STMicroelectronics SensorTile: Accelerometer Sensor Systems and Orientation and Event Detection

Table of Contents

1. Introduction to This Tutorial

The lab steps provide:

1. A review to accelerometer sensor systems and methods for detection of sensor orientation by exploiting gravitational acceleration signals.
2. Measurement of orientation by the polar coordinate system.
3. An introduction to gesture recognition through the use of sensor information and state machine systems for characterizing specific behavior.

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.

1.2. Prerequisite

It is recommended that users have completed and are familiar with the contents of the following tutorials before proceeding.

1. Introduction to SensorTile and the System WorkBench Integrated Development Environment (IDE)
2. Sensor System Signal Acquisition, Event Detection and Configuration on the NXP freedom board from CEE-345 Microprocessor System Design class.
3. Some background on Cartesian and Polar coordinate systems.

2. Cartesian to Polar Coordinate Conversion in Multiaxis Sensing

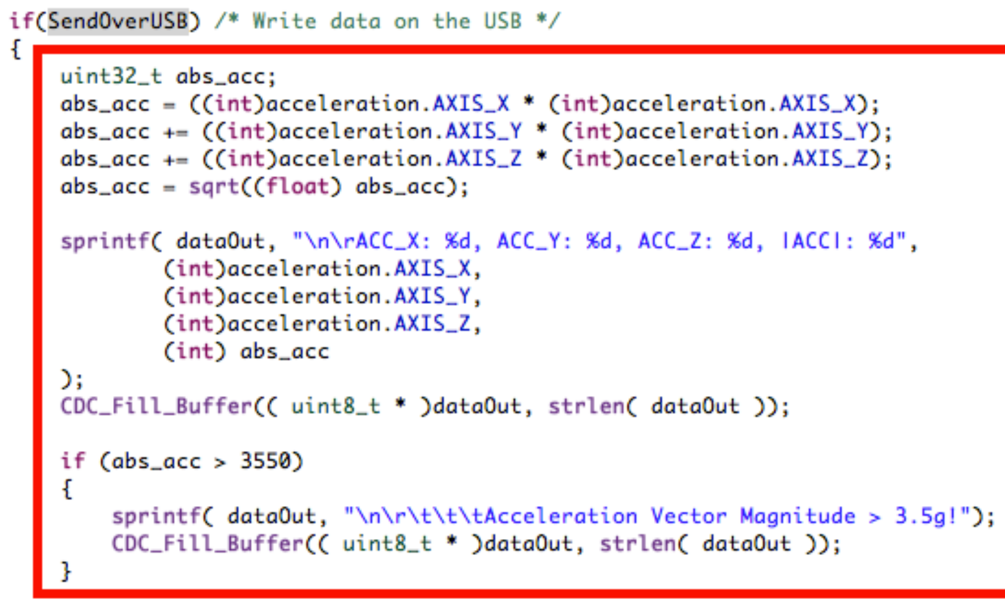
In the last lab, we explored how to build the DataLog program, download and run the program on sensor tile with Putty program. We will now examine the code and how to make modifications, and understand why these modifications are useful building blocks for systems that aim to perform gesture recognition.

1. Open the IDE (Eclipse or System WorkBench) on your personal computer ***and open the DataLog program.***

   Select the same workspace as in previous lab.

2. Once the IDE is open, open the declaration for the function labelled **Accelero_Sensor_Handler.**

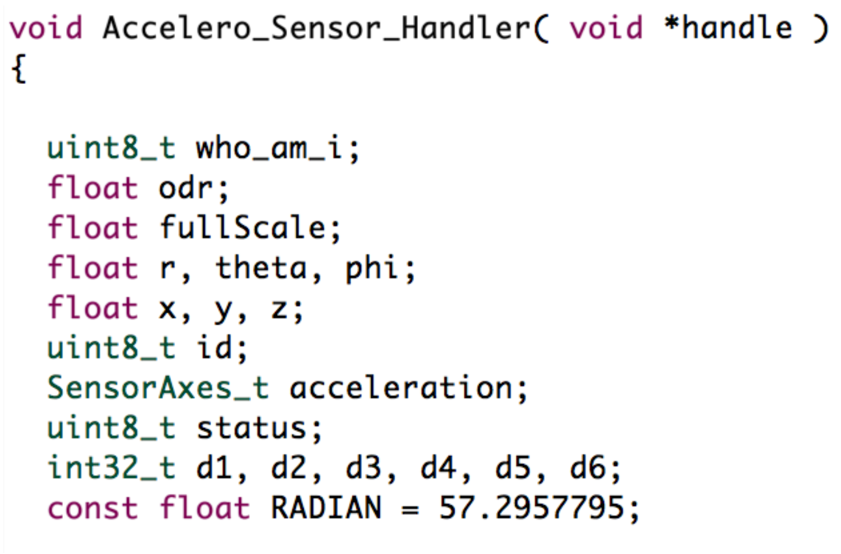
3. Modify the function such that it no longer contains the modifications we made in Tutorial 2. See Figure 1.



```
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, |ACC|: %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)
    {
        sprintf( dataOut, "\n\r\t\t\tAcceleration Vector Magnitude > 3.5g!");
        CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
    }
```

Figure 1: Delete the lines found enclosed in the red box.

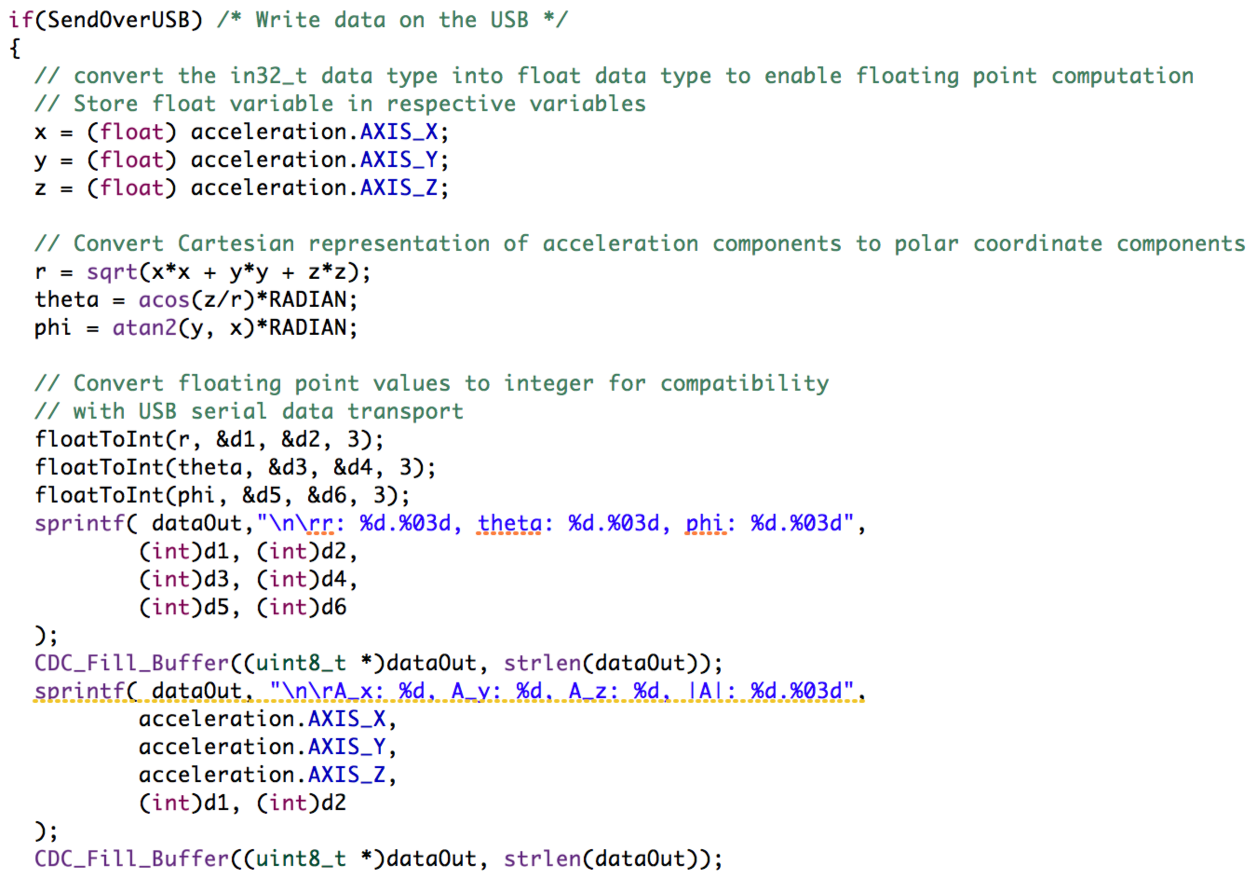
4. Modify the variable declaration section of the **Accelero_Sensor_Handler** function such that it matches Figure 2.



```
void Accelero_Sensor_Handler( void *handle )
{

  uint8_t who_am_i;
  float odr;
  float fullScale;
  float r, theta, phi;
  float x, y, z;
  uint8_t id;
  SensorAxes_t acceleration;
  uint8_t status;
  int32_t d1, d2, d3, d4, d5, d6;
  const float RADIAN = 57.2957795;
```

