PWM ESP32

PWM ESP32 EN

Une brève note sur ESP32 PWM

Le SoC ESP32 est entièrement chargé de périphériques très utiles et PWM en fait partie. Oui. Il existe un bloc matériel dédié pour PWM dans le silicium de l'ESP32. La modulation de largeur d'impulsion ou PWM en bref est une technique établie et largement utilisée pour la fourniture de puissance.

Vous pouvez utiliser le PWM d'ESP32 pour piloter des LED, des moteurs (moteurs à courant continu normaux ainsi que des moteurs sans balais) et des lumières intelligentes. Le contrôleur PWM dans ESP32 se compose de deux sous-modules principaux : Contrôle LED ou périphérique LEDC et modulateur de largeur d'impulsion de contrôle moteur ou périphérique MCPWM.

Même si nous limiterons notre démonstration de PWM dans ESP32 à la décoloration d'une LED, il est bon de connaître le bloc Motor Control PWM (MCPWM) dans ESP32, avec des modules de capture d'entrée.

Si vous avez déjà travaillé avec des moteurs CC sans balais (BLDC), vous vous rendrez compte de l'importance de détecter la position du rotor (à l'aide de capteurs à effet Hall) pour un contrôle précis de la vitesse. Contrôleur PWM LED ESP32 (LEDC)

Le périphérique LEDC de l'ESP32 se compose de 16 canaux PWM capables de générer des formes d'onde indépendantes, principalement pour le contrôle des LED RVB, mais peut également être utilisé à d'autres fins.

Il y a quelques points intéressants sur le contrôleur LED PWM dans ESP32 dont vous devez être conscient.

- 16 canaux PWM indépendants, divisés en groupe de deux avec 8 canaux par groupe.
- Résolution programmable entre 1 bit et 16 bits.
- La fréquence de l'onde PWM dépend de la résolution du PWM.
- Augmente/diminue automatiquement le rapport cyclique sans intervention du processeur.

Configurer les canaux PWM d'ESP32

Vous souvenez-vous de la fonction 'analogWrite()' dans la programmation Arduino ? C'est la fonction responsable de la génération de PWM dans Arduino UNO (et d'autres cartes “Arduino”).

Étant donné que presque tout dans LED PWM d'ESP32 est configurable par l'utilisateur (canal, résolution et fréquence), au lieu d'utiliser la fonction 'analogWrite ()', nous utiliserons un ensemble de fonctions différent (et dédié) pour configurer PWM dans ESP32 .

Voici une liste de toutes les API LEDC exposées par le pilote. Ces fonctions sont écrites pour le port Arduino IDE d'ESP32.

- ledcSetup(canal, fréquence, resolution_bits);
- ledcAttachPin(broche, canal);
- ledcWrite(canal, rapport cyclique);
- ledcRead(canal);
- ledcWriteTone(canal, fréquence);
- ledcWriteNote(canal, note, octave);
- ledcReadFreq(canal);
- ledcDetachPin(broche);

Parmi les 8 fonctions, nous nous concentrerons sur les trois premières, car elles sont plus utiles (et le minimum requis) pour générer du PWM.

Quelques points importants à retenir lors de la configuration du canal PWM dans ESP32 :

- Comme il y a 16 canaux PWM, l'argument 'canal' prend n'importe quelle valeur entre 0 et 15.
- Vient ensuite la fréquence du signal PWM. Vous pouvez définir la fréquence selon vos besoins, comme 1 KHz, 5 KHz, 8 KHz et 10 KHz.
- La résolution du PWM est également configurable et ESP32 PWM peut être programmé n'importe où entre 1 bit et 16 bits de résolution.
- La fréquence et la résolution PWM sont inversement proportionnelles et dépendent de la source d'horloge. Soyez donc prudent lorsque vous sélectionnez les valeurs de fréquence et de résolution.
- Enfin, attribuez une broche GPIO pour la sortie PWM. Vous pouvez attribuer n'importe quelle broche GPIO, mais soyez prudent lors de l'attribution (n'utilisez pas de broches GPIO déjà utilisées comme UART , SPI, etc.).

Le tableau suivant montre quelques fréquences et résolutions PWM couramment utilisées.

