TECHIN512 Lab 3: RC Circuit

**Autumn 2021**  Rev 3 Sept 2021

### Introduction

Learning Objectives: students successfully completing this lab will be able to:

* Explain and compute how much electrical energy can be stored in a capacitor
* Explain a simple switching model of a BJT
* Compute current in a forward-biased LED circuit (LED on).
* Build a simple delay circuit with led, switch, battery, resistor and BJT.
* Explain the working principle of a low-pass RC filter circuit by making a voltage divider analogy
* Apply the RC low pass filter to the Pulse Width Modulation (PWM) implementing a Digital-Analog-Converter (DAC)

### Write-Up

The writeup for this lab should contain the following outline:

1. Title page (see Title Page template posted on Canvas Laboratory page)
2. Introduction   
   ⅓ of a page describing the purpose and goals of this lab *in your own words.* **Do not reproduce any material from this assignment document in any section of your writeup.**
3. Results  
   Each location in the instructions below marked with “>” indicates some data which must appear in your report. Separate each result or related set of results with a section header indicating what it is. For any numerical data or graph describe the meaning of the data.
4. Discussion and Conclusions:

* In ½ to 1 page, summarize the key learning points from the Results above.
* << specific items if nesc>>

### Preparation:

Pre-lab requirement:

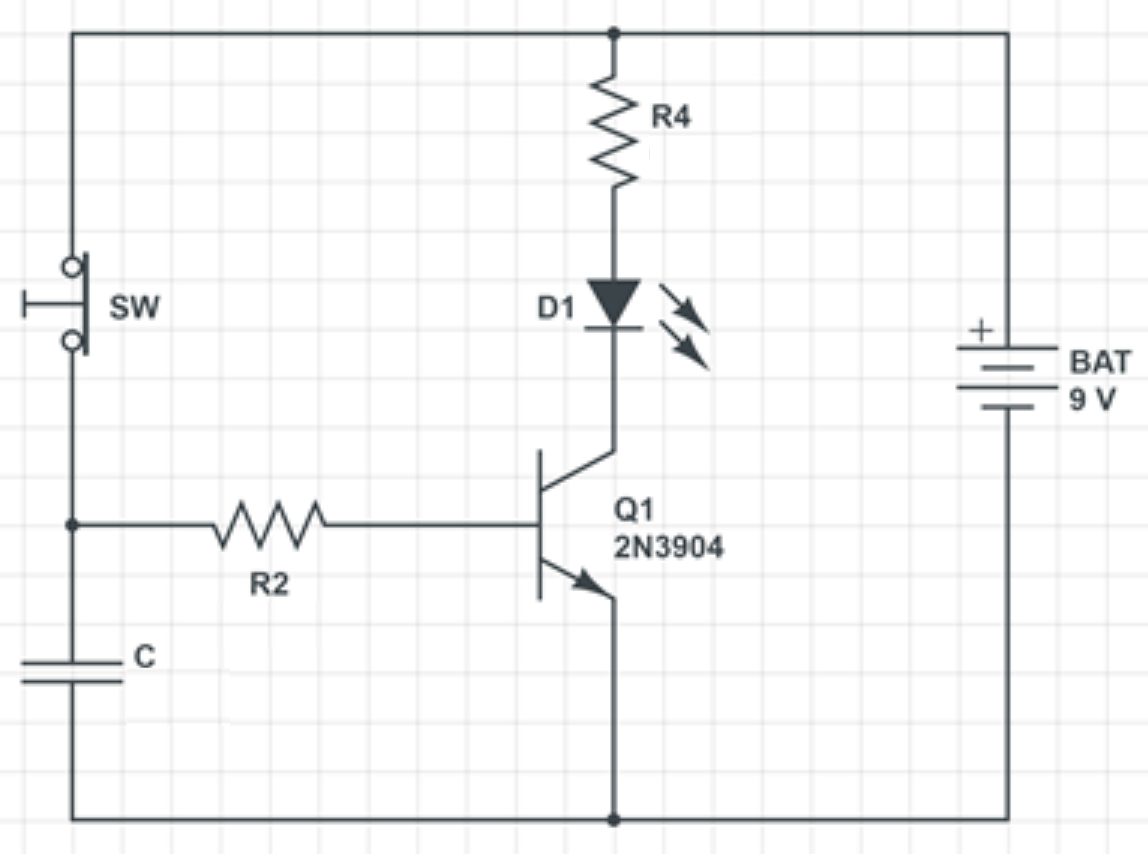
* Review the knowledge of voltage division circuit, RC circuits, and BJTs. See TECHIN 512 Virtual Textboook posted on the Canvas Laboratory page.
* Review resistance, capacitance and impedance through wikipedia or other online resources. <https://en.wikipedia.org/wiki/Electrical_impedance>

