Application Note

BMF055
Example Project – Data Stream

Bosch Sensortec

BOSCH
Invented for life

Application Note:	**BMF055 Example Project – Data Stream**
Document Revision	1.1
Document Release	October 2015
Document Number	BST-BMF055-EX001-01
Technical Reference	0 273 141 235
Notes	Data in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance.

Contents

1 Preface

The application implemented by this project shows how to configure BMF055 smart sensor to read raw sensor data and print it on a terminal software running on a host computer.

The chip uses a USART interface to communicate to a host computer or another MCU. It receives commands and sends messages via USART.

The project is implemented on BMF055 as an all-in-one sensor solution. The project uses a BMF055 shuttle board and the necessary connections to power-up and program the chip.

This project is a reference example that shows how to use basic functions of BMF055 and can be extended and altered to implement desired custom use cases.

2 Prerequisites

In order to program the sensor and run this example, necessary connections should be established on the application board. Detailed steps to set up the platform is provided in a separate document.

A USART connection to a host computer is also needed to be able to see the outcome of the project on a terminal software.

3 Quick Setup

This chapter gives step by step instructions on how to start running this example on a BMF055 Shuttle Board.

3.1 Software and Extensions

Install the latest version of Atmel Studio from Atmel website

1. Open Atmel Studio
2. Go to “Tools -> Extension Manager” and install the latest version of Atmel Software Framework (Version used in this extension is 3.26.0)
3. Go to “Tools -> Extension Manager” and search for “BMF055 Shuttle Board – Data Stream” extension from Bosch Sensortec GmbH (BST) and install it
4. Go to “Tools -> Extension Manager” and search for “Terminal for Atmel Studio” extension from Atmel and install it (It is not necessary to install this extension if you are going to use another terminal software)
5. Restart Atmel Studio
6. Go to “File -> New -> Example Projects”
7. “Below BST – Bosch Sensortec GmbH” find the project named “BMF055_SHUTTLE_BOARD_DATA_STREAM – atsamd20j18a”
8. Select it and press “OK” button
9. Read and accept the license agreement and press “Finish” button to create a new example project

3.2 Hardware

10. Establish the minimum necessary connections; including power, reset and programmer/debugger.
11. Establish a USART connection between the shuttle board and a host computer[*]. Use bridges if necessary.
12. Install required drivers for your virtual COM port.
13. Go to “Start Menu -> Control Panel -> Device Manager”
14. Below “Ports (COM and LPT)” find the virtual COM port that you are going to use and note the COM Port Number

[*] It is assumed that the shuttle board would be interfaced to a terminal software running on a host computer.

15. In Atmel Studio go to “Project -> Properties” and select the tab named “Tool”

16. Below “Selected debugger/programmer” select the “SAM-ICE” tool. And select “SWD” as the interface and save the changes.

3.3 Run the Project

17. In Atmel Studio to “Build -> Build Solution”

    The build process should succeed with no errors or warnings.

18. Go to “Debug -> Start Without Debugging”

19. Wait for the process to be done.

    (Notice the “Ready” message below, on the status bar)

20. Go to “View -> Terminal Window”

21. Select the virtual COM port number that you have previously noted, set Baud to 115200 and select ASCII as terminal’s input format.

22. Press “Connect”

3.4 Check the Use-case

23. As soon as serial connection is established and the code starts running the data stream appears on terminal window.

24. In order to change sensors’ configuration refer to Example Commands.

4 Application Overview

For detailed description of application implementation see Application.

This example code reads the data from the three internal sensors and transmits them out using a UART communication.

The MCU reads sensor data from the internal SPI bus. The SPI bus has a frequency of 10 MHz and sensor data is requested on a period of 100 ms (can be changed, refer to Macro TC6_COUNT_VALUE). Raw sensor data (Binary) is then transmitted using UART with baud rate of 115200 bps. Output can be read on a terminal as shown in Figure 1. Atmel Studio terminal extension can be used here.

The first row includes accelerometer's data for three axes separated with spaces; the second row includes gyroscope's data for the three axes separated with spaces; and the third row includes magnetometer's data for the three axes separated with spaces. Then there would be an empty line and the same structure will be repeated.

Output data is in **ASCII** format and input data is in **hexadecimal**. In order to start the stream the "stream command" (0x01) should be sent by the user and the stream will continue until the "stop command" (0x00) is given. (Started by default)

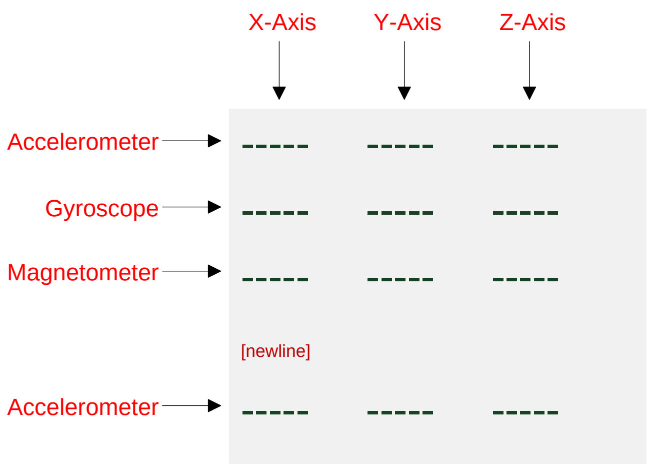


Figure 1- Terminal output for Sensor Data Stream

4.1 MCU Peripherals

Microcontroller's peripherals that are used in this project are listed in Table 1 along with their functionality for this application.

For more information about the peripherals configuration refer to chapter9 ASF Driver Supports Implementation.

Table 1 - MCU Peripherals

Peripheral	Functionality
Clock Source OSC8M	8 MHz clock source of MCU used as the source for multiple modules (Frequency 2 MHz)
Clock Source DFLL48M	DFLL clock source of MCU used as the source for multiple modules (Frequency 24 MHz – Open Loop)
GCLK 0	Generates the main system clock, using DFLL as its source (Frequency 24 MHz)
GCLK 1	Generates clock signal for TC4, using OSC8M as its source (Frequency 500 KHz)
GCLK 2	Generates clock signal for USART, using OSC8M as its source (Frequency 2 MHz)
Timer/Counter 4	Implementing delay function to be used by sensor API (in terms of milliseconds)
Timer/Counter 6	Scheduling sensor data read and USART transmission (Default Period 100 ms)
SERCOM 3 SPI	Communication between MCU and sensors (SPI Master, MSB first, Character size = 8 bits, Frequency 10 MHz)
SERCOM 5 USART	Communication between MCU and the host computer (Baud rate = 112500, Data bits = 8, Parity = None, Stop bit = 1)

4.2 BMF055 Configuration

The example first initializes the sensors before the streaming starts. The default configurations of sensors are as follows.

Table 2 - Default Configuration of BMA280

Parameter	Value
Accelerometer Range	±2 g
Bandwidth	1000 Hz
Power Mode	Normal Mode

Table 3 - Default Configuration of BMG160

Parameter	Value
Gyroscope Range	2000 °/s
Bandwidth	523 Hz (Unfiltered)
Data Rate	2000 Hz
Power Mode	Normal Mode

Table 4 - Default Configuration of BMM150

Parameter	Value
Data Rate	10 Hz
Preset	Regular
Power Mode	Normal Mode

4.2.1 Custom Configuration

User can change sensors’ configuration sending commands via UART interface. List of command values for BMA280, BMG160 and BMM050 are given in tables Table 5, Table 6 and Table 7 respectively.

In order to skip changing a configuration setting, 0xFF value can be given. This value results in no change in corresponding setting to the specific byte.

Note that bytes have to be sent one at a time and in order to change configurations data stream has to be stopped first (sending 0x00).

Figure 2 illustrates the state diagram of this process. Transitions marked by star signs (*) are taken if values in the correct range are given. In case of invalid values a transition to “Stopped” state will be taken. (Transitions to “Stopped” caused by out of range values are not shown in the diagram)

Sample commands are provided in chapter Example Commands.

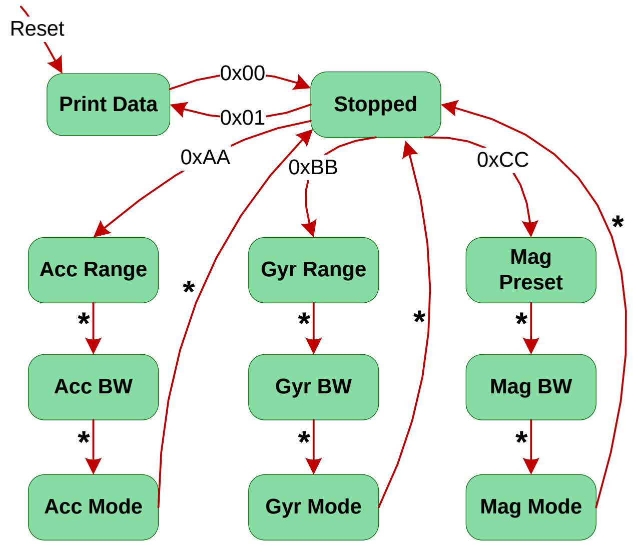


Figure 2 - UART Input Process Statechart

Table 5 - Accelerometer Configuration Commands

Command Value	Function
0xAA	Accelerometer Select
0x03	Range ±2 g
0x05	Range ±4 g
0x08	Range ±8 g
0x0C	Range ±16 g
0x08	Bandwidth 8 Hz
0x09	Bandwidth 16 Hz
0x0A	Bandwidth 31 Hz
0x0B	Bandwidth 63 Hz
0x0C	Bandwidth 125 Hz
0x0D	Bandwidth 250 Hz
0x0E	Bandwidth 500 Hz
0x0F	Bandwidth 1000 (Unfiltered)
0x00	PM Normal Mode
0x01	Mode Low Power 1
0x02	Mode Suspend
0x03	Mode Deep Suspend
0x04	Mode Low Power 2
0x05	Mode Standby Mode

Table 6 - Gyroscope Configuration Commands

Command Value	Function
0xBB	Gyroscope Select
0x04	Range 125 °/s
0x03	Range 250 °/s
0x02	Range 500 °/s
0x01	Range 1000 °/s
0x00	Range 2000 °/s
0x05	Bandwidth 12 Hz
0x04	Bandwidth 23 Hz
0x07	Bandwidth 32 Hz
0x03	Bandwidth 47 Hz
0x06	Bandwidth 64 Hz
0x02	Bandwidth 116 Hz
0x01	Bandwidth 230 Hz
0x00	Bandwidth Unfiltered
0x00	Mode Normal
0x01	Mode Deep Suspend
0x02	Mode Suspend
0x03	Mode Fast Power Up
0x04	Mode Advanced Power Saving

Table 7 - Magnetometer Configuration Commands

Command Value	Function
0xCC	Magnetometer Select
0x01	Preset Low Power
0x02	Preset Regular
0x03	Preset High Accuracy
0x04	Preset Enhanced
0x01	Data Rate 2 Hz
0x02	Data Rate 6 Hz
0x03	Data Rate 8 Hz
0x00	Data Rate 10 Hz
0x04	Data Rate 15 Hz
0x05	Data Rate 20 Hz
0x06	Data Rate 25 Hz
0x07	Data Rate 30 Hz
0x00	Mode Normal
0x01	Mode Forced
0x02	Mode Suspend
0x03	Mode Sleep

5 Hardware Platform

The hardware platform consists of a BMF055 chip as the main processor unit; which is mounted on a shuttle board. In order to power up and run the system, certain connections to the shuttle board are necessary (such as power and program/ debug).

5.1 BMF055

BMF055 is a 9-axis orientation sensor, which includes sensors and a microcontroller in a single package.

BMF055 is a System in Package (SiP), integrating an accelerometer (BMA280), a gyroscope (BMG160), a geomagnetic sensor (BMM050) and a 32-bit microcontroller (ATSAMD20J18A).

Figure 3 shows the basic building blocks of the BMF055 device.

For more information about the device refer to BMF055 Datasheet.

