Alexander Akomer (Partner: Shaheen Ebrahimi)

CSCE 462 – 500 Liu

2/11/2022

Lab 2 Report

1) **What is the highest frequency that your Raspberry Pi can generate using digital output? Why is this the case?**
   The Raspberry Pi digital output can produce frequencies of up to 19.2 MHz. This is because the Raspberry Pi clock frequency caps at 19.2 MHz and the code we have written can operate on the order of constant time – therefore, the digital output should theoretically cap at the clock frequency.

2) **Does the signal with the highest frequency stay steady or fluctuate? If not and there is noise, where does the noise come from?**
   The signal with the highest frequency tends to fluctuate slightly. The noise in these signals comes from a lack of accuracy in calculations; by using a smaller step size (or introducing a fast Fourier transform), this noise can be almost entirely eliminated.

3) **How can you convert a digital PWM (Pulse width modulation) signal into an analog signal? (e.g. possible circuit design, software conversion)**
   A digital PWM signal can be converted into an analog signal using a low-pass filter. Therefore, the very basic requirement for conversion to analog signal is the use of a resistor and capacitor in conjunction with the PWM. Software can also replicate this by simulating the result of a low-pass filter and joining it with the given PWM.

4) **What is the maximum frequency you can produce with your Raspberry Pi functional generator on different wave shapes?**
   The Raspberry Pi functional generator can produce frequencies of up to roughly 1 MHz. This is tested through our code by varying the sample rate at which the output frequency is calculated. Using the oscilloscope and checking against different frequencies (while shortening the sample rate by a factor of 10 with each test), frequencies continued to remain stable up until the sampling rate reached a frequency of roughly 1 MHz. Beyond this, signals began to become unstable.

5) **Explain your design for the functional generator (you can use diagrams to visualize your system state machine, what function you implemented, etc.)**
   The functional generator was comprised of several modular functions. In the main function, a setup_GPIO function was called to setup the GPIO layout, including the port for the button. The rest of the main method polls for event detection of the button press, which called a logic function. The logic function asked for several inputs, of which responses led to respective functions for the square, triangle, or sine wave. A helper method was created (plot) to display graphical outputs while we had no oscilloscope to test with. Polling continues to allow the logic cycle to continue.

Code:

```
import time
import math
import board
import busio
import matplotlib.pyplot as plt
import adafruit_mcp4725
import RPi.GPIO as GPIO
import plotly.express as px
button_io = 26
dac = adafruit_mcp4725.MCP4725(busio.I2C(board.SCL, board.SDA))

def setup_GPIO():
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(button_io, GPIO.IN)

def set_output(volt_out, vcc=3.3):
    dac.raw_value = int(4095/vcc*volt_out)

def square_wave(period, volt_max):
    t = 0.0
    t_step = 0.0001
    zero = True
    while True:
        start = time.time()
        output = 0
        if (t % period <= period/2):
            if (not zero):
                output = volt_max
        else:
            t %= period/2
            zero = not zero

        set_output(output)
        elapsed = time.time() - start
        t += t_step + elapsed
        time.sleep(t_step)

        if not GPIO.input(button_io):
            return

def triangle_wave(period, volt_max):
    t = 0.0
    t_step = 0.0001
    pos = True
    while True:
        start = time.time()
        if (t % period <= period/2):
            if (pos):
                output = t * volt_max / (period/2)
            else:
                output = volt_max - (t * volt_max / (period/2))
        else:
```

```
            t %= period/2
            pos = not pos
        set_output(output)
        elapsed = time.time() - start
        t += t_step + elapsed
        time.sleep(t_step)

        if not GPIO.input(button_io):
            return

def sine_wave(frequency, volt_max):
    t = 0
    t_step = 0.0001
    while True:
        start = time.time()
        output = (volt_max/2)*(1+math.sin(2*math.pi*frequency*t))
        set_output(output)
        elapsed = time.time() - start
        t += t_step + elapsed
        time.sleep(t_step)

        if not GPIO.input(button_io):
            return

def logic():
    while True:
        wave = input("Enter wave (square, triangle, sine): ")
        if (not(wave == "square" or wave == "triangle" or wave == "sine" or wave == "exit")):
            print("Incorrect input, try again")
            continue
        frequency = input("Enter frequency: ")
        voltage = input("Enter max output voltage: ")
        if (wave == "square"):
            square_wave(1/*float(frequency), float(voltage))
            #plot(t,y)
        elif (wave == "triangle"):
            triangle_wave(1/*float(frequency), float(voltage))
            #plot(t,y)
        elif (wave == "sine"):
            sine_wave(float(frequency), float(voltage))
            #plot(t,y)
        elif (wave == "exit"):
            return

def plot(t_vals, y_vals):
    fig = px.line(x=t_vals, y=y_vals, title='Wave Gen Function')
    fig.show()

def main():
    setup_GPIO()
    while True:
        if not GPIO.input(button_io):
            logic()

if __name__ == "__main__":
    main()
```