Parts, tools, supplies required:

* DC power supply(9V- battery), capacitors, resistors, NPN BJT (2N3904), light emitting diode (LED), Arduino or ESP32, wires.
* Function generator, oscilloscope, multimeter.

‘Scope setup:

* Recalibrate your probes.
* Make sure all probes are set to 10x and all input channels are also set to 10x (press the colored button and select on menu).



## **Figure 1: Delay Circuit with BJT. As shown above is the LED lit or not? What happens if you press and hold the button?**

## Parts Needed:

1. Resistors: 250, 1000
2. Capacitor: 1000uF, >= 10V rating
3. On/Off SPDT switch

## Equipment:

* Standard lab bench
* Fluke DMM *OR* bench top DMM

## Part 1: Basic LED Circuit

**Component Values:**

**R4**: 200-250 Ohm

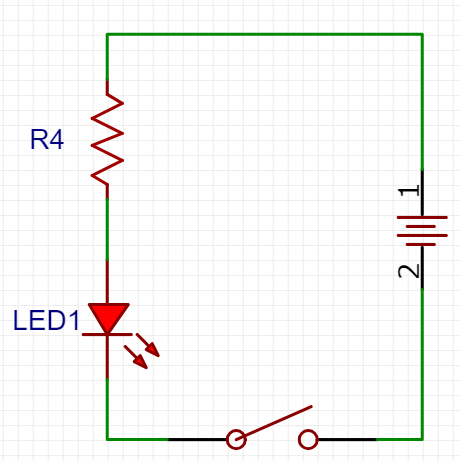
**R2**: 1000 Ohm (1K Ohm)

**C:** 1000 uF

Build the circuit of Figure 1 *with the following simplifications*:

* Do not connect battery positive terminal to R4 until step 1.2 below.
* Omit R2, C, and Q1
* Perform step 1.1
* Place the switch (instead of the location in figure 1) between the “bottom” of the LED (D1) and ground (negative battery terminal)

\*Actually this circuit only have one loop, with LED, R4 and switch.



1.1: **Check your switch.**  As shown in the schematic, the switch (SW) will open the circuit when pressed. Does your switch work this way? Use the DMM to find out if it is “press-to-open” or “press-to-close” type switch, or Single Pole Single Throw Switch type switch.

> take a photo of your switch and report what you measured with DMM, and which type of switch you have.

1.2 **Operate your circuit.**  Repeatedly operate your switch and verify that the LED lights up.

> include a small photo of the lit up LED including the switch position.

> Use DMM to measure two voltages:

1. across the LED
2. across R4.

1.3 **Measure current.**  Set your DMM to current mode and change the red wire to “mA/uA”. Open circuit between +9V and R4 and connect DMM.

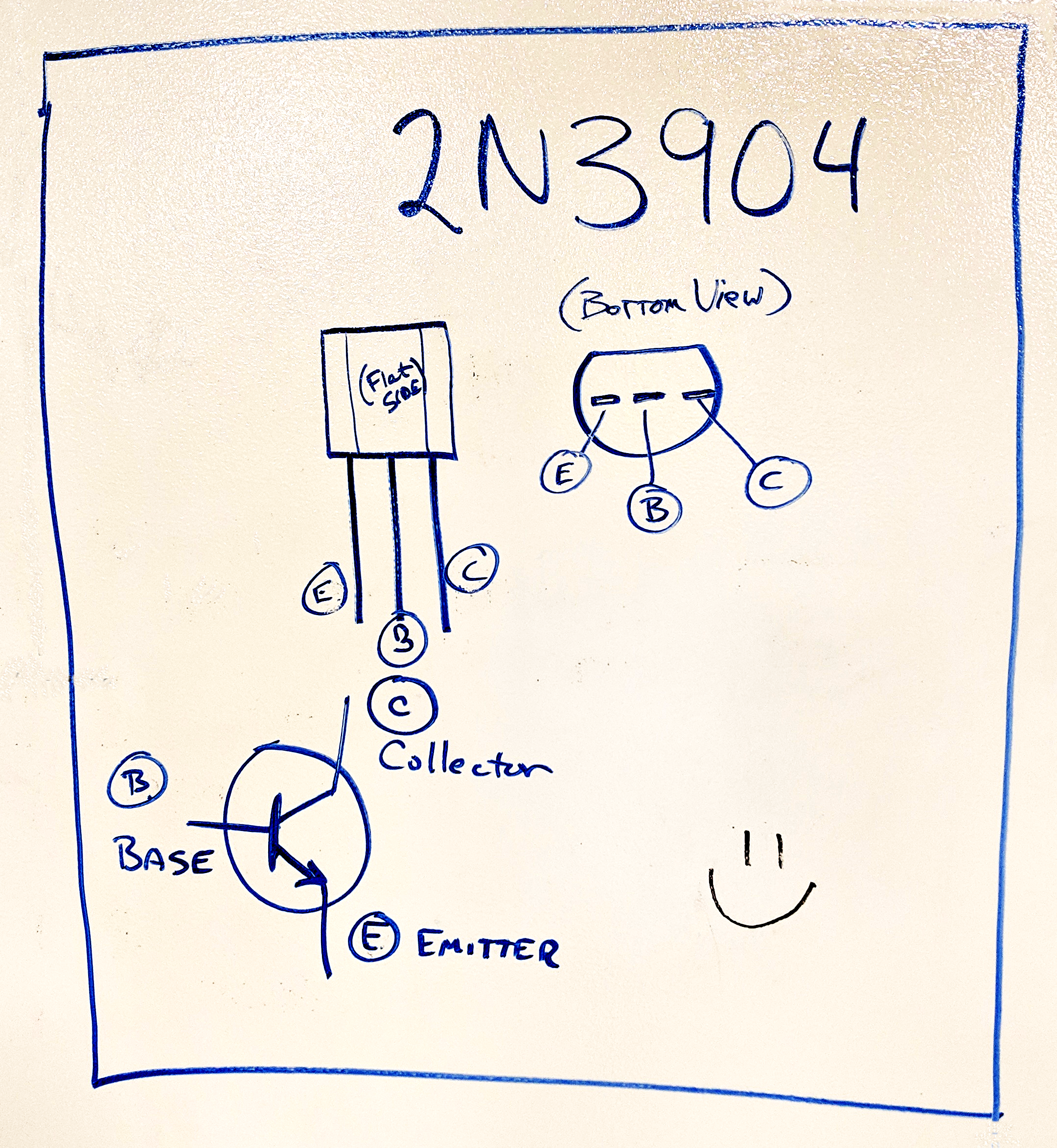
> what is current with switch open and closed. Does LED still light?

> Compute R4 current according to measured voltage (step 1.2.1) and its value (200 Ohms or as built).

## Part 2: BJT Circuit

The Bi-polar Junction Transistor is a versatile device which we will use as a current-controlled switch. A small current going in to the “base” terminal (to the emitter) controls a bigger current flowing from “collector” to “emitter”.

First we will demonstrate this ability using the switch to change base current input from zero to a small value. This will light the LED. It takes some care to wire up the BJT correctly. Compare the schematic to this diagram to identify the correct pins:



2.1 **Build the switching BJT circuit.**  Wire up exactly as shown in Figure 1 *except* leave out the capacitor C. Operate the switch - LED should go on and off.

> take a photo of your circuit build.

2.2 **Measure Currents.**  For **both switch positions**, measure the current in

> R2, the base circuit

> R4, the LED / Collector circuit.

> Explain your results in terms of the switching idea described above.

## Part 3: BJT Timer Circuit

3.1 Now **add the capacitor C** as shown in Figure 1. Operate the switch.

> what happens when you operate the switch?

3.2 **Time responses.**  Connect both ‘scope probes ground clips to negative battery voltage (0V). Connect Channel 1 to the capacitor. Connect Channel 2 to the junction of R4 and the LED.

> capture a screenshot of what happens to these two voltages when you close the switch and then open it.

> Add a horizontal bar to the screenshot to illustrate when the LED is ON.

> Document different observed delay times with at least 3 different capacitors (show three screenshots and give C values). It may take up to several minutes until the LED fully dimmed. If the LED keeps glowing, you can record the time when the voltage across the LED drops 80%.

## **Part 4:** RC Low-Pass Filter

This part will illustrate a practical application of RC circuits: smoothing out pulse-width-modulation to approximate an analog signal. Many low-cost microcontroller (e.g. ESP32 and Arduino) based embedded systems do not have analog outputs, but we can fake one by rapidly flipping a bit on and off.

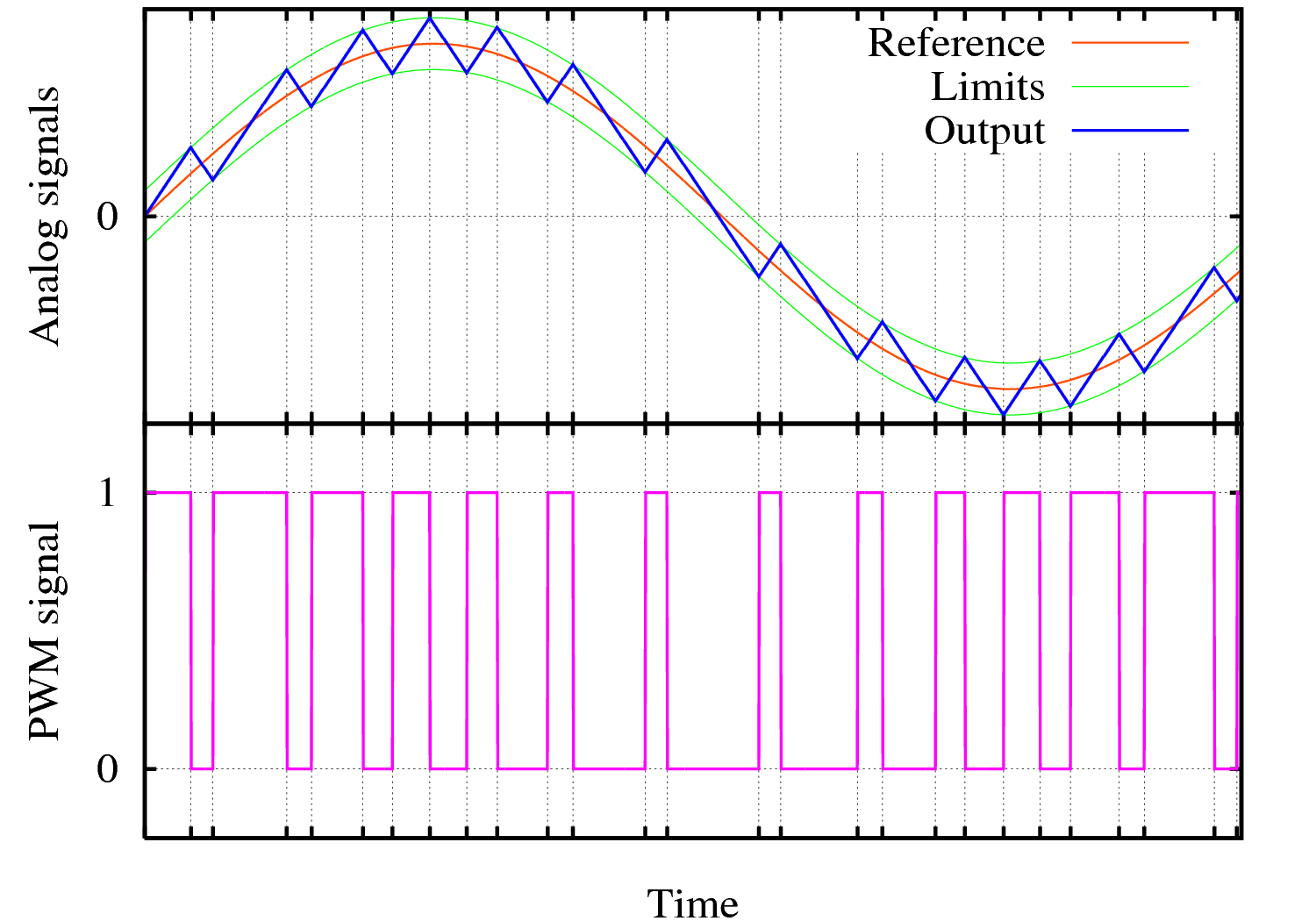
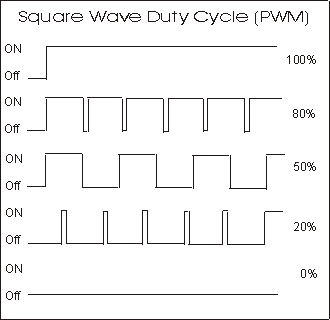
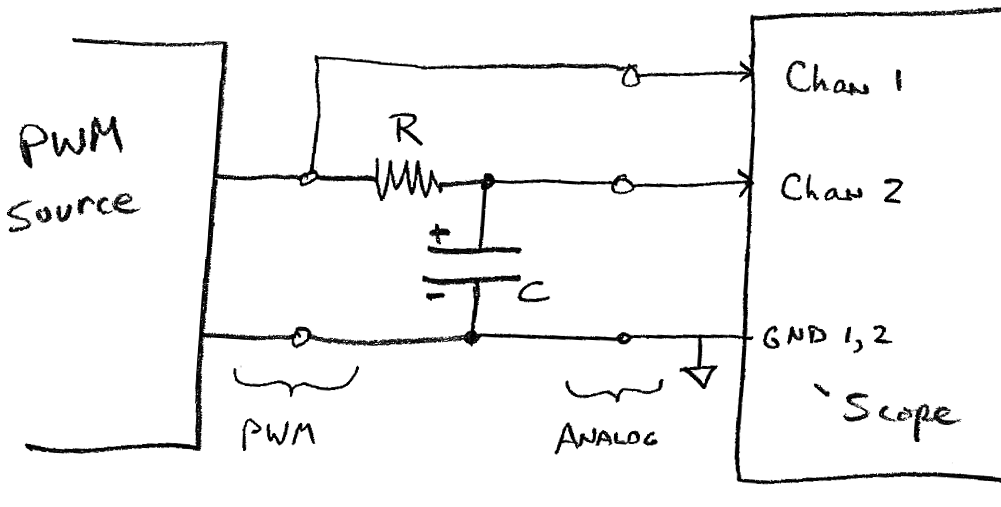


Figure 2: Pulse Width Modulation. Several levels of PWM (left). Smoothing out a PWM signal can approximate an analog signal with lower frequencies (right).