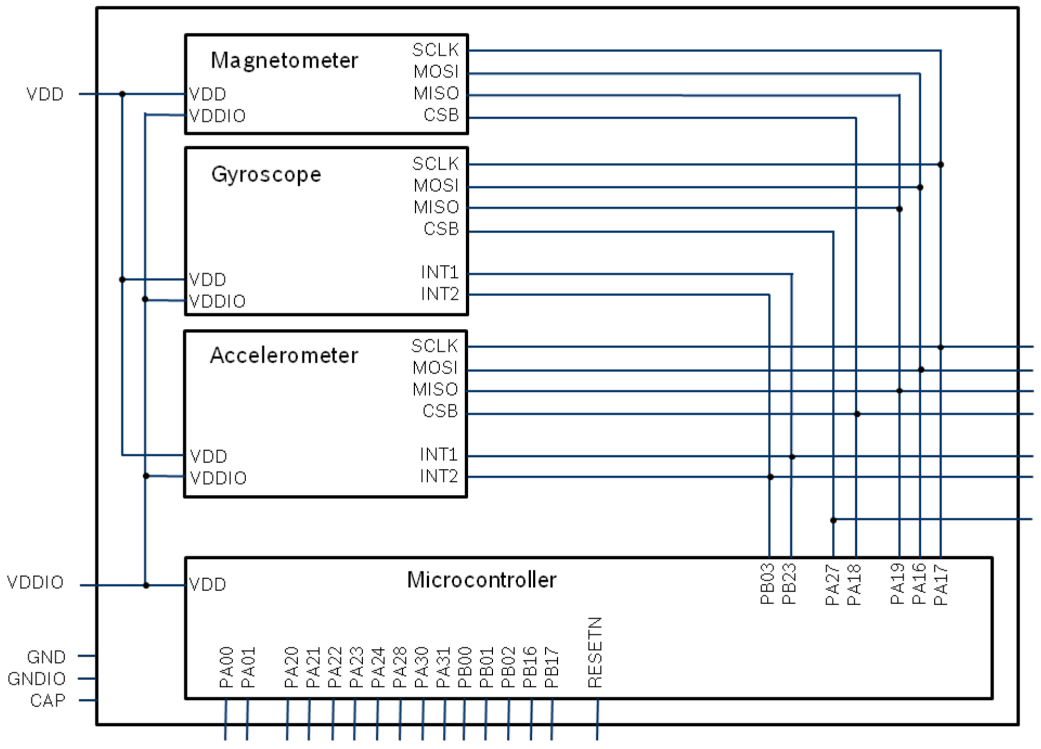


Figure 3 - BMF055 Architecture

5.2 BMF055 Shuttle Board

Bosch Sensortec BMF055 shuttle board is a PCB with a BMF055 Orientation Sensor mounted on it. It has the required decoupling capacitors, an external 32 KHz crystal and its load capacitors and allows easy access to the sensors pins via a simple socket.

Table 8 - BMF055 Shuttle Board Pin-out

Pin No.	Pin Name	BMF055 Pin Connected	SAMD20 Pin Connected	Description
1	VDD	3		VDD
2	VDDIO	28	-	VDDIO
3	GND	2, 25	-	GND
4	MISO	-	-	DNC
5	MOSI	-	-	DNC
6	SCK	-	-	DNC
7	CS	-	-	DNC
8	IO5/INTA	6	PB00	GPIO
9	IO0	5	PB01	GPIO
10	COD_GND	-	-	DNC
11	COD_GND	-	-	DNC
12	COD_GND	-	-	DNC
13	COD_GND	-	-	DNC
14	IO1	4	PB02	GPIO
15	IO2	16	PA22	GPIO
16	IO3	15	PA23	GPIO
17	SDA	20	PB16	GPIO
18	SCL	19	PB17	GPIO
19	IO8	11	RESET	RESET
20	INTB/IO6	10	PA28	GPIO
21	INTC/IO7	14	PA24	GPIO
22	IO4	17	PA21	GPIO
23	COD_GND	-	-	DNC
24	COD_PULL	-	-	DNC
25	COD_GND	-	-	DNC
26	COD_GND	-	-	DNC
27	COD_PULL	-	-	DNC
28	COD_PULL	-	-	DNC
29	SWCLK	8	PA30	Debugging CLK
30	SWDIO	7	PA31	Debugging IO

The shuttle board can be plugged into Bosch Sensortec development tools, custom designed boards or breadboards.

5.3 Power

BMF055 has two distinct power supply pins:

- VDD is the main power supply for the internal sensors
- VDDIO is a separate power supply pin used for the supply of the MCU and the digital interfaces

The voltage supply range for VDD is 2.4V to 3.6V and for VDDIO is 1.7V to 3.6V.

For the switching sequence of power supply VDD and VDDIO it is mandatory that V_{DD} is powered on and driven to the specified level before or at the same time as V_{DDIO} is powered ON. Otherwise there are no limitations on the voltage levels of both pins relative to each other, as long as they are used within the specified operating range.

5.4 Programming/ Debugging

Programming and debugging of the chip is done Serial Wire Debug Interface available. Any debugger that supports the interface can be used. (e.g Atmel SAM-ICE)

5.5 USART Interface

USART pin assignment is given in Table 9. This interface can be used to connect the system to another microcontroller or to a host computer using an RS-232 or a USB bridge.

Table 9 - USART Pin Assignment

Shuttle Board Pin #	Description
17	Tx
18	Rx

6 Design Structure

Figure 4 shows structure of the design, hardware layer and software layers above it.

Atmel Software Framework (ASF) drivers are modules provided by Atmel. They are included in the project unchanged. For more information refer to Atmel application notes listed in Atmel Resources. The version used in this project is ASF (3.26.0).

ASF driver supports are support files for different components of MCU used in the project (Clock, SPI, TC and USART). For each component, they include corresponding driver module and define some wrapper functions (mostly for configuration), callback handlers and few other functions that are needed in the application or sensor APIs. For more information refer to ASF Driver Support.

Sensor supports are support files for the three sensors: BMA2x2, BMG160 and BMM050. They define functions to implement communications between sensor APIs and ASF Drivers (SPI and TC). For more information refer to Sensor API Supports Implementation.

Sensor APIs are provided by Bosch Sensortec and define structures and functions to communicate with sensors. For more information refer to Bosch Sensortec Resources.

For more information about application refer to Quick Setup and Application Implementation.

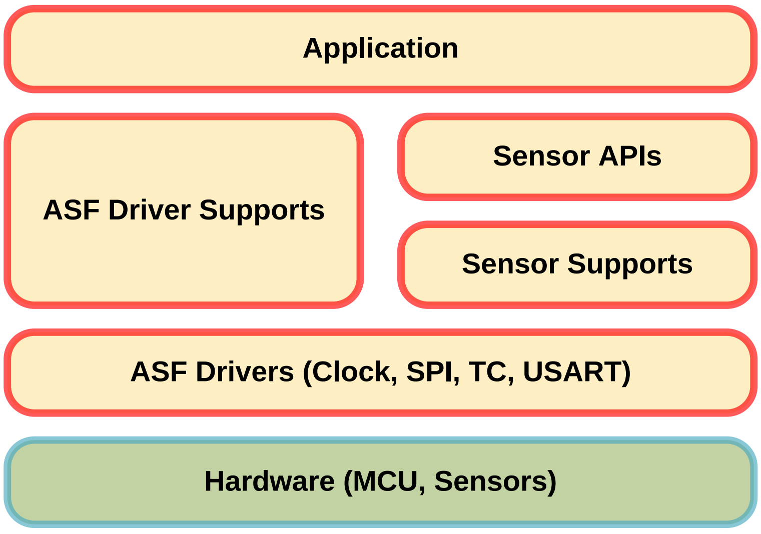
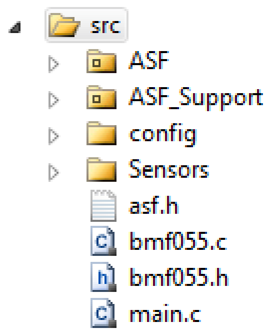


Figure 4 - Design Structure

6.1 Folder Structure

Figure 5 shows the project folder structure.

- “ASF” folder contains the Atmel Software Framework (ASF).
- “ASF_Support” folder contains ASF driver support files to abstract ASF to a higher level which makes them easier to be used in this application.
- “Sensors” Folder contains sensor APIs and sensor API support files. Support files implement the functions that are needed so that the API functions can make use of ASF.
- “asf.h” is the interface header for the Atmel MCU framework.
- “bmf055” files implement application specific functionalities, such as initializing the whole system, configuring required MCU components and sensors, reading and streaming sensor data.
- “main.c” defines the main function of the project.



```
src
  ASF
  ASF_Support
  config
  Sensors
  asf.h
  bmf055.c
  bmf055.h
  main.c
```

Figure 5 - Project Folder Structure

7 Application Implementation

7.1 Main

This chapter describes the main function of the project defined in “main.c” file.

7.1.1 Include

7.1.1.1 Include “bmf055.h”

Application functions, global variables, macro definitions and libraries needed

7.1.2 Function Definition

7.1.2.1 Function main

Initializes the whole system and runs the desired application

```
int main(void)
```

This is the main function of the project. It calls initialization functions of the MCU components and the sensors.

In the infinite loop it repeatedly executes two main tasks:

- Reads USART module read buffer and in case there is any input available the corresponding process function is called.
- Streams sensor data periodically (100 ms) via USART.

7.2 BMF055

This chapter describes the functions declared in “bmf055” file and defined in “bmf055.c” file.

Bmf055 uses Atmel drivers, sensor APIs and driver support facilities to implement the desired application. The application specific functions, constants and variables are all defined here.

7.2.1 Includes

7.2.1.1 Include “asf.h”

All Atmel Software Framework driver files required by the modules added to the project by ASF wizard

7.2.1.2 Include “clock_support.h”

Wrapper functions for ASF clock and generic clock modules

7.2.1.3 Include “spi_support.h”

Wrapper functions for ASF serial peripheral interface module

7.2.1.4 Include “usart_suppot.h”

Wrapper functions for ASF USART module

7.2.1.5 Include “bma2x2_support.h”

Wrapper functions for BMA2x2 connection to the MCU

7.2.1.6 Include “bmg160_support.h”

Wrapper functions for BMG160 connection to the MCU

7.2.1.7 Include “bmm050_support.h”

Wrapper functions for BMM050 connection to the MCU

7.2.2 Type Definitions

7.2.2.1 Typedef enum usart_input_state_type

```
typedef enum usart_input_state_type
{
	USART_INPUT_STATE_PRINT_DATA,
	USART_INPUT_STATE_STOPPED,
	USART_INPUT_STATE_ACC_RANGE,
	USART_INPUT_STATE_ACC_BW,
	USART_INPUT_STATE_ACC_MODE,
	USART_INPUT_STATE_GYR_RANGE,
	USART_INPUT_STATE_GYR_BW,
	USART_INPUT_STATE_GYR_MODE,
    USART_INPUT_STATE_MAG_PRESET,
    USART_INPUT_STATE_MAG_BW,
    USART_INPUT_STATE_MAG_MODE,
    USART_INPUT_STATE_LAST = USART_INPUT_STATE_MAG_MODE
}usart_input_state_t;
```

These type values are states of USART input command process. The state diagram is shown in Figure 2.

7.2.3 Macro Definitions

7.2.3.1 Macro READ_SENSORS_FLAG

```
#define READ_SENSORS_FLAG           tc6_callback_flag
```

TC6 callback flag is defined as read sensors flag; i.e. sensors’ data are read in accordance with TC6 callback.

7.2.3.2 Macro USART_COMMAND_PROCESS_FLAG

```
#define USART_COMMAND_PROCESS_FLAG usart_callback_receive_flag
```

USART receive callback flag is defined as command process flag; i.e. USART command process is executed in accordance with USART receive callback.

7.2.4 Global Variables/ Structures

7.2.4.1 Usart_input_state_type bmf055_input_state

```
usart_input_state_t bmf055_input_state
```

This type holds the state of USART input command process.

7.2.4.2 Integer bmf055_usart_input_buf

```
uint16_t bmf055_usart_input_buf
```

This unsigned 16 bit long integer holds data received via USART.

7.2.5 Local Variables/ Structures

7.2.5.1 Structure bma2x2_accel_data

```
static struct bma2x2_accel_data bma2x2_accel_data
```

This structure holds acceleration data of x, y and z axes.

7.2.5.2 Structure bmg160_gyro_data

```
static struct bmg160_data_t bmg160_gyro_data
```

This structure holds angular velocity data of x, y and z axes.

7.2.5.3 Structure bmm050_mag_data

```
static struct bmm050_mag_data_s16_t bmm050_mag_data
```

This structure holds magnetic field data of x, y and z axes.

7.2.6 Function Definitions

7.2.6.1 Function bmf055_sensors_initialize

Initializes the internal sensors of BMF055

```
void bmf055_sensors_initialize(void)
```

This function calls initialization functions of the three sensors of BMF055; namely BMA280, BMG160 and BMM050.

7.2.6.2 Function bmf055_sensors_data_print

Sends sensor data via USART

```
void bmf055_sensors_data_print(void)
```

This Function reads sensors' output data and sends them via USART in ASCII Format in the order shown in Figure 1.

7.2.6.3 Function bmf055_usart_read_process

Processes USART input commands

```
void bmf055_usart_read_process(void)
```

This function processes received bytes from the UART interface as commands in hexadecimal format and interprets them to change sensors' configurations as described in tables Table 5, Table 6 and Table 7. It implements the state diagram of Figure 2.

8 Sensor API Supports Implementation

8.1 BMA2x2 Support

This chapter describes the declarations and definitions in “bma2x2_support.h” and “bma2x2_support.c” files.

BMA2x2 support defines functions to interface the sensor API with the actual BMA280 accelerometer via SPI. It implements bus read/ write and delay functions that are needed for this communication. It also defines the sensor initialization routine.

8.1.1 Includes

8.1.1.1 Include “bma2x2.h”

Includes BMA2x2 sensor API which provides sensor’s data structures and driver functions

8.1.1.2 Include “spi_support.h”

Includes ASF SPI driver support file which provides SPI master instance, configuration functions and driver functions

8.1.1.3 Include “tc_support.h”

Includes ASF TC driver support file which provides TC instances, configuration functions and driver functions

8.1.2 Type Definitions

N/A

8.1.3 Macro Definitions

8.1.3.1 Macro BMA2x2_SS_PIN

```
#define BMA2x2_SS_PIN   PIN_PA18
```

BMA2x2 SPI slave select pin

8.1.4 Global Variables/ Structures

8.1.4.1 Structure bma2x2

```
struct bma2x2_t bma2x2
```

Instantiates a bma2x2 software instance structure, which holds relevant information about BMA2x2 and links communication to the SPI bus.

8.1.4.2 Structure bma2x2_slave

```
struct spi_slave_inst bma2x2_spi_slave
```

It instantiates an SPI slave software instance structure, used to configure the correct SPI transfer mode settings for an attached slave (here BMA280 is the slave). For example it holds the SS pin number of the corresponding slave.

