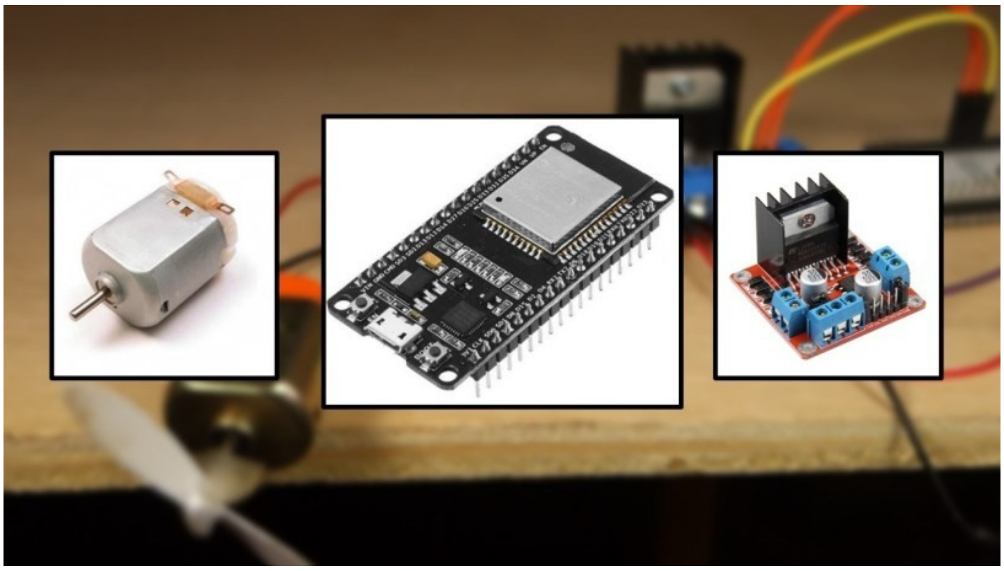
ESP32 WIKI II

ESP32 with DC Motor and L298N Motor Driver – Control Speed and Direction

https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/

This tutorial shows how to control the direction and speed of a DC motor using an ESP32 and the L298N Motor Driver. First, we'll take a quick look on how the L298N motor driver works. Then, we'll show you an example on how to control the speed and direction of a DC motor using the ESP32 with Arduino IDE and the L298N motor driver.



Note: there are many ways to control a DC motor. We'll be using the L298N motor driver. This tutorial is also compatible with similar motor driver modules.

To better understand with this tutorial, you may want to take a look at the following posts:

- Getting Started with ESP32 Dev Module
- Installing the ESP32 Board in Arduino IDE (Windows instructions)
- Installing the ESP32 Board in Arduino IDE (Mac and Linux instructions)
- ESP32 Web Server – Arduino IDE

Parts Required

To complete this tutorial you need the following parts:

ESP32 DOIT DEVKIT V1 Board – read ESP32 Development Boards Review and Comparison

- DC motor
- L298N motor driver
- Power source: 4x 1.5 AA batteries or Bench power supply
- 2x 100nF ceramic capacitors (optional)
- 1x SPDT slide switch (optional)
- Jumper wires

You can use the preceding links or go directly to MakerAdvisor.com/tools to find all the parts for your projects at the best price!

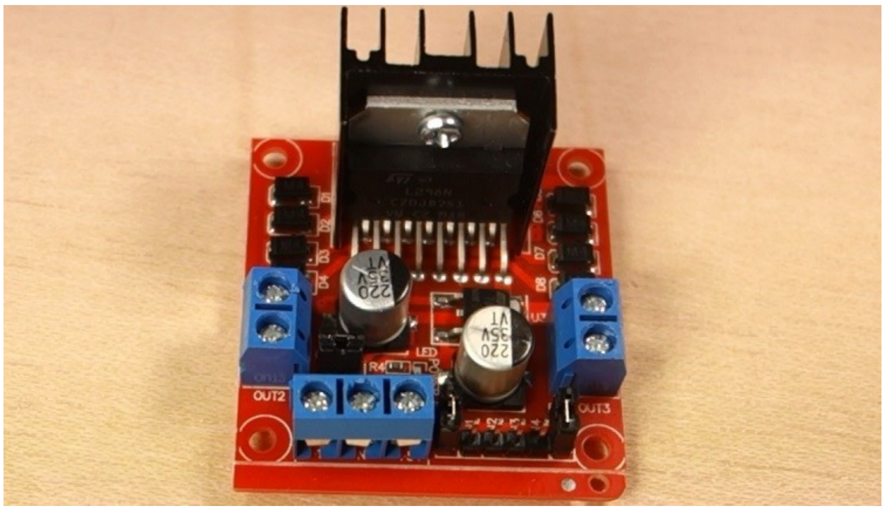
MAKER ADVISOR

Introducing the L298N Motor Driver

There are many ways to control a DC motor. The method we'll use here is suitable for most hobbyist motors, that require 6V or 12V to operate.

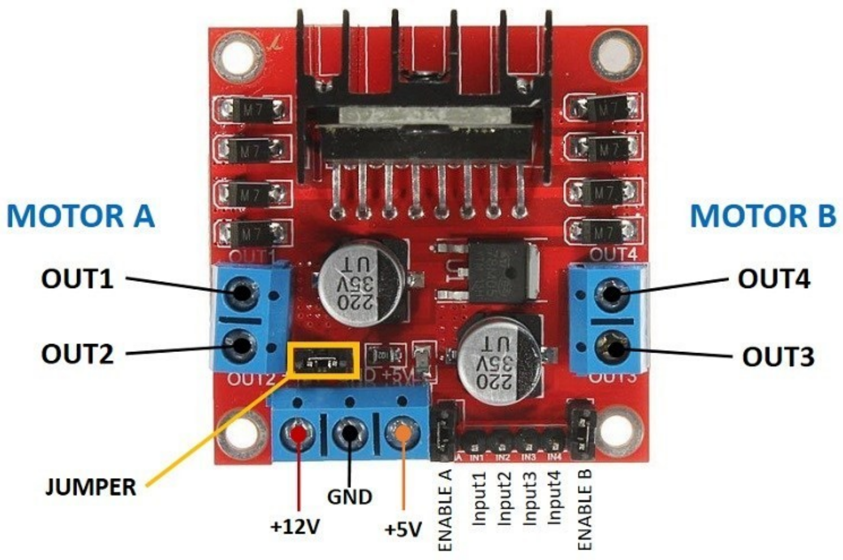
We're going to use the L298N motor driver that can handle up to 3A at 35V. Additionally, it allows us to drive two DC motors simultaneously, which is perfect to build a robot.

The L298N motor driver is shown in the following figure:



L298N Motor Driver pinout

Let’s take a look at the L298N motor driver pinout and see how it works.



The motor driver has a two terminal block in each side for each motor. OUT1 and OUT2 at the left and OUT3 and OUT4 at the right.

- **OUT1**: DC motor A + terminal
- **OUT2**: DC motor A – terminal
- **OUT3**: DC motor B + terminal
- **OUT4**: DC motor B – terminal

At the bottom you have a three terminal block with +12V, GND, and +5V. The +12V terminal block is used to power up the motors. The +5V terminal is used to power up the L298N chip. However, if the jumper is in place, the chip is powered using the motor’s power supply and you don’t need to supply 5V through the +5V terminal.

Note: if you supply more than 12V, you need to remove the jumper and supply 5V to the +5V terminal.

It’s important to note that despite the +12V terminal name, with the setup we’ll use here (with the jumper in place) you can supply any voltage between 6V and 12V. In this tutorial will be using 4 AA 1.5V batteries that combined output approximately 6V, but you can use any other suitable power supply. For example, you can use a bench power supply to test this tutorial.

In summary:

- **+12V**: The +12V terminal is where you should connect your power supply
- **GND**: power supply GND
- **+5V**: provide 5V if jumper is removed. Acts as a 5V output if jumper is in place
- **Jumper**: jumper in place – uses the motors power supply to power up the chip. Jumper removed: you need to provide 5V to the +5V terminal. If you supply more than 12V, you should remove the jumper

At the bottom right you have four input pins and two enable terminals. The input pins are used to control the direction of your DC motors, and the enable pins are used to control the speed of each motor.

- **IN1:** Input 1 for Motor A
- **IN2**: Input 2 for Motor A
- **IN3**: Input 1 for Motor B
- **IN4**: Input 2 for Motor B
- **EN1**: Enable pin for Motor A
- **EN2**: Enable pin for Motor B

There are jumper caps on the enable pins by default. You need to remove those jumper caps to control the speed of your motors.

