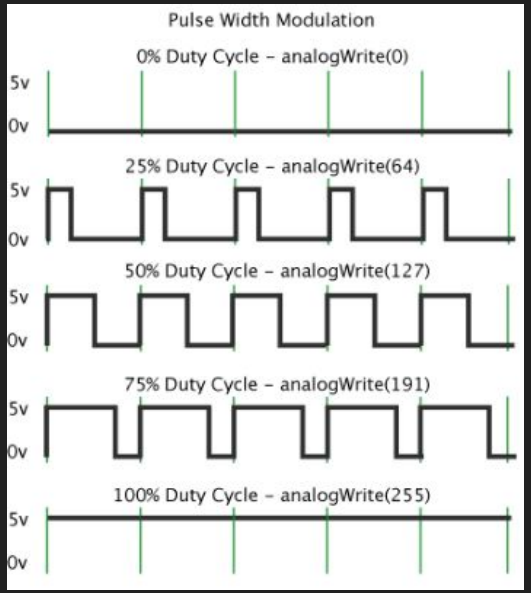
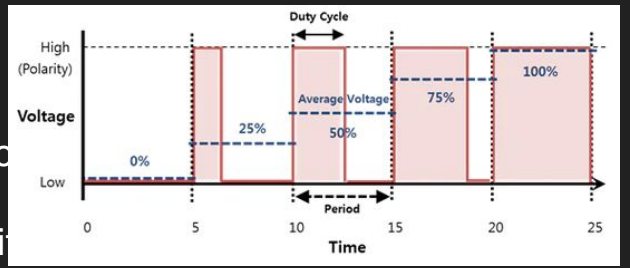
ESP32

Pulse Width Modulation

PWM

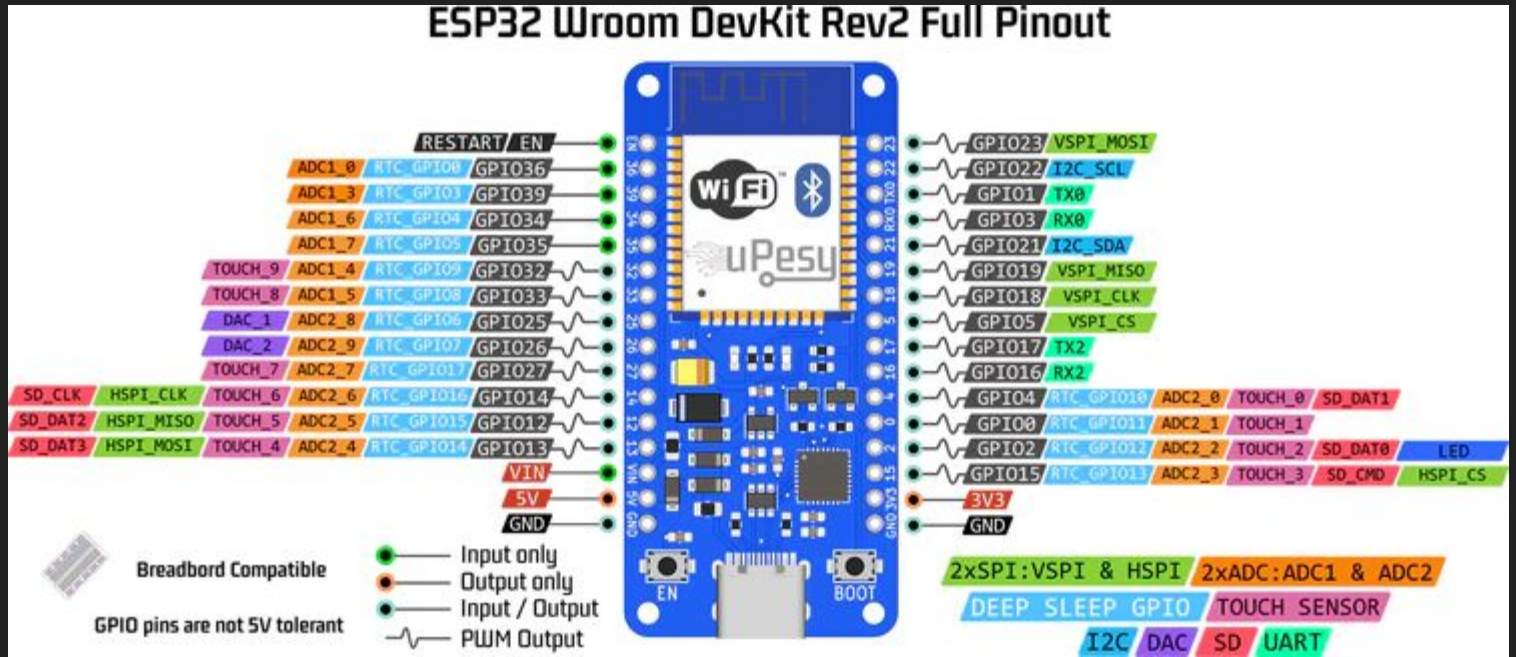
PWM-pulse width modulation
-used with digital pins to mimic analog output signals
-can be used for example to dim a light or control the speed of a motor
-in an Arduino the PWM frequency occurs at 500 Hz=500 times per second (in ESP32 this can be up to 40MHz)
-500 times per second=500 times per 1000 milliseconds
-500/1000=1 pulse per 2 milliseconds
-this means that the Arduino reads and interprets an electrical pulse every 2 ms (2ms is the period)
-in these 2ms you can send a HIGH or LOW signal for a specific duration
-this duration is known as the duty cycle
-a 25% duty cycle means that the signal is high for a half of a millisecond in each 2ms period
-a 50% duty cycle means that it is HIGH for half the period width or fo 1 ms and so on
-in the Arduino/ESP32 we set the duty cycle by using analogWrite(bi resolution range-8 bits Arduino and up to 16 bits ESP32) with 0 meaning 0% duty cycle and 2^8*Arduino) or 2^16(ESP32) being 100% duty cycle and any value in between being a percentage of 255 or 65536 for the duty cycle





