Application Note

BNO055 Xplained Pro
Data Stream

Bosch Sensortec

BOSCH
Invented for life

Application Note:	**BNO055 Xplained Pro - Data Stream**
Document Revision	1.0
Document Release	February 2015
Document Number	BST-BNO055-AN010-00
Technical Reference	0273141209
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 is to read data from a BNO055 smart sensor and transmits them out to a terminal software in a host computer.

This project is implemented on an Atmel evaluation board which is expanded by an Atmel BNO055 extension board.

This project is a reference example that shows how to use basic functions of BNO055 and can be extended and altered to implement desired custom use cases.

2 Prerequisites

To use this example a terminal software is required to be connected to EDBG Virtual COM port. The virtual port would be installed automatically after SAM D21 Xplained board is connected to the host computer via a micro USB cable. For this purpose Atmel Studio Terminal Extension is recommended.

3 Application Overview

For detailed description of application implementation see Application.

The application implemented by this project is to read data from a BNO055 smart sensor and transmits them out to a terminal software in a host computer.

This project is implemented on an Atmel evaluation board which is expanded by an Atmel BNO055 extension board (also known as BNO055 wing board). Figure 1 shows the hardware set. The wing board is connected to EXT1 port of the main board.

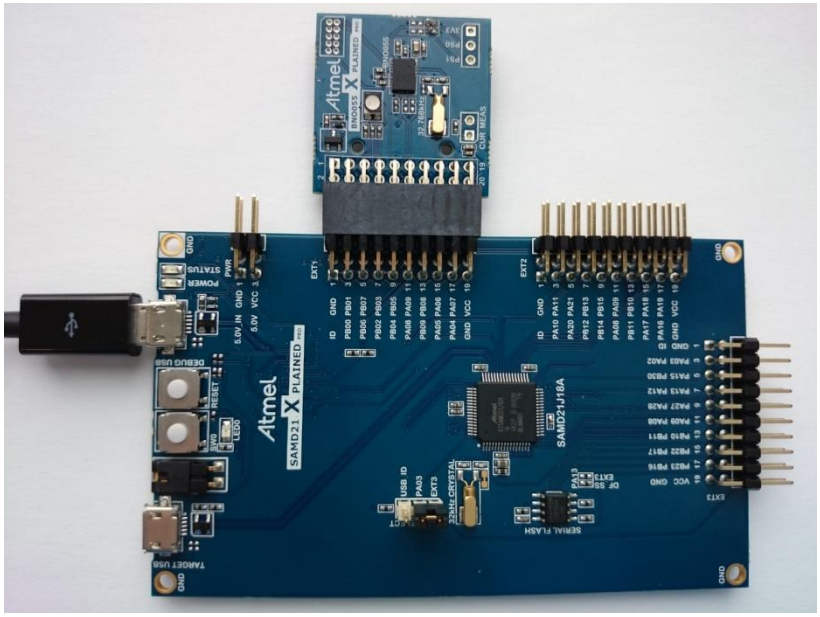

Figure 1 - Main and Extension board connections

BNO055 is a 9-axis smart sensor, which integrates a microcontroller and three orientation sensors (accelerometer, magnetometer, and gyroscope) in a package.

The MCU on the main board reads sensor data via an I2C bus. The I2C bus has a baud rate of 400 KHz and sensor data is requested on a period of 100 ms (can be changed, refer to Macro TC3_COUNT_VALUE). Raw sensor data is then transmitted using UART with baud rate of 115200 bps to the embedded debugger unit (EDBG). EDBG supports a USB Communication Device Class (CDC), which features a virtual COM interface on the device. Output can be read on a terminal software connected to this virtual COM port as shown in Figure 2. Atmel Studio terminal extension can be used here.

Sensor data in this project are the Euler angles, namely Heading (Yaw), Roll and Pitch. Output stream is printed on the terminal in **ASCII** format in an order depicted on Figure 2. In addition to Euler angles, calibration status of the three internal sensors of BNO055 is also printed on the terminal in the form of three digits. The first digit shows the status for accelerometer and the next two for the Gyroscope and Magnetometer respectively. The numbers are in the range of 0 to 3, with 3 indicating fully calibrated and 0 indicating not calibrated.

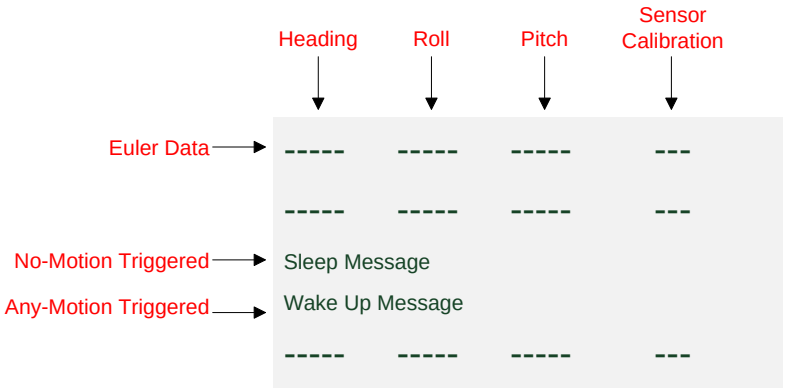

Figure 2- Terminal output for Sensor Data Stream

The sensor also reacts to motion/ no-motion by generating interrupts. If the sensor is left still, the data stream stops. In order to restart the stream the *any-motion* interrupt of the sensor should be triggered. Therefore the board has to be moved or rotated. Data Stream will continue until a *no-motion* interrupt is triggered, i.e. the sensor in not moved for some time. The movement threshold and waiting time to trigger interrupts can be configured by altering the code.

