Lab 5: Interfacing Analog to Digital Converter (ADC) with Microcontroller Report

Tyler Nardecchia
Yerania Hernandez
CSCE 462 - 502
October 21st, 2017

Thinking and Exploring

1) *Summarize the difference between SPI and I2C ports. Explain in what situation using the SPI ports is better than the I2C ports, and vice versa.*

SPI and I2C ports are both bus protocols that allow for serial data transfer. SPI can only have one master device that can control multiple slaves, while I2C can actually allow multiple masters and slaves on the bus. Although the master devices in I2C and SPI can control the clock for all the slaves, I2C slave device also has the capability of modifying the main bus clock. The hardware design implementation for I2C also is simpler than SPI considering it only needs a two wire bus, compared to the SPI that requires an additional wire bus for each slave device. I2C is also system with bidirectional data on the SDA line compared to SPI that is a point-to-point connection that allows data input and output on separate lines (MISO and MOSI).

As a result, using SPI would be better when speed is a primary concern along with not having enough connections to address each device. SPI provides a faster speed and longer ranges driving lines faster than I2C, which uses an open-collector design causing limitations on slew rates. However, I2C would be preferable when there is a need for flexibility. I2C has a need for fewer connections: only two signal wires necessary compared to the SPI where it needs three signal wires plus n number of slave devices. Due to its open-collector design, it allows for flexibility in bus voltage, while in SPI the bus voltage must be something supported by all devices. It also allows for multiple masters adding more flexibility in changing between multiple devices.

2) *What are the various types of ADC's in use? Which type of ADC is MCP3008 and what are its advantages/disadvantages?*

There are various types of analog to digital converters including Flash, Pipelines, Successive-Approximations Register (SAR), Integrating or Dual-slope, and Sigma Delta. MCP3008 actually has a successive approximation register (SAR) architecture. This type of converter evaluates each bit at a time, from most to least significant bit. It will use comparision to set bit to 1 and compare the output of the DAC with the input voltage and then latch the comparator result to the same bit in the register. The advantages of this method is that is has low power consumption and it only uses one comparator. The disadvantages, however, is that the DAC grows with the number of bits and therefore it will take as many cycles to convert this signal as the number of bits that are provided. Also, the mismatch in the DAC limits its linearity and higher resolution successive approximations will be slower as a result.

3) *What is the sampling rate? What is the sampling rate for your mini-oscilloscope?*

The sampling rate of an oscilloscope is 1 divided by the fixed sample interval when the ADC samples the input signal, meaning it is the rate at which the ADC is clocked to digitize the incoming signal. Due to the sample time interval of our mini-oscilloscope, our sampling rate is approximately 10,000 samples per second.

4) *Remember, never use the same raspberry Pi to do a waveform generation and a waveform recognition at the same time. Otherwise, you will generate a waveform that frequency keeps changing or get random readings from the MCP3008. Explain why this is the case.*

Using the same Raspberry Pi to perform waveform generation and waveform recognition simultaneously can cause issues because each function requires a specific, constant delay in between its operations. For example, the DAC uses a step time in which the analog output of the wave is changed, and the ADC uses a constant sampling rate for determining the frequency of the input wave. Using both the DAC and ADC concurrently on the same Raspberry Pi will implement delays for both functions at the same time, which will accumulate the total delay between operations past the desired rate. This will throw off the step time for the DAC as well as the sampling rate for the ADC, causing their results to be unreliable and unpredictable.

5) *It is highly likely that your sampled data contains lots of noise. How you can filter the noise? Explain your method.*

One way to filter out noise from the ADC is by using a low-pass (anti-aliasing) filter prior to the ADC. This method would use a capacitor and resistor along with an operational amplifier in order to act as a buffer right before the connection to the ADC. Additions of digital buffers help isolate the ADC from the data bus. It also helps eliminate the effects of overdriven signals beyond the bandwidth of the filter to avoid saturation. As a result, it will allow us to eliminate the noise we find through our conversion.

```
import time
import ui_plot
import sys
import numpy
from PyQt4 import QtCore, QtGui
import PyQt4.Qwt5 as Qwt
import spidev

#Open SPI Bus
spi = spidev.SpiDev()
spi.open(0,0) #1st Param is Channel

def readChannel(chan):
    adc = spi.xfer2([1, (8+chan)<<4, 0]) #sends 3 bytes
    print(adc)
    data = ((adc[1]&3)<<8) + adc[2]
    print(data)
    return data


#Sampling the input wave
numPoints=100
xs=numpy.arange(numPoints)
ys = [None] * numPoints #Creating an empty array of size numPoints
i = 0
while i<numPoints:
    time.sleep(0.0001)
    ys[i] = readChannel(0)
    i = i+1

#Calculating frequency based on sample
valley1y = 0
valley1x = 0
for x in range (1, numPoints - 1):
    if ys[x] < ys[x - 1] and ys[x] <= ys[x + 1]:
        valley1y = ys[x]
        valley1x = x
        break

valley2y = 0
valley2x = 0
for x in range (valley1x + 1, numPoints - 1):
    if ys[x] < ys[x - 1] and ys[x] <= ys[x + 1]:
        valley2y = ys[x]
        valley2x = x
        break

wavelength = (valley2x - valley1x) / 10000.0
frequency = (1.0 / wavelength)/3 #Calcuation with error and time constant
print("\nFREQUENCY:")
print(frequency)

#Plotting the sample that was calculated in xs and ys
def plotSomething():
    global ys
    ys=numpy.roll(ys,-1)
    #print "PLOTTING"
    c.setData(xs, ys)
    uiplot.qwtPlot.replot()

if __name__ == "__main__":
##    readChannel(0);

    app = QtGui.QApplication(sys.argv)

    ### SET-UP WINDOWS
```

```
67
68      # WINDOW plot
69      win_plot = ui_plot.QtGui.QMainWindow()
70      uiplot = ui_plot.Ui_win_plot()
71      uiplot.setupUi(win_plot)
72      c=Qwt.QwtPlotCurve()
73      c.attach(uiplot.qwtPlot)
74
75      uiplot.timer = QtCore.QTimer()
76      uiplot.timer.start(100.0)
77
78      win_plot.connect(uiplot.timer, QtCore.SIGNAL('timeout()'), plotSomething)
79
80
81      ### DISPLAY WINDOWS
82      win_plot.show()
83
84      #WAIT UNTIL QT RETURNS EXIT CODE
85      sys.exit(app.exec_())
```