Source d'horloge pour LEDC	Fréquence PWM LEDC	Résolution PWM
80 MHz APB_CLK	1 KHz	16 bits
80 MHz APB_CLK	5 KHz	14 bits
80 MHz APB_CLK	10 KHz	13 bits
8MHz RTC8M_CLK	1 KHz	13 bits
8MHz RTC8M_CLK	8 KHz	10 bits
1MHz REF_TICK	1 KHz	10 bits

Fading LED utilisant PWM dans ESP32

Avec toutes les informations nécessaires sur PWM dans ESP32, nous pouvons maintenant procéder à la mise en œuvre de notre premier projet de décoloration d'une LED à l'aide de ESP32 PWM. C'est un projet très simple où la luminosité d'une LED connectée à une broche GPIO d'ESP32 augmentera et diminuera progressivement à plusieurs reprises. [Projets ESP32 pour débutants]

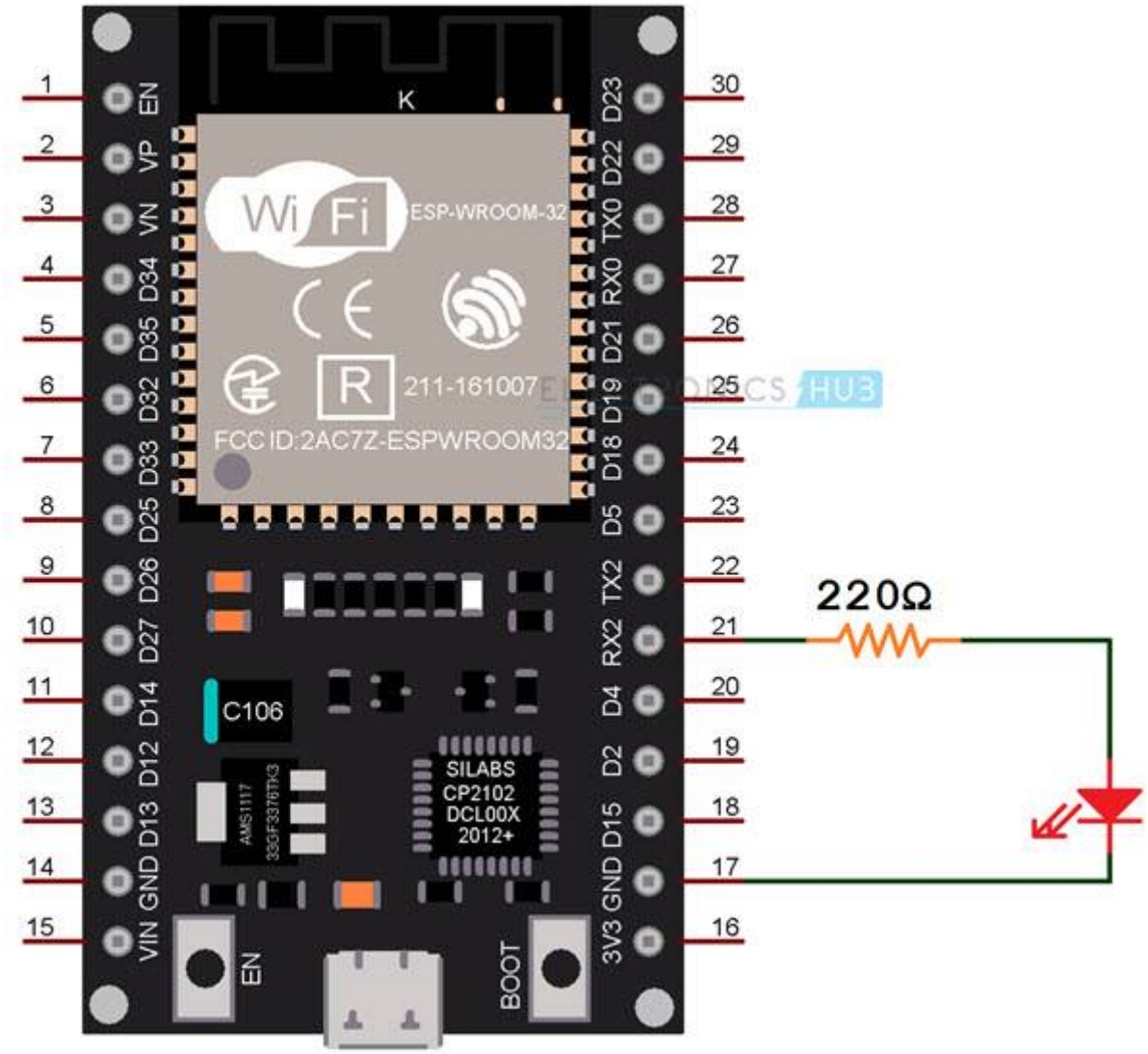
Ce projet consiste davantage à comprendre les fonctions LEDC : ledcSetup, ledcAttachPin et ledcWrite et comment générer du PWM dans ESP32 que la LED qui s'estompe elle-même. Composants requis

