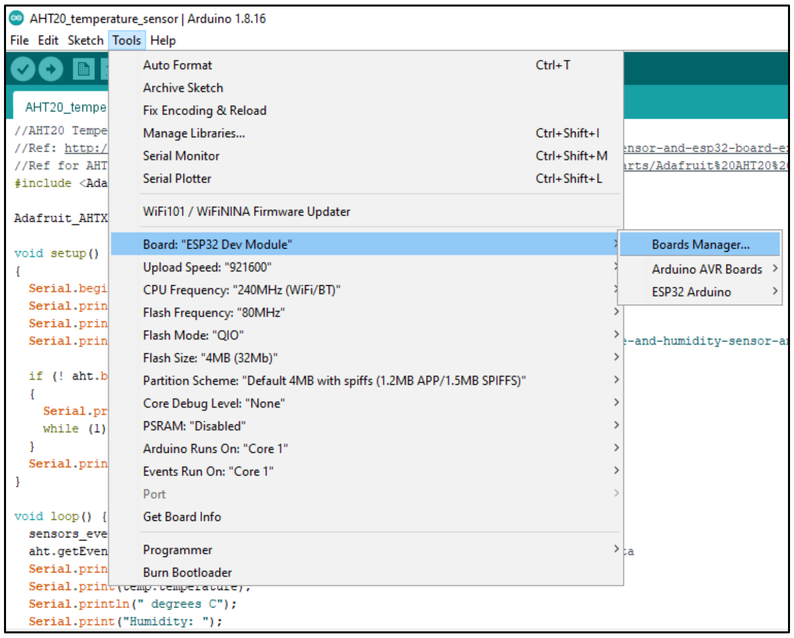
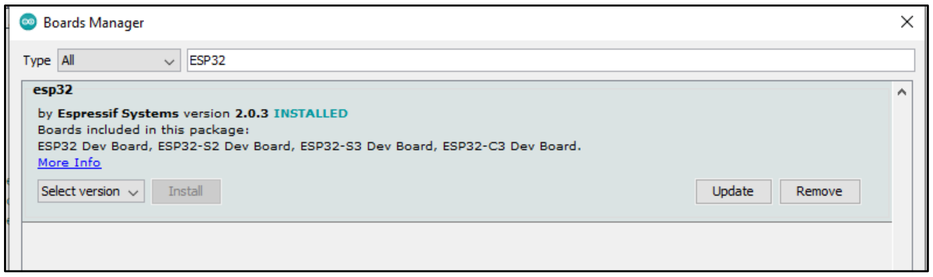
Table of Contents

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
    a. ESP32
    b. LED
    c. Push Button
    d. 220 kΩ Resistor
    e. Bread board
    f. Micro USB cable
    g. Jumper Wires
3. Software Required
    a. Arduino IDE
    b. ESP32 Library
        i. To install it, Open Arduino IDE application and go to "Tools -> Board -> Boards Manager.


AHT20_temperature_sensor | Arduino 1.8.16
File Edit Sketch Tools Help
AHT20_tempe
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Dev Module"
Upload Speed: "921600"
CPU Frequency: "240MHz (WiFi/BT)"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
Core Debug Level: "None"
PSRAM: "Disabled"
Arduino Runs On: "Core 1"
Events Run On: "Core 1"
Port
Get Board Info
Programmer
Burn Bootloader
Boards Manager...
Arduino AVR Boards
ESP32 Arduino

        ii. Then, search "ESP32" library and install it.



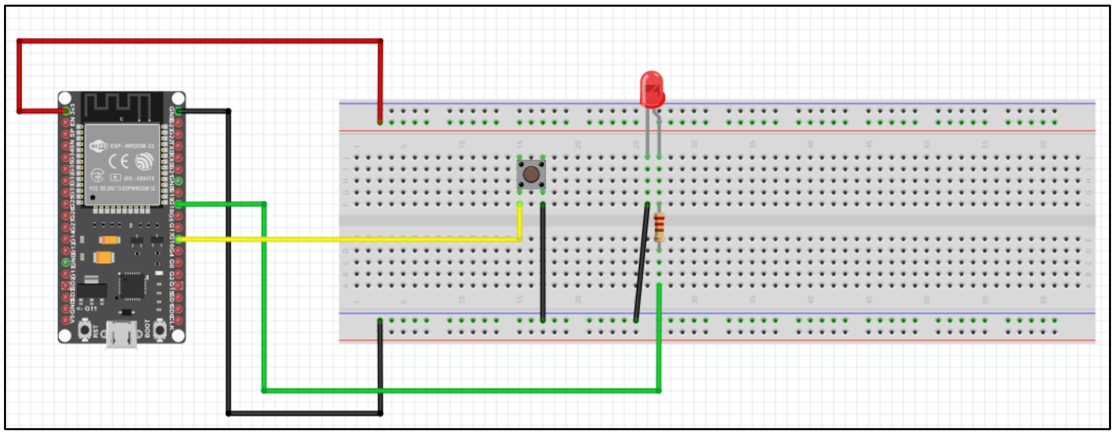
esp32
by Espressif Systems version 2.0.3 INSTALLED
Boards included in this package:
ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-S3 Dev Board, ESP32-C3 Dev Board.
More Info

4. 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".

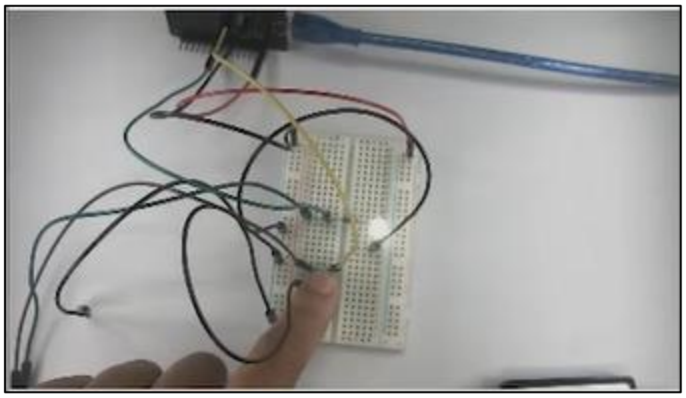
5. Circuit Schematic
   a. LED anode -> 220 kΩ -> GPIO 18
   b. LED cathode -> GND
   c. Push button one pin -> GPIO 14
   d. Push button another pin -> GND

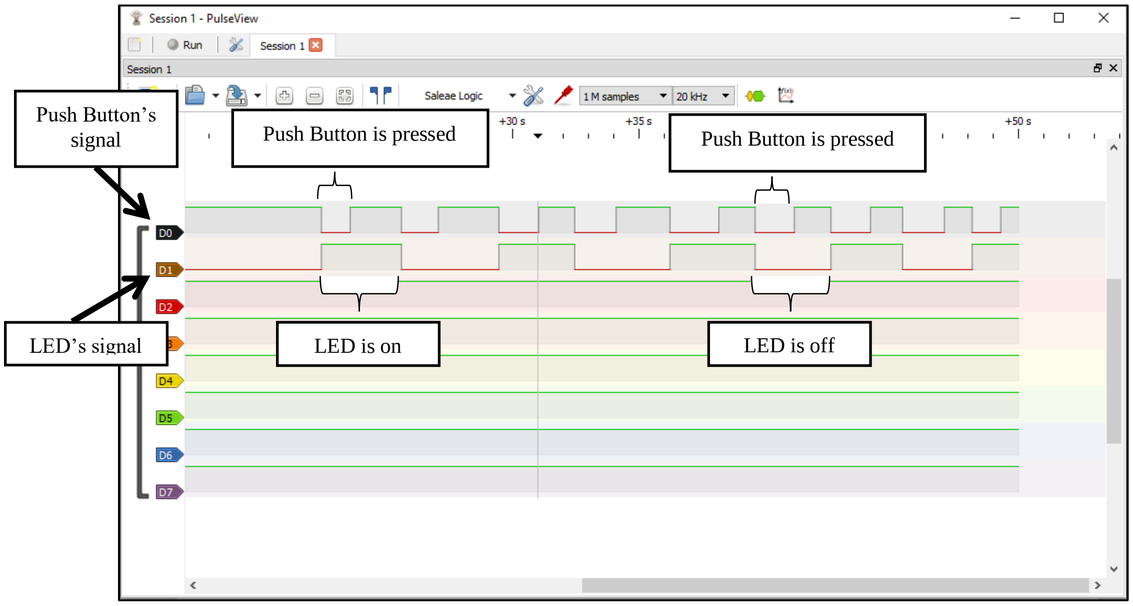


6. Source Code

```
#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);
}
```

7. Connecting the Logic Analyzer with the circuit
    a. LED anode -> Ch1 (Logic Analyzer)
    b. Push button (pin connected with GPIO 16) -> Ch2 (Logic Analyzer)
    c. ESP32 GND -> GND (Logic Analyzer)
8. Results





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
   a. ESP32
   b. RGB LED
   c. 10k Potentiometer
   d. 3 x 220 kΩ Resistors
   e. Bread board
   f. Micro USB cable
   g. Jumper Wires
3. Software Required
   a. Arduino IDE
   b. ESP32 Library
4. How to check common cathode/ anode of RGB LED?

   RGB-LED includes 4 pins:

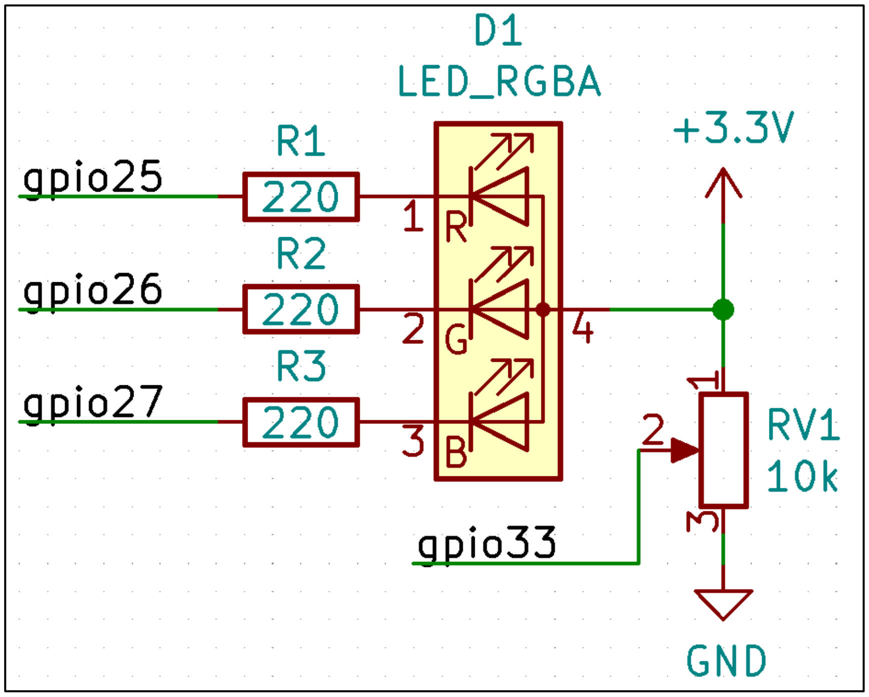
   R(red) pin is to control the red color element.
   G(green) pin is to control the green color element.
   B(blue) pin is to control the blue color element.
   Common (Cathode/Anode) pin is to connect GND (0V) or Vin (3.3V).

   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]

[(Part 1) Check if RGB LED is common cathode or anode - YouTube]

5. Circuit Schematic
    a. RGB LED Red -> 220 kΩ -> GPIO 25
    b. RGB LED Green -> 220 kΩ -> GPIO 26
    c. RGB LED Blue -> 220 kΩ -> GPIO 27
    d. RGB LED Common Cathode -> GND, if RGB LED Common Anode -> 3.3V
    e. POT Pin 1 -> 3.3V
    f. POT Pin 2 -> GPIO 33
    g. POT Pin 3 -> GND



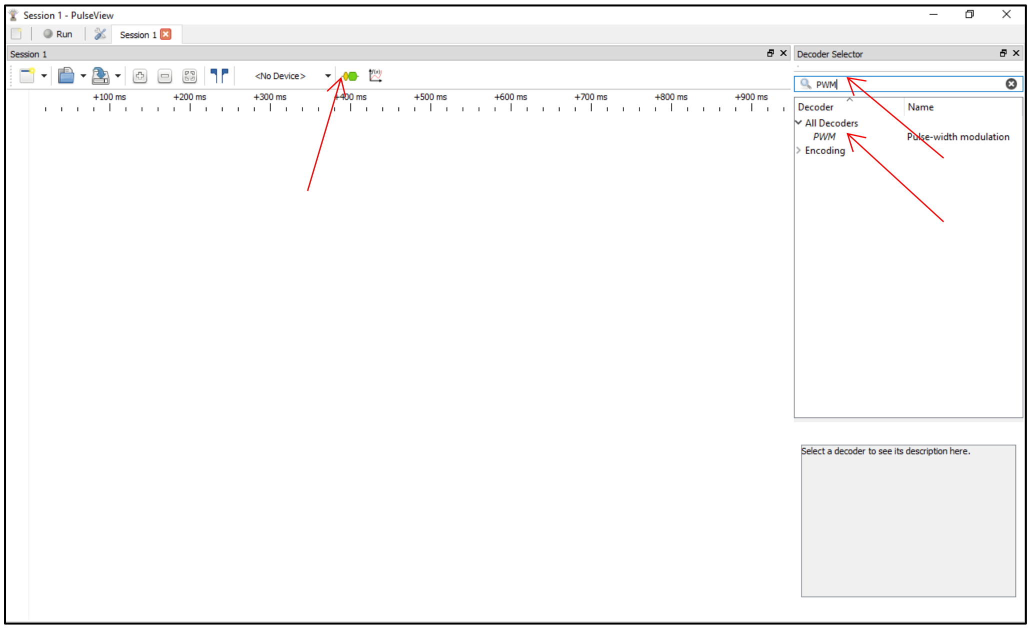
6. Source Code

```
#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;
```

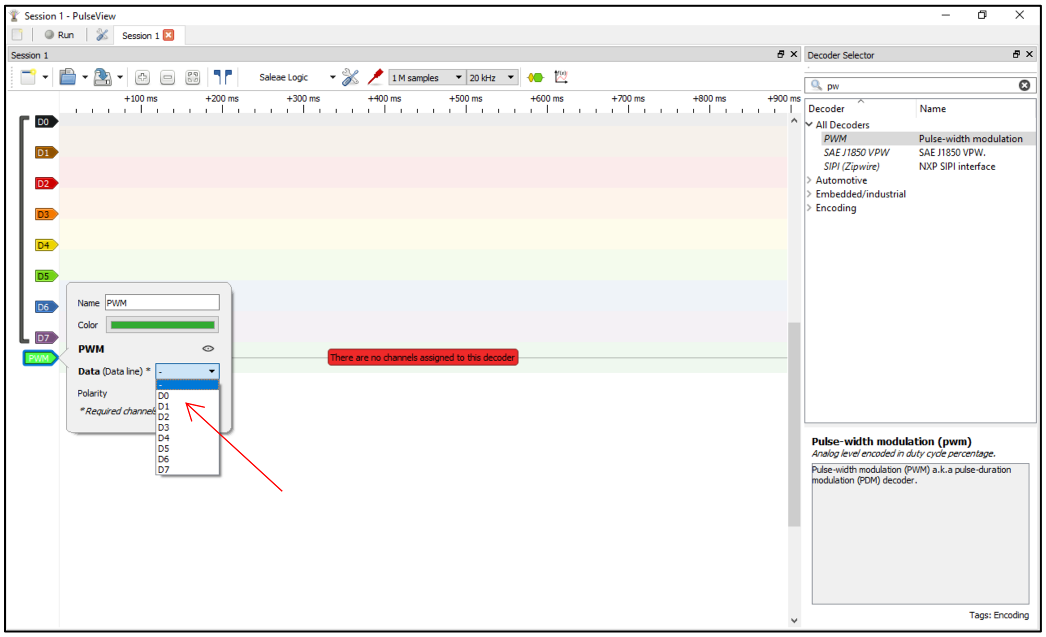
```
void setup() {
  Serial.begin(9600);
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  pinMode(BLUE, OUTPUT);

  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);
}
```

7. Connecting the Logic Analyzer with the circuit
   a. RGB LED Red -> Ch1 (Logic Analyzer)
   b. RGB LED Green-> Ch2 (Logic Analyzer)
   c. RGB LED Blue -> Ch3 (Logic Analyzer)
   d. ESP32 GND -> GND (Logic Analyzer)

8. 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.



