Application Note

BMF055
Example Project – Interrupts

Bosch Sensortec

BOSCH
Invented for life

Application Note:	**BMF055 Example Project – Interrupts**
Document Revision	1.1
Document Release	October 2015
Document Number	BST-BMF055-EX002-00
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 interrupts of accelerometer and gyroscope sensors of BMF055 work.

This example program activates predetermined interrupt engines in BMA280 and BMG160 and causes the MCU and the sensors to switch between active and sleep modes accordingly. 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 example is a demonstration for use cases in which power consumption is critical.

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

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 – Interrupts" extension from Bosch Sensortec GmbH (BST) and install it
5. 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)
6. Restart Atmel Studio
7. Go to "File -> New -> Example Projects"
8. "Below BST – Bosch Sensortec GmbH" find the project named "BMF055_SHUTTLE_BOARD_INTERRUPTS – atsamd20j18a"
9. Select it and press "OK" button
10. Read and accept the license agreement and press "Finish" button to create a new example project

3.2 Hardware

11. Establish the minimum necessary connections as shown in Figure 1; including power, reset and programmer/debugger.
12. Establish a USART connection between the shuttle board and a host computer*. Use bridges if necessary.
13. Install required drivers for your virtual COM port.
14. Go to "Start Menu -> Control Panel -> Device Manager"
15. 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.

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

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

3.3 Run the Project

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

    The build process should succeed with no errors or warnings.

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

20. Wait for the process to be done.

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

21. Go to “View -> Terminal Window”

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

23. Press “Connect”

3.4 Check the Use-case*

24. Initially the sensors are suspended. In order to activate them send “CMD_ON” via USART.

25. Now the sensors are in NORMAL mode and MCU is in SLEEP mode to save power. A double tap wakens the MCU.

26. Now the system is in full power mode and reacts to motion by printing messages on USART. Move the board around and check the data on the terminal window.

* For detailed information refer to Application Overview.

4 Application Overview

For detailed description of application implementation see Application.

This example program activates predetermined interrupt engines in BMA280 and BMG160 and causes the MCU and the sensors to switch between active and sleep modes accordingly. 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 a terminal software running on a computer or to another microcontroller.

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

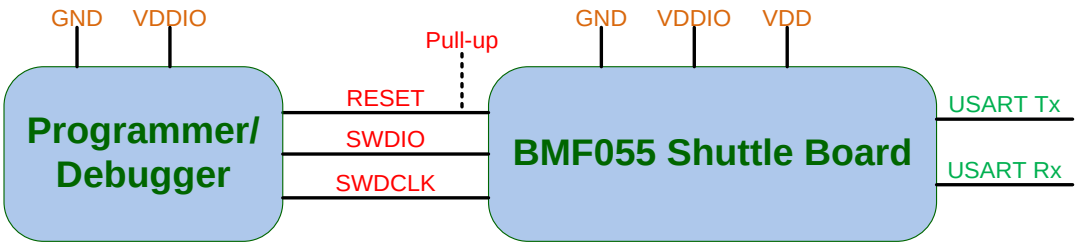

Figure 1 - Minimum Necessary Connections

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. The USART communication has a baud rate of 115200 bps. Atmel Studio terminal extension can be used to interface this communication. This serial interface is used to send commands to the MCU and print messages it sends.

After power-up and initialization, sensors enter SUSPEND mode and stay in that mode until ON command is sent to MCU via USART. If the command is received, the accelerometer and gyroscope enter NORMAL mode and their interrupts are configured as given in Table 5. In this state the MCU enters a sleep mode and the

system waits for a double-tap. After such an event it enters an active state and reacts to motion in the way that it sends a message on USART whenever the sensor starts/stops moving.

The state transition diagram of this design is depicted on Figure 2.

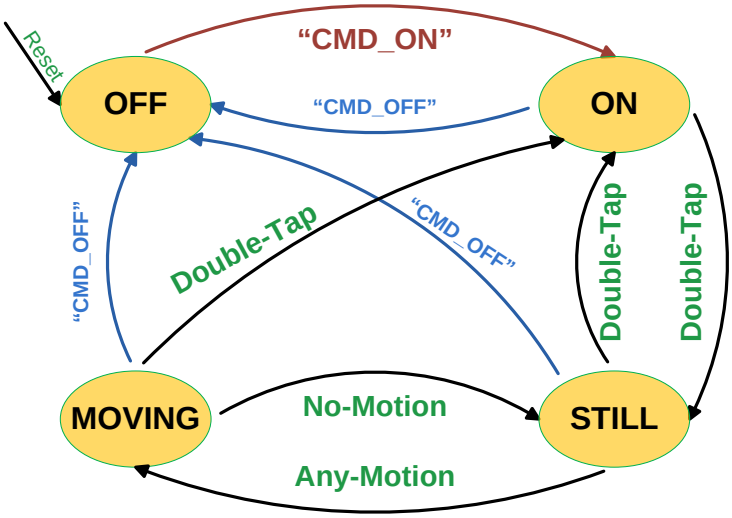

Figure 2 – BMF055 Function State

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 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)
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)
EXTINT Channel 7	External interrupt connected to BMA280 and BMG160 Interrupt pin INT1 (ACC double-tap, Acc Any-Motion and Gyr Any-Motion)
EXTINT Channel 3	External interrupt connected to BMA280 and BMG160 Interrupt pin INT2 (Acc No-Motion)

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	2000 Hz
Power Mode	Normal Mode

Table 3 - Default Configuration of BMG160

Parameter	Value
Gyroscope Range	2000 °/s
Bandwidth	523 Hz
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

Four interrupts are activated and routed to the interrupt pins by setting interrupt mask registers in the two sensors as illustrated on Figure 3. The activated interrupts are: *Slow/No Motion*, *Any Motion* and *Double-tap* of accelerometer and *Any Motion* of Gyroscope and their parameters are configured as given in Table 5.

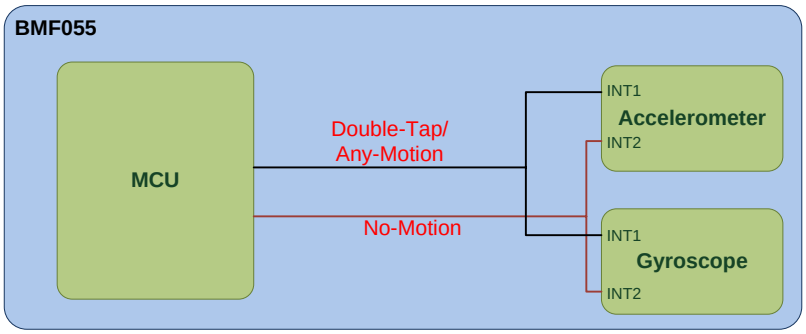

Figure 3 – Interrupt Signals Connection

Table 5 - Masked Interrupts' Configurations

Interrupt	Threshold	Duration	Axes
Accelerometer Tap	1250 mg	250 ms	-
Accelerometer Slow/ No Motion	156.2 mg	2 s	X, Y, Z
Accelerometer Any Motion	156.2 mg	2 samples	X, Y, Z
Gyroscope Any Motion	100 °/s	16 samples	X, Y, Z

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 4 shows the basic building blocks of the BMF055 device.

For more information about the device refer to BMF055 Datasheet.

