UNIVERSITY OF EAST ANGLIA

School of Computing Sciences

E3I Club Short Course — Raspberry Pi Interfacing

ANALOGUE TO DIGITAL CONVERSION — Using the MCP3008 ADC

1 Introduction

In this lab, we will interface the MCP 3008 Analogue to Digital convertor (ADC) to the Raspberry Pi 3 and capture a number of different analogue signals. The MCP 3008 ADC is a general-purpose ADC that gives 10 bits of resolution at a conversion rate of up to 200 000 samples per second. The datasheet is available at `https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf`.

2 Preparation

2.1 Software

Firstly, you may need to download some software to drive the MCP 3008 from the Pi. Check whether you have the directory `Adafruit_Python_MCP3008/` If you do, go to section 2.2. If not, from the terminal, execute the following commands:

```
>sudo apt-get install git build-essential python-dev
>cd ~
>git clone https://github.com/adafruit/Adafruit_Python_MCP3008.git
>cd Adafruit_Python_MCP3008
>sudo python setup.py install
```

Alternatively you can install from pip with:
`>sudo pip install adafruit-mcp3008`
However, if you do this, the examples won't be downloaded.

2.2 Configuring the Interface

You now need to enable the SPI interface on the Pi. There are three methods of doing this, as described at
`http://www.raspberrypi-spy.co.uk/2014/08/enabling-the-spi-interface-on-the-raspberry-pi/`.
Use one of these—the first or second are the easiest.

2.3 Wiring

To connect the Pi to the ADC chip, we use a *breadboard*. A typical breadboard is shown on the left of Figure 1. The holes along a row of the board are electrically connected together. On some

boards (including the one shown here), the holes along an outer *column* are connected: these columns are usually connected to a power supply to provide a voltage "rail". Components' legs are pushed into the holes of the breadboard and can then be connected to other components by inserting connecting wires.

Insert the MCP 3008 chip so that it straddles the central gap of the breadboard, and do the same for your Cobbler connector. Then connect the two chips using the wiring diagram on the right of Figure 1.

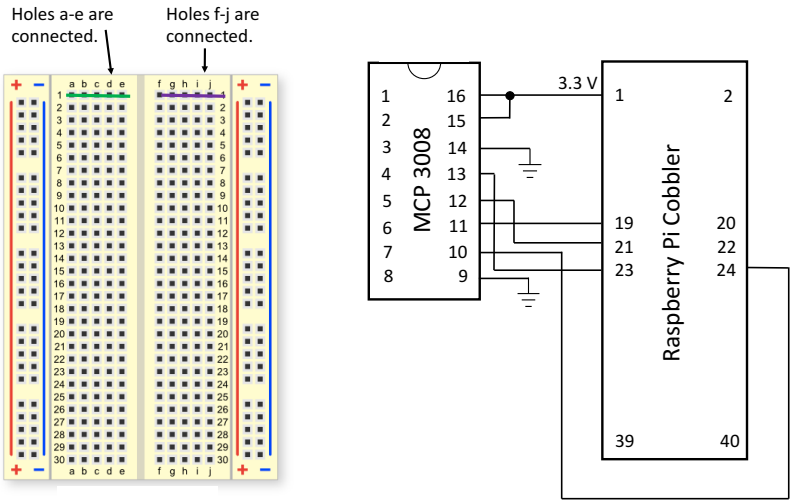

Figure 1: Left: Breadboard. Right: Connecting the MCP 3008 to the Raspberry Pi 3.

3 Calibrating the ADC

When you have made the circuit, it's time to test and calibrate the ADC. To do this, we will generate a voltage that we can control using a potentiometer (as described in the lectures).

Find a breadboard potentiometer in the Lab component store. They look as shown on the left of Figure 2. Any value in the range 1 kΩ to 1 MΩ is suitable. Mount it in the breadboard and connect pin 1 (as shown on the right of Figure 2) to ground and pin 2 to 3.3 V i.e. Cobbler pin 1. Connect the COM (black) connection of a voltmeter to pin 1 and the other voltmeter terminal to pin 3. Use a "pot tweaking tool" to turn the wiper of the potentiometer and note the voltage reading as you turn: you should be able to vary the voltage smoothly from 0 V to 3.3 V.