Name PWM

Color

PWM

Data (Data line) * -

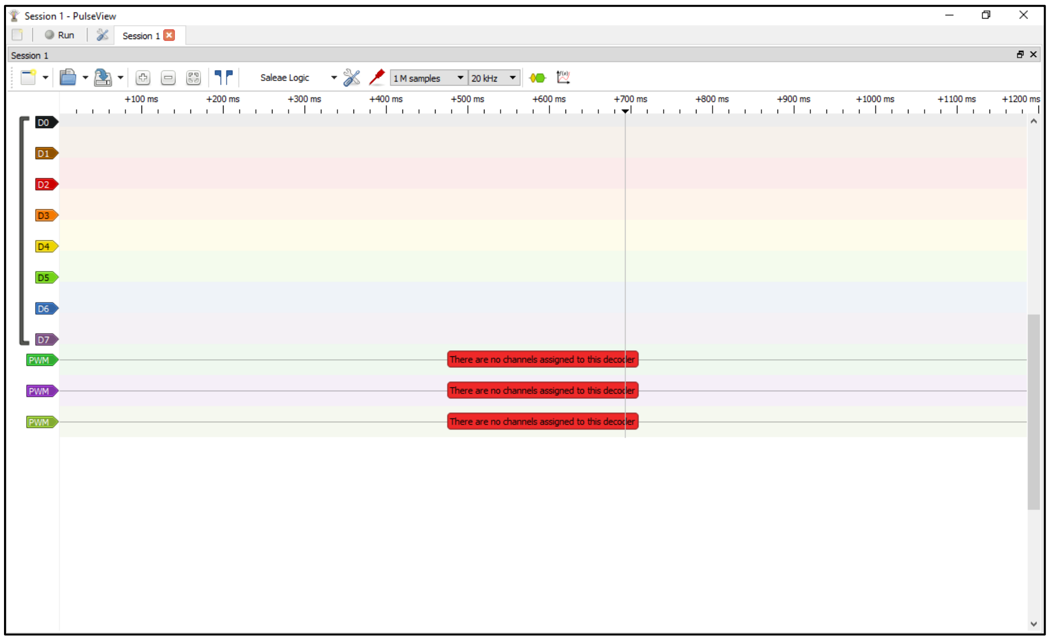
Polarity

*Required channel

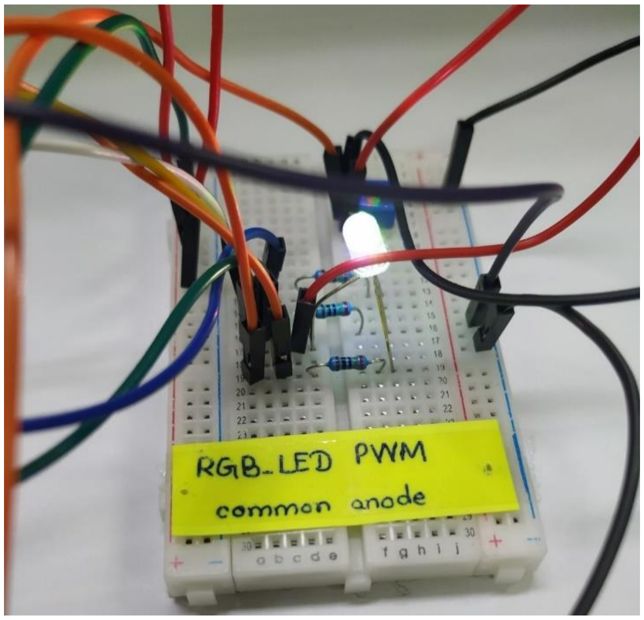
Decoder	Name
All Decoders	
PWM	Pulse-width modulation
SAE J1850 VPW	SAE J1850 VPW.
SIPI (Zipwire)	NXP SIPI interface
Automotive	
Embedded/industrial	
Encoding	

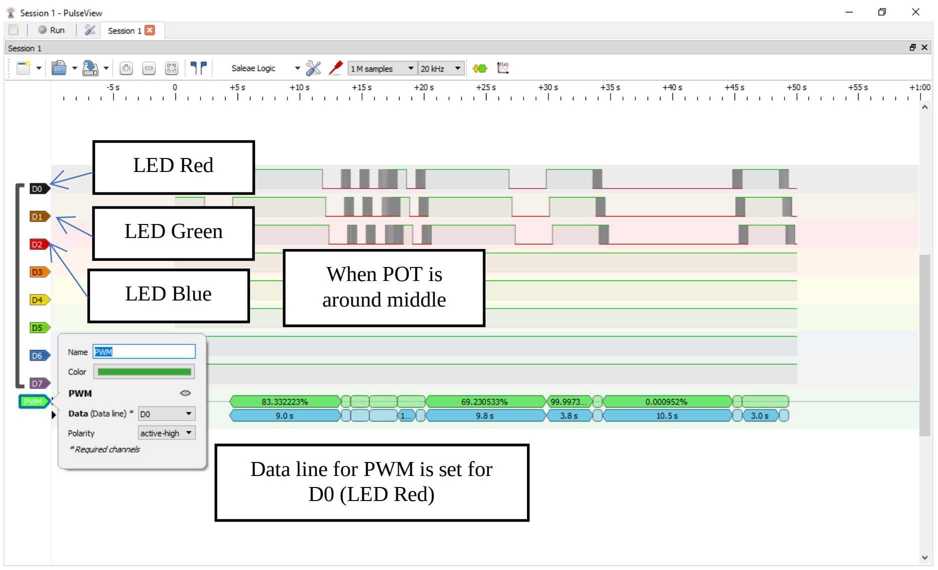
Pulse-width modulation (pwm)
Analog level encoded in duty cycle percentage.
Pulse-width modulation (PWM) a.k.a pulse-duration modulation (PDM) decoder.

You can add more PWM channels.


Session 1 - PulseView
Run
Session 1
Saleae Logic
1 M samples
20 kHz
D0
D1
D2
D3
D4
D5
D6
D7
PWM
PWM
PWM
There are no channels assigned to this decoder
There are no channels assigned to this decoder
There are no channels assigned to this decoder

9. Results





Name: PWM
Color
PWM
Data (Data line) *: D0
Polarity: active-high
* Required channels

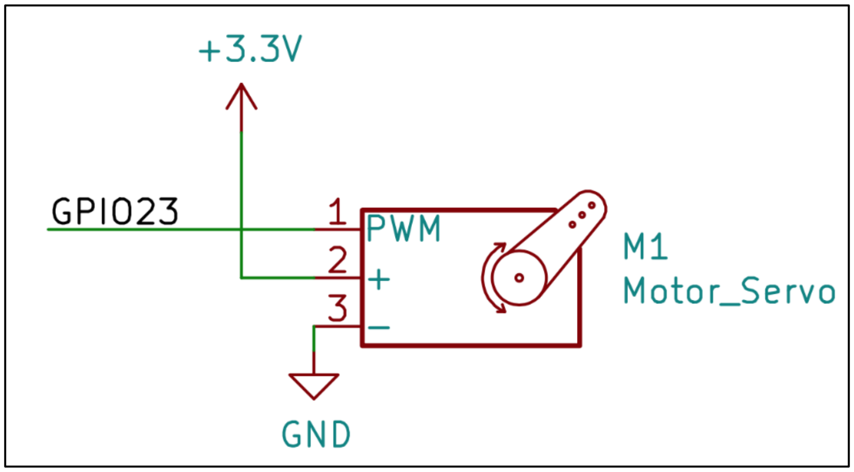
Your Testing: Take a video record where you must explain the following cases.

- 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
   a. ESP32
   b. Micro Servo 9g
   c. Bread board
   d. Micro USB cable
   e. Jumper Wires
3. Software Required
   a. Arduino IDE
   b. ESP32 Library
4. Micro Servo

5. Circuit Schematic
   h. Servo Pin 1 -> GPIO 23
   i. Servo Pin 2 -> 3.3V
   j. Servo Pin 3 -> GND

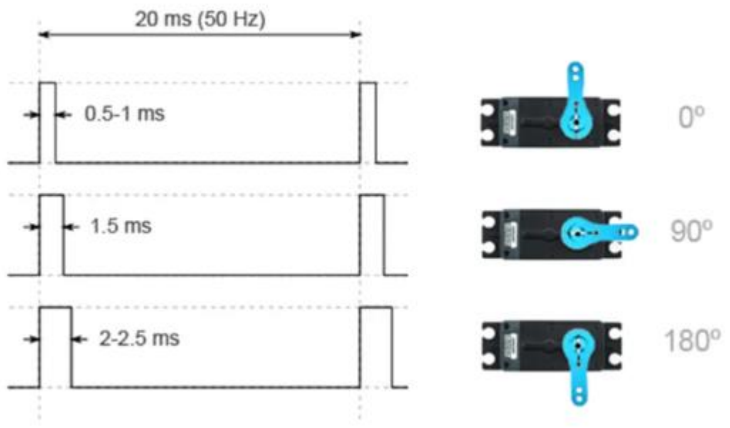


6. Source Code

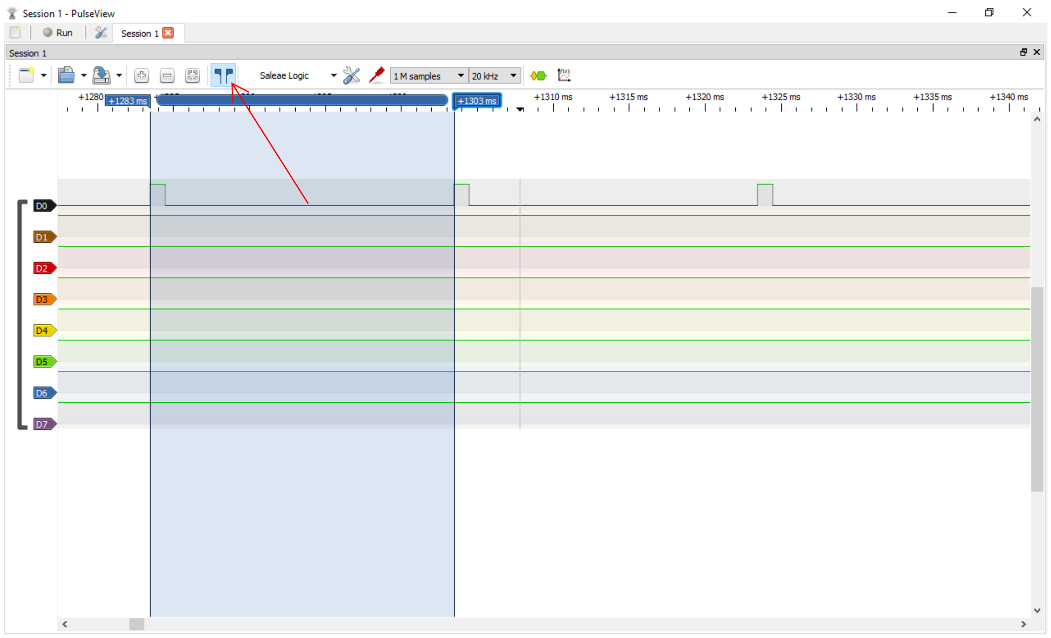
```
#include <Arduino.h>
#include <Servo.h>
Servo servo;
```

```
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);
}
```

7. Connecting the Logic Analyzer with the circuit
   a. Servo Pin 1 -> Ch1 (Logic Analyzer)
   b. ESP32 GND -> GND (Logic Analyzer)
8. Results
   Here is the relationship between duty cycle and servo rotation angle.

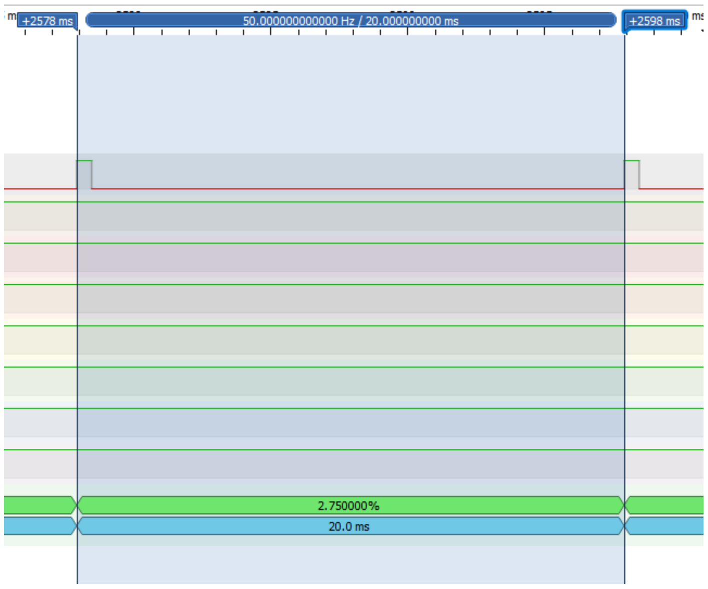


At Pulse View, “show cursors ” is used to read starting and ending timing frame.

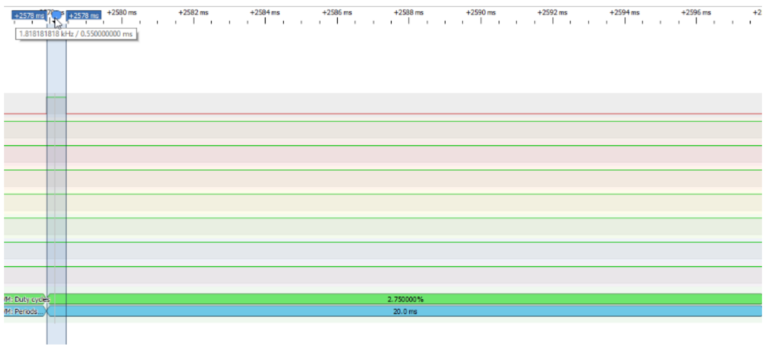


Logic Scope with Pulse View Result of Servo Motor PWM

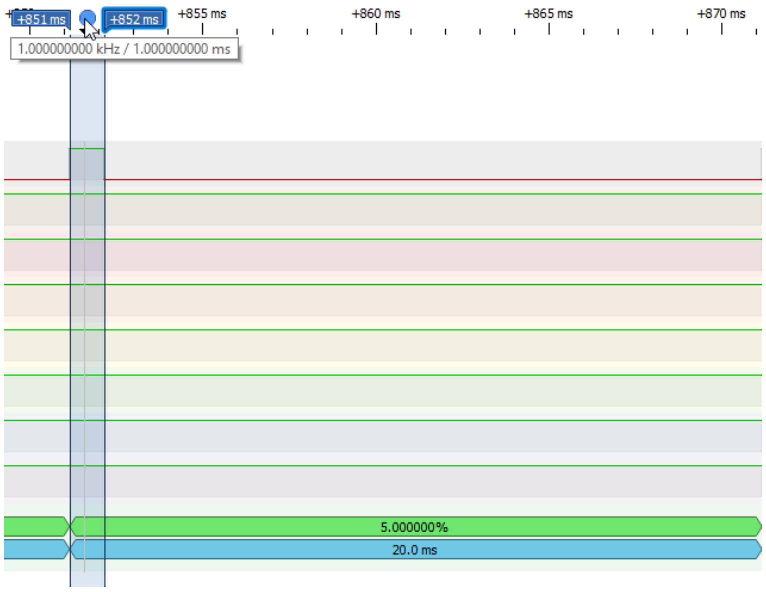
Frequency 50Hz and full duty cycle 20ms.



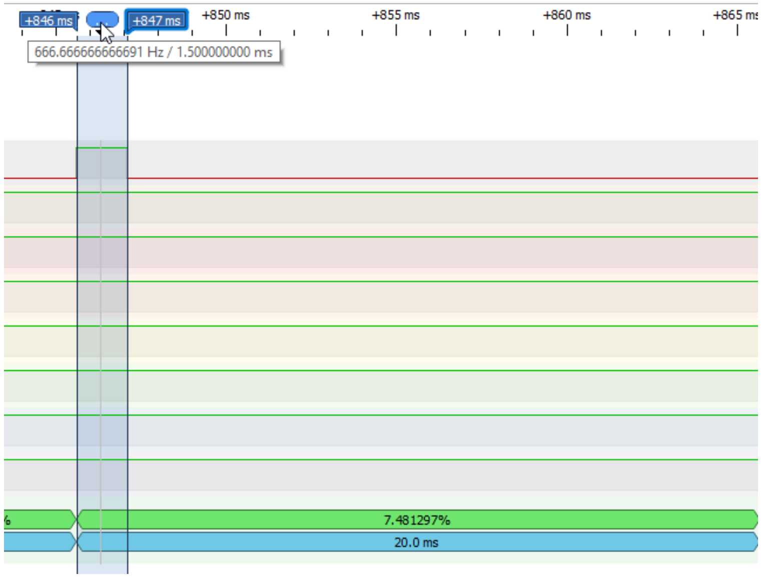
0 Degree Angle (0.55ms)



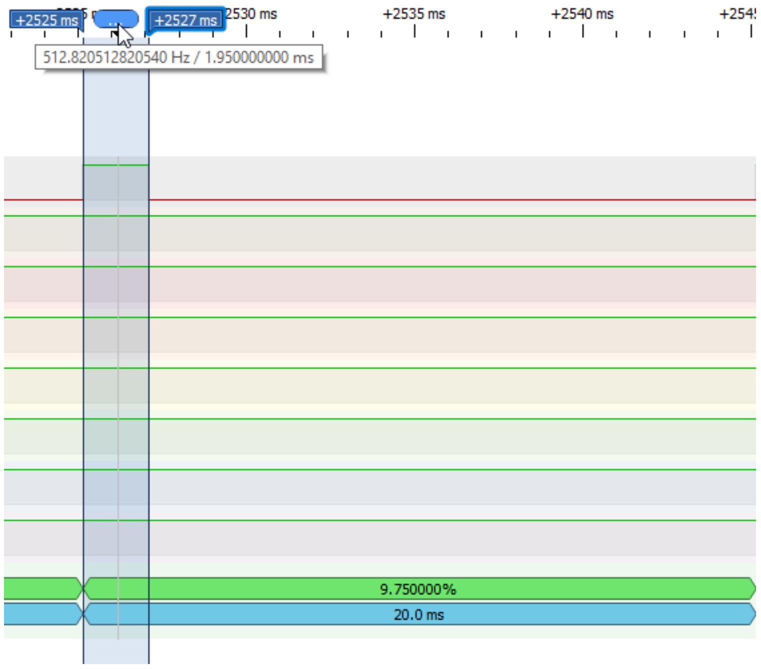
45 Degree Angle (1ms)



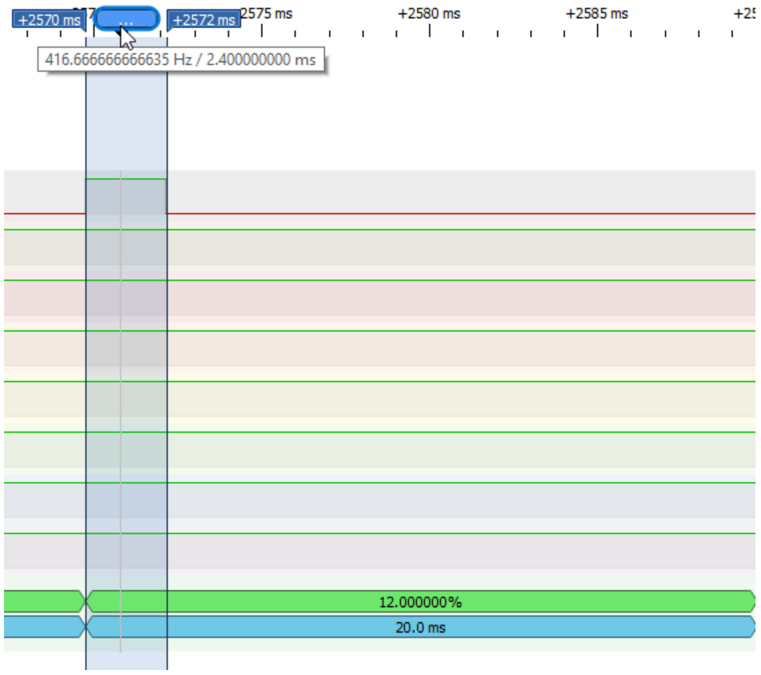
90 Degree Angle (1.5ms)

