Application Note

BMF055
Example Project – BSX Lite Integration

Bosch Sensortec

BOSCH
Invented for life

Application Note:	**BMF055 Example Project – BSX Lite Integration**
Document Revision	1.0
Document Release	October 2015
Document Number	BST-BMF055-EX003-00
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 and the BSX Lite library in order to demonstrate the sensor function on BST Shark Demo software running on a host computer.

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. The sensor produces raw sensor data that are processed using BSX Lite library functions.

This project is a reference example that shows how to use basic functions of BMF055 and BSX Lite 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 the Shark Demo.

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

1. Install the latest version of Atmel Studio from Atmel website
2. Open Atmel Studio
3. Go to “Tools -> Extension Manager” and install the latest version of Atmel Software Framework (Version used in this extension is 3.26.0
4. Go to “Tools -> Extension Manager” and search for “BMF055 Shuttle Board – BSX Lite Integration” extension from Bosch Sensortec GmbH (BST) and install it
5. Restart Atmel Studio
6. Go to “File -> New -> Example Projects”
7. “Below BST – Bosch Sensortec GmbH” find the project named “BMF055_SHUTTLE_BOARD_BSX_LITE_INTEGRATION – 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. Set up the application board and 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
15. In Atmel Studio go to “Project -> Properties” and select the tab named “Tool”

* It is assumed that the shuttle board would be interfaced to a terminal software or the BST Shark Demo running on a host computer.

16. Below “Selected debugger/programmer” select the “SAM-ICE” tool. And select “SWD” as the interface and save the changes. (Settings might differ if you choose a different tool.)

3.3 Run the Project

17. In Atmel Studio go 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)

3.4 Shark Demo Settings

20. Install “BST Shark Demo” software.

21. Go to the folder where the software is installed. (The default address is “C:\Bosch Sensortec\Shark”.)

22. Open the file “\Serial\ app_settings.txt”.

23. Change the field “serial_port_numbert” to the one that you have previously noted.

24. Save and close the file.

25. Run Shark Demo.

26. Select “COM” interface. Click “OK” to establish a connection between the microcontroller and the software.

    (Notice the messages below on the right side of the screen.)

27. After the connection is successfully made, press the play button to start interacting with the sensor.

3.5 Check the Use-case*

28. Move the sensor around to see the shark moving respectively.

* For detailed information refer to Application Overview.

4 Application Overview

For detailed description of application implementation see Application.

This example code reads raw data from the three internal sensors, processes them using BSX Lite sensor fusion library functions and transmits processed sensor data on the USART. On the other side of the USART connection is BST Shark demo 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.

This application 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. A serial connection port is also needed to connect the system to to the Shark Demo running on a computer.

Figure 1 shows the necessary connections to power-on and program the sensor to run this example. For more information refer to Hardware Platform.

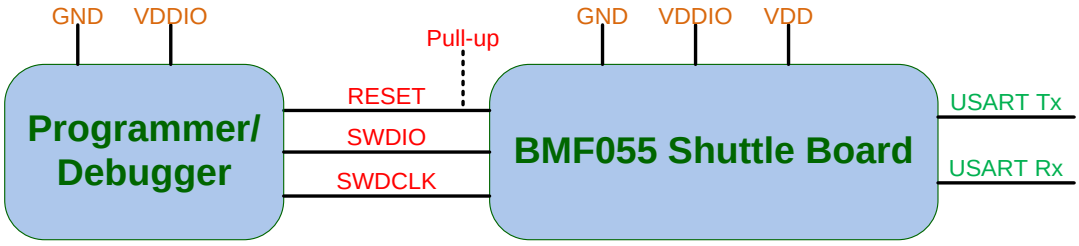

Figure 1 - Minimum Necessary Connections

BMF055 is a 9-axis orientation sensor, which integrates a microcontroller and three orientation sensors (accelerometer, magnetometer and gyroscope) in a single package. The MCU acts as the host. It communicates with the three sensors via an SPI bus and interfaces a serial terminal software host computer (or another MCU).

