



CEE-445 Embedded System Design

2. STMicroelectronics SensorTile Lab: Sensor System Signal Acquisition, Event Detection and Configuration



Table of Contents

1. INTRODUCTION TO THIS TUTORIAL	3
1.1. LIST OF REQUIRED EQUIPMENT AND MATERIALS	3
1.2. PREREQUISITE TUTORIALS	3
2. GETTING STARTED	4
3. MODIFYING SAMPLING RATE: OUTPUT DATA RATE	5
3.1. EXAMINING THE SENSOR SAMPLING RATE AND OUTPUT DATA RATE	5
3.2. MODIFYING THE USB DATA OUTPUT RATE	8
4. CREATING NEW DATA, NEW MESSAGES, AND EVENT DETECTION	10
4.1. COMPUTING VECTOR MAGNITUDE ACCELERATION	10
4.2. DETECTING EVENTS: ACCELERATION VECTOR MAGNITUDE THRESHOLD CROSSING	14
5. MODIFYING DATA ACQUISITION PARAMETERS	16
5.1. ADJUST SCALING FACTOR (FULL-SCALE FS)	16
5.2. MAKING SIMILAR MODIFICATIONS FOR OTHER PARAMETERS	19
6. UNDERSTANDING THE GYROSCOPE	20



1. Introduction

The Lab steps provide:

1. An introduction to control of sensor signal acquisition and sensor system configuration. These topics are fundamental to IoT system development.
2. An introduction to sensor signal detection and notifications.
3. Experience in software system development for the SensorTile IoT system demonstrating important capabilities of the System WorkBench Integrated Development Environment in accelerating system development.

1.1. List of Required Equipment and Materials

- 1) 1x STMicroelectronics SensorTile kit.
- 2) 1x STMicroelectronics Nucleo Board.
- 3) 1x Personal Computer with two USB type-A inputs OR you must have a powered USB hub.
- 4) 1x USB 2.0 A-Male to Micro-B Cable (micro USB cable).
- 5) 1x USB 2.0 A-Male to Mini-B Cable (mini USB cable).
- 6) Network access to the Internet.



2. Getting Started

In the last lab document, we explored how to install and use the IDE. We will now examine more features such as “Open Declaration” to find where functions or variables are defined and how they are implemented in C language. Users will first be guided through the process of opening the C-code source file labelled “main.c” in order to become more familiar with these features.

1. Open the IDE (Eclipse or System WorkBench) on your personal computer.
2. Once the IDE is open, open **main.c**. If **main.c** is not the first file the IDE automatically opens, double click the file labelled “main.c” found under the following directory. Examine Figure 1 for more details.

“STM32L4xx-SensorTile > DataLog > User”

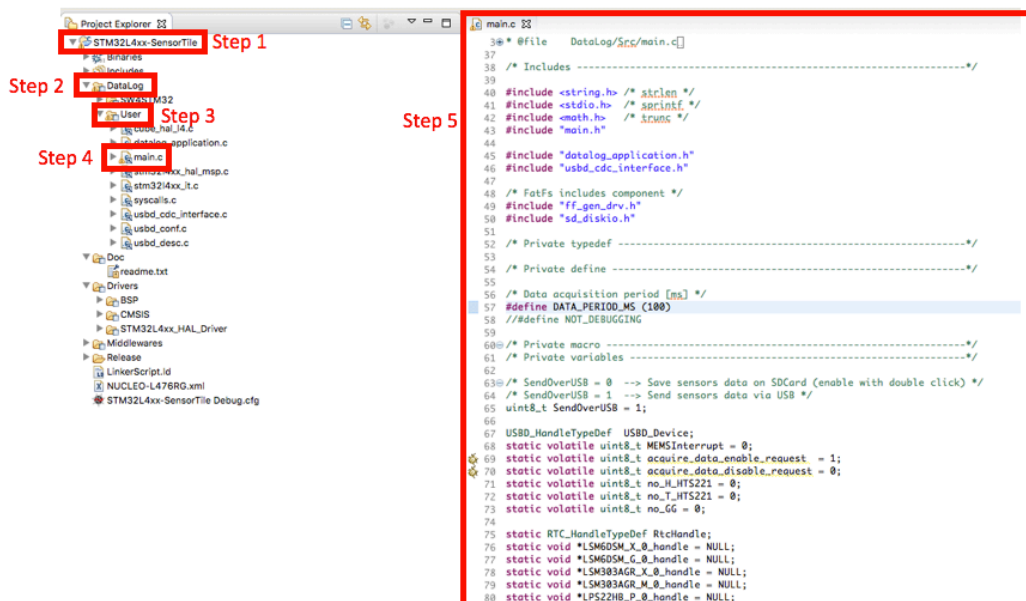


Figure 1: Opening 'main.c'. Make sure that the file looks similar to the file labelled "Step 5", and that the title is 'main.c' as shown in this figure.

3. Find the line in the file that says “int main(void)”.



3. Modifying Sampling Rate: Output Data Rate

3.1. Examining the Sensor Sampling Rate and Output Data Rate

1. Build the project without making any modifications. Run this on the SensorTile in debug mode as instructed in previous lab.
2. Examine the messages received by your personal computer over the serial connection with Putty software.
3. Examine the messages regarding "TimeStamp". These are highlighted in Figure 2.

```

HH:MM:SS.millisec
TimeStamp: 00:01:04.03 T1
ACC_X: -25, ACC_Y: -16, ACC_Z: 1006
GYR_X: 140, GYR_Y: -1960, GYR_Z: 350
MAG_X: -70, MAG_Y: -108, MAG_Z: -280
PRESS: 1001.21
TimeStamp: 00:01:04.14 T2
ACC_X: -26, ACC_Y: -18, ACC_Z: 1008
GYR_X: 140, GYR_Y: -1750, GYR_Z: 420
MAG_X: -76, MAG_Y: -103, MAG_Z: -277
PRESS: 1001.21
TimeStamp: 00:01:04.24 T3
ACC_X: -28, ACC_Y: -18, ACC_Z: 1008

```

Figure 2: Examining differences in TimeStamp.

4. Calculate the difference between 5 successive timestamps (i.e. $t_1 = T_2 - T_1$, $t_2 = T_3 - T_2$, ...).
5. Average these differences $(t_1 + t_2 + t_3 + t_4 + t_5)/5$. What is this average difference? See Figure 2.



We shall refer to this average difference in timestamps as the “USB data period”.

This is not the rate at which data is recorded by the sensor. This is the rate at which data is transmitted to the host computer over the serial USB data transport. The sensor sampling rate will be adjusted in future lab exercise.

6. First, we must find the location where the data rate is being referenced. Once we find this location, we can inspect where the variable to set the output data rate is being defined.

Once we have found where the variable is defined, we can edit the value.

Once the value is edited, the code can be recompiled and uploaded to the board to affect system behavior.

7. Find the line where the output data rate is being used in **main.c**. The line is highlighted in Figure 3.

```

160  /* Initialize and Enable the available sensors */
161  initializeAllSensors();
162  enableAllSensors();
163
164  while (1)
165  {
166      /* Get sysTick value and check if it's time to execute the task */
167      msTick = HAL_GetTick();
168      if(msTick % DATA_PERIOD_MS == 0 && msTickPrev != msTick)
169      {
170          msTickPrev = msTick;
171          if(SendOverUSB)
172          {
173              BSP_LED_On(LED1);
174          }
175          #ifndef NOT_DEBUGGING
176          else if (SD_Log_Enabled)
177          {
178              BSP_LED_On(LED2);
179          }
180          #endif
181          RTC_Handler( &RtcHandle );

```

These lines will help you find the line of interest to us

Look for this line

Figure 3: Finding where output data rate is being used.

Explanation:

`msTick = HAL_GetTick();` // reads the time counter on the SensorTile board.

`HAL_GetTick()` is useful because it returns how many milliseconds have elapsed since the SensorTile started recording data.



```
msTick % DATA_PERIOD_MS
```

This C code mathematical operation is the modulo divide. Here, **msTick** is increasing at each millisecond. When **msTick** is evaluated in this line of execution and when **msTick** acquires a value that is an exact multiple of **DATA_PERIOD_MS**, the modulo operation will return 0. For more information regarding modulo arithmetic, please open the following weblink.

https://en.wikipedia.org/wiki/Modulo_operation

```
msTickPrev != msTick
```

We only want the SensorTile to perform calculations once every time the **DATA_PERIOD_MS** has been reached. However, the processor may reach the **HAL_GetTick()** function several times before 1ms has elapsed. As such, we must track the previous values of **msTick**. If the value has not changed, no action should be taken.

8. Let's examine the time period over which the SensorTile should be outputting data.

First, click the variable **DATA_PERIOD_MS**.

Once the variable is highlighted, right click it.

Once the drop-down menu appears, click "Open Declaration" as shown in Figure 4.

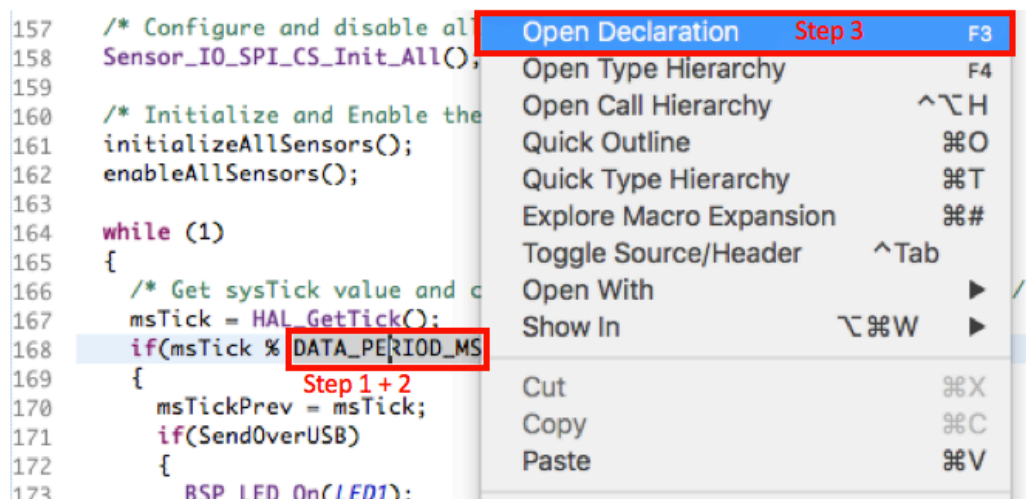


Figure 4: Opening a variable declaration.

Note: this can be done with *any* variable or *any* function. This is a crucial feature of the System WorkBench IDE, as it enables developers to quickly find and inspect functions and variables. These functions and variables could be otherwise impossible to find as large projects contain multiple files and folders where the functions and variables are



being implemented or initialized.

9. Your screen should now advance to another line of code in the same file shown in Figure 5.

```
56 /* Data acquisition per
57 #define DATA_PERIOD_MS
58 // #define NOT_DEBUGGING
59
60 /* Private macro -----
61 /* Private variables --
```

Figure 5: Jumping to the relevant line in the main.c file.

10. Read the definition of DATA_PERIOD_MS. Does it match the value calculated in step 4 of section 3.1 Examining the Sensor Sampling Rate and Output Data Rate? If there is a difference, can you explain the difference?

3.2. Modifying the USB Data Output Rate

1. Terminate and remove all previous applications from the SensorTile. Disconnect and reconnect the SensorTile from your personal computer.
2. Change the variable of **DATA_PERIOD_MS** to be 1000. See Figure 6.

```
55
56 /* Data acquisition period [ms] */
57 #define DATA_PERIOD_MS (1000)
58 // #define NOT_DEBUGGING
59
```

Figure 6: Updating the DATA_PERIOD_MS variable.

3. Save the changes you made to **main.c** and build the project.
4. Run the project on the SensorTile in debug mode.
5. Examine the messages received by your personal computer over the serial connection as instructed in previous lab.
6. Examine the messages regarding "TimeStamp". These are highlighted in Figure 2.



7. Calculate the difference between 5 successive timestamps (i.e. $t_1 = T_2 - T_1$, $t_2 = T_3 - T_2$, ..., $t_5 = T_5 - T_4$). Average these differences $(t_1 + t_2 + t_3 + t_4 + t_5)/5$. What is this average difference? See Figure 2. Does this agree with your expectations?



4. Creating New Data, New Messages, and Event Detection

4.1. Computing Vector Magnitude Acceleration

This section will guide users through the process of adding information regarding computing vector magnitude acceleration from signals obtained from the accelerometer.

