PHYC40970- Advanced Laboratory II

Electronics II

Nathan Power

19311361

Abstract

The aim of this project is to become familiar with the PWM techniques and their applications. The PWM output of a Raspberry Pi Pico and a general-purpose 555 IC will be investigated and then will be used to build a PWM controller for a motor.

Introduction

PWM is a way to control analog devices with a digital output. Using PWM you can output a modulating signal from a digital device to drive an analog device.PWM is not true analog output, however. PWM imitates an analog-type result by applying power in short bursts of regulated voltage

.

When the signal is high, we call this "On time". To describe the amount of On time , we use the concept of duty cycle. The percentage duty cycle describes the percentage of time a digital signal is on over an interval or period of time. This period is the inverse of the frequency of the waveform.

If a digital signal spends half of the time on and the other half off, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave. If the percentage is higher than 50%, the digital signal spends more time in the high state than the low state and vice versa if the duty cycle is less than 50%.

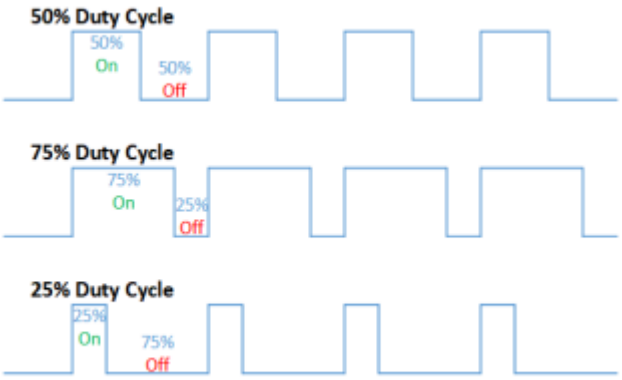


Fig.1 shows PWM outputs of different duty cycles [1]

An example would be to apply voltage to a motor for fractions of a second to make the motor move at the speed you wanted it to. Voltage is being applied and then removed many times in an interval, but what is produced is an analog-like response. The motor does not stop instantly when power is off due to inertia, and so by the time you re-apply power it has only

slowed a small fraction.This way you do not experience an abrupt stop in power if a motor is driven by PWM. [2]
The length of time that a pulse is in a given state is the width of a pulse wave.

A device that is driven by PWM ends up behaving like the average of the pulses.
Going back to our fan motor example, if we know that the high voltage is 24, the low is 0v, and the duty cycle is 50%, then we can determine the average voltage by multiplying the duty cycle by the pulse's high level. If you want the motor to go faster, you can drive the PWM output to a higher duty cycle. The higher the frequency of high pulses, the higher the average voltage and the faster the fan motor will spin.

You see now that PWM, duty cycle, and frequency are interrelated. We use duty cycle and frequency to describe the PWM, and we often talk about frequency in reference to speed. For example, a variable frequency drive motor produces a response like an analog device in the real world.

Duty Cycle x High Voltage Level = Average Voltage *Eq.1*

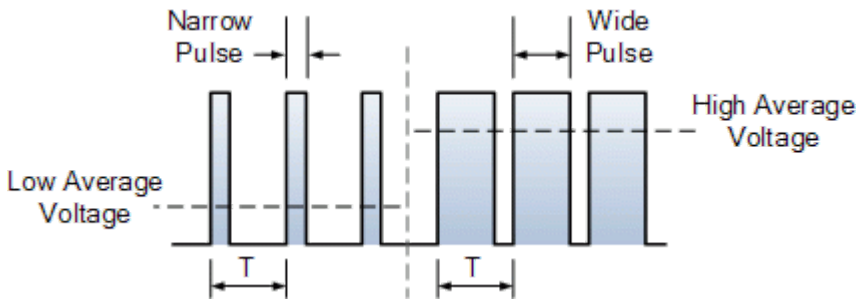


Fig.2 Shows the relation between Duty Cycle and Average Voltage [3]

The duty cycle can change to affect the average voltage that the motor experiences. The frequency of the cycles can increase. The pulse can even be increased in length. These can all happen together, too, but in general, it's easier to think of as either duty cycle increasing or frequency increases to increase the speed of the motor.

Task 1: PI Pico PWM Exercises:

a) Basics:

I.Verify that the PWM output of the Pico looks as expected.

By using the The Raspberry PI Pico micro-controller which produces PWM, we can observe the effects of changing the frequency and duty cycle of our output voltage.

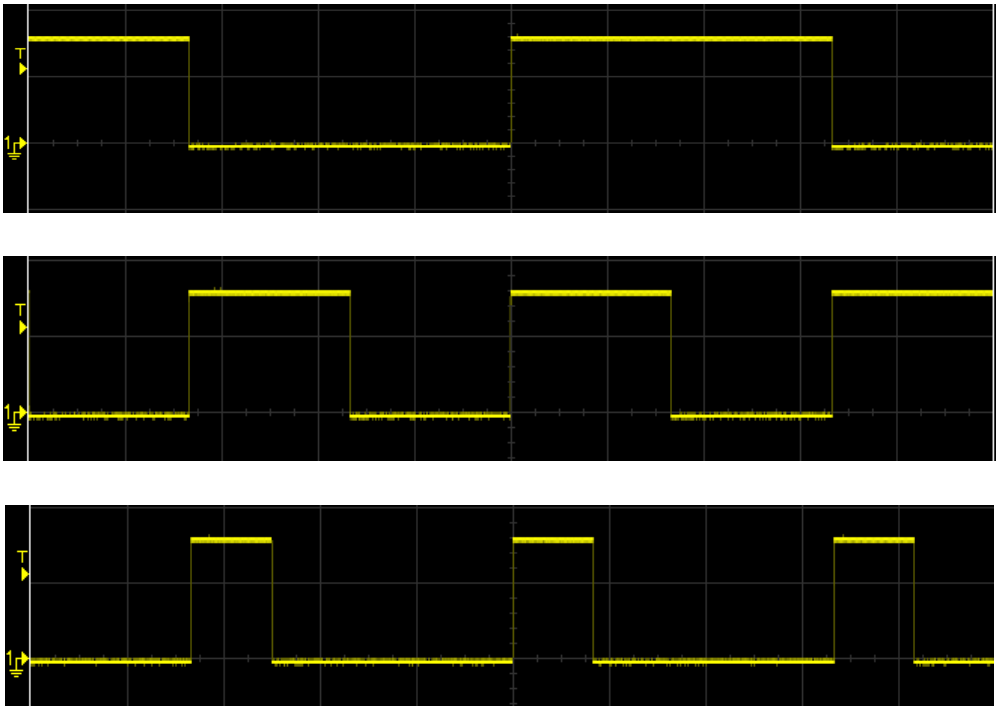


Fig.3 shows three different examples of PWM output from the Pico microcontroller. Each vertical division is 2V and the horizontal 300µs

(a) Corresponds to a frequency of 500Hz with a 50% duty cycle.

(b) Corresponds to a frequency of 1000Hz with a 50% duty cycle.

(c) Corresponds to a frequency of 1000Hz with a 25% duty cycle.

As we can see from the above Fig.3, by changing the frequency and keeping the duty cycle of the PWM constant we increase the rate of the pulses but the pulse remains the same. Alternately, by changing the duty cycle and keeping the frequency constant we see the pulse width change but not the pulse rate. This explains why both increasing frequency and increasing duty cycle would increase the speed of a motor, as both separately increase the time of which the motor is on but for different reasons.

II. Control the brightness of an LED from your computer by specifying a brightness between 0 (off) and 1 (full brightness). You can use the green LED on the Pico board which is on GPIO Pin 25 or an external LED (remember to use a 330 Ω current-limiting resistor as well). No visible flickering should be visible for a given brightness setting. Explain your choice of PWM frequency.