The MCU communicates with the sensors via the internal SPI bus. The bus has a frequency of 10 MHz. In addition to SPI signals there are two interrupt signals that connect the MCU to the accelerometer and the Gyroscope. Raw sensor data is read and sent to BSX Lite on the frequency required by the library (100 Hz). The data is then processed and sent out via the USART port. The USART communication has a baud rate of 115200 bps. Sensor movements can be seen as graphical movements of a shark in Shark Demo software connected to this port.

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 chapter 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 48 MHz – Open Loop)
GCLK 0	Generates the main system clock, using DFLL as its source (Frequency 48 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 10 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

After power-on and initialization, BMF055 internal sensors enter SUSPEND mode. However a configuration function is defined for each sensor that derives the sensor to NORMAL mode and configures it as given in Table 2, Table 3 and Table 4.

Table 2 - Default Configuration of BMA280

Parameter	Value
Accelerometer Range	±2 g
Bandwidth	50 Hz
Power Mode	Normal Mode

Table 3 - Default Configuration of BMG160

Parameter	Value
Gyroscope Range	500 °/s
Bandwidth	64 Hz
Data Rate	2000 Hz
Power Mode	Normal Mode

Table 4 - Default Configuration of BMM150

Parameter	Value
Data Rate	N/A*
Preset	Regular
Power Mode	Forced Mode

* Not applicable in forced mode

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 2 shows the basic building blocks of the BMF055 device.

For more information about the device refer to BMF055 Datasheet.

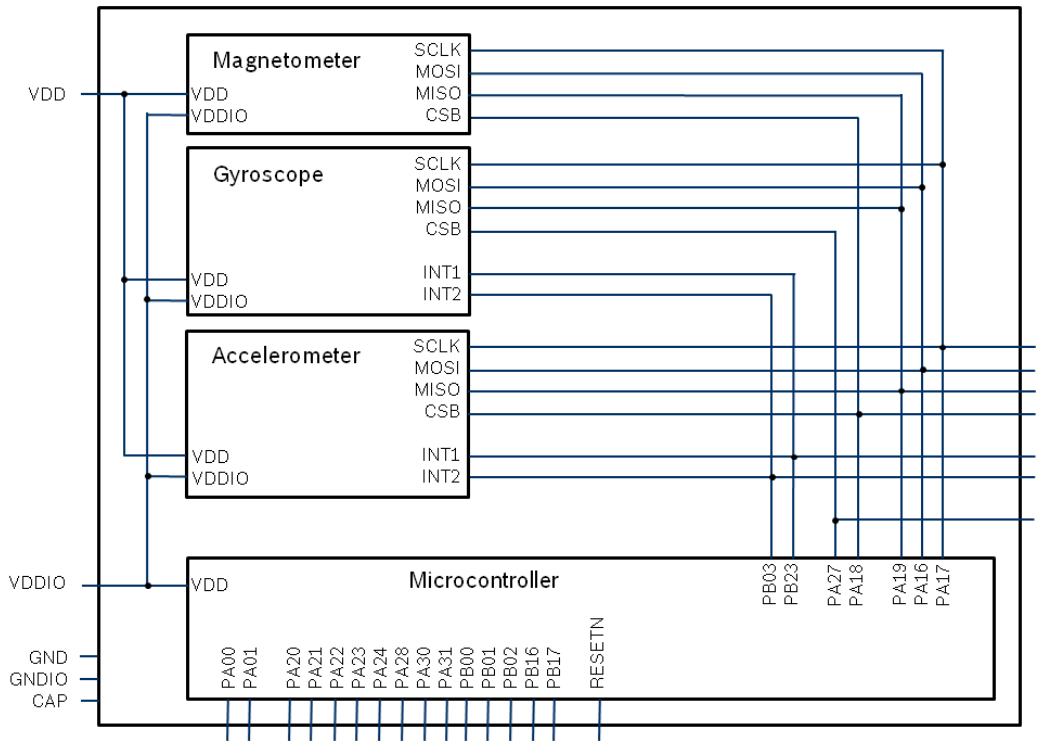

Figure 2 - BMF055 Architecture

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 5 - 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 6. 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 6 - USART Pin Assignment

Shuttle Board Pin #	Description
17	Tx
18	Rx

6 Design Structure

Figure 3 shows structure of the design, hardware layer and software layers above it.

BSX Lite fusion library is a complete 9-axis fusion solution which combines the measurements from 3-axis gyroscope, 3-axis geomagnetic sensor and a 3-axis accelerometer, to provide a robust absolute orientation vector. For more information and detailed integration guideline refer to BSX Lite documents.

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 References. 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: BMA280, 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 Support Implementation.

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

For more information about application refer to Application Overview and Application Implementation.

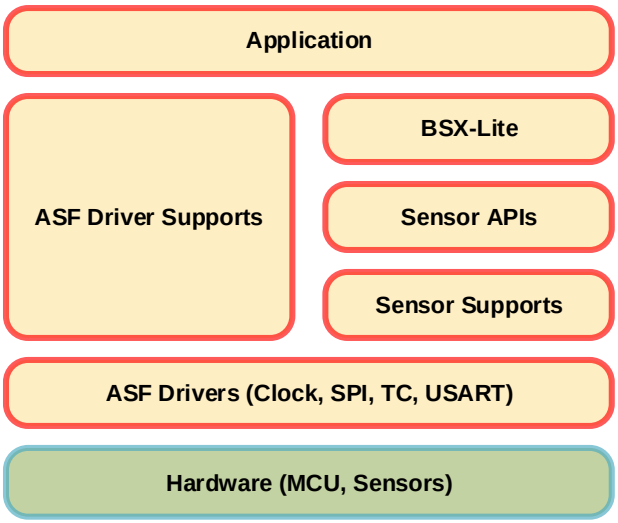

Figure 3 - Design Structure

6.1 Folder Structure

Figure 4 shows the project folder structure.

- “libBSX_Library.a” is the static BSX Lite library which will be linked to the project while compiling
- “BSX_Lite” folder contains the library’s interface headers
- “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.

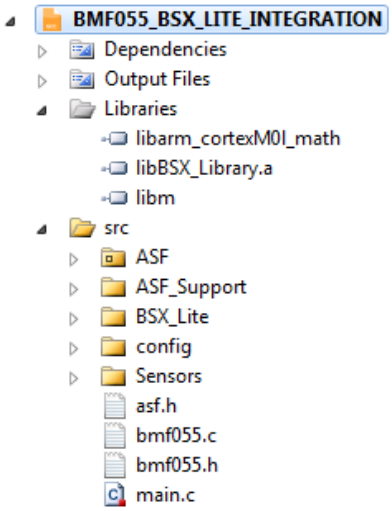

Figure 4 - Project Folder Structure

7 Application Implementation

7.1 Main

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

All the initializations for the microcontroller and the sensors are done here

7.1.1 Includes

7.1.1.1 Include “bmf055.h”

Application functions, global variables, macro definitions and libraries needed

7.1.2 Function Definitions

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, the sensors and the fusion library. It initializes the microcontroller unit and all its required modules such as clock sources, SPI, TC, USART and interrupts.

In the infinite loop it repeatedly checks two flags:

- The USART receive flag which is set in case of receiving a complete message string. It checks the message and takes appropriate action. The messages handled here are required by the Shark Demo protocol.
- The calibration execution flag which is set when the calibration function of the BSX Lite library has to be called.

