

PWM in ESP32

What is PWM?

PWM means **Pulse Width Modulation**. Pulse-width modulation, also known as pulse-duration modulation or pulse-length modulation, is any method of representing a signal as a rectangular wave with a varying duty cycle. PWM is useful for controlling the average power or amplitude delivered by an electrical signal.

Big word? Yes! But don't worry — let's break it into very tiny, easy pieces.

Imagine You Have a Chocolate Tap

You have a tap (like a water tap) but instead of water, **chocolate comes out!** 🍫

You want to fill your cup with chocolate. But wait — the tap can only do two things:

- **Fully ON** (chocolate flows fast!)
- **Fully OFF** (no chocolate at all)

But what if you want just **a little chocolate** in your cup? Or maybe **half the cup**? You can't open the tap halfway — only ON or OFF!

So what do you do?

You **turn ON the tap for a little time**, then **turn it OFF**, then **ON again**, then **OFF again**, very fast.

Example:

- ON for 1 second, OFF for 1 second = Half chocolate.
- ON for 3 seconds, OFF for 1 second = More chocolate!
- ON for 1 second, OFF for 3 seconds = Less chocolate!

This is exactly how PWM works in electronics.

In electronics, **you can't make electricity "half power" easily**. A wire is either getting electricity (**ON**) or not (**OFF**).

But if you want to control something — like:

- A motor to go slower or faster
- A light to be brighter or dimmer
- A sound to be softer or louder

— you need something that gives "some power" instead of all or nothing.

So what does PWM do?

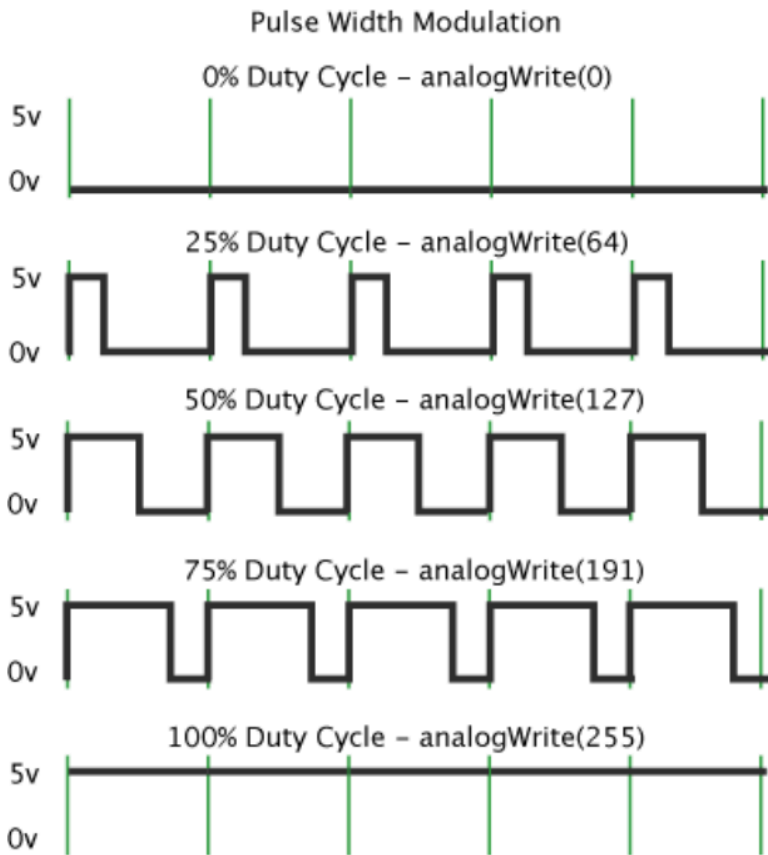
It turns electricity **ON** and **OFF super fast** — like hundreds or thousands of times every second.


The amount of time electricity stays ON vs OFF is called the **duty cycle**.


Duty Cycle Example:


- **100% Duty Cycle** = ON all the time = full power
- **50% Duty Cycle** = ON half the time, OFF half the time = half power
- **25% Duty Cycle** = ON for short time, OFF for long time = low power


Picture Time!



100% Duty Cycle:  (Always ON)

50% Duty Cycle:  (Half ON, half OFF)

25% Duty Cycle:  (Short ON, Long OFF)

0% Duty Cycle:  (Always OFF)

See? The more you keep the line ON, the more energy gets to the motor or light.