Figure 2: Ensuring the correct variables are declared.

5. Note that in the computation of angles using trigonometric function values the multiplying factor of RADIAN = 180 degree/π radians defined above is required.
6. Add the lines of code found in Figure 3 to the **Accelero_Sensor_Handler** function.



```
if(SendOverUSB) /* Write data on the USB */
{
  // convert the in32_t data type into float data type to enable floating point computation
  // Store float variable in respective variables
  x = (float) acceleration.AXIS_X;
  y = (float) acceleration.AXIS_Y;
  z = (float) acceleration.AXIS_Z;

  // Convert Cartesian representation of acceleration components to polar coordinate components
  r = sqrt(x*x + y*y + z*z);
  theta = acos(z/r)*RADIAN;
  phi = atan2(y, x)*RADIAN;

  // Convert floating point values to integer for compatibility
  // with USB serial data transport
  floatToInt(r, &d1, &d2, 3);
  floatToInt(theta, &d3, &d4, 3);
  floatToInt(phi, &d5, &d6, 3);
  sprintf( dataOut,"\n\rr: %d.%03d, theta: %d.%03d, phi: %d.%03d",
          (int)d1, (int)d2,
          (int)d3, (int)d4,
          (int)d5, (int)d6
  );
  CDC_Fill_Buffer((uint8_t *)dataOut, strlen(dataOut));
  sprintf( dataOut, "\n\rA_x: %d, A_y: %d, A_z: %d, |A|: %d.%03d",
          acceleration.AXIS_X,
          acceleration.AXIS_Y,
          acceleration.AXIS_Z,
          (int)d1, (int)d2
  );
  CDC_Fill_Buffer((uint8_t *)dataOut, strlen(dataOut));
```

Figure 3: Adding polar coordinate conversion to SensorTile system.

7. Terminate and remove all previous applications from the SensorTile board.

8. Compile and run the DataLog application on the SensorTile board in debug mode.

9. Examine the data transmitted over the Serial USB connection to your personal computer. Take a screenshot.

10. Orient the SensorTile, such that the value for A_z is approximately +1000, and the two other axes (A_x, A_y) are as close to 0 as possible. Record the values of r, phi and theta.

11. Orient the SensorTile upside down such that the value for A_z is approximately -1000, and the two other axes (A_x, A_y) are as close to 0 as possible. Record the values of r, phi and theta.

12. Repeat steps 9 and 10 until you can fill in the following table.

	R (magnitude)	Phi (azimuth)	Theta (inclination)
$A_z \sim +1000, A_x \sim 0, A_y \sim 0$			
$A_z \sim -1000, A_x \sim 0, A_y \sim 0$			
$A_Y \sim +1000, A_x \sim 0, A_Z \sim 0$			
$A_Y \sim -1000, A_x \sim 0, A_Z \sim 0$			
$A_X \sim +1000, A_Z \sim 0, A_y \sim 0$			
$A_X \sim -1000, A_Z \sim 0, A_y \sim 0$			

3. Demonstration of Gesture Recognition Systems

3.1. Introduction

This section will guide users through the process of modifying the **Accelero_Sensor_Handler** function such that it will be able to recognize a simple gesture. The simple gesture is defined as follows.

1. Place the sensor such that

   The z-axis acceleration, **A_z < -z_thresh** (specifically, the z-component of the acceleration vector is below a certain threshold value) for at least a time, **tau** milliseconds.

   In terms of physical orientation, this corresponds to the sensor being "upside down" for at least "tau" milliseconds.

2. Within the next time, **tau** milliseconds, the sensor must be reoriented such that

   The z-axis acceleration, **A_z > +z_thresh** (Specifically, the z-component of the acceleration vector is above a certain threshold value) for at least **tau** milliseconds.

   In terms of physical orientation, this corresponds to the sensor being "right side up" for at least "tau" milliseconds.

We will model this as a simple state machine, which can be seen in Figure 4 below.

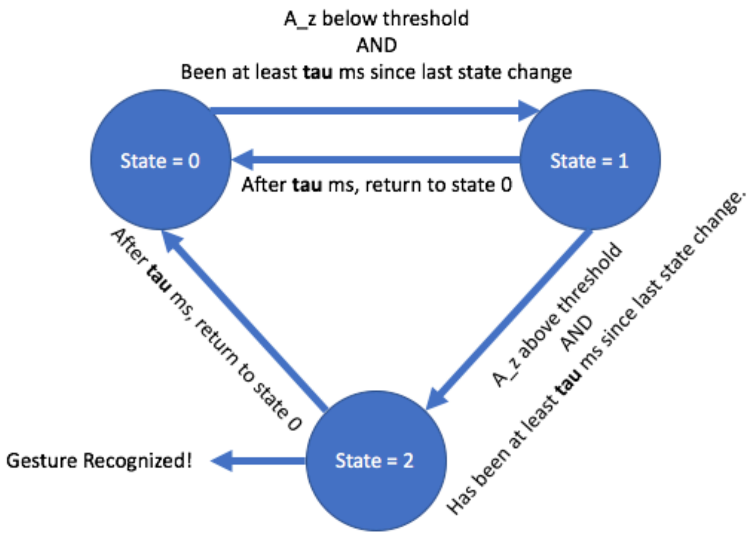

A_z below threshold
AND
Been at least tau ms since last state change
State = 0
State = 1
After tau ms, return to state 0
After tau ms, return to state 0
A_z above threshold
AND
Has been at least tau ms since last state change.
State = 2
Gesture Recognized!

Figure 4: State machine for gesture recognition.

3.2. Modifications to main.c

Make the following modifications to main.c. We are making these modifications so that the **Accelero_Sensor_Handler** function can have access to variables that will help track the previous state and timing information.

1. Declare and initialize the variable **msTickStateChange** as a **uint32_t** to have a value of **0**.

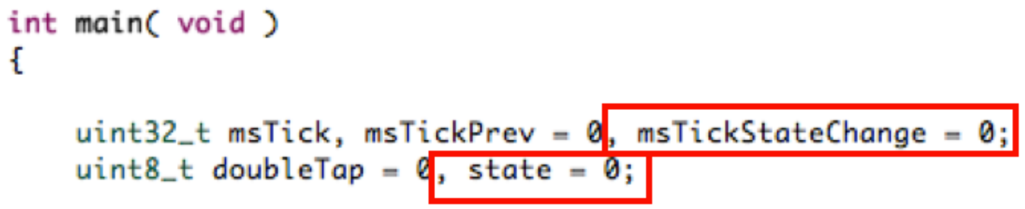
   This variable will be used to track the time at which the state was changed.

   See Figure 5.

2. Declare and initialize the variable **state** as a **uint8_t** to have a value of **0**.

   This variable will be used to track the current state of the system.

   See Figure 5.

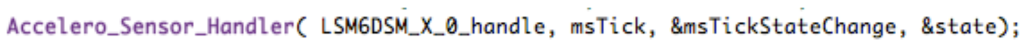


```
int main( void )
{

    uint32_t msTick, msTickPrev = 0, msTickStateChange = 0;
    uint8_t doubleTap = 0, state = 0;
```

Figure 5: Adding the variables to track the time at which the system last observed a change of state and the state variable itself.

3. Pass these variables to the function **Accelero_Sensor_Handler** as shown in Figure 6.

   Notice: there is an ampersand ('&') in front of the variables **msTickStateChange** and **state**. This is because we want the function **Accelero_Sensor_Handler** to modify the values of these variables. In order to modify them, the functions must accept these parameters **by reference**.

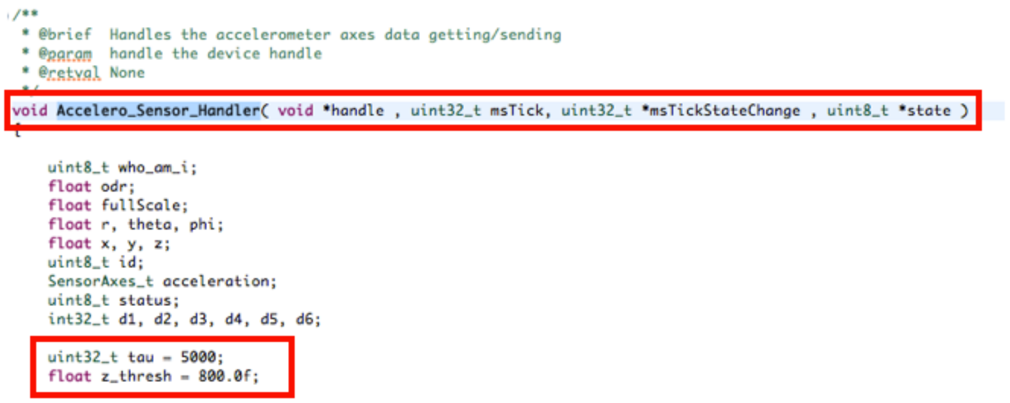


```
Accelero_Sensor_Handler( LSM6DSM_X_0_handle, msTick, &msTickStateChange, &state);
```

Figure 6: Modification to calling the function Accelero_Sensor_Handler.

4. Modify the declaration of **Accelero_Sensor_Handler** so that the project can compile successfully. See Figure 7 for more details.

5. Add the variables **tau** and **z_thresh** as seen in Figure 7.

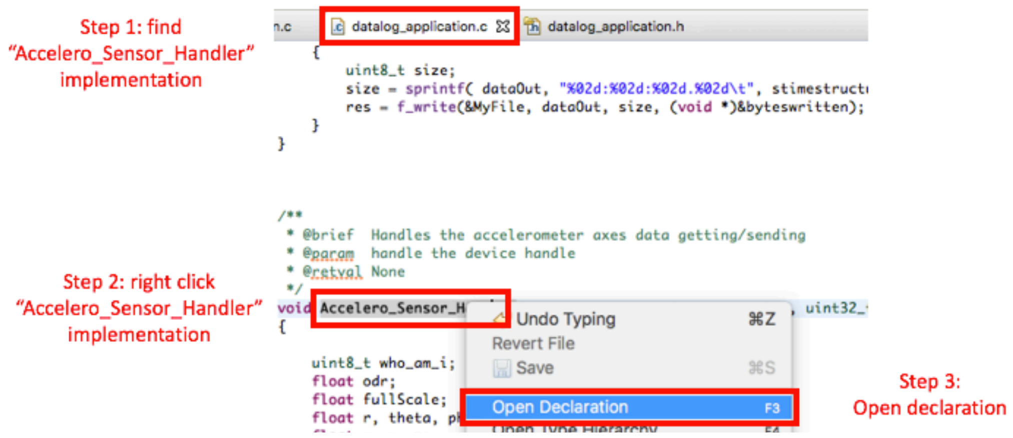


```
/**
 * @brief  Handles the accelerometer axes data getting/sending
 * @param  handle the device handle
 * @retval None
 */
void Accelero_Sensor_Handler( void *handle , uint32_t msTick, uint32_t *msTickStateChange , uint8_t *state )
{

    uint8_t who_am_i;
    float odr;
    float fullScale;
    float r, theta, phi;
    float x, y, z;
    uint8_t id;
    SensorAxes_t acceleration;
    uint8_t status;
    int32_t d1, d2, d3, d4, d5, d6;

    uint32_t tau = 5000;
    float z_thresh = 800.0f;
```

Figure 7: Modifying Accelero_Sensor_Handler implementation to enable access to state selection variables. Notice how we use the '' character here instead of '&'. Your instructor will provide additional guidance on what these symbols mean.*

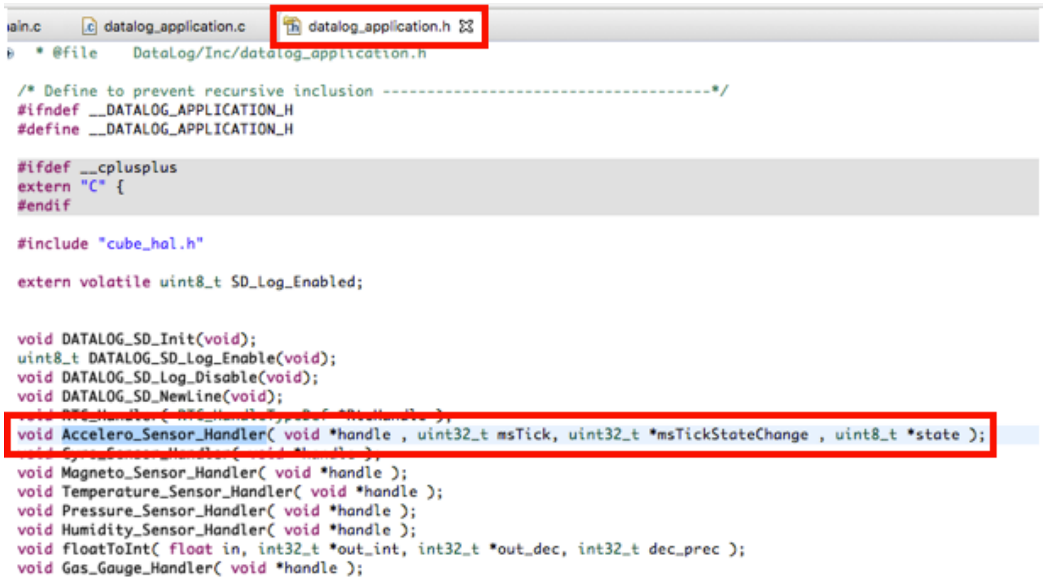
6. This is the **implementation** of the function **Accelero_Sensor_Handler**. The function is currently defined in another file labelled **"datalog_application.h"**. We need to modify this declaration to match the implementation of the function such that the project can compile.

   To find this file, first right-click the function name **"Accelero_Sensor_Handler"** as seen in Figure 8. Then click "Open Declaration". This should open the file labelled **"datalog_application.h"**. Modify the relevant line to match Figure 9.



```
    {
        uint8_t size;
        size = sprintf( dataOut, "%02d:%02d:%02d.%02d\t", stimestructu
        res = f_write(&MyFile, dataOut, size, (void *)&byteswritten);
    }
}


/**
 * @brief  Handles the accelerometer axes data getting/sending
 * @param  handle the device handle
 * @retval None
 */
void Accelero_Sensor_H                                    , uint32_
{

    uint8_t who_am_i;
    float odr;
    float fullScale;
    float r, theta, ph
```

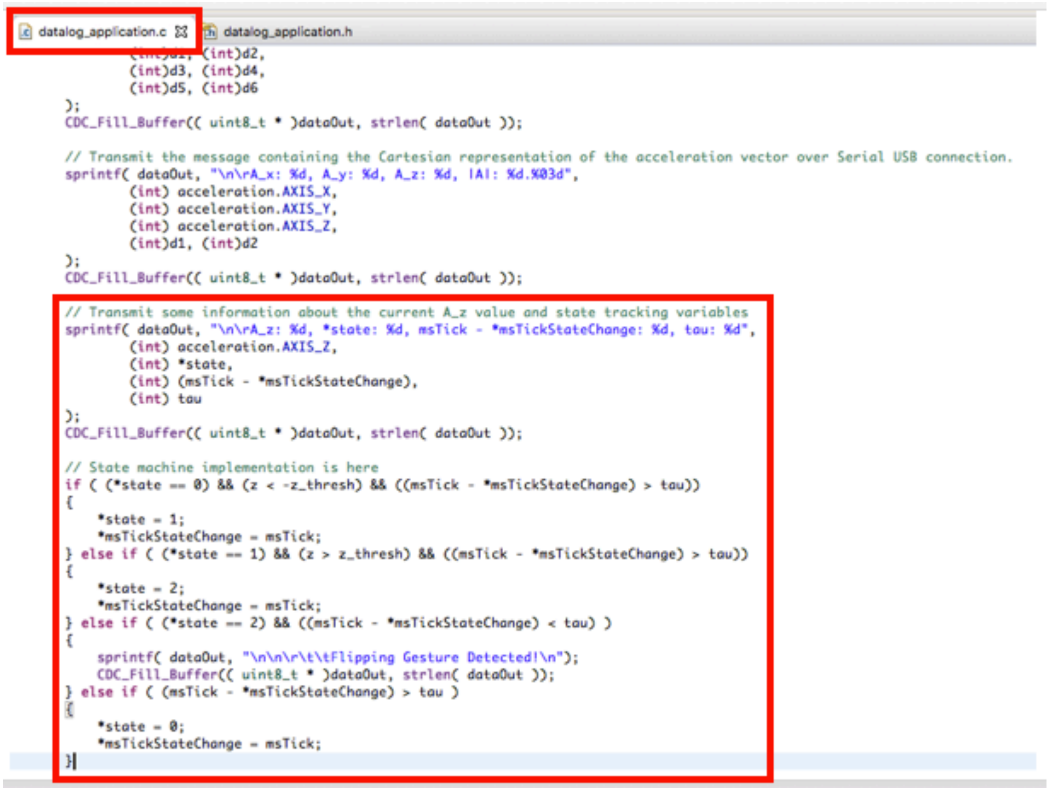
Figure 8: Open the declaration (not implementation) of Accelero_Sensor_Handler.



```
 * @file    DataLog/Inc/datalog_application.h

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __DATALOG_APPLICATION_H
#define __DATALOG_APPLICATION_H

#ifdef __cplusplus
extern "C" {
#endif

#include "cube_hal.h"

extern volatile uint8_t SD_Log_Enabled;


void DATALOG_SD_Init(void);
uint8_t DATALOG_SD_Log_Enable(void);
void DATALOG_SD_Log_Disable(void);
void DATALOG_SD_NewLine(void);
void RTC_Handler( RTC_HandleTypeDef *RtcHandle );
void Accelero_Sensor_Handler( void *handle , uint32_t msTick, uint32_t *msTickStateChange , uint8_t *state );
void Gyro_Sensor_Handler( void *handle );
void Magneto_Sensor_Handler( void *handle );
void Temperature_Sensor_Handler( void *handle );
void Pressure_Sensor_Handler( void *handle );
void Humidity_Sensor_Handler( void *handle );
void floatToInt( float in, int32_t *out_int, int32_t *out_dec, int32_t dec_prec );
void Gas_Gauge_Handler( void *handle );
```

Figure 9: Modify the function declaration to match our implementation.

7. Switch back to the tab labelled **"datalog_application.c"** and add the lines highlighted in the red box in Figure 10 to the file.



```
        (int)d1, (int)d2,
        (int)d3, (int)d4,
        (int)d5, (int)d6
);
CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));

// Transmit the message containing the Cartesian representation of the acceleration vector over Serial USB connection.
sprintf( dataOut, "\n\rA_x: %d, A_y: %d, A_z: %d, |A|: %d.%03d",
        (int) acceleration.AXIS_X,
        (int) acceleration.AXIS_Y,
        (int) acceleration.AXIS_Z,
        (int)d1, (int)d2
);
CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));

// Transmit some information about the current A_z value and state tracking variables
sprintf( dataOut, "\n\rA_z: %d, *state: %d, msTick - *msTickStateChange: %d, tau: %d",
        (int) acceleration.AXIS_Z,
        (int) *state,
        (int) (msTick - *msTickStateChange),
        (int) tau
);
CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));

// State machine implementation is here
if ( (*state == 0) && (z < -z_thresh) && ((msTick - *msTickStateChange) > tau))
{
    *state = 1;
    *msTickStateChange = msTick;
} else if ( (*state == 1) && (z > z_thresh) && ((msTick - *msTickStateChange) > tau))
{
    *state = 2;
    *msTickStateChange = msTick;
} else if ( (*state == 2) && ((msTick - *msTickStateChange) < tau) )
{
    sprintf( dataOut, "\n\n\r\t\tFlipping Gesture Detected!\n");
    CDC_Fill_Buffer(( uint8_t * )dataOut, strlen( dataOut ));
} else if ( (msTick - *msTickStateChange) > tau )
{
    *state = 0;
    *msTickStateChange = msTick;
}
```

Figure 10: Adding code to perform simple gesture detection.

8. Terminate and remove all previous applications from the SensorTile board.

9. Build the project. Solve any errors that you may see. Be very careful in matching the code exactly according to the figures above.

10. Run the application in debug mode on the SensorTile device. Inspect the data transmitted by the SensorTile device to your personal computer.

11. Perform the steps required from the state machine in Figure 4 to make the “Flipping Gesture Detected!” appear on your personal computer. Take a screenshot of this message appearing.