- Carte de développement ESP32 DevKit
- 3 LED de 5 mm
- Résistance 220Ω
- 3 potentiomètres 5KΩ
- Planche à pain

- Fils de connexion
- Câble micro-USB

Schéma

L'image suivante montre la connexion pour la décoloration d'une LED à l'aide du contrôleur ESP32 PWM.



Code

Vous pouvez utiliser n'importe quelle broche GPIO pour émettre le signal PWM. Donc, j'utilise GPIO 16, qui est également la broche RX UART2. Ensuite, nous devons configurer le canal LEDC en utilisant la fonction 'ledcSetup'. Le premier argument est le canal. Toute valeur comprise entre 0 et 15 peut être donnée comme canal.

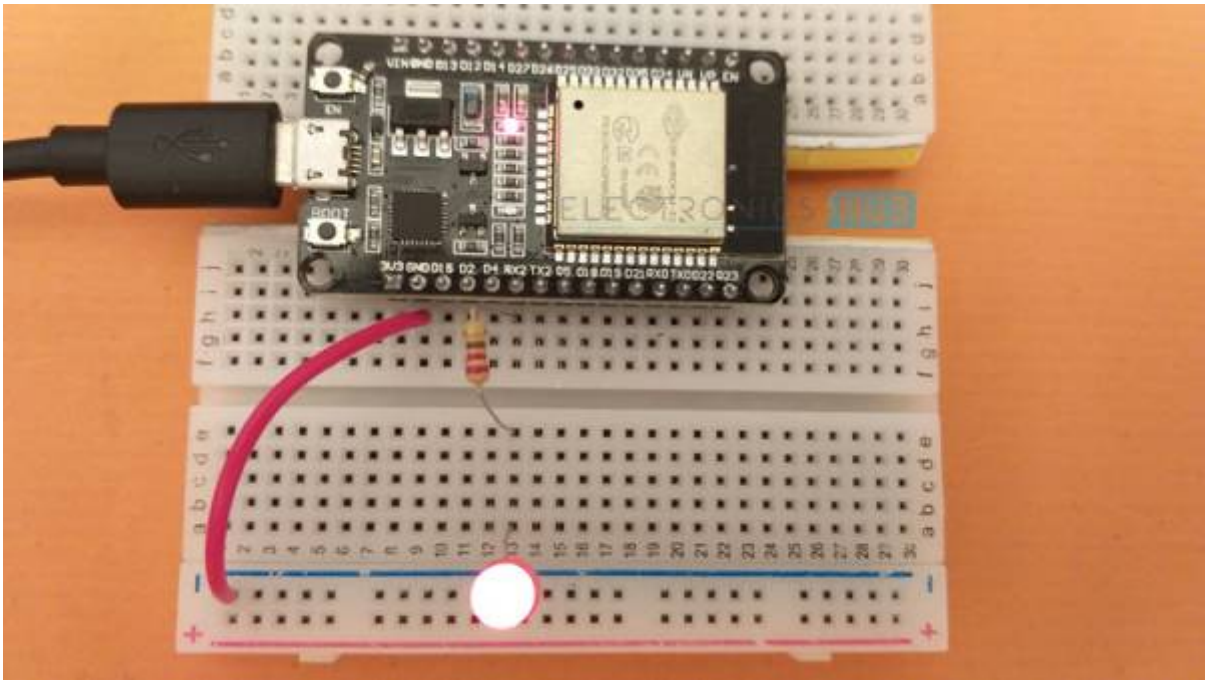
L'argument suivant est la fréquence. Vous pouvez fournir n'importe quelle fréquence, mais pour plus de commodité, je définirai la fréquence sur 5 KHz. De plus, vous devez définir la résolution du PWM. Cette valeur doit être un nombre compris entre 1 et 16. Je suis allé avec une résolution de 10 bits.

Pour le reste des paramètres, reportez-vous au code suivant, où j'ai commenté les lignes importantes.

pwmesp32led.ino

```
const int LEDPin = 16;  /* GPIO16 */

int dutyCycle;
/* Setting PWM Properties */
const int PWMFreq = 5000; /* 5 KHz */
const int PWMChannel = 0;
const int PWMResolution = 10;
const int MAX_DUTY_CYCLE = (int)(pow(2, PWMResolution) - 1);
void setup()
{
  ledcSetup(PWMChannel, PWMFreq, PWMResolution);
  /* Attach the LED PWM Channel to the GPIO Pin */
  ledcAttachPin(LEDPin, PWMChannel);
}
void loop()
{
  /* Increasing the LED brightness with PWM */
  for(dutyCycle = 0; dutyCycle <= MAX_DUTY_CYCLE; dutyCycle++)
  {
    ledcWrite(PWMChannel, dutyCycle);
    delay(3);
    //delayMicroseconds(100);
  }
     /* Decreasing the LED brightness with PWM */
  for(dutyCycle = MAX_DUTY_CYCLE; dutyCycle >= 0; dutyCycle--)
  {
    ledcWrite(PWMChannel, dutyCycle);
    delay(3);
    //delayMicroseconds(100);
  }
}
```



Vous pouvez connecter plusieurs broches GPIO au même canal LEDC PWM. Si vous le faites, toutes les broches GPIO partageront les propriétés du canal (résolution et fréquence).

ESP32 PWM avec ADC

L'une des caractéristiques importantes de PWM dans ESP32 est que les 16 canaux peuvent être configurés indépendamment, c'est-à-dire que chaque canal peut avoir sa propre résolution et sa propre fréquence. Pour le démontrer, utilisons le périphérique ADC et ajustons indépendamment le rapport cyclique de trois canaux PWM LEDC différents en tournant un potentiomètre.

Trois potentiomètres 5KΩ sont connectés à trois broches d'entrée ADC d'ESP32. Sur la base de la sortie de l'ADC, nous définirons le rapport cyclique de trois canaux PWM, qui sont configurés avec des paramètres différents.

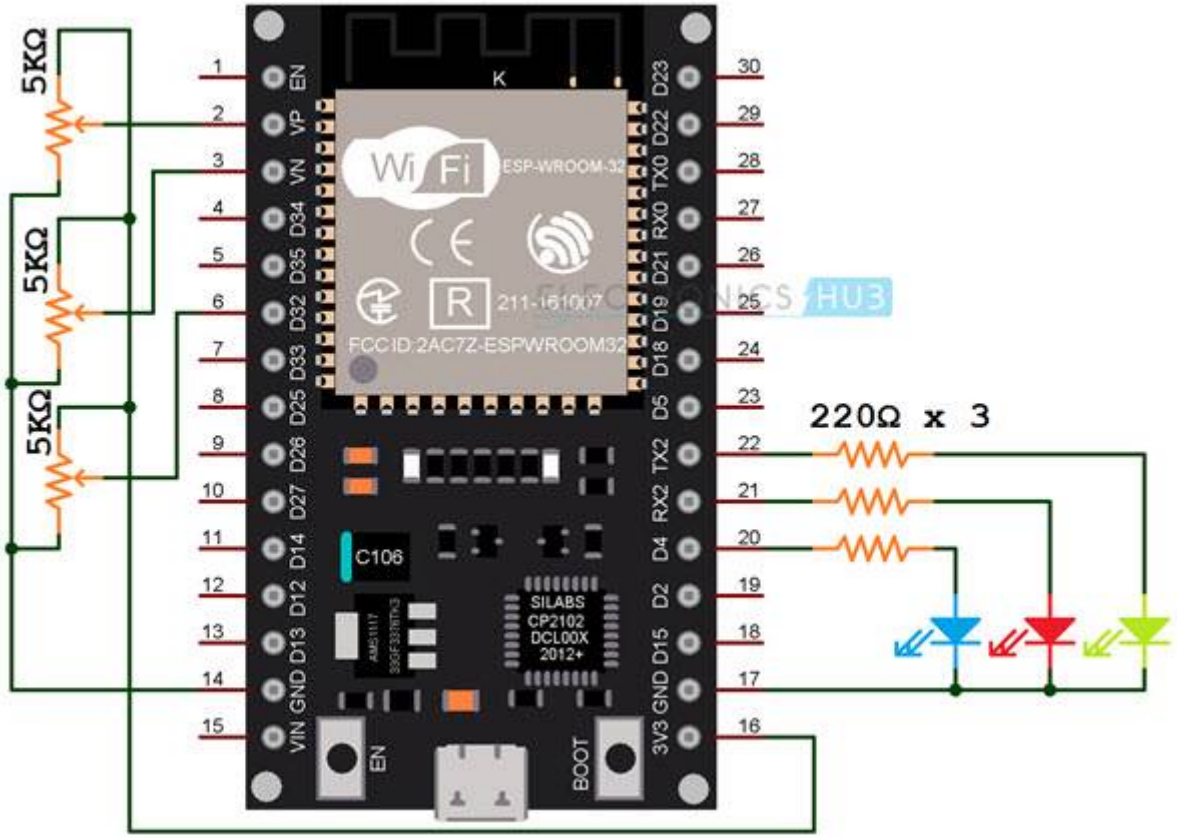
Pour faciliter la compréhension, j'ai connecté trois LED : ROUGE, VERT et BLEU à trois broches GPIO. Ces trois broches GPIO sont attachées à trois canaux LEDC PWM différents et chaque canal est initialisé avec sa propre fréquence et sa propre résolution.

LED	GPIO Pin	PWM Channel	PWM Frequency	PWM Resolution
RED	GPIO 16	0	5000 (5 KHz)	12
GREEN	GPIO 17	2	8000 (8 KHz)	13
BLUE	GPIO 4	4	10000 (10 KHz)	14

Un autre point important à retenir est que la résolution de l'ADC de l'ESP32 est de 12 bits. Nous devons donc mapper cela avec soin sur la résolution PWM, pour obtenir la gamme complète de contrôle.

Schéma

L'image suivante montre les connexions pour régler le rapport cyclique des canaux PWM à l'aide de l'ADC (potentiomètres).