Figure 3 illustrates the sleep state transition of the system and Table 1 - RGB LED Colors shows RGB LED state in different states. In AWAKE mode BNO055 is operational, Euler data is read and printed on the terminal window and the LED color changes due to the orientation state of the sensor. In SLEEP mode BNO055 sleeps, data stream stops and the LED turns green. After reset the system is in AWAKE mode.

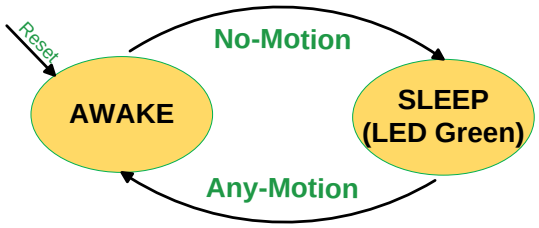

Figure 3 - Sleep State

Table 1 - RGB LED Colors

Color	SLEEP Mode	AWAKE Mode
Red	OFF	Proportional to Pitch
Blue	OFF	Proportional to Heading
Green	ON	OFF

3.1 MCU Peripherals

Microcontroller's peripherals that are used in this project are listed in Table 2 along with their functionality for this application.

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

Table 2 - MCU Peripherals

Peripheral	Functionality
Clock Source OSC8M	8 MHz clock source of MCU used as the source for multiple modules (Frequency 8 MHz)
Clock Source DFLL48M	DFLL clock source of MCU used as the source for multiple modules (Frequency 48 MHz – Closed Loop)
GCLK 0	Generates the main system clock, using DFLL as its source (Frequency 48 MHz)
GCLK 1	Generates clock signal for DFLL, using OSC8M as its source (Frequency 8MHz)
GCLK 3	Generates clock signal for TC3 and TC4, using OSC8M as its source (Frequency 500 KHz)
GCLK 4	Generates clock signal for SERCOM3 USART module, using OSC8M as its source (Frequency 8 MHz)
Timer/Counter 3	Scheduling sensor data reads (Default Period 100 ms)
Timer/Counter 4	Implementing delay function to be used by sensor API (in terms of milliseconds)
Timer/Counter 6	Generating PWM waveforms to derive RGB LED blue and red signals
SERCOM 2 I2C	Communication between MCU and BNO055 (Baud rate 400 KHz)
SERCOM 3 USART	Communication between MCU and the host computer (Baud rate = 112500, Data bits = 8, Parity = None, Stop bit = 1)
EXTINT Channel 4	External interrupt connected to BNO055 Interrupt pin (Acc No Motion, Acc Any Motion and Gyr Any Motion)

3.2 BNO055 Configuration

After power-on, BNO055 smart sensor is configured as given in Table 5. The sensor switches to NDOF fusion mode. In this mode the three internal sensors are configured as given in Table 4.

Three interrupts are activated and routed to the interrupt pin by setting the interrupt mask register. The activated interrupts are: *Slow/No Motion* and *Any Motion* of accelerometer and *Any Motion* of Gyroscope and their parameters are configured as given in Table 5.

Table 3 - Default Configuration of BNO055

Parameter	Value
Operation Mode	NDOF
Power Mode	Low Power
Interrupt	Active
Euler Angles Unit	Degree

Table 4 - BNO055 Internal Sensors' Configuration in NDOF Mode

Sensor	Range	Bandwidth	Power Mode
Accelerometer	±4 g	62.5 Hz	Normal
Magnetometer	N/A	20 Hz	Forced
Gyroscope	2000 °/s	32 Hz	Normal

Table 5 - Masked Interrupts' Configurations

Interrupt	Threshold	Duration	Axes
Accelerometer Slow/ No Motion	39.5 mg	2 s	X, Y, Z
Accelerometer Any Motion	39.5 mg	16 ms	X, Y, Z
Gyroscope Any Motion	4 °/s	375 ms	X, Y, Z

4 Hardware Platform

The hardware platform in this design consists of a BNO055 Xplained Pro extension board connected to a SAM D21 Xplaioned evaluation board. The sensor and SAM D21 microcontroller are connected using an I2C interface. The microcontroller is the host and the I2C bus master.

4.1 SAM D21 Xplained Pro

The main board in this design is an Atmel SAM D21 Xplained pro evaluation kit, which includes a microcontroller unit (MCU), an onboard debugger, a serial flash and three extension connectors. The extension connector pins can be used individually as digital pins or as a port to connect Atmel Xplained extension boards.

Figure 4 shows the block diagram of SAM D21 Pro. For more information about the board refer to Atmel SAM D21 Pro Kit documents.

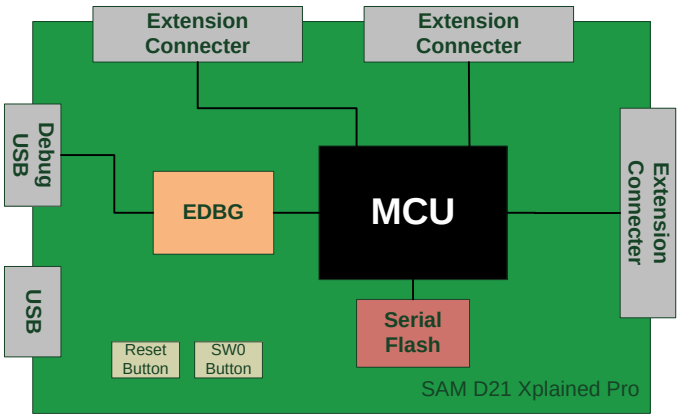

Figure 4- SAM D21 Xplained Pro Block Diagram (This is just an illustration)

4.1.1 Microcontroller

The microcontroller used on the board is ATSAMD21J18A, which is a low power microcontroller using the 32-bit ARM Cortex-M0+ processor.

For more information about the Microcontroller refer to ATSAMD21J18A documents.

4.1.2 Embedded Debugging

SAM D21 Xplained Pro supports onboard programming and debugging. An Atmel Embedded Debugger (EDBG) is included, which is connected to the MCU via a single wire debug interface (SWD) and can be used to flash and debug the software running on the MCU.

EDBG also implements a virtual COM port interface over UART. Its UART interface is connected to SERCOM3 USART module of the MCU, providing an easy communication of data between the MCU and a terminal software. This feature is used in current project to present sensor data.

Programming of the MCU is done in Atmel Studio via SWD and required hardware driver is provided on the board.

For more information about EDBG refer to Atmel Documents.

4.1.3 Serial Flash

The SAM D21 Xplained Pro has an onboard 8Mbit serial flash for non-volatile storage of data. (Not used in this project)

4.2 BNO055 Xplaiend Pro

BNO055 Xplained Pro is an extension board compatible with Atmel Xplained evaluation kits.

The board has a BNO055 chip, an ID module, an RGB LED and a debug interface connected to the sensor. Figure 5 shows the extension board's block diagram.

For more information about the extension board refer to BNO055 Xplained Pro documents from Atmel.

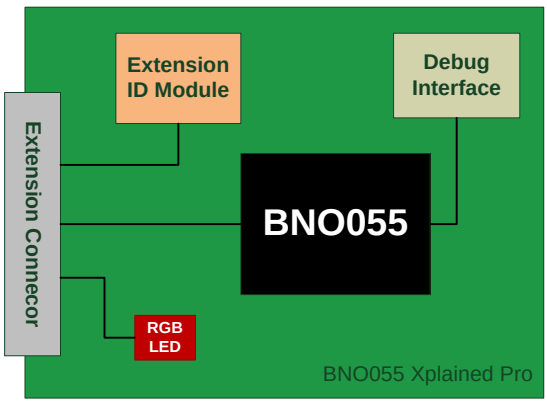

Figure 5 - BNO055 Xplained Pro Block Diagram (This is just an illustration)

4.2.1 BNO055

BNO055 is a 9-axis orientation sensor, which includes sensors and a microcontroller in a single package.

BNO055 is implements an intelligent 9-axis “Absolute Orientation Sensor”, which includes sensors and sensor fusion in a single package.

BNO055 is a System in a Package (SiP), integrating a triaxial 14-bit accelerometer, a triaxial 16-bit gyroscope with a range of ±2000 degrees per second, a triaxial geomagnetic sensor and a 32-bit microcontroller running the company’s BSX3.0 FusionLib software.

For more information about the device refer to BNO055 Datasheet.

4.2.2 Extension ID Module

Identification of extension boards for the Xplained Pro platform provides the ease of use for Atmel products. The intention of this identification is not to protect the hardware from being copied.

Identified expansion boards are reported via the Embedded Debugger (on the main board) to the host PC software, which is Atmel Studio. Based on the detected hardware Atmel Studio will then provide additional information to the user such as link to user guides and relevant datasheets available Atmel Software Framework (ASF) applications for the extension and extension revision and features.

This module is part of the board design and the user shall not alter the configurations.

For more information about Xplained Pro ID system refer to Hardware Development kit User Guide from Atmel.

4.2.3 Debug Interface

BNO055 Xplained Pro features a Cortex® Debug Connector (10-pin) for programming and debugging the sensor. The connector is limited to the SWD interface and is intended for in-system programming and debugging of the sensor in the users final product.

For more information regarding in-system programming/ debugging, refer to relevant documents from Atmel.

5 Design Structure

Figure 6 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 .

ASF driver supports are support files for different components of MCU used in the project (Clock, I2C, TC, USART, EXTINT and I/O). 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 support defines functions to implement communications between BNO055 smart sensor API and ASF Drivers (I2C and TC). For more information refer to Sensor API Support Implementation.

Sensor API is provided by Bosch Sensortec and defines structures and functions to communicate with the sensor. For more information refer to Bosch Sensortec .

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

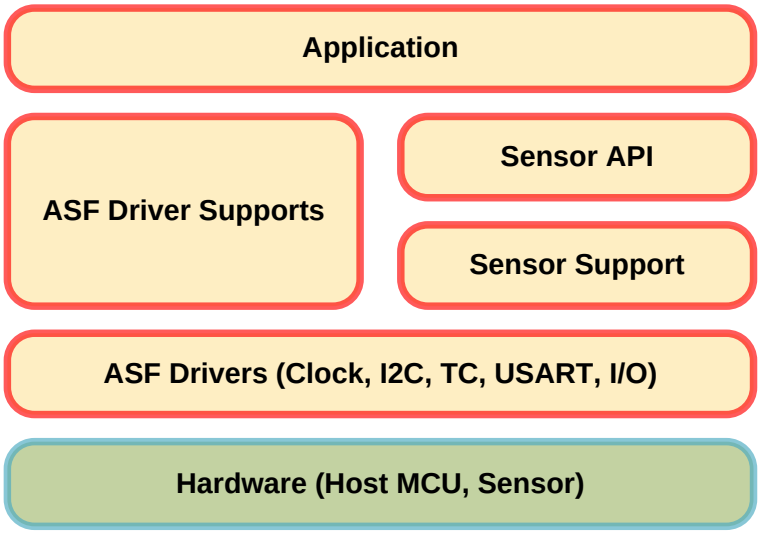

Figure 6 - Design Structure

5.1 Folder Structure

Figure 7 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 BNO055 smart sensor API 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.
- "main.c" defines the *main* function of the project along with implementation of application specific functionalities, such as initializing the whole system, configuring required MCU components and sensors, reading and streaming sensor data.

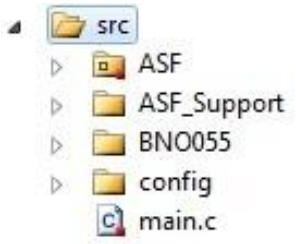

Figure 7 - Project Folder Structure

6 Application Implementation

6.1 Main

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

BNO055 data stream 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.

6.1.1 Includes

6.1.1.1 Include “asf.h”

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

6.1.1.2 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

6.1.1.3 Include “bno050_support.h”

Wrapper functions for BNO055 connection to the host microcontroller

6.1.1.4 Include “clock_support.h”

Wrapper functions for ASF clock and generic clock modules

6.1.1.5 Include “i2c_support.h”

Wrapper functions for ASF I2C module

6.1.1.6 Include “tc_support.h”

Wrapper functions for ASF Timer/Counter module

6.1.1.7 Include “usart_suppot.h”

Wrapper functions for ASF USART module

6.1.1.8 Include “gpio_support.h”

Wrapper functions for ASF I/O pins driver

6.1.1.9 Include “extint_support.h”

Wrapper functions for ASF external interrupt channels module

6.1.2 Type Definitions

N/A

6.1.3 Macro Definitions

6.1.3.1 Macro READ_SENSORS_FLAG

```
#define READ_SENSORS_FLAG          tc3_callback_flag
```

TC3 callback flag is defined as read sensors flag; i.e. sensor data are read in accordance with TC3 callback.

6.1.3.2 Macro SLEEP_STATE_SLEEP

```
#define SLEEP_STATE_SLEEP          UINT8_C(1)
```

This macro defines the value ‘1’ as the sleep state SLEEP. This is the state in which sensor data streaming stops.

6.1.3.3 Macro SLEEP_STATE_AWAKE

```
#define SLEEP_STATE_AWAKE          UINT8_C(0)
```

This macro defines the value ‘0’ as the sleep state AWAKE. This is the state in which the sensor is streaming data.

6.1.3.4 Macro RGB_LED_R

```
#define RGB_LED_R                  EXT1_PIN_8
```

This macro defines the EXT1_PIN_8 pin on the main board as RGB LED red signal.

6.1.3.5 Macro RGB_LED_G

```
#define RGB_LED_G                  EXT1_PIN_6
```

This macro defines the EXT1_PIN_6 pin on the main board as RGB LED green signal.

6.1.3.6 Macro RGB_LED_B

```
#define RGB_LED_B                  EXT1_PIN_7
```

This macro defines the EXT1_PIN_7 pin on the main board as RGB LED blue signal.

6.1.3.7 Macro RGB_LED_G_ON

```
#define RGB_LED_G_ON                false
```

The green light of the RGB LED is connected to a GPIO pin and it glows if the corresponding signal is set LOW.

6.1.3.8 Macro RGB_LED_G_OFF

```
#define RGB_LED_G_OFF               true
```

The green light of the RGB LED is connected to a GPIO pin and it turns off if the corresponding signal is set HIGH.

6.1.3.9 Macro RGB_LED_B_VALUE

```
#define RGB_LED_B_VALUE
      (0xFFFF - (((uint16_t)bno055_euler_data.h) * 0xFFFF / 5759))
```

The blue light of the RGB LED is connected to a PWM waveform output pin and the waveforms duty cycle determines the brightness of the color. The duty cycle depends on the corresponding capture channel compare value. This macro changes the compare value proportional to the Euler heading angle.

6.1.3.10 Macro RGB_LED_B_OFF

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

The blue light of the RGB LED is connected to a PWM waveform output pin and it turns off if the corresponding signal has a duty cycle of '0'. By setting the corresponding capture channel to its maximum value (0xFFFF), the duty cycle goes to zero.

6.1.3.11 Macro RGB_LED_R_VALUE

```
#define RGB_LED_R_VALUE
      (0xFFFF - (((uint16_t)bno055_euler_data.p) * 0xFFFF / 5759))
```

The red light of the RGB LED is connected to a PWM waveform output pin and the waveforms duty cycle determines the brightness of the color. The duty cycle depends on the corresponding capture channel compare value. This macro changes the compare value proportional to the Euler pitch angle.

6.1.3.12 Macro RGB_LED_R_OFF

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

The red light of the RGB LED is connected to a PWM waveform output pin and it turns off if the corresponding signal has a duty cycle of '0'. By setting the corresponding capture channel to its maximum value (0xFFFF), the duty cycle goes to zero.

6.1.3.13 Macro RGB_LED_UPDATE

```
#define RGB_LED_B_R_UPDATE(cc0_value, cc1_value)\
            tc6_capture_channels_update(cc0_value, cc1_value)
```

This macro defines the function *tc6_capture_channel_update* as RGB_LED_B_R_UPDATE.

The blue and red colors of the LED are connected to two PWM signals derived by TC6 and the capture channel values determine the PWM duty cycle and thus the blue and red brightness.

6.1.4 Global Variables/ Structures

6.1.4.1 Structure bno055_euler_data

```
struct bno055_euler_t                bno055_euler_data
```

This structure holds BNO055 Euler data which consists of three Euler angles (Roll, Pitch and Yaw).

6.1.4.2 Integer sleep_state

```
uint8_t sleep_state
```

This variable holds sleep state of the system. In SLEEP state BNO055 goes to sleep mode and sensor data stream stops. Figure 3 shows the corresponding state transition diagram.

6.1.5 Function Definitions

6.1.5.1 Function sensors_init

Initializes BNO055 smart sensor

```
void sensors_init(void)
```

This function calls initialization function of BNO055, sets sensor's power mode to *LOWPOWER*, sets sensor's operation mode to NDOF and enables the required interrupts (Accelerometer Any Motion, Accelerometer No Motion, Gyroscope Any Motion).

6.1.5.2 Function sensors_data_print

Reads output data of the sensor and sends them data via USART

```
void sensors_data_print (void)
```

This function reads output data of BNO055 (Three Euler angles) using sensor API functions and sends sensor data in ASCII Format via USART in the order shown in Figure 2.

6.1.5.3 Function bno055_interrupt_handler_no_motion

No motion interrupt handler

```
void bno055_interrupt_handler_no_motion(void)
```

This function is called when a *no-motion interrupt* is triggered by the accelerometer in BNO055. It turns the LED color to green, stops data stream and sends a message to the terminal window in ASCII format.

6.1.5.4 Function bno055_interrupt_handler_any_motion

Any motion interrupt handler

```
void bno055_interrupt_handler_any_motion(void)
```

This function is called when an *any-motion interrupt* is triggered by the accelerometer or gyroscope in BNO055. It turns off the green light, sends a message to the terminal window in ASCII format and starts data stream.

6.1.5.5 Function bno055_interrupt_handler

Sensor general interrupt handler, calls specific handlers

```
void bno055_interrupt_handler(void)
```

This function is called when an external interrupt is triggered by the sensor, it checks interrupt registers of BNO055 to determine the source and type of interrupt and calls the specific interrupt handler accordingly.

6.1.5.6 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 main board and the sensor. It initializes the host microcontroller unit and all its required modules such as clock sources, I2C, TC, USART, PINMUX and interrupt controllers. It also initializes the global variables and turns the RGB LED off.

In the infinite loop it repeatedly executes two main tasks:

- Streams sensor data periodically (100 ms) via USART
- Checks SW0 pushbutton state and resets BNO055 if pressed

7 Sensor API Support Implementation

7.1 BNO055 Support

This chapter describes the declarations and definitions in “bno055_support.h” and “bno055_support.c” files.

BNO055 support defines functions to interface the sensor API with the actual BNO055 smart sensor via I2C. It implements bus read/write, delay and reset functions that are needed for this communication. It also defines the sensor initialization routine.

7.1.1 Includes

7.1.1.1 Include “bno055.h”

Includes BNO055 sensor API which provides sensor’s data structures and driver functions

7.1.1.2 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

7.1.1.3 Include “i2c_support.h”

Includes ASF I2C driver support file which provides I2C master instance, configuration functions and driver functions

7.1.1.4 Include “tc_support.h”

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

7.1.1.5 Include “gpio_support.h”

Includes ASF GPIO driver support file which provides configuration functions and driver functions for GPIO pins of the MCU

7.1.2 Type Definitions

N/A

7.1.3 Macro Definitions

7.1.3.1 Macro BNO055_I2C_SLAVE_ADDRESS

```
#define BNO055_I2C_SLAVE_ADDRESS          BNO055_I2C_ADDR2
```

This macro determines the I2C slave address of BNO055. The default I2C address of the BNO055 device is 0101001b (0x29). The alternative address 0101000b (0x28), in I2C mode the input pin ADDR0 on the BNO055-Xplained board can be used to select between the two I2C addresses as shown in Table 6.

Table 6 - BNO055 I2C Address Selection

BNO055_I2C_SLAVE_ADDRESS	ADDR0 State	I2C Address
BNO055_I2C_ADDR2	HIGH	0x29
BNO055_I2C_ADDR1	LOW	0x28

7.1.3.2 Macro BNO055_I2C_ADDR_PIN

```
#define BNO055_I2C_ADDR_PIN             EXT1_PIN_5
```

This macro is the external pin on the main board connected to the BNO055 I2C ADDR pin.

7.1.3.3 Macro BNO055_BOOT_LOAD_PIN

```
#define BNO055_BOOT_LOAD_PIN            EXT1_PIN_10
```

This macro is the external pin on the main board connected to the BNO055 BOOTLOAD pin.

7.1.3.4 Macro BNO055_RESET_PIN

```
#define BNO055_RESET_PIN                EXT1_PIN15
```

This macro is the external pin on the main board connected to the BNO055 RESET pin.

7.1.3.5 Macro BNO055_RESET_ACTIVE

```
#define BNO055_RESET_ACTIVE             false
```

This macro determines the active state of BNO055 reset.

7.1.3.6 Macro BNO055_RESET_DELAY_MS

```
#define BNO055_RESET_DELAY_MS           UINT32_C(600)
```

This macro is the delay required to wait for BNO055 chip to reset.

7.1.4 Global Variables/ Structures

7.1.4.1 Structure bno055

```
struct bno055_t bno055
```

It instantiates a BNO055 software instance structure which retains chip ID, internal sensors IDs, I2C address and pointers to required functions (bus read/write and delay functions).

7.1.4.2 Structure bno_i2c_packet

```
struct i2c_packet bno_i2c_packet
```

It instantiates an I2C packet software instance structure which retains I2C slave address, data buffer and data length and is used to read/write data on the I2C bus.

7.1.5 Function Definitions

7.1.5.1 Function bno055_initialize

Initializes BNO055 smart sensor and its required connections

```
void bno055_initialize(void)
```

This function sets the *bus_write*, *bus_read* and *delay_msec* functions of BNO055 API to point to provided functions so that the sensor can communicate with the MCU via I2C. It also resets and initializes the sensor and sets the I2C address of the device.

7.1.5.2 Function bno055_i2c_write

Sends data to BNO055 via I2C

```
int8_t bno055_i2c_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 BNO055 sensor. The communication is done via I2c so ASF I2C driver functions are used.

Table 7 - Function Parameters

Data Direction	Parameter Name	Description

[in]	dev_addr	Device I2C slave address
[in]	reg_addr	Address of destination register
[in]	reg_data	Pointer to data buffer to be sent
[in]	length	Length of the data to be sent

7.1.5.3 Function bno_i2c_read

Receives data from BNO055 on I2C

```
int8_t bno055_i2c_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 BNO055 sensor. The communication is done via I2C so ASF I2C driver functions are used.

Table 8 - Function Parameters

Data Direction	Parameter Name	Description
[in]	dev_addr	Device I2C slave address
[in]	reg_addr	Address of source register
[out]	rx_data	Pointer to data buffer to be received
[in]	length	Length of the data to be received

7.1.5.4 Function bma_delay_msec

Initiates a delay of the length of the argument in milliseconds

```
void bno055_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 - Function Parameters

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

		milliseconds

7.1.5.5 Function bno055_reset

Resets the BNO055 chip

```
void bno055_reset(void)
```

This function triggers a hard reset in BNO055 chip by giving the value ‘0’ to its RESET pin.

8 ASF Driver Supports Implementation

8.1 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.

8.1.1 Includes

8.1.1.1 Include “clock.h”

SAM D21 Clock Driver from Atmel

8.1.1.2 Include “glck,h”

SAM D21 Generic Clock Driver from Atmel

8.1.2 Type Definitions

N/A

8.1.3 Macro Definitions

N/A

8.1.4 Global Variables/ Structures

N/A

8.1.5 Function Definitions

8.1.5.1 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, 3 and 4* 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: 8 MHz
- DFLL: Closed Loop, 48 MHz
- GCLK0: 48 MHz
- GCLK1: 8 MHz
- GLCK3: 500 KHz
- GCLK4: 8 MHz

8.1.5.2 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, changes its mode to closed loop, sets the maximum coarse and fine steps and enables the clock source. (Disabled by default)

8.1.5.3 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 1. (Enabled by default)

8.1.5.4 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 and changes its clock source to DFLL48M, hence providing a 48 MHz clock on GCLK_GENERATOR_0. (Enabled by default)

8.1.5.5 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, hence providing an 8 MHz clock on GCLK_GENERATOR_1. (Disabled by default)

8.1.5.6 Function clock_configure_gclk_generator_3

Configures *generic clock generator 3*

```
void clock_configure_gclk_generator_3(void)
```

This function configures *generic clock generator 3* of the MCU with default configuration values, changes its division factor to 16 and enables the clock generator, hence providing a 500 KHz clock on GCLK_GENERATOR_3. (Disabled by default)

8.1.5.7 Function clock_configure_gclk_generator_4

Configures *generic clock generator 4*

```
void clock_configure_gclk_generator_4(void)
```

This function configures *generic clock generator 4* of the MCU with default configuration values and enables the clock generator, hence providing an 8 MHz clock on GCLK_GENERATOR_4. (Disabled by default)

8.2 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 channel that is connected to the interrupt pin of BNO055 sensor.

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

8.2.1 Includes

8.2.1.1 Include “extint.h”

SAM D21 external interrupt driver from Atmel

8.2.1.2 Include “extint_callback.h”

SAM D21 external interrupt callback diver from Atmel

8.2.1.3 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

8.2.2 Type Definitions

N/A

8.2.3 Macro Definitions

N/A

8.2.4 Global Variables/ Structures

```
volatile bool extint_callback_detect_flag
```

This flag is *false* by default and is set by the external interrupt callback function; i.e. it is set whenever a rising signal edge occurs on the corresponding pin. The flag can be used by other functions to execute desired tasks accordingly.

8.2.5 Function Definitions

8.2.5.1 Function extint_initialize

Initializes the external interrupt channel of the MCU

```
void extint_initialize(void (*extint_handler_function)(void))
```

This function calls configuration function and the callback configuration function of the external interrupt channel. 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 10 - Function Parameters

Data Direction	Parameter Name	Description
[in, out]	extint_handler_function	Pointer to the interrupt handler function

8.2.5.2 Function extint_configure

Configures the external interrupt channel of the MCU

```
void extint_configure (void)
```

This function configures the external interrupt channel available on external port 1 of the main board (EXT1_IRQ_INPUT) 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.

8.2.5.3 Function extint_configure_callbacks

Configures external interrupt 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)

One callback type is enabled:

- Buffer Received: Interrupts after reception jobs.

8.2.5.4 Function extint_detection_callback

Called after external interrupt detected

```
void extint_detection_callback(void)
```

This function is called after detection of rising edge on the corresponding interrupt pin, sets the corresponding flag and calls the handler function.

8.2.5.5 Function Pointer extint_handler_function_ptr

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

```
void (*extint_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 each external interrupt the desired function is called via this pointer.

8.3 GPIO Support

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

GPIO support uses ASF I/O and GPIO driver modules and defines initialization and configuration functions for the microcontroller’s general purpose I/O pins that are needed for different functionalities in this design.

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

8.3.1 Includes

8.3.1.1 Include “pinmux.h”

SAM D21 physical I/O Pins driver from Atmel

8.3.1.2 Include “port.h”

SAM D21 General Purpose Input/Output (GPIO) pins diver from Atmel

8.3.1.3 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

8.3.2 Type Definitions

N/A

8.3.3 Macro Definitions

N/A

8.3.4 Global Variables/ Structures

N/A

8.3.5 Function Definitions

8.3.5.1 Function gpio_initialize

Initializes the GPIO pins of the MCU

```
void gpio_initialize(void)
```

This function calls configuration functions of GPIO pins.

There are a total of four GPIOs that are needed by BNO055-Xplained as mentioned in Table 11.

Table 11 - GPIO Pins

on SAM D21 Xplained	on BNO055 Xplained	Functionality
EXT1_PIN_5	ADDR0	BNO055 I2C address select
EXT1_PIN_6	LEDG	RGB LED Green signal
EXT1_PIN_10	BOOT	BNO055 Boot Loader mode select
EXT1_PIN_15	RESET	BNO055 Reset

8.3.5.2 Function gpio_configure_ext1_pin_5

Configures the corresponding pin as a GPIO

```
void gpio_configure_ext1_pin_5(void)
```

This function configures the EXT1_PIN_5 pin with default multiplexer configuration values and sets the direction to *output*. (GPIO by default)

The functionality of this output pin is to select I2C address of BNO055 between the two available options. (High value results in the address 0x29)

8.3.5.3 Function gpio_configure_ext1_pin_6

Configures the corresponding pin as a GPIO

```
void gpio_configure_ext1_pin_6(void)
```

This function configures the EXT1_PIN_6 pin with default multiplexer configuration values and sets the direction to *output*. (GPIO by default)

The functionality of this output pin is to turn the green light in the RGB LED on or off. (High value turns the green color off)

8.3.5.4 Function gpio_configure_ext1_pin_10

Configures the corresponding pin as a GPIO

```
void gpio_configure_ext1_pin_10(void)
```

This function configures the EXT1_PIN_10 pin with default multiplexer configuration values and sets the direction to *output*. (GPIO by default)

The Functionality of this output pin is to select between bootloader mode or application mode of BNO055. (High value leads to application mode)

8.3.5.5 Function gpio_configure_ext1_pin_15

Configures the corresponding pin as a GPIO

```
void gpio_configure_ext1_pin_15(void)
```

This function configures the EXT1_PIN_15 pin with default multiplexer configuration values and sets the direction to *output*. (GPIO by default)

The functionality of this output pin is to reset BNO055. (High value resets the sensor)

8.4 I2C Support

This chapter describes the functions declared in “i2c_support.h” file and defined in “i2c_support.c” file.

I2C support uses ASF I2C driver modules and defines initialization and configuration functions for the microcontroller’s I2C peripheral that is needed to communicate with BNO055 smart sensor.

For more information about the I2C modules refer to the corresponding application note or the microcontroller datasheet from Atmel.

8.4.1 Includes

8.4.1.1 Include “i2c_common.h”

SAM D21 I2C driver from Atmel

8.4.1.2 Include “i2c_master.h”

SAM D21 I2C master mode driver from Atmel

8.4.1.3 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

8.4.2 Type Definitions

N/A

8.4.3 Macro Definitions

N/A

8.4.4 Global Variables/ Structures

8.4.4.1 Structure i2c_master_instance

```
struct i2c_master_module i2c_master_instance
```

Instantiates a SERCOM I2C driver software structure, used to retain software state information of the associated hardware module instance.

8.4.5 Function Definitions

8.4.5.1 Function i2c_initialize

Initializes I2C module of the MCU

```
void i2c_initialize(void)
```

This function calls configuration functions for I2C master.

8.4.5.2 Function i2c_configure_master

Configures I2C master module of the MCU

```
void i2c_configure_master(void)
```

This function configures the I2C master module with default configuration values, sets baud rate to 400 KHz, sets I2C master pads, initializes i2c sercom module of the corresponding extension port (EXT1_I2C_SERCOM) with I2C master configurations and enables the module. (Disabled by default)

8.5 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.

8.5.1 Includes

8.5.1.1 Include “tc.h”

SAM D21 Timer Counter driver from Atmel

8.5.1.2 Include “tc_interrupt.h”

SAM D21 Timer Counter callback driver from Atmel

8.5.2 Type Definitions

N/A

8.5.3 Macro Definitions

8.5.3.1 Macro COUNT_MAX_32_BIT

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

Maximum value of a 32-bit counter

8.5.3.2 Macro COUNT_MAX_16_BIT

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

Maximum value of a 16-bit counter

8.5.3.3 Macro TC3_PERIOD_100MS

```
#define TC3_PERIOD_100MS     COUNT_MAX_16BIT - UINT16_C(50000)
```

Loading this value onto the 16-bit count register of TC3, causes the counter to overflow after 100 milliseconds. (Assuming it has a 500 kHz clock source)

8.5.3.4 Macro TC3_PERIOD_50MS

```
#define TC3_PERIOD_50MS       COUNT_MAX_16BIT - UINT16_C(25000)
```

Loading this value onto the 16-bit count register of TC3, causes the counter to overflow after 50 milliseconds. (Assuming it has a 500 kHz clock source)

8.5.3.5 Macro TC3_COUNT_VALUE

```
#define TC3_COUNT_VALUE       TC3_PERIOD_100MS
```

This macro defines the value loaded onto TC3 count register initially and after each overflow. The default value selected causes the counter to overflow on a period of 100 milliseconds.

8.5.4 Global Variables/ Structures

8.5.4.1 Structure tc3_instance

```
struct tc_module tc3_instance
```

Instantiates a TC software instance structure, used to retain software state information of the associated hardware module instance, which in this case is TC3.

8.5.4.2 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.

8.5.4.3 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.

8.5.4.4 Boolean tc3_callback_flag

```
volatile bool tc3_callback_flag
```

This flag is *false* by default and is set by TC3 callback function; i.e. it is set whenever TC6 counter overflows. The flag can be used by other functions to execute desired tasks accordingly.

8.5.4.5 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.

8.5.5 Function Definitions

8.5.5.1 Function tc_initialize

Initializes TC3, TC4 and TC6 timer/counter modules of the MCU

```
void tc_initialize(void)
```

This function calls configuration functions and callback configuration functions of TC3, TC4 and TC6. After initialization, the TC3, TC4 and TC6 are configured and enabled.

8.5.5.2 Function tc3_configure

Configures TC3 of the MCU

```
void tc3_configure(void)
```

This function configures TC3 with default configuration values, sets the counter size to 16 bits, sets its clock source to GCLK_GENERATOR_3 and enables the module. (Disabled by default)

8.5.5.3 Function tc3_configure_callbacks

Configures TC3 callback register

```
void tc3_configure_callbacks(void)
```

This function configures TC3 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.

8.5.5.4 Function tc3_callback

Called when TC3 counter is equal to its capture channel 0 value

```
void tc3_callback(struct tc_module *const module_inst_ptr)
```

This Function is called whenever TC3 counter overflows. It reloads the counter value and sets the corresponding flag.

Table 12 - Parameters

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

8.5.5.5 Function tc3_stop_counter

Stops TC3 counter

```
void tc3_stop_counter(void)
```

This function calls TC3 stop counter function.

8.5.5.6 Function tc3_start_counter

Starts TC3 counter

```
void tc3_start_counter(void)
```

This function calls TC3 start counter function.

8.5.5.7 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)

8.5.5.8 Function tc4_configure_callbacks

Configures TC6 callback register

```
void configure_tc4_callbacks(void)
```

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

One callback type is enabled:

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

8.5.5.9 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 13 - Parameters

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

8.5.5.10 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 14 - Parameters

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

8.5.5.11 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 16 bits, sets its clock source to GCLK_GENERATOR_1 and enables the module. (Disabled by default)

Both capture channels (CCs) of the module are enabled and their output pin multiplexer is set so that it generates two PWM waveforms which are connected to LEDB and LEDR pins of the BNO055-Xplained and are used to derive blue and red signals of the RGB LED on the extension board.

8.5.5.12 Function tc6_capture_channels_update

Changes the compare value of two capture channels of TC6

```
void tc6_capture_channels_update
                        (uint32_t cc0_value, uint32_t cc1_value)
```

This function changes the compare value of CC0 and CC1 of TC6 to the given values, so that the duty cycle of generated PWM waveforms change accordingly.

Table 15 - Parameters

Data Direction	Parameter Name	Description
[in]	cc0_value	Compare Value of Capture Channel 0
[in]	cc1_value	Compare Value of Capture Channel 1

8.6 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.

8.6.1 Includes

8.6.1.1 Include “usart.h”

SAM D21 SERCOM USART driver from Atmel

8.6.1.2 Include “usart_interrupt.h”

SAM D21 SERCOM USART Asynchronous driver from Atmel

8.6.1.3 Include “samd21_xplained_pro.h”

Includes board specified macros such as external ports definitions and components available via these ports

8.6.2 Type Definitions

N/A

8.6.3 Macro Definitions

8.6.3.1 Macro

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

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

8.6.3.2 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.

8.6.4 Global Variables/ Structures

8.6.4.1 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.

8.6.4.2 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.

8.6.5 Function Definitions

8.6.5.1 Function usart_initialize

Initializes the USART module of the MCU

```
void usart_initialize(void)
```

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

8.6.5.2 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_4, sets the pads, initializes the corresponding SERCOM module with the USART configuration and enables the module. (Disabled by default)

The SERCOM Module used here interfaces the embedded debugger CDC gateway USART (EDBG_CDC_MODULE) so that data will be transmitted to a virtual COM port on the host computer.

8.6.5.3 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)

One callback type is enabled:

- Buffer Received: Interrupts after reception jobs.

8.6.5.4 Function usart_callback_receive

Called after USART receptions

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

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

Table 16 - Parameters

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

9 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.

9.1 Bosch Sensortec References

BNO055 Sensor Driver

https://github.com/BoschSensortec/BNO055_driver

BNO055 Data Sheet

http://ae-bst.resource.bosch.com/media/products/dokumente/bno055/BST_BNO055_DS000_12.pdf

9.2 Atmel References

9.2.1 Data Sheets

Atmel-42181: SAM-D21_Datasheet

http://www.atmel.com/Images/Atmel-42181-SAM-D21_Datasheet.pdf

9.2.2 User Guides

Atmel-42220: SAM D21 Xplained Pro Evaluation Kit User Guide

http://www.atmel.com/Images/Atmel-42220-SAMD21-Xplained-Pro_User-Guide.pdf

Atmel-42096: Microcontroller Embedded Debugger (EDBG) User Guide

http://www.atmel.com/images/atmel-42096-microcontrollers-embedded-debugger_user-guide.pdf

Atmel-42091: Hardware Development kit (HDK) User Guide

https://gallery.atmel.com/docs/Hardware_Development_Kit.pdf

9.2.3 Application Notes

Atmel-42117: SAM D20/D21 I2C Driver (SERCOM I2C) Application Note

http://www.atmel.com/Images/Atmel-42117-SAM-D20-D21-I2C-Bus-Driver-SERCOM-I2C_Application-Note_AT03250.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-Clock-Management-Driver-CLOCK_Application-Note_AT03259.pdf

Atmel-42256: SAM D21 Timer/Counter Driver (TC) Application Note

http://www.atmel.com/Images/Atmel-42256-SAM-D21-Timer-Counter-for-Control-Applications-Driver-TCC_AP-Note_AT07058.pdf

Atmel-42112: SAM D20/D21 External Interrupt Driver (EXTI-NT) Application Note

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

Atmel-42121: SAM D20/D21 Pin Multiplexer (PINMUX) Application Note

http://www.atmel.com/Images/Atmel-42121-SAM-D20-D21-Pin-Multiplexer-Driver-PINMUX_Application-Note_AT03262.pdf

10 Legal disclaimer

10.1 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.

10.2 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.

10.3 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.

11 Document History and Modifications

Rev. No.	Chapter	Description of Modification/ Changes	Date
1.0		Document Created	2015-02-09

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

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

Modifications reserved | Printed in Germany
Specifications subject to change without notice
Document number: BST-BNO055-AN010-00
Revision_1.0_022015