Pulse width modulation (PWM) is a series of pulses which turn ON at regular time intervals but can stay on for different amounts of time. We express this as a percentage of time that the signal is ON. The percentage of ON time is called the duty cycle. If the duty cycle is high, then more energy is delivered to a load. If we switch ON and OFF quickly, then we can modulate this duty-cycle up and down to approximate an analog output. Please review [this introduction to PWM](https://learn.sparkfun.com/tutorials/pulse-width-modulation/all) from Sparkfun if you are not already familiar with it. PWM signals have a lot of high frequencies in them due to the switching frequency and also harmonics from the rectangular wave shape. But they also have a DC component (0 frequency) which depends only on the ratio of on to off times (duty cycle). We will build a low-pass filter to get rid of the high frequencies and only let through the lowest frequencies to reveal the DC component of the PWM signal.

By “filtering out” the high switching related frequencies, a RC low-pass filter can convert a PWM signal from ESP32 to a “smooth” DC voltage (by filtering out the switching frequency).

Our overall system is shown in Figure 3.



**Figure 3. Using an RC circuit to “smooth” a PWM signal into an analog signal. The “low-pass” RC circuit shown eliminates high frequency changes and only passes through low frequencies.**

### **Procedure**:

1. Generate a PWM signal where you can control the duty cycle. For the source of the PWM signal you have two options:
   1. Use ESP32 with the code in lab manual or use an arduino with code in **Appendix B** (below) (determine the PWM switching frequency of this code).
   2. Use the signal generator with appropriate settings. PWM frequency should be 1.0 kHz. Amplitude should be Vmin = 0.0, Vmax = 3.5V.
2. Design an RC low-pass circuit. Determine a cutoff frequency 10x lower than your PWM switching frequency. Determine your RC time constant based on this frequency (See **Appendix A**). Set R = 1000 Ohms, determine your “C” value. If exact C value is not available, pick the closest available and   
   > document changes in circuit design and parameters due to this approximate value.
3. Connect oscilloscope to circuit as shown in Figure 3. Set vertical controls for Channels 1 and 2 to show both waves. Set the time base to show 10 cycles of the PWM input.
4. Measure input and output for duty cycle of 20%, 60%, 90%  
   > capture screen shots of the three duty cycles.
5. Repeat step two, but set the cutoff frequency 100x lower than the PWM frequency. Measure input and output for duty cycle 50%  
    > what is the wave shape of the output signal (screenshot)?
6. Using this new RC circuit,   
   > Make a graph of the DC value of the output vs. PWM duty cycle for duty cycle values of {10%, 30%, 50%, 75%,90%}

> Document your circuit design and how you chose the right resistor and capacitor.

> Include a schematic diagram of your circuit design with each component marked with its value.

> One photo of your circuit on the breadboard.

**Appendix A: Low Pass Filter Design**

(for a detailed Tutorial see https://www.electronics-tutorials.ws/filter/filter\_2.html )

The simple RC low-pass filter of Figure 3 works like a voltage divider. **The “resistance” of a capacitor depends inversely on frequency** and is referred to as *reactance*,

.

Considering the RC circuit as a voltage divider:

(for EE’s it should be )

Point is, as gets smaller gets smaller. Since gets smaller as frequency goes up, gets smaller as frequency goes up. It’s useful to determine the frequency where . We call this the “cutoff” frequency.

Solving:

The RC circuit “passes through” frequencies below and “blocks” frequencies above .

*Example:*

We want to cut off frequencies above 1000Hz and R = 1000 Ohms, find C.

Solution:

,

**Appendix B: Arduino Code (PWM for Sine Wave)**

**http://forcetronic.blogspot.com/2018/02/converting-arduino-pwm-output-to-dac.html**

/\*This code was made for a video tutorial on the ForceTronics YouTube Channel called

\* Converting an Arduino PWM Output to a DAC Output. This code is free to use and