Set the output from the potentiometer to 3.3 V (maximum) and connect pin 3 to pin 1 of the MCP 3008 (which is input channel 0). Run the program `simpletest.py` by executing

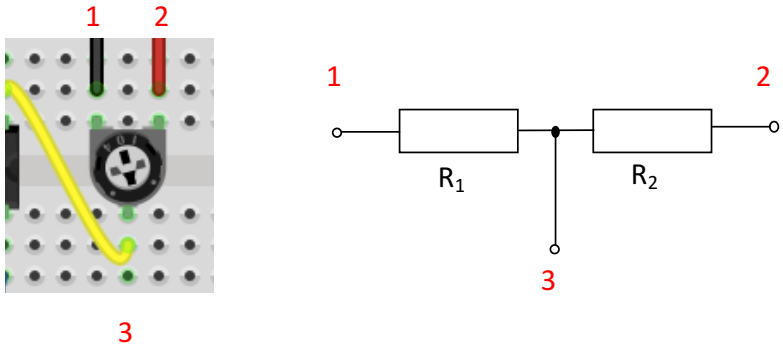

Figure 2: Breadboard potentiometer and its circuit diagram.

the command `sudo simpletest.py` in the directory `Adafruit_Python_MCP3008/examples`. `simpletest.py` samples the output from each channel of the ADC every 0.1 seconds. The program displays the values of all 8 channels. You should see that channel 0 is fixed at a value of 1023 (the maximum for a 10 bit convertor) and the other channels display low random numbers. Check that turning the wiper a little reduces the voltage and the output code on channel 0 decreases.

The random numbers from the other channels are a consequence of *electronic noise*. If inputs are left unconnected, they tend to pick up noise from the circuit. Connect one of the other 8 channels to ground: you will see that its output is now fixed at code 0.

Make a plot of the input voltage against the output code. This enables you to calibrate the ADC. With `simpletest.py` running, adjust the potentiometer to give the codes (say) $0, 100, 200, \ldots 1000$, and each time you obtain one of these codes, make a note of the corresponding input voltage. Use any plotting program to plot the voltage against its corresponding code (`Matplotlib` is very good for this). You should get something like Figure 3. What change in voltage is required to change the code by one LSB? *Challenge:* as you are only using one channel, can you modify `simpletest.py` so it reads and displays only channel 0?

4 Capturing a sine-wave

In this section, we will capture a varying signal and demonstrate the perils of aliasing. We will use a signal generator rather than a real sensor to generate the input signal to the ADC, as this gives us complete control over the amplitude and frequency content of the signal.

Each workplace in the Lewin Lab has a *signal generator*. It looks as shown in Figure 4. The generator can output sine-waves, square-waves or pulses. Set it to generate a 1 Hz sine wave of amplitude about 1.5 V with a minimum value of 0 volts. This is done by setting:

