Alexander Akomer (Partner: Shaheen Ebrahimi)

CSCE 462 – 500 Liu

2/25/2022

Lab 3 Report

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 draws less power than I2C ports and is more viable in short-term applications. I2C ports, being more viable in long-term applications, are less susceptible to noise and also support multiple devices on the same bus without multiple select lines (useful for supporting several peripherals at the same time).

2) **What are the various types of ADCs in use? Which type of ADC is MCP3008 and what are its advantages/disadvantages?**
There are several types of ADCs, including flash ADCs, sigma-delta ADCs, dual slope converters, and successive approximation converters. The MCP3008 is a successive approximation converter, which boasts high speeds and average accuracies. The advantage is the versatility of applications it can be applied to due to its good tradeoff between speed and cost. A disadvantage is the speed limit of 5 Msps in addition to slow processing speeds (because of a high resolution).

3) **What is the sampling rate for your oscilloscope?**
Our oscilloscope’s sampling rate is not limited by sleeps or any other software configurations; the code is designed to maximize the sampling rate of the hardware. The Raspberry Pi 4 is limited by the MCP3008, which has a sampling rate of 200kHz. Because our code has about 4 operations per sample, it can be estimated that the sampling rate of the oscilloscope is roughly 50 kHz.

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 both waveform generation and recognition are performed on the same Raspberry Pi, it is important to note that processes can never truly run in parallel, especially given the fact that the code is not multithreaded and therefore must truly run sequentially. Therefore, the processor must context switch between the jobs of generation and recognition and can never truly measure a perfectly-generated waveform.

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.**
Noise can be filtered using several techniques. A common method to filter this noise out is using Fourier transforms. However, in our code, we simply filtered the code using a

normalization function which divides by the amplitude to account for outliers. In addition, a cleanData function rounds data points to within a tolerance of each other to supply cleaner data. In the end, wave type determinations can be created by using a HashMap to create a histogram of voltages, and then checking the standard deviation across these cleaned data points.

Code:

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

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

# create the cs (chip select)
cs = digitalio.DigitalInOut(board.D22)

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

# create an analog input channel on pin 0
chan0 = AnalogIn(mcp, MCP.P0)

tolerance = 0.01     # to keep from being jittery we'll only change


# Round and normalize data
def cleanData(data):
    for i in range(len(data)):
        data[i] = round(abs(data[i]), 2)
    return data

# Get the maximum value in the dictionary
def dictMax(dictionary):
    max_key = -1
    max_val = -1
    for i in dictionary.keys():
        if dictionary[i] > max_val:
            max_key = i
            max_val = dictionary[i]
    return (max_key, max_val)

# Determine the wave type
def waveType(slopes, period, prev_wave):
    frequency = 1/period
    histogram = {}

    for i in range(len(slopes)):
        if (slopes[i] in histogram):
            histogram[slopes[i]] += 1
        else:
            histogram[slopes[i]] = 1

    (key, val) = dictMax(histogram)
    if (val / len(slopes) >= 0.80): # if mainly zero slopes then square wave
        return "Square"

    slopes = [i for i in slopes if i != 0] # remove zeros
    stdev = statistics.stdev(slopes) # take standard deviation

    if (abs(frequency - 10) < 1):
        if (stdev < 10):
            return "Triangle"
        elif (stdev < 25):
            return "Sine"
        else:
            return prev_wave

    if (abs(frequency - 25) < 2):
        if (stdev < 30):
            return "Triangle"
        elif (stdev < 55):
            return "Sine"
        else:
            return prev_wave

    if (abs(frequency - 50) < 3):
        if (stdev < 70):
            return "Triangle"
        elif (stdev < 140):
            return "Sine"
        else:
            return prev_wave

# Get the slopes from input t and y
def slopes(t, y):
    m = []
    for i in range(1, len(t)):
        m.append(abs((y[i] - y[i-1]) / (t[i] - t[i-1])))
    return m

# Take given amount of data samples of input wave
def waveData(samples):
    t = []
    y = []
    s = 0
    t_start = time.time()
    while (s < samples):
        y.append(chan0.voltage)
        t.append(time.time() - t_start)
        s += 1
    return (t, y)

# Find peak to peak period
def pkpPeriod():
    delta_y = []
    first_edge = False

    prev_volt = chan0.voltage
    start = time.time()

    while (True):
        curr_volt = chan0.voltage
        delta = curr_volt - prev_volt
        delta_y.append(delta)

        if (len(delta_y) > 1): # ignore first iteration
            if (delta_y[len(delta_y)-2] <= 0 and delta_y[len(delta_y)-1] > 0): # min value
                if (abs(delta_y[len(delta_y)-1]) - abs(delta_y[len(delta_y)-2]) > tolerance):
                    if (not first_edge):
                        start = time.time()
                    else:
                        return time.time() - start
                    first_edge = True
            prev_volt = curr_volt

# Main logic method
def logic():
    prev_wave = 'xxx'
    period = pkpPeriod()
    print("Frequency: ", round(1/period, 4), " Hz")
    while (True):
        (t, y) = waveData(1000)
        y = cleanData(y)
        m = slopes(t, y)
        wave = waveType(m, period, prev_wave)
        if (wave != prev_wave):
            print(wave, "wave")
            prev_wave = wave

# Run logic
logic()
```

Schematic: