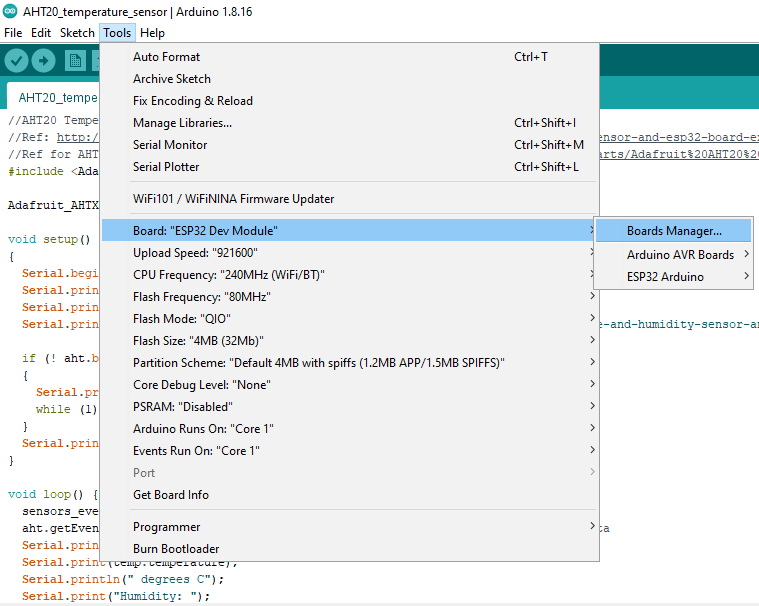
**Table of Contents**

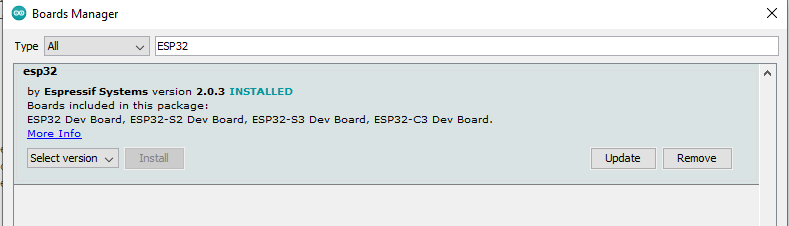
|  |  |
| --- | --- |
| **Contents** | **Page No.** |
| Part A: LED Toggle with Push Button (Pull Up/Down Resistor) | **P2** |
| Part B: RGB LED Pulse Width Modulation (PWM) | **P5** |
| Part C: Servo Motor Pulse Width Modulation (PWM) | **P10** |
| Part D: Digital Microphone with Inter-IC Sound protocol (I2S) | **P15** |
| Part E: Speaker with Digital to Analog Converter (DAC) | **P17** |

***Part A: LED Toggle with Push Button (Pull Up/Down Resistor)***

1. In this scenario, LED’s on/off is controlled by the push button. And the digital signals of LED and push button are checked via the Logic Analyzer and Pulse View. When the push button is pressed, LED is on and then when the button is pressed again, LED is off.
2. Hardware Required
   1. ESP32
   2. LED
   3. Push Button
   4. 220 kΩ Resistor
   5. Bread board
   6. Micro USB cable
   7. Jumper Wires
3. Software Required
   1. Arduino IDE
   2. ESP32 Library
      1. To install it, Open Arduino IDE application and go to “Tools -> Board -> Boards Manager.



* + 1. Then, search “ESP32” library and install it.

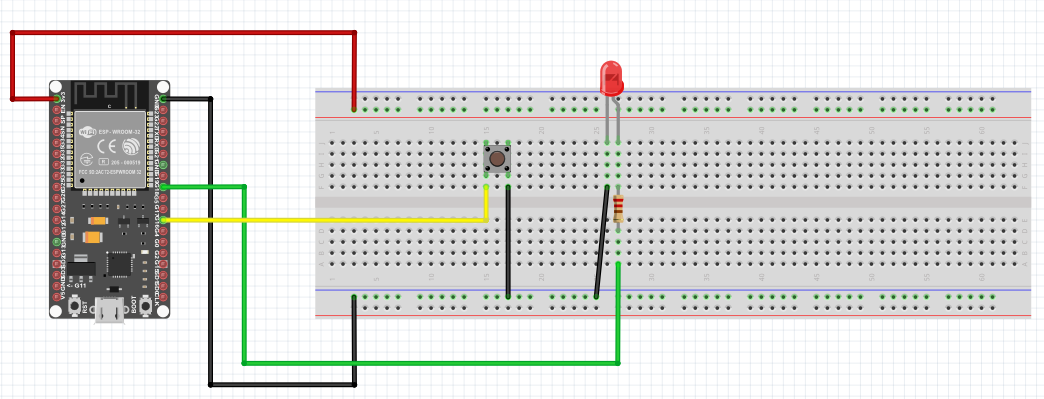


1. How the Push Button Work’s?

Push Button is an electronic component that connects two points in a circuit when the push button is pressed. When the pushbutton is opened (not pressed) there is no connection between the two legs of the push button. Since one pin of the push button is connected to 3.3V through the pull-up resistor and this situation is defined as HIGH.

When the pushbutton is closed (pressed), there is a connection between its two legs, connecting the pin to ground, so the condition is mentioned as LOW. The pin is still connected to input voltage, but the resistor in-between them means that the pin is close to “GND”.

1. Circuit Schematic
   1. LED anode -> 220 kΩ -> GPIO 18
   2. LED cathode -> GND
   3. Push button one pin -> GPIO 14
   4. Push button another pin -> GND



1. Source Code

#define BUTTON\_PIN 14 // ESP32 pin GIOP14, which connected to button

#define LED\_PIN 18 // ESP32 pin GIOP18, which connected to led

// The below are variables, which can be changed

int led\_state = LOW; // the current state of LED

int button\_state; // the current state of button

int last\_button\_state; // the previous state of button

int sensorValue;

void setup() {

Serial.begin(9600); // initialize serial

pinMode(BUTTON\_PIN, INPUT\_PULLUP); // set ESP32 pin to input pull-up mode

pinMode(LED\_PIN, OUTPUT); // set ESP32 pin to output mode

button\_state = digitalRead(BUTTON\_PIN);

}

void loop() {

last\_button\_state = button\_state; // save the last state

button\_state = digitalRead(BUTTON\_PIN); // read new state

if (last\_button\_state == HIGH && button\_state == LOW) {

Serial.println("The button is pressed");

// toggle state of LED

led\_state = !led\_state;

// control LED arccoding to the toggled state

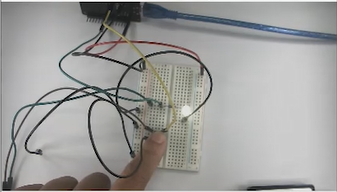
digitalWrite(LED\_PIN, led\_state);

}

Serial.println(sensorValue);

}

1. Connecting the Logic Analyzer with the circuit
   1. LED anode -> Ch1 (Logic Analyzer)
   2. Push button (pin connected with GPIO 16 ===-]pb) -> Ch2 (Logic Analyzer)
   3. ESP32 GND -> GND (Logic Analyzer)
2. Results





Push Button’s signal

LED’s signal

LED is off

Push Button is pressed

Push Button is pressed

LED is on

**Your Testing: Take a video record where you must explain the relationship of the signal between Push button and LED via Logic Analyzer together with your circuit.**

***Part B: RGB LED Pulse Width Modulation (PWM)***

1. In this scenario, PWM of RGB LED is controlled by Potentiometer. And the PWM signals RGB LED are checked via the Logic Analyzer (Pulse View).
2. Hardware Required
   1. ESP32
   2. RGB LED
   3. 10k Potentiometer
   4. 3 x 220 kΩ Resistors
   5. Bread board
   6. Micro USB cable
   7. Jumper Wires
3. Software Required
   1. Arduino IDE
   2. ESP32 Library
4. How to check common cathode/ anode of RGB LED?