PWM

The duty cycle determines how often the LED or motor is turned on and off. The more the LED is on the more it appears to be bright and the less it's on the less bright it appears. Same with a motor: higher duty cycle means power on for longer and therefore more power in general.



You can only use PWM on specific pins. In most microcontrollers this is designated by the ~ symbol. It is also normally correlated with pins capable of DAC (digital to analog). You can not use PWM on pins 34,35,36 and 39.

PWM

To use customized PWM in the esp32 you must follow 5 steps:

- Choose a PWM channel (0 - 15)
- Choose the PWM frequency
- Choose the resolution of the pulse width between 1 and 16 bits

  (1 bit=0-1, 2 bits=0-3, 3 bits=0-7, 4 bits=0-15 ….8 bits=0-255…16 bits=0-65535)

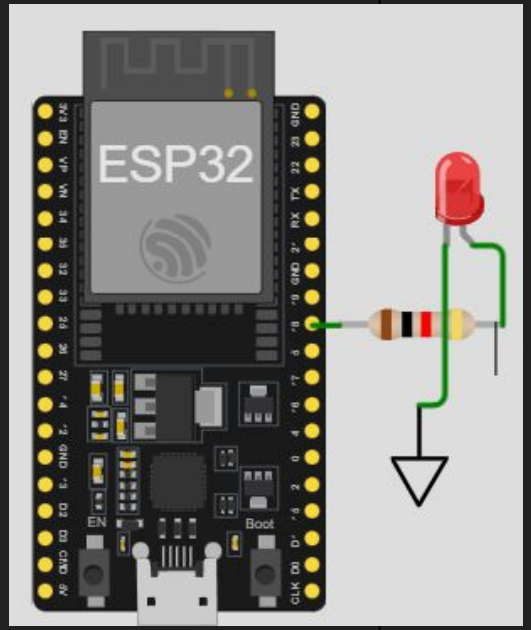
  Note: you can attach multiple pins to the same channel

- Choose the GPIO pin which will generate the PWM signal
- Assign the value of the voltage you want at the output

```
int pwmChannel = 0; // Selects channel 0
int frequence = 1000; // PWM frequency of 1 KHz
int resolution = 8; // 8-bit resolution, 256 possible values
int pwmPin = 23;

void setup(){
    // Configuration of channel 0 with the chosen frequency and resolution
    ledcSetup(pwmChannel, frequence, resolution);

    // Assigns the PWM channel to pin 23
    ledcAttachPin(pwmPin, pwmChannel);

    // Create the selected output voltage
    ledcWrite(pwmChannel, 127); // 1.65 V
}

void loop(){
}
```