The SensorTile accelerometer measures acceleration on each of three orthogonal axes, X, Y, and Z with acceleration values of a_x , a_y , and a_z , respectively.

Vector magnitude acceleration, a_{mag} , is defined as,

$$a_{mag} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

In this section, the steps required to compute and the transmit vector magnitude acceleration are demonstrated.

1. Terminate and remove all previous applications. Disconnect and reconnect the SensorTile from your personal computer.
2. Open main.c, change the variable of **DATA_PERIOD_MS** to be 100. See Figure 6. (Note: change the value to 100 instead of 1000).
3. Open the declaration for the function **Accelero_Sensor_Handler** the same way you found the declaration for **DATA_PERIOD_MS** in step 8 from section 3.1 Examining the Sensor Sampling Rate and Output Data Rate.

The function is called in main.c. It can be found in the **while (1)** loop we were inspecting earlier. See Figure 7 for more details.

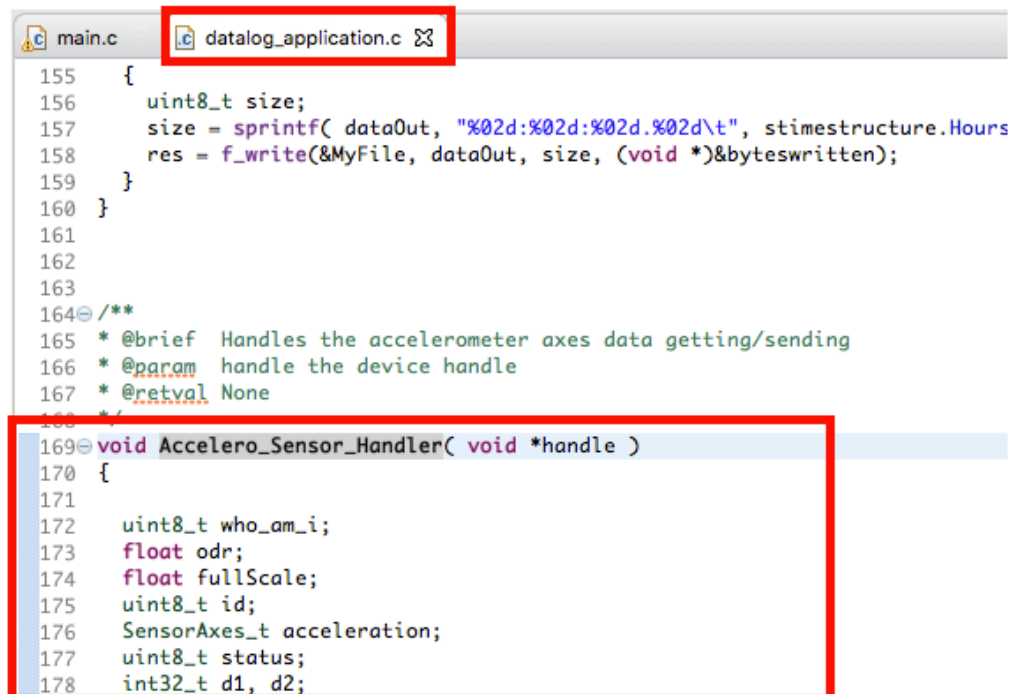


```
while (1)
{
    /* Get sysTick value and check if it's time to execute the task */
    msTick = HAL_GetTick();
    if(msTick % DATA_PERIOD_MS == 0 && msTickPrev != msTick)
    {
        msTickPrev = msTick;
        if(SendOverUSB)
        {
            BSP_LED_On(LED1);
        }
#ifdef NOT_DEBUGGING
    else if (SD_Log_Enabled)
    {
        BSP_LED_On(LEDSDW);
    }
#endif
    RTC_Handler( &RtcHandle );

    Accelero_Sensor_Handler( LSM6DSM_X_0_handle );
}
```

Figure 7: Inspecting the Accelero_Sensor_Handle() function.

4. A new file should appear on your tab, and should be open to the function declaration as seen in Figure 8.



```
main.c | datalog_application.c
155 {
156     uint8_t size;
157     size = sprintf( dataOut, "%02d:%02d:%02d.%02d\t", stimestructure.Hours
158     res = f_write(&MyFile, dataOut, size, (void *)&byteswritten);
159 }
160 }
161
162
163
164 /**
165  * @brief Handles the accelerometer axes data getting/sending
166  * @param handle the device handle
167  * @retval None
168  */
169 void Accelero_Sensor_Handler( void *handle )
170 {
171
172     uint8_t who_am_i;
173     float odr;
174     float fullScale;
175     uint8_t id;
176     SensorAxes_t acceleration;
177     uint8_t status;
178     int32_t d1, d2;
```

Figure 8: Definition of the Accelero_Sensor_Handle() function.



5. Scroll down until you find the **sprintf** function as highlighted in Figure 9.

```
BSP_ACCELER0_Get_Instance( handle, &id );
BSP_ACCELER0_IsInitialized( handle, &status );
old_verbose = verbose;
if ( status == 1 )
{
    if ( BSP_ACCELER0_Get_Axes( handle, &acceleration ) == COMPONENT_ERROR )
    {
        acceleration.AXIS_X = 0;
        acceleration.AXIS_Y = 0;
        acceleration.AXIS_Z = 0;
    }

    if(SendOverUSB) /* Write data on the USB */
    {
        sprintf( dataOut, "\n\rACC_X: %d, ACC_Y: %d, ACC_Z: %d", (int)acceleration.AXIS_X, (int)acceleration.AXIS_Y, (int)acceleration.AXIS_Z );
        CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
    }
}
```

Figure 9: Calling `sprintf()` and `CDC_Fill_Buffer()` to send data over USB serial.

For more information regarding **sprintf** please open the following weblink on a browser on your personal computer.

<https://linux.die.net/man/3/sprintf>

6. We are going to add some information to the end of the Accelerometer output to inform the user of the magnitude of the acceleration vector. To do this we must modify the code such that it matches the screenshot in Figure 10.

```
if ( status == 1 )
{
    if ( BSP_ACCELER0_Get_Axes( handle, &acceleration ) == COMPONENT_ERROR )
    {
        acceleration.AXIS_X = 0;
        acceleration.AXIS_Y = 0;
        acceleration.AXIS_Z = 0;
    }
}

if(SendOverUSB) /* Write data on the USB */
{
    uint32_t abs_acc;
    abs_acc = ((int)acceleration.AXIS_X * (int)acceleration.AXIS_X);
    abs_acc += ((int)acceleration.AXIS_Y * (int)acceleration.AXIS_Y);
    abs_acc += ((int)acceleration.AXIS_Z * (int)acceleration.AXIS_Z);
    abs_acc = sqrt((float) abs_acc);

    sprintf( dataOut, "\n\rACC_X: %d, ACC_Y: %d, ACC_Z: %d, |ACCI: %d",
        (int)acceleration.AXIS_X,
        (int)acceleration.AXIS_Y,
        (int)acceleration.AXIS_Z,
        (int) abs_acc
    );
    CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
}
```

Figure 10: Computing and displaying the magnitude of the acceleration vector.

7. Save the changes you made and build the project.



8. Run the project on the SensorTile in debug mode.
9. Examine the messages received by your personal computer over the serial connection as instructed in previous lab.
10. Take a screen shot of the messages you receive over the USB Serial port for your record.



4.2. Detecting Events: Acceleration Vector Magnitude Threshold Crossing

This section will demonstrate how sensor signals may be monitored and events may be detected. In addition, this introduces a method for adding a message to indicate that the acceleration vector magnitude has exceeded a certain threshold value.

1. Terminate and remove all previous applications from the SensorTile. Disconnect and reconnect the SensorTile from your personal computer.
2. Open the declaration for the function **Accelero_Sensor_Handler** the same way you found the declaration for **DATA_PERIOD_MS** in step 3 from section 4.1 Computing Vector Magnitude Acceleration.
3. Modify the function such that it matches Figure 11.

```
if(SendOverUSB) /* Write data on the USB */
{
    uint32_t abs_acc;
    abs_acc = ((int)acceleration.AXIS_X * (int)acceleration.AXIS_X);
    abs_acc += ((int)acceleration.AXIS_Y * (int)acceleration.AXIS_Y);
    abs_acc += ((int)acceleration.AXIS_Z * (int)acceleration.AXIS_Z);
    abs_acc = sqrt((float) abs_acc);

    sprintf( dataOut, "\n\rACC_X: %d, ACC_Y: %d, ACC_Z: %d, |ACCL: %d",
        (int)acceleration.AXIS_X,
        (int)acceleration.AXIS_Y,
        (int)acceleration.AXIS_Z,
        (int) abs_acc
    );
    CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));

    if (abs_acc > 1550)
    {
        sprintf( dataOut, "\n\r\t\t\tAcceleration Vector Magnitude > 1.5g!");
        CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
    }
}
```

Figure 11: Adding acceleration threshold crossing detection.

