ENGD2103 EMBEDDED SYSTEMS FUNDAMENTALS
Laboratory guidance notes
Week 4: Formative Assignment – Part 3 – Accelerometer

Author: M A Oliver, J A Gow, T Allidina *v1.0*

Table of Contents

Version record

Version	Notes	Date	By
1.0	Initial version	21/10/22	MAO

PLEASE READ THIS CAREFULLY

This is the fourth of a series of guidance documents to get you started with your Embedded Systems Fundamentals projects. They are aligned with the key aspects of the formative coursework specification, and thus this document should be read in conjunction with the coursework specification document. These documents will start off with some fairly close guidance - however as you develop your skills the notes will provide less and less detailed guidance and become more of an overview.

1 Aims and objectives

During the formative assignment, you will be writing three short pieces of code. This document outlines the third part: an orientation detector using an accelerometer.

Prerequisite: you should have constructed the breadboard layout and completed the previous two tasks before commencing this task.

2 Overview

The purpose of this task is to give you experience of:-

- A 'lightweight' introduction to I^{2}C interfacing using the 'Wire' library on the Arduino platform
- Driving and interrogating an I^{2}C device; in this case an accelerometer
- Interpreting the data from the accelerometer

3 Task Requirements

In this task, you will be required to write firmware for the Arduino that samples gravitational acceleration, every 333ms, along x-, y- and z-axes.

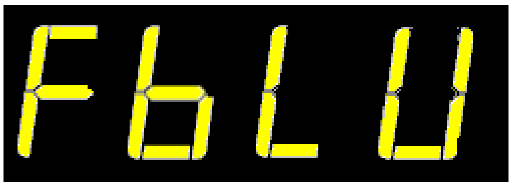
From the x, y and z values of gravitational acceleration, your firmware will be required to determine the orientation of the accelerometer.

Firstly, you should design your firmware such that it writes the x-, y- and z-values of gravitational acceleration to the serial port. Using the outputted values, select threshold values for determining the orientation. The choice of threshold values should be documented in your code. **Hint:** look for the dominant component.

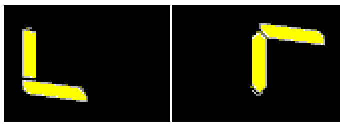
The results of the orientation need to be displayed on the 7-segment LED display introduced in the previous exercise. The LED display should display:-

'F'	when the board is lying flat
'b'	when the board is lying flat, with its base facing upwards
'L'	when the board is held in landscape
'U'	when the board is held upside-down in landscape
'l'	when the board is held in portrait (to the left)
'r'	when the board is held in portrait (to the right)

The following image shows how the upper-case characters 'F', 'b', 'L' and 'U' are to appear on the 7-segment display.



The following image shows how the lower-case characters 'l' and 'r' are to appear on the 7-segment display.



All these characters should be displayed so they can be read normally according to the current orientation of the breadboard.

If, in a particular iteration, none of the thresholds are exceeded, the previous orientation should be displayed.

4 Using the Serial Port for Debugging

The Arduino has a built-in hardware UART (Universal Asynchronous Receive Transmit). This is used for uploading your firmware to the module; something you will be familiar with. It can also be used as a serial port for debugging purposes.

The hardware UART consists of two digital signal lines:-

Receive (RX)	Located on Arduino Digital Pin 0
Transmit (TX)	Located on Arduino Digital Pin 1

Hence, any discouragement for using these lines as general I/O lines.

This section will not cover the details of serial transmission, but will give a brief introduction, by means of an example, to using the Arduino Serial library for debugging.

Consider the following sketch:-

```
void setup()
{
    int value;

    // Set-up Serial port to run at 115200 Baud
    Serial.begin(115200);

    // Assign a number to 'value'
    value = -12345;

    // Send the contents of 'value' to the serial port
    Serial.print("The variable value is: ");
    Serial.println(value);
    Serial.println("Done!");
}

void loop()
{
    // put your main code here, to run repeatedly:
}
```

Breaking this down, the line:-

```
Serial.begin(115200);
```

initializes the serial port to run at 115200 Baud (bits/second). This is a commonly-used Baud rate. Another commonly-used Baud rate is 9600 Baud, but it is significantly slower.

The line:-

```
value = -12345;
```

simply sets the variable 'value' to -12345.

The line:-

```
Serial.print("The variable value is: ");
```

writes a text string to the serial port.

The line:-

```
    Serial.println(value);
```

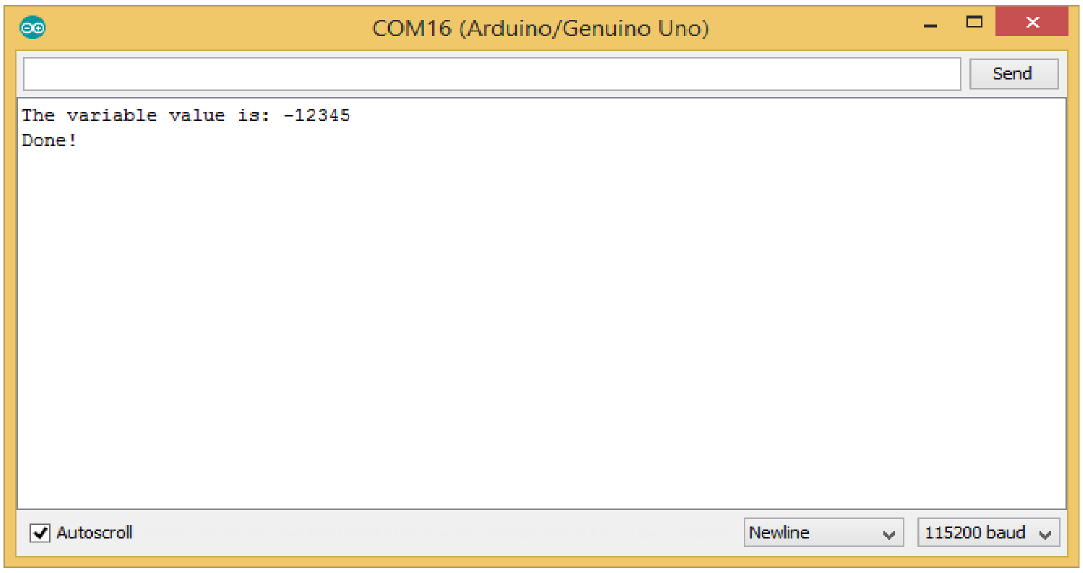
writes the contents of 'value' to the serial port. The difference between `Serial.print()` and `Serial.println()` is that the latter appends a carriage-return and a newline to the serial output.

Unfortunately, `Serial.print()` and `Serial.println()` are not as versatile as `printf()` which you may have encountered if you have had prior experience with C. It is possible to display multiple items with a single `printf()` statement. However it could take multiple `Serial.print()` and `Serial.println()` statements to output the same material via the serial port.

To use the Serial Port for debugging, the Arduino Serial Monitor should be used. In the Arduino IDE, select "Tools" then "Serial Monitor" to invoke the Serial Monitor Interface. At the bottom-right hand corner of the Serial Monitor interface, there is a pull-down menu for setting the Baud rate. This needs to be set to "115200 baud" for this particular example.

On changing the Baud rate, the Arduino will be reset and the program contained within the module will be re-run.

For this example, the following should be observed:-

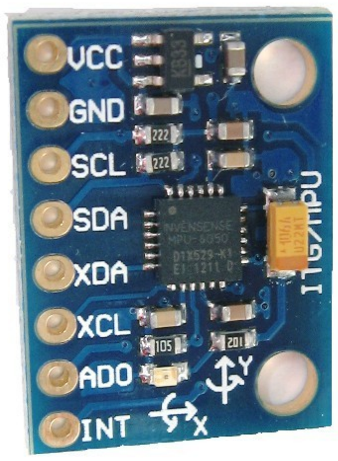


```
The variable value is: -12345
Done!
```

Autoscroll ✓ | Newline | 115200 baud

5 The GY-521 Module

The InvenSense MPU-6050 is an integrated circuit that contains both a 3-axis accelerometer and a 3-axis gyroscope. The MPU-6050 interfaces to a host platform, such as the Arduino, via the I^2C bus.



The datasheet for the MPU-6050 may be found at:-

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

Now the GY-521 is a module containing the MPU-6050, a voltage regulator and other components to facilitate interfacing.

The pin-out is as follows:-

PIN	DESCRIPTION	NOTES
VCC	Positive Supply	Connect to +5V
GND	Ground	Connect to GND
SCL	I^2C Clock	Connect to A5 (SCL) on Arduino Uno
SDA	I^2C Data	Connect to A4 (SDA) on Arduino Uno
XDA	External SDA	Not used in this exercise. (Can be used by the MPU-6050 to communicate with a slave device.)
XCL	External XCL	Not used in this exercise. (Can be used by the MPU-6050 to communicate with a slave device.)
AD0	Address	Not used in this exercise The MPU-6050 has a default I^2C address of 0x68. (Pulling this line high changes the I^2C address to 0x69, allowing two MPU-6050 devices to co-exist on the same I^2C bus.
INT	Interrupt Pin	Not used in this exercise

The MPU-6050 contains a number of registers for configuring and acquiring data. These can be found in the following datasheet.

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf

The table below shows some useful registers. Some of these are of general interest, but will not be needed for this exercise.

ADDRESS	NAME	NOTES
0x3B	ACCEL_XOUT (MSB)	X-component of acceleration (MSB)
0x3C	ACCEL_XOUT (LSB)	X-component of acceleration (LSB)
0x3D	ACCEL_YOUT (MSB)	Y-component of acceleration (MSB)
0x3E	ACCEL_YOUT (LSB)	Y-component of acceleration (LSB)
0x3F	ACCEL_ZOUT (MSB)	Z-component of acceleration (MSB)
0x40	ACCEL_ZOUT (LSB)	Z-component of acceleration (LSB)
0x41	TEMP_OUT (MSB)	Temperature (MSB)
0x42	TEMP_OUT (LSB)	Temperature (LSB)
0x43	GYRO_XOUT (MSB)	Gyroscope – X-component (MSB)
0x44	GYRO_XOUT (LSB)	Gyroscope – X-component (LSB)
0x45	GYRO_YOUT (MSB)	Gyroscope – Y-component (MSB)
0x46	GYRO_YOUT (LSB)	Gyroscope – Y-component (LSB)
0x47	GYRO_ZOUT (MSB)	Gyroscope – Z-component (MSB)
0x48	GYRO_ZOUT (LSB)	Gyroscope – Z-component (LSB)
0x6B	PWR_MGMT_1	Power management register. This has several settings. Writing the value 0x00 to this register will awaken the MPU-6050.

The X-, Y- and Z-components of gravitational acceleration are represented by a 16-bit signed value. The accelerometer defaults to a range of +/-2g.

6 The Wire Library

The following sketch demonstrates how the Wire library can be used to set-up and perform I^{2}C reads; in this case from the accelerometer. There are a few blanks for you to complete before this code will compile and work.

```
#include <Wire.h>

#define MPU_6050          // TODO: Add I2C address of MPU6050
#define PWR_MGMT_1        // TODO: Add address of PWR_MGMT_1 register
#define ACCEL_ZOUT_HI     // TODO: Add ACCEL_ZOUT (MSB) register

void setup()
{
    // Set-up Serial port to run at 115,200 Baud
    Serial.begin(115200);

    // Set-up the I2C for Accelerometer / Gyro
    Wire.begin();

    //Put the MPU6050 IC into the correct operating mode
    Wire.beginTransmission(MPU_6050);
    Wire.write(PWR_MGMT_1);  // Power Management 1 Register
    Wire.write(0);     // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
}

void loop()
{
    int   AccZ;

    // Tell the MPU-6050 that its data registers need to be
    // read, commencing with MSB of the Z-component
    // of acceleration
    Wire.beginTransmission(MPU_6050);
    Wire.write(ACCEL_ZOUT_HI);
    Wire.endTransmission(false);

    // Attempt to read 2 consecutive bytes from the MPU6050.
    // From above, these will be the MSB of the Z-component of
    // acceleration followed by its LSB.
    Wire.requestFrom(MPU_6050, 2 ,true);  // request 2 registers
    if (Wire.available() >= 2)
    {
        // Two 8-bit reads to give 16-bit result
        AccZ = Wire.read() << 8;    // Read and shift MSB
        AccZ |= Wire.read();        // Append the LSB

        Serial.print("Z-component of acceleration = ");
        Serial.println(AccZ);
    }

    delay(1000);
}
```

7 Hardware Abstraction Layer (HAL)

A Hardware Abstraction Layer is beneficial for this task; Arduino-specifics such as calls to the Wire library can be deployed in the HAL.

Consider having two functions in the HAL: one to power the accelerometer (that can be called from `setup()`), and another to read the x-, y- and z-values of gravitational acceleration. You will be able to use these functions in the summative assignment during the second half of the semester.

You can use the HAL rolled-over from Task 2 as a start point. The new functionality can simply be added (rather than starting from scratch) to create an extended HAL.