\* modify at your own risk

\*/

uint8\_t pVal = 127; //PWM value

const float pi2 = 6.28; //Pie times 2, for building sinewave

const int samples = 100; //number of samples for Sinewave. This value also affects frequency

int WavSamples[samples]; //Array for storing sine wave points

int count = 0; //tracks where we are in sine wave array

void setup() {

Serial.begin(115200); //for debugging

pinMode(10, OUTPUT); //pin used for analog voltage value

pinMode(4,OUTPUT); //pin used to fake PWM for sinewave

setPwmFrequency(10,1); //function for setting PWM frequency

analogWrite(10,127); //set duty cycle for PWM

float in, out; //used for building sine wave

for (int i=0;i<samples;i++) //loop to build sinewave

{

in = pi2\*(1/(float)samples)\*(float)i; //calculate value for sine function

WavSamples[i] = (int)(sin(in)\*127.5 + 127.5); //get sinewave value and store in array

Serial.println(WavSamples[i]); //for debugging

}

}

void loop() {

if(count > samples) count = 0; //reset the count once we are through array

bitBangPWM(WavSamples[count],4); //function for turning sinewave into "fake" PWM signal

count++; //increment position in array

}

//Function to bit bang a PWM signal (we are using it for the sinewave)

//input are PWM high value for one cycle and digital pin for Arduino

//period variable determines frequency along with number of signal samples

//For this example a period of 1000 (which is 1 millisecond) times 100 samples is 100 milli second period so 10Hz

void bitBangPWM(unsigned long on, int pin) {

int period = 1000; //period in micro seconds

on = map(on, 0, 255, 0, period); //map function that converts from 8 bits to range of period in micro sec

//Serial.println(on); //debug check

unsigned long start = micros(); //get current value of micro second timer as start time

digitalWrite(pin,HIGH); //set digital pin to high

while((start+on) > micros()); //wait for a time based on PWM duty cycle

start = micros();

digitalWrite(pin,LOW); //set digital pin to low

while((start+(period - on)) > micros()); //wait for a time based on PWM duty cycle

}

/\*\*

\* https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM

\* Divides a given PWM pin frequency by a divisor.

\*

\* The resulting frequency is equal to the base frequency divided by

\* the given divisor:

\* - Base frequencies:

\* o The base frequency for pins 3, 9, 10, and 11 is 31250 Hz.

\* o The base frequency for pins 5 and 6 is 62500 Hz.

\* - Divisors:

\* o The divisors available on pins 5, 6, 9 and 10 are: 1, 8, 64,

\* 256, and 1024.

\* o The divisors available on pins 3 and 11 are: 1, 8, 32, 64,

\* 128, 256, and 1024.

\*

\* PWM frequencies are tied together in pairs of pins. If one in a

\* pair is changed, the other is also changed to match:

\* - Pins 5 and 6 are paired on timer0

\* - Pins 9 and 10 are paired on timer1

\* - Pins 3 and 11 are paired on timer2

\*

\* Note that this function will have side effects on anything else

\* that uses timers:

\* - Changes on pins 3, 5, 6, or 11 may cause the delay() and

\* millis() functions to stop working. Other timing-related

\* functions may also be affected.

\* - Changes on pins 9 or 10 will cause the Servo library to function

\* incorrectly.

\*

\* Thanks to macegr of the Arduino forums for his documentation of the

\* PWM frequency divisors. His post can be viewed at:

\* http://forum.arduino.cc/index.php?topic=16612#msg121031

\*/

void setPwmFrequency(int pin, int divisor) {

byte mode;

if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {

switch(divisor) {

case 1: mode = 0x01; break;

case 8: mode = 0x02; break;

case 64: mode = 0x03; break;

case 256: mode = 0x04; break;

case 1024: mode = 0x05; break;

default: return;

}

if(pin == 5 || pin == 6) {

TCCR0B = TCCR0B & 0b11111000 | mode;

} else {

TCCR1B = TCCR1B & 0b11111000 | mode;

}

} else if(pin == 3 || pin == 11) {

switch(divisor) {

case 1: mode = 0x01; break;

case 8: mode = 0x02; break;

case 32: mode = 0x03; break;

case 64: mode = 0x04; break;

case 128: mode = 0x05; break;

case 256: mode = 0x06; break;

case 1024: mode = 0x07; break;

default: return;

}

TCCR2B = TCCR2B & 0b11111000 | mode;

}

}