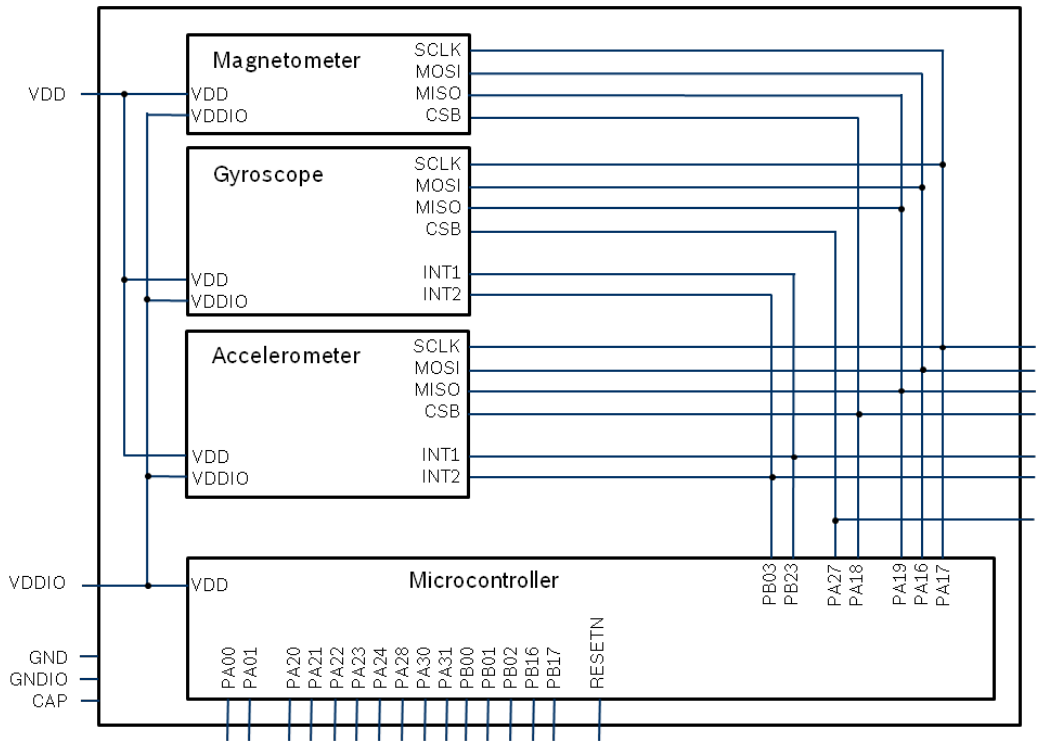

Figure 4 - BMF055 Architecture

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.

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

Table 6 - 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

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 7. 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 7 - UART Pin Assignment

Shuttle Board Pin #	Description
17	Tx
18	Rx

6 Design Structure

Figure 5 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 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, EXTINT, 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.

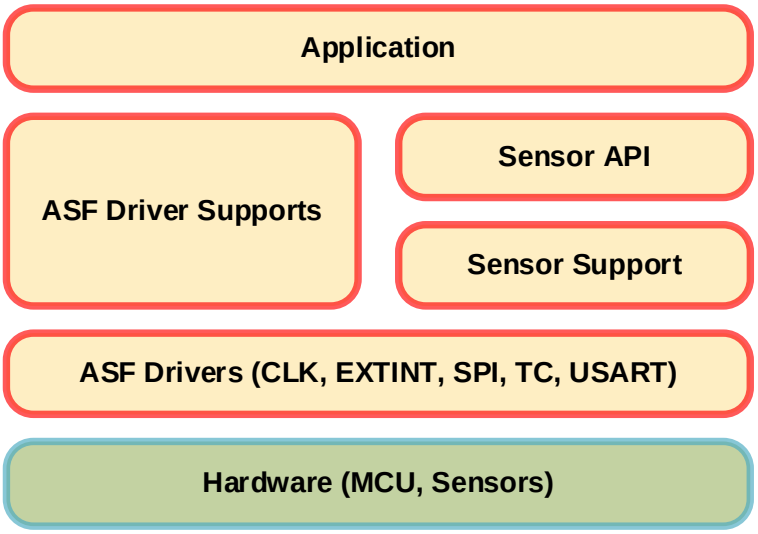

Figure 5 - Design Structure

6.1 Folder Structure

Figure 6 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.

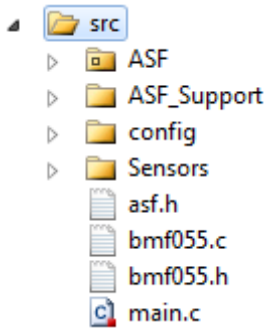

Figure 6 - Project Folder Structure

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 and the sensors. It initializes the microcontroller unit and all its required modules such as clock sources, SPI, TC, USART and interrupts. It also initializes function state variable and prints the state message on USART.

In the infinite loop it repeatedly checks the USART receive flag. In case of receiving a valid command it calls the handler function giving appropriate event as the parameter and goes back to checking USART. Corresponding event handler takes appropriate action regarding the system state and outputs.

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 “tc_support.h”

Wrapper functions for ASF Timer/Counter module

7.2.1.5 Include “usart_suppot.h”

Wrapper functions for ASF USART module

7.2.1.6 Include “extint_support.h”

Wrapper functions for ASF external interrupt channels 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

7.2.2 Type Definitions

7.2.2.1 Typedef enum bmf055_function_state_type

```
typedef enum bmf055_function_state_type
{
      BMF055_FUNCTION_STATE_OFF,
      BMF055_FUNCTION_STATE_ON,
```

```
      BMF055_FUNCTION_STATE_ACTIVE_STILL,
      BMF055_FUNCTION_STATE_ACTIVE_MOVING,
      BMF055_FUNCTION_STATE_LAST=BMF055_FUNCTION_STATE_ACTIVE_MOVING
}bmf055_function_state_t
```

This type's values are valid function states of the system. The state diagram is shown in Figure 2.

7.2.3 Macro Definitions

7.2.3.1 Macro BMF055_EVENT_CMD_ON

```
#define BMF055_EVENT_CMD_ON             1
```

Defines the CMD_ON command received via USART as event number 1. This event happens whenever the CMD_ON command is received on USART.

7.2.3.2 Macro BMF055_EVENT_CMD_OFF

```
#define BMF055_EVENT_CMD_OFF            2
```

Defines the CMD_OFF command received via USART as event number 2. This event happens whenever the CMD_OFF command is received on USART.

7.2.3.3 Macro BMF055_EVENT_TAP_ANY_MOTION

```
#define BMF055_EVENT_TAP_ANY_MOTION     3
```

Defines any-motion interrupt as event number 3. This event is triggered whenever an interrupt is sensed on INT1 pin of the MCU. The masked interrupts on INT1 are double-tap of BMA280 and any-motion of BMA280 and BMG160.

7.2.3.4 Macro BMF055_EVENT_NO_MOTION

```
#define BMF055_EVENT_NO_MOTION          4
```

Defines no-motion interrupt as event number 4. This event is triggered whenever an interrupt is sensed on INT2 pin of the MCU. The masked interrupt on INT2 is no-motion off BMA280.

7.2.4 Global Variables/ Structures

7.2.4.1 Bmf055_function_state_type bmf055_function_state

```
bmf055_function_state_t bmf055_function_state
```

This variable holds function state of the system.

7.2.5 Function Definitions

7.2.5.1 Function bma_2x2_configure

Wakes up and configures the sensor

```
void bma2x2_configure(void)
```

This function initiates a soft reset in BMA280 sensor that causes sensor to enter NORMAL mode. It then configures the interrupts.

Three interrupts are configured and activated as follows (For more information, refer to BMF055 Configuration):

- Any-motion (Slope) Interrupt
- Double-tap interrupt
- No-motion interrupt

7.2.5.2 Function bmg_160_configure

Wakes up and configures the sensor

```
void bmg160_configure(void)
```

This function initiates a soft reset in BMG160 sensor that causes sensor to enter NORMAL mode. It then configures the interrupts.

One interrupt is configured and activated as follows (For more information, refer to

BMF055 Configuration):

- Any-motion Interrupt

7.2.5.3 Function bmm_150_configure

Wakes up and configures the sensor

```
void bmm150_configure(void)
```

[not used in this example]

7.2.5.4 Function state_event_handler

Called after a state event is triggered

```
void state_event_handler(uint8_t this_resource)
```

This function is called after a state event is triggered i.e. after a valid command is received on USART or an activated sensor interrupt is sensed. It sets the next state and calls the corresponding state transition handler function.

Table 8 – Function Parameters

Data Direction	Parameter Name	Description
[in]	this_resource	Holds the number of the resource that invoked the interrupt which called this function

7.2.5.5 Function state_trans_off_to_on

State transition handler function

```
void state_trans_off_to_on(void)
```

This function is called when a transition from the state OFF to ON occurs. It reconfigures the sensors and prints the proper message on USART.

7.2.5.6 Function state_trans_on_to_still

State transition handler function

```
void state_trans_on_to_still(void)
```

This function is called when a transition from the state ON to STILL occurs. It prints the proper message on USART.

7.2.5.7 Function state_trans_still_to_on

State transition handler function

```
void state_trans_still_to_on(void)
```

This function is called when a transition from the state STILL to ON occurs. It prints the proper message on USART.

7.2.5.8 Function state_trans_still_to_moving

State transition handler function

```
void state_trans_stil_to_moving(void)
```

This function is called when a transition from the state STILL to MOVING occurs. It prints the proper message on USART.

7.2.5.9 Function state_trans_moving_to_still

State transition handler function

```
void state_trans_moving_to_still(void)
```

This function is called when a transition from the state MOVING to STILL occurs. It prints the proper message on USART.

7.2.5.10 Function state_trans_moving_to_on

State transition handler function

```
void state_trans_moving_to_on(void)
```

This function is called when a transition from the state MOVING to ON occurs. It prints the proper message on USART.

7.2.5.11 Function state_trans_to_off

State transition handler function

```
void state_trans_to_off(void)
```

This function is called when a transition to the state OFF occurs (from any of the other states). It changes the sensors' modes to SUSPEND and prints the proper message on USART.

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 9 - 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 10 - 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 11 - 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 12 - 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 13 - 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 14 - 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 15 - 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 16 - 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 17 - 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 “gIck,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: 24 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 External Interrupt Support

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

External interrupt support uses ASF EXTINT driver modules and defines initialization and configuration functions for the microcontroller’s external interrupt channels that are connected to the interrupt pins of BMA280 and BMG160 sensors.

For more information about the EIC (External Interrupt Controller) peripheral refer to the corresponding application note or the microcontroller datasheet from Atmel.

9.2.1 Includes

9.2.1.1 Include “extint.h”

SAM D20 external interrupt driver from Atmel

9.2.1.2 Include “extint_callback.h”

SAM D20 external interrupt callback diver from Atmel

9.2.2 Type Definitions

N/A

9.2.3 Macro Definitions

N/A

9.2.4 Function Definitions

9.2.4.1 Function extint_initialize

Initializes the external interrupt channel of the MCU

```
void extint_initialize(
      void (*interrupt_handler_function)(uint_8))
```

This function calls configuration functions of the two external interrupt channels that are required for this application. 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.

Table 18 - Function Parameters

Data Direction	Parameter Name	Description
[in]	interrupt_handler_function	Pointer to the interrupt handler function

9.2.4.2 Function extint_configure_PB23

Configures the external interrupt channel of the MCU

```
void extint_configure_PB23 (void)
```

This function configures the external interrupt channel number 7 available on PB23 pin of the MCU with default configuration values, sets the corresponding *pin* and *pinmux* assignments, set pull up/down to *DOWN*, sets its detection criteria to *rising-edge and* initializes the corresponding channel with the EXTINT configuration.

9.2.4.3 Function extint_configure_PB03

Configures the external interrupt channel of the MCU

```
void extint_configure_PB03 (void)
```

This function configures the external interrupt channel number 3 available on PB03 pin of the MCU with default configuration values, sets the corresponding *pin* and *pinmux* assignments, set pull up/down to *DOWN*, sets its detection criteria to *rising-edge and* initializes the corresponding channel with the EXTINT configuration.

9.2.4.4 Function interrupt_handler_function_ptr

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

```
void (*interrupt_handler_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 external interrupt from any of the pins configured here the desired function is called via this pointer.

9.2.4.5 Function extint_configure_callbacks

Configures USART callback register

```
void extint_configure_callbacks(void)
```

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

Two callbacks are enabled:

- Interrupt detection on pin PB23
- Interrupt detection on pin PB03

9.2.4.6 Function extint_detection_callback_PB23

Called after interrupt detection on PB23

```
void extint_detection_callback_PB23 (void)
```

This function is called interrupt detection on PB23. It calls the handler function on the application side.

9.2.4.7 Function extint_detection_callback_PB03

Called after interrupt detection on PB03

```
void extint_detection_callback_PB03 (void)
```

This function is called interrupt detection on PB03. It calls the handler function on the application side.

9.3 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.

9.3.1 Includes

9.3.1.1 Include “spi.h”

SAM D20 Serial Peripheral Interface Driver from Atmel

9.3.1.2 Include “spi_interrupt.h”

SAM D20 Serial Peripheral Interface callback mode Driver from Atmel

9.3.2 Type Definitions

N/A

9.3.3 Macro Definitions

9.3.3.1 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.

9.3.3.2 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.

9.3.4 Global Variables/ Structures

9.3.4.1 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.

9.3.5 Function Definitions

9.3.5.1 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.

9.3.5.2 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)

9.3.5.3 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

9.4 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.

9.4.1 Includes

9.4.1.1 Include “tc.h”

SAM D20 Timer Counter driver from Atmel

9.4.1.2 Include “tc_interrupt.h”

SAM D20 Timer Counter callback driver from Atmel

9.4.2 Type Definitions

N/A

9.4.3 Macro Definitions

9.4.3.1 Macro COUNT_MAX_16_BIT

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

Maximum value of a 16-bit counter

9.4.4 Global Variables/ Structures

9.4.4.1 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.

9.4.4.2 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.

9.4.5 Function Definitions

9.4.5.1 Function tc_initialize

Initializes TC4 timer/counter modules of the MCU

```
void tc_initialize(void)
```

This function calls configuration function and callback configuration function of TC4.

9.4.5.2 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.

9.4.5.3 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

9.4.5.4 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

9.4.5.5 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

9.5 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.

9.5.1 Includes

9.5.1.1 Include “usart.h”

SAM D20 SERCOM USART driver from Atmel

9.5.1.2 Include “usart_interrupt.h”

SAM D20 SERCOM USART Asynchronous driver from Atmel

9.5.2 Type Definitions

N/A

9.5.3 Macro Definitions

9.5.3.1 Macro

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

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

9.5.3.2 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.

9.5.4 Global Variables/ Structures

9.5.4.1 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.

9.5.4.2 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.

9.5.4.3 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.

9.5.4.4 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.

9.5.4.5 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.)

9.5.4.6 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.

9.5.5 Function Definitions

9.5.5.1 Function usart_initialize

Initializes the USART module of the MCU

```
void usart_initialize(void)
```

This function calls configuration function of the USART module. The module is used in this application in polled mode but callback configuration function is provided and can be used in case of a later need to change.

9.5.5.2 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)

9.5.5.3 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 transmisssion jobs.

9.5.5.4 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 22 - Parameters

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

9.5.5.5 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 23 - Parameters

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

10 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.

10.1 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

10.2 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 AT03246: SAM D20/D21 External Interrupt Driver (EXTINT) Application Note

http://www.atmel.com/Images/Atmel-42112-SAM-D20-D21-External-Interrupt-Driver-EXTINT_Application-Note_AT03246.pdf

Atmel AT03245: SAM D20/D21 Event System Driver

http://www.atmel.com/Images/Atmel-42108-SAM-D20-D21-Event-System-Driver_Application-Note_AT03245.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

11 Legal disclaimer

11.1 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.

11.2 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.

11.3 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.

12 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	

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

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

Modifications reserved | Printed in Germany
Specifications subject to change without notice