RGB-LED includes 4 pins:

R(red) pin is to control the red color element.

G(green) pin is to control the green color element.

B(blue) pin is to control the blue color element.

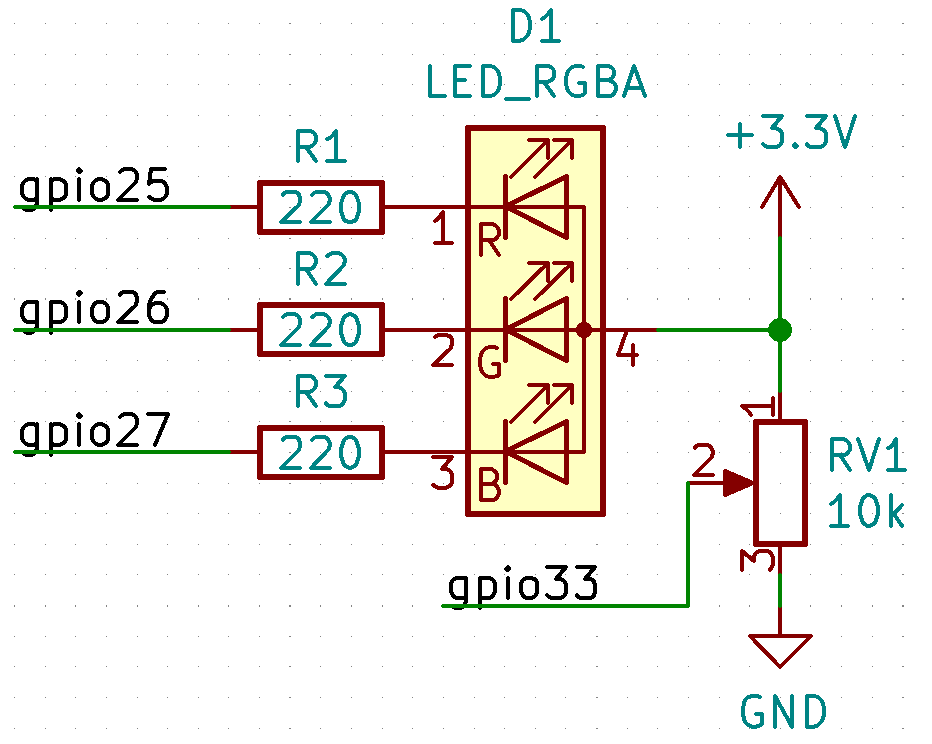
Common (Cathode/Anode) pin is to connect GND (0V) or Vin (3.3V).

Note: Here are the links where you can find how to check RGB LED common cathode/ anode using multimeter.

[Clear common cathode RGB red green blue LED component testing by electronzap - YouTube](https://www.youtube.com/watch?v=dXwgCI6EXUo)

[(Part 1) Check if RGB LED is common cathode or anode - YouTube](https://www.youtube.com/watch?v=qEUrdrS5Md4)

1. Circuit Schematic
   1. RGB LED Red -> 220 kΩ -> GPIO 25 .
   2. RGB LED Green -> 220 kΩ -> GPIO 26
   3. RGB LED Blue -> 220 kΩ -> GPIO 27
   4. RGB LED Common Cathode -> GND, if RGB LED Common Anode -> 3.3V
   5. POT Pin 1 -> 3.3V
   6. POT Pin 2 -> GPIO 33
   7. POT Pin 3 -> GND



1. Source Code

#define POT 33

#define RED 25

#define GREEN 26

#define BLUE 27

#define PWM\_CH\_1 0

#define PWM\_CH\_2 1

#define PWM\_CH\_3 2

#define PWM\_FREQ 15000

#define PWM\_RES 8 // Resolution in bits 12

int aValue;

void setup() {

Serial.begin(9600);

pinMode(RED, OUTPUT);

pinMode(GREEN, OUTPUT);

pinMode(BLUE, OUTPUT);

ledcSetup(PWM\_CH\_1, PWM\_FREQ, PWM\_RES);

ledcSetup(PWM\_CH\_2, PWM\_FREQ, PWM\_RES);

ledcSetup(PWM\_CH\_3, PWM\_FREQ, PWM\_RES);

ledcAttachPin(RED, PWM\_CH\_1);

ledcAttachPin(GREEN, PWM\_CH\_2);

ledcAttachPin(BLUE, PWM\_CH\_3);

}

void loop() {

aValue = analogRead(POT); //8, 12 bits ADC

Serial.println("aValue =" + String(aValue));

ledcWrite(PWM\_CH\_1, aValue);

delay(250);

ledcWrite(PWM\_CH\_2, aValue);

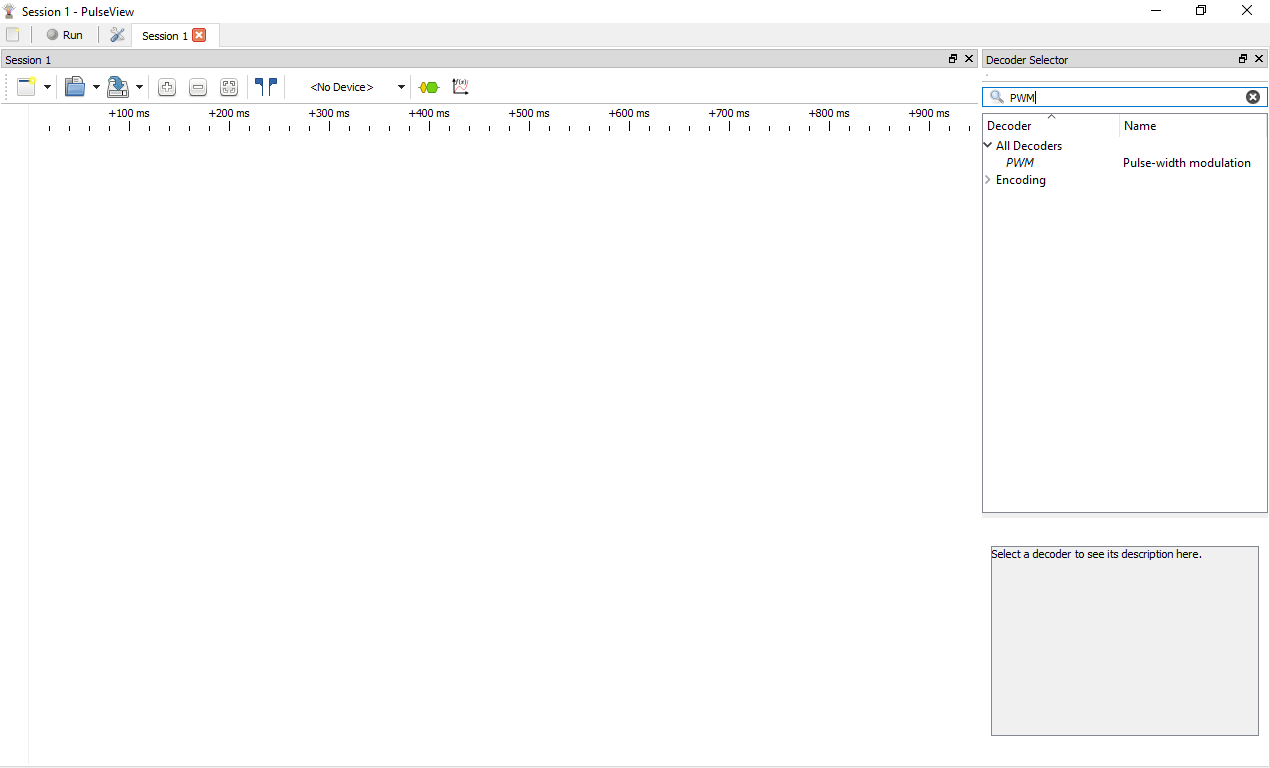
delay(250);

ledcWrite(PWM\_CH\_3, aValue);