Control DC motors with the L298N

Now that you're familiar with the L298N Motor Driver, let's see how to use it to control your DC motors.

Enable pins

The enable pins are like an ON and OFF switch for your motors. For example:

- If you send a **HIGH signal** to the enable 1 pin, motor A is ready to be controlled and at the maximum speed;
- If you send a **LOW signal** to the enable 1 pin, motor A turns off;
- If you send a **PWM signal**, you can control the speed of the motor. The motor speed is proportional to the duty cycle. However, note that for small duty cycles, the motors might not spin, and make a continuous buzz sound.

SIGNAL ON THE ENABLE PIN	MOTOR STATE
HIGH	Motor enabled
LOW	Motor not enabled
PWM	Motor enabled: speed proportional to duty cycle

Input pins

The input pins control the direction the motors are spinning. Input 1 and input 2 control motor A, and input 3 and 4 control motor B.

- If you apply LOW to input1 and HIGH to input 2, the motor will spin forward;
- If you apply power the other way around: HIGH to input 1 and LOW to input 2, the motor will rotate backwards. Motor B can be controlled using the same method but applying HIGH or LOW to input 3 and input 4.

Controlling 2 DC Motors – ideal to build a robot

If you want to build a robot car using 2 DC motors, these should be rotating in specific directions to make the robot go left, right, forward or backwards.

For example, if you want your robot to move forward, both motors should be rotating forward. To make it go backwards, both should be rotating backwards.

To turn the robot in one direction, you need to spin the opposite motor faster. For example, to make the robot turn right, enable the motor at the left, and disable the motor at the right. The following table shows the input pins' state combinations for the robot directions.

DIRECTION	**INPUT 1**	**INPUT 2**	**INPUT 3**	**INPUT 4**
Forward	0	1	0	1
Backward	1	0	1	0
Right	0	1	0	0
Left	0	0	0	1
Stop	0	0	0	0

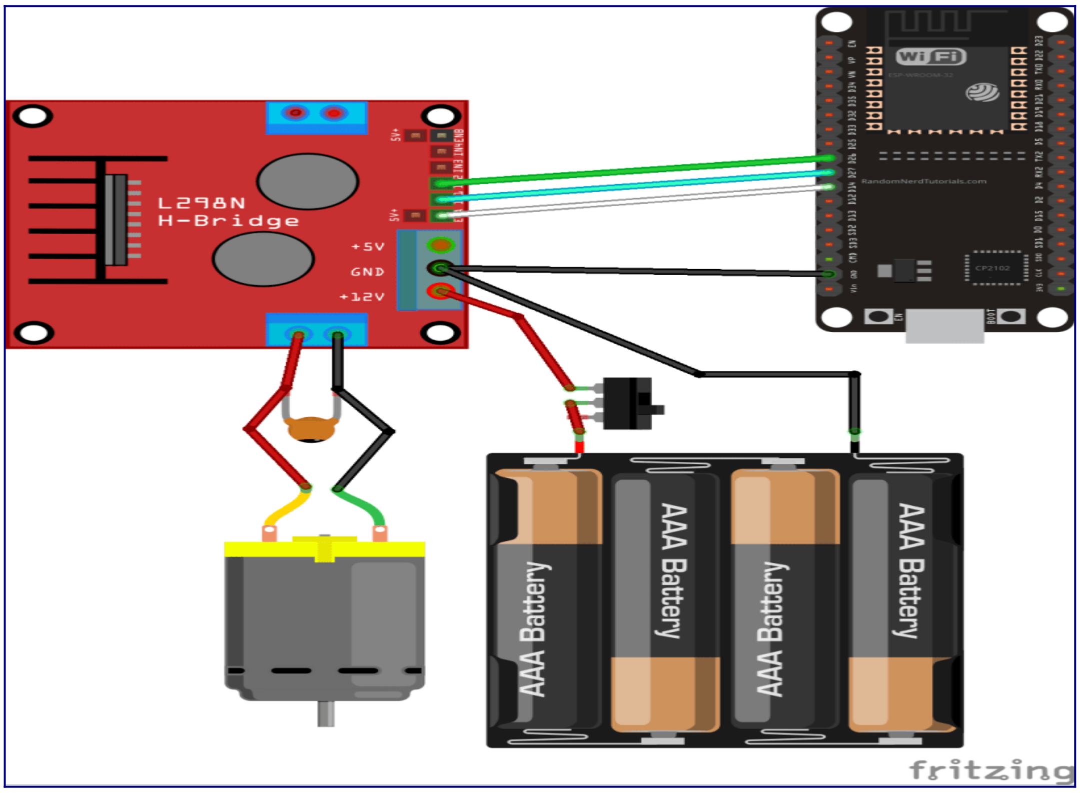
Recommended reading: Build Robot Car Chassis Kit for ESP32, ESP8266, Arduino, etc…