pwmledPotentiometre.ino

```
const int redLEDPin = 16;  /* GPIO16 */
const int greenLEDPin = 17;  /* GPIO17 */
const int blueLEDPin = 4;  /* GPIO4 */
uint16_t redDutyCycle;
uint16_t greenDutyCycle;
uint16_t blueDutyCycle;

const int redPWMFreq = 5000; /* 5 KHz */
const int redPWMChannel = 0;
const int redPWMResolution = 12;
const int RED_MAX_DUTY_CYCLE = (int)(pow(2, redPWMResolution) - 1);

const int greenPWMFreq = 8000; /* 8 KHz */
const int greenPWMChannel = 2;
const int greenPWMResolution = 13;
const int GREEN_MAX_DUTY_CYCLE = (int)(pow(2, greenPWMResolution) - 1);

const int bluePWMFreq = 10000; /* 10 KHz */
const int bluePWMChannel = 4;
```

```
const int bluePWMResolution = 14;
const int BLUE_MAX_DUTY_CYCLE = (int)(pow(2, bluePWMResolution) - 1);

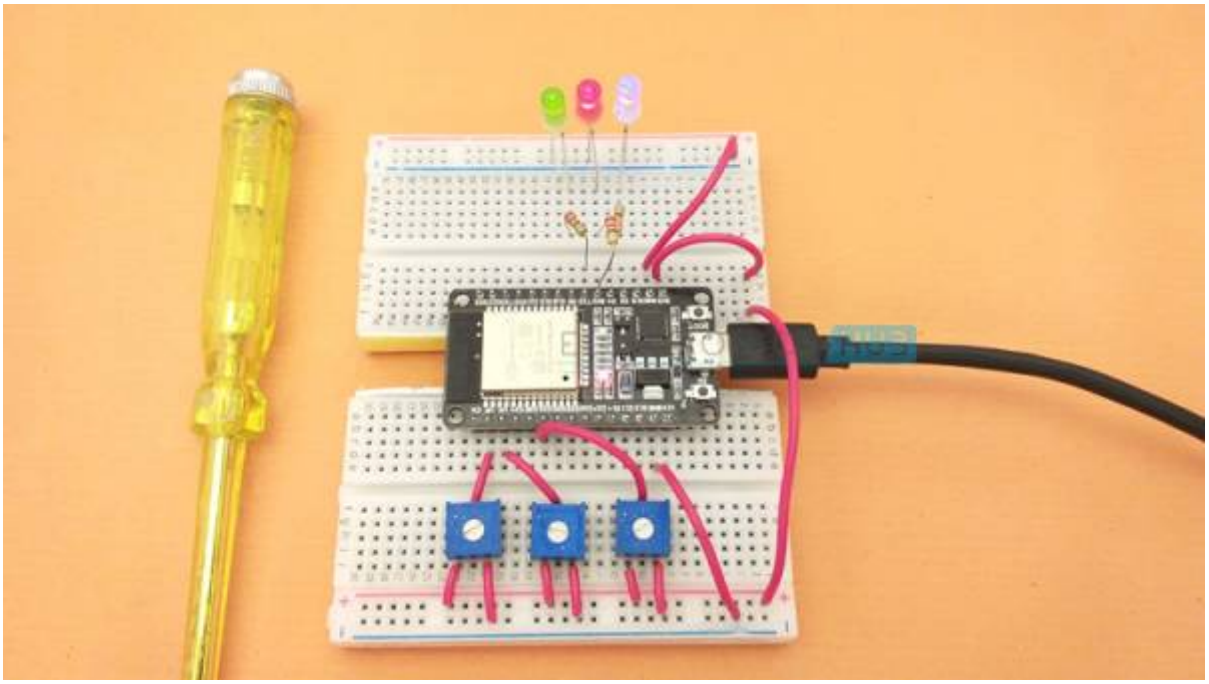
const int ADC_RESOLUTION = 4095; /* 12-bit */


void setup()
{
  /* Initialize Serial Port */
  Serial.begin(115200);
  /* Initialize PWM Channels with Frequency and Resolution */
  ledcSetup(redPWMChannel, redPWMFreq, redPWMResolution);
  ledcSetup(greenPWMChannel, greenPWMFreq, greenPWMResolution);
  ledcSetup(bluePWMChannel, bluePWMFreq, bluePWMResolution);
  /* Attach the LED PWM Channel to the GPIO Pin */
  ledcAttachPin(redLEDPin, redPWMChannel);
  ledcAttachPin(greenLEDPin, greenPWMChannel);
  ledcAttachPin(blueLEDPin, bluePWMChannel);

}
void loop()
{
  /* Read Analog Input from three ADC Inputs */
  redDutyCycle = analogRead(A0);
  greenDutyCycle = analogRead(A3);
  blueDutyCycle = analogRead(A4);
  /* Map ADC Output to maximum possible dutycycle */
  //redDutyCycle = map(redDutyCycle, 0, ADC_RESOLUTION, 0,
RED_MAX_DUTY_CYCLE);
  greenDutyCycle = map(greenDutyCycle, 0, ADC_RESOLUTION, 0,
GREEN_MAX_DUTY_CYCLE);
  blueDutyCycle = map(blueDutyCycle, 0, ADC_RESOLUTION, 0,
BLUE_MAX_DUTY_CYCLE);
  /* Set PWM Output of Channel with desired dutycycle */
  ledcWrite(redPWMChannel, redDutyCycle);
  ledcWrite(greenPWMChannel, greenDutyCycle);
  ledcWrite(bluePWMChannel, blueDutyCycle);

  Serial.println("RED -- GREEN -- BLUE");
  Serial.print(redDutyCycle);
  Serial.print(" -- ");
  Serial.print(greenDutyCycle);
  Serial.print(" -- ");
  Serial.print(blueDutyCycle);
  Serial.print("\n");

  delay(1000);
}
```



From:
https://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
https://chanterie37.fr/fablab37110/doku.php?id=start:esp32:pwm

Last update: **2022/12/20 17:31**