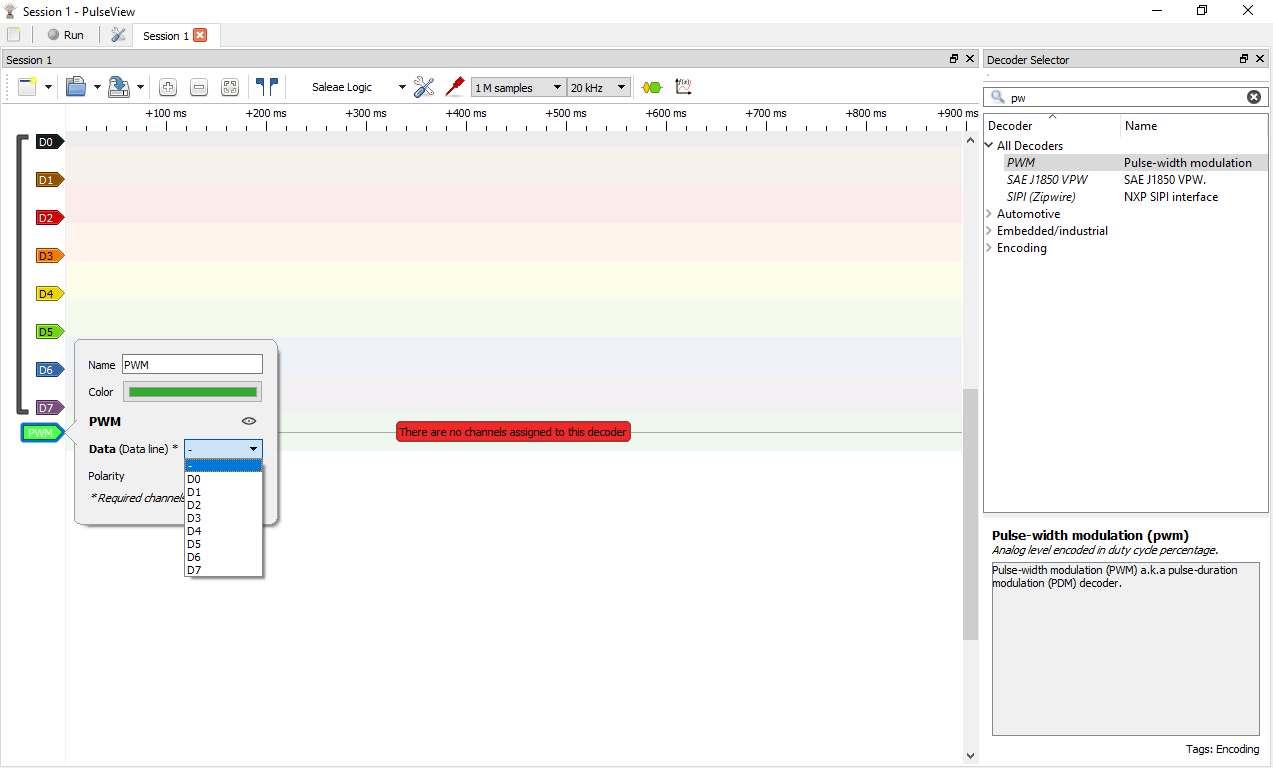
delay(250);

}

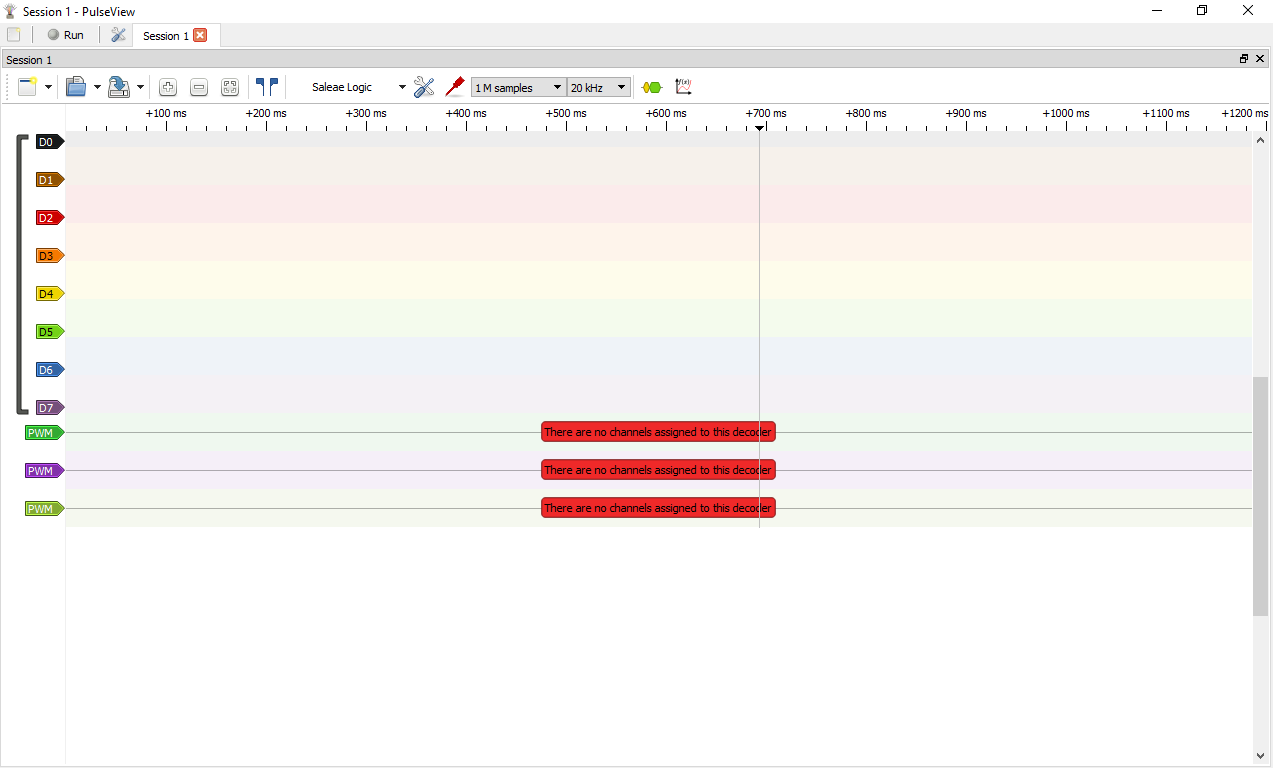
1. Connecting the Logic Analyzer with the circuit
   1. RGB LED Red -> Ch1 (Logic Analyzer)
   2. RGB LED Green-> Ch2 (Logic Analyzer)
   3. RGB LED Blue -> Ch3 (Logic Analyzer)
   4. ESP32 GND -> GND (Logic Analyzer)
2. To check PWM value at Pulse View, click on the protocol decoder  icon and then type “PWM” in search bar and double clicked on PWM to read its value.