Control DC Motor with ESP32 – Speed and Direction

Now that you know how to control a DC motor with the L298N motor driver, let's build a simple example to control the speed and direction of one DC motor.

Schematic

The motor we'll control is connected to the motor A output pins, so we need to wire the ENABLEA, INPUT1 and INPUT2 pins of the motor driver to the ESP32. Follow the next schematic diagram to wire the DC motor and the L298N motor driver to the ESP32.



The DC motor requires a big jump in current to move, so the motors should be powered using an external power source from the ESP32. As an example, we're using 4AA batteries, but you can use any other suitable power supply. In this configuration, you can use a power supply with 6V to 12V.

The switch between the battery holder and the motor driver is optional, but it is very handy to cut and apply power. This way you don't need to constantly connect and then disconnect the wires to save power.

We recommend soldering a 0.1uF ceramic capacitor to the positive and negative terminals of the DC motor, as shown in the diagram to help smooth out any voltage spikes. (Note: the motors also work without the capacitor.)

Preparing the Arduino IDE

There's an add-on for the Arduino IDE allows you to program the ESP32 using the Arduino IDE and its programming language. Follow one of the next tutorials to prepare your Arduino IDE to work with the ESP32, if you haven't already.

- **Windows** instructions – ESP32 Board in Arduino IDE
- **Mac and Linux** instructions – ESP32 Board in Arduino IDE

After making sure you have the ESP32 add-on installed, you can continue this tutorial.

Uploading code

The following code controls the speed and direction of the DC motor. This code is not useful in the real world, this is just a simple example to better understand how to control the speed and direction of a DC motor with the ESP32.

```
/*********
  Rui Santos
  Complete project details at http://randomnerdtutorials.com
*********/

// Motor A
int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 14;

// Setting PWM properties
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8;
int dutyCycle = 200;

void setup() {
  // sets the pins as outputs:
  pinMode(motor1Pin1, OUTPUT);
  pinMode(motor1Pin2, OUTPUT);
  pinMode(enable1Pin, OUTPUT);

  // configure LED PWM functionalitites
  ledcSetup(pwmChannel, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(enable1Pin, pwmChannel);

  Serial.begin(115200);

  // testing
  Serial.print("Testing DC Motor...");
}

void loop() {
  // Move the DC motor forward at maximum speed
  Serial.println("Moving Forward");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, HIGH);
  delay(2000);

  // Stop the DC motor
  Serial.println("Motor stopped");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  delay(1000);

  // Move DC motor backwards at maximum speed
  Serial.println("Moving Backwards");
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  delay(2000);

  // Stop the DC motor
  Serial.println("Motor stopped");
  digitalWrite(motor1Pin1, LOW);
  digitalWrite(motor1Pin2, LOW);
  delay(1000);

  // Move DC motor forward with increasing speed
  digitalWrite(motor1Pin1, HIGH);
  digitalWrite(motor1Pin2, LOW);
  while (dutyCycle <= 255){
    ledcWrite(pwmChannel, dutyCycle);
    Serial.print("Forward with duty cycle: ");
    Serial.println(dutyCycle);
    dutyCycle = dutyCycle + 5;
    delay(500);
  }
  dutyCycle = 200;
}
```

Upload the code to your ESP32. Make sure you have the right board and COM port selected. Let’s take a look on how the code works.