By activating Pin25 on the microcontroller, the LED can be turned on. As mentioned before, the changing of the duty cycle of PWM causes the average output voltage to increase or decrease. In this example, the average output voltage controls the brightness of the LED. The Pico microcontroller's duty cycle works between the ranges of 0(off) and 65535(100%). Therefore, changing the brightness of the LED is simply done by taking discrete fractions of 65535 e.g 65535//4 would correspond to 25% brightness of the LED. By choosing a high enough PWM frequency the flickering of the light can be eliminated to the eye. As the frequency of the PWM controls the rate of the light pulses, and the fact that the human eye cannot perceive flickering above 60hz for digital displays anything above this would suffice. This begs the question, are 120Hz monitors really necessary if the eye can only perceive up to 60Hz?

III. Make the LED brightness "linearly" brighten and fade with a frequency of approximately 0.25 Hz using 100 points over each cycle. Does the variation look linear to the eye? Note: the eye has a logarithmic response to changes in light intensity so consider using logarithmic steps for the brightness changes and visually compare to the linear steps.

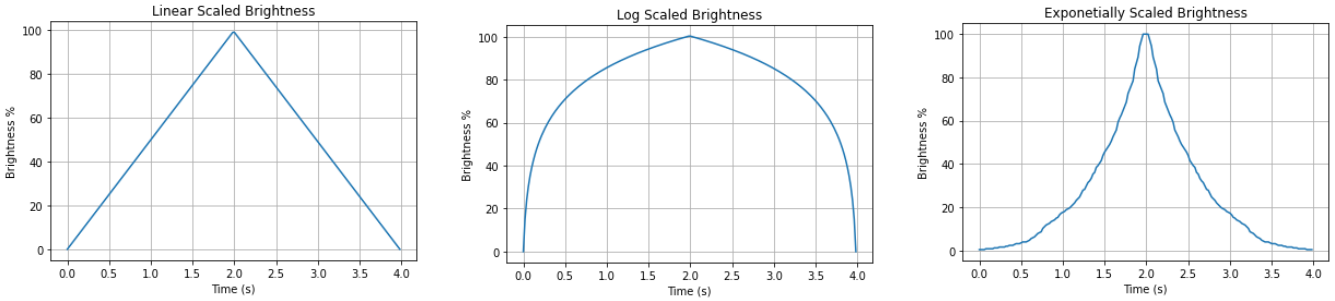


Fig.4 shows three different examples of PWM output from the Pico microcontroller.
(a) Corresponds to a linear scale.
(b) Corresponds to a log scale.
(c) Corresponds to an exponential.

By alternating the duty cycle of the PWM at discrete steps it is possible to produce different waveforms such as sine waves and triangular waves. This way the rate at which the LED brightens (and dims) can be controlled and altered to whatever shape desired. However when viewing the LED gets brighter and dimmer in a linear scale, the light doesn't appear to follow the linear pattern. It appears, to the eye, to take longer to get to the maximum and then stays at the max for a longer period. Similarly, the exponential scale does not appear to be an even function to the eye either. However when the LED is controlled by the logarithmic scale it seems to be an even and smooth climb and descent to and from the LED's maximum brightness. The reason for this may be that the eye has a logarithmic response to changes in light intensity. When we measure sound intensity, we use a

logarithmic scale because the human ear responds to sound pressure levels in a logarithmic way. Taking this into account, we may begin to suspect it is not the eye or ear that reacts this way but may be the brain. The brain, not the eye, may do the seeing.
The eye transmits information to the brain just like the ear. Which may explain the desire some people may have for better spec monitors. Different people have different tolerances for flicker due to the speed of their nervous systems. Some people may be able to hear higher frequency sounds than others as well as higher flicker rates of lights.

(b) Simple DAC:

Create a simple DAC (Digital to Analog) converter by supplying the PWM signal to a low pass filter (i.e. the analogue output is the averaged/smoothed PWM signal).

The DAC voltage observed at the output of the low-pass filter is determined by just two parameters, the duty cycle and the PWM signal's high voltage. In the frequency domain, a low-pass filter suppresses higher-frequency components of an input signal. The time-domain equivalent of this effect is smoothing, or averaging. By low-pass filtering a PWM signal we are extracting its average value. The low pass filter only allows low frequency signals from 0Hz to its cut-off frequency, point to pass while blocking those any higher.
The cut-off frequency of a RC filter is obtained from the well known equation,

$$f_{cut-off} = \frac{1}{2\pi RC} \qquad \textit{Eq.2}$$

The time constant of a series RC circuit is defined as the time taken by the capacitor to charge up to 63.2% of the final steady state value and also it is defined as the time taken by the capacitor to discharge to 36.8% of steady state value. It is given by the product of the R and C values of the low pass filter.

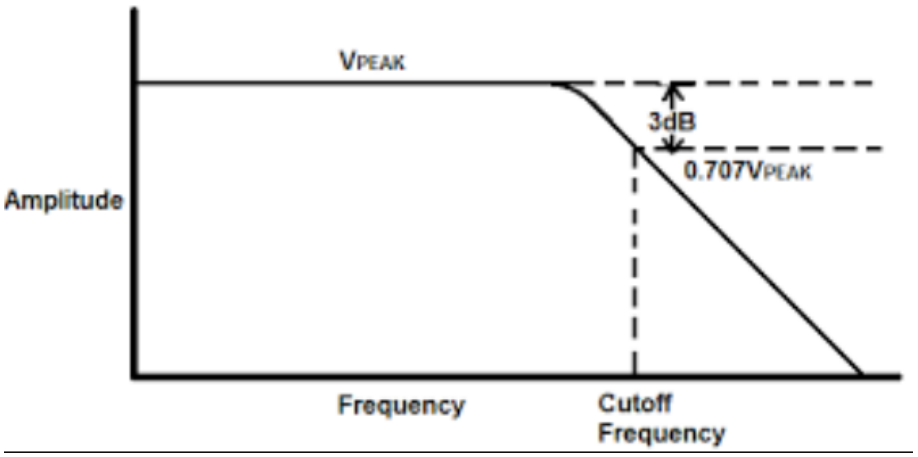


Fig.5 shows the attenuation of amplitude of waves passing through a RC low-pass filter. [5]

I. How does the choice of the R and C values combined with the PWM frequency impact on the smoothness of the output of the DAC and what are the implications for producing high frequency signals with the DAC? Note: in this case it is useful to consider the charging and discharging of the capacitor.

When constructing my low pass filter, I picked the values of C = 1μF and R = 47kΩ to demonstrate the effect changing frequency had on the output of the DAC. This gave me a cut-off frequency of 3.38 Hz. Changing my R and C values would have given me a different frequency cut-off point. As we can see from the figure below, Fig.6, the frequency of the PWM climbs higher and higher past the cut-off frequency and the DAC output's ripples become smaller and smaller. This is because the capacitor has less time to charge and discharge meaning the capacitor never gets to fully charge. The charging and discharging times get smaller therefore the output starts to focus around the average voltage level.
At higher and higher frequencies, the capacitive reactance approaches zero, making a capacitor behave like a wire.The downside of having a long time constant however, is the slow response of the output voltage to changes in the PWM duty cycle. Also at higher frequencies, any jitter or timing errors within the DAC will be more noticeable and cause more of an effect.

