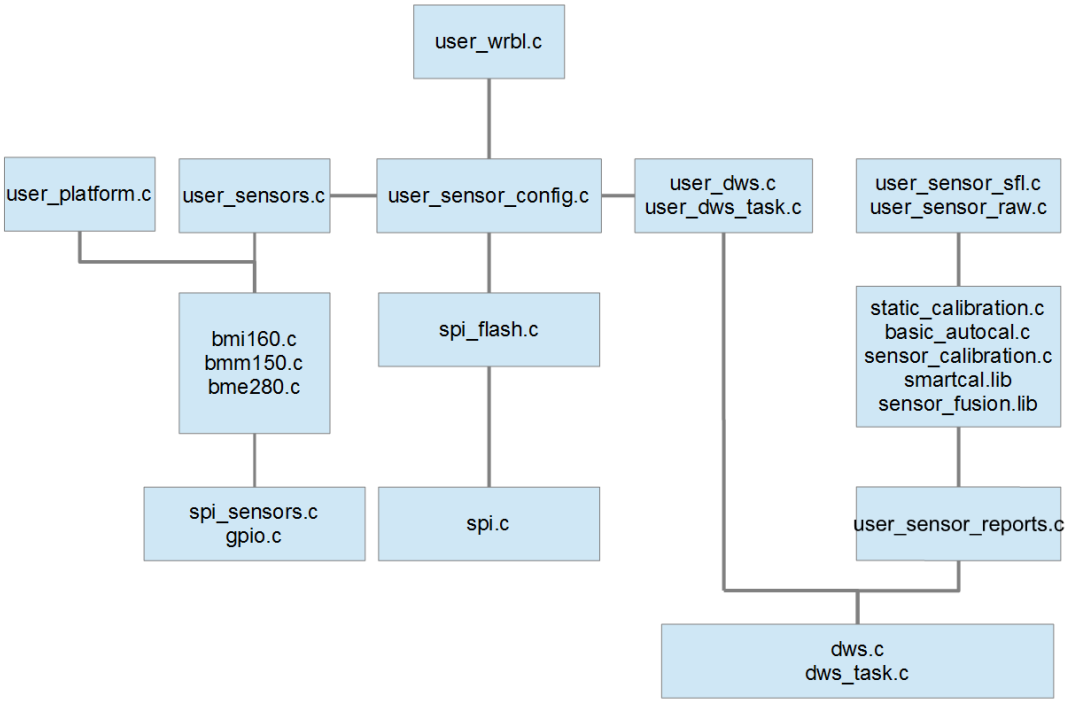

Figure 9: IoT DK software architecture

5.2 Source files

Some of the project files of the IoT DK belong to the DA1458x SDK or are project specific. For more information on the SDK, please consult references [1] and [2].

Table 1: IoT DK source file overview

Group	Files	Description
Boot (SDK)	system_ARMCM0.c boot_vectors.s hardfault_handler.c	Application start-up, ISR vectors, hard fault handler
Architecture (SDK)	arch_main.c jump_table.c arch_sleep.c nmi_handler.c arch_system.c arch_console.c nvds.c arch_patch.c wlan_coex.c	Main loop, kernel scheduling, system setup, system clocks, common sysRAM/ROM code structure (jump table), sleep modes API, NVDS.
Drivers (SDL)	syscntl.c gpio.c adc.c spi_flash.c spi.c wkupct_quadec.c	Peripheral drivers for System Control, GPIO, ADC, SPI, SPI Flash and Wakeup Controller.
BLE (SDK)	rf_580.c rwble.c rwip.c gapm.c	Radio and BLE related functions for interrupt and sleep mode.
Profiles (SDK)	prf_utils.c attm_db_128.c custom_common.c	Profile related functions and utilities.
App (SDK)	app_default_handlers.c app.c app_task,c app_security.c app_security_task.c app_entry_point.c app_msg_utils.c app_easy_msg_utils.c app_easy_security.c app_easy_timer.c	Application support files: security functions and ‘easy’ functions.
User Platform Files (IoT DK)	user_periph_setup.c spi_sensor.c bme280.c bmi160.c bmm150.c user_platform.c	Project/platform specific files: sensor drivers, peripheral setup functions and SPI handling utilities.
User Application (IoT DK)	user_wrbl.c user_dws.c user_dws_task.c	Main application files. File user_wrbl.c contains the connect/disconnect/advertise handlers. Files user_dws.c and user_wrbl_task.c contain the user interface to the Dialog Wearable Service.
User Profiles (IoT DK)	dws.c user_dws_task.c	Custom service functions and FSM handlers.

Group	Files	Description
Sensor Calibration (IoT DK)	basic_autocal.c static_calibration.c sensor_calibration.c smartfusion_autocal.lib	Magnetometer calibration functions, specific to each calibration mode.
User Sensors (IoT DK)	user_sensors.c user_sensor_config.c user_sensor_reports.c user_sensor_raw.c user_sensor_sfl.c	 RAW project only. SFL project only.

Table 2: IoT specific source files

File name	Description
user_wrbl.c	Contains all top level BLE callback functions for connection, disconnection and advertising.
user_platform.c	Contains platform specific code: SPI chip select, interrupt configuration and interrupt callback functions.
user_sensors.c	Contains sensor modes, interrupt user level configuration functions and the main application's finite state machine function user_wrbl_timer_handler.
bmi160.c bmm150.c bme280.c	Modified sensor drivers, originals provided by Bosch Sensortec. They provide all elementary functions to access the variety of features of the sensors. These files use the spi_sensors.c and gpio.c.
spi_sensors.c gpio.c	Driver files for low level SPI access.
user_sensor_config.c	Functions that allow the device to store/retrieve its configuration to/from the internal Flash memory.
user_dws.c user_dws_task.c	Provide the user space access to the DWS custom service.
user_sensor_sfl.c user_space_raw.c	User space sensor data processing for SFL and RAW projects respectively.
static_calibration.c basic_autocal.c sensor_calibration.c smartfusion_autocal.lib	The calibration functions for the three modes, static, basic auto and SmartFusion auto.
sensor_fusion.lib	The sensor fusion functions, SFL project only.
user_sensor_reports.c	The user space functions that are specific for sending reports (sensor and sensor fusion data) to the central device.
dws.c dws_task.c	The custom service files.

Both the Sensor Fusion and Raw projects contain a header file for each source file that contains function prototypes and definitions. Some header files contain information that is essential for the operation and configuration of the IoT project. Note that header files are different for each project and reside in folder projects\target_apps\wrbl\wrbl_XXX\src\config. The header files are listed in Table 3.

Table 3: Header files for IoT DK configuration

File name	Description
da1458x_config_basic.h (SDK)	Basic configuration of DA14583 for security, watchdog and print functions.
da1458x_config_advanced.h (SDK)	Advanced features of DA14583, such as low power clock source.
user_config.h (SDK)	Configure sleep modes, advertise and parameter update related data.

File name	Description
user_app_wrbl_config.h (IoT DK)	Application specific definitions: enable or disable magnetometer calibration, motion wakeup and LED signalling.
user_callback_config.h (SDK)	Application callback functions.
user_modules_config.h (IoT DK)	User related module activation/deactivation.
user_periph_setup.h (IoT DK)	Peripheral related definitions (GPIO).
user_config_sw_version.h (IoT DK)	SW version.
user_profiles_config.h (IoT DK)	Included profiles.
user_dws_config.h (IoT DK)	Definition of DWS Service.

5.3 Application configuration

A group of compilation switches allows control of the application's behaviour. The application has been extensively tested to correctly compile and operate using the default values. The most important switches are listed in Table 4.

Table 4: Configuration parameters

Name	Default value	Description
user_config.h		
app_default_sleep_mode	ARCH_EXT_SLEEP_ON	Use ARCH_SLEEP_OFF when connecting a SW debugger.
USER_DEVICE_NAME	'IoT-DK-SFL'	The advertising name string, see section 7.2.
connection_param_configuration	Min-Max connection interval: 20 ms to 40 ms, Latency: 4 (events missed), Supervision timeout: 2 s.	Recommended settings. For more information see section 7.1.
da1458x_config_basic.h		
CFG_PRINTF	DISABLED	Enables/disables the use of UART debug messages.
BLE_APP_SENSOR_FUSION	ALWAYS ENABLED	Always enabled in SFL project.
BLE_APP_WRBL_RAW	ALWAYS ENABLED	Always enabled in RAW project.
CFG_WDOG	ENABLED	Enables the Watchdog timer.
da1458x_config_advanced.h		
CFG_LP_CLK	LP_CLK_XTAL32	LP_CLK_XTAL32: Use external XTAL32 as a low power clock (recommended). LP_CLK_RCX20: Use internal RCX20 as a low power clock.
CFG_NVDS_TAG_BD_ADDRESS	{0x46, 0x19, 0x00, 0xCA, 0xEA, 0x80}	Defines the BD address if it is not programmed in OTP.
da1458x_config_advanced.h		
FAST_ADV_INTERVAL	160 (= 100 ms)	The advertising interval. Unit: 0.625 ms.
FAST_ADV_TO	6000 (= 60 s)	The time that the device advertises before it goes to sleep. Unit: 10 ms.

Name	Default value	Description
USE_MOTION_WAKEUP	ENABLED	Enables the motion wakeup procedure. For more information see section 7.3
USE_LED	ENABLED	Enables the use of the LED device to signal advertising and connection. For more information see section 8.4.2.
MAGNETO_CAL_ENABLED	ENABLED	Enables the calibration function of magnetometer. For more information see sections 8.8 and 9.
SENSOR_SAMPLE_RATE_IMU	SENSOR_SAMPLE_RATE_IMU_25Hz (RAW Project) SENSOR_SAMPLE_RATE_IMU_100Hz (SFL Project)	Defines the default rates after a Reset to Defaults command is issued. See section 6.1.3.4.

5.4 Memory map

The project's default configuration has been carefully chosen to allow compilation even with the Keil MDK-Lite tools that have a 32 kB image size limitation. When the user wants to activate additional features (such as BLE security) a Keil tools licence is required. Also, image sizes above 32 kB cannot be stored in OTP and the deep sleep feature of DA14583 cannot be used.

The RAM base address is 0x00000000 and the project memory layout is described in the project scatter file `scatterfile_common_mod_0x0.sct` located in the scatter file directory `\sdk_580\sdk\common_project_files\scatterfiles\`. It may be reviewed by using the Linker tab at project settings. The scatter file is common to both RAW and SFL projects.

Table 5: IoT Sensor DK memory configuration

Section name	Section base address	Section size (B)	Contents
LR_IROM1	BASE_ADDRESS	0x160	Boot vectors
LR_IROM2	BASE_ADDRESS + 0x160	0x160	Jump table
LR_IROM3	BASE_ADDRESS + 0x2C0	0x80	Timeout table
LR_IROM4	BASE_ADDRESS + 0x340	0x100	NVDS storage
ER_IROM5	BASE_ADDRESS + 0x440	0x93C0	Vectors, libraries, application code
RW_IRAM51	BASE_ADDRESS + 0x9000 - NON_RET_HEAP_SIZE	1036 NON_RET_HEAP_SIZE	Non retainable heap
RW_IRAM52	BASE_ADDRESS + 0x9000	0x20	Reserved by ROM code
RW_IRAM53	BASE_ADDRESS + 0x9020	0x1E0	Zero initialised data
RW_IRAM54	BASE_ADDRESS + 0x9200	0x600	Stack
ZI_RET20	0x00080768	0x2898	Zero initialised Extended Sleep retention RAM
ZI_RET21	0x82A20 EXCHANGE_MEMORY_BASE	1504 EXCHANGE_MEMORY_SIZE	Exchange memory

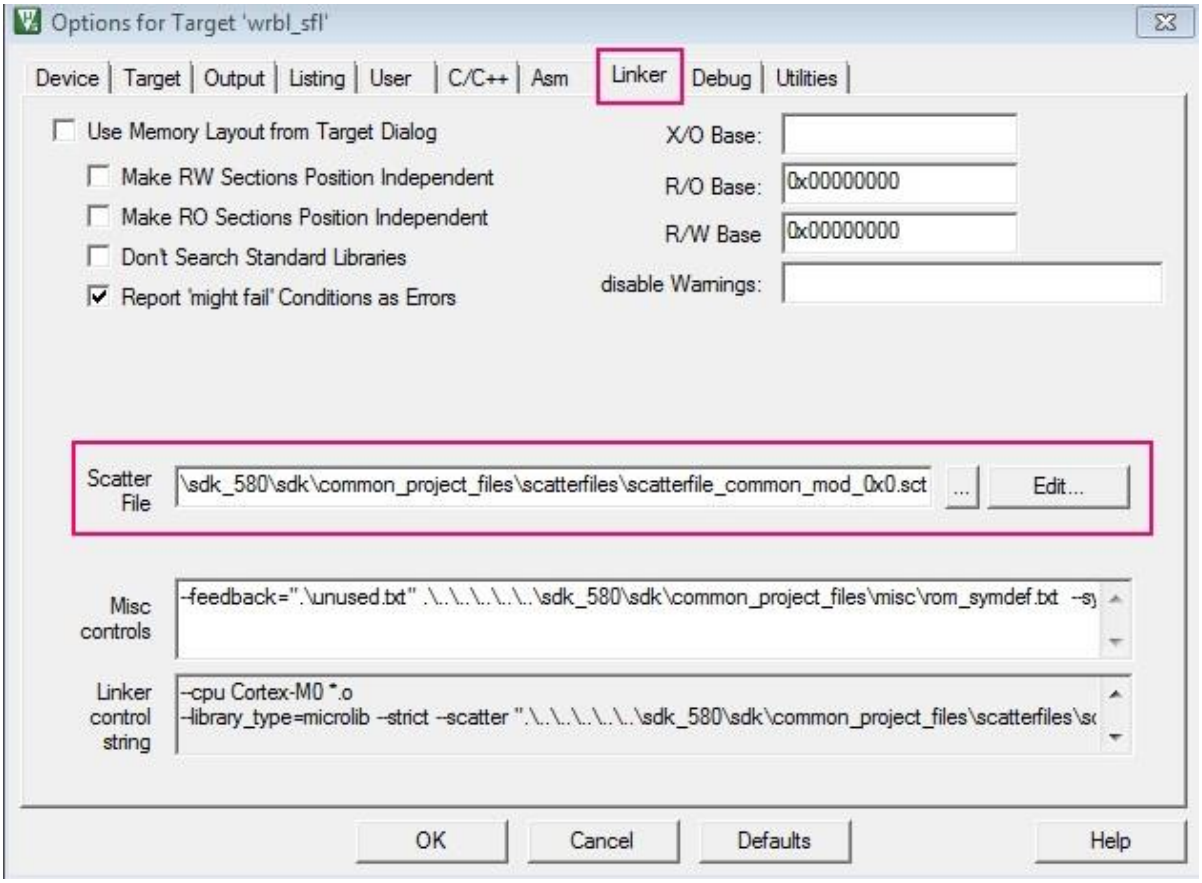

Figure 10: Access of project scatter file

6 Dialog Wearable Service

The IoT Sensor reference design contains one custom service: Dialog Wearable Service (DWS). This service includes a number of Characteristics that are listed below. The Dialog Wearable Service provides a means to:

- Transfer raw and calibrated sensor data.
- Transfer sensor fusion data.
- Configure the device, such as setting of operating parameters.
- Control the device, such as start/stop and load/store to non-volatile memory.

The details of the DW Service are outlined in Table 6.

Table 6: DW Service characteristics

Service/ Characteristic	UUID	Properties (Note 1)	Size (B)	Description
wrbl_dws_svc	2EA7-8970-7D44-44BB-B097-2618-3F40-2400	RD	16	Service attribute
wrbl_dws_accel_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2401	NTF	25	Accelerometer Report
wrbl_dws_gyro_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2402	NTF	25	Gyroscope Report
wrbl_dws_mag_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2403	NTF	25	Magnetometer Report
wrbl_dws_baro_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2404	NTF	25	Barometer Report
wrbl_dws_hum_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2405	NTF	25	Humidity Report
wrbl_dws_temp_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2406	NTF	25	Temperature Report
wrbl_dws_sensf_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2407	NTF	25	Sensor Fusion Report
wrbl_dws_feat_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2408	RD	25	Device Features
wrbl_dws_control_char	2EA7-8970-7D44-44BB-B097-2618-3F40-2409	WR	32	Control Point
wrbl_dws_control_reply_char	2EA7-8970-7D44-44BB-B097-26183F40-240A	NTF	32	Command Reply

Note 1 RD: read, WR: write, NTF: notify, IND: indicate.

6.1.1 Features report

Upon connection the central device should read the features characteristic in order to determine the capabilities and the firmware version of the device connected. Note that the IoT application is using only byte number 6 to decide whether it will include its 3D ‘d’ page option in the GUI.

Table 7: Features report structure

Offset (B)	Name	Description
0	accelerometer_en	1: Accelerometer exists
1	gyro_en	1: Gyroscope exists
2	magn_en	1: Magnetometer exists
3	pressure_en	1: Barometer exists

Offset (B)	Name	Description
4	humidity_en	1: Hygrometer exists
5	temp_en	1: Thermometer exists
6	s_fusion_en	1: Sensor Fusion exists
7 to 22	version[]	Version number, ASCII

6.1.2 Report structures for sensor fusion and raw data

The sensor and sensor fusion data are transferred over BLE, encapsulated in a report structure. These structures are described as C-code structures. Each report starts with an 8-bit ucReportId specific for each report type. The next two fields (ucSensorState and ucSensorEvent) are reserved for future use.

Table 8: Report structure for sensor data

Report ID (1 Byte)	Sensor State (1 Byte)	Sensor Event (1 Byte)	Sensor Data (N Bytes)
1 to 7	Always 0x2	Always 0x3	Depends on Sensor Type

Table 9: Report IDs for sensor and sensor fusion reports

Report ID	Report type
1	ACCELEROMETER_REPORT_ID
2	GYROSCOPE_REPORT_ID
3	MAGNETOMETER_REPORT_ID
4	PRESSURE_REPORT_ID
5	HUMIDITY_REPORT_ID
6	TEMPERATURE_REPORT_ID
7	SENSOR_FUSION_REPORT_ID
8	COMMAND_REPLY_REPORT_ID

Table 10: Report structure for accelerometer, gyroscope and magnetometer

Report field	Field size (B)	Description
ucReportId	1	1, 2 or 3 (see Table 9)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Always 3 (Update Value)
val_x	2	X axis value for the selected sensor
val_y	2	Y axis value for the selected sensor
val_z	2	Z axis value for the selected sensor

Table 11: Report structure for barometer, hygrometer and thermometer

Report field	Field size (B)	Description
ucReportId	1	4, 5 or 6 (see Table 9)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Always 3 (Update Value)
Val32	4	Sensor value

Table 12: Report structure for sensor fusion (SFL projects only)

Report field	Field size (B)	Description
ucReportId	1	7 (see Table 9)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Always 3 (Update Value)
val_w	2	W sensor fusion value
val_x	2	X sensor fusion value
val_y	2	Y sensor fusion value
val_z	2	Z sensor fusion value

6.1.3 Report structures for configuration and control

The DW Service provides the wrbl_dws_control_char (WR) and wrbl_dws_control_reply_char (NTF) characteristics for configuring and controlling the device. The device may also send unsolicited messages (such as STOP) to signal events.

Typically, the central device issues a command using the control characteristic and waits for a reply from the notification reply characteristic. The replies issued by the IoT sensor always start with byte 0x08 (COMMAND_REPLY_REPORT_ID, omitted from the following tables), followed by the Command ID and the command data.

Table 13: Report structure for commands

Report ID (1 byte)	Command ID (1 byte)	Command data (N bytes)
COMMAND_REPLY_REPORT_ID =8	See following tables.	Depending on Command ID, varies in length and field types.

6.1.3.1 Start command

Table 14: Start command

Offset (B)	Description	Value
0	Command ID	1

Reply: see Table 15.

Table 15: Start Command reply

Offset (B)	Description	Value
0	Command ID	1
1	Running Status	1: Running

6.1.3.2 Stop command

Table 16: Stop command

Offset (B)	Description	Value
0	Command ID	0

Reply: see Table 17.

Table 17: Stop Command reply

Offset (B)	Description	Value
0	Command ID	0
1	Running Status	0: Stopped

6.1.3.3 Read parameters from Flash memory

Table 18: Read Flash command

Offset (B)	Description	Value
0	Command ID	2

Reply: None.

6.1.3.4 Reset to factory defaults

Table 19: Reset to Defaults (RtD) command

Offset (B)	Description	Value
0	Command ID	3

Reply: None.

6.1.3.5 Store basic configuration in Flash memory

Table 20: Store Basic Configuration command

Offset (B)	Description	Value
0	Command ID	4

Reply: None.

6.1.3.6 Store calibration coefficients and control configuration in Flash memory

Table 21: Store Calibration and Control command

Offset (B)	Description	Value
0	Command ID	5

Reply: None.

6.1.3.7 Return running status

Table 22: Return Running Status command

Offset (B)	Description	Value
0	Command ID	6

Reply: see Table 23.

Table 23: Return Running Status reply

Offset (B)	Description	Value
0	Command ID	6
1	Running Status	0: Stopped 1: Running

6.1.3.8 Reset sensor fusion and calibration configuration

Table 24: Reset Sensor Fusion and Calibration Configuration command

Offset (B)	Description	Value
0	Command ID	7

Reply: None.

6.1.3.9 Basic configuration

Table 25: Basic Configuration command

Offset (B)	Description	Value
0	Command ID	10
1	Sensor Combination	2: Gyro 3: Accel + Gyro 5: Accel + Mag 7: Accel + Gyro + Mag
2	Accelerometer Range	0x03: 2G 0x05: 4G 0x08: 8G 0x0C: 16G
3	Accelerometer Rate	0x01: 0.78 Hz 0x02: 1.56 Hz 0x03: 3.12 Hz 0x04: 6.25 Hz 0x05: 12.5 Hz 0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz
4	Gyroscope Range	0x00: 2000 deg/s 0x01: 1000 deg/s 0x02: 500 deg/s 0x03: 250 deg/s
5	Gyroscope Rate	0x01: 0.78 Hz (Note 1) 0x02: 1.56 Hz (Note 1) 0x03: 3.12 Hz (Note 1) 0x04: 6.25 Hz (Note 1) 0x05: 12.5 Hz (Note 1) 0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz
6	Magnetometer Rate	Reserved for future use
7	Environmental Sensors Rate	1: 0.5 Hz 2: 1 Hz 4: 2 Hz
8	Sensor Fusion Rate	10: 10 Hz 15: 15 Hz 20: 20 Hz 25: 25 Hz
9	Sensor Fusion Raw Data Enable	0: Disabled 1: Enabled
10	Calibration Mode	0: None 1: Static 2: Continuous 3: One Shot

Offset (B)	Description	Value
11	Auto Calibration Mode	0: Basic 1: SmartFusion

Note 1 Can be used only in a RAW project.

Reply: None.

6.1.3.10 Read basic configuration

Table 26: Read Basic Configuration command

Offset (B)	Description	Value
0	Command ID	11

Reply: see Table 27.

Table 27: Read Basic Configuration reply

Offset (B)	Description	Value
0	Command ID	11
1	Sensor Combination	2: Gyro 3: Accel + Gyro 5: Accel + Mag 7: Accel + Gyro + Mag
2	Accelerometer Range	0x03: 2G 0x05: 4G 0x08: 8G 0x0C: 16G
3	Accelerometer Rate	0x01: 0.78 Hz 0x02: 1.56 Hz 0x03: 3.12 Hz 0x04: 6.25 Hz 0x05: 12.5 Hz 0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz
4	Gyroscope Range	0x00: 2000 deg/s 0x01: 1000 deg/s 0x02: 500 deg/s 0x03: 250 deg/s
5	Gyroscope Rate	0x01: 0.78 Hz (Note 1) 0x02: 1.56 Hz (Note 1) 0x03: 3.12 Hz (Note 1) 0x04: 6.25 Hz (Note 1) 0x05: 12.5 Hz (Note 1) 0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz
6	Magnetometer Rate	Reserved for future use
7	Environmental Sensors Rate	1: 0.5 Hz 2: 1 Hz 4: 2 Hz
8	Sensor Fusion Rate	10: 10 Hz 15: 15 Hz 20: 20 Hz 25: 25 Hz

Offset (B)	Description	Value
9	Sensor Fusion Raw Data Enable	0: Disabled 1: Enabled
10	Calibration Mode	0: None 1: Static 2: Continuous 3: One Shot
11	Auto Calibration Mode	0: Basic 1: SmartFusion

Note 1 Can be used only in a RAW project.

6.1.3.11 Set sensor fusion coefficients

Table 28: Set Sensor Fusion Coefficients command

Offset (B)	Description	Value
0	Command ID	12
1	BETA A LSB	Sensor Fusion Beta A Gain (LSB)
2	BETA A MSB	Sensor Fusion Beta A Gain (MSB)
3	BETA M LSB	Sensor Fusion Beta M Gain (LSB)
4	BETA M MSB	Sensor Fusion Beta M Gain (MSB)
5:8	TEMPERATURE_REPORT_ID	Reserved

Reply: None.

6.1.3.12 Read sensor fusion coefficients

Table 29: Read Sensor Fusion Coefficients command

Offset (B)	Description	Value
0	Command ID	13

Reply: see Table 30.

Table 30: Read Sensor Fusion Coefficients reply

Offset (B)	Description	Value
0	Command ID	13
1	BETA A LSB	Sensor Fusion Beta A Gain (LSB)
2	BETA A MSB	Sensor Fusion Beta A Gain (MSB)
3	BETA M LSB	Sensor Fusion Beta M Gain (LSB)
4	BETA M MSB	Sensor Fusion Beta M Gain (MSB)
5:8	TEMPERATURE_REPORT_ID	Reserved

6.1.3.13 Set calibration coefficients

Table 31: Set Calibration Coefficients command

Offset (B)	Description	Value
0	Command ID	14
1	Sensor Type	2: Magnetometer

Offset (B)	Description	Value
2	Q Format	Integer value
3:8	Offset Vector (3 x int16)	Integer value
9:26	Matrix 3 x 3 x int16	Signed fixed point value

Reply: None.

6.1.3.14 Read calibration coefficients

Table 32: Read Calibration Coefficients command

Offset (B)	Description	Value
0	Command ID	15

Reply: see Table 33.

Table 33: Read Calibration Coefficients reply

Offset (B)	Description	Value
0	Command ID	15
1	Sensor Type	Magnetometer = 2
2	Q Format	Integer value
3:8	Offset Vector 3x1 int16	Integer value
9:26	Matrix 3x3 int16	Signed fixed point value

6.1.3.15 Calibration control

Table 34: Set Calibration Control Flags command

Offset (B)	Description	Value
0	Command ID	16
1	Sensor Type	2: Magnetometer
2:3	Calibration Control Flags	Byte 2: see Table 35 Byte 3: see Table 36
4:15	Calibration Parameters	See Table 37

Table 35: Calibration Control Flags (byte 2)

Calibration mode	Bit 7 reserved	Bit 6 reserved	Bit 5 offset apply	Bit 4 matrix apply	Bit 3 offset update	Bit 2 matrix update	Bit 1 init from static	Bit 0 converged (read only)
Static	X	X	1: Yes	1: Yes	0: No	0: No	0: No	1: Yes
Basic Auto	X	X	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes
SmartFusion Auto	X	X	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes

Table 36: Calibration Control Flags (byte 3)

Calibration mode	Bit 7 reserved	Bit 6 reserved	Bit 5 reserved	Bit 4 reserved	Bit 3 reserved	Bit 2 reserved	Bit 1 reserved	Bit 0 settled
Static	X	X	X	X	X	X	X	0: No

Calibration mode	Bit 7 reserved	Bit 6 reserved	Bit 5 reserved	Bit 4 reserved	Bit 3 reserved	Bit 2 reserved	Bit 1 reserved	Bit 0 settled
Basic Auto	X	X	X	X	X	X	X	1: Yes
SmartFusion Auto	X	X	X	X	X	X	X	1: Yes

Table 37: Calibration Parameters

Calibration mode	Byte 4:5	Byte 6:7	Byte 8:9	Byte 10:15
Static	Reserved	Reserved	Reserved	Reserved
Basic	ref_mag	mag_range	Reserved	Reserved
SmartFusion	ref_mag	mag_range	mu	Reserved

6.1.3.16 Read calibration control

Table 38: Read Calibration Control command

Offset (B)	Description	Value
0	Command ID	17

Reply: see Table 39.

Table 39: Read Calibration Control reply

Offset (B)	Description	Value
0	Command ID	17
1	Sensor Type	2: Magnetometer
2:3	Calibration Control Flags	Byte 2: see Table 35 Byte 3: see Table 36
4:15	Calibration Parameters	See Table 37

6.1.3.17 Fast accelerometer calibration

Table 40: Fast Accelerometer Calibration command

Offset (B)	Description	Value
0	Command ID	18

Reply: see Table 41.

Table 41: Fast Accelerometer Calibration reply

Offset (B)	Description	Value
0	Command ID	18
1	Fast Calibration Status	0: Stopped 1: Started

7 Operation overview

7.1 Connection parameters

Upon connection the device issues a parameter update request to the central device with the following parameters (`user_connection_param_conf`, user_config.h):

```
Connection interval min/max: 20 ms/40 ms
Connection Latency: 4 (events missed)
Supervision timeout: 2 s
```

7.2 Advertising and scanning

The format of the advertising data is listed in Table 42. For more information see Ref. [7].

Table 42: Advertising and Scan Response data

Data type	Length (B)	Value	Description
0x01: Flags	2	0x06	General Discoverable, BR-EDR not supported.
0x02: List of 16 bit UUIDs	3	0xA72E	Part of the UUID. The central application uses this value to filter the IoT devices.
0x09: Complete Local Name	11	IoT-DK-SFL or IoT-DK-RAW	
0xFF: Manufacturer Specific Data (Scan Response)	6	0xD200, <X1,X2,X3>	0xD200: Dialog Semiconductor manufacturer ID <X1,X2,X3>: Last three bytes of the BD address. This is used when the actual BD address is not available to the central application.

The advertising interval is 200 ms by default (`FAST_ADV_INTERVAL`).

7.3 General operation

Upon application initialisation the function `user_wrbl_app_on_init()` initialises the sensors to low power mode and the default advertising starts for `FAST_ADV_TO` seconds (default value: 60 s). When the timeout occurs without connection, the function `user_app_on_adv_undirect_complete()` sets the device in a low power mode, where the CPU is in Sleep mode and the accelerometer in low power anymotion detection mode. If a motion is detected, exceeding a specific threshold, then the CPU wakes up and starts advertising.

When the device is connected to a central device the function `user_on_connection()` executes all necessary initialisations:

- Load the sensor configuration from Flash memory.
- Initialise the sensors.
- Initialise the sensor operation control and data structures.
- Start BLE parameter updating.

During this initialisation the run-time FSM polling timer is started. The `user_wrbl_timer_handler()` function is the timer callback function that runs every 500 ms (`WRBL_FSM_INTV_MS10`). This handler has four system states in connected mode (see Figure 11):

- `ACQUISITION_STATE_IDLE`: In this state the sensors are in sleep mode. The BLE is in connected state but no sensor data acquisition is performed.
- `ACQUISITION_STATE_INIT`: When a START command is issued, all the sensors, calibration and data are initialised. The DA14583 interrupt engine is initialised to serve interrupts originating from the BMI160 accelerometer/gyro module. The code in this state is executed once and then switches to the `ACQUISITION_STATE_RUN` state.

- ACQUISITION_STATE_RUN: This is the normal running state when the system is fully operational. This state also controls the acquisition of the environmental sensors contained in BME280.
- ACQUISITION_STATE_STOP: When a STOP command is issued, the code in this state is executed once before returning to ACQUISITION_STATE_IDLE state. In this state the sensor operation and interrupts are suspended and the calibration parameters are saved.

In ACQUISITION_STATE_RUN state, interrupts from the BMI160 are serviced by the wkup_intr_1_cb() interrupt callback handler. The SFL Project and the RAW Project use different interrupt callbacks.

In the SFL Project the interrupt service routine receives a BMI160 FIFO 'watermark' (WM) interrupt, which signals that the FIFO level has reached a certain number of bytes. Then the CPU burst reads the SPI peripheral in order to empty the FIFO and store the sensor data in CPU RAM. Next the CPU sends a DWS_SENSOR_DATA_RDY_IND message in order to continue processing of data to user space. The magnetometer operates in forced mode, synchronised with this interrupt. This means that the magnetometer rate is the same as the WM level interrupt rate. The WM interrupt rate is calculated in user_calc_wm_level() (section 7.3.1) according to the desired Sensor Fusion Rate. So in this mode the Sensor Fusion data generation rate, the WM Level Interrupt rate and the Magnetometer rate are the same.

In the RAW Project the interrupt service routine receives a BMI160 DATA READY interrupt, which signals that a set of data is ready. The CPU then reads the data from the sensors and sends it to user space using a DWS_SENSOR_DATA_RDY_IND message. The magnetometer is synchronised with this interrupt via the Accelerometer Data Ready flag. So the Magnetometer rate is the same as Accelerometer rate.

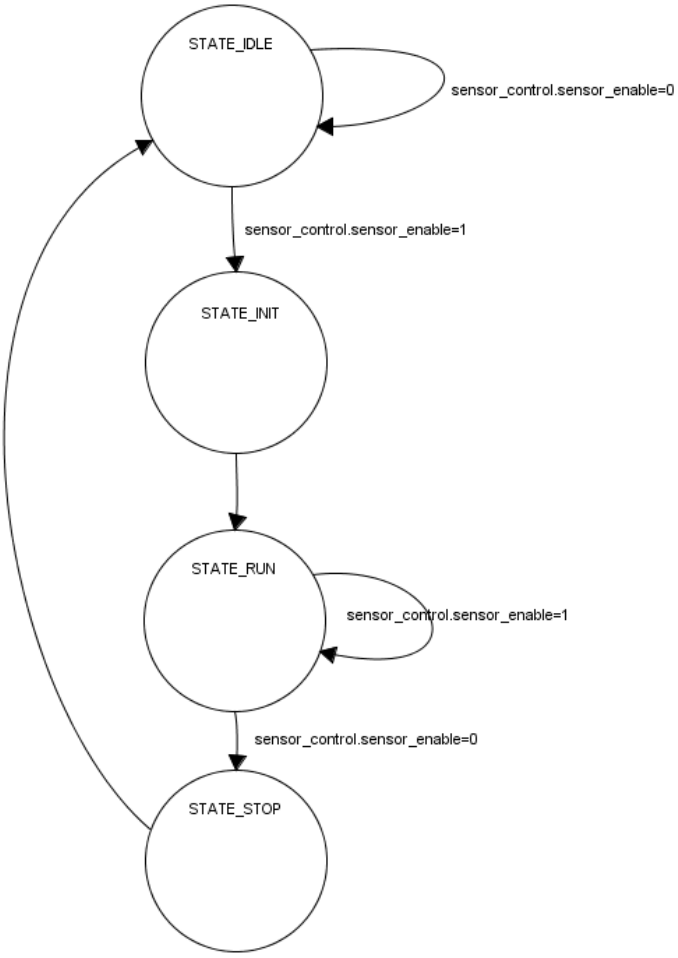

Figure 11: IoT DK system states

In the SFL Project the raw sensor data, that includes accelerometer, gyroscope and magnetometer data, is processed by `user_process_fifo_frames()` after reception of a `DWS_SENSOR_DATA_RDY_IND` message. The data is read from the `accel_gyro_mag_data` array and passed to the Sensor Fusion update function `user_sensor_fusion_process()`. After the FIFO array has been processed and the Sensor Fusion algorithm has been updated, a part of the raw data is sent over BLE using function `user_add_sensor_report_acc_gyro_mag()`. Finally, the Sensor Fusion updated information is sent over BLE using the function `user_add_sensor_report_sensor_fusion()`.

In the RAW Project the raw sensor data, that includes accelerometer, gyroscope and magnetometer data, is processed by `user_process_raw_samples()` after reception of a `DWS_SENSOR_DATA_RDY_IND` message. This function sends all the received sensor data over BLE to the central device using the function `user_add_sensor_report_acc_gyro_mag()`.

In the user space processing of both projects, every magnetometer sample is passed through the `cal_process()` function, which processes the sample according to the selected calibration mode.

The device is controlled by the central application using the DWS profile control point and control point reply. The `user_dws_write_ind_handler()` function handles the reception of commands, and a typical reply is sent over BLE using the `user_send_command_reply()` function.

7.3.1 FIFO watermark level calculation

The WM interrupt rate is calculated in `user_calc_wm_level()` according to the desired Sensor Fusion Rate using the following calculation:

$$Watermark\ Level = floor[\frac{6 \times (ODR_{acc} + ODR_{gyr}) + MAX(ODR_{acc}, ODR_{gyr})}{4 \times ODR_{sf}}]$$

The Watermark Level provided to the BMI160 driver is expressed in multiples of 4 bytes. The desired sensor fusion data rate in combination with data rates of accelerometer and gyroscope determines the actual sensor fusion data rate. Since the set point of watermark level is a multiple of 4 bytes and each sensor data FIFO frame size is 6 bytes, the desired sensor fusion data rate cannot be met when the combination of the result of this function is not an integer.

For example when:

$ODR_{sf} = 15\ Hz;\ ODR_{acc} = 100\ Hz;\ ODR_{gyr} = 100\ Hz$

then:

$$Watermark\ Level = floor\left(\frac{86.6}{4}\right) = 21$$

and the actual sensor fusion output data rate is:

$$ODR_{sf}(actual) = \frac{1000}{\left(10\ ms * \left(floor\left[\frac{4 \times 21}{2 \times 6}\right]\right)\right)} = \frac{1000}{70\ ms} = 14.28\ Hz$$

7.4 Non-volatile Flash memory

The DA14583 contains 128 kB of Flash memory that holds three sections:

1. IoT DK program code at address 0x00000.
2. Basic configuration information at address 0x18000.
3. Calibration configuration at address 0x19000.

Sections 2 and 3 contain a magic number that is tested every time the device boots. If the magic number is not present, the section is overwritten with the default values. The fields and default values are listed in Table 43 and Table 44.

Table 43: Basic configuration (Flash address 0x18000)

Offset (B)	Size (B)	Parameter name	Values	Default value/ meaning
0	2	Magic number	0x84DA 0x32CB	0x84DA: SFL 0x32CB: RAW
2	1	sensor_combination	See Table 25	0xF: all sensors enabled
3	1	accel_range	See Table 25	0x03: 2G
4	1	accel_rate	See Table 25	0x06: 25 Hz
5	1	gyro_range	See Table 25	0x00: 2000 deg/s
6	1	gyro_rate	See Table 25	0x06: 25 Hz
7	1	mag_rate	Not used	Not applicable
8	1	env_rate	See Table 25	0x01: 0.5 Hz
9	1	sfl_rate	See Table 25	10: 10 Hz
10	1	sfl_raw_en	See Table 25	1: Enabled
11	1	cal_mode	See Table 25	1: Static
12	1	auto_cal_mode	See Table 25	0: Basic
13	4	Reserved	-	Not applicable

Table 44: Calibration configuration (Flash address 0x19000)

Offset (B)	Size (B)	Parameter name	Values	Default value/ meaning
0	2	Magic number	0x84DA 0x32CB	0x84DA: SFL 0x32CB: RAW
2	2	beta_a	Unsigned Q15 fixed-point format in range 0x0000 (0) to 0x8000 (1.0).	0x028F (= 655): 0.02
4	2	beta_m	Unsigned Q15 fixed-point format in range 0x0000 (0) to 0x8000 (1.0).	0x028F (= 655): 0.02
6	4	Reserved	-	Not applicable
Static calibration coefficients				
10	1	Sensor Type	2	2: magnetometer
11	1	q_format	Integer in range 0 to 15.	14: Q format of matrix
12	6	offset_vector	Integers in range -32768 to 32767.	3 x 0x0000
18	18	matrix	Signed fixed-point format in range 0x8000 to 0x7FFF.	9 x 0x0000
Auto calibration coefficients				
36	1	Sensor Type	2	2: magnetometer
37	1	q_format	Integer in range 0 to 15.	14: Q format of matrix
38	6	offset_vector	Integers in range -32768 to 32767.	3 x 0x0000

Offset (B)	Size (B)	Parameter name	Values	Default value/ meaning
44	18	matrix	Signed fixed-point format in range 0x8000 to 0x7FFF.	9 x 0x0000
SmartFusion calibration coefficients				
62	1	Sensor Type	2	2: magnetometer
63	1	q_format	Integer in range 0 to 15.	14: Q format of matrix
64	6	offset_vector	Integers in range -32768 to 32767.	3 x 0x0000
70	18	matrix	Signed fixed-point format in range 0x8000 to 0x7FFF.	9 x 0x0000
Static calibration parameters and flags				
88	1	Sensor Type	2	2: magnetometer
89	2	Flags	See Table 35	0x000C
91	2	ref_mag	Not used	0
93	2	mag_range	Not used	0
95	2	mu	Not used	0
97	6	Reserved	-	Not applicable
Auto calibration parameters and flags				
103	1	Sensor type	2	2: magnetometer
104	2	Flags	See Table 35	0x007C
106	2	ref_mag	Unsigned integer in range 0 to 32767.	500
108	2	mag_range	Unsigned Q15 fixed-point format in range 0x0000 (0) to 0x8000 (1.0).	0x3333 (= 13107): 0.4
110	2	mu	Not used	0
112	6	Reserved	-	Not applicable
SmartFusion auto calibration parameters and flags				
118	1	Sensor type	2	2: magnetometer
119	2	Flags	See Table 35	0x007C
121	2	ref_mag	Unsigned integer in range 0 to 32767.	500
123	2	mag_range	Unsigned Q15 fixed-point format in range 0x0000 (0) to 0x8000 (1.0).	0x3333 (= 13107): 0.4
125	2	mu	Signed Q19 fixed-point format in range 0x8000 (-0.0625) to 0x0000 (0).	0xFE66 (= -410): -7.82×10^{-4}
127	6	Reserved	-	Not applicable

8 Code overview

The DA14568x SDK5 provides a set of hook functions in the file `user_config.h` in order to control key functions of system operation, such as application entry, advertising start and completion, connection and disconnection.

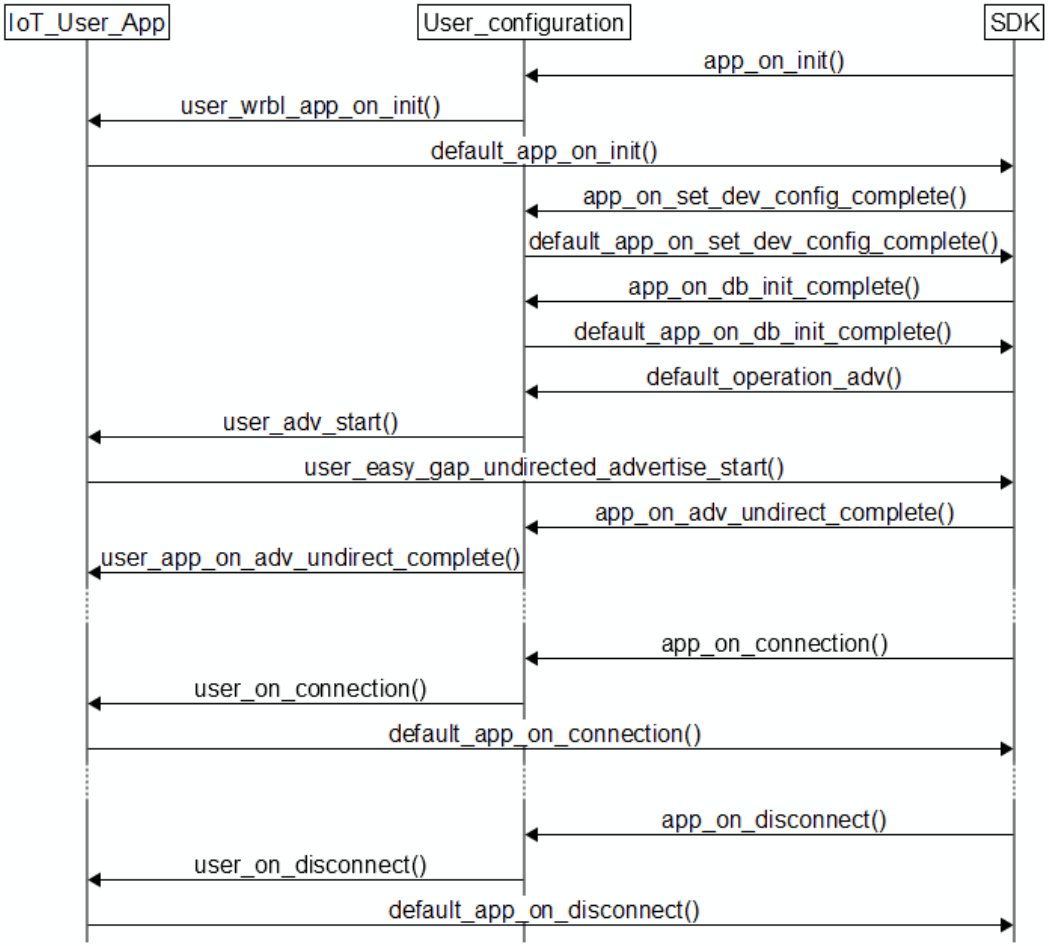

Figure 12: IoT sensor application flow

8.1 Entry points

8.1.1 user_wrbl_app_on_init()

This function is the entry point of the application. It initialises the sensors to the default unconnected behaviour for advertising.

8.2 Advertising

8.2.1 user_adv_start()

This function starts the advertising with a specific behaviour:

- Initialises the advertising timer to 60 s (`FAST_ADV_TO`). When this timer expires, the function `user_adv_timer_handler()` will be executed to terminate advertising.
- Initialises the LED blink timer.
- Sets the appropriate SCAN information using `user_easy_gap_undirected_advertise_start()`.

8.2.2 user_add_adv_info()

Modifies the default Advertise and Scanning information. Advertising parameters are listed in section 7.2.

8.2.3 user_app_on_adv_undirect_complete()

This function is called after expiration of the advertising timer. The accelerometer is configured for low power operation with the ‘anymotion’ interrupt function set. All other sensors are put in Sleep mode. In this setup the CPU is in Extended Sleep mode and wakes up only when it receives an interrupt from BMI160. Then the interrupt handler `wkup_intr_non_connected_cb()` is executed, which initiates advertising.

8.3 Connection and disconnection

8.3.1 user_on_connection()

This is the connection callback function, which executes all the necessary procedures in order to set up the system for run-time operation:

- Stops the advertising timer.
- Starts the blink timer to indicate connection event.
- Sets the sensors to the default parameters and then suspends the sensors.
- Starts the BLE parameter update procedure.
- Starts the main FSM.

8.3.2 user_on_disconnect()

This is the disconnection callback function, which executes all the necessary procedures in order to suspend run-time operation and re-initiate advertising.

8.4 Timer handlers

The system implements three timers, one for advertising termination after timeout, one for LED indication handling and one for handling the main system states during connection.

8.4.1 user_adv_timer_handler

The advertising timer handler is called after a timeout of `FAST_ADV_TO` (60 s) to terminate advertising.

8.4.2 user_led_timer_handler

The LED timer handler handles the LED blinking that signals advertising and connection. During advertising the LED repeatedly blinks with the `ADVERTISE_LED_ON_TIME` on time (default: 100 ms) and the `ADVERTISE_LED_OFF_TIME` (default: 900 ms). When a connection is established the LED blinks `CONNECTED_LED_BLINKS` number of times (default: 10) with on time `CONNECTED_LED_ON_TIME` (100 ms) and off time `CONNECTED_LED_OFF_TIME` (100 ms).

8.4.3 user_wrbl_timer_handler

This is the main FSM that handles the system states when the device is connected. Its operation is described in detail in section 7.3.

8.5 Interrupt handlers

The interrupt handlers that are specific to IoT DK all handle the external wakeup via port P0_6. This pin is connected to the BMI160 interrupt output INT1.

8.5.1 wkup_intr_non_connected_cb()

This is the interrupt handler that is called when the ‘anymotion’ interrupt of BMI160 is activated in non-connected sleep state. It wakes up the controller and restarts advertising (see section 8.2.3).

8.5.2 wkup_intr_1_cb() - SFL

In the SFL project this interrupt handler is called upon a BMI160 ‘watermark’ interrupt. This signals that the BMI160 FIFO level has exceeded the `sensor_config.fifo_wm_level` number of bytes. The CPU then empties the FIFO and sends a `DWS_SENSOR_DATA_RDY_IND` message to the user space for further processing.

8.5.3 wkup_intr_1_cb() - RAW

In the RAW project this interrupt handler is called upon a BMI160 ‘data ready’ interrupt. This signals that accelerometer or gyroscope data is available for reading. The FIFO mode is not used in this version. The CPU then reads the data and sends a `DWS_SENSOR_DATA_RDY_IND` message to the user space for further processing.

8.6 Sensor acquisition

The system implements different interfaces for the configuration, control and data acquisition from the sensor modules (BMI160, BMM150 and BME280), which are used to implement the upper layer of the sensor-data-oriented application functions. The driver architecture provided in the IoT SDK is presented in Figure 13.

The following sections list the various functions of the BMI160, BMM150 and BME280 sensor module driver libraries. For each sensor module, the corresponding driver provides the necessary abstraction to the application, when access for control and data acquisition is required. Although the sensor modules provide different functionality and corresponding options for the type of sensor data as well as the way sensor data is acquired, all sensor drivers have the following common functionality:

- INIT-CONFIG: Initialising, setting the power mode state and configuring the operation mode of the sensor module.
- SAMPLE: Reading sensor data produced by the sensor module.
- CMD: Sending commands to the sensor module.
- STATE: Getting the status of the sensor module.
- CONFIG: Setting/getting specific setting options for the sensor module.
- LOW-ACCESS: Setting/getting register values provided by the sensor module.

The source code for the sensor drivers is located in folder `wrbl_ref/projects/target_apps/wrbl/common/src/driver`.

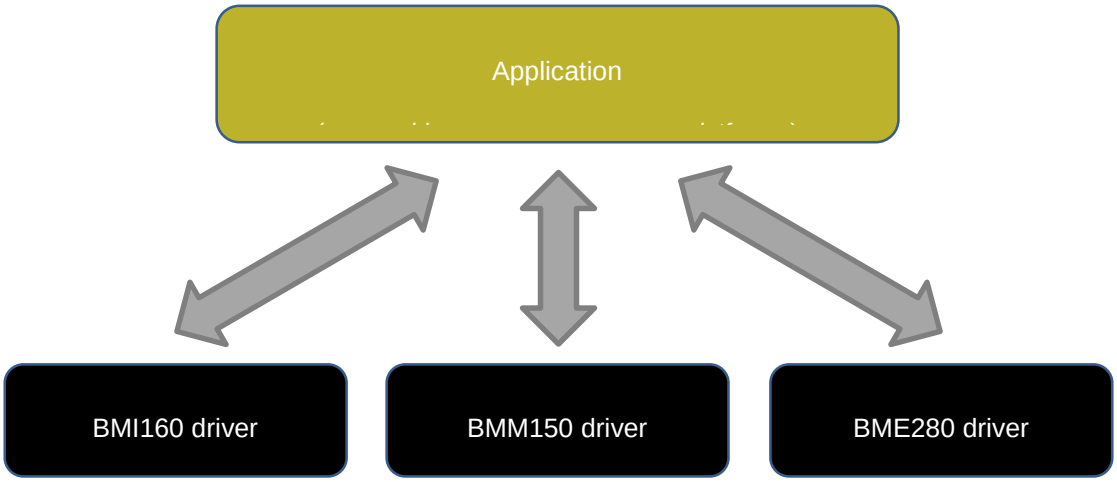

Figure 13: Sensor drivers

8.6.1 BMI160 sensor module

The following sections list the functions of the BMI160 driver library used by the application level functions presented in section 8.6. General guidelines for using this sensor driver are as follows:

- Initialize the module using `bmi160_initialize_sensor()`.
- Set the power state of the module by sub-setting the power state of each included sensor (i.e. accelerometer, gyroscope) using `bmi160_set_sensor_state()`.
- Configure the module to operate in a predefined mode using `bmi160_config_running_mode()`.
- Read inertial sensor (`sensor = accel, gyro`) data using the `bmi160_read_[sensor]_[x|y|z]()` set of functions.
- Read temperature sensor data using `bmi160_read_temperature()`.
- Read step counter data using `bmi160_read_step_counter()`.
- When the sensor module is configured with a mode that uses the internal FIFO to store data, read a set of sensor data using `bmi160_read_fifo_data()`.
- Make appropriate decisions by getting the status of the sensor module (`state_feature = accel_state, gyro_state, drdy_accel_stat, drdy_gyro_stat, etc`.) using the `bmi160_get_[state_feature]()` set of functions.
- Configure differently or extend the sensor module's predefined operating mode in terms of specific setting features (`setting_feature = accel_out_data_rate, accel_bandwidth, accel_range, etc`.) using the `bmi160_set_[setting_feature]()` set of functions.
- Get the current configuration of the sensor module in terms of a specific setting feature using the `bmi160_get_[setting_feature]()` set of functions.
- Access the sensor module's registers (`reg`) when low level control and data handling is required, using the `bmi160_[set/get]_reg_[reg]()` set of functions.

8.6.1.1 bmi160_initialize_sensor()

Function name	**int8_t bmi160_initialize_sensor()**
Function description	(INIT-CONFIG) Initializes the BMI160 sensor module, creating the appropriate instance handle for the module which includes references to the spi bus access functions used for reading/writing values to the registers, as well as setting the state of accelerometer and gyroscope sensors to ACC_NORMAL and GYR_NORMAL, respectively.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.2 bmi160_set_sensor_state()

Function name	**int8_t bmi160_set_sensor_state(uint8_t state)**
Function description	(INIT-CONFIG) Sets the state of a sensor or a set of sensors integrated in BMI160 sensor module.
Parameters	state : the state to set for a sensor or a set of sensors ACC_SUSPEND, ACC_NORMAL, ACC_LOWPOWER, GYR_SUSPEND, GYR_NORMAL, GYR_FASTSTARTUP, ACC_ADVLP_GYR_ADVLP, ACC_IMU_GYR_IMU, ACC_IMU_FIFO_GYR_IMU_FIFO
Return values	0 for successful execution; ≠0 for error
Notes	The ACC_ADVLP_GYR_ADVLP, ACC_IMU_GYR_IMU and ACC_IMU_FIFO_GYR_IMU_FIFO options refer to states of sensors also configured with a predefined running mode. On successful execution it is introduced a ACC_MODE_TRANSITION_DELAY or a GYR_MODE_TRANSITION_DELAY delay.

8.6.1.3 bmi160_config_running_mode()

Function name	**int8_t bmi160_config_running_mode(uint8_t v_running_mode_u8)**
Function description	(INIT-CONFIG) Sets the running mode of the BMI160 sensor module.
Parameters	v_running_mode_u8 : the running mode to set for a set of sensors STANDARD_UI_IMU_FIFO, STANDARD_UI_IMU, STANDARD_UI_ADVANCEPOWERSAVE
Return values	0 for successful execution; ≠0 for error
Notes	The STANDARD_UI_IMU_FIFO mode enables the use of FIFO for storing accelerometer and gyroscope sensor data at the predefined rates of ACCEL_DATA_RATE and GYRO_DATA_RATE, respectively, as well as interrupt triggering when FIFO data size reaches the predefined level of FIFO_WATERMARK_LEVEL. The STANDARD_UI_IMU mode enables the regular operation of accelerometer and gyroscope sensors, that is, sampling data at the predefined rates of ACCEL_DATA_RATE and GYRO_DATA_RATE, respectively. The STANDARD_UI_ADVANCEPOWERSAVE mode enables the low-power operation of accelerometer and gyroscope sensors, that is, sampling data at the predefined rates of ACCEL_DATA_RATE and GYRO_DATA_RATE, respectively, while suspending gyroscope operation when no motion is detected and activating it when any (significant) motion is detected. On successful execution it is introduced a ACC_MODE_TRANSITION_DELAY or a GYR_MODE_TRANSITION_DELAY delay.

8.6.1.4 bmi160_read_fifo_data()

Function name	**int8_t bmi160_read_fifo_data(uint8_t *v_fifo_data_u8, uint16_t v_len_u16)**
Function description	(SAMPLE) Reads sensor data stored in the FIFO.
Parameters	v_fifo_data_u8 : the buffer to transfer the FIFO sensor data v_len_u16 : the size of data (in bytes) to transfer
Return values	0 for successful execution; ≠0 for error
Notes	The maximum size of FIFO in bytes is 1024 (FIFO_FRAME)

8.6.1.5 bmi160_get_sensor_time()

Function name	**int8_t bmi160_get_sensor_time(uint32_t* v_sensor_time_u32)**
Function description	(STATE) Reads sensor time provided by the BMI160 sensor module in units of 0.039ms.
Parameters	v_sensor_time_u32 : the sensor time
Return values	0 for successful execution; ≠0 for error
Notes	1024 is 40ms, 256 is 10ms, 64 is 2.5ms, 16 is 0.625ms etc.

8.6.1.6 bmi160_read_accel_xyz()

Function name	**int8_t bmi160_read_accel_xyz(struct bmi160_accel_t *accel_xyz)**
Function description	(SAMPLE) Reads accelerometer xyz-axis data.
Parameters	accel_xyz : the accelerometer xyz-axis data
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.7 bmi160_read_gyro_xyz()

Function name	**int8_t bmi160_read_gyro_xyz(struct bmi160_gyro_t *gyro_xyz)**
Function description	(SAMPLE) Reads gyroscope xyz-axis data.
Parameters	gyro_xyz : the gyroscope xyz-axis data
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.8 bmi160_read_temperature()

Function name	**int8_t bmi160_read_temperature(int16_t *v_temp_s16)**
Function description	(SAMPLE) Reads temperature of BMI160 sensor module data.
Parameters	v_temp_s16 : the temperature data
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.9 bmi160_read_step_counter()

Function name	`int8_t bmi160_read_step_counter(int16_t *v_step_cnt_s16)`
Function description	(SAMPLE) Reads the number of detected steps.
Parameters	`v_step_cnt_s16` : the step counter data
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.10 bmi160_flush_fifo()

Function name	`int8_t bmi160_flush_fifo()`
Function description	(CMD) Flushes FIFO content.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.11 bmi160_reset_intrs()

Function name	`int8_t bmi160_reset_intrs()`
Function description	(CMD) Resets interrupts.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.12 bmi160_clear_step_counter()

Function name	`int8_t bmi160_clear_step_counter()`
Function description	(CMD) Clears step counter data.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.13 bmi160_soft_reset()

Function name	`int8_t bmi160_soft_reset()`
Function description	(CMD) Performs a soft reset to the BMI160 sensor module.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	On successful execution it is introduced a `GYR_MODE_TRANSITION_MAX_DELAY` delay.

8.6.1.14 bmi160_get_chip_id()

Function name	**int8_t bmi160_get_chip_id(uint8_t* v_chip_id_u8)**
Function description	(STATE) Reads the chip id of the BMI160 sensor module.
Parameters	v_chip_id_u8 : the chip id
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.15 bmi160_get_accel_state()

Function name	**int8_t bmi160_get_accel_state(uint8_t *v_accel_power_mode_state_u8)**
Function description	(STATE) Reads the power mode state of the accelerometer.
Parameters	v_accel_power_mode_state_u8 : the accelerometer power mode state ACC_SUSPEND, ACC_NORMAL, ACC_LOWPOWER
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.16 bmi160_get_gyro_state()

Function name	**int8_t bmi160_get_gyro_state(uint8_t *v_gyro_power_mode_state_u8)**
Function description	(STATE) Reads the power mode state of the gyroscope.
Parameters	v_gyro_power_mode_state_u8 : the gyroscope power mode state GYR_SUSPEND, GYR_NORMAL, GYR_LOWPOWER
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.17 bmi160_get_drdy_accel_stat()

Function name	**int8_t bmi160_get_drdy_accel_stat(uint8_t* v_data_rdy_u8)**
Function description	(STATE) Reads the status of data ready (drdy) status bit, indicating whether a new accelerometer sample value has been set to the corresponding data register.
Parameters	v_data_rdy_u8 : the value of the accelerometer drdy interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.18 bmi160_get_drdy_gyro_stat()

Function name	**int8_t bmi160_get_drdy_gyro_stat(uint8_t* v_data_rdy_u8)**
Function description	(STATE) Reads the status of data ready (drdy) status bit, indicating whether a new gyroscope sample value has been set to the corresponding data register.
Parameters	v_data_rdy_u8 : the value of the gyroscope drdy interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.19 bmi160_get_step_stat()

Function name	`int8_t bmi160_get_step_stat(uint8_t* v_step_intr_u8)`
Function description	(STATE) Reads the status of the step detection interrupt status bit, indicating the detection of a step.
Parameters	`v_step_intr_u8` : the value of the step detection interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.20 bmi160_get_significant_motion_stat()

Function name	`int8_t bmi160_get_significant_motion_stat(uint8_t* v_significant_motion_intr_u8)`
Function description	(STATE) Reads the status of the significant motion interrupt status bit, indicating the detection of a significant motion event.
Parameters	`v_significant_motion_intr_u8` : the value of the significant motion interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.21 bmi160_get_any_motion_stat()

Function name	`int8_t bmi160_get_any_motion_stat(uint8_t* v_any_motion_intr_u8)`
Function description	(STATE) Reads the status of the any-motion interrupt status bit, indicating the detection of an ‘anymotion’ event.
Parameters	`v_any_motion_intr_u8` : the value of the any-motion interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.22 bmi160_get_drdy_stat()

Function name	`int8_t bmi160_get_drdy_stat(uint8_t* v_data_rdy_intr_u8)`
Function description	(STATE) Reads the status of the data ready (drdy) status bit, indicating whether a new sensor sample value has been set to the corresponding data register.
Parameters	`v_data_rdy_intr_u8` : the value of the drdy interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.23 bmi160_get_fifo_full_stat()

Function name	`int8_t bmi160_get_fifo_full_stat(uint8_t* v_fifo_full_intr_u8)`
Function description	(STATE) Reads the status of the FIFO full interrupt status bit, indicating whether data in FIFO have reached the maximum size.
Parameters	`v_fifo_full_intr_u8` : the value of the FIFO full interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.24 bmi160_get_fifo_watermark_stat()

Function name	**int8_t bmi160_get_fifo_watermark_stat(uint8_t* v_fifo_wm_intr_u8)**
Function description	(STATE) Reads the status of the FIFO watermark (wm) interrupt status bit, indicating whether data in FIFO have reached the level indicated by FIFO_WATERMARK_LEVEL.
Parameters	v_fifo_wm_intr_u8 : the value of the FIFO wm interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.25 bmi160_get_no_motion_stat()

Function name	**int8_t bmi160_get_no_motion_stat(uint8_t* v_no_motion_intr_u8)**
Function description	(STATE) Reads the status of the no-motion interrupt status bit, indicating the detection of a no-motion event.
Parameters	v_no_motion_intr_u8 : the value of the no-motion interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.26 bmi160_[set/get]_accel_out_data_rate()

