CSCE 462 Lab 3 Report

Osric Nagle and Jonah Taylor

2/22/2021

Questions to Explore

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.
    a. SPI is a high-speed, low-power communication method. It is capable of transmitting data at very high speeds, but it is more limited in the scope of its applications. On the other hand, I2C is a widely-followed standard that can be used for communication between many types of devices. It is much slower than SPI, but it requires fewer wires and provide greater power output.
    b. SPI ports are useful for applications where the Raspberry Pi needs to write/read data very quickly. The much faster transfer speed of SPI ports make them essential for data-intensive operations. On the other hand, I2C is useful for controlling devices or reading small amounts of data. Its simpler setup and standardization makes it a convenient option for most uses.
2. What are the various types of ADCs in use? Which type of ADC is MCP3008 and what are its advantages/disadvantages?

    The five major types of ADC in use today are:

    - Successive Approximation (SAR) ADC - This type of ADC consists of
    - Delta-sigma ADC
    - Dual Slope ADC
    - Pipelined ADC
    - Flash ADC

    The MCP3008 is of the SAR type, which is the most used type of ADC found across all industries/uses of ADCs. This type of ADC is used often due to its inherent balance of speed and resolution, as well as its ability to handle a wide range of expected and unusual signals. Since it is the oldest type of ADC, it has some of the most inexpensive and reliable chip designs, adding another layer of benefit to using this type of ADC. However, a notable disadvantage to SAR type ADCs are their complete lack of built-in aliasing filter capabilities. Thus, any use of this type of ADC must actively plan around this by filtering at a high enough rate above the frequency of the input signal, or if this is not possible, adding additional hardware to filter the signal before it is completely digitized by the ADC. Once signal aliases become digitized, it is impossible to remove them with software.

    Information obtained from:
    https://dewesoft.com/daq/types-of-adc-converters
    https://components101.com/ics/mcp3008-adc-pinout-equivalent-datasheet

3. What is the sampling rate for your oscilloscope?
    a. To measure the sampling rate of the oscilloscope, we kept a counter of each time we measured a (smoothed) data point from the input, and divided that by the time since the program started. Using this method, we found that the sampling rate for our oscilloscope

was 550 Hz. The method we used to compute the sampling rate is included in the code at the bottom of this report (commented out).

4. If you use the same Raspberry Pi to do waveform generation and waveform recognition at the same time, you might generate a waveform that the frequency keeps changing and get random readings from the MCP3008. Explain why this is the case.
   If the same device is used to both generate and interpret a waveform, then the processor on the Raspberry Pi must compute all tasks required for both the wave generation and wave sensing programs. Any non-parallelizable parts of these two programs will have to occur in series. Thus, the computation time of the wave sensing program would cause the wave generation function to be delayed in ways it would not realize/not account for, causing constant frequency shifts in the generated waves. The sensing function would be able to determine these changes, and would thus continuously output different frequencies for the function.

5. It is highly likely that your sampled data contains lots of noise (in amplitude and frequency). How can you filter the noise? Explain your method.

   In our code, we decided to deal with inherent noise in our data in three separate ways. In the first part of our function, we chose to minimize noise by finding the first voltage value that was greater than a threshold value, and once that was found, we would wait until the first value found that was lower than that threshold value. This method was used in tandem with an averaging function, which simply took an average of two sampled values and returned it (which would help factor out some noise). We chose to only average two points based on experiments, where we found that an average of two proved to be roughly as effective as an average of more points but retained the lowest speed (since fewer points had to be sampled). The two noise removing methods helped eliminate enough noise to give a correct classification of intended function types, as well as a close estimation of frequency. The final solution for filtering noise is simply by displaying the function type and frequency as soon as the code recalculated them. By doing so, the erroneous measurements are surrounded and greatly outnumbered by the correct function/frequency values, and so it becomes obvious to the user what the correct values are.

Code:

```
#import relevant libraries
import time
import os
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import math

#set up board + ADC chip
```

```
#create spi bus
spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

#create the chip select (which port slave chip is connected to)
cs = digitalio.DigitalInOut(board.D22)

#create MCP object
mcp = MCP.MCP3008(spi, cs)

#create an analog input channel on pin 0 of ADC
input_chan = AnalogIn(mcp, MCP.P0)

#General structure of program:
desired_voltage = 1
sample_average = 0.0
error_threshold = 0.1
#num_samples = 0  # used for oscilloscope frequency measurement

def collectData():
    #global num_samples  # used for oscilloscope frequency measurement
    sum = 0
    for i in range(0, 2):
        sum += input_chan.voltage
    output = sum / 2.0
    #num_samples += 1  # used for oscilloscope frequency measurement
    return output

#Period finding function
#can change the zero requirement to just any arbitrary voltage value,
#in case not all functions are centered at zero.
def find_period():
    while collectData() > desired_voltage:
        pass
    while collectData() < desired_voltage:
        pass
    time_begin = time.time()        #first zero reached

    #Loop until next zero voltage value found
    while collectData() > desired_voltage:
        pass
```

```
    while collectData() < desired_voltage:
        pass
    time_end = time.time()          #second zero reached
    period = time_end - time_begin
    return period

try:
    #startTime = time.perf_counter()  # used for oscilloscope frequency
measurement
    while(True):
        #1. Find frequency (how? - check for first instance of zero, sleep
for a time)
        period = find_period()
        #sample a data point 1/8 of the way through the cycle
        sleeptime = abs(.125 * period - 0.00225)
        time.sleep(sleeptime)
        sample_average = collectData() # input_chan.voltage

        #2. Set the max and min voltage values
        min_volt = 0
        max_volt = 2

        #3. Using previous two, use expected function values at specific
intervals to determine
        # type of function.
        # specific interval = 1/8 * period
        # assumption: function is rising to its max value when we make the
check.
        expected_amplitude_sin =  (max_volt - min_volt) * 0.707 / 2 +
(max_volt + min_volt) / 2
        expected_amplitude_square = max_volt
        expected_amplitude_tri = .75 * (max_volt - min_volt) + min_volt

        #Compare sample sum to expected values
        if abs(expected_amplitude_sin - sample_average) < error_threshold:
            #Sin wave likely found
            print("Sin,   ", end='')
        elif abs(expected_amplitude_tri - sample_average) <
error_threshold:
            #Tri wave likely found
```

```
            print("Tri,   ", end='')
        elif abs(expected_amplitude_square - sample_average) <
error_threshold:
            #Square wave likely found
            print("Square ", end='')
        else:
            print("bruh   ", end='')

        #print("Sampling Frequency = ", num_samples / (time.perf_counter()
- startTime))  # used for oscilloscope frequency measurement
        print("Frequency = ", 1 / period)

except KeyboardInterrupt:
    print("Exiting...")
```