Bobby Vielma CSCE 462

Alexia Perez 2/5/2021

Lab 2 Report

1. What is the highest frequency that your Raspberry Pi can generate using digital output?

Why is this the case?

* The hardware can generate up to 19.2 MHz while the software can generate from 1Hz to a few KHz.

1. Does the signal with the highest frequency stay steady or fluctuate? If not and there is

noise, where does the noise come from?

* For the most part it is steady. There was some interference when physically touching the cable.
* However, there was always some small level of noise affecting the signal, which became more apparent when working with the higher frequencies.
* This is due to the fact that higher frequency waves are less stable, allowing for the hardware to cause more apparent noise, while at lower frequencies, the wave’s stability counteracts the noise better. It can be lessened depending on the techniques used, probing type, and oscilloscope settings.

1. How can you convert a digital PWM (Pulse width modulation) signal into an analog

signal. (e.g. possible circuit design, software conversion)

* In order to convert a digital signal into an analog signal using a PWM, we must use an analog low-

pass filter.

* The easiest implementation of a low-pass filter is a first order, passive RC circuit.
* An advantage of using the PWM method combined with an RC low-pass filter is that the Digital-to-Analog conversion can be configured by software.

1. What is the maximum frequency you can produce with your Raspberry Pi functional

generator on different wave shapes?

* In the report it stated that the maximum frequency should be 20Hz but in practice we were only able to get 10-15Hz for the square wave, 25Hz for the triangular wave, and 25.2Hz for the sin wave.
* We stopped seeing an increase in the frequency reading when we asked the program to request 630, 500, and 600 Hz for the above waves respectively. This is likely due to the fact that at these high frequencies the sleep time is no longer the limiting step, it is the code’s own execution time.

1. Explain your design for the functional generator (you can use diagrams to visualize your

system state machine, what function you implemented, etc.)

* **Square:** For making the square generator we simply set the voltage to the passed in voltage for a time of 1/frequency/2. Then set the voltage to 0 for the same amount of time.
* **Triangle:** For the triangle we first have to get the voltage per step, for a total of 25 steps. We then have two for-loops: one for the positive slope and one for the negative slope. For the first loop we start at voltage 0 and increase it every iteration by the voltage per step, adding to itself and then sleep for 1/frequency/50. For the second loop we do the process in reverse, starting at the max voltage and decreasing it every iteration by the voltage per step until we reach zero, sleeping for 1/frequency/50.
* **Sine:** To create the sine waves we first have to get the voltage per step for each iteration. We calculated this as 2pi/25, 25 being the number of steps. We then set the dac voltage to the voltage/2 + voltage/2 \* math.sin(temp), with temp starting at 0 and adding the voltage per step every iteration to it. Then we sleep for 1/frequency/50.

**Code**

from gpiozero import Button

import Adafruit\_MCP4725

import RPi.GPIO as GPIO

import math

import time

GPIO.setmode(GPIO.BCM)

dac = Adafruit\_MCP4725.MCP4725()

button = Button(4)

try:

while True:

if button.is\_pressed:

shape = input("Enter 1 for a square wave, 2 for a triangle wave, and 3 for a sin wave: ")

f = float(input("Enter the desired frequency (Hz, max of 20): "))

v = int(input("Enter the desired maximum output voltage (mV): "))

if shape == 1: #square

while not button.is\_pressed:

dac.set\_voltage(v)

time.sleep(1/f/2)

dac.set\_voltage(0)

time.sleep(1/f/2)

elif shape == 2: #triangle

while not button.is\_pressed:

temp = 0

numSteps = 25

stepSize = v/numSteps #fixed amount of steps per voltage input

for i in range (numSteps):

dac.set\_voltage(temp)

temp += stepSize

time.sleep(1/f/50)

temp = v

for j in range (numSteps):

dac.set\_voltage(temp)

temp -= stepSize

time.sleep(1/f/50)

elif shape == 3: #sin

while not button.is\_pressed:

temp = 0

numSteps = 50

stepSize = 2 \* math.pi/numSteps

for i in range (numSteps):

dac.set\_voltage(v/2+ int(v/2\*math.sin(temp)))

temp += stepSize

time.sleep(1/f/50)

else:

print ("Invalid input")

except KeyboardInterrupt:

GPIO.cleanup()