Function name	**int8_t bmi160_set_accel_out_data_rate(uint8_t v_output_data_rate_u8)** **int8_t bmi160_get_accel_out_data_rate(uint8_t* v_output_data_rate_u8)**
Function description	(CONFIG) Sets/gets the accelerometer output data rate (odr), that is, the sampling data rate.
Parameters	v_output_data_rate_u8 : the value of the accelerometer output data rate BMI160_ACCEL_OUTPUT_DATA_RATE_RESERVED, BMI160_ACCEL_OUTPUT_DATA_RATE_0_78HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_1_56HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_3_12HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_6_25HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_12_5HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_25HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_50HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_100HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_200HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_400HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_800HZ, BMI160_ACCEL_OUTPUT_DATA_RATE_1600HZ
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.27 bmi160_[set/get]_accel_undersampling_enable()

Function name	**int8_t bmi160_set_accel_undersampling_enable(uint8_t v_accel_under_sampling_u8)** **int8_t bmi160_get_accel_undersampling_enable(uint8_t* v_accel_under_sampling_u8)**
Function description	(CONFIG) Sets/gets the accelerometer under-sampling mode (acc_us). If under-sampling is disabled the filter accelerometer digital filter configuration is enabled, and vice versa.
Parameters	v_accel_under_sampling_u8 : the value of the accelerometer under-sampling mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	If digital filter configuration is enabled (acc_us=0), the options are: OSR4 mode, OSR2 mode, normal mode, CIS mode If under-sampling mode is enabled (acc_us=1), the options are: averaging 1 (AVG1 or no-averaging), 2 (AVG2), 4 (AVG4), 8 (AVG8), 16 (AVG16), 32 (AVG32), 64 (AVG64), 128 (AVG128) samples

8.6.1.28 bmi160_[set/get]_accel_bandwidth()

Function name	**int8_t bmi160_set_accel_bandwidth(uint8_t v_bw_u8)** **int8_t bmi160_get_accel_bandwidth(uint8_t* v_bw_u8)**
Function description	(CONFIG) Sets/gets the accelerometer bandwidth, that is, either the Over-Sampling Rate (OSR) using the integrated accelerometer digital filter (acc_us under-sampling parameter is disabled) or the under-sampling rate/AVeraGing (AVG) (acc_us under-sampling parameter is enabled).
Parameters	v_bw_u8 : the value of the accelerometer bandwidth BMI160_ACCEL_OSR4_AVG1, BMI160_ACCEL_OSR2_AVG2, BMI160_ACCEL_NORMAL_AVG4, BMI160_ACCEL_CIC_AVG8, BMI160_ACCEL_RES_AVG16, BMI160_ACCEL_RES_AVG32, BMI160_ACCEL_RES_AVG64, BMI160_ACCEL_RES_AVG128
Return values	0 for successful execution; ≠0 for error
Notes	If digital filter configuration is enabled (acc_us=0), the options are: OSR4 mode, OSR2 mode, normal mode, CIS mode If under-sampling mode is enabled (acc_us=1), the options are: averaging 1 (AVG1 or no-averaging), 2 (AVG2), 4 (AVG4), 8 (AVG8), 16 (AVG16), 32 (AVG32), 64 (AVG64), 128 (AVG128) samples

8.6.1.29 bmi160_[set/get]_accel_range()

Function name	**int8_t bmi160_set_accel_range(uint8_t v_range_u8)** **int8_t bmi160_get_accel_range(uint8_t* v_range_u8)**
Function description	(CONFIG) Sets/gets the accelerometer g-range.
Parameters	v_range_u8 : the value of the accelerometer range BMI160_ACCEL_RANGE_2G, BMI160_ACCEL_RANGE_4G, BMI160_ACCEL_RANGE_8G, BMI160_ACCEL_RANGE_16G
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.30 bmi160_[set/get]_gyro_out_data_rate()

Function name	**int8_t bmi160_set_gyro_out_data_rate(uint8_t v_output_data_rate_u8)** **int8_t bmi160_get_gyro_out_data_rate(uint8_t* v_output_data_rate_u8)**
Function description	(CONFIG) Sets/gets the gyroscope output data rate (odr), that is, the sampling data rate.
Parameters	v_output_data_rate_u8 : the value of the gyroscope output data rate BMI160_GYRO_OUTPUT_DATA_RATE_RESERVED, BMI160_GYRO_OUTPUT_DATA_RATE_25HZ, BMI160_GYRO_OUTPUT_DATA_RATE_50HZ, BMI160_GYRO_OUTPUT_DATA_RATE_100HZ, BMI160_GYRO_OUTPUT_DATA_RATE_200HZ, BMI160_GYRO_OUTPUT_DATA_RATE_400HZ, BMI160_GYRO_OUTPUT_DATA_RATE_800HZ, BMI160_GYRO_OUTPUT_DATA_RATE_1600HZ, BMI160_GYRO_OUTPUT_DATA_RATE_3200HZ
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.31 bmi160_[set/get]_gyro_bandwidth()

Function name	**int8_t bmi160_set_gyro_bandwidth(uint8_t v_bw_u8)** **int8_t bmi160_get_gyro_bandwidth(uint8_t* v_bw_u8)**
Function description	(CONFIG) Sets/gets the gyroscope bandwidth, that is, the Over-Sampling Rate (OSR) using the integrated gyroscope digital filter.
Parameters	v_bw_u8 : the value of the gyroscope bandwidth BMI160_GYRO_OSR4_MODE, BMI160_GYRO_OSR2_MODE, BMI160_GYRO_NORMAL_MODE, BMI160_GYRO_CIC_MODE
Return values	0 for successful execution; ≠0 for error
Notes	The options for the digital filter configuration are: OSR4 mode, OSR2 mode, normal mode, CIS mode

8.6.1.32 bmi160_[set/get]_gyro_range()

Function name	**int8_t bmi160_set_gyro_range(uint8_t v_range_u8)** **int8_t bmi160_get_gyro_range(uint8_t* v_range_u8)**
Function description	(CONFIG) Sets/gets the gyroscope angular rate measurement range or resolution.
Parameters	v_range_u8 : the value of the gyroscope range BMI160_GYRO_RANGE_2000_DEG_SEC, BMI160_GYRO_RANGE_1000_DEG_SEC, BMI160_GYRO_RANGE_500_DEG_SEC, BMI160_GYRO_RANGE_250_DEG_SEC, BMI160_GYRO_RANGE_125_DEG_SEC
Return values	0 for successful execution; ≠0 for error
Notes	Angular rate measurement range results in specific resolution, as follows: ±2000º/s range → 16.4 LSB / º/s ⇔ 61.0 mº/s / LSB resolution ±1000º/s range → 32.8 LSB / º/s ⇔ 30.5 mº/s / LSB resolution ± 500º/s range → 65.6 LSB / º/s ⇔ 15.3 mº/s / LSB resolution ± 250º/s range → 131.2 LSB / º/s ⇔ 7.6 mº/s / LSB resolution ± 125º/s range → 262.4 LSB / º/s ⇔ 3.8 mº/s / LSB resolution

8.6.1.33 bmi160_[set/get]_fifo_accel_downsampling_enable()

Function name	**int8_t bmi160_set_fifo_accel_downsampling_enable(uint8_t v_fifo_down_u8)** **int8_t bmi160_get_fifo_accel_downsampling_enable(uint8_t* v_fifo_down_u8)**
Function description	(CONFIG) Sets/gets the accelerometer down-sampling rate for the data stored to the FIFO (acc_fifo_downs_rate), that is, the rate of samples that are dropped.
Parameters	v_fifo_down_u8 : the value of the accelerometer FIFO down-sampling rate
Return values	0 for successful execution; ≠0 for error
Notes	FIFO down-sampling rate for the accelerometer is given by $2^{Val(acc\_fifo\_downs\_rate)}$

8.6.1.34 bmi160_[set/get]_fifo_accel_filter_data_enable()

Function name	**int8_t bmi160_set_fifo_accel_filter_data_enable(uint8_t v_accel_fifo_filter_u8)** **int8_t bmi160_get_fifo_accel_filter_data_enable(uint8_t* v_accel_fifo_filter_u8)**
Function description	(CONFIG) Sets/gets filtered/pre-filtered mode for the accelerometer data stored to the FIFO.
Parameters	v_accel_fifo_filter_u8 : the value of the accelerometer FIFO data filter mode FILTER_DATA, UNFILTER_DATA
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.35 bmi160_[set/get]_fifo_gyro_downsampling_enable()

Function name	**int8_t bmi160_set_fifo_gyro_downsampling_enable(uint8_t v_fifo_down_u8)** **int8_t bmi160_get_fifo_gyro_downsampling_enable(uint8_t* v_fifo_down_u8)**
Function description	(CONFIG) Sets/gets the gyroscope down-sampling rate for the data stored to the FIFO (gyr_fifo_downs_rate), that is, the rate of samples that are dropped.
Parameters	v_fifo_down_u8 : the value of the gyroscope FIFO down-sampling rate
Return values	0 for successful execution; ≠0 for error
Notes	FIFO down-sampling rate for the gyroscope is given by $2^{Val(gyr\_fifo\_downs\_rate)}$

8.6.1.36 bmi160_[set/get]_fifo_gyro_filter_data_enable()

Function name	**int8_t bmi160_set_fifo_gyro_filter_data_enable(uint8_t v_gyro_fifo_filter_u8)** **int8_t bmi160_get_fifo_gyro_filter_data_enable(uint8_t* v_gyro_fifo_filter_u8)**
Function description	(CONFIG) Sets/gets filtered/pre-filtered mode for the gyroscope data stored to the FIFO.
Parameters	v_gyro_fifo_filter_u8 : the value of the gyroscope FIFO data filter mode FILTER_DATA, UNFILTER_DATA
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.37 bmi160_[set/get]_fifo_watermark_level()

Function name	**int8_t bmi160_set_fifo_watermark_level(uint8_t v_fifo_wm_u8)** **int8_t bmi160_get_fifo_watermark_level(uint8_t* v_fifo_wm_u8)**
Function description	(CONFIG) Sets/gets the FIFO watermark level (wm), that is, the level in bytes of the stored data in the FIFO that triggers the respective FIFO watermark interrupt.
Parameters	v_fifo_wm_u8 : the value of the FIFO watermark level (in 4-byte units)
Return values	0 for successful execution; ≠0 for error
Notes	FIFO watermark level is measured in 4 bytes units.

8.6.1.38 bmi160_[set/get]_fifo_stop_on_full_enable()

Function name	**int8_t bmi160_set_fifo_stop_on_full_enable(uint8_t v_fifo_stop_on_full_u8)** **int8_t bmi160_get_fifo_stop_on_full_enable(uint8_t* v_fifo_stop_on_full_u8)**
Function description	(CONFIG) Sets/gets the mode of stopping writing samples into FIFO when it is full.
Parameters	v_fifo_stop_on_full_u8 : the value of the FIFO stop-on-full behaviour mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.39 bmi160_[set/get]_fifo_time_enable()

Function name	**int8_t bmi160_set_fifo_time_enable(uint8_t v_fifo_time_u8)** **int8_t bmi160_get_fifo_time_enable(uint8_t* v_fifo_time_u8)**
Function description	(CONFIG) Sets/gets the mode of setting a FIFO sensor time frame after the last valid data frame.
Parameters	v_fifo_time_u8 : the value of the FIFO sensor time setting mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.40 bmi160_[set/get]_fifo_header_enable()

Function name	**int8_t bmi160_set_fifo_header_enable(uint8_t v_fifo_header_u8)** **int8_t bmi160_get_fifo_header_enable(uint8_t* v_fifo_header_u8)**
Function description	(CONFIG) Sets/gets the frame header FIFO data storing mode. If disabled, FIFO operated in frame header-less mode.
Parameters	v_fifo_header_u8 : the value of the frame header FIFO data storing mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.41 bmi160_[set/get]_fifo_accel_enable()

Function name	**int8_t bmi160_set_fifo_accel_enable(uint8_t v_fifo_accel_u8)** **int8_t bmi160_get_fifo_accel_enable(uint8_t* v_fifo_accel_u8)**
Function description	(CONFIG) Sets/gets the accelerometer FIFO data storing mode. If enabled, accelerometer xyz-axis data are stored to the FIFO.
Parameters	v_fifo_accel_u8 : the value of the accelerometer FIFO data storing mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.42 bmi160_[set/get]_fifo_gyro_enable()

Function name	**int8_t bmi160_set_fifo_gyro_enable(uint8_t v_fifo_gyro_u8)** **int8_t bmi160_get_fifo_gyro_enable(uint8_t* v_fifo_gyro_u8)**
Function description	(CONFIG) Sets/gets the gyroscope FIFO data storing mode. If enabled, gyroscope xyz-axis data are stored to the FIFO.
Parameters	v_fifo_gyro_u8 : the value of the gyroscope FIFO data storing mode BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.43 bmi160_[set/get]_any_motion_x_enable()

Function name	**int8_t bmi160_set_any_motion_x_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_any_motion_x_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of any-motion on x-axis.
Parameters	v_intr_enable_u8 : the value of the any-motion on x-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.44 bmi160_[set/get]_any_motion_y_enable()

Function name	**int8_t bmi160_set_any_motion_y_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_any_motion_y_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of any-motion on y-axis.
Parameters	v_intr_enable_u8 : the value of the any-motion on y-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.45 bmi160_[set/get]_any_motion_z_enable()

Function name	**int8_t bmi160_set_any_motion_z_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_any_motion_z_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of any-motion on z-axis.
Parameters	v_intr_enable_u8 : the value of the any-motion on z-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.46 bmi160_[set/get]_drdy_enable()

Function name	**int8_t bmi160_set_drdy_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_drdy_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the data ready (drdy) interrupt.
Parameters	v_intr_enable_u8 : the value of the data ready interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.47 bmi160_[set/get]_fifo_full_enable()

Function name	**int8_t bmi160_set_fifo_full_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_fifo_full_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the FIFO full interrupt.
Parameters	v_intr_enable_u8 : the value of the FIFO full interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.48 bmi160_[set/get]_fifo_watermark_enable()

Function name	**int8_t bmi160_set_fifo_watermark_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_fifo_watermark_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the FIFO watermark (wm) interrupt.
Parameters	v_intr_enable_u8 : the value of the FIFO watermark interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.49 bmi160_[set/get]_no_motion_x_enable()

Function name	**int8_t bmi160_set_no_motion_x_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_no_motion_x_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of no-motion on x-axis.
Parameters	v_intr_enable_u8 : the value of the no-motion on x-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.50 bmi160_[set/get]_no_motion_y_enable()

Function name	**int8_t bmi160_set_no_motion_y_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_no_motion_y_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of no-motion on y-axis.
Parameters	v_intr_enable_u8 : the value of the no-motion on y-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.51 bmi160_[set/get]_no_motion_z_enable()

Function name	**int8_t bmi160_set_no_motion_z_enable(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_no_motion_z_enable(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of the interrupt of no-motion on z-axis.
Parameters	v_intr_enable_u8 : the value of the no-motion on z-axis interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.52 bmi160_[set/get]_step_detector_enable()

Function name	**int8_t bmi160_set_step_detector_enable(uint8_t v_step_intr_u8)** **int8_t bmi160_get_step_detector_enable(uint8_t* v_step_intr_u8)**
Function description	(CONFIG) Sets/gets the state of the step detection interrupt.
Parameters	v_intr_enable_u8 : the value of the step detection interrupt enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.53 bmi160_[set/get]_intr_1_output_edge_level()

Function name	**int8_t bmi160_set_intr_1_output_edge_level(uint8_t v_intr_edge_ctrl_u8)** **int8_t bmi160_get_intr_1_output_edge_level(uint8_t* v_intr_edge_ctrl_u8)**
Function description	(CONFIG) Sets/gets the trigger condition, edge or level, .for interrupts mapped to pin INT1 of BMI160 sensor module.
Parameters	v_intr_edge_ctrl_u8 : the value of the trigger condition for INT1 BMI160_LEVEL, BMI160_EDGE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.54 bmi160_[set/get]_intr_1_output_level()

Function name	**int8_t bmi160_set_intr_1_output_level(uint8_t v_intr_level_u8)** **int8_t bmi160_get_intr_1_output_level(uint8_t* v_intr_level_u8)**
Function description	(CONFIG) Sets/gets the level condition - polarity, high or low, .for interrupts mapped to pin INT1 of BMI160 sensor module.
Parameters	v_intr_level_u8 : the value of the level condition - polarity for INT1 BMI160_LEVEL_LOW, BMI160_LEVEL_HIGH
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.55 bmi160_[set/get]_intr_1_output_odrn_pshpll()

Function name	**int8_t bmi160_set_intr_1_output_odrn_pshpll(uint8_t v_intr_output_type_u8)** **int8_t bmi160_get_intr_1_output_odrn_pshpll(uint8_t* v_intr_output_type_u8)**
Function description	(CONFIG) Sets/gets the type, open-drain or push-pull, .for interrupts mapped to pin INT1 of BMI160 sensor module.
Parameters	v_intr_level_u8 : the value of the interrupt type for INT1 BMI160_PUSH_PULL, BMI160_OPEN_DRAIN
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.56 bmi160_[set/get]_intr_1_output_enable()

Function name	**int8_t bmi160_set_intr_1_output_enable(uint8_t v_output_enable_u8)** **int8_t bmi160_get_intr_1_output_enable(uint8_t* v_output_enable_u8)**
Function description	(CONFIG) Sets/gets the operation mode of pin INT1 of BMI160 sensor module as output.
Parameters	v_output_enable_u8 : the value of the INT1 output mode enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.57 bmi160_[set/get]_intrs_latched()

Function name	**int8_t bmi160_set_intrs_latched(uint8_t v_latch_intr_u8)** **int8_t bmi160_get_intrs_latched(uint8_t* v_latch_intr_u8)**
Function description	(CONFIG) Sets/gets BMI160 sensor module interrupts' latch duration.
Parameters	v_latch_intr_u8 : the value of the interrupt latch duration BMI160_LATCH_DUR_NONE, BMI160_LATCH_DUR_312_5_MICRO_SEC, BMI160_LATCH_DUR_625_MICRO_SEC, BMI160_LATCH_DUR_1_25_MILLI_SEC, BMI160_LATCH_DUR_2_5_MILLI_SEC, BMI160_LATCH_DUR_5_MILLI_SEC, BMI160_LATCH_DUR_10_MILLI_SEC, BMI160_LATCH_DUR_20_MILLI_SEC, BMI160_LATCH_DUR_40_MILLI_SEC, BMI160_LATCH_DUR_80_MILLI_SEC, BMI160_LATCH_DUR_160_MILLI_SEC, BMI160_LATCH_DUR_320_MILLI_SEC, BMI160_LATCH_DUR_640_MILLI_SEC, BMI160_LATCH_DUR_1_28_SEC, BMI160_LATCH_DUR_2_56_SEC, BMI160_LATCHED
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.58 bmi160_[set/get]_any_motion_intr_1()

Function name	**int8_t bmi160_set_any_motion_intr_1(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_any_motion_intr_1(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of mapping any-motion interrupt to BMI160 sensor module pin INT1.
Parameters	v_intr_enable_u8 : the value of the any-motion interrupt mapping enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.59 bmi160_[set/get]_no_motion_intr_1()

Function name	**int8_t bmi160_set_no_motion_intr_1(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_no_motion_intr_1(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of mapping no-motion interrupt to BMI160 sensor module pin INT1.
Parameters	v_intr_enable_u8 : the value of the no-motion interrupt mapping enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.60 bmi160_[set/get]_fifo_full_intr_1()

Function name	**int8_t bmi160_set_fifo_full_intr_1(uint8_t v_intr_fifo_full_u8)** **int8_t bmi160_get_fifo_full_intr_1(uint8_t* v_intr_fifo_full_u8)**
Function description	(CONFIG) Sets/gets the state of mapping FIFO full interrupt to BMI160 sensor module pin INT1.
Parameters	v_intr_fifo_full_u8 : the value of the FIFO full interrupt mapping enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.61 bmi160_[set/get]_fifo_watermark_intr_1()

Function name	**int8_t bmi160_set_fifo_watermark_intr_1(uint8_t v_intr_fifo_wm_u8)** **int8_t bmi160_get_fifo_watermark_intr_1(uint8_t* v_intr_fifo_wm_u8)**
Function description	(CONFIG) Sets/gets the state of mapping FIFO watermark (wm) interrupt to BMI160 sensor module pin INT1.
Parameters	v_intr_fifo_wm_u8 : the value of the FIFO watermark interrupt mapping enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.62 bmi160_[set/get]_drdy_intr_1()

Function name	**int8_t bmi160_set_drdy_intr_1(uint8_t v_intr_enable_u8)** **int8_t bmi160_get_drdy_intr_1(uint8_t* v_intr_enable_u8)**
Function description	(CONFIG) Sets/gets the state of mapping data ready (drdy) interrupt to BMI160 sensor module pin INT1.
Parameters	v_intr_enable_u8 : the value of the drdy interrupt mapping enable bit BMI160_DISABLE, BMI160_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.63 bmi160_[set/get]_any_motion_duration()

Function name	**int8_t bmi160_set_any_motion_duration(uint8_t v_any_motion_durn_u8)** **int8_t bmi160_get_any_motion_duration(uint8_t* v_any_motion_durn_u8)**
Function description	(CONFIG) Sets/gets the any-motion interrupt trigger delay duration.
Parameters	v_any_motion_durn_u8 : the value of the any-motion interrupt duration
Return values	0 for successful execution; ≠0 for error
Notes	Any-motion duration can be given by (v_any_motion_durn_u8 + 1)

8.6.1.64 bmi160_[set/get]_slow_no_motion_duration()

Function name	**int8_t bmi160_set_slow_no_motion_duration(uint8_t v_slow_no_motion_u8)** **int8_t bmi160_get_slow_no_motion_duration(uint8_t* v_slow_no_motion_u8)**
Function description	(CONFIG) Sets/gets the slow/no-motion interrupt trigger delay duration.
Parameters	v_slow_no_motion_u8 : the value of the slow/no-motion interrupt duration
Return values	0 for successful execution; ≠0 for error
Notes	Slow/no-motion duration can be given by: if v_slow_no_motion_u8 < 16 → (v_slow_no_motion_u8%16 + 1)×1.28s (1.28s - 20.48s duration range) if 16 ≤ v_slow_no_motion_u8 < 32 → (v_slow_no_motion_u8%16 + 5)×5.12s (25.6s - 102.4s duration range) if 32 ≤ v_slow_no_motion_u8 → (v_slow_no_motion_u8%32 + 11)×10.24s (112.64s - 430.08s duration range)

8.6.1.65 bmi160_[set/get]_any_motion_threshold()

Function name	**int8_t bmi160_set_any_motion_threshold(uint8_t v_any_motion_thres_u8)** **int8_t bmi160_get_any_motion_threshold(uint8_t* v_any_motion_thres_u8)**
Function description	(CONFIG) Sets/gets the any-motion interrupt threshold. It is accelerometer range-dependent.
Parameters	v_any_motion_thres_u8 : the value of the any-motion interrupt threshold
Return values	0 for successful execution; ≠0 for error
Notes	Any-motion interrupt threshold can be given by: if v_any_motion_thres_u8 = 0 → threshold is 1.95mg (2g range), 3.91mg (4g range), 7.81mg (8g range), 15.63mg (16g range) if v_any_motion_thres_u8 ≠ 0 → threshold is v_any_motion_thres_u8×3.91mg (2g range), v_any_motion_thres_u8×7.81mg (4g range), v_any_motion_thres_u8×15.63mg (8g range), v_any_motion_thres_u8×31.25mg (16g range)

8.6.1.66 bmi160_[set/get]_slow_no_motion_threshold()

Function name	**int8_t bmi160_set_slow_no_motion_threshold(uint8_t v_slow_no_motion_thres_u8)** **int8_t bmi160_get_slow_no_motion_threshold(uint8_t* v_slow_no_motion_thres_u8)**
Function description	(CONFIG) Sets/gets the slow/no-motion interrupt threshold. It is accelerometer range-dependent.
Parameters	v_slow_no_motion_thres_u8 : the value of the slow/no-motion interrupt threshold
Return values	0 for successful execution; ≠0 for error
Notes	Slow/no-motion interrupt threshold can be given by: if v_slow_no_motion_thres_u8 = 0 → threshold is 1.95mg (2g range), 3.91mg (4g range), 7.81mg (8g range), 15.63mg (16g range) if v_slow_no_motion_thres_u8 ≠ 0 → threshold is v_slow_no_motion_thres_u8×3.91mg (2g range), v_slow_no_motion_thres_u8×7.81mg (4g range), v_slow_no_motion_thres_u8×15.63mg (8g range), v_slow_no_motion_thres_u8×31.25mg (16g range)

8.6.1.67 bmi160_[set/get]_slow_no_motion_select()

Function name	**int8_t bmi160_set_slow_no_motion_select(uint8_t v_intr_slow_no_motion_select_u8)** **int8_t bmi160_get_slow_no_motion_select(uint8_t* v_intr_slow_no_motion_select_u8)**
Function description	(CONFIG) Sets/gets the slow/no-motion interrupt type, that is, slow or no motion as trigger condition for the slow/no motion interrupt.
Parameters	v_intr_slow_no_motion_select_u8 : the value of the slow/no-motion interrupt type SLOW_MOTION, NO_MOTION
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.68 bmi160_[set/get]_significant_motion_select()

Function name	**int8_t bmi160_set_significant_motion_select(uint8_t v_intr_significant_motion_select_u8)** **int8_t bmi160_get_significant_motion_select(uint8_t* v_intr_significant_motion_select_u8)**
Function description	(CONFIG) Sets/gets the significant/any-motion interrupt type, that is, significant or anymotion as trigger condition for the significant/anymotion interrupt.
Parameters	v_intr_significant_motion_select_u8 : the value of the significant/any-motion interrupt type ANY_MOTION, SIGNIFICANT_MOTION
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.69 bmi160_[set/get]_significant_motion_skip()

Function name	**int8_t bmi160_set_significant_motion_skip(uint8_t v_int_sig_mot_skip_u8)** **int8_t bmi160_get_significant_motion_skip(uint8_t* v_int_sig_mot_skip_u8)**
Function description	(CONFIG) Sets/gets the significant-motion interrupt skip time (t_skip), that is, the time that needs to pass, so that a new detection of significant motion can begin (of t_proof duration) and finally trigger the significant/anymotion interrupt.
Parameters	v_int_sig_mot_skip_u8 : the value of the significant-motion interrupt interrupt skip time BMI160_SIGNIFICANT_MOTION_SKIP_1_5_SEC, BMI160_SIGNIFICANT_MOTION_SKIP_3_SEC, BMI160_SIGNIFICANT_MOTION_SKIP_6_SEC, BMI160_SIGNIFICANT_MOTION_SKIP_12_SEC
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.70 bmi160_[set/get]_significant_motion_proof()

Function name	**int8_t bmi160_set_significant_motion_proof(uint8_t v_significant_motion_proof_u8)** **int8_t bmi160_get_significant_motion_proof(uint8_t* v_significant_motion_proof_u8)**
Function description	(CONFIG) Sets/gets the significant-motion interrupt proof time (t_proof), that is, the time during which a new detection of significant motion has to happen (after t_skip time of the first detection), so that a significant/anymotion interrupt can be triggered.
Parameters	v_significant_motion_proof_u8 : the value of the significant-motion interrupt skip time BMI160_SIGNIFICANT_MOTION_PROOF_0_25_SEC, BMI160_SIGNIFICANT_MOTION_PROOF_0_5_SEC, BMI160_SIGNIFICANT_MOTION_PROOF_1_SEC, BMI160_SIGNIFICANT_MOTION_PROOF_2_SEC
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.71 bmi160_[set/get]_gyro_sleep_trigger()

Function name	**int8_t bmi160_set_gyro_sleep_trigger(uint8_t v_gyro_sleep_trigger_u8)** **int8_t bmi160_get_gyro_sleep_trigger(uint8_t* v_gyro_sleep_trigger_u8)**
Function description	(CONFIG) Sets/gets the trigger condition for the gyroscope transition to sleep power mode.
Parameters	v_gyro_sleep_trigger_u8 : the value of the gyroscope sleep transition trigger condition BMI160_GYRO_SLEEP_TRIGGER_DISABLE, BMI160_GYRO_SLEEP_TRIGGER_INT2, BMI160_GYRO_SLEEP_TRIGGER_NOTINT1, BMI160_GYRO_SLEEP_TRIGGER_NOTINT1_OR_INT2, BMI160_GYRO_SLEEP_TRIGGER_NOMOTION, BMI160_GYRO_SLEEP_TRIGGER_ANYMOTION_OR_INT2, BMI160_GYRO_SLEEP_TRIGGER_ANYMOTION_OR_NOTINT1, BMI160_GYRO_SLEEP_TRIGGER_ANYMOTION_OR_NOTINT1_OR_INT2
Return values	0 for successful execution; ≠0 for error
Notes	If a gyroscope wakeup condition (ref. in bmi160_set_gyro_wakeup_trigger) holds at the same time, then sleep transition is cancelled.

8.6.1.72 bmi160_[set/get]_gyro_wakeup_trigger()

Function name	**int8_t bmi160_set_gyro_wakeup_trigger(uint8_t v_gyro_wakeup_trigger_u8)** **int8_t bmi160_get_gyro_wakeup_trigger(uint8_t* v_gyro_wakeup_trigger_u8)**
Function description	(CONFIG) Sets/gets the trigger condition for the gyroscope transition to normal power mode.
Parameters	v_gyro_wakeup_trigger_u8 : the value of the gyroscope wakeup transition trigger condition BMI160_GYRO_WAKEUP_TRIGGER_DISABLE, BMI160_GYRO_WAKEUP_TRIGGER_INT1, BMI160_GYRO_WAKEUP_TRIGGER_ANYMOTION, BMI160_GYRO_WAKEUP_TRIGGER_ANYMOTION_AND_INT1
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.1.73 bmi160_[set/get]_gyro_sleep_trigger_state()

Function name	**int8_t bmi160_set_gyro_sleep_trigger_state(uint8_t v_gyro_sleep_state_u8)** **int8_t bmi160_get_gyro_sleep_trigger_state(uint8_t* v_gyro_sleep_state_u8)**
Function description	(CONFIG) Sets/gets the sleep state of the gyroscope when a sleep transition is performed, i.e. GYR_SUSPEND or GYR_FASTSTARTUP.
Parameters	v_gyro_sleep_state_u8 : the value of the gyroscope sleep state on sleep transition BMI160_GYRO_SLEEP_TRIGGER_STATE_FAST_STARTUP, BMI160_GYRO_SLEEP_TRIGGER_STATE_SUSPEND
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2 BMM150 sensor module

The following sections list the functions of the BMM150 (or BMM050) driver library used by the application level functions presented in section 8.6. General guidelines for using this sensor driver are as follows:

- Initialise the module using `bmm150_initialize_sensor()`.
- Set the power state of the module by setting the power state of the magnetometer sensor using `bmi150_set_sensor_state()`.
- Configure the module to operate in a predefined mode using `bmi160_config_preset_mode()`.
- Read magnetometer sensor data using `bmm150_read_mag_xyz()`.
- Make appropriate decisions by getting the status of the sensor module (`state_feature = drdy_mag_stat, low_thres_z _stat, etc`.), using the `bmm150_get_[state_feature]()` set of functions.
- Configure differently or extend the sensor module's predefined operating mode in terms of specific setting features (`setting_feature = mag_out_data_rate,mag_repetition_xy, mag_repetition_z, etc`.) using the `bmm150_set_[setting_feature]()` set of functions.
- Get the current configuration of the sensor module in terms of a specific setting feature using the `bmm150_get_[setting_feature]()` set of functions.

8.6.2.1 bmm150_initialize_sensor()

Function name	**int8_t bmm150_initialize_sensor()**
Function description	(INIT-CONFIG) Initializes the BMM150 sensor module, creating the appropriate instance handle for the module which includes references to the spi bus access functions used for reading/writing values to the registers, as well as setting the state of magnetometer sensor to MAG_SLEEP.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.2 bmm150_set_sensor_state()

Function name	**int8_t bmm150_set_sensor_state(uint8_t state)**
Function description	(INIT-CONFIG) Sets the state of the magnetometer sensor integrated in BMM150 sensor module.
Parameters	state : the state to set for the magnetometer sensor MAG_NORMAL, MAG_FORCED, MAG_SUSPEND, MAG_SLEEP
Return values	0 for successful execution; ≠0 for error
Notes	On successful execution it is introduced a BMM050_DELAY_SUSPEND_SLEEP delay.

8.6.2.3 bmm150_config_preset_mode()

Function name	**int8_t bmm150_config_preset_mode(uint8_t v_preset_mode_u8)**
Function description	(INIT-CONFIG) Sets a predefined operation mode for the BMI160 sensor module.
Parameters	v_preset_mode_u8 : the predefined operation mode to set for the magnetometer sensor MAG_PRESETMODE_LOWPOWER, MAG_PRESETMODE_REGULAR, MAG_PRESETMODE_HIGHACCURACY, MAG_PRESETMODE_ENHANCED
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.4 bmm150_read_mag_xyz()

Function name	**int8_t bmm150_read_mag_xyz(struct bmm050_mag_data_s16_t *mag_xyz)**
Function description	(SAMPLE) Reads magnetometer xyz-axis data.
Parameters	mag_xyz : the magnetometer xyz-axis data
Return values	0 for successful execution; ≠0 for error
Notes	Magnetometer axis sensor data values are measured in µT.

8.6.2.5 bmm150_get_drdy_mag_stat()

Function name	**int8_t bmm150_get_drdy_mag_stat(uint8_t* v_data_rdy_u8)**
Function description	(STATE) Reads the status of data ready (drdy) status bit, indicating whether a new magnetometer sample value has been set to the corresponding data register.
Parameters	v_data_rdy_u8 : the value of the magnetometer drdy interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.6 bmm150_get_low_thres_z_stat()

Function name	**int8_t bmm150_get_low_thres_z_stat(uint8_t* v_low_thres_u8)**
Function description	(STATE) Reads the status of low threshold magnetometer data on z-axis status bit, indicating whether a magnetometer sample value has been less than the defined low threshold (ref. bmm150_set_mag_low_thres()).
Parameters	v_data_rdy_u8 : the value of the magnetometer low threshold on z-axis interrupt status bit
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.7 bmm150_set_mag_out_data_rate()

Function name	**int8_t bmm150_set_mag_out_data_rate(uint8_t v_output_data_rate_u8)**
Function description	(CONFIG) Sets the magnetometer output data rate (odr), that is, the sampling data rate.
Parameters	v_output_data_rate_u8 : the value of the magnetometer output data rate BMM050_DATA_RATE_10HZ, BMM050_DATA_RATE_02HZ, BMM050_DATA_RATE_06HZ, BMM050_DATA_RATE_08HZ, BMM050_DATA_RATE_15HZ, BMM050_DATA_RATE_20HZ, BMM050_DATA_RATE_25HZ, BMM050_DATA_RATE_30HZ
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.8 bmm150_set_mag_repetition_xy()

Function name	**int8_t bmm150_set_mag_repetition_xy(uint8_t v_rep_xy_u8)**
Function description	(CONFIG) Sets the number of repetitions for magnetometer xy-axis.
Parameters	v_rep_xy_u8 : the value of the magnetometer xy-axis repetitions
Return values	0 for successful execution; ≠0 for error
Notes	The performed number of repetitions are given by: num. of xy-repetitions = 1+2×v_rep_xy_u8

8.6.2.9 bmm150_set_mag_repetition_z()

Function name	**int8_t bmm150_set_mag_repetition_z(uint8_t v_rep_z_u8)**
Function description	(CONFIG) Sets the number of repetitions for magnetometer z-axis.
Parameters	v_rep_z_u8 : the value of the magnetometer z-axis repetitions
Return values	0 for successful execution; ≠0 for error
Notes	The performed number of repetitions are given by: num. of z-repetitions = 1+v_rep_z_u8

8.6.2.10 bmm150_set_mag_low_thres()

Function name	**int8_t bmm150_set_mag_low_thres(uint8_t v_thres_u8)**
Function description	(CONFIG) Sets the low threshold level for magnetometer sensor data, defining the condition for triggering low threshold interrupt events..
Parameters	v_thres_u8 : the value of the magnetometer data low threshold level
Return values	0 for successful execution; ≠0 for error
Notes	1 bit corresponds to roughly 6µT.

8.6.2.11 bmm150_set_low_thres_z_enable()

Function name	**int8_t bmm150_set_low_thres_z_enable(uint8_t v_intr_enable_u8)**
Function description	(CONFIG) Sets the state of the interrupt of low threshold magnetometer sensor data on z-axis.
Parameters	v_intr_enable_u8 : the value of the low threshold on z-axis interrupt enable bit BMM050_DISABLE, BMM050_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.12 bmm150_set_drdy_enable()

Function name	**int8_t bmm150_set_drdy_enable(uint8_t v_intr_enable_u8)**
Function description	(CONFIG) Sets the state of mapping the data ready (drdy) interrupt status to the DRDY pin of the BMM150 sensor module.
Parameters	v_intr_enable_u8 : the value of the data ready interrupt mapping to DRDY pin enable bit BMM050_DISABLE, BMM050_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.13 bmm150_set_int_enable()

Function name	**int8_t bmm150_set_int_enable(uint8_t v_intr_enable_u8)**
Function description	(CONFIG) Sets the state of mapping interrupts' status to the INT pin of the BMM150 sensor module.
Parameters	v_intr_enable_u8 : the value of the interrupts mapping to INT pin enable bit BMM050_DISABLE, BMM050_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.14 bmm150_set_drdy_polarity()

Function name	**int8_t bmm150_set_drdy_polarity(uint8_t v_intr_polarity_u8)**
Function description	(CONFIG) Sets the DRDY interrupt pin polarity.
Parameters	v_intr_polarity_u8 : the value of the DRDY pin polarity BMM050_POLARITY_LOW, BMM050_POLARITY_HIGH
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.15 bmm150_set_int_latched()

Function name	**int8_t bmm150_set_int_latched(uint8_t v_intr_latch_u8)**
Function description	(CONFIG) Sets the state of the INT interrupt pin as latched.
Parameters	v_intr_latch_u8 : the value of the interrupt latching for the INT pin BMM050_DISABLE, BMM050_ENABLE
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.2.16 bmm150_set_int_polarity()

Function name	**int8_t bmm150_set_int_polarity(uint8_t v_intr_polarity_u8)**
Function description	(CONFIG) Sets the INT interrupt pin polarity.
Parameters	v_intr_polarity_u8 : the value of the INT pin polarity BMM050_POLARITY_LOW, BMM050_POLARITY_HIGH
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3 BME280 sensor module

The following section lists the various functions of the BME280 driver library used by the application level functions presented in section 8.6. General guidelines for using this sensor driver are as follows:

- Initialise the module using `bme280_initialize_sensor()`.
- Set the power state of the module by setting the power state of the environmental sensors using `bme280_set_sensor_state()`.
- Read environmental sensor data using `bme280_read_pressure_temperature_humidity()`.
- Configure differently or extend the sensor module's predefined operating mode in terms of specific setting features (`setting_feature=oversamp_pressure, oversamp_temperature, oversamp_humidity, etc`.) using the `bme280_set_[setting_feature]()` set of functions.

8.6.3.1 bme280_initialize_sensor()

Function name	**int8_t bme280_initialize_sensor()**
Function description	(INIT-CONFIG) Initializes the BMM280 sensor module, creating the appropriate instance handle for the module which includes references to the spi bus access functions used for reading/writing values to the registers, as well as setting the state of environmental sensors to ENV_FORCED.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3.2 bme280_set_sensor_state()

Function name	**int8_t bme280_set_sensor_state(uint8_t state)**
Function description	(INIT-CONFIG) Sets the state of the environmental sensors integrated in BME280 sensor module.
Parameters	state : the state to set for the environmental sensors ENV_SLEEP, ENV_FORCED, ENV_NORMAL
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3.3 bme280_read_pressure_temperature_humidity()

Function name	**int8_t bme280_read_pressure_temperature_humidity(uint32_t *v_pressure_u32, int32_t *v_temperature_s32, uint32_t *v_humidity_u32)**
Function description	(SAMPLE) Reads pressure, temperature and humidity sensor data.
Parameters	v_pressure_u32 : the pressure sensor data v_temperature_s32 : the temperature sensor data v_humidity_u32 : the humidity sensor data
Return values	0 for successful execution; ≠0 for error
Notes	Pressure, temperature and humidity sensor data values are measured in 1 Pa, 0.01 ºC and 1/1024 %Rh units, respectively.

8.6.3.4 bme280_set_soft_rst()

Function name	**int8_t bme280_set_soft_rst()**
Function description	(CMD) Performs a soft reset to the BME280 sensor module.
Parameters	None
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3.5 bme280_set_oversamp_pressure()

Function name	**int8_t bme280_set_oversamp_pressure(uint8_t v_value_u8)**
Function description	(CONFIG) Sets the oversampling mode for pressure sensor data. Setting the oversampling mode reduces noise in sensor data.
Parameters	v_value_u8 : the value of the pressure oversampling mode 0(Skipped), BME280_OVERSAMP_1X, BME280_OVERSAMP_2X, BME280_OVERSAMP_4X, BME280_OVERSAMP_8X, BME280_OVERSAMP_16X
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3.6 bme280_set_oversamp_temperature

Function name	**int8_t bme280_set_oversamp_temperature(uint8_t v_value_u8)**
Function description	(CONFIG) Sets the oversampling mode for temperature sensor data. Setting the oversampling mode reduces noise in sensor data.
Parameters	v_value_u8 : the value of the temperature oversampling mode 0(Skipped), BME280_OVERSAMP_1X, BME280_OVERSAMP_2X, BME280_OVERSAMP_4X, BME280_OVERSAMP_8X, BME280_OVERSAMP_16X
Return values	0 for successful execution; ≠0 for error
Notes	

8.6.3.7 bme280_set_oversamp_humidity

Function name	**int8_t bme280_set_oversamp_humidity(uint8_t v_value_u8)**
Function description	(CONFIG) Sets the oversampling mode for humidity sensor data. Setting the oversampling mode reduces noise in sensor data.
Parameters	v_value_u8 : the value of the humidity oversampling mode 0(Skipped), BME280_OVERSAMP_1X, BME280_OVERSAMP_2X, BME280_OVERSAMP_4X, BME280_OVERSAMP_8X, BME280_OVERSAMP_16X
Return values	0 for successful execution; ≠0 for error
Notes	

8.7 BLE interface

8.7.1 user_add_sensor_report_acc_gyro_mag()

This function sends the sensor data in the form of a notification (report) to the central device. A sensor data notification is called a report. This function forms a report for the accelerometer, gyroscope or magnetometer. A `DWS_VAL_NTF_REQ` message is formed and sent to the DWS profile task. The `dws_val_ntf_req_handler` handles this message at the DWS profile side.

8.7.2 user_add_sensor_report_pht()

This function is the same as `user_add_sensor_report_acc_gyro_mag()` except that it handles the environmental BME280 sensor values.

8.7.3 user_add_sensor_report_sensor_fusion()

This function is the same as `user_add_sensor_report_acc_gyro_mag()` except that it handles the sensor fusion data. This function is used only in the SFL project.

8.7.4 user_send_command_reply()

Some of the central device commands require a reply that may include device status and configuration data. Using this command the user may send a reply or an unsolicited notification to the central application. The message of type `DWS_VAL_NTF_REQ` is handled by the DWS profile task.

8.8 Calibration

8.8.1 user_mag_autocal_setup()

This function initialises the magnetometer calibration instance structure according to the current calibration mode and configuration.

8.8.2 user_mag_set_coeffs()

This function initialises the calibration coefficients within the magnetometer calibration instance structure, applying any necessary type conversions.

8.8.3 user_mag_get_coeffs()

This function reads the calibration coefficients from the magnetometer calibration instance structure, applying any necessary type conversions.

8.9 Sensor fusion

8.9.1 user_sensor_fusion_init()

This function initialises the sensor fusion parameter structure according to the current sensor configuration.

8.9.2 user_sensor_fusion_process()

This function performs sensor fusion processing, applying any necessary type conversions and transforms the sensor data into the coordinate system used.

9 Sensor Calibration Library

9.1 Overview

The Sensor Calibration Library provides functionality for compensating the imperfections and distortions that are often associated with MEMS sensors. While primarily provided to compensate for magnetometer hard iron offset and soft iron spherical distortions, it can also be used to correct simpler offset and sensitivity imperfections found in gyroscopes and accelerometers.

9.1.1 Modes of operation

The Sensor Calibration Library supports three different modes of operation to meet differing user requirements. The supported calibration modes are as follows:

- **Static Calibration mode:** When the sensor distortions are measureable, stable and consistent between devices, static calibration is the preferred mode of operation, as it generally gives the best performance in terms of distortion correction. In this mode, static calibration parameters are used to initialise the calibration routine, which are then applied unmodified. These static calibration parameters are calculated offline by the device manufacturer, by analysing recordings of raw sensor data made under controlled conditions and then stored in either the firmware or the device's non-volatile memory.
- **Continuous Automatic Calibration mode**: When the sensor distortions are unstable and/or inconsistent between devices, continuous automatic calibration is the preferred mode of operation, as it allows the calibration parameters to be determined automatically at runtime without requiring them to be built into the firmware or programmed into non-volatile memory. In this mode the auto-calibration function continually monitors the sensor data for distortions and adapts the calibration parameters to compensate for them.
- **One-shot Automatic Calibration mode**: When the sensor distortions are relatively stable in the short term, one-shot auto-calibration mode may be preferable. In this mode the auto-calibration function is run upon device initialisation, but when it is complete (once suitable calibration parameters have been determined) updating of the calibration parameters is disabled.

9.1.2 Automatic calibration routines

The Sensor Calibration Library provides two different automatic calibration routines that can be used in both continuous and one-shot modes:

- **Basic Auto-calibration**: The basic auto-calibration routine calculates the maximum and minimum raw sensor values and from these it calculates basic offset and scaling distortions and corrects for them. This algorithm presumes that the data received from the sensor is a clean representation of the external stimulus (gravitational/geomagnetic field) and is free from external distortions and excessive noise. It also presumes that the sensor distortions are static. When this is not the case the algorithm will give distorted results, so it is recommended to only use it for short periods (in one-shot mode) or in controlled conditions (in a calibration rig). Once distorted, the algorithm will not recover unless it is re-initialised.
- **SmartFusion Auto-calibration**: To address the shortcomings of the basic auto-calibration routine a more sophisticated algorithm is also provided. In addition to calculate and correct basic offset and scaling distortions, this algorithm can also compensate for more complicated spherical distortions (e.g. magnetometer soft iron distortions). It is also able to cope with gradual changes in sensor distortions and external environmental conditions over time (some adaptation time is required). The algorithm is not however designed to cope with environments with large variability (e.g. moving through non-uniform magnetic fields in the case of magnetometer calibration or non-rotational movement in the case of accelerometer calibration) and continuous mode should not be used in this case. If operating in these sorts of environments it is recommended to initially perform automatic calibration in one shot mode in a controlled environment and then continue applying the resultant calibration coefficients in static mode when moving into a more uncontrolled environment.

9.2 API

The various calibration routines provided by the Sensor Calibration Library are designed to work together and complement each other. Where they share common functionality, the routines use generic controls, parameters and code. The more advanced routines re-use functionality of the basic routines. For example, the basic auto-calibration routine re-uses the static calibration routine to apply its calibration parameters to the sensor data.

More specifically, the parameter structures for all routines are designed to overlap, so that they can share the same location in memory, thus sharing common parameters. This has been done to aid switching between calibration modes without unnecessary copying of parameter data between different routines and to minimise the memory footprint.

Wrapper code (`sensor_calibration.h|c`) has been provided, which implements the overlapping of the various calibration routines and provides a common interface through which they can be called.

9.2.1 Memory allocation

An instance of the appropriate calibration parameter structure (`static_calibration_params`, `basic_autocal_params` or `smartfusion_autocal_params`) or combined wrapper instance structure (`cal_instance`) must be instantiated, either statically or on the heap, and must be maintained during the lifecycle of sensor calibration processing.

9.2.2 Initialisation

Before processing can be performed on an instance of a calibration routine, a subset of parameters within the appropriate parameter structure must be initialised. These parameters are as follows:

- Generic – Common to all
  - `in_data`: Pointer from which to read raw sensor input data.
  - `out_data`: Pointer to which to write processed sensor output data.
- Static calibration – In addition to the generic parameters:
  - `offset`: Offset vector coefficients to be subtracted from sensor data in same representation as raw sensor data.
  - `matrix`: 3 x 3 matrix of signed fixed point coefficients to apply to sensor data.
  - `q_format`: Q format of matrix coefficients.
  - `flags`: Control flags

    `apply`: Flag to control whether or not calibration coefficients are applied.

    `matrix_apply`: Flag to control whether or not matrix coefficients are applied in addition to offset.

    `offset_post_apply`: Flag to control whether vector coefficients are applied before or after the matrix coefficients are applied.
- Basic Auto-calibration – In addition to the static calibration parameters:
  - `ref_mag`: Reference magnitude indicating expected geomagnetic field strength in same representation as raw sensor data.
  - `mag_range`: Q14 unsigned fixed point scaler indicating range +/- reference magnitude of valid sensor data.
  - `flags`: Control flags

    `update`: Flag to control whether or not calibration coefficients are updated.

    `matrix_update`: Flag to control whether or not matrix coefficients are updated in addition to offset.

    `init_from_static`: Flag to control whether or not calibration coefficients are initialised externally by user or should be reset by initialisation routine.

- SmartFusion Auto-calibration – In addition to the static calibration parameters:
  - `mu`: Fixed point convergence speed.
- Sensor Calibration Wrapper – If the wrapper instance structure is used then the mode field should be initialised to indicate which calibration routine should be used. The valid modes specified by the `cal_mode` enumeration are:
  - `CAL_NONE`: No calibration routine is applied.
  - `CAL_STATIC`: The static calibration routine is applied.
  - `CAL_BASIC_AUTOCAL`: The basic auto-calibration routine is applied.
  - `CAL_SMARTFUSION_AUTOCAL`: The SmartFusion auto-calibration routine is applied.

Once the calibration parameters have been initialised, it is necessary to call the appropriate routine's initialisation function (if it has one) or the `cal_init()` function, when the wrapper is being used.

Note: It is presumed that un-initialised parameters will be reset to zero. It is advised to use the function `memset()` to reset the parameter structure before initialisation.

When using the wrapper, the `controls` field is a union of a 16-bit `word` and the overlapped calibration routine `flags` bit field structure, allowing the flags to be set individually or all at once.

For reasons of more optimal code generation for the ARM M0, the pointers to the input/output data vectors reference 32-bit signed integer types. However, the magnitude of the vectors that these values represent should not exceed a signed 16-bit range (-32768 ≤ |v| ≤ 32767).

Similarly the fixed point `matrix` coefficients also use 32-bit types, but should not exceed a signed 16-bit range. In cases where the matrix coefficients represent values greater than one, they should be scaled to a 16-bit range and the `q_format` parameter adjusted accordingly. For example, to represent coefficients in the range of +/- 2.0, `q_format` should be set to 14 and the coefficients scaled by 2^{14}. Values of +2.0 (32768 once scaled) should saturate at 32767 to prevent overflow.

Depending on the algorithm used to calculate the calibration coefficients, the `offset` vector may need to be applied before or after applying the `matrix`. The static calibration routine supports both methods, but this should be indicated by setting the `offset_post_apply` appropriately. When set, the offset vector will be applied after applying the matrix.

When applying static calibration it is necessary to initialise the calibration coefficients (`offset` and `matrix`) to be applied.

When applying auto calibration, in some cases it may be preferred to backup and restore the calculated calibration coefficients between instantiations, rather than starting from scratch each time. In this case it is necessary to prevent the calibration routine's initialisation function from resetting the coefficients by setting the `init_from_static` flag.

In order to work correctly, the auto calibration routines need to know the expected magnitude of the vector produced by the sensor, once distortions have been removed. This reference magnitude is provided by initialising the `ref_mag` parameter. The value of this parameter should ideally be determined by performing an external measurement of the gravitational/geomagnetic field strength and expressing this in the same format and sensitivity as generated by the sensor. Failing this, the value can be set according to estimated or published values.

As there is inevitably some margin of error in setting of the reference magnitude, the `mag_range` parameter has been provided to specify the tolerance for the reference magnitude. This should be set in order to encompass the full range of expected acceptable magnitude values. For example, when the expected magnitude is 1000 and the desired tolerance +/-10 % (a range from 900 to 1100), the `mag_range` should be set to 0.1 (3277 in Q15 fixed point). In case the actual magnitude lies outside this range, the auto calibration routine will never complete. Conversely when the range is set too wide the calibration routine will detect completion too early and imperfect calibration will result. In case of noisy sensor data, the magnitude range can also be used to filter out outliers to some extent.

9.2.3 Processing

Sensor calibration processing is performed either by calling the appropriate process function on an instantiation of the related parameter structure or by calling function `cal_process()` when using the wrapper.

Prior to calling the process function, it is necessary to indicate to the algorithm whether or not the sensor data is valid and has been updated by setting the `in_data_valid` flag. This is done to prevent invalid data from getting into the calibration routine and corrupting its operation. Examples of invalid data include null data while the sensor is starting up or saturated sensor data. Detection of these conditions are sensor specific, so this must be done prior to calling the calibration routine.

In the case of SmartFusion auto-calibration it is possible to speed up the rate of convergence by calling the process function multiple times between samples. In this case the `in_data_valid` flag should only be set when the sensor data is updated.

In some cases it may be desirable to disable the application of the calibration coefficients at runtime or selectively only apply offset correction. The `apply` flag enables/disables the application of the calibration coefficients altogether. The `matrix_apply` flag enables/disables the application of just the matrix coefficients. It is not possible to apply the matrix in isolation.

In some cases (for instance upon completion of one-shot mode) it may be desirable to disable the updating of the calibration coefficients by the auto calibration routine at runtime or selectively only update the offset coefficients. The `update` flag enables/disables the updating of the calibration coefficients altogether. The `matrix_update` flag enables/disables the update of just the matrix coefficients. It is not possible to update the matrix in isolation.

Once the sensor calibration routine's processing cycle is complete, the calibrated sensor data can be read from the location referenced by the `out_data` pointer. The `out_data_valid` flag can also be read to determine whether the calibration routine detected the sensor data to be within its valid range of acceptable magnitudes (as specified by `ref_mag` and `mag_range`).

In the case of basic auto-calibration, the `settled` flag can be used to determine the status of the calibration routine and whether the calibration coefficients have settled and valid calibration is being applied.

10 Sensor Fusion Library

10.1 Overview

The Sensor Fusion Library provides the SmartFusion Attitude and Heading Reference System (AHRS) for fusing sensor data from MEMS gyroscopes, accelerometers and magnetometers in order to determine and track the absolute orientation of the device in which they are mounted.

The algorithm presumes that the sensor data supplied to it has been calibrated and all distortions have been compensated for. When this not the case the performance will be compromised and drift artefacts may be observed.

10.1.1 Modes of operation

The SmartFusion AHRS algorithm support different modes of operation depending on what types of sensor information are available. The supported modes are as follows:

- **Gyroscope, Accelerometer and Magnetometer (GAM) mode**: With information from all sensors, the algorithm is able to track the absolute orientation of the device and compensate for any drift in the gyroscope data and noise in the accelerometer and/or magnetometer data.
- **Gyroscope and Accelerometer (GA) mode**: With information from only the gyroscope and the accelerometer, the algorithm is able to track the orientation of the device and compensate for any drift in the pitch and roll components of the gyroscope data (but not the heading!) and noise in the accelerometer data. The reference heading is taken to be whatever the heading is at initialisation, but this will drift over time.
- **Gyroscope Only (G) mode**: With information from only the gyroscope, the algorithm is able to track the orientation of the device but will be unable to compensate for any drift in the gyroscope data. The reference orientation is taken to be whatever the orientation is at initialisation, but this will drift over time.
- **Accelerometer and Magnetometer (AM) mode**: With information from only the accelerometer and the magnetometer, the algorithm is able to track the absolute orientation of the device, but is less able to compensate for noise in the accelerometer/magnetometer data.

Note: Gyroscope and Magnetometer (GM) mode is not supported.

10.2 API

10.2.1 Memory allocation

An instance of the `SmartFusionAHRS_param` structure must be instantiated, either statically or on the heap, and must be maintained during the lifecycle of sensor fusion processing.

10.2.2 Initialisation

Before processing can be performed on an instance of the SmartFusion AHRS algorithm, a subset of the parameters within the `SmartFusionAHRS_param` structure must be initialised. These parameters are as follows:

- `controls`: Algorithm control flags.
- `g_vec_ptr`: Pointer from which to read the gyroscope input data.
- `m_vec_ptr`: Pointer from which to read the magnetometer input data.
- `a_vec_ptr`: Pointer from which to read the accelerometer input data.
- `g_scale`: Gyro scaling factor.
- `beta_a`: Scaling factor controlling the relative weight of accelerometer data.
- `beta_m`: Scaling factor controlling the relative weight of magnetometer data.
- `q`: Output quaternion representing the absolute real world orientation of the device.

The `controls` field is a union of an 8-bit `word` and the `SmartFusionAHRS_flags` bit field structure to allow the algorithm control flags to be set, individually or all at once. These flags should be initialised to indicate which sensors are available by setting the appropriate `g_vec_valid`, `a_vec_valid` and `m_vec_valid` flags to '1' or '0' to indicate the mode of operation.

The pointers to the gyroscope, accelerometer and magnetometer data vectors reference 32-bit signed integer types. However, the magnitude of the vectors that these values represent should not exceed a signed 16-bit range (-32768 ≤ |v| ≤ 32767). In case data for a particular sensor is unavailable (i.e. when using the algorithm in GA, G and AM modes), the associated pointer should be set to null (0x00000000).

`g_scale`, `beta_a` and `beta_m` are all Q15 unsigned fixed point parameters representing positive scaling factors in the range of 0 to 1.0 (32768).

To aid in the setting of the `g_scale` parameter, the `SmartFusionAHRS_set_gyro_scale()` function has been provided. This function takes the following parameters:

- `gyro_fullscale_degrees`: The gyroscope sensitivity specified in degree/s per full-scale value (range is -32768 to +32767). For example, if set to 2000 then a gyro value of 1 corresponds to a rate of rotation of 2000 / 32768 = 0.061 degree/s.
- `gyro_sampling_rate`: Rate at which the gyroscope is sampled in Hz.

Suitable values for `beta_a` and `beta_m` should be selected to match the requirements in terms of the maximum rate of rotation and tolerance to noise. Increasing these values will allow faster rates of rotation to be tracked at the expense of tolerance to accelerometer/magnetometer noise.

`q` must be initialised to the reference orientation [32767, 0, 0, 0] (i.e. upright and facing north).

10.2.3 Processing

Sensor fusion algorithm processing is performed by calling the `SmartFusionAHRS_update()` function on an instantiation of the `SmartFusionAHRS_param` structure. Prior to calling this function, the contents of the vectors referenced by `g_vec_ptr`, `m_vec_ptr` and `a_vec_ptr` should be updated with the appropriate sensor data.

In cases where the sensors are sampled at different rates and data from individual sensors are not available at every processing cycle, the flags for these inputs should be set to '1' when they have data available and '0' when not.

The algorithm uses a right-handed coordinate system, where the x-axis is aligned with north, the y-axis is aligned with east and the z-axis is aligned with down. The gyroscope, accelerometer and magnetometer data must be converted to this coordinate space for the algorithm to function correctly.

Note: Positive gyroscope values represent a clockwise rotation around the associated axis (when looking in the direction it is pointing).

The algorithm also presumes that the sensors have been sampled synchronously at a regular interval and that all distortions (e.g. magnetometer hard/soft iron distortions) have been properly compensated for. Algorithm performance will be degraded when this is not the case.

Once processing is complete, an updated orientation estimate can be read from `q`. This parameter represents the orientation in unit quaternion form, consisting of four elements in Q15 signed fixed point representation.

11 IoT DK Android and iOS application

11.1 Installing from the AppStore

The IoT Sensor application for iOS devices is available for downloading from the AppStore.

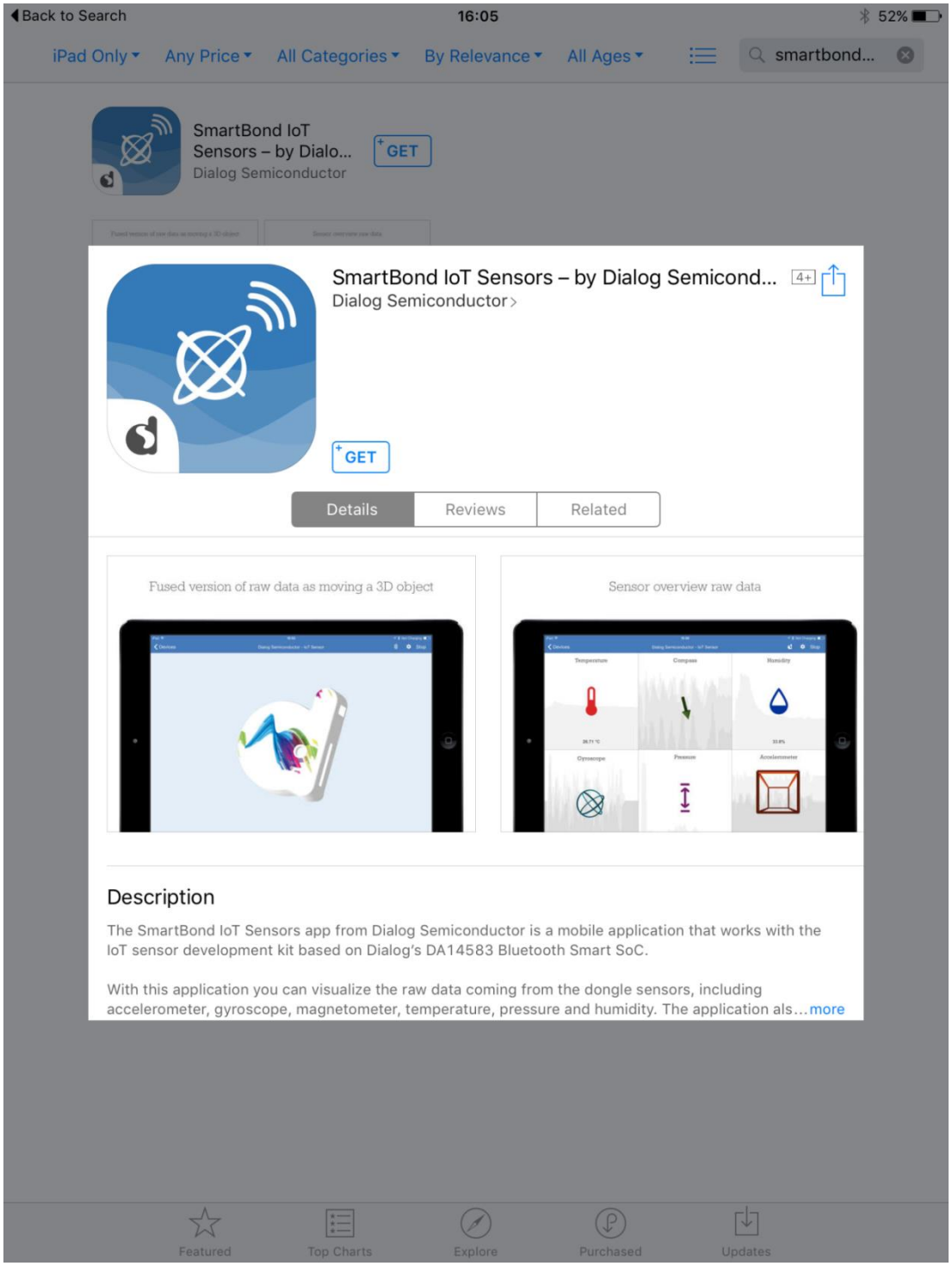

Figure 14: Installing the IoT sensor application from the AppStore

11.2 Installing from the PlayStore

The IoT Sensor application for Android devices is available for downloading from the PlayStore.