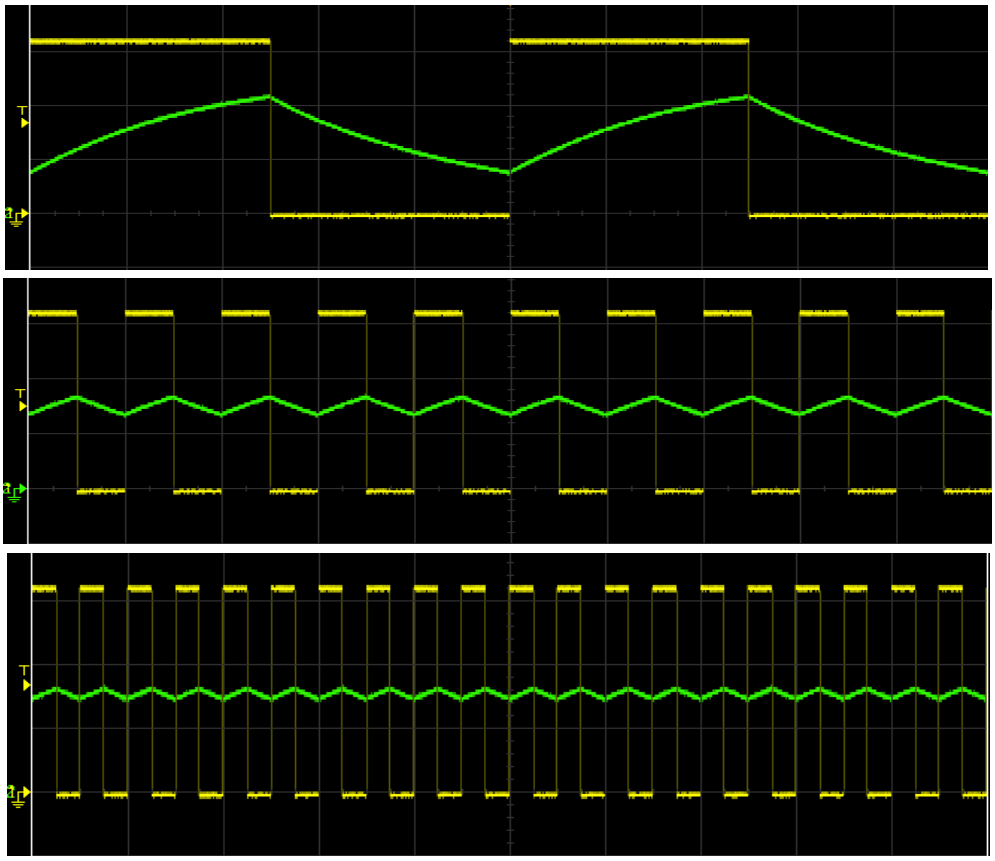


Fig.6 shows three different examples of PWM output with 50% duty cycle from the Pico microcontroller. Each vertical division is 1V and the horizontal 20ms
(a)Corresponds to a frequency of 10Hz.
(b)Corresponds to a frequency of 50Hz.
(c)Corresponds to a frequency of 100Hz.

II. Determine an R and C combination so that the signal has less than 5% variation for a 50% duty cycle of a PWM frequency of 20 kHz. Make sure to include any calculations in your report. Verify the performance by measuring on an oscilloscope.

When choosing the RC time constant for the circuit, it is important to not overshoot the values as the downside of having a long time constant however, is the slow response of the output voltage to changes in the PWM duty cycle which becomes important in the next part of the report. We know that a higher RC constant will give us less variation from our average voltage. Therefore using the formula below we can calculate the amount of voltage ripple will occur in our circuit;[4]

$$V_{ripple} = \frac{T_{PWM}}{4RC} V_{high} \qquad \textit{Eq.3}$$

Using this equation, I came to the conclusion that for my circuit with the PWM frequency of 20k Hz and duty cycle of 50% an RC combination of 500Ω and 1μF would be ideal to give a peak to peak voltage ripple of 5%.

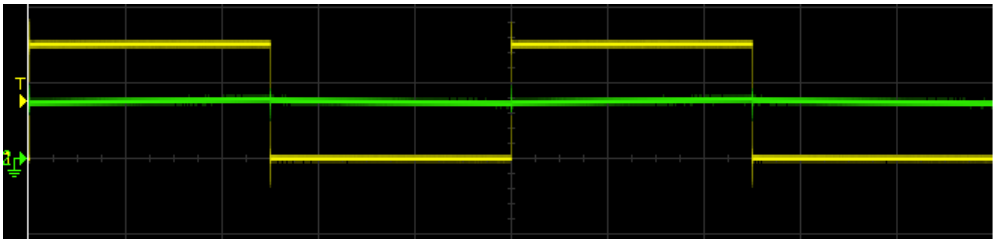


Fig.7 shows the PWM output (yellow) from the Pico microcontroller and the DAC output (green) through our RC low-pass filter. Each vertical division is 2V and the horizontal 100μs.

When the DAC output was observed on the oscilloscope and the peak to peak variation of the average voltage was taken, it was found that the voltage varied peak to peak by 140.75mV for an average voltage of 1.506 mV which corresponds to a 4.75% variation.

III. Explain why the output shouldn't be used unbuffered and how unity gain op-amp (voltage follower) can be used to buffer the output signal (there is no need to build the voltage follower).

The DAC may be sensitive to any loads put on it, so trying to drive something directly from the DAC may distort the signal heavily. A load can be anything such as a resistor, speaker, or motor. This sensitivity to load means the signal may change if too much or too little of a load is seen at the output of the DAC. An op amp circuit is a circuit with a very high input impedance. This high input impedance is the reason unity gain buffers are used.

If the load is low impedance, the load demands and draws a huge amount of current. If a load has very low resistance, it draws huge amounts of current. This causes huge amounts of power to be drawn by the power source and, because of this, causes high disturbances and use of the power source powering the load.

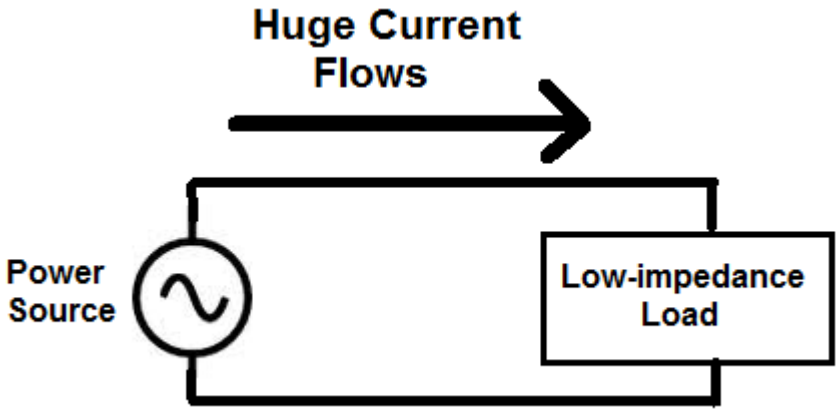


Fig.8 shows a circuit connected to a load of low-impedance [6]

When a circuit has a very high input impedance, very little current is drawn from the circuit. Thus, the power of the circuit isn't affected when current is feeding a high impedance load. Because the op amp has such a high impedance, it draws very little current. And because an op amp that has no feedback resistors gives the same output, the circuit outputs the same signal that is fed in.The output of the buffer can then be connected to anything within its own limitations without affecting the output from the DAC.