1. FUNCTION button selector to the sine-wave symbol

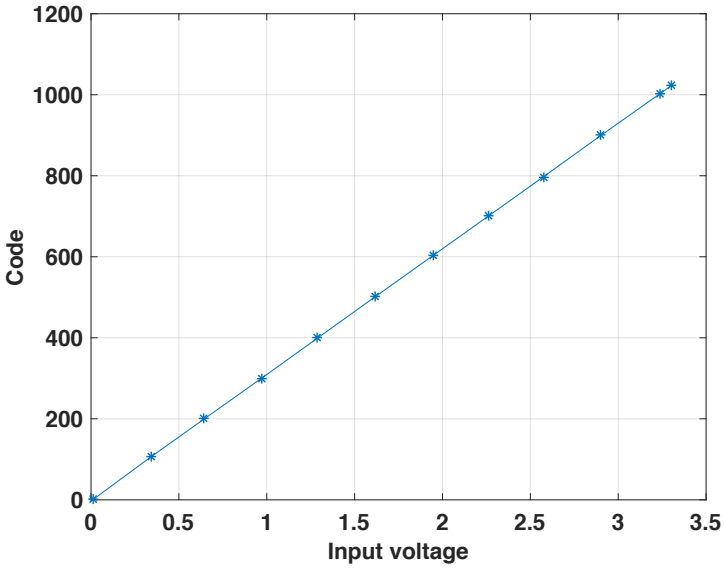

Figure 3: Input voltage vs. output code for ADC.

Figure 4: Signal generator set to generate a 1 Hz sine-wave

2. RANGE Hz button selector to 1
3. The dial to 1.0 (the dial value multiplies the RANGE Hz value)
4. AMPL button pushed IN and turned approximately to position shown in Figure 2
5. OFFSET ADJ turned slightly beyond centre as shown in Figure 2

Before connecting to the ADC 3008, first check that the amplitude and range of the sine wave voltage is correct. Connect the signal generator to the voltmeter using a BNC to crocodile clips

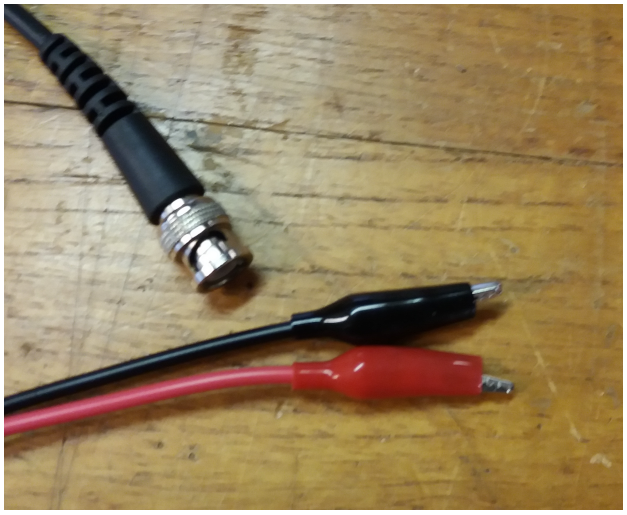

Figure 5: BNC to croc-clip leads.

lead, shown in Figure 5. The ADC 3008 accept signals in the range 0 V to 3.3 V (in the way we have configured it) so the sine wave should occupy this full range. The generator voltage is slowly varying, so the voltmeter is constantly changing its reading, but adjust the AMPL control and the OFFSET ADJ control until the highest voltage you see is just below 3.3 V and the lowest 0 V. When you are happy that the signal is about right, connect the output of the signal generator to channel 0 of the ADC. Use the code `sample_and_plot.py` to plot the resulting sine wave and confirm its frequency. *Challenge:* Find out how to write the samples to a file on your Pi.

4.1 Aliasing

Look again at the lecture where I described aliasing. In the example, the frequency of the sine-wave to be sampled was f_0 Hz and it was sampled at $f_s = 4f_0/3$ Hz. This violates the Nyquist criterion, which says that we must sample at *at least* $2f_0$ Hz to acquire the waveform correctly. The next slide stated that the result of this undersampling would be a sine-wave of frequency $f_0/3$ Hz, as the sine wave is aliased by reflection in a "mirror" at $f_s/2$ Hz. Let's check this result. If we make $f_0 = 6$ Hz, then $f_s = 4f_0/3 = 8$ Hz and sampling at this frequency should result in an aliased waveform of frequency $f_0/3 = 2$ Hz.

Set the frequency of the signal generator to 6 Hz: the voltmeter can also measure frequency by turning its dial to "Hz". Set `samplingPeriod` in `sample_and_plot.py` to 0.125 and run the program. Confirm that the frequency of the wave observed is 2 Hz.

Prof. Stephen Cox
January 12, 2017