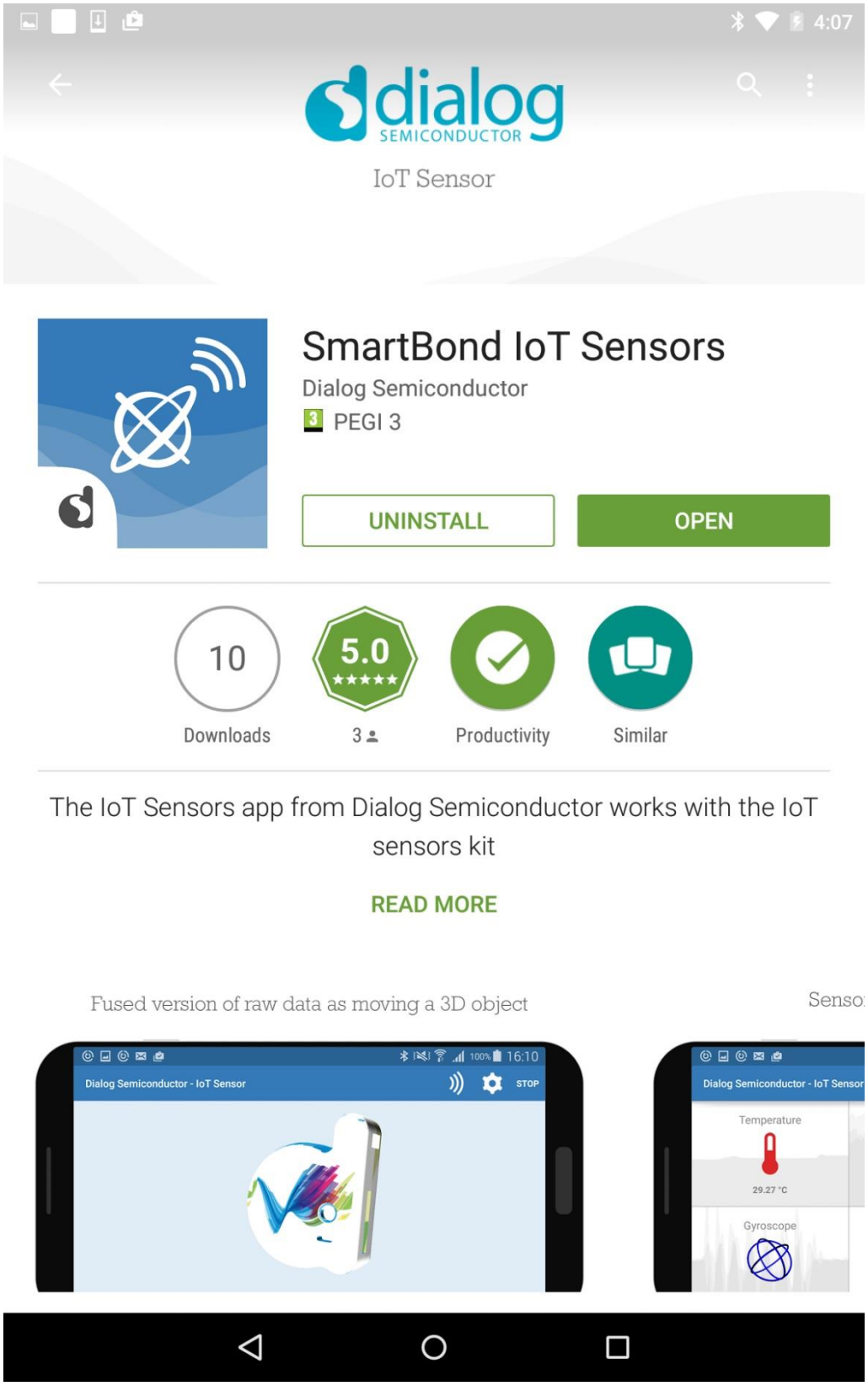

Figure 15: Installing the IoT Sensor application from the PlayStore

11.3 Scan screen

Open the IoT sensor application and move the IoT device to initiate advertising. The IoT devices that are advertising will be listed on the Scan screen (see Figure 16).

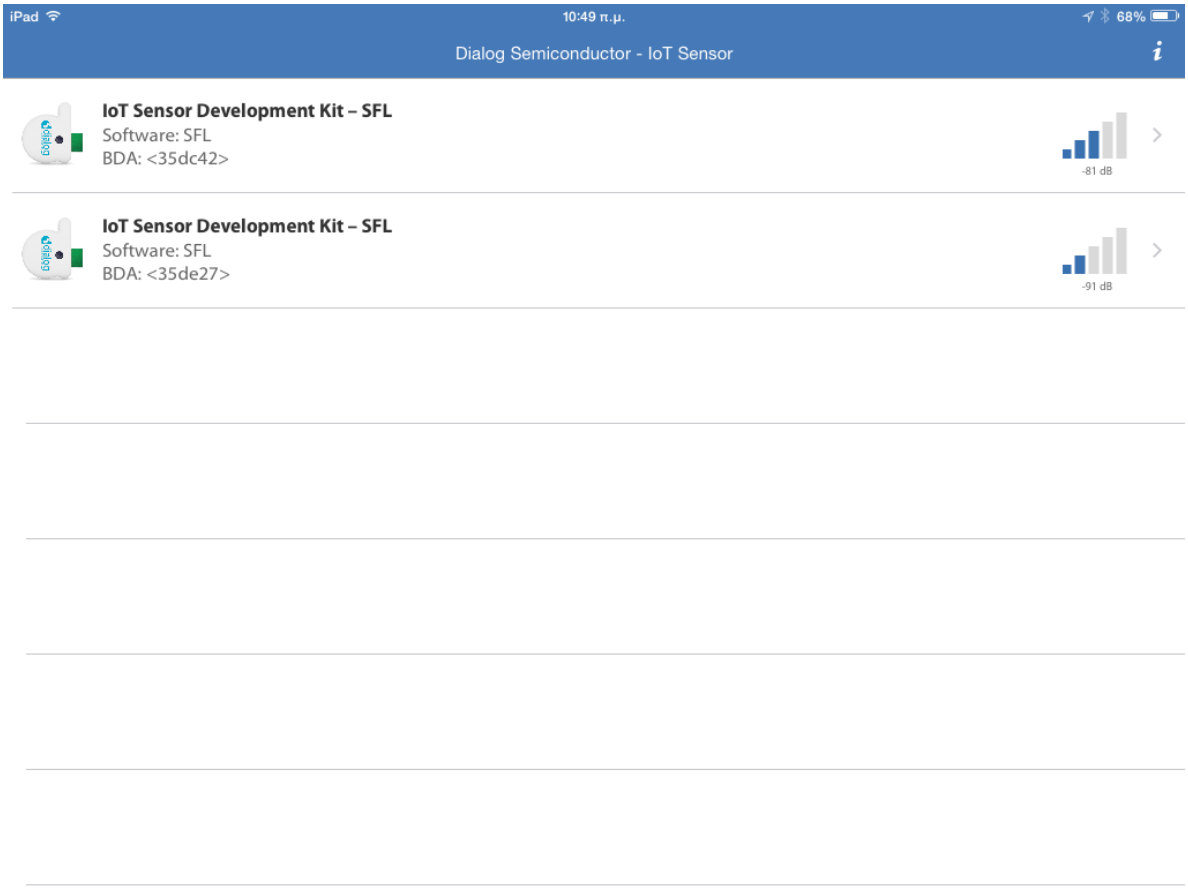

Figure 16: IoT sensor application - Scan screen

Only IoT-DK devices are listed on this page because the application filters out BLE devices that do not have the advertise values listed in section 7.2. Two fields are listed on the Scan screen: the device type and the last three bytes of the BD address. Tap on a device in the list to connect to it. Note that a listed device might not be active (and the LED does not blink) when the advertise timeout has expired. To update the Scan list touch the screen to scroll downwards (iOS) or press the Scan button (Android).

11.4 Sensor screen

Upon successful connection the Sensor screen appears (see Figure 17). This screen illustrates the values of raw sensor data. When an SFL project is programmed, the Dialog ‘d’ symbol (2) appears at the top right of the screen.

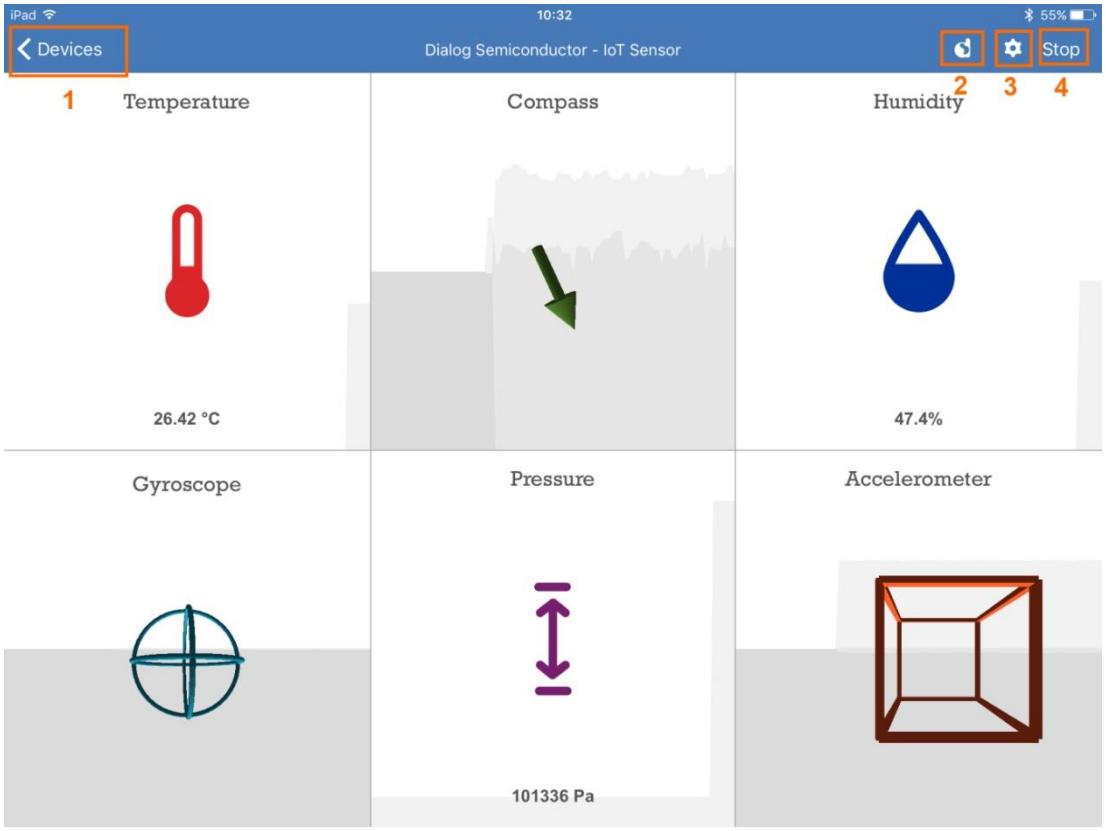

Figure 17: IoT sensor application - Sensor screen

When connected to an IoT sensor device, the application automatically sends a START command (see section 6.1.3.1). The user may at any moment stop data streaming and pause the operation using the STOP button.

The user can navigate the application via the Sensor screen using the screen buttons 1 through 4:

1. Return to the Scan screen (see Figure 16) and disconnect.
2. Go to the 3D screen (SFL project only, see Figure 22).
3. Go to the Configuration screen (see Figure 18).
4. Start or Stop streaming.

11.5 Basic Configuration screen

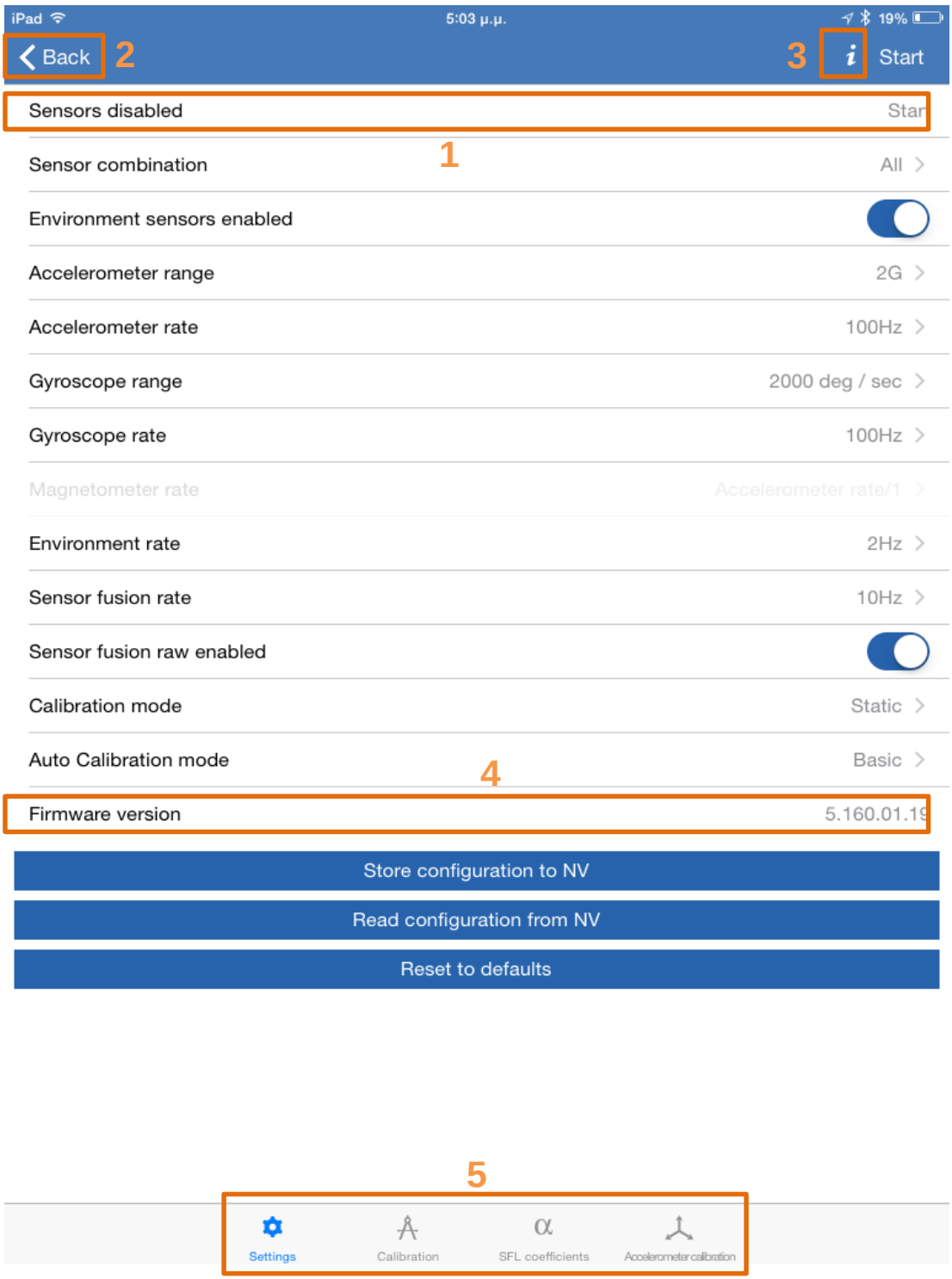

Figure 18: IoT sensor application - Configuration screen

The Configuration screen (see Figure 18) contains Basic and Advanced settings.

Basic settings are:

- Accelerometer range/rate.
- Gyroscope range/rate.

- Magnetometer rate, as a function of accelerometer rate (RAW project only).
- Sensor fusion data rate (SFL project only).
- Environmental sensors enable.
- Sensor fusion raw data transmission enable (SFL project only). The maximum raw data transfer rate to the central device is the Sensor fusion data rate.
- Calibration modes

It is important to note the following points:

- Do not change the settings while the IoT DK is in running state. First press the STOP button.
- A modified setting will be applied after the next START command. The new setting will not be retained upon disconnection, unless the ‘Store configuration to NV’ button has been pressed.
- All modified settings only exist in RAM until the ‘Store configuration to NV’ button is pressed. Press the ‘Read Configuration from NV’ button to restore the previous settings. To restore the default settings press the ‘Reset to defaults’ button.
- The IoT sensor firmware version appears at the bottom of the Configuration screen (4).

At any moment the user may start or stop the device operation using the Sensors Enabled field (1) at the top of the Configuration screen, or return to the Sensor screen by using the Back button (2).

11.6 Magnetometer calibration screen

A number of advanced configuration pages are available (5) in order to precisely adjust parameters of magnetometer calibration and SmartFusion algorithm.

Figure 19: Magnetometer calibration parameters

The calibration page (Figure 19) contains the controls and parameters for the various calibration modes. Only the controls and parameters for the currently enabled calibration mode can be modified and those that apply to other modes are greyed out.

The controls and parameters are:

- Apply – Enables/Disables application of calibration coefficients on raw sensor data.
- Matrix apply – Enables/Disables application of calibration matrix on raw sensor data. If disabled, only offset vector is applied. This control only applies to basic auto-calibration.
- Update – Enables/Disables updating of calibration coefficients.
- Matrix update – Enables/Disables updating of calibration matrix. If disabled, only offset vector is updated. This control only applies to basic auto-calibration.
- Initialise from Static Coeffs – If enabled, the calibration coefficients algorithms are restored from the previous instantiation of the auto-calibration rather than being reset to default values.
- Reference Magnitude – The expected magnitude of the magnetometer vector.
- Magnitude Range – The expected range of magnitude of the magnetometer vector.
- Mu – The rate of convergence.

Note: The fixed point parameters are expressed in their raw integer form. For a more detailed description of the behaviour of these controls and parameters please refer to section 9.2.2.

11.6.1 Magnetometer calibration file

It is possible to send the calibration coefficients for the currently enabled calibration mode from the application to the device. The coefficients must be loaded from file using the associated button. The coefficients can then be stored in Flash memory for subsequent initialisation of the calibration algorithm. Similarly, the current set of calibration coefficients can also be saved to file.

The format of the file uses an **INI** format of the form:

```
[magnetometer calibration]
sensor_type    = 2
q_format       = 14
offset_vector  = 0, 0, 0
matrix         = 16384, 0, 0, 0, 16384, 0, 0, 0, 16384
```

All values are integers with the exception of the `matrix` coefficients which are fixed point values (with precision specified by `q_format`) expressed in raw integer form. The `sensor_type` should always be set to 2 (representing magnetometer).

The `offset_vector` coefficients are listed in [x, y, z] order.

The matrix coefficients are listed in [row, column] order:
`[0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2].`

11.6.2 Magnetometer auto-calibration procedure

When operating in auto-calibration mode it is necessary to perform a calibration procedure in order provide the calibration algorithm with enough raw sensor data for it to accurately determine the hard and soft iron magnetometer distortions that exist and calculate the calibration coefficients required to compensate for them. Until this calibration procedure has been completed, the magnetometer will not be properly calibrated and drift artefacts will be observable in the 3D orientation visualisation.

This calibration procedure involves rotating the board through a representative set of orientations so that a complete sphere of magnetometer vector positions has been captured. In order to produce sufficient data for successful calibration, it is recommended to rotate the board randomly through all orientations as quickly as possible for about 10 to 20 seconds.

The calibration procedure should be performed in a uniform and static magnetic environment. When the magnetic environment changes the calibration procedure must be repeated in order to allow the algorithm to adapt.

When operating in an environment with much variability in the strength and direction of the magnetic field, it is recommended that the auto-calibration procedure is first performed in a controlled location. Auto-calibration should then be disabled by setting the ‘Update’ control setting to OFF to prevent further modification of the calibration coefficients. This will prevent the constantly changing magnetic environment from upsetting the calibration algorithm and introducing drift artefacts in the 3D visualisation.

11.6.2.1 Prerequisites for magnetometer calibration

- The device should be away from any sources of magnetic interference. Electric or electronic devices located near the IoT sensor might influence the outcome. Also ferromagnetic materials used in office desks or chairs should be kept away from the device while calibrating.
- The device should be kept within a single location while calibrating it. The calibration algorithms are designed to compensate within a stable and uniform magnetic environment. When sudden changes in the magnetic environment occur, it will take some time for the calibration algorithm to adapt.
- The magnetization level of the battery is of importance, but the calibration algorithm will fail, even without attached battery, when the other conditions are not met. The temperature of the battery is also important and should be kept stable during the calibration process.
- Make sure the IoT sensor battery is not depleted.

11.6.2.2 Procedure of magnetometer calibration

1. Connect to the device using the provided iOS or Android IoT application.
2. Stop the device, go to the basic configuration page.
3. Select “Reset to Defaults”.
4. Select Calibration Mode: Continuous and Auto Calibration Mode: Basic or SmartFusion.
5. Press “Write configuration to NV” in order to preserve the settings.
6. Press START and rotate the device with random orientations with 1 to 2 rotations per second for 15 to 20 seconds. When rotation is stopped, the 3D image will stabilize and the magnetometer value will be in effect in the sensor fusion algorithm.
7. When the calibration fails, restart the procedure after checking the battery and changing the calibration environment (sources of magnetic interference).
8. Every time STOP is pressed, the calibration coefficients are saved.

11.7 SmartFusion coefficients screen

The SFL coefficients screen (Figure 20) contains parameters for the sensor fusion algorithm. These parameters are:

- Beta A – Accelerometer scaling factor
- Beta M – Magnetometer scaling factor

Note: These are fixed point parameters that are expressed in their raw integer form. For a more detailed description of the behaviour of these parameters please refer to section 10.2.2.

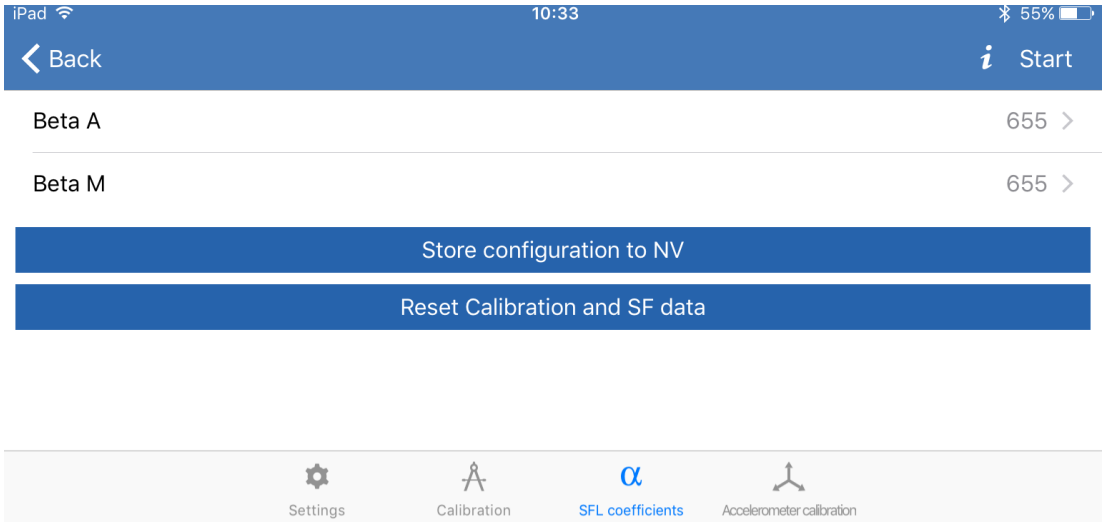

Figure 20: SmartFusion coefficients

11.8 Information screen

Pressing the ‘i’ icon on the Scan and Configuration screens will open the Information screen (Figure 21), which contains the version of the iOS/Android application and contact information.

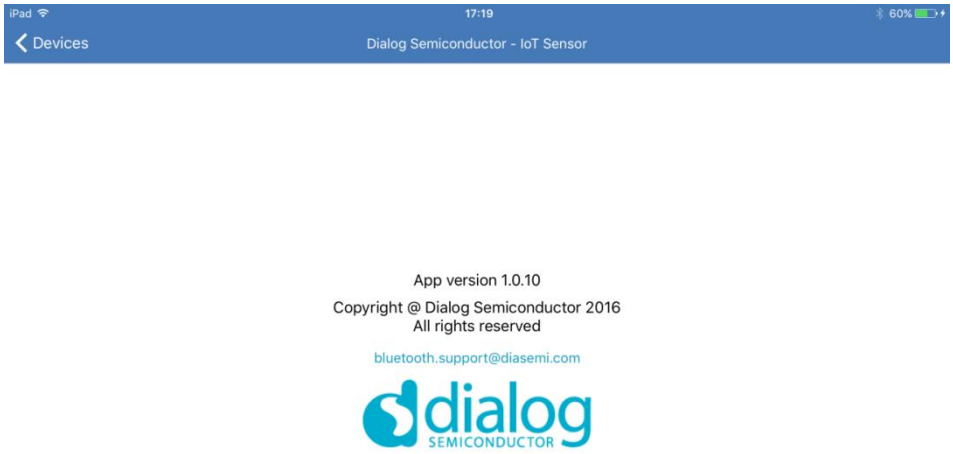

Figure 21: IoT sensor application – Information screen

11.9 3D screen

The user may navigate to the 3D screen (see Figure 22) from the Sensor screen (SFL project only). This is a 3D illustration of the position of the IoT sensor board. Notice that when we move the IoT sensor, the 3D drawing immediately adjusts itself to the new position.

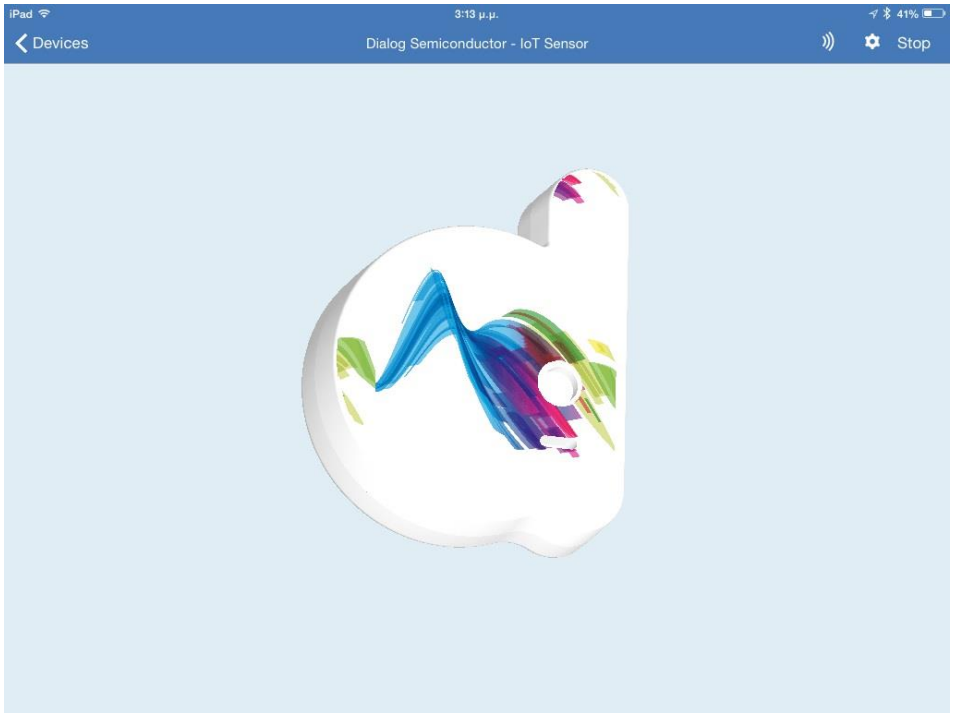

Figure 22: IoT sensor application – 3D screen (SFL only)

11.10 Fast accelerometer calibration screen

Figure 23: Fast accelerometer calibration

Accelerometer offsets around 0G may be observed (Figure 24) when placing the device on a flat horizontal surface. In order to compensate IoT sensor includes a fast calibration method (Figure 23). This method is simple and less accurate, but it is fast and can be performed with little means.

11.10.1 Accelerometer calibration

1. Place the device on a stable flat surface, the LED facing upwards.
2. Connect the device to the IoT Sensor application.
3. Press STOP
4. Navigate to the Fast Accelerometer Page (Figure 23).
5. Press Start Calibration.
6. The test will be concluded within few seconds and the “Calibrated” message will appear. The calibration offsets are now saved in Flash memory and the sensor board has returned to its idle status.

The next time the device is started to the IoT Sensor application the ‘d’ on the 3D screen will be aligned horizontally (Figure 25).

Note: The accelerometer calibration offset will be reset when the device is returned to defaults.

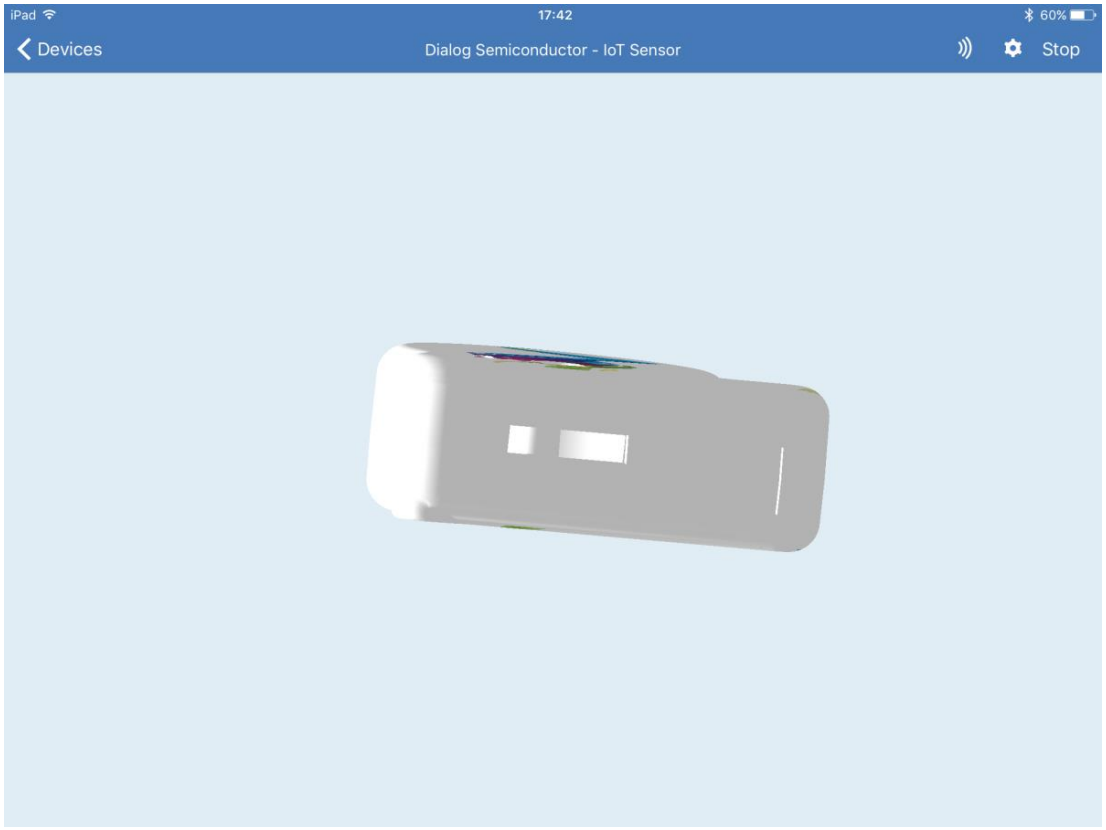

Figure 24: Accelerometer offset tilts the 3D object

Figure 25: Fast accelerometer calibration applied

12 Code size and performance measurements

12.1 Power measurements

12.1.1 Raw data application

Settings:

- Accelerometer: 12.5 Hz
- Gyro: 25 Hz
- Magneto: 12.5 Hz
- Environmental: 0.5 Hz

Consumption:

- Connected idle (no motion): ~560 µA
  - Without gyroscope: ~470 µA
- Connected active (device is moving, gyroscope active): ~1.540 mA

12.1.2 Sensor fusion application

Settings:

- Accelerometer: 12.5 Hz
- Gyro: 25 Hz
- Sensor Fusion data and Magneto: 10 Hz
- Environmental: 0.5 Hz

Consumption:

- Connected: ~1.3 mA
  - Without magnetometer: ~1.17 mA
  - Without gyroscope: 570 µA

12.1.3 Non connected state

Advertising: ~110 µA

Power save mode (not advertising, waiting for accelerometer interrupt to start advertising): ~11 µA

12.2 Code size

12.2.1 Sensor fusion project (SFL)

Code size: 30.5 kB

RAM size: 8.97 kB

Sensor Fusion code size: 1710 B

12.2.2 Raw data project (RAW)

Code size: 28.5 kB

RAM size: 8.42 kB

12.3 Bit rate performance

12.3.1 SFL project bit rate

The bit rate is a function of the sensor rates and the type of application.

For SFL the report size is 11 bytes. When only Sensor Fusion is used at 10 Hz without raw data the bit rate is 10*11 = 110 B/s = 880 bit/s.

For the accelerometer, gyroscope and magnetometer the report size is 9 bytes. In the SFL project the raw data is sent at the Sensor Fusion rate maximum, In RAW project all the raw data is sent.

For example, in a typical SFL project with raw data enabled:

Acc/Gyro/Mag rate: 50 Hz/50 Hz/10 Hz, Sensor Fusion rate 10 Hz

The bit rate is: 10*9 + 10*9 + 10*9 + 10*11 = 380 B/s = 3.04 kbit/s.

12.3.2 RAW project bit rate

In a typical RAW project with all sensors enabled:

Acc/Gyro/Mag rate: 25 Hz/50 Hz/25 Hz

The bit rate is: 25*9 + 50*9 + 25*9 = 900 B/s = 7.2 kbit/s.

Note: In the SFL project the magnetometer rate is the same as the Sensor Fusion rate, in the RAW project the magnetometer rate is the same as the accelerometer rate.

12.4 SmartFusion performance and memory footprint

The performance is expressed in million clock cycles per second (MCPS) of the ARM M0 processor. The memory size is expressed in KiB = 1024 B.

12.4.1 Calibration library

The following figures are for continuous calibration using SmartFusion auto-calibration for sensor data, which is sampled and processed at a rate of 10 Hz.

Peak MCPS: 0.043

Average MCPS: 0.014

Memory size: 2.98 KiB

12.4.2 Sensor fusion library

The following figures are for SmartFusion AHRS processing of gyroscope and accelerometer data sampled at 100 Hz and magnetometer data sampled at 10 Hz.

Peak MCPS: 0.23

Average MCPS: 0.17

Memory size: 1.87 KiB

12.4.3 Calibration and sensor fusion combined

The following figures are for SmartFusion AHRS processing of gyroscope and accelerometer data sampled at 100 Hz and magnetometer data sampled at 10 Hz, with continuous calibration using SmartFusion auto-calibration applied to the magnetometer data.

Peak MCPS: ~0.3

Average MCPS: ~0.2

Memory size: ~5 KiB

12.5 Latency

The latency of the raw data depends on factors such as interrupt latency, connection interval and central application stack processing. With a typical connection interval of 30 ms and using the Android or iOS central application provided by Dialog (see section 11) the raw samples latency has been measured at ~130 ms.

The sensor fusion quaternion output, which is always updated to the latest sensors samples, also takes ~130 ms to travel, same as any report notification. When using a sensor fusion output rate of 10 Hz (100 ms) the maximum delay of the output is ~230 ms. Increasing the sensor fusion rate will decrease the latency, but this is less power efficient.

Appendix A Fixed point representation

Many of the parameters used by the Sensor Calibration and Fusion libraries use fixed point values and their representations are documented using 'Q' notation. These values use underlying integer types but the values they represent are non-integer real numbers.

Fixed point values divide up the bits within the word of the integer type so that the higher order bits represent the integer part of the value and the lower order bits represent the fractional part, with the Q format determining the number of fractional bits. For example, a 16 bit Q8 fixed point value will have 8 lower order fractional bits (bits 0:7) and 8 higher order integer bits (bits 8:15).

To convert a real number to fixed point representation one must simply multiply it by the power of 2 corresponding to the Q format. For example, to convert 0.25 to a Q15 fixed point representation, multiply it by 2^{15} (= 32768) giving the value 8192.

Conversely, to convert a fixed point value to its real representation one must divide it by the power of 2 corresponding to the Q format. For example, to convert 16384 in Q13 fixed point representation to its real value, divide it by 2^{13} (= 8192) giving the value 2.0.

Appendix B Quaternion representation of rotations and orientations

The SmartFusion library makes extensive use of the quaternion number system both to perform the sensor fusion calculations for reasons of computational efficiency and accuracy. The calculated output orientation is also expressed in unit quaternion form, which in this case is represented as a rotation from the reference position (upright and facing north).

Quaternions share similarities with complex numbers and are essentially just a three dimensional quantity expressed in complex form. As such they have a four elements [*a*, *b***i**, *c***j**, *d***k**], the first of which (*a*) represents the real part and the remaining three (*b***i**, *c***j**, *d***k**) represent the imaginary part.

When applied to three-dimensional rotations, the elements of the quaternion are expressed as [**w, x, y, z**]. The imaginary part is a Cartesian vector representing the axis of rotation and the relative magnitudes of the real and imaginary parts represent the amount of rotation around that axis. When the magnitude of the quaternion is not equal to one (not a unit quaternion), an additional scaling is applied to the size of the rotated object.

A more detailed explanation of quaternions and their application to three dimension rotations can be found on Wikipedia [8][9].

Some examples of simple rotations/orientations expressed in unit quaternion form shown in the following figures.

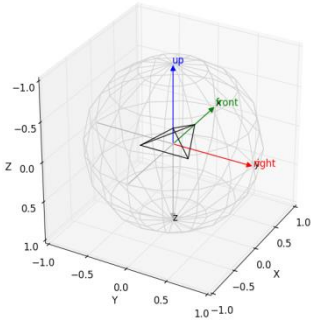

Figure 26: [1, 0, 0, 0], No rotation or upright facing north (reference orientation)

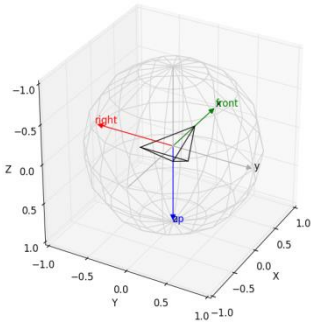

Figure 27: [0, 1, 0, 0], 180° rotation around x-axis or upside down facing north

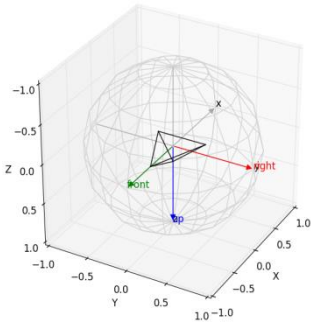

Figure 28: [0, 0, -1, 0], -180° rotation around y-axis or upside down facing south

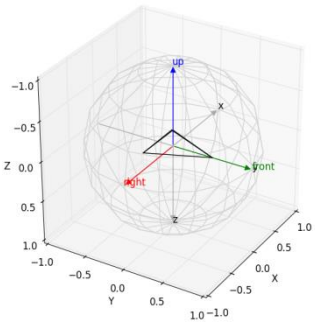

Figure 29: [2-0.5, 0, 0, 2-0.5], 90° rotation around z-axis or upright facing east

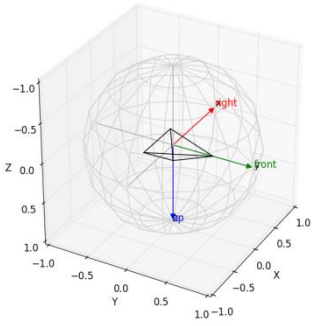

Figure 30: [0, 2-0.5, 2-0.5, 0], 180° rotation around x=y or upside down facing east

Note: The indicated orientations presume a NED coordinate system.

13 Known issues

The range of the accelerometer cannot be changed by the user. The following patch can prevent such issue:

Add the following code @ bmi160.c line 477:

```
s_bmi160.delay_msec(C_BMI160_ONE_U8X);
bmi160_set_accel_range(BMI_160_ACCEL_RANGE);
s_bmi160.delay_msec(C_BMI160_ONE_U8X);
bmi160_set_gyro_range(BMI_160_GYRO_RANGE);
```

Revision history

Revision	Date	Description
1.0	08-Mar-2016	Initial version.
1.1	29-Mar-2016	Rearranged sections 4 and 11. Added sections for CIB connection and magnetometer calibration. Changed figures 16, 20, 23. Added figures 5 and 6.
1.2	02-June-2016	A section “Known issues” has been added

Status definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

Disclaimer

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including without limitation the specification and the design of the related semiconductor products, software and applications.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Customer notes that nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document are subject to Dialog Semiconductor's Standard Terms and Conditions of Sale, unless otherwise stated.

RoHS Compliance

Dialog Semiconductor complies to European Directive 2001/95/EC and from 2 January 2013 onwards to European Directive 2011/65/EU concerning Restriction of Hazardous Substances (RoHS/RoHS2).
Dialog Semiconductor's statement on RoHS can be found on the customer portal https://support.diasemi.com/. RoHS certificates from our suppliers are available on request.

Contacting Dialog Semiconductor

United Kingdom (Headquarters)
Dialog Semiconductor (UK) LTD
Phone: +44 1793 757700

Germany
Dialog Semiconductor GmbH
Phone: +49 7021 805-0

The Netherlands
Dialog Semiconductor B.V.
Phone: +31 73 640 8822

Email:
enquiry@diasemi.com

North America
Dialog Semiconductor Inc.
Phone: +1 408 845 8500

Japan
Dialog Semiconductor K. K.
Phone: +81 3 5425 4567

Taiwan
Dialog Semiconductor Taiwan
Phone: +886 281 786 222

Web site:
www.dialog-semiconductor.com

Singapore
Dialog Semiconductor Singapore
Phone: +65 64 8499 29

Hong Kong
Dialog Semiconductor Hong Kong
Phone: +852 3769 5200

Korea
Dialog Semiconductor Korea
Phone: +82 2 3469 8200

China (Shenzhen)
Dialog Semiconductor China
Phone: +86 755 2981 3669

China (Shanghai)
Dialog Semiconductor China
Phone: +86 21 5424 9058