Real-Life Examples of PWM

1. **Fan Speed Controller:**
PWM tells the fan to run slowly or fast by giving small or big ON-time.
2. **LED Light Dimmer:**
PWM controls how bright your night light is by turning it ON and OFF fast.
3. **Robot Motors:**
PWM controls how fast wheels turn by sending little bursts of electricity.
4. **Sound Beeps in Toys:**
PWM changes the speaker vibrations to make different beep sounds.

Why not just give less voltage instead?

Because electronic parts like switches and transistors are **better and safer when fully ON or fully OFF** — not in the middle!

PWM lets us **pretend to give "half power"** by switching fast — saving energy and making parts last longer.

Summary (Like You Tell a Toy Robot):

PWM is like:

- **Switching ON and OFF fast**
- **How long it stays ON vs OFF = Power level**
- It can control lights, motors, sounds, fans — almost everything electronic!

ESP32 LED PWM Controller

The ESP32 has an LED PWM controller with 6 to 16 independent channels (depending on the ESP32 model) that can be configured to generate PWM signals with different properties.

There are different functions you can use to generate PWM signals and achieve the same results. You can use `analogWrite` (like in Arduino boards) or you can use LEDC functions.

`analogWrite`

The most basic function is `analogWrite` which accepts as arguments the GPIO where you want to generate the PWM signal and the duty cycle value (ranging from 0 to 255).

```
void analogWrite(uint8_t pin, int value);
```

For example:

```
void analogWrite(2, 180);
```

Set the Frequency and Resolution

You can set the resolution and frequency of the PWM signal on a selected pin by using the `analogWriteResolution` and `analogWriteFrequency` functions.

To set the resolution:

```
void analogWriteResolution(uint8_t pin, uint8_t resolution);
```

To set the frequency:

```
void analogWriteFrequency(uint8_t pin, uint32_t freq);
```

LEDC Functions

Alternatively, you can use the Arduino-ESP32 LEDC API. First, you need to set up an LEDC pin. You can use the `ledcAttach` or `ledcAttachChannel` functions.

`ledcAttach`

The `ledcAttach` function sets up an LEDC pin with a given frequency and resolution. The LEDC channel will be selected automatically.

```
bool ledcAttach(uint8_t pin, uint32_t freq, uint8_t resolution);
```

This function will return `true` if the configuration is successful. If `false` is returned, an error occurs and the LEDC channel is not configured.

`ledcAttachChannel`

If you prefer to set up the LEDC channel manually, you can use the `ledcAttachChannel` function instead.

```
bool ledcAttachChannel(uint8_t pin, uint32_t freq, uint8_t resolution, uint8_t channel);
```

This function will return `true` if the configuration is successful. If `false` is returned, an error occurs and the LEDC channel is not configured.

`ledcWrite`

Finally, after setting the LEDC pin using one of the two previous functions, you use the `ledcWrite` function to set the duty cycle of the PWM signal.

```
void ledcWrite(uint8_t pin, uint32_t duty);
```

This function will return **true** if setting the duty cycle is successful. If **false** is returned, an error occurs and the duty cycle is not set.

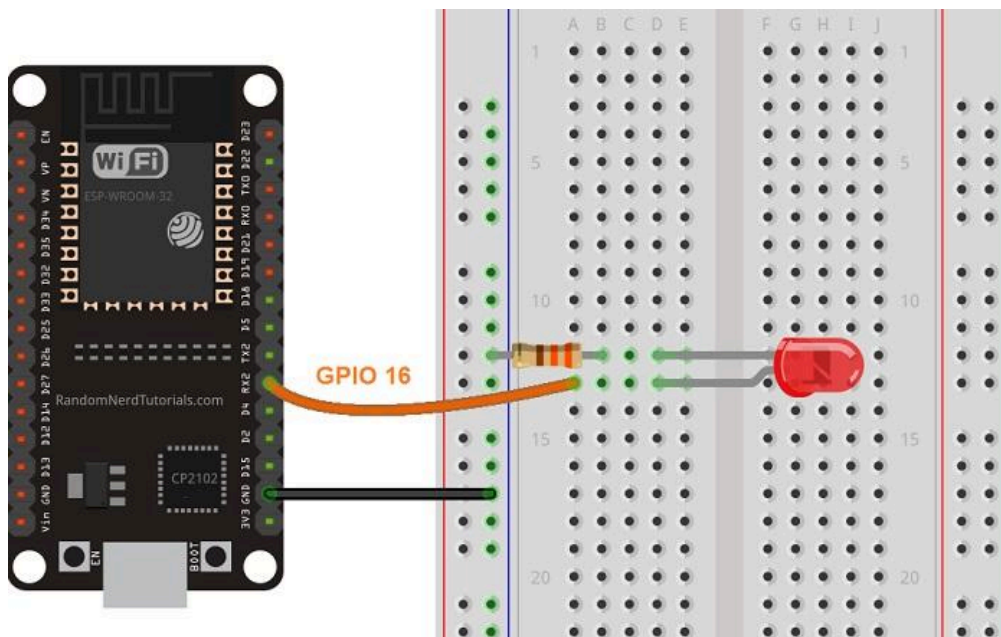
For more information and all functions of the LEDC PWM controller, [check the official documentation](#).

Dimming an LED with the ESP32

To show you how to generate PWM signals with the ESP32, we'll create two simple examples that dim the brightness of an LED (increase and decrease brightness over time). We'll provide an example using `analogWrite` and another using the LEDC functions.

Schematic

Wire an LED to your ESP32 as in the following schematic diagram. The LED should be connected to **GPIO 16**.



Objective

This program controls the brightness of an LED using **Pulse Width Modulation (PWM)** on an Arduino board. The LED gradually increases in brightness from completely OFF to fully ON, and then dims back down to OFF in a smooth fading loop.

Complete Source Code

```
#include <Arduino.h>
#define led 18

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  for(int dutycycle = 0; dutycycle <= 255; dutycycle++) {
    analogWrite(led, dutycycle);
    delay(15);
  }

  for(int dutycycle = 255; dutycycle >= 0; dutycycle--) {
    analogWrite(led, dutycycle);
    delay(15);
  }
}
```

Detailed Explanation

a) Header File Inclusion

```
#include <Arduino.h>
```

This line includes the Arduino core library, giving access to essential functions such as `pinMode()`, `digitalWrite()`, `analogWrite()`, and others.

b) Pin Definition

```
#define led 18
```

- Defines the symbol **led** as **pin number 18**.
- This improves code readability, so you can use `led` instead of typing **18** repeatedly.

c) setup() Function

```
void setup() {
  pinMode(led, OUTPUT);
}
```

- This function runs **once when the Arduino starts**.
- `pinMode(led, OUTPUT);`:
Sets **GPIO 18** as an output pin so the Arduino can control an LED connected to this pin.

d) loop() Function

```
void loop() {
  for(int dutycycle = 0; dutycycle <= 255; dutycycle++) {
    analogWrite(led, dutycycle);
    delay(15);
  }
}
```

```

for(int dutycycle = 255; dutycycle >= 0; dutycycle--) {
    analogWrite(led, dutycycle);
    delay(15);
}
}

```

This function runs **continuously in a loop**. It consists of two parts:

First for loop (Increasing Brightness)

```

for(int dutycycle = 0; dutycycle <= 255; dutycycle++) {
    analogWrite(led, dutycycle);
    delay(15);
}

```

- Starts with **dutycycle = 0** (LED fully OFF).
- Increases duty cycle to **255** (LED fully ON) step by step.
- `analogWrite(led, dutycycle);`:
Sends PWM signal to LED with the current **duty cycle**.
 - **0** = LED OFF
 - **255** = LED maximum brightness
- `delay(15);`: Waits **15 milliseconds** before increasing brightness — this slows down the brightness change so the human eye can see a smooth fade effect.

Second for loop (Decreasing Brightness)

```

for(int dutycycle = 255; dutycycle >= 0; dutycycle--) {
    analogWrite(led, dutycycle);
    delay(15);
}

```

- Starts with **dutycycle = 255** (LED fully ON).
- Decreases duty cycle to **0** (LED OFF) step by step.
- Performs the same delay and PWM control to smoothly reduce brightness.

4. PWM in Arduino (`analogWrite`)

- **PWM (Pulse Width Modulation)** is used to simulate analog output using digital signals.
- The function `analogWrite(pin, value)` sets the **duty cycle** of the PWM signal:
 - **value = 0 to 255**:
 - **0**: 0% duty cycle (always OFF)
 - **255**: 100% duty cycle (always ON)
- This method is used for dimming LEDs, controlling motor speed, generating audio tones, etc.

5. Important Notes

- The LED should be connected to **pin 18**, which must support PWM (on some boards like ESP32, GPIO 18 is PWM-capable).
- A **current-limiting resistor** (usually 220Ω to 330Ω) is recommended in series with the LED to prevent damage.
- On **standard Arduino UNO boards**, PWM is supported only on pins 3, 5, 6, 9, 10, 11. **Pin 18** is used on boards like **ESP32 or others** where GPIO 18 is PWM-capable.

6. Summary

This program creates a **breathing LED effect** by:

1. Gradually increasing LED brightness using PWM.
2. Gradually decreasing LED brightness back to zero.
3. Repeating this smoothly forever.