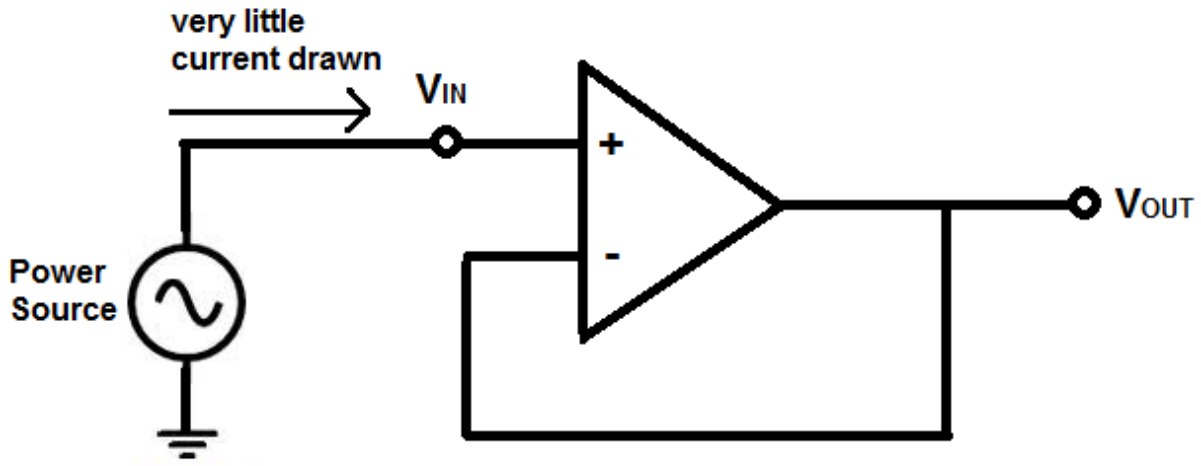


Fig.9 shows a circuit connected to a load of low-impedance [6]

c) Generating analogue output functions:

I. Generate a triangular signal and display it on an oscilloscope with the period of the signal set by a variable in your program. Verify with triangular waves of period 100 ms and 50 ms.

By alternating the duty cycle of the PWM at discrete steps it is possible to produce different waveforms such as sine waves and triangular waves. I produced a triangular wave by

changing the duty cycle of the PWM linearly therefore producing a triangular waveform. It is also possible to change the output's period and frequency by alternating the code to change the time it takes to produce the wave. This way, it's possible to produce waves of desired frequency by changing the variable in the code accordingly.

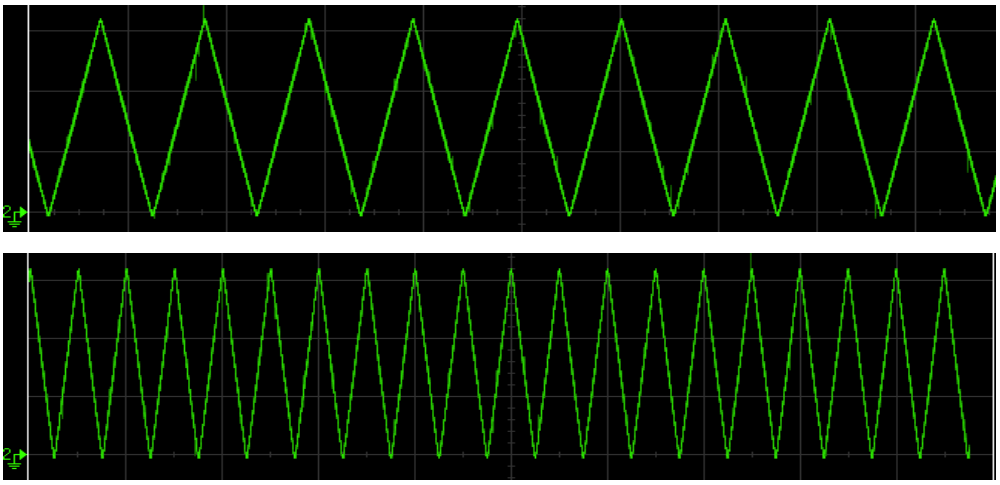


Fig.10 shows two different examples of triangular analogue output from the DAC. Each vertical division is 1V and the horizontal 100ms
(a)Corresponds to a period of 100ms.
(b)Corresponds to a period of 50ms.

II. Generate a sine wave where you can set its frequency (in Hz) in your code and record a few different frequencies on an oscilloscope, verifying the correct frequency is produced. Look at frequencies in the range 10 to 100 Hz. Note: What happens if you try to generate higher frequency sine waves? (Note: you might compare to the 3 dB "cutoff" frequency of the filter you chose).

By alternating the duty cycle of the PWM at steps corresponding to a sine wave I produced the waveforms seen below in Fig.10. As seen below this technique produced smooth accurate sine waves at frequencies between 10 and 100 Hz.

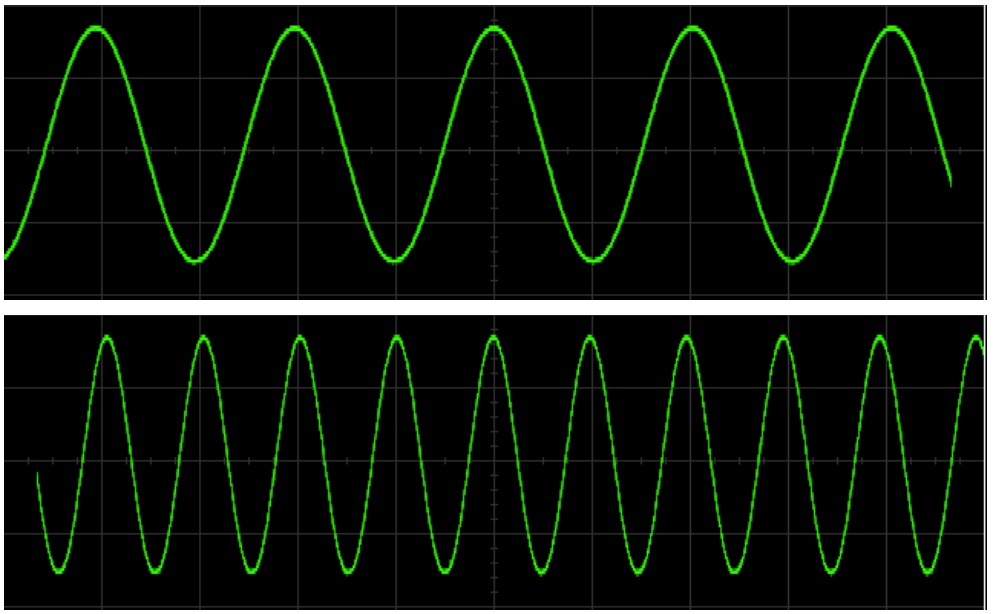


Fig.11 shows two different examples of sine wave analogue output from the DAC. Each vertical division is 1V and the horizontal 50ms
(a)Corresponds to a period of 100ms.
(b)Corresponds to a period of 50ms.

However, when producing high frequency sine waves the output no longer looks smooth and the peak to peak voltage has been attenuated. The low pass filter of our circuit with an RC combination of 500Ω and 1μF gives a cut-off frequency of 318 Ω. That is, the frequency above which the output voltage falls below 70.7% of the input voltage. The further the frequency rises above this cut-off point, the more the wave's amplitude will decrease as seen in Fig.5. The wave appears to be less smooth due to the sampling rate and amount of samples being taken.

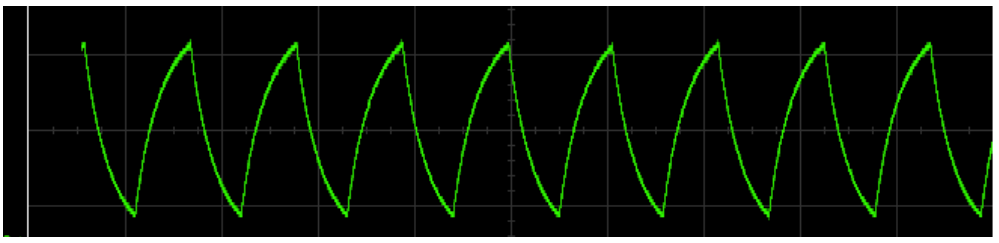


Fig.10 shows two different examples of sine wave analogue output from the DAC with a period of a period of 5ms.

Task 2, 555 timer:

(A) Simulate the 555 Timer as an Astable Oscillator using TINA circuit to become familiar with its use as an oscillator. Use your simulations to understand what controls the frequency and duty cycle of the output.

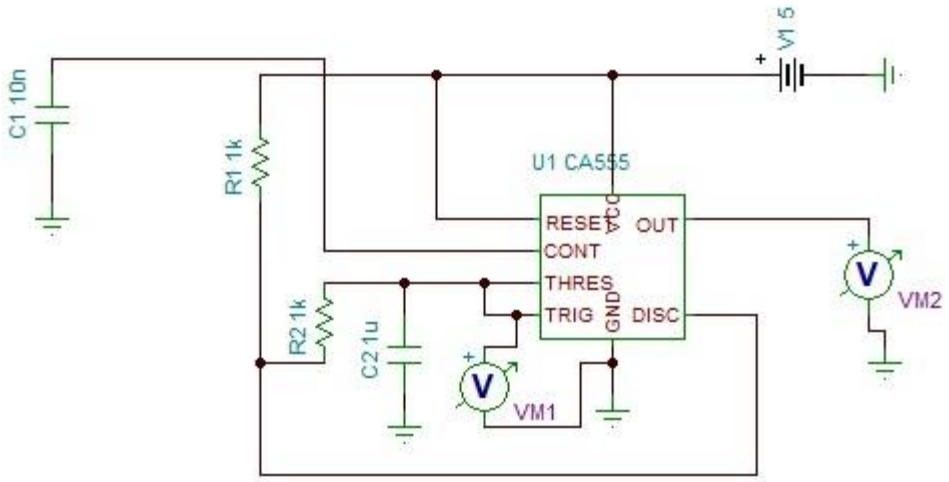


Fig.11 shows the circuit diagram for the simulation for the 555 Timer used as an Astable Oscillator.

In the 555 Oscillator circuit above, pin 2 and pin 6 are connected together allowing the circuit to re-trigger itself allowing it to operate as a free running oscillator. During each cycle capacitor, C1 charges up through both timing resistors, R1 and R2 but discharges itself only through resistor R2 as the other side of R2 is connected to the discharge terminal, pin 7. The individual times required to complete one charge and discharge cycle of the output is given as:

Eq.4

$$t_1 = 0.693(R_1 + R_2).C$$

and

$$t_2 = 0.693 \times R_2 \times C$$

When connected as an astable multivibrator, the output from the 555 Oscillator will continue indefinitely charging and discharging between 2/3Vcc and 1/3Vcc until the power supply is removed

Eq.5

$$f = \frac{1}{T} = \frac{1.44}{(R_1 + 2R_2).C}$$

By altering the time constant of just one of the RC combinations, the duty cycle of the output waveform can be set and is given as the ratio of resistor R2 to resistor R1. The Duty Cycle for the 555 Oscillator is given by:

Eq.6

$$\text{Duty Cycle} = \frac{T_{ON}}{T_{OFF} + T_{ON}} = \frac{R_1 + R_2}{(R_1 + 2R_2)} \%$$

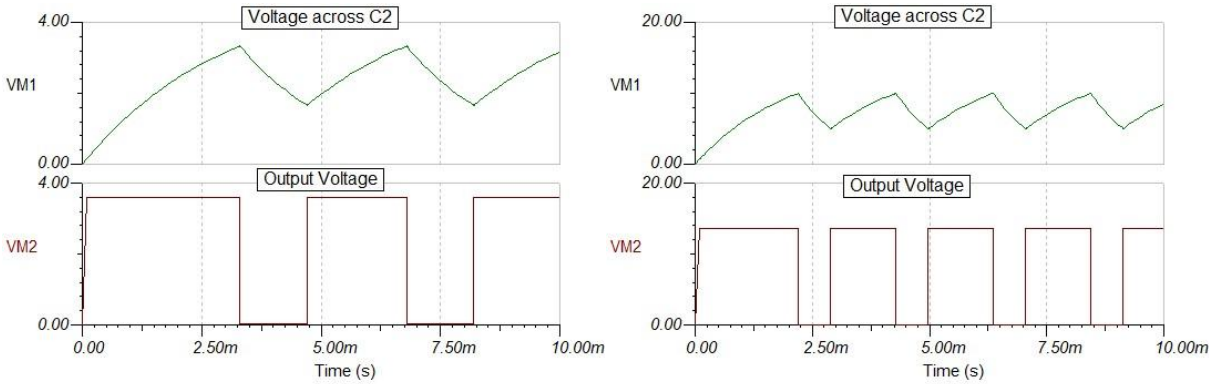


Fig.12 shows two different examples from the Astable Oscillator above.
(a)Corresponds to values of R1= 1kΩ R2= 2kΩ C2= 100nF.
(b)Corresponds to values of R1= 10kΩ R2= 10kΩ C2= 100nF.

As we can see from the figure above , Fig.12, by changing the ratio of the two resistors in the circuit, the duty cycle is then changed. In Fig.13, we can see that by changing the capacitor value we can change the length of the period. This can be achieved by changing the resistor values also. However by changing the capacitor value we can maintain the same duty cycle.

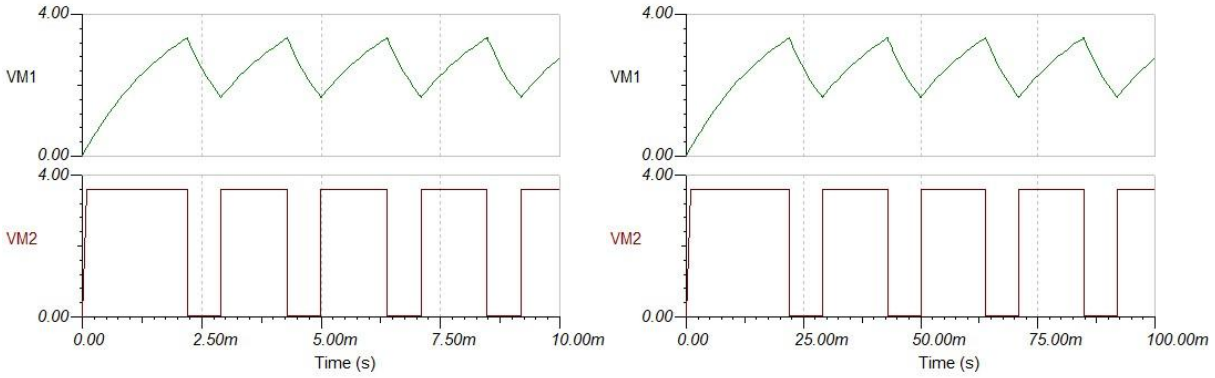


Fig.12 shows two different examples from the Astable Oscillator above.
(a)Corresponds to values of R1= 1kΩ R2= 1kΩ C2= 1uF.
(b)Corresponds to values of R1= 1kΩ R2= 1kΩ C2= 10uF.

(B) Simulate the 555 Timer as a PWM generator with the duty cycle controlled by a potentiometer. Use an oscilloscope to view the output and investigate the effect of changing the potentiometers value.

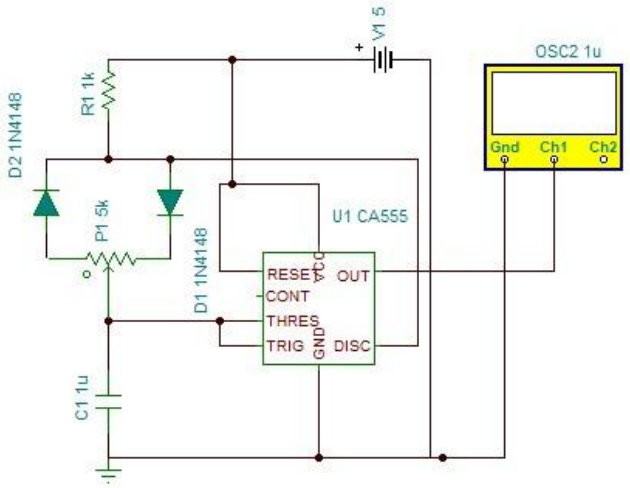


Fig.13 shows the circuit diagram for the simulation for the 555 Timer used as a PWM generator controlled by a potentiometer.

In the circuit in Part (A), as the timing capacitor, C charges through resistors R1 and R2 but only discharges through resistor R2 the output duty cycle can be varied between 50 and 100% by changing the value of resistor R2. By decreasing the value of R2 the duty cycle increases towards 100% and by increasing R2 the duty cycle reduces towards 50%. If the resistor R2 is very large relative to resistor R1 the output frequency of the 555 astable circuit will be determined by R2 x C only. [7]
This means that with this basic astable 555 oscillator the duty cycle will never go below 50% as the presence of resistor R2 prevents this. In other words we cannot make the outputs on time shorter than the off time, as (R1 + R2)C will always be greater than the value of R1 x C. One way to overcome this problem is to connect a signal bypassing diode in parallel with resistor R2 (now the potentiometer P1) as shown in this part. This way we can access duty

cycles below 50%. In this configuration the on time will depend on the resistor R1, the left side of the potentiometer and the capacitor C1, while the off time will depend on the capacitor C1 and the right side of the potentiometer. We can also notice that in this configuration the period of one cycle, thus the frequency, will always be the same, because the total resistance, while charging and discharging, will remain the same.

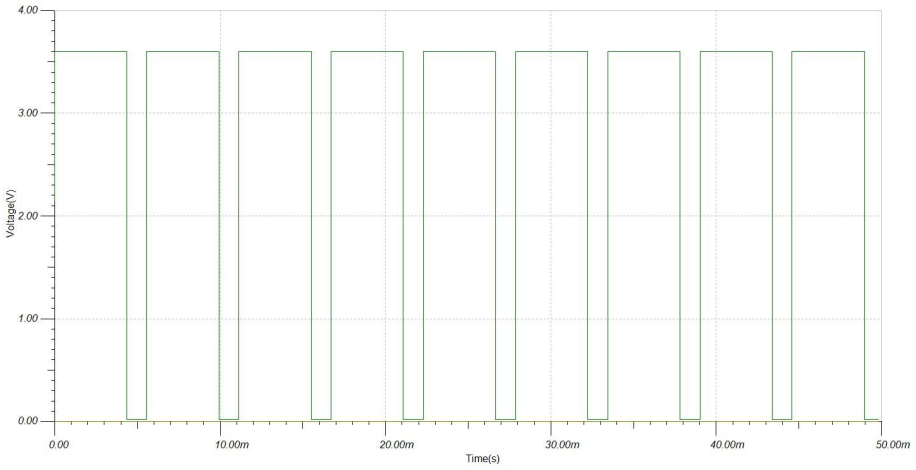
Fig.14 shows PWM outputs from the circuit above.

A. Corresponds to the potentiometer at 25%

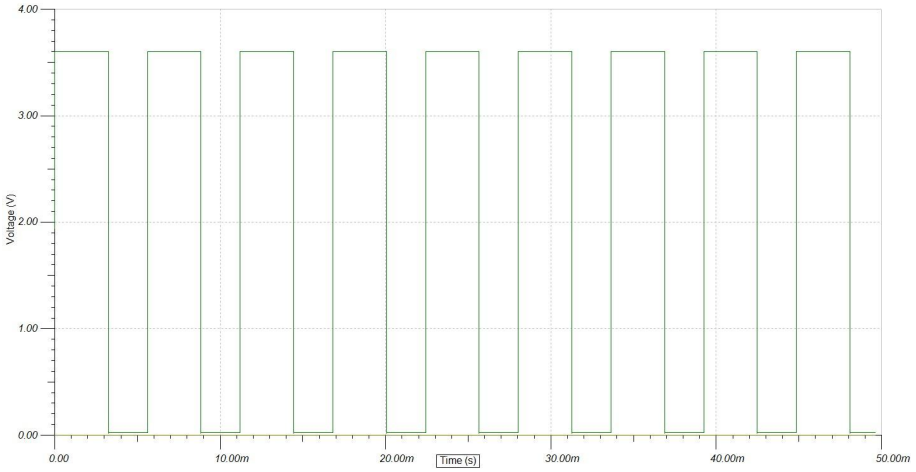
B. Corresponds to the potentiometer at 50%

C. Corresponds to the potentiometer at 75%

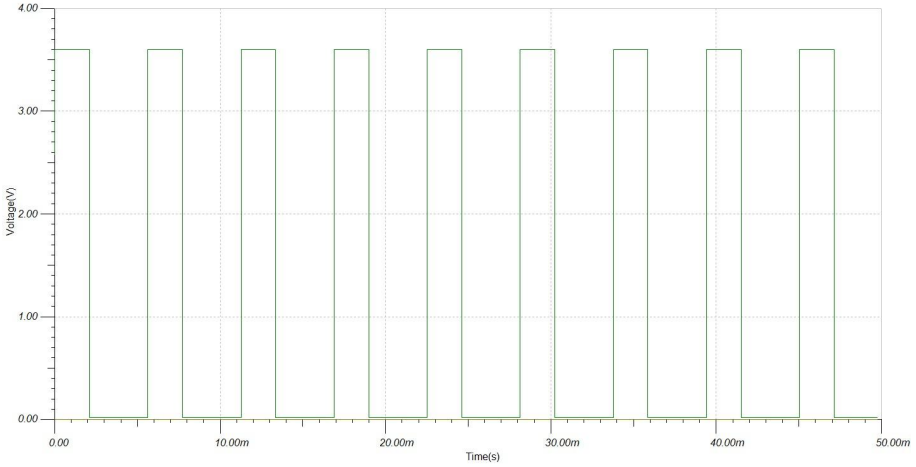
A.



B.



C.



(C) Build the circuit on a breadboard and use a DC motor as the load, by sending the PWM signal to the base of the transistor. Use 5 V from a bench power supply as VCC, source voltage, for the circuit.

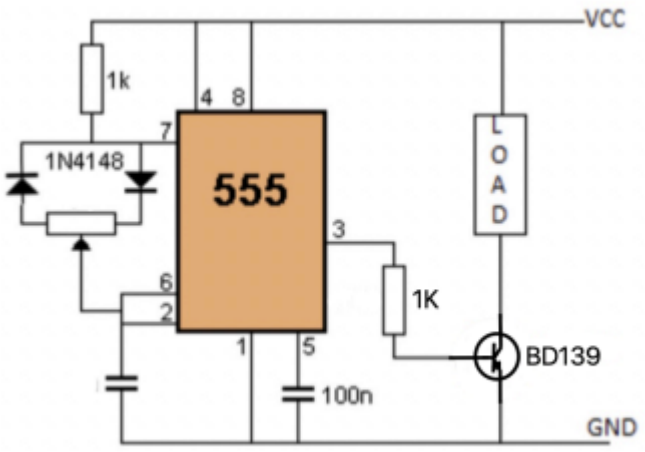


Fig.15 shows the circuit diagram for the 555 Timer PWM Motor Speed Controller Circuit

Usually the R1 resistance is much smaller than the resistance of the potentiometer, for example, 1K compared to 47K of the potentiometer that I chose to use in this circuit. In that way we have more control over the charging and discharging resistance in the circuit. The control pin of the 555 Timer is not used but it's connected to a capacitor in order to eliminate any external noise from that terminal. The output of the 555 timer can sink or source a current of 200mA to the load which is not sufficient to drive the motor. So if the motor that we want to control exceeds this rating we need to use a transistor for driving the motor. In this circuit, I used a BD139 transistor which can handle a current up to 1.5A. By changing the RC values and the duty cycle of my output I could change the speed of the motor.

A device that is driven by PWM ends up behaving like the average of the pulses.
If you want the motor to go faster, you can drive the PWM output to a higher duty cycle. The higher the frequency of high pulses, the higher the average voltage and the faster the motor will spin.
The output of the 555 timer needs to be connected to the base of the transistor through a resistor, and in my case I used a 1k resistor. For preventing any voltage spikes produced by the motor I also used a diode connected in parallel with the motor.

(D) Come up with another application for a 555 Timer chip and prototype it. It does not have to be PWM and can include other components!

My prototype using the 555 timer was a way to control the blinking of LED lights for uses such as christmas lights. In the circuit below, Fig.16, the lights blinking rate can be sped up and slowed down by changing the potentiometer. The further that the potentiometer is changed from 50% the faster the lights will blink as the flow of current experiences.

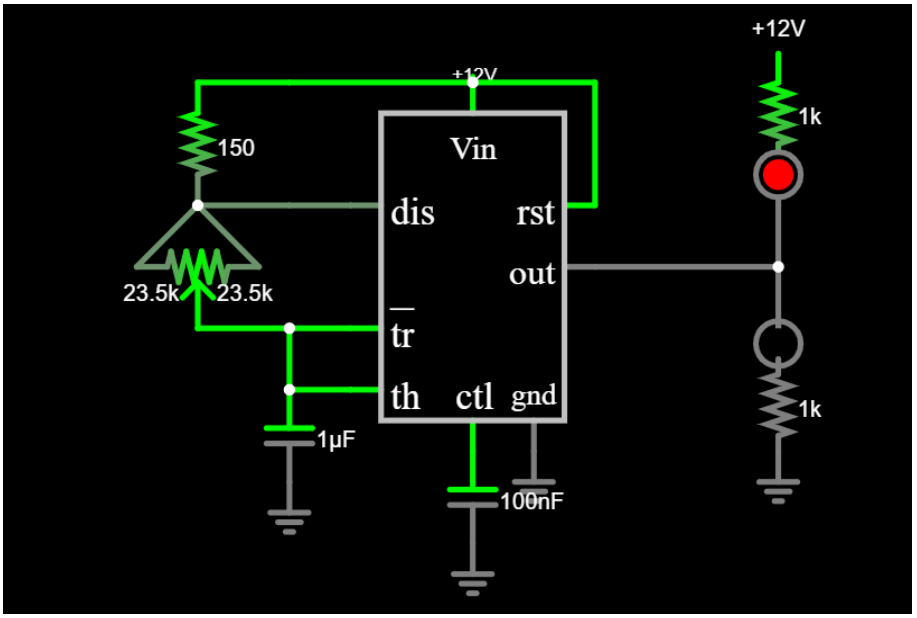


Fig.16 shows the circuit diagram for the Christmas lights prototype mk1.

With some alterations to my original design, I built another variation of Christmas lights as seen in Fig.17. With the changing of the potentiometer percentage, I can control which LED lights up and for how long. The 10K ohm potentiometer is high enough to stop one of the LEDs from blinking when turned towards the other. When turned to 50% a small amount of light does blink and a higher resistance rating would stop all light completely. However when turned away from 50% more the intensity of light increases and the LED light pulse gets shorter meanwhile the other LED gets dimmer and its pulse (which is not visible) gets longer. This prototype allows the user to control which lights turn on and for how long and how intense the light will be. The total period of the two LED's correspond to the "on" and "off" times of the PWM and this period stays constant however the duty cycle changes with the potentiometer. One LED corresponds to the "on" time and the other the "off" time.

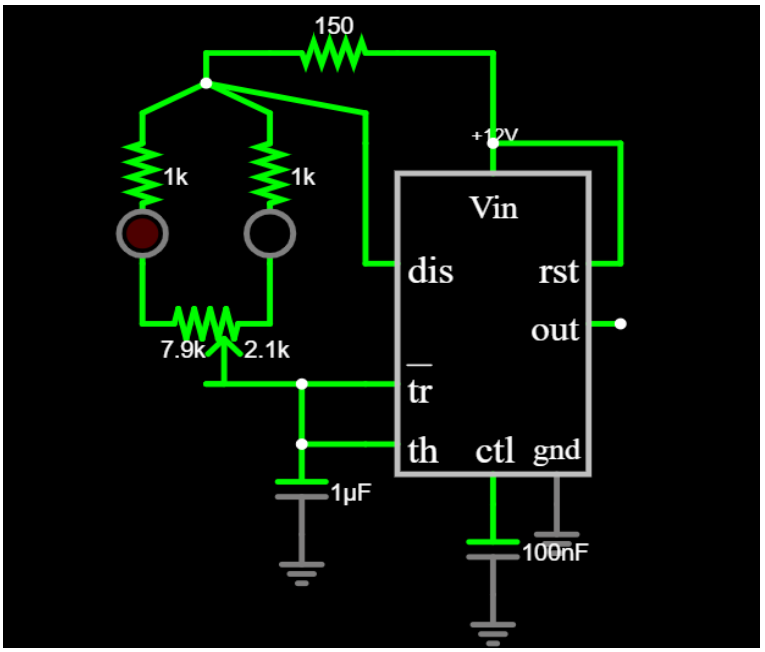


Fig.17 shows the circuit diagram for the Christmas lights prototype mk2.

References

[1] *555 Timer & Raspberry Pi Pico Pulse-Width Modulation Lab Manual, UCD School Of Physics*
[2]Heath, J. (2017). PWM: Pulse Width Modulation: What is it and how does it work? [online] Analog IC Tips. Available at: https://www.analogictips.com/pulse-width-modulation-pwm/.
[3]Electronics Tutorials (2013). Pulse Width Modulation Used for Motor Control. [online] Basic Electronics Tutorials. Available at: https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html.
[4] Cancel PWM DAC ripple with analog subtraction, Stephen Woodward, https://www.edn.com/cancel-pwm-dac-ripple-with-analog-subtraction/
[5]Audio Science Review (ASR) Forum. (n.d.). How do speaker manufacturers calculate the -6dB frequency cutoff? [online] Available at: https://www.audiosciencereview.com/forum/index.php?threads/how-do-speaker-manufacturers-calculate-the-6db-frequency-cutoff.9695/ [Accessed 20 Nov. 2022].
[6]Learningaboutelectronics.com. (2020). What is a Unity Gain Buffer? [online] Available at: http://www.learningaboutelectronics.com/Articles/Unity-gain-buffer.
[7] Basic Electronics Tutorials. (2018). 555 Oscillator Tutorial - The Astable Multivibrator. [online] Available at: https://www.electronics-tutorials.ws/waveforms/555_oscillator.html.
[8]Dejan (2018). How To Make a PWM DC Motor Speed Controller using the 555 Timer IC. [online] HowToMechatronics. Available at: https://howtomechatronics.com/how-it-works/electronics/how-to-make-pwm-dc-motor-speed-controller-using-555-timer-ic/.
[9] Falstad.com. (2019). Circuit Simulator Applet. [online] Available at: https://www.falstad.com/circuit/.

Electronics 2

1 Part A I

In [ ]:
```
from machine import Pin, PWM
import time

pwm1 = PWM(Pin(1))
pmw1.duty_u16(65535//2)
pwm1.freq(1000)
```

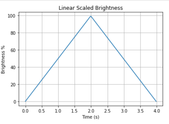
1 Part A II

In [ ]:
```
from machine import Pin, PWM
pmw25 = PWM(Pin(25))
pmw25.freq(1000)
def brightness(x):
    # for 0<=x<=1
    return pmw25.duty_u16(65535*x)
#brightness(1) = full brightness
#brightness(0.5) = half brightness
#brightness(0) = off
```

1 Part A III (Linear)

In [ ]:
```
from machine import Pin, PWM
import time

pwm25 = PWM(Pin(25))


pwm25.freq(1000)


while True:
    start = time.ticks_ms()
    for duty in range(0,65535, 656):
        pwm25.duty_u16(duty)
        time.sleep(0.025)
    for duty in range(65535, 0, -656):
        pwm25.duty_u16(duty)
        time.sleep(0.025)
        end = time.ticks_ms()
    print(time.ticks_diff(end, start))
```

In [117...
```
x = range(0,100, 1)
t = [x/50 for x in range(0,200,1)]
con = np.concatenate((x,x[::-1]))
plt.plot(t,con)
plt.grid(True)
plt.xlabel("Time (s)")
plt.ylabel("Brightness %")
plt.title("Linear Scaled Brightness");
```



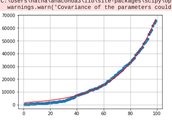
1 Part A III (Exponential Scale)

In [ ]:
```
import math
from machine import Pin, PWM
import time


z = [257, 257,257, 257, 514, 514, 514, 514, 514, 771, 771, 771,
     1028, 1028, 1028, 1028, 1285, 1285, 1542, 1542, 1542, 1799,
     2056, 2056, 2056,2313, 2570, 2570, 2570, 2827, 3084,3598, 3855,
     4112, 4626, 5140, 5397, 5654, 6682,7196, 7710, 7967, 8224, 8738,
     8995, 9252, 9766, 10023, 10794, 11308, 11565, 11822, 12336,12593,
     12850, 13364, 13878,14392, 15163,15677, 16191, 17218, 18247, 18504,
     19532, 20560, 21074,21845, 23130,23644, 25186, 25700, 26471, 27242,
     28784, 29812, 30583, 31354, 32125, 33153, 34438,35466, 36494, 38807,
     40092,41120, 42405, 43690, 44975, 47545, 48830, 50115, 51400, 54998,56797,
     58596, 61937, 63736,65535] #values picked to give a smooth exponential curve
pwm25 = PWM(Pin(25))

pwm25.freq(1000)

while True:
    start = time.ticks_ms()
    for duty in z:
        pwm25.duty_u16(duty)
        time.sleep(0.02)
    for duty in z[::-1]:
        pwm25.duty_u16(duty)
        time.sleep(0.02)
        end = time.ticks_ms()
    print(time.ticks_diff(end, start))
```

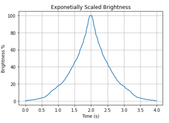
In [86]:
```
import numpy as np
from scipy.optimize import curve_fit
def func(x,a,b,c):
    return np.exp((x-a)/b) + 1/c
x=range(1,100,1)
params, covs = curve_fit(func, x, z)
a, b, c = params[0], params[1], params[2]
yfit1 = np.exp((x-a)/b) + 1/c
plt.plot(x, yfit1,"r")
plt.scatter(x,z)
plt.grid(True)
print(a,b,c);
# f(x) = 1/-2.4e+25 * exp((x+195.6)/26.5)
```

```
-195.60559285182853 26.518397859432607 -2.3985252767541705e+25
```

```
C:\Users\natha\anaconda3\lib\site-packages\scipy\optimize\minpack.py:828: OptimizeWarning: Covariance of the parameters could not be estimated
  warnings.warn('Covariance of the parameters could not be estimated',
```



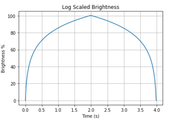
In [138...
```
z = [257, 257,257, 257, 514, 514, 514, 514, 514, 771, 771, 771,
     1028, 1028, 1028, 1028, 1285, 1285, 1542, 1542, 1542, 1799,
     2056, 2056, 2056,2313, 2570, 2570, 2570, 2827, 3084,3598, 3855,
     4112, 4626, 5140, 5397, 5654, 6682,7196, 7710, 7967, 8224, 8738,
     8995, 9252, 9766, 10023, 10794, 11308, 11565, 11822, 12336,12593,
     12850, 13364, 13878,14392, 15163,15677, 16191, 17218, 18247, 18504,
     19532, 20560, 21074,21845, 23130,23644, 25186, 25700, 26471, 27242,
     28784, 29812, 30583, 31354, 32125, 33153, 34438,35466, 36494, 38807,
     40092,41120, 42405, 43690, 44975, 47545, 48830, 50115, 51400, 54998,56797,
     58596, 61937, 63736,65535, 65535]
z = [x/(655.35) for x in z]
t = [x/50 for x in range(0,200,1)]
con = np.concatenate((z,z[::-1]))
plt.plot(t,con)
plt.grid(True)
plt.xlabel("Time (s)")
plt.ylabel("Brightness %")
plt.title("Exponetially Scaled Brightness");
```



1 Part A III (Log Scale)

In [ ]:
```
import math
from machine import Pin, PWM
import time


z = [14261*math.log(x) for x in range(1,101,1)]
#14261 = scaling factor to make max 65535
pwm25 = PWM(Pin(25))

pwm25.freq(1000)

while True:
    start = time.ticks_ms()
    for duty in z:
        pwm25.duty_u16(duty)
        time.sleep(0.02)
    for duty in z[::-1]:
        pwm25.duty_u16(duty)
        time.sleep(0.02)
        end = time.ticks_ms()
    print(time.ticks_diff(end, start))
```

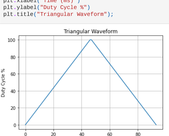
In [131...
```
z = [14261*math.log(x) for x in range(1,101,1)]
z = [x/(655.35) for x in z]
t = [x/50 for x in range(0,200,1)]
con = np.concatenate((z,z[::-1]))
plt.plot(t,con)
plt.grid(True)
plt.xlabel("Time (s)")
plt.ylabel("Brightness %")
plt.title("Log Scaled Brightness");
```



1 Part C I - Triangular Wave

In [ ]:
```
from machine import Pin, PWM
import time

pwm14 = PWM(Pin(14))
#period = 50ms
#period*x = 50ms/x
x= 0.5
pwm14.freq(20000)
period = int(65535//23)
period =x*period
z = range(0,65535, int(period))

while True:
    start = time.ticks_ms()
    for duty in z:
        pwm14.duty_u16(duty)
        time.sleep(0.001)
    for duty in z[::-1]:
        pwm14.duty_u16(duty)
        time.sleep(0.001)
    end = time.ticks_ms()
    print(time.ticks_diff(end, start))
```

In [152...
```
x= 0.5
period = int(65535//23)
period =x*period
z = range(0,65535, int(period))
z = [x/655.35 for x in z]
con = np.concatenate((z,z[::-1]))
plt.plot(con)
plt.grid(True)
plt.xlabel("Time (ms)")
plt.ylabel("Duty Cycle %")
plt.title("Triangular Waveform");
```



1 Part C II - Sine Wave

In [ ]:
```
import math
from machine import Pin, PWM
import time

#period wanted in ms = pw
pw=50
x=pw/220
period = 2*(math.pi)//x
#x = 1 period = 220ms
#220ms*x = period
z = [int((32767/67)*(math.sin(p/200 + 250)/(100) + 0.01))
     for p in range(0, int(2*(math.pi)*200), int(period))]

# 3276767 is scaling factor =(65535?100)/2

# + 0.01 moves the wave above the x axis and away from minus numbers

# + 250 moves the wave along its path to give symmetry and start the wave at a consistent postion

# the factors of 200 confine the wave to a singular period while taking 200 points.
#This number is arbitrary but will change the period of the wave due to executional time

pwm14 = PWM(Pin(14))

pwm14.freq(20000)


while True:
    start = time.ticks_ms()
    for duty in z:
        pwm14.duty_u16(duty)
        time.sleep(0.001)
        end = time.ticks_ms()
    print(time.ticks_diff(end, start))
```

In [151...
```
pw=50
x=pw/220
period = 2*(math.pi)//x
z = [int((32767/67)*(math.sin(p/200 + 250)/(100) + 0.01))
     for p in range(0, int(2*(math.pi)*200), int(period))]
z = [x/655.35 for x in z]
plt.plot(z)
plt.grid(True)
plt.xlabel("Time (ms)")
plt.ylabel("Duty Cycle %")
plt.title("Sine Waveform");
```

In [ ]: