CSCI 250 Python Computing: Building a Sensor System
TR 12:30-1:45 and 2:00-3:15 – MZ022 – Spring 2018
Lab 4: Acceleration
Wendy Fisher

Corresponding Learning Outcome:

- Develop and run basic Python functions and programs in the Linux environment to collect data from sensors using the Raspberry Pi Hardware (e.g., optical, acoustic, acceleration, magnetic field).

  During this lesson, students will learn how to:
  - Connect and collect data from the Accelerometer.
  - Reference hookup guide and information from Sparkfun's website.
  - Practice Python coding and concepts such as: looping and storing data into array (or list).
  - Create and use object-oriented code design: classes and methods.

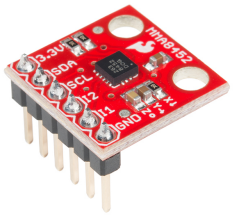
Let's begin:

1. Pre-lab assignment – if you have not watched the video on I2C yet, please do so now – you will need your headphones (in discussions on Canvas) – title: "PRELAB Video" on I2C communication" – it is 20 minutes long and very informative (thanks Ben).

   NOTE: The video mentions address 0xD1 but we use 0x1D… it appears in the sample code correctly.

2. Collect the equipment.
   The first thing to do is make sure you have the proper electronics ready … like following a recipe. Know what you are working with and the supplies you need. As you move forward into the Capstone, knowing how to find information about your various sensors is key.

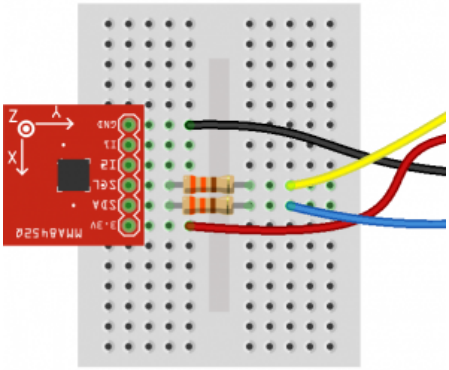
   SparkFun Triple Axis Accelerometer Breakout - MMA8452Q (with Headers)
   Reference Sparkfun website: https://www.sparkfun.com/products/13926

   By now, you know what you need to get your Raspberry Pi up and running, and collecting the proper wires, breadboard, and 330Ohm resistors from your box. You will also be using the I2C libraries (we enabled these in Lab2) for the main part of this lesson (if you do the extra credit, you determine what is needed).

   Remember: it is good practice to discharge yourself before handing electronics; you can easily do this by touching a grounded object.

3. Breadboard the accelerometer.
   Reference the actual chip and the hookup guide, you will quickly learn how to wire-up the components on your own. The last column in table below helps … try to understand each connect as if you did it without the photo or the table.

   Part of our table is directly from the hookup guide for the Accelerometer on Sparkfun's site - they use the Arduino … wiring is similar, even if code is different:
   https://learn.sparkfun.com/tutorials/mma8452q-accelerometer-breakout-hookup-guide?_ga=1.248608839.137750560.1486666660

Pin Label	Pin Function	Input/Output	Notes	Connect To Pi
3.3V	Power Supply	Input	Should be between 1.95 - 3.6V	3.3V
SDA	I2C Data Signal	Bi-directional	Bi-directional data line. Voltage should not exceed power supply (e.g. 3.3V).	SDA* 330 Ohm Resistor
SCL	I2C Clock Signal	Input	Master-controlled clock signal. Voltage should not exceed power supply (e.g. 3.3V).	SCL* 330 Ohm Resistor
I2	Interrupt 2	Output	Programmable interrupt — can indicate data ready, orientation change, tap, and more.	** Not used for this Lab
I1	Interrupt 1	Output	Programmable interrupt — can indicate data ready, orientation change, tap, and more.	** Not used for this Lab
GND	Ground	Input	0V/common voltage.	GND

** Documentation suggests using the 330 Ohm Resistance for SDA and SCL pins.*
*** We can ignore I1 and I2 (the "Interrupt" address channels and can be used for many things such as changing the slave address but we only have 1 device so its pretty pointless for this lab.)*

4. Basic data collection and testing your setup.
    a. Installing extra libraries to get us going:
        - You enabled I2C in Lab 2, so you should not have any problems with that part … if you want to check, you can … it is located in your RPi Configuration, under Interfaces.
        - We will need the following to do the work with the I2C (for libraries and function calls):
          sudo apt-get update
          sudo apt-get install build-essential libi2c-dev i2c-tools python-dev libffi-dev
          sudo apt-get install python3-smbus

    b. I2C – checking the address for the I2C address:
        - For the RPi, the address for the BUS we are using (BUS1) should be 0x1D, please check it by doing the below. If so, you are good to go, and ask if you have problem.