Declaring motor pins

First, you define the GPIOs the motor pins are connected to. In this case, Input 1 for motor A is connected to GPIO 27, the Input 2 to GPIO 26, and the Enable pin to GPIO 14.

```
int motor1Pin1 = 27;
int motor1Pin2 = 26;
int enable1Pin = 14;
```

Setting the PWM properties to control the speed

As we've seen previously, you can control the DC motor speed by applying a PWM signal to the enable pin of the L298N motor driver. The speed will be proportional to the duty cycle. To use PWM with the ESP32, you need to set the PWM signal properties first.

```
const int freq = 30000;
const int pwmChannel = 0;
const int resolution = 8;
int dutyCycle = 200;
```

In this case, we're generating a signal of 30000 Hz on channel 0 with a 8-bit resolution. We start with a duty cycle of 200 (you can set a duty cycle value from 0 to 255).

For the frequency we're using, when you apply duty cycles smaller than 200, the motor won't move and will make a weird buzz sound. So, that's why we set a duty cycle of 200 at the start.

Note: the PWM properties we're defining here are just an example. The motor works fine with other frequencies.

setup()

In the setup(), you start by setting the motor pins as outputs.

```
pinMode(motor1Pin1, OUTPUT);
pinMode(motor1Pin2, OUTPUT);
pinMode(enable1Pin, OUTPUT);
```

You need to configure a PWM signal with the properties you've defined earlier by using the ledcSetup() function that accepts as arguments, the pwmChannel, the frequency, and the resolution, as follows:

```
ledcSetup(pwmChannel, freq, resolution);
```

Next, you need to choose the GPIO you'll get the signal from. For that use the ledcAttachPin() function that accepts as arguments the GPIO where you want to get the signal, and the channel that is generating the signal. In this example, we'll get the signal in the enable1Pin GPIO, that corresponds to GPIO 14. The channel that generates the signal is the pwmChannel, that corresponds to channel 0.

```
ledcAttachPin(enable1Pin, pwmChannel);
```

Moving the DC motor forward

In the loop() is where the motor moves. The code is well comment on what each part of the code does. To move the motor forward, you set input 1 pin to LOW and input 2 pint to HIGH. In this example, the motor speeds forward for 2 seconds (2000 milliseconds).

```
// Move the DC motor forward at maximum speed
Serial.println("Moving Forward");
digitalWrite(motor1Pin1, LOW);
digitalWrite(motor1Pin2, HIGH);
delay(2000);
```

Moving the DC motor backwards

To move the DC motor backwards you apply power to the motor input pins the other way around. HIGH to input 1 and LOW to input 2.

```
// Move DC motor backwards at maximum speed
Serial.println("Moving Backwards");
digitalWrite(motor1Pin1, HIGH);
digitalWrite(motor1Pin2, LOW);
delay(2000);
```

Stop the DC motor

To make the DC motor stop, you can either set the enable pin to LOW, or set both input 1 and input 2 pins to LOW. In this example we're setting both input pins to LOW.

```
// Stop the DC motor
Serial.println("Motor stopped");
digitalWrite(motor1Pin1, LOW);
digitalWrite(motor1Pin2, LOW);
delay(1000);
```

Controlling the DC motor speed

To control the DC motor speed, we need to change the PWM signal duty cycle. For that you use the ledcWrite() function that accepts as arguments the PWM channel that is generating the signal (not the output GPIO) and the duty cycle, as follows.

```
ledcWrite(pwmChannel, dutyCycle);
```

In our example, we have a while loop that increases the duty cycle by 5 in every loop.

```
// Move DC motor forward with increasing speed
digitalWrite(motor1Pin1, HIGH);
digitalWrite(motor1Pin2, LOW);
while (dutyCycle <= 255){
  ledcWrite(pwmChannel, dutyCycle);
  Serial.print("Forward with duty cycle: ");
  Serial.println(dutyCycle);
  dutyCycle = dutyCycle + 5;
  delay(500);
}
```

When the while condition is no longer true, we set the duty cycle to 200 again.

```
dutyCycle = 200;
```

Watch the Video Demonstration

Watch the next video to see the project in action:

https://youtu.be/amknFvyjW78

Wrapping Up

In this tutorial we’ve shown you how to control the direction and speed of a DC motor using an ESP32 and the L298N motor driver. In summary:

- To control the direction the DC motor is spinning you use the input 1 and input 2 pins;
- Apply LOW to input 1 and HIGH to input 2 to spin the motor forward. Apply power the other way around to make it spin backwards;
- To control the speed of the DC motor, you use a PWM signal on the enable pin. The speed of the DC motor is proportional to the duty cycle.

We hope you’ve found this tutorial useful.