Lab 4: Use of Digital to Analog converter along with Microcontroller Report

Tyler Nardecchia
Yerania Hernandez
CSCE 462 - 502
October 13th, 2017

Thinking and Exploring

1) *Find out the highest frequency that a raspberry pi can generate using digital output and write down the answer and explanation in your lab report.*

   The highest frequency that a Raspberry Pi can generate using digital output and write will depend based on the language and the specific library used. This is due to the speed difference when running interpreted code compared to compiled code. In this case we implemented the square wave in C and used the wiringPi library, which would ideally result in 4.1 to 4.6 MHz.

2) *Does the highest reading steadily stay at 5V? If not and there is noise, where does the noise come from?*

   Although the reading of the generated wave is able to represent a square wave and demonstrates an approximate 5 V voltage at the highest reading, it does not steadily hold on to this voltage. The result ends up demonstrating a jitter or noise that is present at the high and low reading of voltages for the generated wave. This jitter will exist due to the Raspberry Pi running on Linux OS, which ia multi-tasking operating system. Although it might seem as if the cursor is functioning while the program is running, Linux OS is actually only providing time-slices to a program and to the drivers. It happens so quickly that it is hard to actually notice, but this is all possible because the OS is able to service other tasks after taking away the control the program has for a short time.

3) *Explore and write down how you can convert a digital PWM (Pulse width modulation) into analog signal.*

   One of the ways to convert PWM to an analog signal is through hardware components, such as connecting an ADC chip to the board through serial peripheral interface (SPI) or even I2C. Another hardware method in combination with software is actually implementing a low-pass filter for the PWM signal. Creating a basic single-pole RC filter would work if you don't mind the output to have a jitter and ripple effect. However to improve this filter we can another RC filter in series in order to add another pole with a steeper cut-off frequency. As a result of these two RC filters, the PWM output signal above the cut-off frequency will be filtered out and will give us a voltage that is proportional to the duty cycle. For this method, you obviously will need resistors, capacitors, and maybe even a comparator in order to compare it to a target analog voltage. Another method could also be interfacing the Arduino and Raspberry Pi 3 by making use of the Arduino development kit. This will allow us to make use of the real-time response between the Arduino board by providing high precisions and low jitter. We could use the pySerial library in order control the Arduino's I/Os through the RaspberryPi, allowing us to read and send data back and forth.

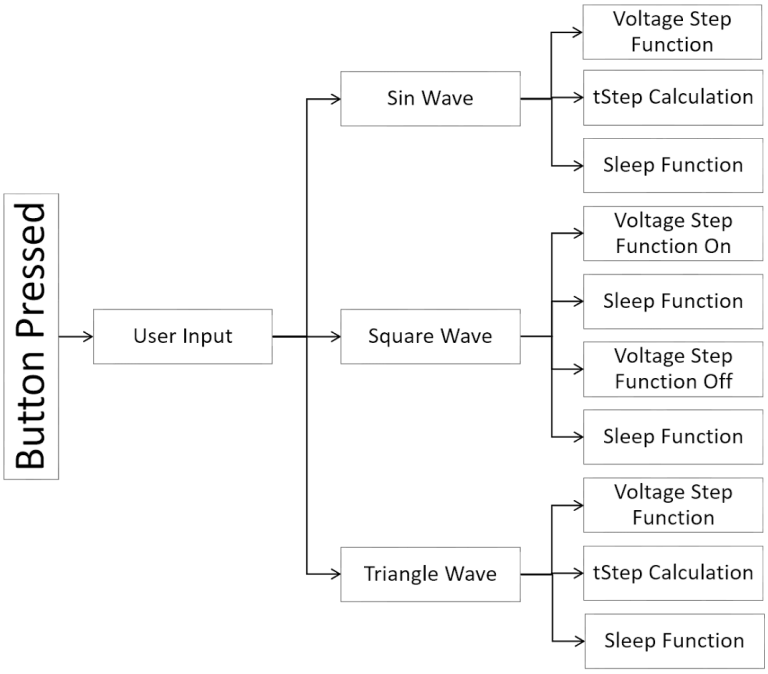
4) *What is the max frequency you can produce by your raspberry pi functional generator? Write it down in your report.*

In order to determine the maximum frequency that can be produced by our function generator, we reduced the amount of operations performed in our code to be as efficient as possible while still being capable of producing a square wave (which is the simplest wave to implement). This involves incorporating a loop whose only operation is to repeatedly write a 1 and then a 0 to a GPIO pin with no waiting in between, thus creating a high-frequency square wave. The resultant maximum frequency on the oscilloscope came out to be **353 kHz** using Python as our coding language.

5) *Explain your design for the functional generator (you can use diagram to visualize your system state machine, what function you implemented, etc)*

The design behind the function generator is to implement three separate functions; one for each shape of wave that the user is allowed to output (square, sinusoidal, and triangular). Each of these functions take inputs for both the maximum voltage as well as the maximum frequency. A step size is used for each function in order to provide a time interval for which the output voltage will need to be recalculated. Each shape has their own specific calculation that has to be made in accordance with the voltage and frequency parameters at every "tStep" interval. In between these short intervals, the system sleeps for a particular amount of time.

Prior to the call of these functions, the system uses a while loop to wait for a button to be pressed. Once the button is pressed, the system then uses "input" functions to prompt the user to specify the shape of the wave, maximum voltage, and frequency for which they want the respective GPIO to output. Once these three parameters have been entered, the correspondent shape function will be called, passing the other two arguments into it.



```python
import time
import math
import RPi.GPIO as IO
# Import the MCP4725 module.
import Adafruit_MCP4725

# Create a DAC instance.
dac = Adafruit_MCP4725.MCP4725()

IO.setmode(IO.BOARD)
IO.setup(15, IO.IN) #Setup input from button

#Setup for sin wave with voltage and frequency input
def sin_wave(volt_input, freq_input):
    t = 0.0
    tStep = 0.0005
    scale = 4096/3.3
    while True:
        voltage = scale * ((volt_input - 1) + math.sin(freq_input*t*(50/3.0)))
        dac.set_voltage(int(voltage))
        t += tStep
        time.sleep(0.0005)

#Setup for square wave with voltage and frequency input
def square_wave(volt_input, freq_input):
    while True:
        voltage = volt_input * (4096/3.3)
        dac.set_voltage(int(voltage))
        time.sleep((1.0/freq_input)/2.0)
        voltage = 0
        dac.set_voltage(int(voltage))
        time.sleep((1.0/freq_input)/2.0)

#Setup for triangle wave with voltage and frequency input
def triangle_wave(volt_input, freq_input):
    tStep = 0.005
    scale = volt_input*(4096 / 3.3)
    slope = 2*volt_input*freq_input*(10/8.4)

    while True:
        voltage = 0
        while voltage<volt_input:
            voltage += slope * tStep
            dac.set_voltage(int(voltage*(4096/3.3)))
            time.sleep(0.005)
        while voltage >= 0:
            voltage -= slope * tStep
            dac.set_voltage(int(voltage*(4096/3.3)))
            time.sleep(0.005)

#Waiting for button input to call on the specific waveform
while True:
    if(IO.input(15)):
        shape_input = input("Shape of Waveform: ")
        volt_input = input("Max output voltage: ")
        freq_input = input("Frequency: ")

        if (shape_input == 'sin'):
            sin_wave(volt_input, freq_input)
        elif (shape_input == 'triangle'):
            triangle_wave(volt_input, freq_input)
        elif (shape_input == 'square'):
            square_wave(volt_input, freq_input)
        else:
            print "Wrong shape input"

```