CSCI 250 Python Computing: Building a Sensor System
TR 12:30-1:45 and 2:00-3:15 – MZ022 – Spring 2018
Lab 3: Experimenting with Sound
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:
  - Setup a circuit to read values from the Trimpot knob and ADC chip and play tones through the various piezo elements.
  - Referencing datasheets/specification sheets for sensors.
  - Complete an algorithm from pseudocode to control the sensors (read and write).
  - Practice Python concepts: using variables, importing libraries, array slicing, file input, using functions, casting floating pointing numbers to integers, and looping.

Purpose:

The purpose of this lab is to continue practicing with the Analog to Digital Converter (ADC) Chip and an introduction how to use additional sensors (e.g., Trimpot and Piezo) on your Raspberry Pi.

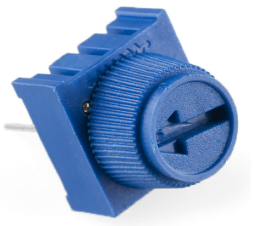
Overview of the ADC and SPI:

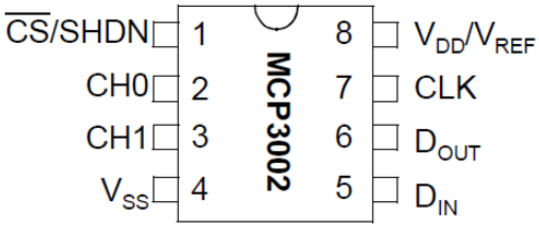
Last lab we focused mainly on the hardware of an ADC. This lab will be concerned about SPI, one of the most popular communication protocols (the other one we will learn later is I2C). A communication protocol is basically used to transfer data between micro-computers (e.g., Raspberry Pi) and add-on devices (our ADC MCP3002). An ADC is a type of sensor that converts analog voltage to digital voltage, which a microcontroller such as the RPi can read. Our ADC is a 10-bit analog to digital converter with only 2 channels. That means we can measure voltage to the precision that 10 bits gives us. For 0 – 3.3 V we only have an error of 0.003 volts! (1024 / 3.3 = 0.003).

PART 1 – Using the Trimpot 10K with Knob

Hardware: Let's begin:

The fist sensor we will be using is the Trimpot 10K with Knob and the values will come from the ADC.
Note: Reference your What's in the Box Worksheet and: https://www.sparkfun.com/products/9806





Note: if you forget how to use the ADC, go back to the Lab2 Instructions. The three pins on the Trimpot: 3.3V left or right, GND other left or right, analog middle pin – to channel 0 on the ADC.

Software: Time to code it up:

1. Open your IDE, create a new python file, the below is a template if you need it, and save it as sound.py (or something you deem worthy).

2. Include the proper libraries (numpy, time, spidev)

3. Instead of using the reader.py file, we are going to code up our own spi reader that only works with our ADC Chip (MCP3002). Though this program will only work on one model, understanding the process will help you set up other SPI devices in the future.

    a. Add the below ADC reader code to the top of your file for setup – before you use it and after you import the spidev library.

```
#Author:
#Date:
#Description:
#Stop … Think Exercise Answers:  a1, a2, a3, and c1

#import libraries:
#numpy for arrays
#time to use the sleep command
#spidev as spi for working with the ADC

#ADC reader code ... inline vs. included library
spi = spidev.SpiDev()         #create spidev object
spi.open(0,0)                 #(port, channel)
spi.max_speed_hz = 10000000   #optional, use so you don't overwork RPi
```

Stop … Think: Exercise: Learning to read the datasheet/specification sheets:

To write meaningful code to hardware you need to understand the hardware well. You will need the datasheet to complete this. I will help you out since this is first datasheet. Find the datasheet on the *Sparkfun site: Analog to Digital Converter - MCP3002, Documents, Datasheet.*

First, go to Figure 6-1. **Answer these questions at the top of your code based off this figure**.

a1. At what clock value is data transmitted? (0-falling or 1-rising?)
a2. At what clock value is data received? (0-falling or 1-rising?)
a3. How many bytes are transmitted/received? (recall 1 byte = 8 bits)

b. Define a new function (see Ch. 5.1 for reference), this is new, so we will give a sample here:

```
def readAdc(channel):
        #You may use the incoming parameter to make more flexible later
        #Read the raw data for channel 0 using the xfer2 method, which
        #sends AND receives depending on the clock rise/fall.
        r = spi.xfer2([int('01100000',2), 15])

        #Get data
        #get 10 bit bitstring from r[0]
        s = bin(r[0])[2:].zfill(10)
        #append 8 '0's to last 2 bits from r[0]
        data = int(s[8:] + '0'*8, 2) + r[1]
        return data
```

c. Finish up with this basic algorithm:

1. Using a while true loop
2. Read in the raw data from the ADC channel calling the **readADC** function you just created.
3. Print the returned values to the screen.
4. As the program runs, turn the knob and to verify the circuit is working properly. You should see the values fluctuate between 0-MAX_VALUE ** see below Stop … Think Exercise for this value.