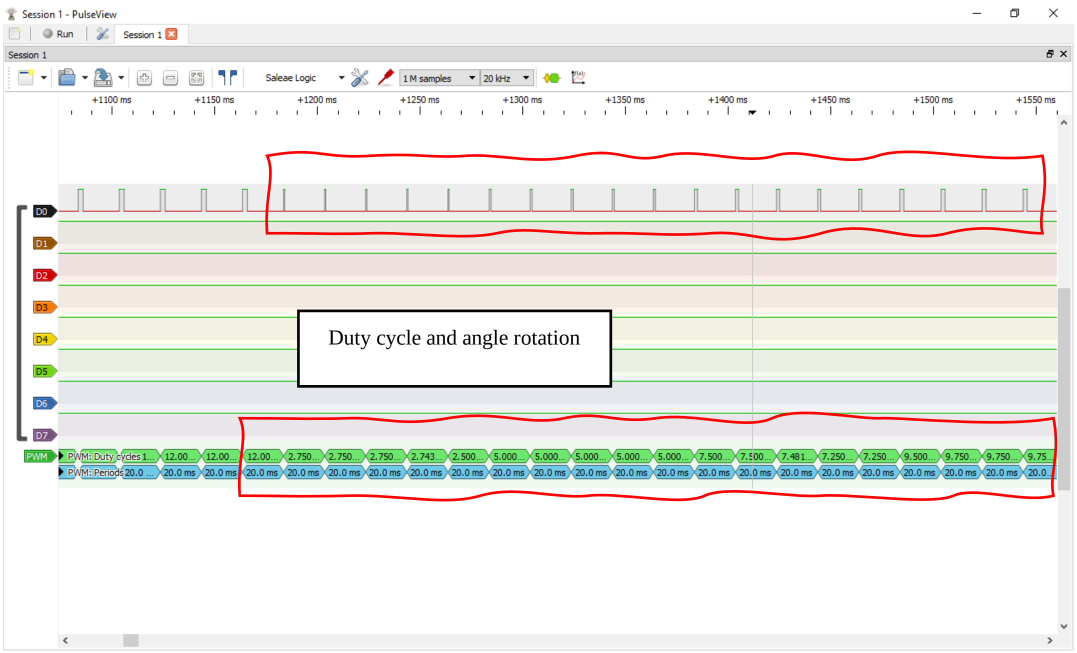


135 Degree Angle (1.95ms)



180 Degree Angle (2.4ms)





```
PWM: Duty cycles 1... 12.00... 12.00... 12.00... 2.750... 2.750... 2.750... 2.743... 2.500... 5.000... 5.000... 5.000... 5.000... 5.000... 7.500... 7.500... 7.481... 7.250... 7.250... 9.500... 9.750... 9.750... 9.75...
PWM: Periods 20.0 ... 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0 ms 20.0...
```

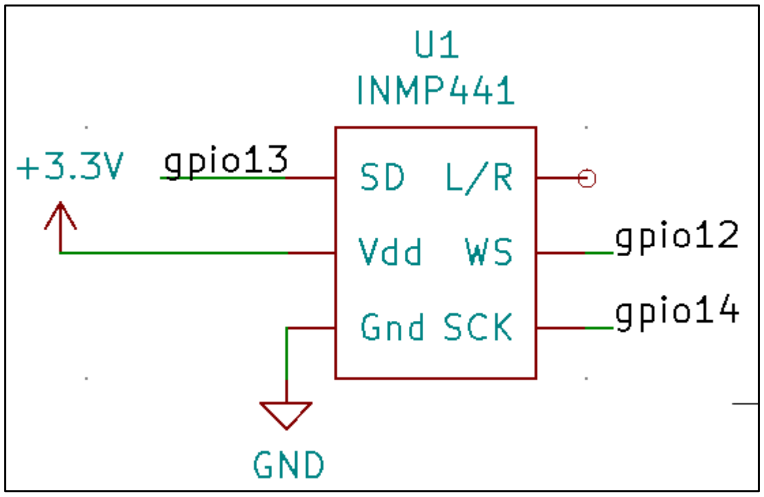
Your Testing: Take a video record where you must explain the relationship between duty cycle and servo motor rotation angle as follows.

- **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)

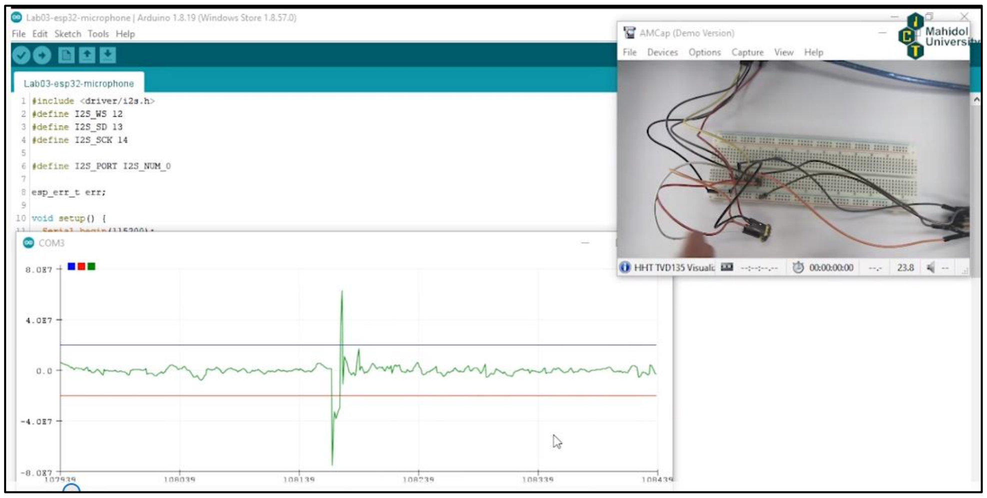
1. In this scenario, the analog output signal of digital microphone is checked with serial plotter of Arduino IDE.
2. Hardware Required
    a. ESP32
    b. Digital Microphone (mems)
    c. Bread board
    d. Micro USB cable
    e. Jumper Wires
3. Software Required
    a. Arduino IDE
    b. ESP32 Library

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);
  }
}
```

5. Result


Lab03-esp32-microphone | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help
Lab03-esp32-microphone
COM3
AMCap (Demo Version)
File Devices Options Capture View Help
Mahidol University
8.0E7
4.0E7
0.0
-4.0E7
-8.0E7

```
#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() {
```

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.

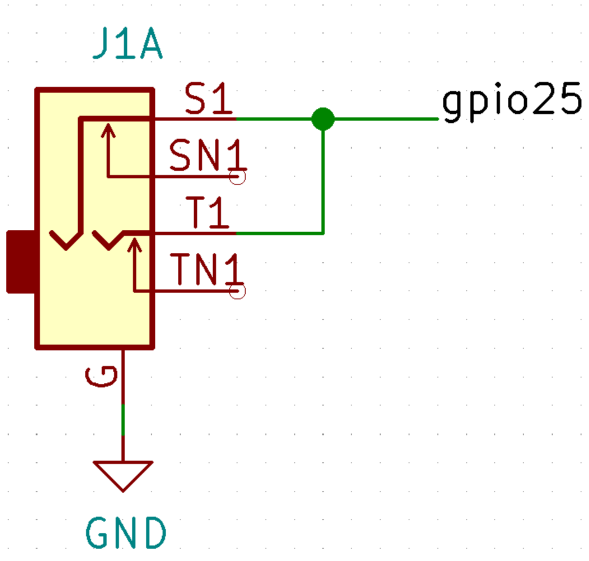
Part E: Speaker with Digital to Analog Converter (DAC)

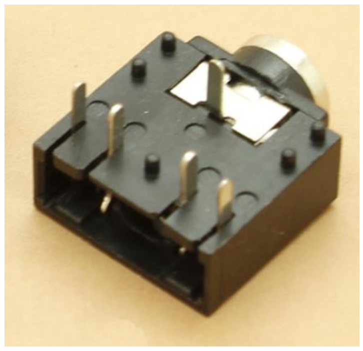
1. In this scenario, the analog output signal of speaker is controlled with POT and checked with Pico scope.
2. Hardware Required
   a. ESP32
   b. Digital Speaker
   c. Stereo Jack
   d. Micro USB cable
   e. Jumper Wires

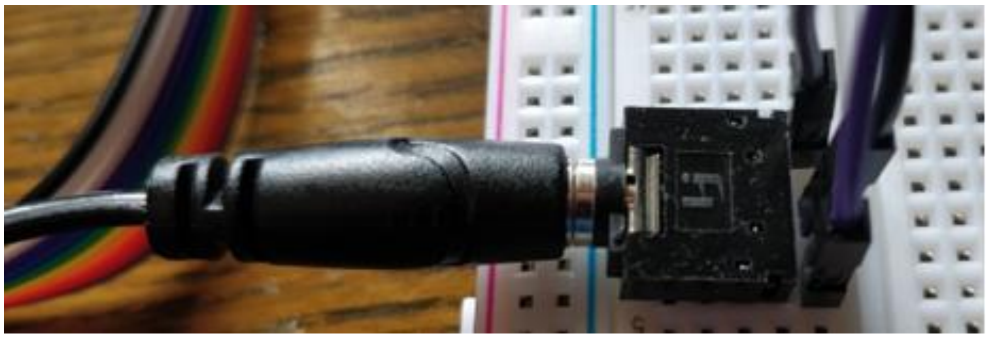
3. Software Required
    a. Arduino IDE
    b. ESP32 Library
4. Circuit Schematic

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.

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







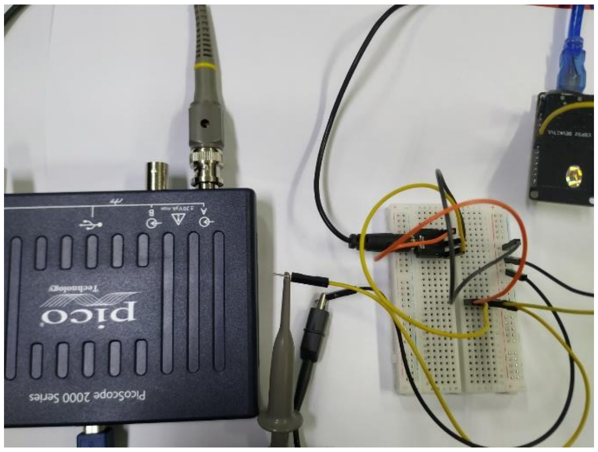
5. Code

```
#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);
}
```

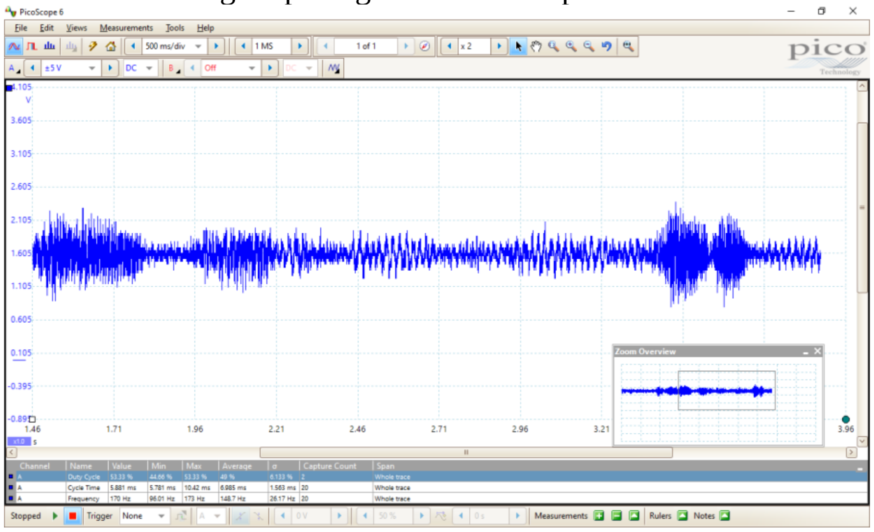
```
void loop() {
  DacAudio.FillBuffer();
  DacAudio.DacVolume=50;
  if(Sound.Playing==false)
    DacAudio.Play(&Sound);
  Serial.println(DemoCounter++);
}
```

6. 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

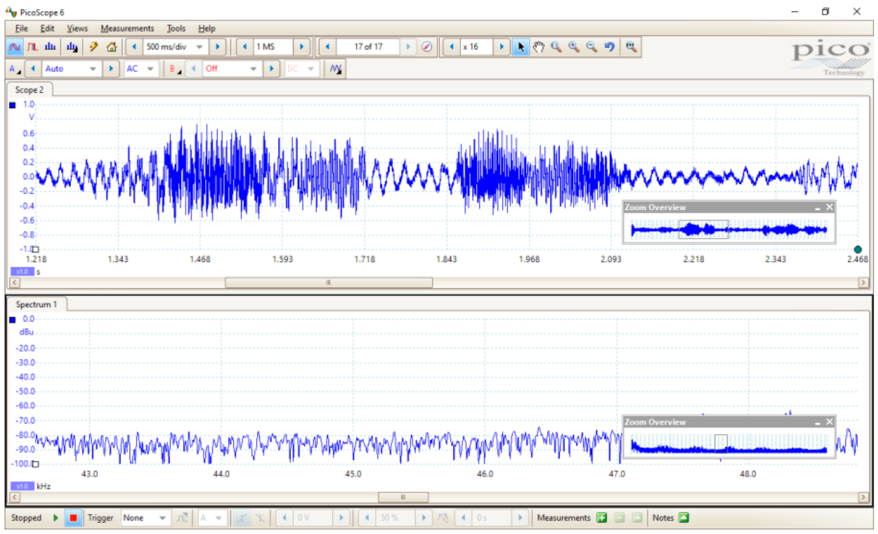


Here is the analog output signal at Pico Scope.



Channel	Name	Value	Min	Max	Average	σ	Capture Count	Span
A	Duty Cycle	53.33 %	44.66 %	53.33 %	49 %	6.133 %	2	Whole trace
A	Cycle Time	5.881 ms	5.781 ms	10.42 ms	6.985 ms	1.563 ms	20	Whole trace
A	Frequency	170 Hz	96.01 Hz	173 Hz	148.7 Hz	26.17 Hz	20	Whole trace

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.