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 “string.h”

Definitions for memory and string functions

7.2.1.3 Include “clock_support.h”

Wrapper functions for ASF clock and generic clock modules

7.2.1.4 Include “spi_support.h”

Wrapper functions for ASF serial peripheral interface module

7.2.1.5 Include “tc_support.h”

Wrapper functions for ASF Timer/Counter module

7.2.1.6 Include “usart_suppot.h”

Wrapper functions for ASF USART module

7.2.1.7 Include “bma2x2_support.h”

Wrapper functions for BMA2x2 connection to the MCU

7.2.1.8 Include “bmg160_support.h”

Wrapper functions for BMG160 connection to the MCU

7.2.1.9 Include “bmm050_support.h”

Wrapper functions for BMM050 connection to the MCU

7.2.1.10 Include “BsxLiteFusionLibrary.h”

This file provides an interface to use the functionality of nine degree of freedom software library

7.2.1.11 Include “BsxLibraryCalibConstants.h”

This file provides constants definition for calibration modules

7.2.1.12 Include “BsxLibraryCalibConstants.h”

This file provides constants definition

7.2.1.13 Include “BsxLibraryDataTypes.h”

This file provides data types used by library

7.2.1.14 Include “BsxLibraryErrorConstants”

This file provides error constants definition

7.2.2 Type Definitions

N/A

7.2.3 Macro Definitions

7.2.3.1 Macro low_byte(x)

```
#define low_byte(x)          ((x) & 0xFF)
```

This macro function extracts the low byte of a longer integer.

7.2.3.2 Macro high_byte(x)

```
#define high_byte(x)         (((x)>>8) & 0xFF)
```

This macro function extracts the high byte of a 16-bit integer.

7.2.3.3 Macro SHARK_DEMO_PACKET_LENGTH

```
#define SHARK_DEMO_PACKET_LENGTH         63
```

USART packet length defined by the Shark Demo protocol

7.2.3.4 Macro SHARK_DEMO_PACKET_SYNCH_BYTE_0

```
#define SHARK_DEMO_PACKET_SYNCH_BYTE_0   0x04
```

Synchronization byte 0 value defined by the Shark Demo protocol

7.2.3.5 Macro SHARK_DEMO_PACKET_SYNCH_BYTE_1

```
#define SHARK_DEMO_PACKET_SYNCH_BYTE_1   0x11
```

Synchronization byte 1 value defined by the Shark Demo protocol

7.2.3.6 Macro QUAT_SCALING_FACTOR

```
#define QUAT_SCALING_FACTOR                16393
```

Scaling factor for converting quaternion data floating point values to integer defined by BSX Lite

7.2.3.7 Macro COUNTER_10_MS

```
#define COUNTER_10_MS                      10
```

Local time is incremented by this value at each call of the scheduled task (100 Hz)

7.2.3.8 Macro PROCESS_ACC_GYRO_DATA

```
#define PROCESS_ACC_GYRO_DATA              10
```

Factor to determine the period for processing accelerometer and gyroscope data is milliseconds (frequency = 100 Hz)

7.2.3.9 Macro PROCESS_MAG_DATA

```
#define PROCESS_MAG_DATA                   40
```

Factor to determine the period for processing magnetometer data is milliseconds (frequency = 25 Hz)

7.2.4 Global Variables/ Structures

7.2.4.1 Integer local_timestamp

```
uint32_t local_timestamp
```

This variable holds the local time stamp required for the fusion library functions. It is incremented by the value defined by Macro COUNTER_10_MS.

7.2.4.2 Structure s_workingmode

```
ts_workingModes s_workingmode
```

This type is defined by the BSX Lite library and the structure holds the working mode for the library.

7.2.4.3 Structure timestamp_dataxyz

```
libraryinput_t timestamp_dataxyz
```

This type is defined by the BSX Lite library and the structure holds the sensor raw data to be used by library functions to generate sensor fusion data.

7.2.4.4 Boolean execute_calibration_flag

```
bool execute_calibration_flag
```

The calibration execution flag is set when the calibration function of the BSX Lite library has to be called.

