Bobby Vielma CSCE 462

Alexia Perez

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.
   1. I2C: I2C is basically a two-wire communication protocol. It uses only two-wire for communication. In which one wire is used for the data (SDA) and other wire is used for the clock (SCL). In I2C, both buses are bidirectional, which means the master able to send and receive the data from the slave
   2. SPI: SPI follows the master and slave architecture and communication is always started by the master. Like I2C it is also a synchronous communication protocol because the clock is shared by the master and slave. SPI is supported only multi-slave does not support multi-master and slaves are selected by the slave select signal. In SPI during the communication data is shifted out from the master and shifted into the slave vice- versa through the shift register.
   3. SPI is better for short distances and I2C is better for long distances. SPI is also faster than I2C and draws less power, but is more susceptible to noise. In any case in which you only need to communicate over short distances and care about speed and efficiency over accuracy, use the SPI.
2. What are the various types of ADCs in use? Which type of ADC is MCP3008 and what are its advantages/disadvantages?
   1. There are five types:
      1. Successive Approximation (SAR) **ADC**.
      2. Delta-sigma (ΔΣ) **ADC**.
      3. Dual Slope **ADC**.
      4. Pipelined **ADC**.
      5. Flash **ADC**.
   2. The MCP3008 is a 10-bit, 8-channel, SPI-based ADC
      1. Pros: Easy to use, cheap, pros that come with being SPI based
      2. Cons: Not the fastest nor the most precise
3. What is the sampling rate for your oscilloscope?
   1. 650Hz
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.
   1. Having the same Raspberry Pi do waveform generation and waveform recognition eats up power, bandwidth, and storage. Since the MCP3008 is already not the fastest nor most precise device any fluctuations in processing power by the Raspberry Pi may cause changes and fluctuations in readings from the MCP3008.
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.
   1. It could be possible to take a rolling average of both the amplitude and frequency and display the averages instead of each individual value. This is a sort of “meet-in-the-middle” approach.Through researching we also found that making sure any unused pins were grounded so voltages do not leak into other pins should also be done if not already.

**Code**

import busio

import digitalio

import board

import adafruit\_mcp3xxx.mcp3008 as MCP

from adafruit\_mcp3xxx.analog\_in import AnalogIn

import time

import matplotlib.pyplot as plt

spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)

cs = digitalio.DigitalInOut(board.D22)

mcp = MCP.MCP3008(spi, cs)

chan0 = AnalogIn(mcp, MCP.P0)

def gatherData():

voltageList = []

timeList = []

startTime = time.time()

while time.time() - startTime < 1: #gather 1s of data points

voltageList.append(chan0.voltage)

timeList.append(time.time()-startTime)

smoothedVoltageList = []

smoothedVoltageList.append(voltageList[0])

for i in range(1, len(voltageList)-1): #smooth data with rolling average spanning 3 numbers

avg = (voltageList[i-1] + voltageList[i] + voltageList[i+1]) / 3

smoothedVoltageList.append(avg)

smoothedVoltageList.append(voltageList[len(voltageList)-1])

#plot the 1s of data to prove the actual frequency output by the function generator (we used our generator from lab 2 on another Pi)

plt.plot(timeList, voltageList, 'o', color="black")

plt.show()

return voltageList, smoothedVoltageList, timeList

def characterizeWaveform(rawVoltageList, voltageList, timeList):

length = len(voltageList)

max = 0.0

min = 100000000.0

for i in range(length): #obain max and min from smoothed voltage list

if (voltageList[i] > max):

max = voltageList[i]

if (voltageList[i] < min):

min = voltageList[i]

if (i == length - 1):

continue

#test if it is a square wave

numMin = 0

numMax = 0

tol = (max-min) \* 0.1 #tolerance to account for noise (experimentally determined)

for i in range(length): #determine amount of points that lie on extremes

if max - voltageList[i] < tol:

numMax = numMax + 1

elif voltageList[i] - min < tol:

numMin = numMin + 1

if numMin + numMax > (length \* 0.8): #if at 80% of points lie on extremes it is a square wave

print("Square")

squareFreq(rawVoltageList, timeList, min, max)

return

#test if its a triangle or sin wave

Edge = []

start = -1

end = -1

side = -1 #0 is falling side, 1 is rising

consOpp = 0 #account for fluctuations

for i in range(1, length): #find one continuous edge (rising or falling) and store the start and end indices

if (voltageList[i] - voltageList[i-1] >= 0):

if (side == -1):

side = 1

elif (side == 0):

consOpp = consOpp + 1

if (consOpp >= 5): #account for noise causing fluctuations

if (start == -1):

start = i-1 - consOpp

side = 1

else:

end = i-1 - consOpp

break

else:

consOpp = 0

else:

if (side == -1):

side = 0

elif (side == 1):

consOpp = consOpp + 1

if (consOpp >= 5): #account for noise causing fluctuations

if (start == -1):

start = i-1 - consOpp

side = 0

else:

end = i-1 - consOpp

break

else:

consOpp = 0

for i in range(start+1, end): #use the indices to store the change in voltage between each point

Edge.append(voltageList[i-1]-voltageList[i])

sum = 0

#obtain average change between points

for i in range(len(Edge)):

sum = sum + Edge[i]

IncrAvg = sum / len(Edge)

#count the number of changes in two consecutive points that equal the average for the edge

nIncrAvg = 0

tol = (max-min) \* 0.0075 #tolerance to account for noise (experimentally determined)

for i in range(len(Edge)):

if abs(IncrAvg - Edge[i]) < tol:

nIncrAvg = nIncrAvg + 1

if nIncrAvg > len(Edge) \* 0.5: #if 50% (experimentally determined) of the increases were equal to the average it is a triangle

print("Triangle")

sinTriangleFreq(voltageList, timeList, min, max)

else:

print("Sin")

sinTriangleFreq(voltageList, timeList, min, max)

def squareFreq(voltageList, timeList, min, max):

length = len(voltageList)

tol = (max-min) \* 0.2

maxStart = -1

maxFinish = -1

minStart = -1

minFinish = -1

lastExtreme = -1 #-1 for no last extreme, 0 for min, 1 for max

for i in range (length): #find a series of consecutive maxes and mins - the longer of which is half a cycle

if max - voltageList[i] < tol:

if lastExtreme == 0:

minFinish = i-1

if maxStart != -1:

break

lastExtreme = 1

if maxStart == -1:

maxStart = i

elif voltageList[i] - min < tol:

if lastExtreme == 1:

maxFinish = i-1

if minStart != -1:

break

lastExtreme = 0

if minStart == -1:

minStart = i

maxTime = timeList[maxFinish] - timeList[maxStart]

minTime = timeList[minFinish] - timeList[minStart]

if maxTime > minTime:

freq = 1.0 / 2.0 / maxTime

else:

freq = 1.0 / 2.0 / minTime

print(str(freq) + "Hz")

def sinTriangleFreq (voltageList, timeList, min, max):

length = len(voltageList)

tol= (max-min) \* 0.05

min1 = -1

min2 = -1

maxSeen = False

for i in range (length): #find 2 minimum points, a maximum must be inbetween - indicates one full cycle

if voltageList[i] - min < tol:

minSeen = True

if min1 == -1:

min1 = i

elif maxSeen:

min2 = i

break

elif max - voltageList[i] < tol:

if min1 == -1:

continue

maxSeen = True

freq = 1 / (timeList[min2] - timeList[min1])

print(str(freq) + "Hz")

while True:

rawVoltageList, smoothedVoltageList, timeList = gatherData()

characterizeWaveform(rawVoltageList, smoothedVoltageList, timeList)