8.1.5 Function Definitions

8.1.5.1 Function bma_init

Initializes BMA280 accelerometer sensor and its required connections

```
void bma_init(void)
```

This function configures BMA280 as an SPI slave module, sets the *bus_write*, *bus_read* and *delay_msec* functions of the sensor API to point to the provided functions so that the sensor can communicate with the MCU via SPI. It also initializes the sensor.

8.1.5.2 Function bma_spi_write

Sends data to BMA280 via SPI

```
int8_t bma_spi_write(uint8_t dev_addr,
                     uint8_t reg_addr,
                     uint8_t *reg_data,
                     uint8_t length)
```

This Function implements *bus_write* function, which is used by sensor API to send data and commands to BMA280 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 10 - Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of destination register
[in]	reg_data	Data buffer to be sent
[in]	length	Length of the data to be

		sent

8.1.5.3 Function bma_spi_read

Receives data from BMA280 on SPI

```
int8_t bma_spi_read(uint8_t dev_addr,
                    uint8_t reg_addr,
                    uint8_t *rx_data,
                    uint8_t length)
```

This Function implements *bus_read* function, which is used by sensor API to receive data from BMA280 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 11 - Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of source register
[out]	rx_data	Data buffer to be received
[in]	length	Length of the data to be received

8.1.5.4 Function bma_delay_msec

Initiates a delay of the length of the argument in milliseconds

```
void bma_delay_msec(uint32_t msec)
```

This function implements *delay_msec* function which is used by the sensor API to cause enough delay for the sensor so that it executes specific commands or provides required data.

Table 12 - Parameters

Data Direction	Parameter Name	Description
[in]	msec	Delay length in terms of milliseconds

8.2 BMG160 Support

This chapter describes the declarations and definitions in "bmg160_support.h" and "bmg160_support.c" files.

BMG160 support defines functions to interface the sensor API with the actual BMG160 gyroscope via SPI. It implements bus read/ write and delay functions that are needed for this communication. It also defines the sensor initialization routine.

8.2.1 Includes

8.2.1.1 Include "bmg160.h"

Includes BMG160 sensor API which provides sensor's data structures and driver functions

8.2.1.2 Include "spi_support.h"

Includes ASF SPI driver support file which provides SPI master instance, configuration functions and driver functions

8.2.1.3 Include "tc_support.h"

Includes ASF TC driver support file which provides TC instances, configuration functions and driver functions

8.2.2 Type Definitions

N/A

8.2.3 Macro Definitions

8.2.3.1 Macro BMG160_SS_PIN

```
#define BMG160_SS_PIN   PIN_PA27
```

BMG160 SPI slave select pin

8.2.4 Global Variables/ Structures

8.2.4.1 Structure bmg160

```
struct bmg160_t bmg160
```

Instantiates a bmg160 software instance structure, which holds relevant information about BMG160 and links communication to the SPI bus.

8.2.4.2 Structure bmg160_slave

```
struct spi_slave_inst bmg160_spi_slave
```

It instantiates an SPI slave software instance structure, used to configure the correct SPI transfer mode settings for an attached slave (here BMG160 is the slave). For example it holds the SS pin number of the corresponding slave.

8.2.5 Function Definitions

8.2.5.1 Function bmg_init

Initializes BMG160 gyroscope sensor and its required connections

```
void bmg_init(void)
```

This function configures BMG160 as an SPI slave module, sets the *bus_write*, *bus_read* and *delay_msec* functions of the sensor API to point to the provided functions so that the sensor can communicate with the MCU via SPI. It also initializes the sensor.

8.2.5.2 Function bmg_spi_write

Sends data to BMG160 via SPI

```
int8_t bmg_spi_write(uint8_t dev_addr,
                     uint8_t reg_addr,
                     uint8_t *reg_data,
                     uint8_t length)
```

This Function implements *bus_write* function, which is used by sensor API to send data and commands to BMG160 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 13 - Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of destination register
[in]	reg_data	Data buffer to be sent
[in]	length	Length of the data to be sent

8.2.5.3 Function bmg_spi_read

Receives data from BMG160 on SPI

```
int8_t bmg_spi_read(uint8_t dev_addr,
                    uint8_t reg_addr,
                    uint8_t *rx_data,
                    uint8_t length)
```

This Function implements *bus_read* function, which is used by sensor API to receive data from BMG160 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 14 - Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of source register
[out]	rx_data	Data buffer to be received
[in]	length	Length of the data to be received

8.2.5.4 Function bmg_delay_msec

Initiates a delay of the length of the argument in milliseconds

```
void bmg_delay_msec(uint32_t);
```

This function implements *delay_msec* function which is used by the sensor API to cause enough delay for the sensor so that it executes specific commands or provides required data.

Table 15 - Parameters

Data Direction	Parameter Name	Description
[in]	msec	Delay length in terms of milliseconds

8.3 BMM050 Support

This chapter describes the declarations and definitions in “bmm050_support.h” and “bmm050_support.c” files.

BMM050 support defines functions to interface the sensor API with the actual BMM050 magnetometer via SPI. It implements bus read/ write and delay functions that are needed for this communication. It also defines the sensor initialization routine.

8.3.1 Includes

8.3.1.1 Include “bmm050.h”

Includes BMM050 sensor API which provides sensor’s data structures and driver functions

8.3.1.2 Include “spi_support.h”

Includes ASF SPI driver support file which provides SPI master instance, configuration functions and driver functions

8.3.1.3 Include “tc_support.h”

Includes ASF TC driver support file which provides TC instances, configuration functions and driver functions

8.3.2 Type Definitions

N/A

8.3.3 Macro Definitions

8.3.3.1 Macro BMM050_SS_PIN

```
#define BMM050_SS_PIN   PIN_PA18
```

BMA2x2 SPI slave select pin

8.3.4 Global Variables/ Structures

8.3.4.1 Structure bmm050

```
struct bmm050 bmm050
```

Instantiates a bmm050 software instance structure, which holds relevant information about BMM050 and links communication to the SPI bus.

8.3.4.2 Structure bmm050_slave

```
struct spi_slave_inst bmm050_spi_slave
```

It instantiates an SPI slave software instance structure, used to configure the correct SPI transfer mode settings for an attached slave (here BMM050 is the slave). For example it holds the SS pin number of the corresponding slave.

8.3.5 Function Definitions

8.3.5.1 Function bmm_init

Initializes BMM050 magnetometer sensor and its required connections

```
void bmm_init(void)
```

This function configures BMM050 as an SPI slave module, sets the *bus_write*, *bus_read* and *delay_msec* functions of the sensor API to point to the provided functions so that the sensor can communicate with the MCU via SPI. It also initializes the sensor.

8.3.5.2 Function bmm_spi_write

Sends data to BMM050 via SPI

```
int8_t bmm_spi_write(uint8_t dev_addr,
                     uint8_t reg_addr,
                     uint8_t *reg_data,
                     uint8_t length)
```

This Function implements *bus_write* function, which is used by sensor API to send data and commands to BMM050 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 16 - Parameters

Data Direction	**Parameter Name**	**Description**
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of destination register
[in]	reg_data	Data buffer to be sent
[in]	length	Length of the data to be sent

8.3.5.3 Function bmm_spi_read

Receives data from BMM050 on SPI

```
int8_t bmm_spi_read(uint8_t dev_addr,
                    uint8_t reg_addr,
                    uint8_t *rx_data,
                    uint8_t length)
```

This Function implements *bus_read* function, which is used by sensor API to receive data from BMM050 sensor. The communication is done via SPI so ASF SPI driver functions are used.

Table 17 - Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address (not used)
[in]	reg_addr	Address of source register
[out]	rx_data	Data buffer to be received
[in]	length	Length of the data to be received

8.3.5.4 Function bmm_delay_msec

Initiates a delay of the length of the argument in milliseconds

```
void bmm_delay_msec(uint32_t)
```

This function implements *delay_msec* function which is used by the sensor API to cause enough delay for the sensor so that it executes specific commands or provides required data.

Table 18 - Parameters

Data Direction	Parameter Name	Description
[in]	msec	Delay length in terms of milliseconds

9 ASF Driver Supports Implementation

9.1 Clock Support

This chapter describes the functions declared in “clock_support.h” file and defined in “clock_support.c” file.

Clock support uses ASF clock management modules and defines initialization and configuration functions for the microcontroller’s clock sources and generic clock generators that are needed for this application.

For more information about the clocking system refer to the corresponding application note or the microcontroller datasheet from Atmel.

9.1.1 Includes

9.1.1.1 Include “clock.h”

SAM D20 Clock Driver from Atmel

9.1.1.2 Include “glck,h”

SAM D20 Generic Clock Driver from Atmel

9.1.2 Type Definitions

N/A

9.1.3 Macro Definitions

N/A

9.1.4 Global Variables/ Structures

N/A

9.1.5 Function Definitions

9.1.5.1 Function clock_initialize

Initializes clock sources, generic clock generators and system main clock of the MCU

```
void clock_initialize(void)
```

This function calls configuration functions for DFLL48M and OSC8M clock sources, generic clock generators 1 and 2 and the main clock. After initialization, the clock sources’ and generic clock generators’ values are as follows:

- OSC8M: 2 MHz

DFLL:	Open Loop, 48 MHz
GCLK1:	500 KHz
GLCK2:	2 MHz

9.1.5.2 Function clock_configure_dfll

Configures the DFLL48M clock source of the MCU

```
void clock_configure_dfll(void)
```

This function configures DFLL48M clock source of the MCU with default configuration values and enables the clock source. (Disabled by default)

9.1.5.3 Function configure_osc8m

Configures the 8 MHz internal clock source of the MCU

```
void clock_configure_osc8m(void)
```

This function configures OSC8M clock source of the MCU with default configuration values changes the clock source's prescaler to 4. (Enabled by default)

9.1.5.4 Function clock_configure_system_clock

Configures system main clock source

```
void clock_configure_system_clock(void)
```

This function configures *generic clock generator 0* of the MCU, which is used as the main clock source, with default configuration values, changes its clock source to DFLL48M and changes its division factor to 2, hence providing a 24 MHz clock on GCLK_GENERATOR_0. (Enabled by default)

9.1.5.5 Function clock_configure_gclk_generator_1

Configures *generic clock generator 1*.

```
void clock_configure_gclk_generator_1(void)
```

This function configures *generic clock generator 1* of the MCU with default configuration values, changes its division factor to 4 and enables the clock generator, hence providing a 500 KHz clock on GCLK_GENERATOR_1. (Disabled by default)

9.1.5.6 Function clock_configure_gclk_generator_2

Configures *generic clock generator 2*

```
void clock_configure_gclk_generator_2(void)
```

This function configures *generic clock generator 2* of the MCU with default configuration values and enables the clock generator, hence providing a 2 MHz clock on GCLK_GENERATOR_2. (Disabled by default)

9.2 SPI Support

This chapter describes the functions declared in “spi_support.h” file and defined in “spi_support.c” file.

SPI support uses ASF SPI driver modules and defines initialization, configuration and callback functions for the microcontroller’s SPI peripheral that is needed to communicate with the three internal sensors.

For more information about the SPI modules refer to the corresponding application note or the microcontroller datasheet from Atmel.

9.2.1 Includes

9.2.1.1 Include “spi.h”

SAM D20 Serial Peripheral Interface Driver from Atmel

9.2.1.2 Include “spi_interrupt.h”

SAM D20 Serial Peripheral Interface callback mode Driver from Atmel

9.2.2 Type Definitions

N/A

9.2.3 Macro Definitions

9.2.3.1 Macro SPI_BAUDRATE_10M

```
#define SPI_BAUDRATE_10M      UINT32_C(10000000)
```

This is a value of 10 million that can be used to set SPI frequency to 10 MHz.

9.2.3.2 Macro SPI_BAUDRATE

```
#define SPI_BAUDRATE           SPI_BAUDRATE_10M
```

This macro defines the value loaded onto SPI baud rate register initially. The default value selected sets SPI frequency to 10 MHz.

9.2.4 Global Variables/ Structures

9.2.4.1 Structure spi_master_instance

```
struct spi_module spi_master_instance
```

Instantiates a SERCOM SPI driver software structure, used to retain software state information of the associated hardware module instance.

9.2.5 Function Definitions

9.2.5.1 Function spi_initialize

Initializes SPI module of the MCU

```
void spi_initialize(void)
```

This function calls configuration functions for SPI master and callbacks of the module.

9.2.5.2 Function spi_configure_master

Configures SPI master module of the MCU

```
void spi_configure_master(void)
```

This function configures the SPI master module with default configuration values, sets spi baud rate to 1 MHz, sets SPI master pads, initializes SERCOM3 with SPI master configurations and enables the module. (Disabled by default)

9.2.5.3 Function spi_configure_slave

Configures an SPI slave

```
void spi_configure_slave(
                struct spi_slave_inst *slave_inst_ptr,
                uint8_t const ss_pin)
```

This function configures the SPI slave referenced in the arguments with default configuration values and sets the its slave select pin to the pin number given in the arguments.

Table 19 - Parameters

Data Direction	Parameter Name	Description
[out]	slave_inst_ptr	Pointer to the SPI slave software instance struct
[in]	ss_pin	SPI slave-select pin number

9.3 TC Support

This chapter describes the functions declared in “tc_support.h” file and defined in “tc_support.c” file.

TC support uses ASF timer/counter driver modules and defines initialization, configuration and callback functions for the microcontroller’s timer/counter peripherals that that are needed for scheduling tasks or initiating delays. In addition to these some wrapper functions are defined that are used in the application.

For more information about the TC peripherals refer to the corresponding application note or the microcontroller datasheet from Atmel.

9.3.1 Includes

9.3.1.1 Include “tc.h”

SAM D20 Timer Counter driver from Atmel

9.3.1.2 Include “tc_interrupt.h”

SAM D20 Timer Counter callback driver from Atmel

9.3.2 Type Definitions

N/A

9.3.3 Macro Definitions

9.3.3.1 Macro COUNT_MAX_32_BIT

```
#define COUNT_MAX_32BIT      UINT32_C(0xFFFFFFFF)
```

Maximum value of a 32-bit counter

9.3.3.2 Macro TC6_PERIOD_1000MS

```
#define TC6_PERIOD_1000MS    COUNT_MAX_32BIT - UINT32_C(500000)
```

Loading this value onto the 32-bit count register of TC6, causes the counter to overflow after 1000 milliseconds. (Assuming it has a 500 kHz clock source.)

9.3.3.3 Macro TC6_PERIOD_100MS

```
#define TC6_PERIOD_100MS     COUNT_MAX_32BIT - UINT32_C(50000)
```

Loading this value onto the 32-bit count register of TC6, causes the counter to overflow after 100 milliseconds. (assuming it has a 500 kHz clock source)

9.3.3.4 Macro TC6_PERIOD_10MS

```
#define TC6_PERIOD_10MS       COUNT_MAX_32BIT - UINT32_C(5000)
```

Loading this value onto the 32-bit count register of TC6, causes the counter to overflow after 10 milliseconds. (Assuming it has a 500 kHz clock source)

9.3.3.5 Macro TC6_COUNT_VALUE

```
#define TC6_COUNT_VALUE       TC6_PERIOD_100MS
```

This macro defines the value loaded onto TC6 count register initially and after each overflow. The default value selected causes the counter to overflow on a period of 100 milliseconds.

9.3.4 Global Variables/ Structures

9.3.4.1 Structure tc4_instance

```
struct tc_module tc4_instance
```

Instantiates a TC software instance structure, used to retain software state information of the associated hardware module instance, which in this case is TC4.

9.3.4.2 Structure tc6_instance

```
struct tc_module tc6_instance
```

Instantiates a TC software instance structure, used to retain software state information of the associated hardware module instance, which in this case is TC6.

9.3.4.3 Boolean tc4_callback_flag

```
volatile bool tc4_callback_flag
```

This flag is *false* by default and is set by TC4 callback function; i.e. it is set whenever TC4 counter register value is equal to the value set in its capture channel 0. The flag can be used by other functions to execute desired tasks accordingly.

9.3.4.4 Boolean tc6_callback_flag

```
volatile bool tc6_callback_flag
```

This flag is *false* by default and is set by TC6 callback function; i.e. it is set whenever TC6 counter overflows. The flag can be used by other functions to execute desired tasks accordingly.

9.3.5 Function Definitions

9.3.5.1 Function tc_initialize

Initializes the TC4 and TC6 timer/counter modules of the MCU

```
void tc_initialize(void)
```

This function calls configuration functions and callback configuration functions of TC4 and TC6. After initialization, the TC6 and TC4 are configured and enabled.

9.3.5.2 Function tc4_configure

Configures TC4 of the MCU.

```
void configure_tc4(void)
```

This function configures TC4 with default configuration values, sets its clock source to GCLK_GENERATOR_1, sets the capture channel 0 value to 500 and enables the module. (Disabled by default)

9.3.5.3 Function tc4_configure_callbacks

Configures TC6 callback register

```
void configure_tc4_callbacks(void)
```

This function configures TC4 callback register with required callback types and their handler functions, resets the flags and enables the callbacks. (Disabled by default)

One callback type is enabled:

- Capture Counter Channel 0: interrupts when counter value is equal to channel 0 value

9.3.5.4 Function tc4_callback

Called when TC4 counter is equal to its capture channel 0 value

```
void tc4_callback(struct tc_module *const module_inst_ptr)
```

This Function is called whenever TC4 counter value is equal to the value in its capture channel 0 and sets the corresponding flag.

Table 20 - Parameters

Data Direction	Parameter Name	Description
[in]	module_inst_ptr	Pointer to the TC software instance struct that invokes the corresponding interrupt

9.3.5.5 Function tc4_wait_for_msec

Implements a delay of the length of the argument in milliseconds

```
void tc4_wait_for_msec(uint32_t msec)
```

This function sets the CC Channel 0 of TC4 according to *msec* value so that it takes *msec* milliseconds for the TC module to give an interrupt. After the interrupt is triggered it stops the counter and resets the corresponding interrupt flag.

Table 21 - Parameters

Data Direction	Parameter Name	Description
[in]	msec	Delay length in terms of milliseconds

9.3.5.6 Function tc6_configure

Configures TC6 of the MCU

```
void tc6_configure(void)
```

This function configures TC6 with default configuration values, sets the counter size to 32 bits, sets its clock source to GCLK_GENERATOR_1 and enables the module. (Disabled by default)

9.3.5.7 Function tc6_configure_callbacks

Configures TC6 callback register

```
void tc6_configure_callbacks(void)
```

This function configures TC6 callback register with required callback types and their handler functions, resets the flags and enables the callbacks. (Disabled by default)

One callback type is enabled:

- Counter Overflow: interrupts when the counter overflows.

9.3.5.8 Function tc6_callback

Called when TC6 counter is equal to its capture channel 0 value

```
void tc6_callback(struct tc_module *const module_inst_ptr)
```

This Function is called whenever TC6 counter overflows. It reloads the counter value and sets the corresponding flag.

Table 22 - Parameters

Data Direction	Parameter Name	Description
[in]	module_inst_ptr	Pointer to the TC software instance struct that invokes the corresponding interrupt

9.3.5.9 Function tc6_stop_counter

Stops TC6 counter

```
void tc6_stop_counter(void)
```

This function calls TC6 stop counter function.

9.3.5.10 Function tc6_start_counter

Starts TC6 counter

```
void tc6_start_counter(void)
```

This function calls TC6 start counter function.

9.4 USART Support

This chapter describes the functions declared in “usart_support.h” file and defined in “usart_support.c” file.

USART support uses ASF USART driver modules and defines initialization, configuration and callback functions for the microcontroller’s USART peripheral that is needed to communicate with to an external device (here a serial terminal).

For more information about the USART peripherals refer to the corresponding application note or the microcontroller datasheet from Atmel.

9.4.1 Includes

9.4.1.1 Include “usart.h”

SAM D20 SERCOM USART driver from Atmel

9.4.1.2 Include “usart_interrupt.h”

SAM D20 SERCOM USART Asynchronous driver from Atmel

9.4.2 Type Definitions

N/A

9.4.3 Macro Definitions

9.4.3.1 Macro

```
#define USART_BAUDRATE_115200       UINT32_C(115200)
```

This is a value of 115200 that can be used to set USART baud rate.

9.4.3.2 Macro

```
#define USART_BAUDRATE              USART_BAUDRATE_115200
```

This macro defines the value loaded onto USART module’s baud rate register initially. The default value selected sets the rate to 115200.

9.4.4 Global Variables/ Structures

9.4.4.1 Structure usart_instance

```
struct usart_module usart_instance
```

Instantiates a SERCOM USART driver software instance structure, used to retain software state information of the associated hardware module instance.

9.4.4.2 Boolean usart_callback_receive_flag

```
volatile bool usart_callback_receive_flag
```

This flag is *false* by default and is set by USART receive callback function; i.e. it is set after each USART reception. The flag can be used by other functions to execute desired tasks accordingly.

9.4.4.3 Boolean usart_callback_transmit_flag

```
volatile bool usart_callback_transmit_flag
```

This flag is *false* by default and is set by USART receive callback function; i.e. it is set after each USART reception. The flag can be used by other functions to execute desired tasks accordingly.

9.4.4.4 Integer usart_rx_byte

```
uint16_t usart_rx_byte
```

This variable holds USARTS received bytes to be saved to in the Rx buffer one by one. (Note: ASF functions need a 16-bit integer.)

9.4.5 Function Definitions

9.4.5.1 Function usart_initialize

Initializes the USART module of the MCU

```
void usart_initialize(void)
```

This function calls configuration function and the callback configuration function of the USART module.

9.4.5.2 Function usart_configure

Configures the USART module of the MCU

```
void usart_configure (void)
```

This function configures the USART module with default configuration values, sets its baud rate to 115200, sets its clock source to GCLK_GENERATOR_2, sets the pads, initializes SERCOM5 with the USART configuration and enables the module. (Disabled by default)

9.4.5.3 Function usart_configure_callbacks

Configures USART callback register

```
void usart_configure_callbacks(void)
```

This function configures USART callback register with required callback types and their handler functions, resets the corresponding flags and enables the callbacks. (Disabled by default)

Two callback types are enabled:

- Buffer Received: Interrupts after reception jobs.
- Buffer Transmitted: Interrupts after transmission jobs.

9.4.5.4 Function usart_callback_receive

Called after USART receptions

```
void usart_callback_receive(
                        struct usart_module *const usart_module_ptr)
```

This function is called after each reception (1 Byte) of the USART, sets the corresponding flag.

Table 23 - Parameters

Data Direction	Parameter Name	Description
[in]	usart_module_ptr	Pointer to the USART software instance struct that invokes the corresponding interrupt

9.4.5.5 Function usart_callback_transmit

Called after USART receptions

```
void usart_callback_transmit(
                  struct usart_module *const usart_module_ptr)
```

This function is called after each trasmission job of the USART is completed. It sets the corresponding flag.

Table 24 - Parameters

Data Direction	Parameter Name	Description
[in]	usart_module_ptr	Pointer to the USART software instance struct that invokes the corresponding interrupt

10 Examples

10.1 Example Commands

Configuration of the three sensors (Accelerometer BMA280, Gyroscope BMG160 and Magnetometer BMM050) can be changed via commands provided in tables Table 5, Table 6 and Table 7. Some examples for different possibilities are provided in this chapter. Note that bytes have to be sent one at a time.

10.1.1 Change Accelerometer's Configuration

00	Stop Stream
AA	Accelerometer Configuration Select
03	Acceleration Measurement Range: ±2 g
08	Data Filter Bandwidth: 8 Hz
00	Mode: Normal
01	Start Stream

10.1.2 Change Gyroscope's Configuration

00	Stop Stream
BB	Gyroscope Configuration Select
03	Angular Velocity Measurement Range: 125 °/s
05	Data Filter Bandwidth: 12 Hz
00	Mode: Normal
01	Start Stream

10.1.3 Change Magnetometer's Configuration

00	Stop Stream
CC	Magnetometer Configuration Select
01	Preset Mode: Low Power
00	Data Rate: 10 Hz (Ignored because of preset mode)
00	Mode: Normal
01	Start Stream

11 Resources

This reference example utilizes Atmel toolchain as possible. Microcontroller component drivers are the ones provided by Atmel Software Framework (ASF) and the support function is written similar to examples provided by Atmel application notes.

11.1 Bosch Sensortec Resources

BMA2x2 Accelerometer Sensor Driver

https://github.com/BoschSensortec/sensor_drivers/tree/master/BMA2x2

BMG160 Gyroscope Sensor Driver

https://github.com/BoschSensortec/sensor_drivers/tree/master/BMG160

BMM050 Magnetometer Sensor Driver

https://github.com/BoschSensortec/sensor_drivers/tree/master/BMM050

11.2 Atmel Resources

Atmel-42129-SAM-D20_Datasheet

http://www.atmel.com/Images/atmel-42129-sam-d20_datasheet.pdf

Atmel AT03255: SAM D20/D21 Serial Peripheral Interface Driver (SERCOM SPI) Application Note

http://www.atmel.com/Images/Atmel-42115-SAM-D20-D21-Serial-Peripheral-Interface-Driver-SERCOM-SPI_Application-Note_AT03255.pdf

Atmel 42118: SAM D20/D21 Serial-USART Driver (SERCOM USART) Application Note

http://www.atmel.com/Images/Atmel-42118-SAM-D20-D21-Serial-USART-Driver-SERCOM-USART_Application-Note_AT03256.pdf

Atmel 42119: SAM D20/D21 System Clock Management Driver (SYSTEM CLOCK) Application Note

http://www.atmel.com/Images/Atmel-42119-SAM-D20-D21-System-Clock-Management-Driver-SYSTEM-CLOCK_Application-Note_AT03259.pdf

Atmel 42123: SAM D20/D21 Timer/Counter Driver (TC) Application Note

http://www.atmel.com/Images/Atmel-42123-SAM-D20-D21-Timer-Counter-Driver-TC_Application-Note_AT03263.pdf

Atmel 42120: SAM D20/D21 System Driver (SYSTEM) Application Note

http://www.atmel.com/Images/Atmel-42120-SAM-D20-D21-System-Driver-SYSTEM_Application-Note_AT03260.pdf

12 Legal disclaimer

12.1 Engineering samples

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

12.2 Product Use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or security sensitive systems. Security sensitive systems are those for which a malfunction is expected to lead to bodily harm or significant property damage. In addition, they are not fit for use in products which interact with motor vehicle systems.

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

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

The purchaser must monitor the market for the purchased products, particularly with regard to product safety, and inform Bosch Sensortec without delay of all security relevant incidents.

12.3 Application Examples and Hints

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

13 Document History and Modifications

Rev. No.	Chapter	Description of Modification/ Changes	Date
1.0		Document Created	07.10.2015
1.1		Reference to old accelerometer is modified to show the actual accelerometer which is BMA280	28.10.2015

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

contact@bosch-sensortec.com
www.bosch-sensortec.com