ESP32 LED Brightness Control using LEDC PWM

1. Objective

This program controls the brightness of an LED connected to the ESP32 development board using the **LEDC PWM module**. The LED's brightness gradually increases from completely OFF to fully ON and then dims back to OFF repeatedly, creating a smooth fading effect.

2. Complete Source Code

```
#include <Arduino.h>

#define led 18           // GPIO pin connected to the LED
#define freq 5000        // PWM frequency set to 5 KHz
#define res 8            // PWM resolution set to 8 bits (0-255 duty cycle range)
#define pwm_ch 0         // LEDC PWM channel 0

void setup() {
    ledcSetup(pwm_ch, freq, res); // Configure PWM channel 0 with specified frequency and resolution
    ledcAttachPin(led, pwm_ch);   // Attach the LED pin (GPIO 18) to PWM channel 0
}

void loop() {
    // Gradually increase LED brightness
    for(int i = 0; i <= 255; i++) {
        ledcWrite(pwm_ch, i); // Set PWM duty cycle (LED brightness)
        delay(15);             // Small delay for smooth brightness transition
    }

    // Gradually decrease LED brightness
    for(int i = 255; i >= 0; i--) {
        ledcWrite(pwm_ch, i); // Set PWM duty cycle (LED brightness)
        delay(15);             // Small delay for smooth brightness transition
    }
}
```

3. Step-by-Step Explanation

a) Header File Inclusion

```
#include <Arduino.h>
```

- Includes the core Arduino library which provides basic functions such as `setup()`, `loop()`, `delay()`, etc.

b) Macro Definitions

```
#define led 18
#define freq 5000
#define res 8
#define pwm_ch 0
```

- **led (18)**: The GPIO pin number where the LED is connected.
- **freq (5000)**: Sets the PWM frequency to **5 KHz**.
- **res (8)**: Sets the PWM resolution to **8 bits** (meaning duty cycle values range from 0 to 255).
- **pwm_ch (0)**: Using **PWM channel 0** of the ESP32's LEDC module.

c) setup() Function

```
void setup() {
    ledcSetup(pwm_ch, freq, res);    // Configure PWM channel 0
    ledcAttachPin(led, pwm_ch);      // Attach GPIO 18 to PWM channel 0
}
```

- **ledcSetup(channel, freq, resolution)**:
 - Configures **PWM Channel 0** with:
 - Frequency: 5 KHz.
 - Resolution: 8 bits (duty cycle values: 0–255).
- **ledcAttachPin(GPIO, channel)**:
 - Attaches **GPIO 18** to **PWM Channel 0** so this pin outputs the PWM signal.

d) loop() Function

```
void loop() {
    // Gradually increase LED brightness
    for(int i = 0; i <= 255; i++) {
        ledcWrite(pwm_ch, i);
        delay(15);
    }

    // Gradually decrease LED brightness
    for(int i = 255; i >= 0; i--) {
        ledcWrite(pwm_ch, i);
        delay(15);
    }
}
```

- **First for loop (Increasing brightness)**:
 - The loop variable **i** increases from 0 to 255.
 - **ledcWrite(pwm_ch, i)** sets the PWM duty cycle:
 - **0** = LED completely OFF.
 - **255** = LED fully ON.
 - **delay(15)** introduces a 15ms delay between brightness steps to ensure a smooth fade.

- **Second for loop (Decreasing brightness):**
 - The loop variable `i` decreases from 255 back to 0.
 - Same duty cycle principle applies — this dims the LED smoothly to OFF.

4. Working Principle:

- **PWM (Pulse Width Modulation)** allows control of the LED's brightness by adjusting the "ON" time vs "OFF" time (duty cycle).
- With an **8-bit resolution**, the duty cycle can be adjusted from **0 to 255**:
 - **0** = 0% ON (always off).
 - **255** = 100% ON (fully on).
 - Values in between give varying levels of brightness.
- The **LEDC PWM module** in ESP32 handles this smoothly using its hardware capabilities, without burdening the CPU.

5. Circuit Diagram Description:

ESP32 GPIO 18 -----> Resistor (220Ω) -----> LED -----> GND

- The **resistor** limits current to protect the LED from burning.
- The **LED anode (+)** connects to GPIO 18 via the resistor.
- The **LED cathode (-)** connects to GND.

6. Summary:

The LED smoothly fades in and out using ESP32's hardware PWM.

LEDC PWM module is used — better than the old `analogWrite()` function.

Frequency, resolution, and channel are fully customizable for other purposes like motor or buzzer control.