4. Save the changes you made and build the project.
5. Run the project on the SensorTile in debug mode.
6. Examine the messages received by your personal computer over the serial connection.



7. Shake the sensor gently enough to make sure you don't disconnect any of the cables,

but strongly enough to ensure the new message appears.

Take a screenshot this message appearing over the USB Serial port.



5. Modifying Data Acquisition Parameters

This section demonstrates methods for modification of sensor data acquisition parameters. In particular, this focuses on methods for adjusting sensor measurement range. The SensorTile microaccelerometer Full-Scale measurement range (FS) is adjusted.

5.1. Adjust Scaling Factor (Full-Scale FS)

1. Terminate and remove all previous applications from the SensorTile. Disconnect and reconnect the SensorTile from your personal computer.
2. Open the declaration for the function **Accelero_Sensor_Handler** the same way you found the declaration for **DATA_PERIOD_MS** in step 3 from section 4.1 Computing Vector Magnitude Acceleration.
3. Modify the function such that it matches Figure 12.

```
if(SendOverUSB) /* Write data on the USB */
{
    uint32_t abs_acc;
    abs_acc = ((int)acceleration.AXIS_X * (int)acceleration.AXIS_X);
    abs_acc += ((int)acceleration.AXIS_Y * (int)acceleration.AXIS_Y);
    abs_acc += ((int)acceleration.AXIS_Z * (int)acceleration.AXIS_Z);
    abs_acc = sqrt((float) abs_acc);

    sprintf( dataOut, "\n\rACC_X: %d, ACC_Y: %d, ACC_Z: %d, |ACCL: %d",
            (int)acceleration.AXIS_X,
            (int)acceleration.AXIS_Y,
            (int)acceleration.AXIS_Z,
            (int) abs_acc
            );
    CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));

    if (abs_acc > 3550) Old value: 1550. New value: 3550.
    {
        sprintf( dataOut, "\n\r\t\t\t\tAcceleration Vector Magnitude > 3.5g!");
        CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
    }
}
```

Figure 12: Modifying acceleration threshold crossing detection.

4. Save the changes you made and build the project.
5. Run the project on the SensorTile in debug mode.
6. Examine the messages received by your personal computer over the serial connection.



Example.

7. To demonstrate operation, shake the sensor (taking care not to disconnect cables).

Notice how the message no longer appears as a result of this change in Full-Scale range.

This is to be expected. Notice that the absolute value of the acceleration for each of the three axes (ACC_X, ACC_Y, ACC_Z) never exceed 2000.

If we calculate the maximum vector magnitude, we should expect the following.

$$|A_{max}| = \left| \begin{pmatrix} 2000 \\ 2000 \\ 2000 \end{pmatrix} \right| \cong 3464$$

As such, if we wish to see the magnitude exceed a value of 3.5g, we would need to adjust the parameters of the sensor such that it can record values greater than 2000 for each axis.

8. Terminate and remove all previous applications from the SensorTile. Disconnect and reconnect the SensorTile from your personal computer.
9. Open **main.c** and find the **while (1)** loop. Modify the file directly before the **while (1)** loop to match Figure 13. We make the modification **outside** of the while loop because we do not want to be constantly adjusting the Full-Scale range of the values the sensor returns. We only want to change it once.

```

/* Initialize and Enable the available sensors */
initializeAllSensors();
enableAllSensors();

BSP_ACCELER0_Set_FS_Value( LSM6DSM_X_0_handle , 4.0f);

while (1)
{
    /* Get sysTick value and check if it's time to execute the task */
    msTick = HAL_GetTick();
    if(msTick % DATA_PERIOD_MS == 0 && msTickPrev != msTick)
    {
        msTickPrev = msTick;
        if(SendOverUSB)
        {
            BSP_LED_On(LED1);
        }
    }
    #ifndef NOT_DEBUGGING
    else if (SD_Log_Enabled)
    {
        BSP_LED_On(LEDSD);
    }
    #endif
    RTC_Handler( &RtcHandle );

    Accelero_Sensor_Handler( LSM6DSM_X_0_handle );
}

```

Figure 13: Adjusting the full scale (FS) range of each axis of data read from the accelerometer.



10. Save the changes you made and build the project.
11. Run the project on the SensorTile in debug mode.
12. Examine the messages received by your personal computer over the serial connection.
13. Shake the sensor gently enough to make sure you don't disconnect any of the cables, but strongly enough to ensure the new message appears.

Take a screenshot this message appearing over the USB Serial port.



5.2. Making Similar Modifications for Other Parameters

The tables below summarize some of the important functions available to adjust the data acquisition parameters for the Accelerometer and Gyroscope sensors.

Accelerometer (acceleration):

Example Function Call	Function Use	List of Valid Second Arguments	Unit
<code>BSP_ACCELER0_Set_FS_Value(LSM6DSM_X_0_handle, 4.0f)</code>	Set the range of the axes	2.0f, 4.0f, 8.0f, 16.0f	g (x 9.81 ms ⁻²)
<code>BSP_ACCELER0_Set_ODR_Value(LSM6DSM_X_0_handle, 13.0f)</code>	Sets output data rate of sensor. This is different from the rate at which data is reported over serial USB.	13.0f, 26.0f, 52.0f, 104.0f, 208.0f, 416.0f, 833.0f, 1660.0f, 3330.0f 6660.0f	Hz

Gyroscope (angular velocity):

Example Function Call	Function Use	List of Valid Second Arguments	Unit
<code>BSP_GYRO_Set_FS_Value(LSM6DSM_G_0_handle, 245.0f)</code>	Set the range of the axes	245.0f, 500.0f, 1000.0f 2000.0f	Degrees per second
<code>BSP_GYRO_Set_ODR_Value(LSM6DSM_G_0_handle, 4.0f)</code>	Sets output data rate of sensor. This is different from the rate at which data is reported over serial USB.	13.0f, 26.0f, 52.0f, 104.0f, 208.0f, 416.0f, 833.0f, 1660.0f, 3330.0f 6660.0f	Hz



6. Introduction to the Gyroscope

Complete the following tasks in order to develop a deeper understanding of the Gyroscope systems available on the SensorTile board.

1. Open the declaration for the function **Gyro_Sensor_Handler**. The function is called in main.c. It can be found in the **while (1)** loop we were inspecting earlier. It can be found directly after the call to **Accelero_Sensor_Handler** as seen in Figure 7.
2. Modify this function such that it now computes the vector magnitude computation and transmits this computed value over the serial USB connection. This should be similar to the procedure followed to generate Figure 10. The key data structure we need to inspect here is **angular_velocity** instead of **acceleration**.

Terminate and remove all previous programs from the SensorTile board.

Save the changes you made and build the project.

Run the project on the SensorTile in debug mode.

Examine the messages received by your personal computer over the serial connection.

Take a screenshot these messages appearing over the USB Serial port.

Take a screenshot of the largest gyroscope angular velocity on a single axis (i.e. not the vector magnitude) when shaking the sensor.

3. Modify the system to send a message over the serial USB connection when the vector magnitude of the angular velocity exceeds a threshold value of **40,000** (this corresponds to an angular velocity of 40 degrees per second). This should be similar to the procedure followed to generate Figure 11.

Terminate and remove all previous programs from the SensorTile board.

Save the changes you made and build the project.

Run the project on the SensorTile in debug mode.

Examine the messages received by your personal computer over the serial connection.

Shake the sensor gently enough to make sure you don't disconnect any of the cables, but strongly enough to ensure the new message appears.



Take a screenshot this message appearing over the USB Serial port.

4. Modify the system so that the Full-Scale range of the Gyroscope sensor is 245.0 dps. This should be similar to the procedure followed to generate Figure 13. Refer to section 5.2 Making Similar Modifications for Other Parameters for more information on what function to use to make this modification.

Terminate and remove all previous programs from the SensorTile board.

Save the changes you made and build the project.

Run the project on the SensorTile in debug mode.

Take a screenshot of the largest gyroscope angular velocity on a single axis (i.e. not the vector magnitude) when shaking the sensor.