ESP32 PWM Example

This **ESP32 PWM example** explains how to use the Pulse Width Modulation (PWM) with the ESP32-WROOM32 DevKit.

**ledcWrite(pinChannel, dutyCycle);**
ESP32 can generate PWM on all IO pins. In the ESP32 analogWrite will not work, is different than the Arduino Uno. ESP32 uses 8, 10, 12, 15-bit resolution for PWM generation PWM value. Arduino Uses 8-Bit Resolution i.e.PWM range is 0-254.

So, in order to use PWM, we can call the ledcWrite(pinChannel, dutyCycle);

We are using on board LED, which is no GPIO2



ledcWrite(pinChannel, dutyCycle);, Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightnesses or drive a motor at various speeds. After a call to ledcWrite(pinChannel, dutyCycle);, the pin will generate a steady square wave of the specified duty cycle until the next call to ledcWrite(pinChannel, dutyCycle); (or a call to digitalRead() or digitalWrite() on the same pin). The frequency of the PWM signal on most pins is approximately 1 KHz.

You do not need to call pinMode() to set the pin as an output before calling ledcWrite(pinChannel, dutyCycle);.

**Before we start actual programming lets have a look at What is PWM?**

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (3.3 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of “on time” is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 3.3V controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Arduino’s PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to ledcWrite(pinChannel, dutyCycle); is on a scale of 0 – 1023, such that ledcWrite(pinChannel, 1023); requests a 100% duty cycle (always on), and analogWrite(512) is a 50% duty cycle (on half the time) for example.



**LED Fading Program using ESP8266 PWM Function**

|  |  |
| --- | --- |
| 12345678910111213141516171819202122232425262728293031323334353637383940414243444546474849505152535455565758596061626364656667686970717273 | /\* \* Copyright (c) 2018, circuits4you.com \* All rights reserved./\* Generates PWM on Internal LED Pin GPIO 2 of ESP32\*/ #define LED 2 //On Board LED int brightness = 0;    // how bright the LED isint fadeAmount = 5;    // how many points to fade the LED by // setting PWM propertiesconst int freq = 5000;const int ledChannel = 0;const int resolution = 10; //Resolution 8, 10, 12, 15 //=======================================================================//                    Power on setup//=======================================================================void setup() {  Serial.begin(115200);  pinMode(LED,OUTPUT);    // configure LED PWM functionalitites  ledcSetup(ledChannel, freq, resolution);    // attach the channel to the GPIO2 to be controlled  ledcAttachPin(LED, ledChannel);} //=======================================================================//                    Main Program Loop//=======================================================================void loop() {  //PWM Value varries from 0 to 1023    Serial.println("10 % PWM");  ledcWrite(ledChannel, 102);  delay(2000);   Serial.println("20 % PWM");  ledcWrite(ledChannel,205);  delay(2000);   Serial.println("40 % PWM");  ledcWrite(ledChannel,410);  delay(2000);   Serial.println("70 % PWM");  ledcWrite(ledChannel,714);  delay(2000);   Serial.println("100 % PWM");  ledcWrite(ledChannel,1024);  delay(2000);   //Continuous Fading  Serial.println("Fadding Started");  while(1)  {    // set the brightness of pin 2:    ledcWrite(ledChannel, brightness);      // change the brightness for next time through the loop:    brightness = brightness + fadeAmount;      // reverse the direction of the fading at the ends of the fade:    if (brightness <= 0 || brightness >= 1023) {      fadeAmount = -fadeAmount;    }    // wait for 30 milliseconds to see the dimming effect    delay(10);  }}//======================================================================= |

Upload program in your ESP and open serial monitor.