NOTE: Make sure your code is working properly before moving on … I will be running your code to test this range of values for grading.

Stop … Think: Exercise: Learning to convert binary number to decimal:

Next, you can look this up if you don't know … **Answer this question at the top of your code**.

The raw data will just return an array of numbers that don't have any meaning to it. It's on you the programmer to make it mean something. What do we want back exactly? Well we have a 10-bit ADC so we want values ranging from 0 - MAX_VALUE.

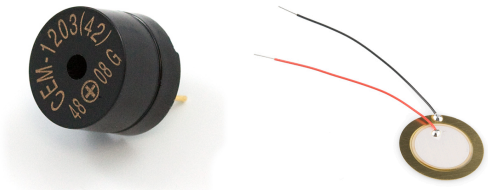
c1. What is the value for MAX_VALUE … max value 10 bits can recognize (e.g., 1111111111)?

PART 2 – Let's get some sound going!

Hardware: Let's add some more:

The next sensor we will be using is the Piezo Speaker (and/or the Piezo Element) and the values will be controlled from the GPIO and volume levels controlled by the Trimpot. Yes, that's right – so fun!

Note: Reference your What's in the Box Worksheet and: https://www.sparkfun.com/products/7950 and https://www.sparkfun.com/products/10293.



*You won't have to change up your current circuit … just **add** a few wires! Piezo's are easy …*

1. *One leg goes to ground on the speaker (black on element)*
2. *The other is for analog data (red on element) [Add a wire to connecting the Trimpot and Piezo]*
3. *The data leg (or red on element) ALSO needs to be connected to a GPIO pin of your choosing.*

Software: Time to code it up again:

a. Copy/paste or include the provided **buzz** function from the buzz.py file located in Canvas.

   NOTE: For the base lab we will do a naive hardware approach (software approach works better but we've doing enough coding for today).

b. Add the code to use the GPIO – recall Lab 1?! (include the proper library, setup for pin mode, setup the GPIO pin for output to control the Piezo) …
   HINT: the pin variable used in the buzz function is called buzzerPin)

c. Test that your circuit works by calling the buzz function in your while loop. You can choose a random pitch and duration to plug in the buzz function to (e.g., pitch = 500, duration = 0.2).

   NOTE: When the Trimpot is in the "middle" the sound will be louder, when at the ends it will be quieter. That is because our potentiometer is currently acting a voltage divider (has all three pins hooked in), which gives good readings but doesn't give a consistent relationship with current. We would rather have our potentiometer as a "variable" resistor as that will affect the current going into the speaker more (I = V/R) as we hold V constant. To change your potentiometer into a variable resistor (called a rheostat) take off it's ground pin.

Your volume control should now work…

d. Finally, time to play an American classic tune! Here is the basic algorithm:
   I. Download the song.txt file from Canvas (put it in the same directory as your .py file).

   II. Read in the data file into an array …Column 1 holds the data for the pitch and column 2 holds the data for the duration.
   HINT: (data = np.loadtxt("filename"))

   III. Use your python skills to slice the incoming array and cast the pitch from a floating-point number to an integer.
   HINT: a function in the numpy library to convert to int: pitches = np.int_(data[:,0])

   IV. Create a for loop to pass each row of data properly into the provided **buzz** function to play the song!
   HINT: To loop like through 2 arrays at the same time you can do the following:

```
for val1, val2 in zip(arr1, arr2):
        #do stuff … like calling the buzz function maybe :)
```

   The zip function is built into to python and it basically allows you to iterate through multiple "iterable data structures" (arrays, dictionaries, and more). It's pretty good to have in your knowledge base.

Submission Details – upload the following on Canvas:
(50 pts) Picture of your circuit with ADC and Trimpot Knob and either one of the Piezo elements.
(50 pts) Python Code (.py file that we can run).

Creative Extensions … recall, these are vague because 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!

1. (5 pts) Use Channel 1 instead of Channel 0. Reference the datasheets and provide comments of this in your code showing how you would do it in all places needed.
2. (5 pts) pass an extra parameter to readADC function that lets you change the channel. Use integers as arguments (0 = CH0 and 1=CH1) NO CONDITIONAL LOGIC ALLOWED (if/else). (hint: use bitwise operations).
3. Here is a datasheet for another ADC that could be used for projects that need more channels, the MCP3008: http://ww1.microchip.com/downloads/en/DeviceDoc/21295C.pdf. You can answer these with comments at end of code or just submit the comment directly on canvas.
    a. (5 pts) What bit array you would pass into xfer2 function if you wanted to use channel 8.
    b. (5 pts) How you would extract the correct 10 bits from xfer2's return.