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_remapped_mag_s16_data_t bmm050_mag_data
```

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

7.2.5.4 Structure bmf055_quarternion_data_f

```
static ts_dataquatf32  bmf055_quarternion_data_f
```

This structure holds BMF055 quaternion data which consists of four quaternion components (w,x,z,y) in floating point format.

7.2.5.5 Integer bmf055_acc_calib_stat

```
static BSX_U8 bmf055_acc_calib_stat
```

This variable holds calibration status of the accelerometer inside the BMF055, which is a number in the range of 0 to 3, with 3 indicating fully calibrated and 0 indicating not calibrated.

7.2.5.6 Integer bmf055_gyr_calib_stat

```
static BSX_U8 bmf055_gyr_calib_stat
```

This variable holds calibration status of the gyroscope inside the BMF055, which is a number in the range of 0 to 3, with 3 indicating fully calibrated and 0 indicating not calibrated.

7.2.5.7 Integer bmf055_mag_calib_stat

```
static BSX_U8 bmf055_mag_calib_stat
```

This variable holds calibration status of the magnetometer inside the BMF055, which is a number in the range of 0 to 3, with 3 indicating fully calibrated and 0 indicating not calibrated.

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. The sensors are then configured as required for the BSX Lite library

7.2.6.2 Function bsx_lite_initialize

Configures the BSX Lite library parameters

```
BSX_S8 bsx_lite_initialize (void)
```

This function initializes BSX Lite library by configuring the initialization parameters and setting its working mode.

7.2.6.3 Function bsx_lite_raw_feed

Provides raw sensor data for BSX Lite and calls the process functions

```
void bsx_lite_raw_feed(void)
```

7.2.6.4 Function bmf055_data_print

Reads output data of the fusion library and sends them over USART

```
void bmf055_data_print (void)
```

This function reads sensor fusion data (Quaternion data) using BSX Lite functions. It forms the packet to be sent and sends sensor data in the format defined by the Shark Demo communication protocol via USART.

7.2.6.5 Function tc_callback_handler

Schedules the sensor read and data print tasks

```
void bmf055_timer_handler (void)
```

This function is called as the callback function for TC6, which means it is called every 10 millisecond (frequency of 100 Hz). It calls functions to read raw sensor data, passing the data to the fusion library, calling library process functions and sending output packets to Shark Demo via USART.

7.2.6.6 Function conv_scale_float_int16

Converts a float value to a 16-bit integer regarding the scaling factor

```
int16_t conv_scale_float_int16(float input_float, float scaling)
```

This function gets a floating point value and a scaling factor and returns the 16-bit integer as the converted value. It takes care of the boundary values.

Data Direction	Parameter Name	Description
[in]	input_float	Input floating point value to be converted
[in]	scaling	Scaling factor for the conversion

8 Sensor API Support 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 and sets its power mode to SUSPEND.

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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

Table 7 - 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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

Table 8 - 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 9 - 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 and sets its power mode to SUSPEND.

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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

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

8.2.5.3 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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

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

8.2.5.4 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 12 - 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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

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

8.3.5.3 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.

SPI is used in *polled* mode and transaction request is repeated in a while loop if not successful. In order to avoid infinite loops a loop counter variable is defined to break if a certain limit is passed (Default limit is 100 loops).

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

8.3.5.4 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 15 - 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 of the system (*generic clock generator 0*). After initialization, the clock sources’ and generic clock generators’ frequencies are as follows:

- OSC8M: 2 MHz
- DFLL: Open Loop, 48 MHz
- GCLK0: 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 1, hence providing a 48 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. The module is used in this application in polled mode but callback configuration functions are provided and can be used in case of a later need to change.

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 16 - 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_16_BIT

```
#define COUNT_MAX_16BIT      UINT16_C(0xFFFF)
```

Maximum value of a 16-bit counter

9.3.3.2 Macro COUNT_MAX_32_BIT

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

Maximum value of a 32-bit counter

9.3.3.3 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.)

9.3.3.4 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)

9.3.3.5 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)

9.3.3.6 Macro TC6_COUNT_VALUE

```
#define TC6_COUNT_VALUE      TC6_PERIOD_10MS
```

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 10 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 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.

9.3.4.3 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.

9.3.4.4 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 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. It also assigns a reference to the interrupt handler function, so that after an interrupt the desired function, defined in the application side, is called.

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_3, sets the capture channel 0 value to 500 and enables the module. (Disabled by default)

The clock source of this TC has a frequency of 500 KHz, so counting up to 500 takes 1 millisecond.

The counter is stopped after initialization. Whenever a delay is needed in the application, counter start would be triggered.

9.3.5.3 Function tc4_configure_callbacks

Configures TC4 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 17 - 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 18 - 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 19 - 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_callback_function_ptr

Pointer to the interrupt handler function defined in the application part of the design

```
void (*tc6_callback_function_ptr) (void)
```

This pointer is assigned in the initialization function to point to a desired interrupt handler which is defined application part of the design.

After an overflow in TC6 the desired function is called via this pointer.

9.3.5.10 Function tc6_stop_counter

Stops TC6 counter

```
void tc6_stop_counter(void)
```

This function calls TC6 stop counter function.

9.3.5.11 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 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 array usart_rx_string[64]

```
uint8_t usart_rx_string[64]
```

This is the USART Rx buffer. Received bytes are saved in the buffer until a complete command is received. Then the buffer is read and processed in the application.

9.4.4.5 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.)

9.4.4.6 Integer usart_rx_count

```
uint16_t usart_rx_count
```

This variable holds the index of the USART Rx buffer. After a complete command is processed the count is reset to zero.

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 callback function of the USART module. The module is used in this application in callback mode.

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 module of the microcontroller 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 job of the USART is completed.

In this project we receive string commands on USART. The last character of all commands is newline ('\n'). So USART buffer is filled byte by byte until the end character indicates that a complete command is received in which case the corresponding flag is set.

Table 20 - 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 21 - Parameters

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

10 Appendix

10.1 Shark Demo Communication Protocol

In order to establish a connection with BST Shark Demo and communicate with it, there are commands that have to be sent in a certain order. Table 22 shows step-by-step communication process between host hardware and the Shark Demo GUI. The host hardware in the current project is Atmel SAM D21 Xplained pro board and the commands are sent and received by the host microcontroller on the board.

Table 23 shows the packet structure for data to be sent to Shark Demo. The length of the packet is 63 bytes and only the bytes which are specified in the table are important for this application. (Other bytes' values do not matter.)

Table 22 - Steb-by-step Communication process of Shark Demo GUI

Step #	Host Hardware	Client (Shark Demo GUI)
1		Dialog box will be opened with USB/COM option. Select COM option and press OK
2		1. Once OK button is pressed, UI will start checking with available COM ports by sending the command**:** **"board_get_sid()\r\n"** 2. Rightmost bottom text shows port check status.
3	Receives the string **"board_get_sid()\r\n"**	
4	Sends the string **"board_sid(A0) \r\n"**	
5		1. Validates the response for **board_get_sid()** by checking the response contains a substring as **BOARD/board**. 2. No check has been performed for valid sensor. 3. Rightmost bottom text shows port connection status.

7		Waits for Play button to be pressed
8		When Play button is pressed, sends the command: **“bsx_get_data(-1,4)\r\n”**
9	Receives the string: **“bsx_get_data(-1,4)\r\n”**	
10	Starts Streaming Quaternions, Calibration Status and Checksum as per protocol described	
11		Continuously receives quaternion, calibration status and updates UI.
12		Waits for Pause button press
13		When Pause button is pressed, sends the command: **“bsx_get_data(0,4)\r\n”**
14	Receives the string: **“bsx_get_data(0,4)\r\n”**	
15	Stops Streaming	

Table 23 - Data Packet

Byte #	Data	Value
0	Synch Tag	0x04
1		0x11
…		
38	Quaternion Values	w MSB
39		w LSB
40		x MSB
41		x LSB
42		y MSB
43		y LSB
44		z MSB
45		z LSB
…		
52	Check Sum	CRC MSB
53		CRC LSB
…		
58	Calibration Status	Accelerometer Calib
59		Magnetometer Calib
60		Gyroscope Calib
…		
62		

11 References

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 References

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 References

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	17.11.2015

Bosch Sensortec GmbH
Gerhard-Kindler-Strasse 8
72770 Reutlingen / Germany

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