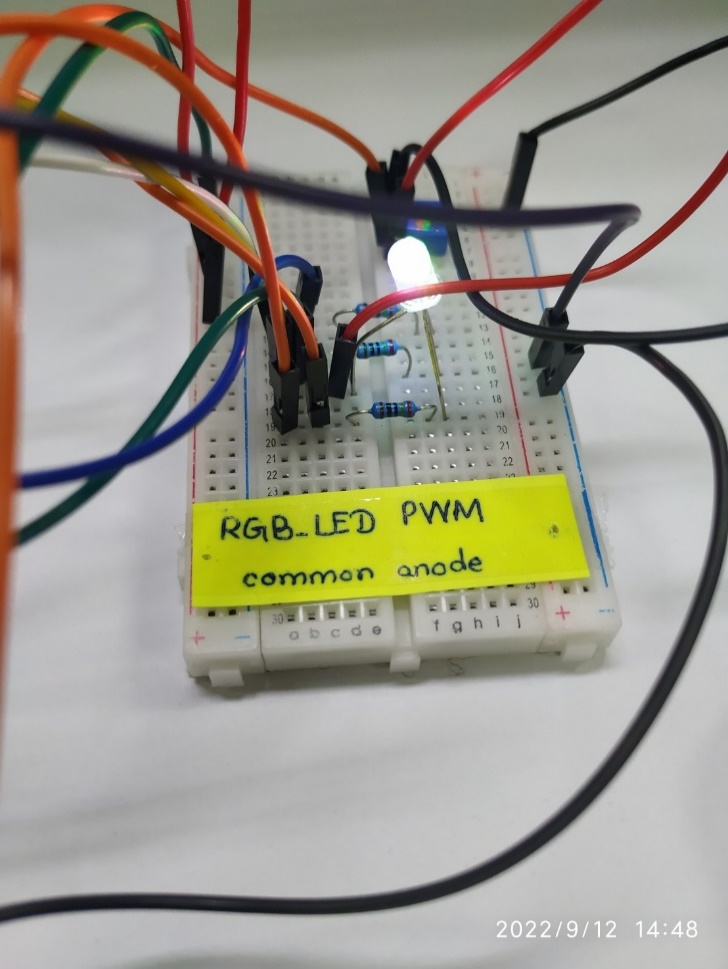
At PWM, you can choose the data line as your preferred.

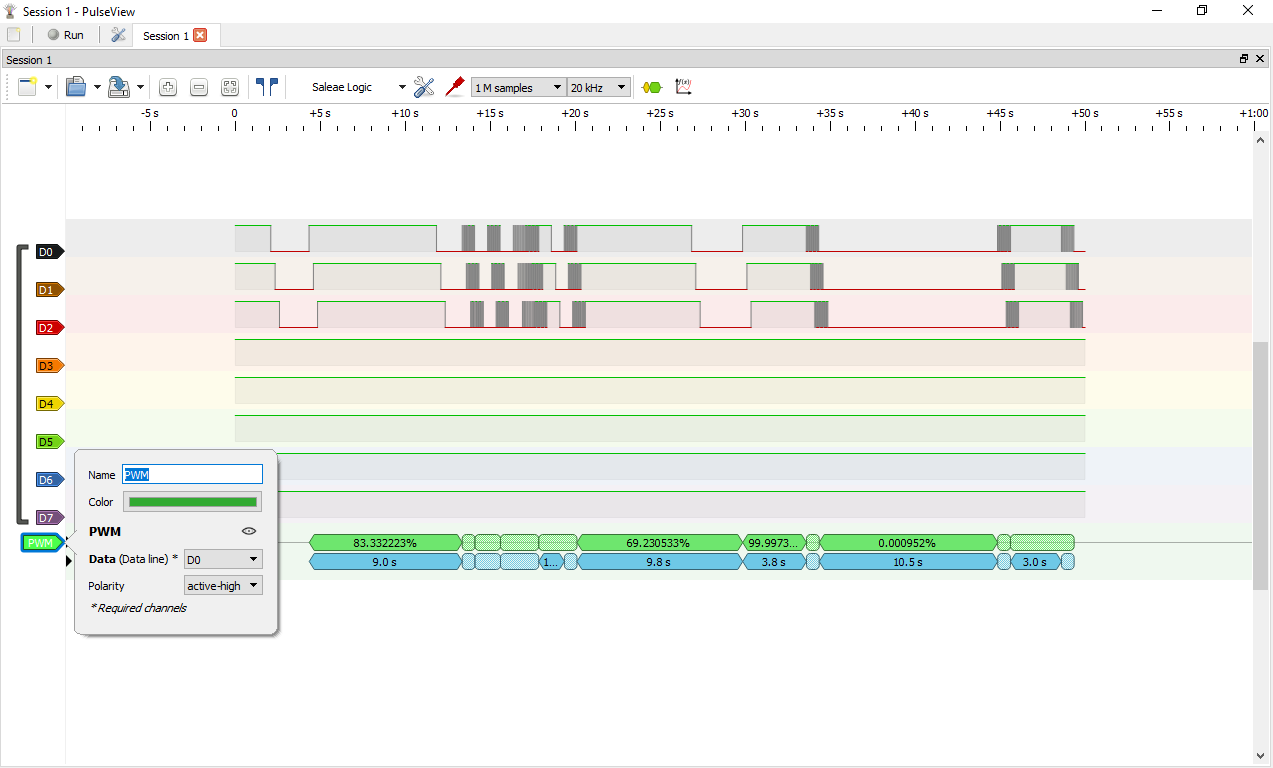


You can add more PWM channels.



1. Results





Data line for PWM is set for D0 (LED Red)

When POT is around middle

LED Red

LED Green

LED Blue

**Your Testing: Take a video record where you must explain the following cases.**

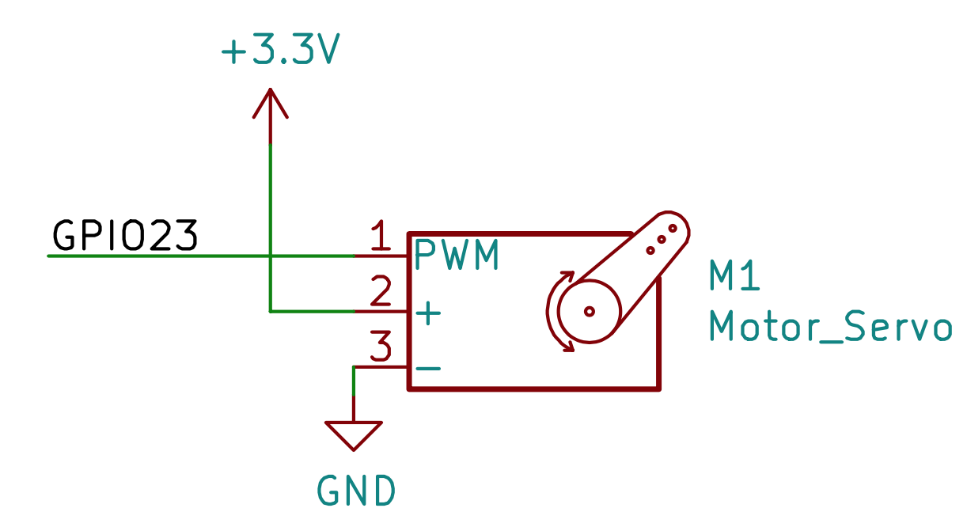
* Turn the potentiometer to around middle and use the logic analyzer to measure GPIO25, 26, 27. Capture the waveforms together with your circuit.
* What happens if you change PWM resolution of PWM in the source code to 16 bits? Try sweeping the potentiometer from minimum to maximum while capturing the waveform at GPIO25. Compare this to the previous configuration (12-bit PWM). Capture the waveform picture and explain what the differences in the waveforms between 16-bit and 12-bit configurations are.
* Try changing the frequency of PWM\_CH\_2 to 10000 and PWM\_CH\_3 to 8000. Use the logic analyzer to measure GPIO25, 26, 27. Capture the VDO of the screen and explain the differences among these 3 channels.

***Part C: Servo Motor Pulse Width Modulation (PWM)***

1. In this scenario, the relationship between duty cycle and angle of the servo motor are checked via the Logic Analyzer (Pulse View).
2. Hardware Required
   1. ESP32
   2. Micro Servo 9g
   3. Bread board
   4. Micro USB cable
   5. Jumper Wires
3. Software Required
   1. Arduino IDE
   2. ESP32 Library
4. Micro Servo



1. Circuit Schematic
   1. Servo Pin 1 -> GPIO 23
   2. Servo Pin 2 -> 3.3V
   3. Servo Pin 3 -> GND



1. Source Code

#include <Arduino.h>

#include <Servo.h>

Servo servo;

void setup ()

{

servo.attach(4);//PWM Pin settings , And GPIO The pin number corresponds to .

}

void loop ()

{

// To 0°

servo.write(0);

delay(100);

// To 45°

servo.write(45);

delay(100);

// To 90°

servo.write(90);

delay(100);

// To 135°

servo.write(135);

delay(180);

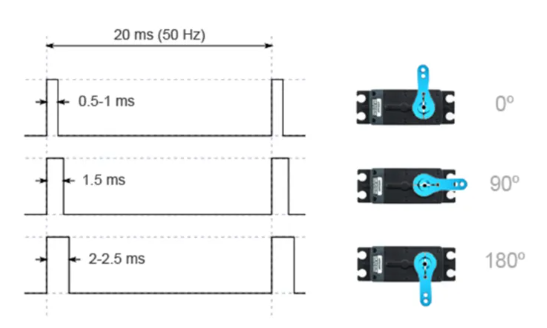
// To 0°

servo.write(180);

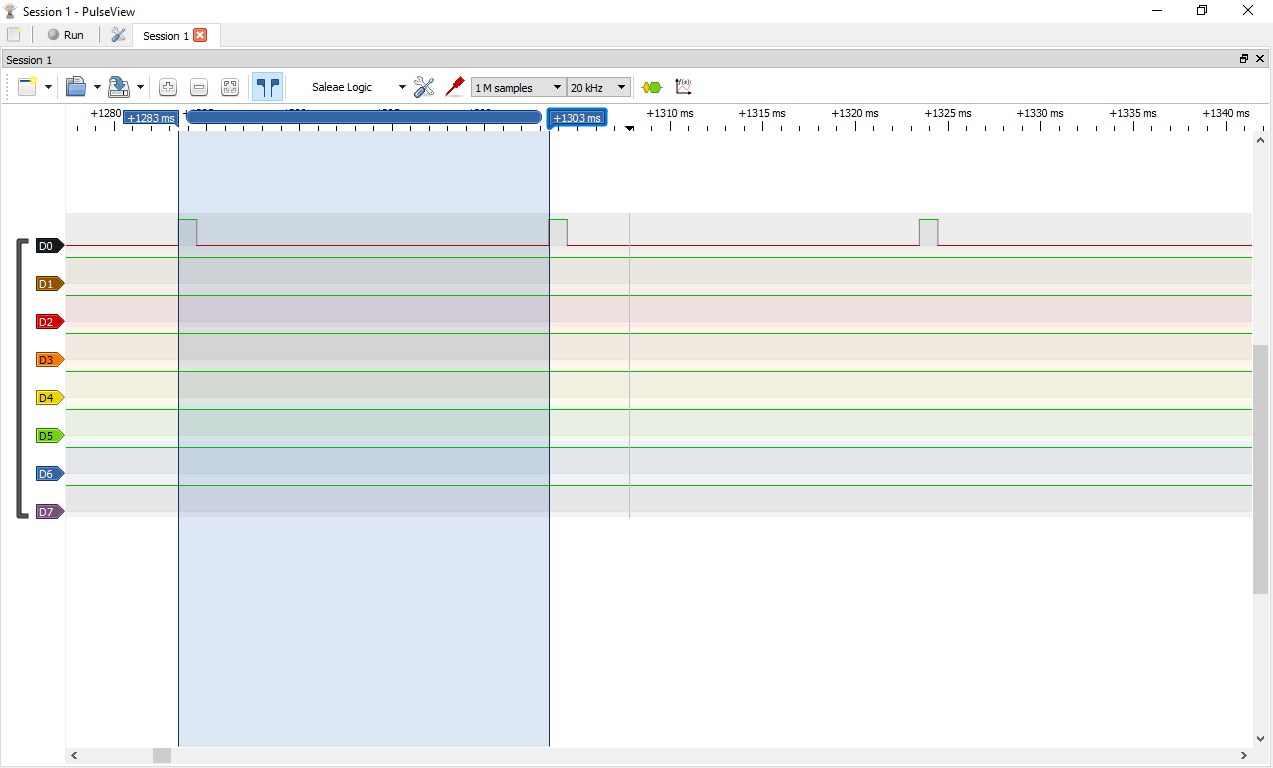
delay(100);

1. } Connecting the Logic Analyzer with the circuit
   1. Servo Pin 1 -> Ch1 (Logic Analyzer)
   2. ESP32 GND -> GND (Logic Analyzer)
2. Results

Here is the relationship between duty cycle and servo rotation angle.

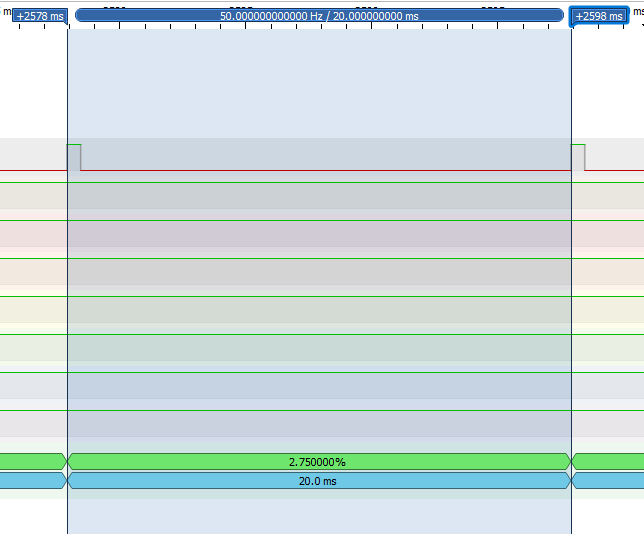


At Pulse View, “show cursors ” is used to read starting and ending timing frame.

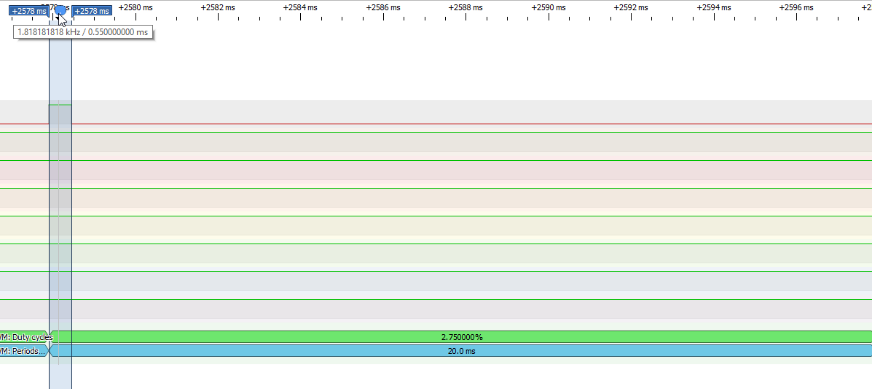


*Logic Scope with Pulse View Result of Servo Motor PWM*

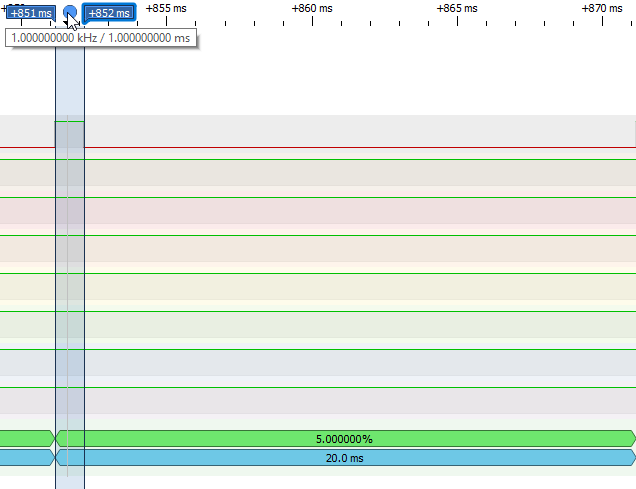
Frequency 50Hz and full duty cycle 20ms.



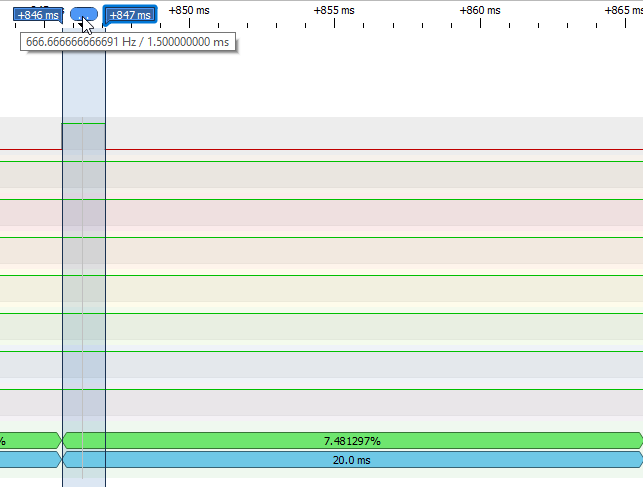
0 Degree Angle (0.55ms)



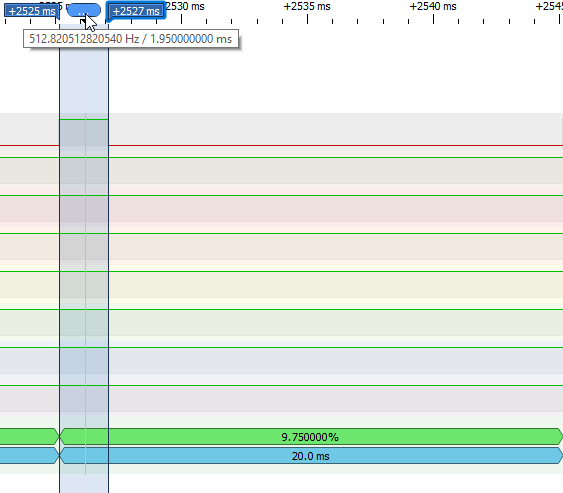
45 Degree Angle (1ms)



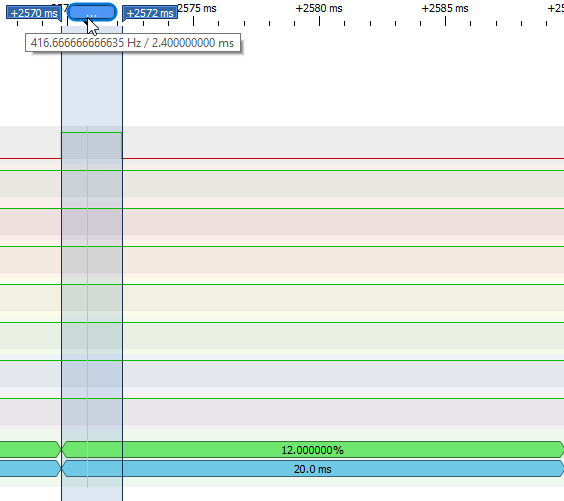
90 Degree Angle (1.5ms)

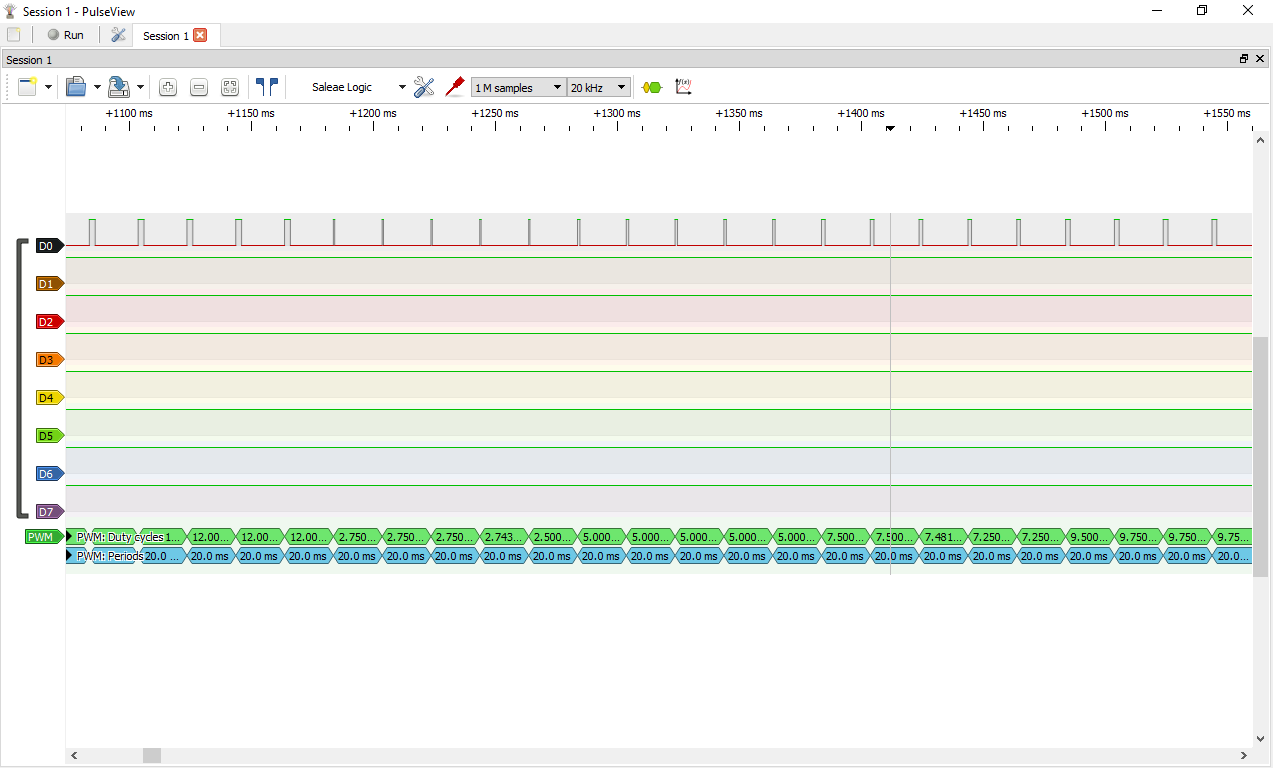


135 Degree Angle (1.95ms)



180 Degree Angle (2.4ms)





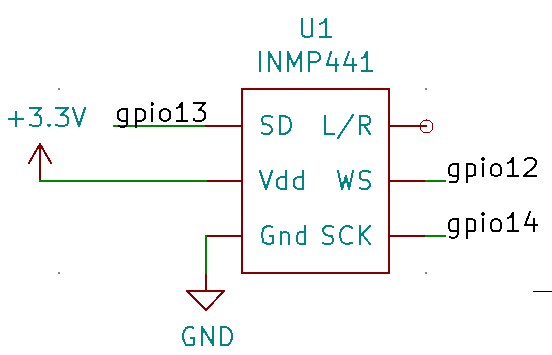
Duty cycle and angle rotation

**Your Testing: Take a video record where you must explain the relationship between duty cycle and servo motor rotation angle as follows.**

* **0 Degree Angle (0.55ms)**
* **45 Degree Angle (1ms)**
* **90 Degree Angle (1.5ms)**
* **135 Degree Angle (1.95ms)**
* **180 Degree Angle (2.4ms)**

***Part D: Digital Microphone with Inter-IC Sound protocol (I2S)***

1. In this scenario, the analog output signal of digital microphone is checked with serial plotter of Arduino IDE.
2. Hardware Required
   1. ESP32
   2. Digital Microphone (mems)
   3. Bread board
   4. Micro USB cable
   5. Jumper Wires
3. Software Required
   1. Arduino IDE
   2. ESP32 Library
4. Circuit Schematic



#include <driver/i2s.h>

#define I2S\_WS 12

#define I2S\_SD 13

#define I2S\_SCK 14

#define I2S\_PORT I2S\_NUM\_0

esp\_err\_t err;

void setup() {

Serial.begin(115200);

Serial.println("Setup I2S ...");

delay(1000);

i2s\_install();

i2s\_setpin();

i2s\_start(I2S\_PORT);

delay(500);

}

void loop() {

int32\_t sample = 0;

int bytes = i2s\_pop\_sample(I2S\_PORT, (char\*)&sample, portMAX\_DELAY);

Serial.print(20000000); //Upper bound in Serial Plotter

Serial.print(" ");

Serial.print(-20000000); //Lower bound in Serial Plotter

Serial.print(" ");

if(bytes > 0){

Serial.println(sample);

}

}

void i2s\_install(){

const i2s\_config\_t i2s\_config = {

.mode = i2s\_mode\_t(I2S\_MODE\_MASTER | I2S\_MODE\_RX),

.sample\_rate = 16000, //16 kHz

.bits\_per\_sample = I2S\_BITS\_PER\_SAMPLE\_32BIT,

.channel\_format = I2S\_CHANNEL\_FMT\_ONLY\_LEFT, //Use left channel

.communication\_format = i2s\_comm\_format\_t(I2S\_COMM\_FORMAT\_I2S | I2S\_COMM\_FORMAT\_I2S\_MSB),

.intr\_alloc\_flags = ESP\_INTR\_FLAG\_LEVEL1, // interrupt level 1

.dma\_buf\_count = 8, // number of buffers

.dma\_buf\_len = 64, // samples per buffer (minimum is 8)

};

err = i2s\_driver\_install(I2S\_PORT, &i2s\_config, 0, NULL);

if (err != ESP\_OK) {

Serial.printf("Failed installing driver: %d\n", err);

while (true);

}

Serial.println("I2S driver installed.");

}

void i2s\_setpin(){

const i2s\_pin\_config\_t pin\_config = {

.bck\_io\_num = I2S\_SCK,

.ws\_io\_num = I2S\_WS,

.data\_out\_num = I2S\_PIN\_NO\_CHANGE,

.data\_in\_num = I2S\_SD

};

err = i2s\_set\_pin(I2S\_PORT, &pin\_config);

if (err != ESP\_OK) {

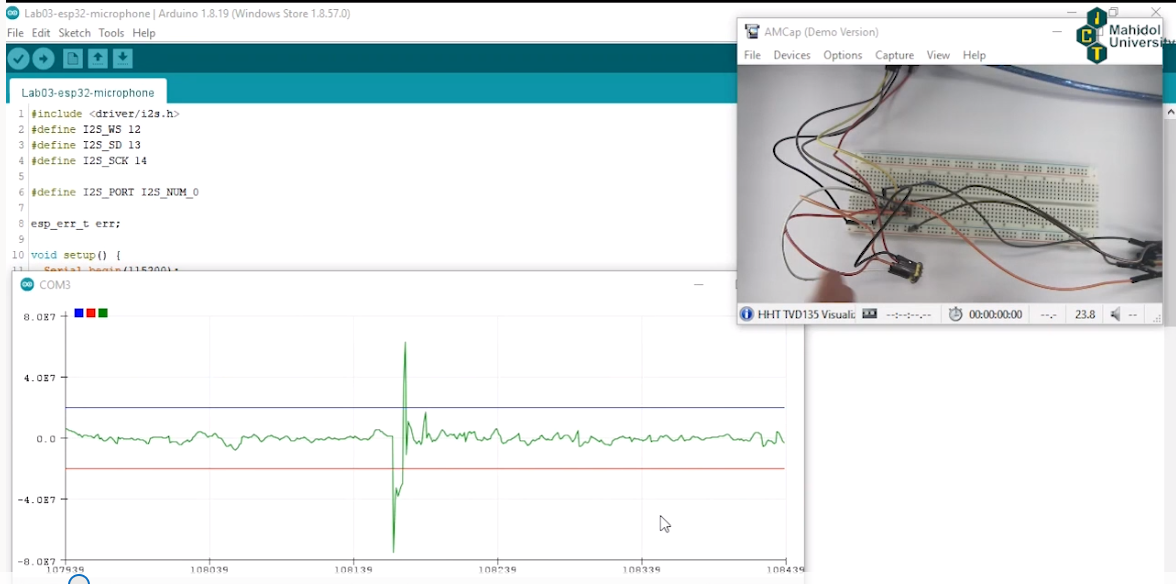
Serial.printf("Failed setting pin: %d\n", err);

while (true);

}

}

1. Result



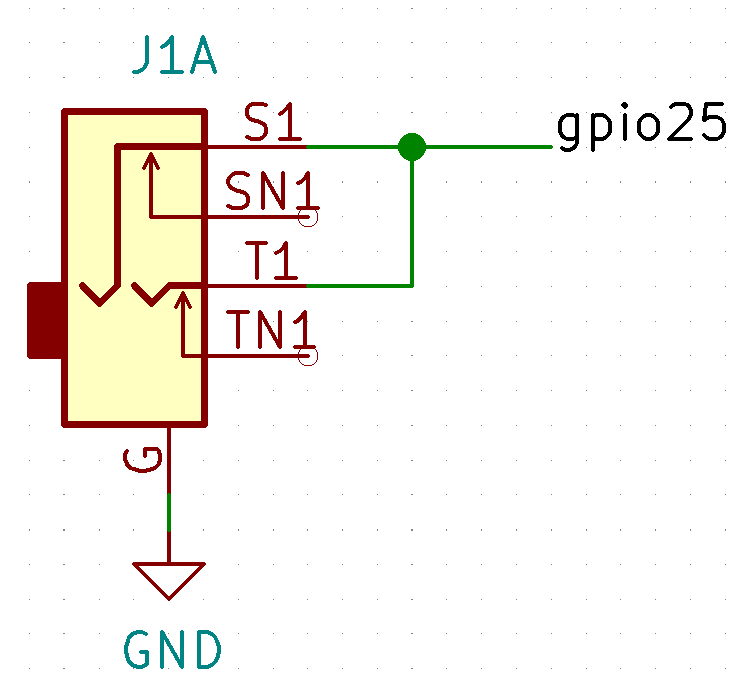
**Your Testing: Take a video record at Serial Plotter when 1) you do not make any sound, 2) you clap your hands, and 3) you turn on music.**

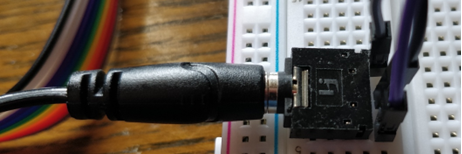
***Part E: Speaker with Digital to Analog Converter (DAC)***

1. In this scenario, the analog output signal of speaker is controlled with POT and checked with Pico scope.
2. Hardware Required
   1. ESP32
   2. Digital Speaker
   3. Stereo Jack
   4. Micro USB cable
   5. Jumper Wires
3. Software Required
   1. Arduino IDE
   2. ESP32 Library
4. Circuit Schematic

Note the pin layout of the female stereo jack you have below. The speakers are powered by the USB port which you can plug it to a computer or a battery charger. The audio jack of the speakers is plugged to the female jack on the breadboard.

1Pins 2 and 5, or S1 and T1 in the diagrams can be interchangeable. They are for audio left and right channels.





1. Code

#include "SoundData.h"

#include "src/XT\_DAC\_Audio/XT\_DAC\_Audio.h"

XT\_Wav\_Class Sound(sample);

XT\_DAC\_Audio\_Class DacAudio(25,0);

uint32\_t DemoCounter=0;

void setup() {

Serial.begin(115200);

Serial.println(A0);

}

void loop() {

DacAudio.FillBuffer();

DacAudio.DacVolume=50;

if(Sound.Playing==false)

DacAudio.Play(&Sound);

Serial.println(DemoCounter++);

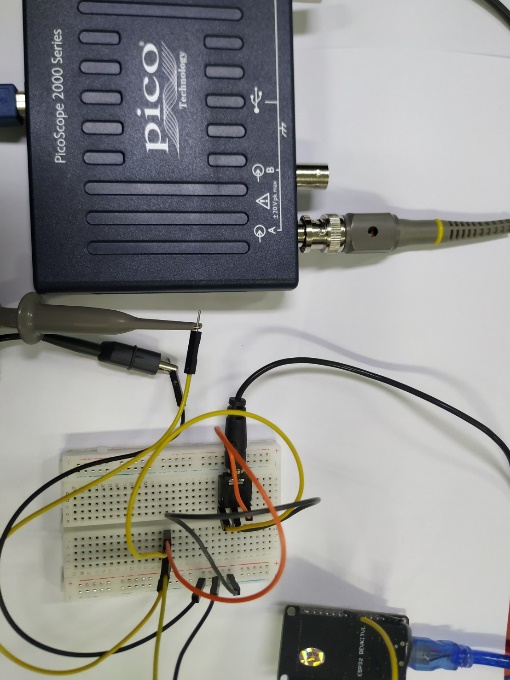
}

1. Results

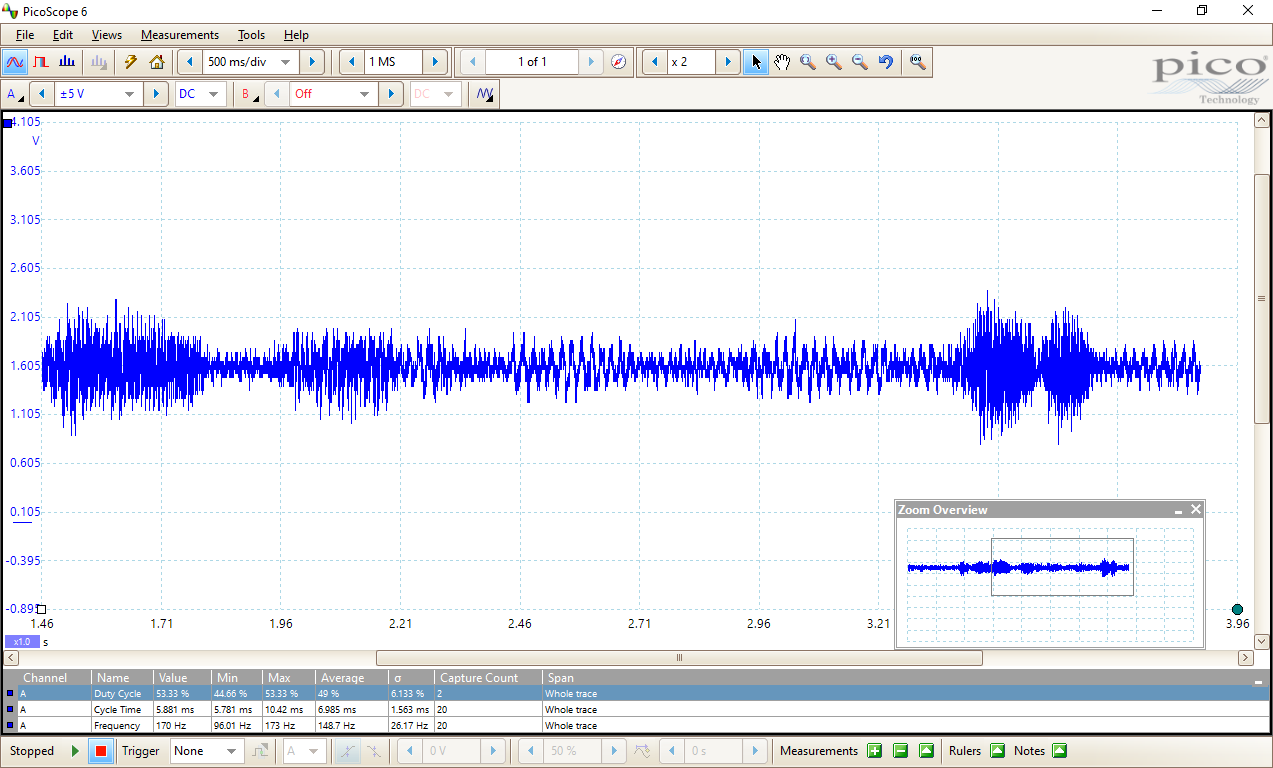
Let’s see the analog output of the audio by using Pico Scope. Use Channel “A” of Pico scope to connect with the output pin of speaker as follow. Each channel has two pins (Signal and GND).

Pico Channel A signal pin -> stereo jack S1 and T1

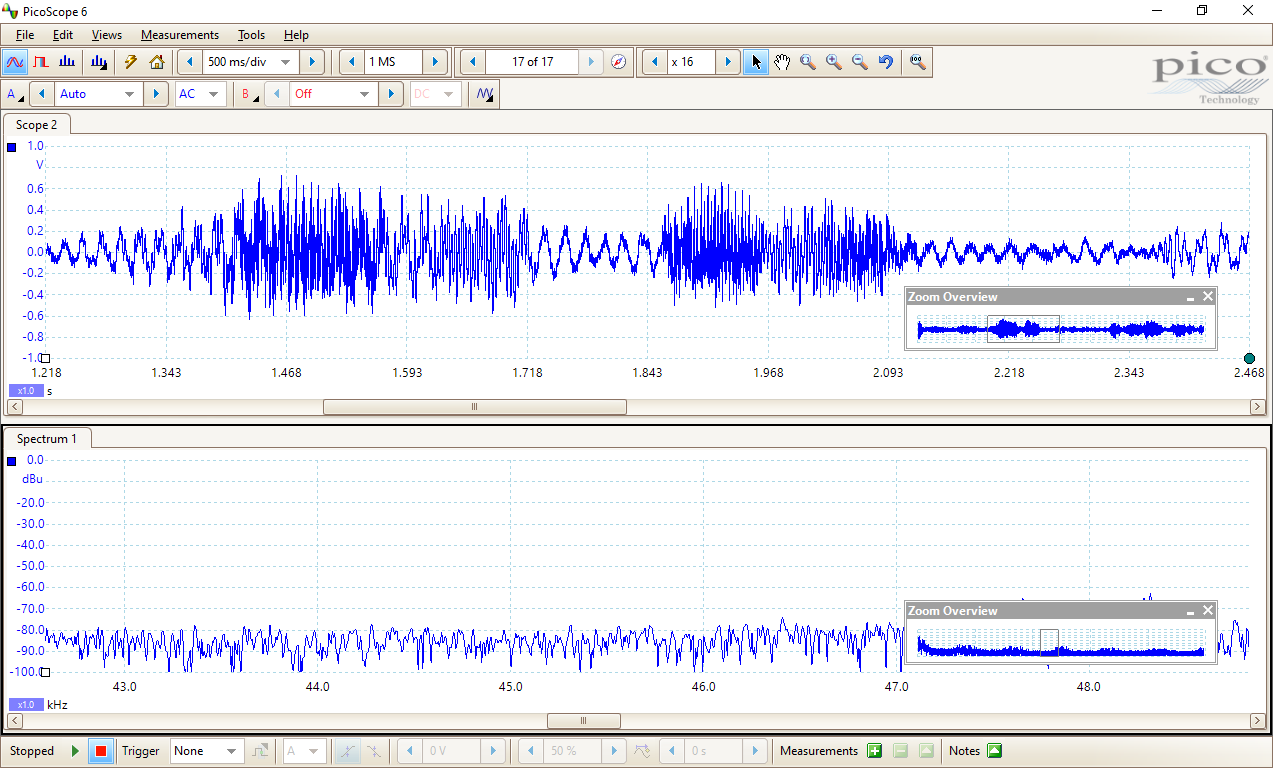
Pico Channel A GND pin -> GND



Here is the analog output signal at Pico Scope.



Go to Views>Add View> Spectrum to check signal level on a vertical axis plotted against frequency on the horizontal axis.



**Your Testing: Take a video record to show the output result of the following conditions.**

Add a potentiometer to GPIO33 like in A1 and map the analogRead’s values to range 0-100 to be the volume value of DacAudio.DacVolume. Then modify the program so that the potentiometer controls the sound volume.