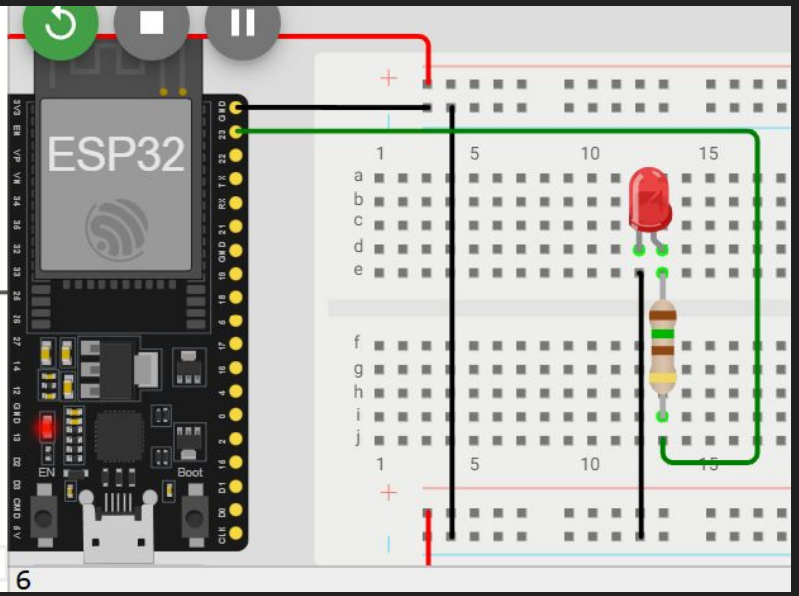
Try this code in wokwi-attach LED to pin 18 of the esp32

```
const int PWM_CHANNEL = 0;    // ESP32 has 16 channels which can generate 16 independent waveforms
const int PWM_FREQ = 1000;     // Recall that Arduino Uno is ~490 Hz. Official ESP32 example uses 5,000Hz
const int PWM_RESOLUTION = 8;// We'll use same resolution as Uno (8 bits, 0-255) but ESP32 can go up to 16 bits
// The max duty cycle value based on PWM resolution (will be 255 if resolution is 8 bits)
const int MAX_DUTY_CYCLE = (int)(pow(2, PWM_RESOLUTION) - 1);
const int LED_1_OUTPUT_PIN = 18;
const int DELAY_MS = 4; // delay between fade increments
void setup() {
  // Sets up a channel (0-15), a PWM duty cycle frequency, and a PWM resolution (1 - 16 bits)
  // ledcSetup(uint8_t channel, double freq, uint8_t resolution_bits);
  ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RESOLUTION);
  // ledcAttachPin(uint8_t pin, uint8_t channel);
  ledcAttachPin(LED_1_OUTPUT_PIN, PWM_CHANNEL);
}
void loop() {
  // fade up PWM on given channel
  for(int dutyCycle = 0; dutyCycle <= MAX_DUTY_CYCLE; dutyCycle++){
    ledcWrite(PWM_CHANNEL, dutyCycle);
    delay(DELAY_MS);
  }
  // fade down PWM on given channel
  for(int dutyCycle = MAX_DUTY_CYCLE; dutyCycle >= 0; dutyCycle--){
    ledcWrite(PWM_CHANNEL, dutyCycle);
    delay(DELAY_MS);
  }
}
```



Second Method that Avoids Using a Channel (use this for newer Arduino IDE)(only supported with Arduino 3.x+)

```
void setup() {
  Serial.begin(115200);
  ledcAttach(23,5000,10);
  for(int x=0;x<1024;x++)
  {
    Serial.println(x);
    ledcWrite(23,x);
    delay(10);
  }
  for(int x=1023;x>=0;x--)
  {
    Serial.println(x);
    ledcWrite(23,x);
    delay(10);
  }
}
```



Recommendations

You can not arbitrarily choose the frequency and resolution. These are dependant on the ESP32 model and the resolution supported changes depending on the frequency. Normally the higher the frequency the lower the available resolution. For example a frequency of 5KHz can have a maximum resolution of 13 bits while a higher frequency of 20 KHz can have only a 2 bit resolution.

Normally a 500-1000 Hz frequency with an 8 bit resolution should be enough for most tasks).