          Open terminal and type: **sudo i2cdetect –y 1**
          [You will see a grid appear, and a bunch of dashes … and a sole number of 1d]

    c. Before we implement a more "object oriented design" and reading our data using objects and a class, let's do some initial code. Sample code to collect and print accelerometer data – download the file: **AccelerometerLabBasic.py** from Canvas in Files->Labs – this matches the code in the pre-lab video (your real programming skills come into play with the object-oriented code in #5 :-).

    d. Explanation of Most Significant Byte (MSB) and Least Significant Byte (LSB) – since we are collecting 7 "bytes" of data that include the following: **Status register, X-Axis MSB, X-Axis LSB, Y-Axis MSB, Y-Axis LSB, Z-Axis MSB, Z-Axis LSB**, an explanation may be in order!

       Using the following output from experimentation, the following decimal values are from one read:

Status register	X-Axis MSB	X-Axis LSB	Y-Axis MSB	Y-Axis LSB	Z-Axis MSB	Z-Axis LSB
255	1	208	1	80	63	144

How to interpret these values … we are reading "bytes" of data and the device is giving us 2 of them for each axis (x,y, and z) … we need to combine these and interpret into something that makes more sense – so here are a few handy notes (and a formula for use later):

- 1 byte = 8 bits, then 2 bytes = divide by 16 bits.
- The value in each byte can be 0-255 - Since 00000000 is the smallest, you can represent 256 things with a byte = multiply MSB by 256.
- The check and subtractions are just keeping it in a range for readability/acceleration values.

```
MSB_x = data[1]
LSB_x = data[2]
numberOfBits = 16

xAccl = (MSB_x * 256 + LSB_x) / numberOfBits
if xAccl > 2047 :
    xAccl -= 4096
```

Sample code adapted from the example on this github site:
https://github.com/ControlEverythingCommunity/MMA8452Q/blob/master/Python/MMA8452Q.py

5. Object-oriented code.
   Next, update the basic sample code to use the concepts we have been learning in class:
   a. Fill in the header information with your name, the date, and a better description of the lab.
   b. Make a class to store Acceleration Data. The class name should be **Accelerometer** and should contain (at least):
      i. __init__ method with takes 3 parameters: x- acceleration value, y-acceleration value, and z- acceleration value – with defaults set to zero.
      ii. printData() method that prints x,y,z acceleration values formatted for example:
          Acceleration in x is 2
          Acceleration in y is 4
          Acceleration in z is -5
      iii. printCoord() method that converts the object to a string and prints out like a coordinate: i.e., if x-value = 2, y-value = 3, z-value = -2, then this should return a string (-2, 3, -2)
   c. When done with the creation of your class, use it! For example, from the main area:
      i. Create an array (or list) of **Accelerometer** objects [we have done this with basic data types, like integer, think about how this may work – I have provided a hint below … and/or look it up!] [arrays/lists Ch. 2.2]. There are many ways to do this, the basic idea is: read data, create object, store values, append this object to the array/list… here is a really rough, untested hint:

```
myArrayOrList = []
for i in range(xx_you_define_this):
    #read in data and convert to realistic values
    x = Accelerometer(xx_you_add_this)
    myArrayOrList.append(x)
```

      ii. Using a for loop, collect data and store it in the array (or list) [loops Ch. 2.4].
      iii. After data is collected, print it to the screen (and/or to a file if you prefer).

Submission Details – upload the following on Canvas:
(50 pts) Picture of your completed circuit.
(50 pts) Python Code (.py file) with changes defined in #5 above (please do not just submit the sample file we gave you – it needs to have classes, methods, arrays, etc.).

Creative Extensions … it is up to you to figure out how to add the following for possible extra credit, you can reference sparfun.com or raspberrypi.org and/or search on the internet… have fun and be creative!

Code based (10 pts each) code for:

1. Create a method that converts an array (or list) of objects to a string.
    a. For example: if your data is in an array (or list), let's say x, then myPrint(x) will print a nice string.
    b. Hint: python calls a method called __repr__ on each element in the array (or list).
2. Add another method to the Accelerometer class and explain what is does and/or why it is useful.
3. Simplify the code (namely the i2c stuff outside the class) to be in another class.
    a. You are free to do whatever makes stuff "easier" in your mind.
    b. Ex) Perhaps just calling a read() instead of read_byte_data(paramters) and have a method to change the slave address if the user needs too.
    c. This should show how classes make abstractions and your programs become easier to manage the more classes you have (i.e., you could just copy/paste this class into a new project if you ever need it as well).

Circuit (10 pts each) photo of circuit/code for:

4. Add some LEDs and have them change based on some orientation … be creative!
5. Connect another device that uses i2c communication and write some basic code showing how whatever you configured works. You can connect another device to the RPi or another device to the accelerometer maybe or both. You will likely have to look through documentation on the component to figure out which registers activate it.

    This will be handy to know and/or use for the Capstone! Connecting multiple devices is done frequently in larger scale projects and can present many problems.