Common Resolutions and Frequencies

Resolution (bits)	Max Frequency (Hz)
1-bit	40,000,000 (40 MHz)
2-bit	20,000,000 (20 MHz)
3-bit	10,000,000 (10 MHz)
4-bit	5,000,000 (5 MHz)
8-bit	312,500 (312.5 kHz)
10-bit	78,125 (78.125 kHz)
12-bit	19,531 (19.531 kHz)
13-bit	9,766 (9.766 kHz)
14-bit	4,883 (4.883 kHz)
15-bit	2,441 (2.441 kHz)
16-bit	1,221 (1.221 kHz)

Same PWM on Multiple Pins

Attach the pins you want the same PWM signal on to the same channel.

```
const int LED_1_OUTPUT_PIN = 18;
const int LED_2_OUTPUT_PIN = 19;

const int DELAY_MS = 4;  // delay between fade increments
void setup() {
  // Sets up a channel (0-15), a PWM duty cycle frequency, and a PWM resc
  // ledcSetup(uint8_t channel, double freq, uint8_t resolution_bits);
  ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RESOLUTION);
  // ledcAttachPin(uint8_t pin, uint8_t channel);
  ledcAttachPin(LED_1_OUTPUT_PIN, PWM_CHANNEL);
  ledcAttachPin(LED_2_OUTPUT_PIN, PWM_CHANNEL);
}
```

AnalogWrite

The LEDC library was built into the earlier versions of the ESP32 core and was necessary to use PWM. The most recent versions of the ESP32 core can now use the analogWrite functionality.
analogWrite(pin, resolution)

The default resolution is 8 bits (0-255) and the default frequency is 1000Hz.

Additional Notes

-analogWrite has a fixed frequency of ~1000Hz and a 8 bit resolution
-ledcWrite has adjustable frequency range of 1Hz-40MHz and up to 16 bits resolution
-ledcWrite is better suited to DC and servo motor control because you can adjust the frequency (most DC motors work well at 5KHz+ and servos at 50 Hz)
-use the ESP32Servo library for servos as it will set these values automatically
-analogWrite is good for simple PWM as in LED dimming

Sample Motor Control Code

```
const int motorPin = 16;  // Any GPIO (even non-traditional PWM pins)
const int freq = 5000;    // 5kHz (good for DC motors)
const int channel = 0;    // PWM channel (0-15)
const int resolution = 8; // 8-bit (0-255)

void setup() {
  ledcSetup(channel, freq, resolution);
  ledcAttachPin(motorPin, channel);
}

void loop() {
  ledcWrite(channel, 128);  // 50% duty cycle
  delay(1000);
}
```

Questions and Exercises

1. How many different pins can be assigned for use with PWM in the esp32?
2. What is a channel?
3. If you set a frequency of 2000 Hz to be used with PWM in the esp32 what is the duration of each pulse?
4. Using a resolution of 8 bits means you can use what numbers to define the duty cycle?
5. What numbers can be used with a resolution of 6 bits? 10 bits? 14 bits? 16 bits?
6. If I used a frequency of 2000 Hz and a resolution of 10 bits, what would the HIGH duration be if I used the resolution value of 512?
7. Create a circuit with 5 LEDs. One is off, The second is at 25 brightness, 3rd 50% brightness, 4th 75% brightness and then finally the last is at 100%. https://wokwi.com/projects/432690302033456129
8. Create a circuit that turns LED on gradually over 5 seconds and then gradually turns it off again over 5 Seconds. https://wokwi.com/projects/432690405809987585
9. Create a circuit with 2 buttons and a LED. Use one button to gradually increase the brightness and one to gradually decrease the brightness on the LED. https://wokwi.com/projects/432690843300055041

1. The ESP32 has 16 PWM channels, and any GPIO pin (except input-only pins like GPIO34-39) can be assigned for PWM output
2. A channel in the context of ESP32 PWM (also called LEDC – LED Control) refers to a virtual PWM generator
3. Frequency = 2000 Hz Period = 1/2000 seconds = **0.0005 seconds = 500 microseconds**

   Each pulse lasts 500 microseconds.

4. An **8-bit resolution** gives you 28=25628=256 levels. So, you can define duty cycle using values from **0 to 255**.
5. With a resolution of 6 bits, you can use duty cycle values from 0 to 63. A 10-bit resolution allows values from 0 to 1023, 14-bit allows values from 0 to 16,383, and 16-bit supports values ranging from 0 to 65,535.
6. To calculate the **HIGH duration** of a PWM signal, use the following formula:

   HIGH duration=Duty Cycle×PeriodHIGH duration=Duty Cycle×Period

Frequency = **2000 Hz** → Period = 1200020001 = **0.0005 s = 500 microseconds**
Resolution = **10 bits** → Max value = **1023**
Duty value = **512**

$$\text{Duty Cycle} = \frac{512}{1023} \approx 0.5005$$

The **HIGH duration** is approximately **250.25 microseconds**.

$$\text{HIGH duration} = 0.5005 \times 500\,\mu s \approx 250.25\,\mu s$$