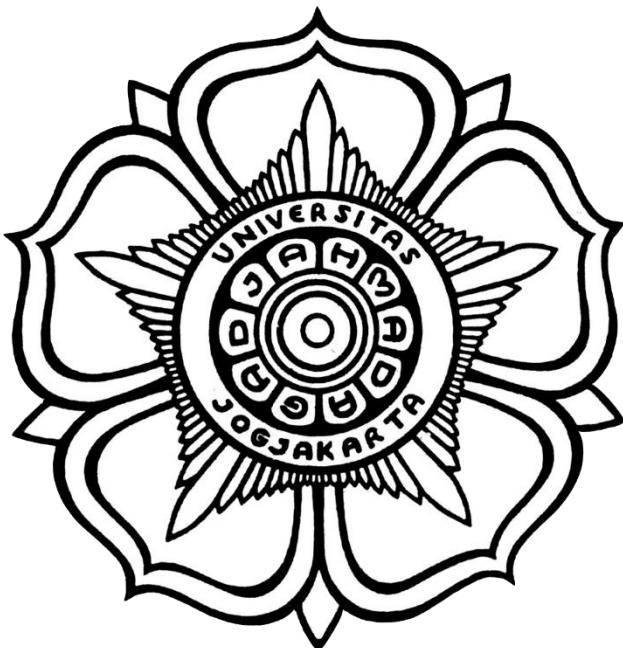


LAPORAN PRAKTIKUM ELEKTRONIKA MESIN LISTRIK DAN TEKNIK KENDALI

“Rangkuman Mengenai ESP 32”

Dosen Pengampu: Irfan Bahiuddin, ST, M.Phil., Ph.D.



Disusun Oleh:

Kelompok 14

Dirgantoro Gerardiawan (19/447114/SV/16833)

Kelas: ARM 2

TEKNOLOGI REKAYASA MESIN

DEPARTEMEN TEKNIK MESIN SEKOLAH VOKASI

UNIVERSITAS GADJAH MADAYOGYAKARTA

2022

1. Mempelajari Spesifikasi

Spesifikasi mikrokontroler esp32

ESP32 adalah mikrokontroler yang dikenalkan oleh Espressif System merupakan penerus dari mikrokontroler ESP8266. Pada mikrokontroler ini sudah tersedia modul WiFi dalam chip sehingga sangat mendukung untuk membuat sistem aplikasi Internet of Things. ESP32 sendiri tidak jauh berbeda dengan ESP8266 yang familiar di pasaran, hanya saja ESP32 lebih komplek dibandingkan ESP8266, cocok untuk sobat dengan proyek yang besar.

berikut ini merupakan spesifikasi yang dimiliki oleh mikrokontroler ESP32 :

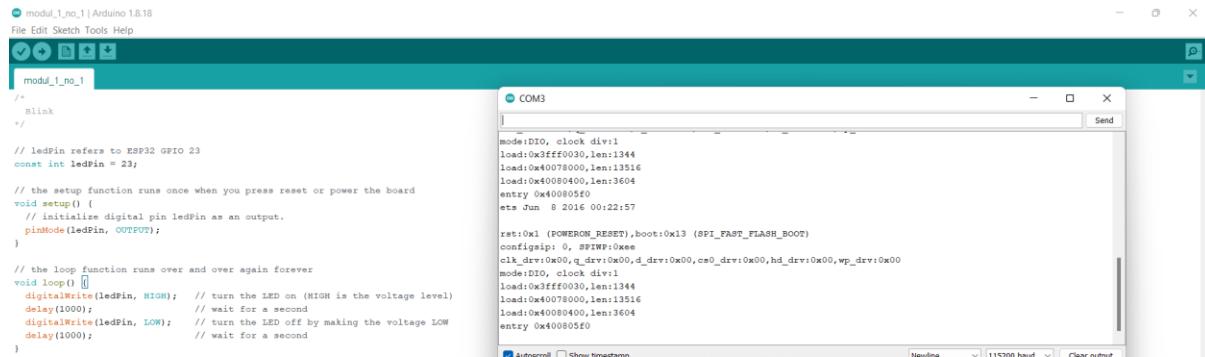
Specifications – ESP32 DEVKIT V1 DOIT

| | |
|------------------------|--|
| Number of cores | 2 (dual core) |
| Wi-Fi | 2.4 GHz up to 150 Mbits/s |
| Bluetooth | BLE (Bluetooth Low Energy) and legacy Bluetooth |
| Architecture | 32 bits |
| Clock frequency | Up to 240 MHz |
| RAM | 512 KB |
| Pins | 30 or 36 (depends on the model) |
| Peripherals | Capacitive touch, ADC (analog to digital converter), DAC (digital to analog converter), I2C (Inter-Integrated Circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I2S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (pulse width modulation), and more. |

Spesifikasi chip ESP32 :

- ESP32 memiliki dual core processor, ini berarti memiliki 2 prosesor.
- Memiliki Wifi dan bluetooth built-in, yang berarti kita dapat melakukan pekerjaan kontrol yang membutuhkan bluetooth dan Wifi secara langsung tanpa perlu menambah modul lain.
- ESP32 dapat menjalankan program 32 bit.
- Frekuensi clock bisa mencapai 240 MHz dan memiliki RAM 512 kB.
- Produk modul yang menggunakan chip ESP32 ini memiliki 30 atau 36 pin.
- Memiliki berbagai macam fungsi yang tersedia, seperti: tombol tekan, ADC,DAC, UART, SPI, I2C dan banyak lagi.
- Terdapat sensor efek hall built-in dan sensor suhu built-in.

Contoh kasus :



```
modul_1_no_1 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_1
/*
  Blink
*/
// ledPin refers to ESP32 GPIO 23
const int ledPin = 23;

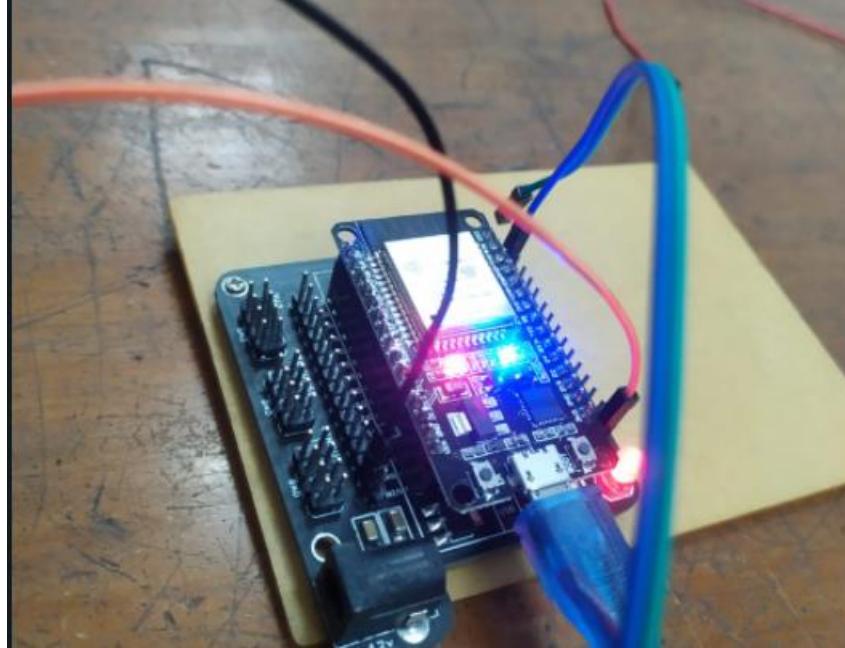
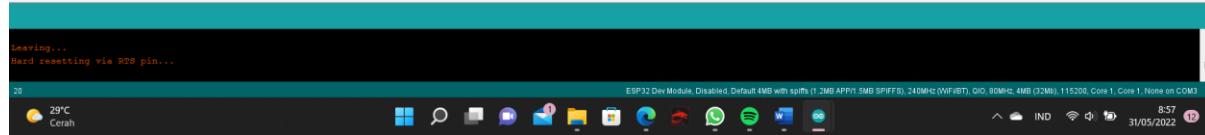
// the setup function runs once when you press reset or power the board
void setup() {
  // initializes digital pin ledPin as an output.
  pinMode(ledPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(ledPin, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}

COM3
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0
etc Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config:0x1F,clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x400805f0

Autoscroll Show timestamp
Newline 115200 baud Clear output
```

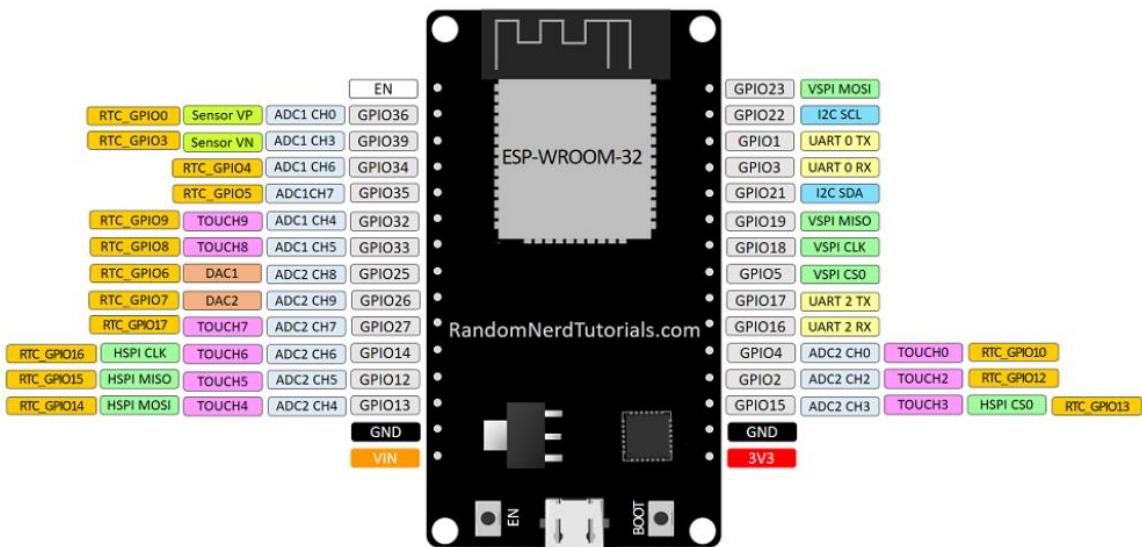


2. Mempelajari Pinout/kaki-kaki beserta fungsinya

Version with 30 GPIOs

ESP32 DEVKIT V1 - DOIT

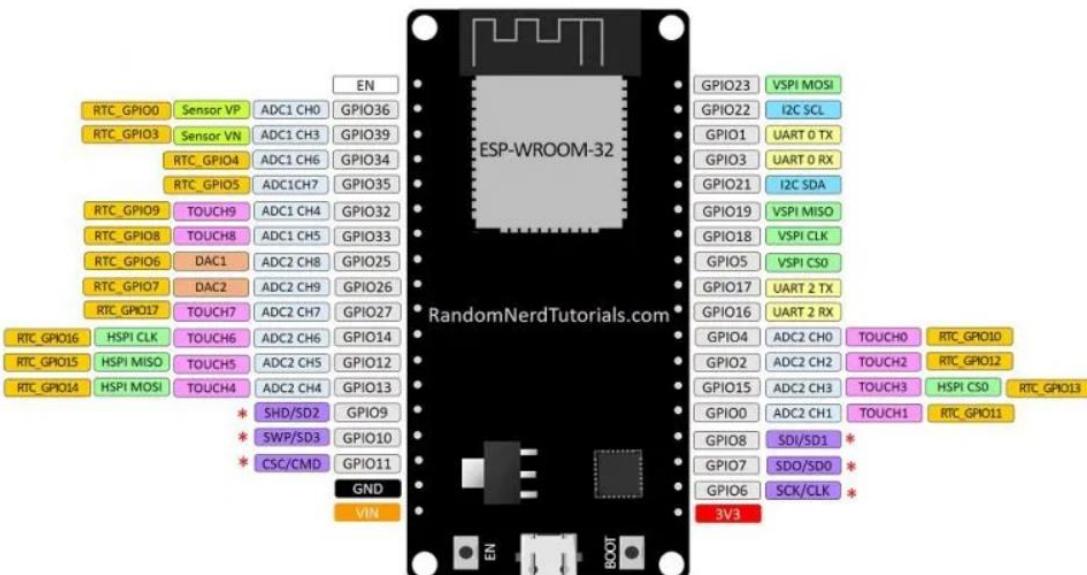
version with 30 GPIOs



Version with 36 GPIOs

ESP32 DEVKIT V1 - DOIT

version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SD1/SD1, SHD/SD2, SWP/SD3 and CSC/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

Terlihat bahwa versi esp32 di atas memiliki banyak fungsi dalam satu pin. Ini adalah fungsi multiplexing dari chip esp32 yang memungkinkan satu pin didefinisikan menjadi beberapa fungsi. Namun, jika pengguna tidak menyetel pin pada saat enkripsi, pin tersebut akan digunakan secara default.

| GPIO | Input | Output | Notes |
|------|-----------|--------|---------------------------------------|
| 0 | pulled up | OK | outputs PWM signal at boot |
| 1 | TX pin | OK | debug output at boot |
| 2 | OK | OK | connected to on-board LED |
| 3 | OK | RX pin | HIGH at boot |
| 4 | OK | OK | |
| 5 | OK | OK | outputs PWM signal at boot |
| 6 | X | X | connected to the integrated SPI flash |
| 7 | X | X | connected to the integrated SPI flash |
| 8 | X | X | connected to the integrated SPI flash |
| 9 | X | X | connected to the integrated SPI flash |
| 10 | X | X | connected to the integrated SPI flash |
| 11 | X | X | connected to the integrated SPI flash |
| 12 | OK | OK | boot fail if pulled high |
| 13 | OK | OK | |
| 14 | OK | OK | outputs PWM signal at boot |
| 15 | OK | OK | outputs PWM signal at boot |
| 16 | OK | OK | |
| 17 | OK | OK | |
| 18 | OK | OK | |
| 19 | OK | OK | |
| 21 | OK | OK | |
| 22 | OK | OK | |
| 23 | OK | OK | |
| 25 | OK | OK | |
| 26 | OK | OK | |
| 27 | OK | OK | |
| 32 | OK | OK | |
| 33 | OK | OK | |
| 34 | OK | | input only |
| 35 | OK | | input only |
| 36 | OK | | input only |
| 39 | OK | | input only |

Sedangkan untuk warna hijau, dapat disimpulkan bahwa dapat digunakan seperti halnya yang menggunakan warna kuning, tetapi jika terdeteksi kuning, harus lebih diperhatikan karena kegagalan dapat terjadi, terutama pada saat startup.
 warna merah maka sangat untuk tidak direkomendasikan untuk digunakan pada input serta output.

Contoh kasus :

```
modul_1_no_2 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_2
/*
  Blink
*/
// ledPin refers to ESP32 GPIO 23
const int ledPin = 23;

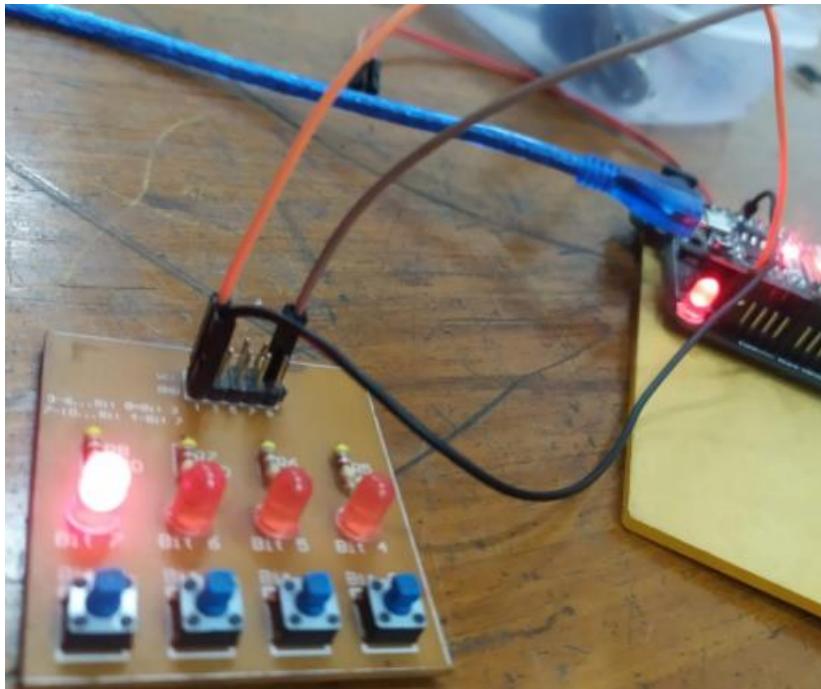
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin ledPin as an output.
  pinMode(ledPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                 // wait for a second
  digitalWrite(ledPin, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                 // wait for a second
}

Done compiling.

Sketch uses 210609 bytes (16%) of program storage space. Maximum is 1310720 bytes.
Global Variables use 16044 bytes (4%) of dynamic memory, leaving 311636 bytes for local variables. Maximum is 327680 bytes.

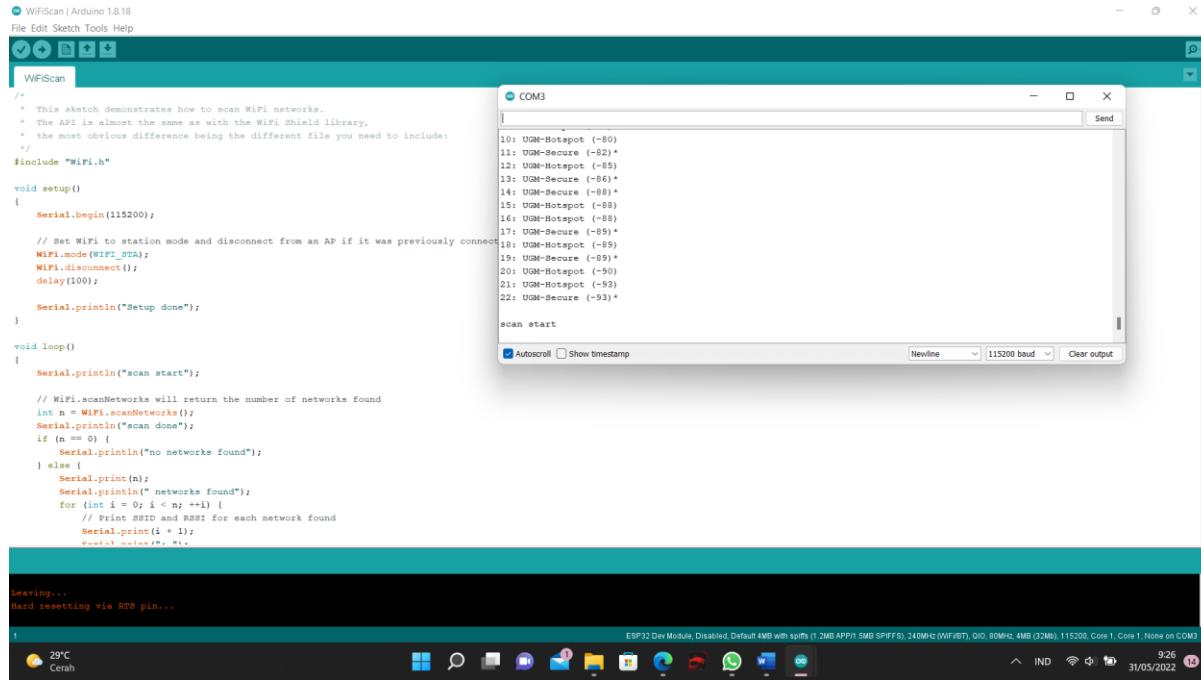
29
26°C
Hujan ringan
ESP32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), DIO, 80MHz, 4MB (32MB), 115200, Core 1, Core 1, None on COM3
^ INO 042 01/06/2022
```



3. Mempelajari penggunaan Arduino IDE (seting/konfigurasi)

Arduino IDE merupakan sebuah Software Compiler dari ARDUINO yang dapat digunakan untuk melakukan programming logic microcontroller salah satunya ESP32

Mendeteksi WiFi disekitar :



The screenshot shows the Arduino IDE interface with the WiFiScan sketch open. The code is as follows:

```
/* This sketch demonstrates how to scan WiFi networks.
 * The API is almost the same as with the WiFi_Shield library,
 * the most obvious difference being the different file you need to include:
 */
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previously connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(1000);

    Serial.println("Setup done");
}

void loop()
{
    Serial.println("scan start");

    // WiFi.scanNetworks will return the number of networks found
    int n = WiFi.scanNetworks();
    Serial.println("scan done");
    if (n == 0) {
        Serial.println("no networks found");
    } else {
        Serial.print(n);
        Serial.println(" networks found");
        for (int i = 0; i < n; ++i) {
            // Print SSID and RSSI for each network found
            Serial.print(i + 1);
            Serial.print(" - ");
            Serial.print(WiFi.SSID(i));
            Serial.print(" (");
            Serial.print(WiFi.RSSI(i));
            Serial.println(")");
        }
    }
}

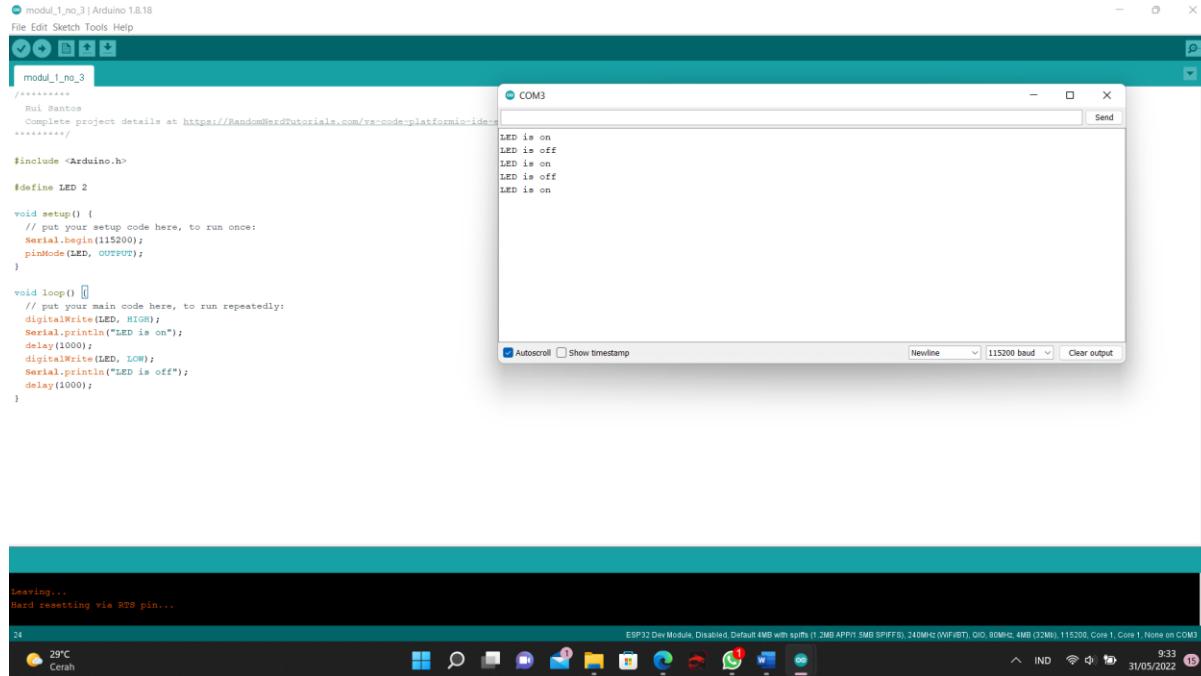
leaving...
Hard resetting via RTS pin...
```

The serial monitor window shows the results of the WiFi scan:

```
COM3
10: OGM-Hotspot (-80)
11: OGM-Secure (-02)*
12: OGM-Hotspot (-05)
13: OGM-Secure (-06)*
14: OGM-Secure (-08)*
15: OGM-Hotspot (-08)
16: OGM-Hotspot (-08)
17: OGM-Secure (-09)*
18: OGM-Hotspot (-09)
19: OGM-Secure (-09)*
20: OGM-Hotspot (-90)
21: OGM-Hotspot (-93)
22: OGM-Secure (-93)*

scan start
Autoscroll Show timestamp
Newline 115200 baud Clear output
```

Led ON/OFF



The screenshot shows the Arduino IDE interface with the modul_1_no_3 sketch open. The code is as follows:

```
/*
  Rui Santos
  Complete project details at https://RandomNerdTutorials.com/esp32-code-platformio-ide/
*/

#include <Arduino.h>

#define LED 2

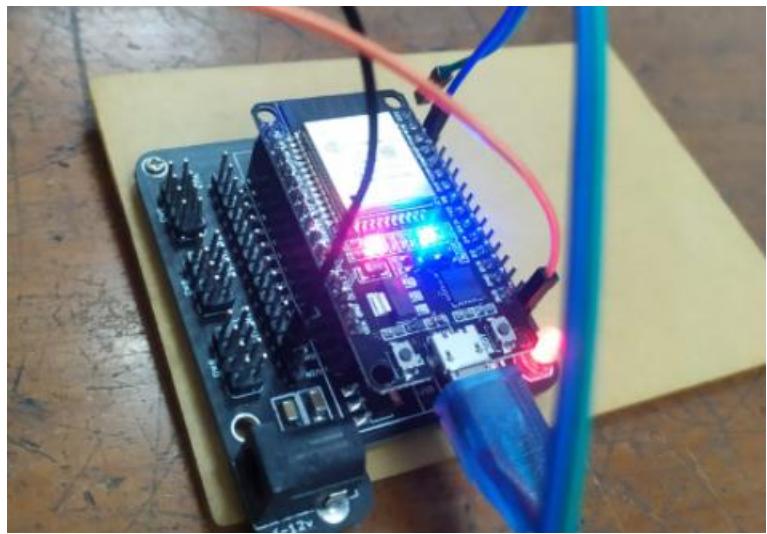
void setup() {
    // put your setup code here, to run once:
    Serial.begin(115200);
    pinMode(LED, OUTPUT);
}

void loop() {
    // put your main code here, to run repeatedly:
    digitalWrite(LED, HIGH);
    Serial.println("LED is on");
    delay(1000);
    digitalWrite(LED, LOW);
    Serial.println("LED is off");
    delay(1000);
}
```

The serial monitor window shows the LED state changes:

```
COM3
LED is on
LED is off
LED is on
LED is off
LED is on

Autoscroll Show timestamp
Newline 115200 baud Clear output
```



4. Mempelajari Output/Input

- 18 buah kanal Analog-to-Digital Converter (ADC)
- 3 buah antarmuka Serial-Parallel Interface (SPI)
- 3 buah antarmuka UART
- 2 buah antarmuka I2C
- 16 kanal output PWM
- 2 Digital-to-Analog Converter (DAC)
- 2 buah antarmuka I2S
- 10 buah GPIO Capacitive Sensing

a. ESP32 Input Digital

Input digital dapat dilakukan di ESP32 dengan melakukan konfigurasipin seperti langkah berikut :

Pertama, atur GPIO yang ingin di baca sebagai INPUT, menggunakanfungsi pinMode() sebagai berikut:

```
pinMode(GPIO, INPUT);
```

Untuk membaca input digital, seperti tombol, Kita bisa menggunakan fungsi digitalRead(), yang menerima sebagai argumen, GPIO (nomor int) yang dimaksud.

```
digitalRead(GPIO);
```

Semua GPIO ESP32 dapat digunakan sebagai input, kecuali GPIO 6hingga 11 (karena sudah terhubung ke flash SPI terintegrasi).

b. Kontrol Output Digital ESP32

Output digital dapat dilakukan di ESP32 dengan melakukan konfigurasipin seperti langkah berikut :

Pertama, Kita perlu mengatur GPIO yang ingin di kontrol sebagaiOUTPUT. Menggunakan fungsi pinMode() sebagai berikut:

```
pinMode(GPIO, OUTPUT);
```

Untuk mengontrol output digital, Kami hanya perlu menggunakan digitalWrite() fungsi, yang menerima sebagai argument, GPIO (nomor int), dan statusnya HIGH atau LOW.

```
digitalWrite(GPIO, STATE);
```

Semua GPIO dapat digunakan sebagai output kecuali GPIO 6 hingga 11(terhubung ke flash SPI terintegrasi) dan GPIO 34, 35, 36 dan 39 (hanya input GPIO);

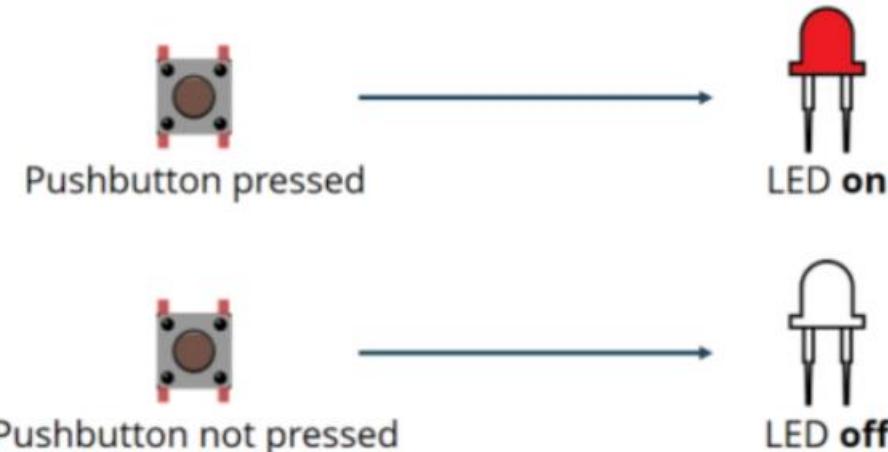
c. Mengunggah Kode

Sebelum mengklik tombol unggah, buka **Alat > Papan** , dan pilih papan yang digunakan. Dalam kasus ini adalah **papan DOIT ESP32DEVKIT V1**. Buka **Alat >**

Port dan pilih port COM yang terhubung dengan ESP32. Kemudian, tekan tombol unggah dan tunggu pesan “**Done Upload**”.

Jika terlihat banyak titik (..._...) pada jendela debugging dan pesan “Gagal terhubung ke ESP32: Waktu habis menunggu header paket”, itu berarti perlu menekan tombol BOOT on-board ESP32 setelah titik mulai muncul

Contoh kasus :



The screenshot shows the Arduino IDE interface. The code in the editor is:

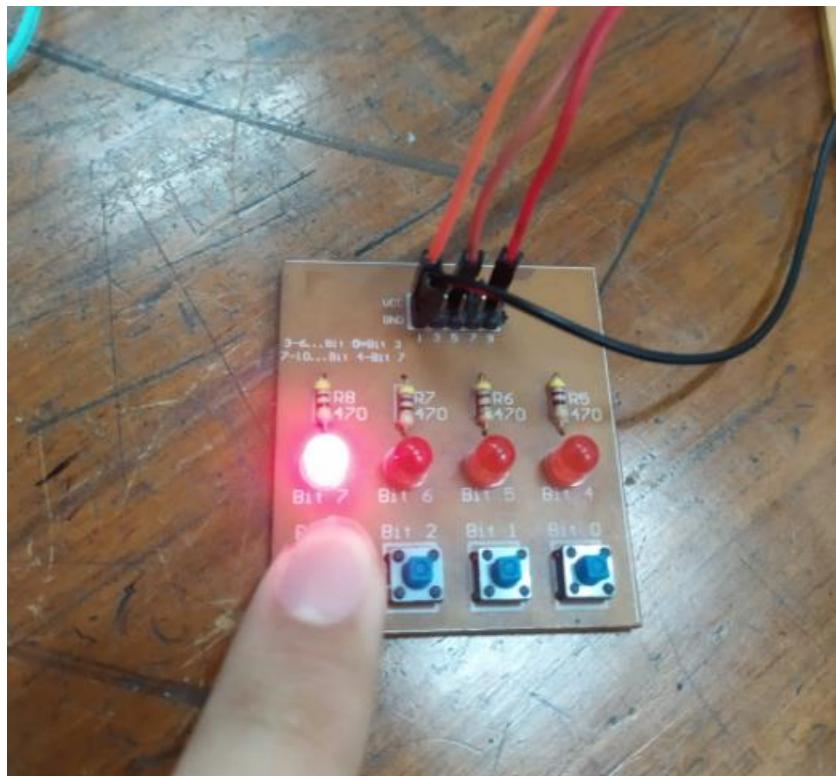
```
// modul_1_no_4 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_4
// Complete Instructions: https://RandomNerdTutorials.com/esp32-digital-inputs-outputs
// set pin numbers
const int buttonPin = 4; // the number of the pushbutton pin
const int ledPin = 5; // the number of the LED pin

// variable for storing the pushbutton status
int buttonState = 0;

void setup() {
  Serial.begin(115200);
  // initialize the pushbutton pin as an input
  pinMode(buttonPin, INPUT);
  // initialize the LED pin as an output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the state of the pushbutton value
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);
  // check if the pushbutton is pressed.
  // if it is, the buttonstate is HIGH
  if (buttonState == HIGH) {
    // turn LED on
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off
    digitalWrite(ledPin, LOW);
  }
}
```

The serial monitor window shows the output "1" repeated multiple times, indicating the state of the pushbutton. The system is currently in a loop, as evidenced by the "leaving..." message at the bottom of the terminal window.



5. Mempelajari tentang PWM

a. Pengontrol PWM LED ESP32

ESP32 memiliki pengontrol PWM LED dengan 16 saluran independen yang dapat dikonfigurasi untuk menghasilkan sinyal PWM dengan properti yang berbeda. Berikut langkah-langkah untuk meredupkan LED dengan PWM menggunakan Arduino IDE:

- i. Pilih saluran PWM. Ada 16 saluran dari 0 hingga 15.
- ii. Kemudian, atur frekuensi sinyal PWM. Untuk LED, frekuensi 5000 Hz baik-baik saja untuk digunakan.
- iii. Atur resolusi siklus tugas sinyal: ESP32 memiliki resolusi dari 1 hingga 16 bit. Kami akan menggunakan resolusi 8-bit, yang berarti dapat mengontrol kecerahan LED menggunakan nilai dari 0 hingga 255.
- iv. Selanjutnya, tentukan ke GPIO atau GPIO mana sinyal akan muncul. Dengan menggunakan fungsi berikut:

`ledcAttachPin(GPIO, channel)`

Fungsi ini menerima dua argumen. Yang pertama adalah GPIO yang akan mengeluarkan sinyal, dan yang kedua adalah saluran yang akan menghasilkan sinyal.

- v. Terakhir, untuk mengontrol kecerahan LED menggunakan PWM, digunakan fungsi berikut:

`ledcWrite(channel, dutycycle)`

Fungsi ini menerima sebagai argumen saluran yang menghasilkan sinyal PWM, dan siklus kerja.

b. Menentukan tempat pin LED terpasang dengan GPIO 16

```
const int ledPin = 16; // 16 corresponds to GPIO16
```

Kemudian, mengatur properti sinyal PWM. Menentukan frekuensi 5000 Hz, memilih saluran 0 untuk menghasilkan sinyal, dan mengatur resolusi 8 bit. Kita dapat memilih properti lain, berbeda dari ini, untuk menghasilkan sinyal PWM yang berbeda.

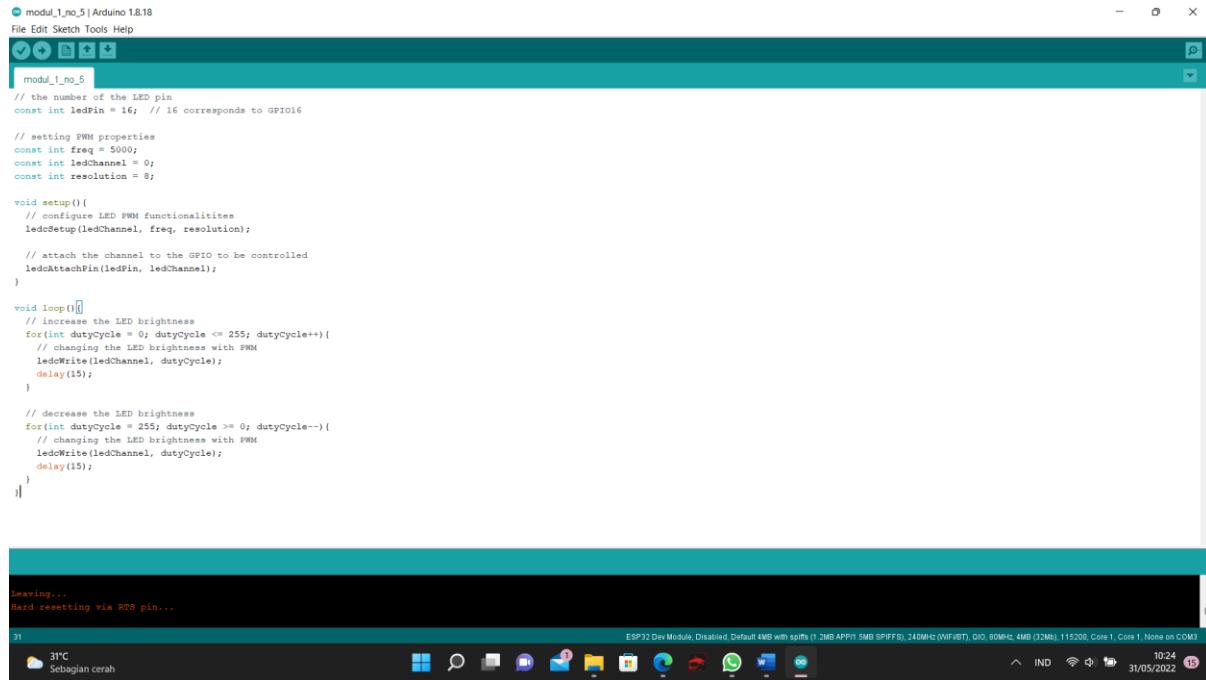
```
const int freq = 5000;  
const int ledChannel = 0;  
const int resolution = 8;
```

Dalam setup(), perlu mengonfigurasi LED PWM dengan properti yang telah ditentukan sebelumnya dengan menggunakan fungsi ledcSetup() yang diterima sebagai argumen, ledChannel, frekuensi, dan resolusi, sebagai berikut:

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

Dalam loop, akan memvariasikan siklus tugas antara 0 dan 255 untuk meningkatkan kecerahan LED.

Contoh kasus :



```
// modul_1_no_5 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_5
modul_1_no_5
// the number of the LED pin
const int ledPin = 16; // 16 corresponds to GPIO16

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

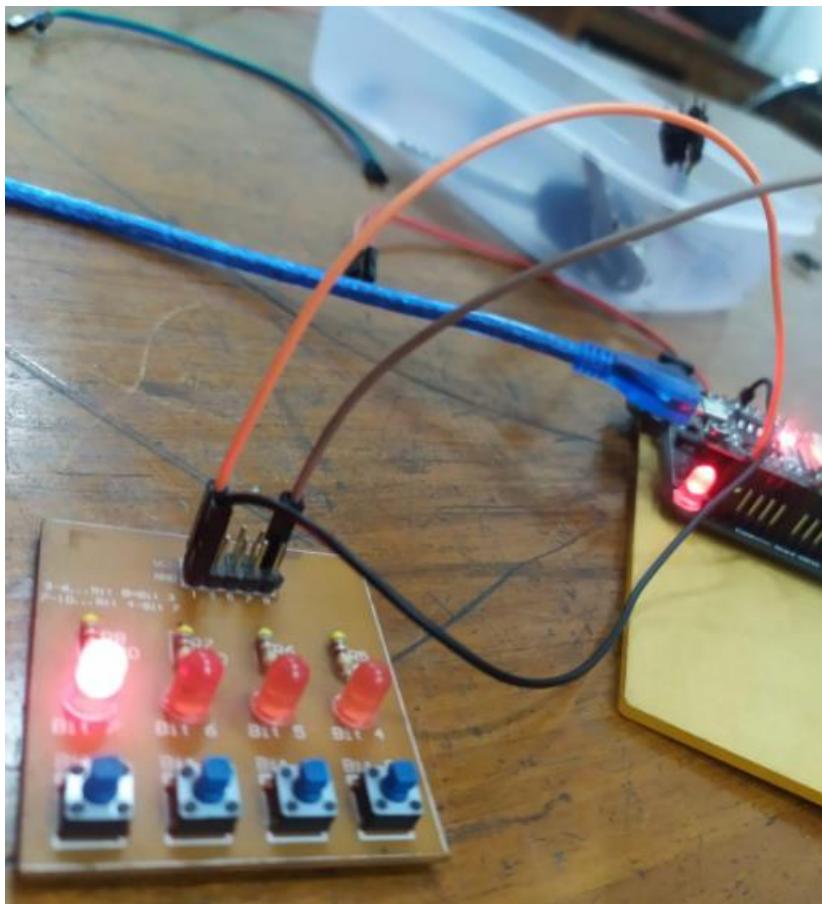
void setup(){
  // configure LED PWM functionalitites
  ledcSetup(ledChannel, freq, resolution);

  // attach the channel to the GPIO to be controlled
  ledcAttachPin(ledPin, ledChannel);
}

void loop(){
  // increase the LED brightness
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }

  // decrease the LED brightness
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }
}

Leaving...
Hard resetting via RST pin...
31 31°C Sebagian cerah 10:24 ESP32 Dev Module, Disabled. Default 4MB with splits (1.2MB APP/1.5MB SPIFFS), 240MHz (WIFi6), C10, 80MHz, 4MB (32MB), 115200, Core 1, Core 1. None on COM1
IND INDO 31/05/2022
```



6. Mempelajari tentang perintah interrupt dan millis

a. Cara Kerja :

Untuk men-trigger suatu kerja dari sensor gerak PIR. Dapat membuat sesuatu menjadi terjadi dengan sendirinya dalam suatu program mikrokontroller dan juga dapat memecahkan masalah waktu.

Untuk mengatur interupsi di Arduino IDE, kami menggunakan `AttachInterrupt()` fungsi, yang menerima sebagai argumen: pin GPIO, nama fungsi yang akan dieksekusi, dan mode:

`attachInterrupt(digitalPinToInterrupt(GPIO), function, mode);`

Argumen ketiga adalah modus. Ada 5 mode berbeda:

- i. LOW: untuk memicu interupsi setiap kali pin LOW;
- ii. HIGH: untuk memicu interupsi setiap kali pin HIGH;
- iii. CHANGE: untuk memicu interupsi setiap kali pin berubah nilai – misalnya dari HIGH ke LOW atau LOW ke HIGH;
- iv. FALLING: ketika pin berubah dari HIGH ke LOW;
- v. RISING: untuk memicu ketika pin berubah dari LOW ke HIGH

b. Fungsi Milis

Menggunakan fungsi yang disebut `millis()` dengan tujuan dapat mengembalikan jumlah milidetik yang telah berlalu sejak program pertama kali dimulai.

`millis()`

Fungsi ini dapat dengan mudah memverifikasi berapa banyak waktu yang telah berlalu tanpa memblokir kode yang lainnya.

c. Fungsi Delay

Delay berfungsi untuk menerima nomor int tunggal sebagai argumen. Angka ini menunjukkan waktu dalam milidetik program harus menunggu sampai pindah ke baris kode berikutnya.

`delay(time in milliseconds)`

Fungsi pemblokiran mencegah program melakukan hal lain sampai tugas tertentu selesai. Jika pekerjaan membutuhkan banyak tugas untuk dilakukan secara bersamaan, hal itu tidak dapat menggunakan `delay()`. Untuk sebagian besar proyek, harus menghindari penggunaan delay dan menggunakan pengatur waktu sebagai gantinya.

Adapun fungsi delay dan milis, delay berfungsi untuk menunda program dalam waktu tertentu . Sedangkan milis berfungsi untuk mengembalikan jumlah detik yang ada tanpa harus memblokir program.

Contoh kasus mengedipkan LED dengan Millis :

Kodingan berikut menunjukkan bagaimana dapat digunakan millis () berfungsi untuk membuat proyek LED berkedip. Itu menyalakan LED selama 1000 milidetik, dan kemudian mematikannya.

```
modul_1_no_6 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_6
// The value will quickly become too large for an int to store
unsigned long previousMillis = 0; // will store last time LED was updated

// constants won't change :
const long interval = 1000; // interval at which to blink (milliseconds)

void setup() {
  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // here is where you'd put code that needs to be running all the time.

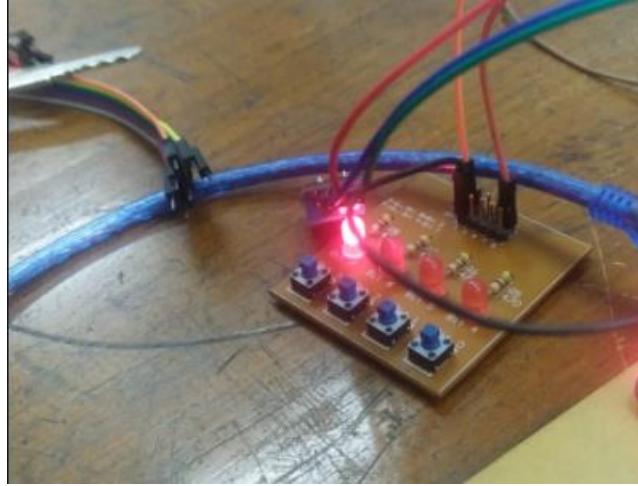
  // check to see if it's time to blink the LED; that is, if the
  // difference between the current time and last time you blinked
  // the LED is bigger than the interval at which you want to
  // blink the LED.
  unsigned long currentMillis = millis();

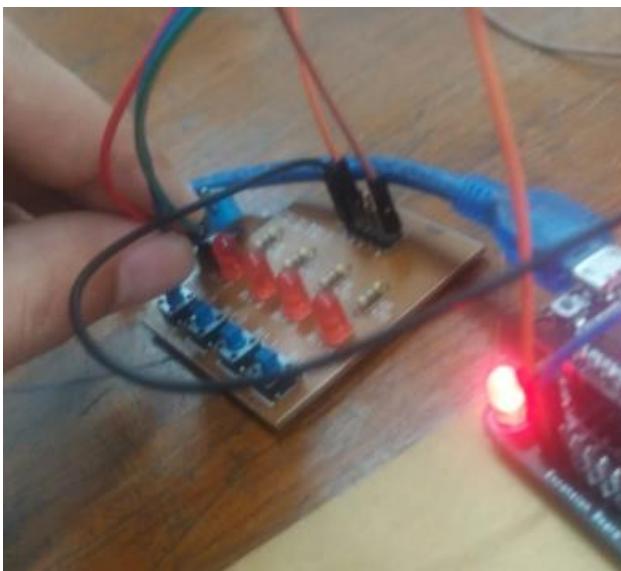
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // if the LED is off turn it on and vice-versa:
    if (ledState == LOW) {
      ledState = HIGH;
    } else {
      ledState = LOW;
    }

    // set the LED with the ledState of the variable:
    digitalWrite(ledPin, ledState);
  }
}

Leaving...
Hard resetting via RTS pin...
47
```





b. Menggunakan Sensor gerak (suhu)

The screenshot shows the Arduino IDE interface. The left pane displays the C++ code for a sketch named 'modul_1_no_6_2'. The code includes functions for motion detection using a PIR sensor, setting up a serial port for debugging, and managing an LED. The right pane shows a terminal window titled 'COM3' displaying the boot logs of an ESP32 Dev Module. The logs include information about the CPU reset, memory configuration, and clock settings. At the bottom, there is a status bar with system icons and a timestamp.

```
modul_1_no_6_2

void IRAM_ATTR detectMovement() {
    Serial.println("MOTION DETECTED!!!");
    digitalWrite(Led, HIGH);
    startTimer = true;
    lastTrigger = millis();
}

void setup() {
    // Serial port for debugging purposes
    Serial.begin(115200);

    // PIR Motion Sensor mode INPUT_PULLUP
    pinMode(motionSensor, INPUT_PULLUP);
    // Set motionSensor pin as interrupt, assign interrupt function and set RISING mode
    attachInterrupt(digitalPinToInterrupt(motionSensor), detectMovement, RISING);

    // Set LED to LOW
    pinMode(Led, OUTPUT);
    digitalWrite(Led, LOW);
}

void loop() {
    // Current time
    now = millis();
    // Turn off the LED after the number of seconds defined in the timeSeconds variable
    if(startTimer && (now - lastTrigger > (timeSeconds*1000))) {
        Serial.println("Motion stopped...");
        digitalWrite(Led, LOW);
        startTimer = false;
    }
}

Leaving...
Serial resetting via RTS pin...

82
```

COM3

Send

ELF file SMAX256: 0000000000000000

Rebooting...

ets Jun 8 2016 00:22:57

rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configip: 0, SFIFW:0xee
clk_drv:0x000, q_drv:0x000, d_drv:0x000, cs0_drv:0x000, hd_drv:0x000, wp_drv:0x00
modeID:0, clock div:1
loadi:0x3ffff000, len:1344
loadi:0x40078000, len:13516
loadi:0x40080400, len:3604
entry: 0x400005E0
MOTION DETECTED!!
Motion stopped...

Autoscroll Show timestamp

Nevilne 115200 baud Clear output

10:55 31/10/2022

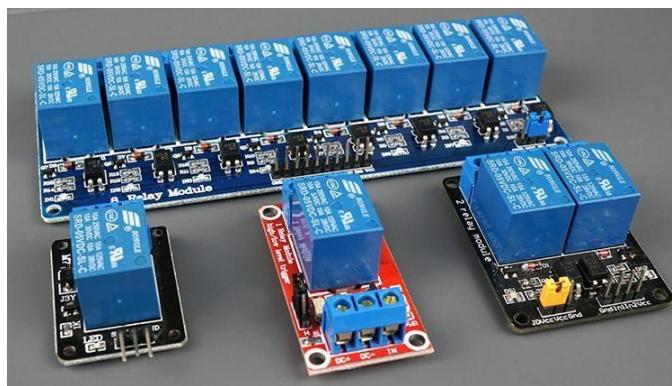
7. Mempelajari tentang relay

a. Memperkenalkan Relay

Relay adalah saklar yang dioperasikan secara listrik dan seperti saklar lainnya, relay dapat dihidupkan atau dimatikan, membiarkan arus mengalir atau tidak.

b. Modul relay 1, 2, 4, 8, 16 Saluran

Ada modul relay yang berbeda dengan jumlah saluran yang berbeda. dapat ditemukan modul relay dengan satu, dua, empat, delapan, dan bahkan enam belas saluran. Jumlah saluran menentukan jumlah keluaran yang dapat dikendalikan.



Ada modul relay yang elektromagnetnya dapat ditenagai oleh 5V dan dengan 3.3V. Keduanya dapat digunakan dengan ESP32 – atau dapat menggunakan pin VIN (yang menyediakan 5V) atau pin 3.3V. Selain itu, beberapa dilengkapi dengan optocoupler internal yang menambahkan "lapisan" perlindungan ekstra, yang secara optik mengisolasi ESP32 dari sirkuit relay.

c. Pin Kontrol

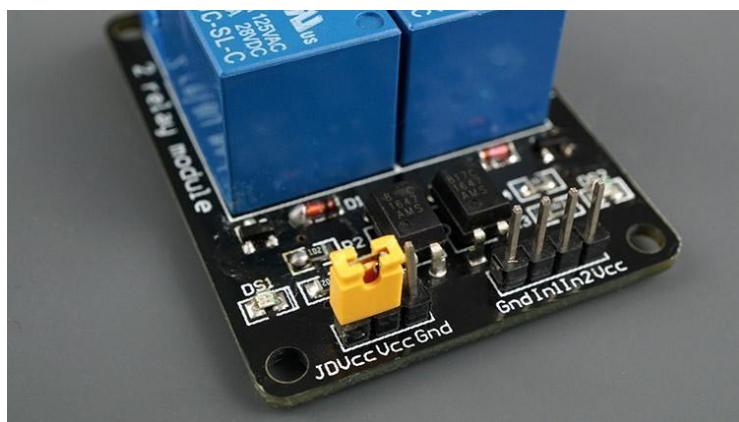


Sisi tegangan rendah memiliki satu set empat pin dan satu set tiga pin. Set pertama terdiri dari **VCC** dan **GND** untuk menyalaikan modul, dan masukan 1 (**IN1**) dan masukan 2 (**IN2**) untuk mengontrol relay bawah dan atas, masing-masing

Sinyal yang dikirim ke pin IN, menentukan apakah relay aktif atau tidak. relay dipicu ketika input berjalan di bawah sekitar 2V.

- **Normally Closed configuration (NC) :**
 - Sinyal HIGH – arus mengalir
 - Sinyal LOW – arus **tidak** mengalir
- **Normally Open configuration (NO) :**
 - Sinyal HIGH – arus **tidak** mengalir
 - Sinyal LOW – arus mengalir

d. Pilihan Power Supply

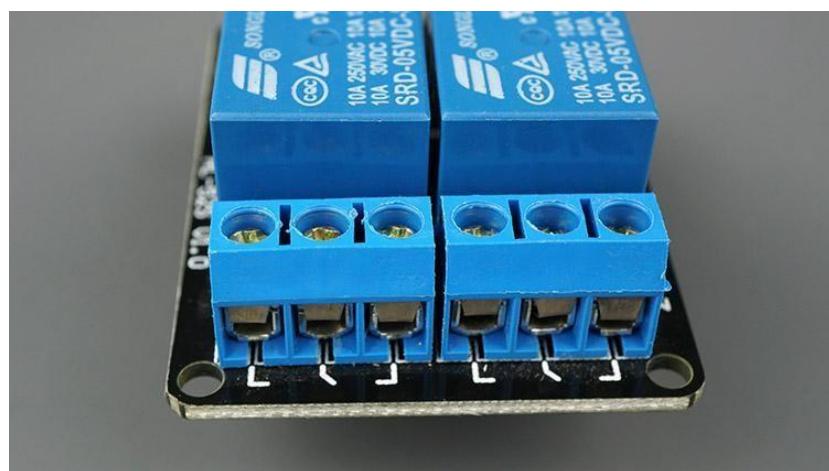


Set kedua pin terdiri dari: **GND**, **VCC**, dan **JD-VCC** pin. **JD-VCC** pin memberi daya pada elektromagnet relay. Perhatikan bahwa modul memiliki tutup jumper yang menghubungkan pin VCC dan JD-VCC; yang ditampilkan di sini berwarna kuning, tetapi warna mungkin berbeda

e. Menghubungkan Modul Relay ke ESP32

Hubungkan modul relay ke ESP32 seperti yang ditunjukkan pada diagram berikut. Diagram menunjukkan pengkabelan untuk modul relay 2 saluran, pengkabelan dengan jumlah saluran yang berbeda serupa

f. Koneksi Tegangan Listrik



Modul relay yang ditunjukkan pada foto sebelumnya memiliki dua konektor, masing-masing dengan tiga soket: common (**COM**), NormallyClosed

(**NC**), and Normally Open (**NO**).

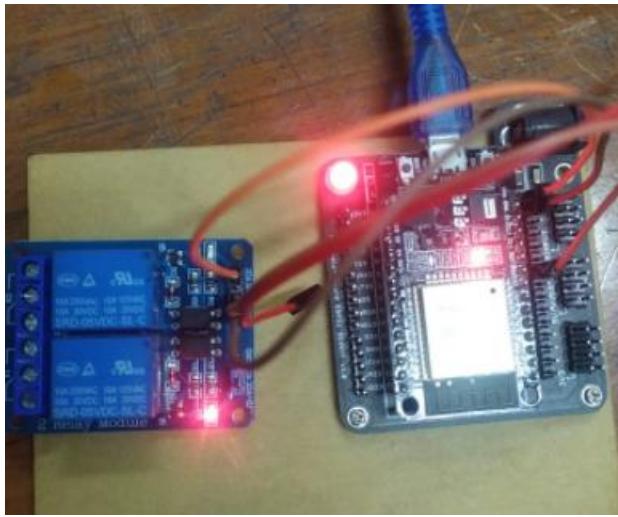
- **COM:** hubungkan arus yang ingin dikendalikan (tegangan listrik).
- **NC (Normally Closed):** konfigurasi yang biasanya tertutup digunakan bila ingin relay ditutup secara default
- **NO (Normally Open):** konfigurasi yang biasanya terbuka bekerja sebaliknya: tidak ada koneksi antara pin NO dan COM, sehingga sirkuit terputus kecuali jika ingin mengirim sinyal dari ESP32 untuk menutup sirkuit.

g. Kontrol Beberapa Relay dengan Server Web ESP32



Di bagian ini, kami telah membuat contoh server web yang memungkinkan fungsi untuk mengontrol relay sebanyak yang diinginkan melalui server web

Contoh kasus relay :



```
modul_1_no_7 | Arduino 1.8.18
File Edit Sketch Tools Help
Serial Monitor
modul_1_no_7
*****
Rui Santos
Complete project details at https://RandomNerdTutorials.com/esp32-relay-module-ac-wireless/
The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the software.
*****
const int relay = 26;

void setup() {
  Serial.begin(115200);
  pinMode(relay, OUTPUT);
}

void loop() {
  // Normally Open configuration, send LOW signal to let current flow
  // (if you're using Normally Closed configuration send HIGH signal)
  digitalWrite(relay, LOW);
  Serial.println("Current Flowing");
  delay(5000);

  // Normally Open configuration, send HIGH signal stop current flow
  // (if you're using Normally Closed configuration send LOW signal)
  digitalWrite(relay, HIGH);
  Serial.println("Current not Flowing");
  delay(5000);
}
```

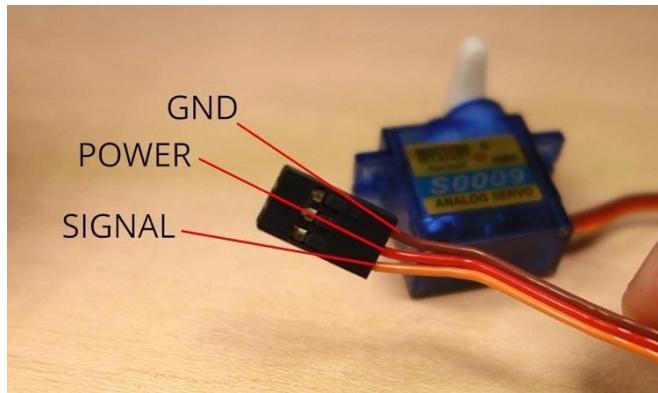
```
Leaving...
Hard resetting via RTS pin...

28 30°C Sebagian cerah
ESP32 Dev Module, Disabled, Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (2MB), Core 1, Core 1, None on COM3
^ IND WiFi 11:32 31/05/2022 10
```

8. Mempelajari Esp32 Servo

a. Menghubungkan Motor Servo ke ESP32

Motor servo memiliki tiga kabel: daya, ground, dan sinyal. Daya biasanya berwarna merah, GND berwarna hitam atau coklat, dan kabel sinyal biasanya berwarna kuning, oranye, atau putih.

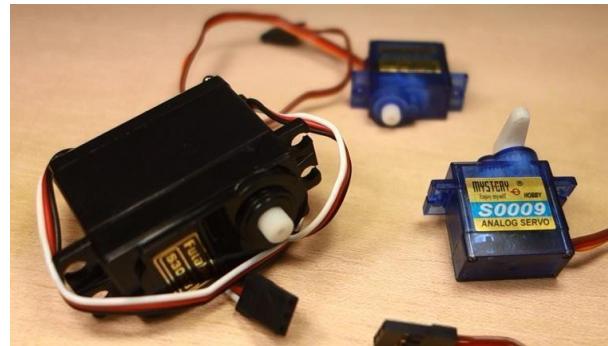


| Kabel | Warna |
|----------|----------------------------|
| Kekuatan | Merah |
| GND | Hitam, atau coklat |
| Sinyal | Kuning, orange, atau putih |

Saat menggunakan servo kecil seperti S0009 seperti yang ditunjukkan pada gambar di bawah, Kami dapat menyalakannya langsung dari ESP32.



Tetapi jika Kami menggunakan lebih dari satu servo atau jenis lainnya, Kami mungkin perlu menyalakan servo Kami menggunakan power supply eksternal.



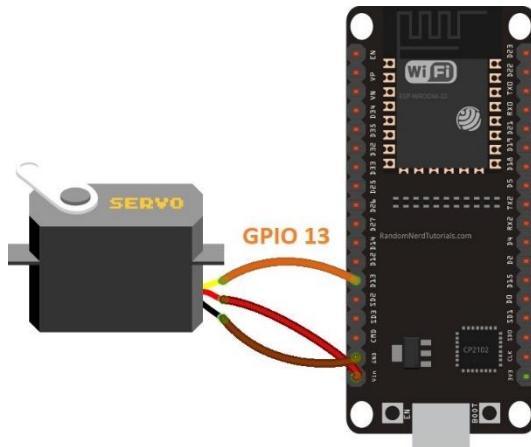
Jika Kami menggunakan servo kecil seperti S0009, Kami harus menghubungkan:

- i. GND -> ESP32 GND pin;
- ii. Daya -> ESP32 VIN pin;
- iii. Sinyal -> **GPIO 13** (atau pin PWM apa pun).

Catatan: dalam hal ini, Kami dapat menggunakan GPIO ESP32 apa pun, karena GPIO apa pun dapat menghasilkan sinyal PWM. Namun, kami tidak menyarankan penggunaan GPIO 9, 10, dan 11 yang terhubung ke flash SPI terintegrasi dan tidak disarankan untuk penggunaan lain.

• Skema

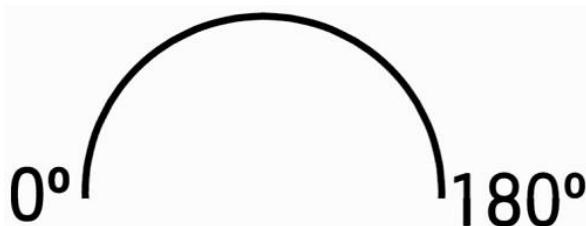
Dalam contoh ini, kami akan menghubungkan kabel sinyal ke **GPIO 13**.



(Skema ini menggunakan versi modul [ESP32 DEVKIT V1](#) dengan 36 GPIO – jika yang digunakan model lain, periksa pinout untuk papan yang Kami gunakan.)

b. Bagaimana Cara Mengontrol Motor Servo?

Kita dapat memposisikan poros servo dalam berbagai sudut dari 0 hingga 180°. Servo dikendalikan menggunakan sinyal modulasi lebar pulsa (PWM). Artinya sinyal PWM yang dikirim ke motor akan menentukan posisi poros.



Untuk mengendalikan motor cukup menggunakan kemampuan PWM dari ESP32 dengan mengirimkan sinyal 50Hz dengan lebar pulsa yang sesuai. Atau Kami dapat menggunakan perpustakaan untuk membuat tugas ini lebih sederhana.

c. Mempersiapkan Arduino IDE

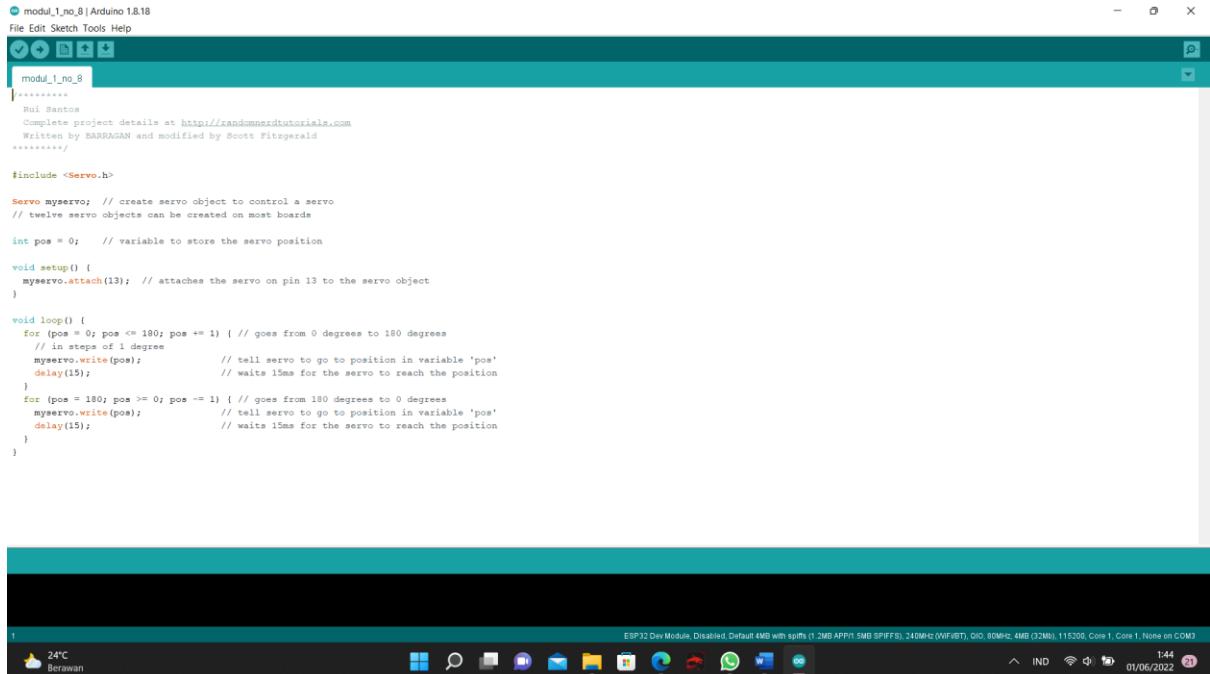
Add-on untuk Arduino IDE memungkinkan untuk memprogram ESP32 menggunakan Arduino IDE dan bahasa pemrogramannya.

d. Menginstal ESP32 Arduino Servo Library

[ESP32 Arduino Servo Library](#) memudahkan untuk mengontrol motor servo. Dengan ESP32 yang menggunakan Arduino IDE. Ikuti langkah-langkah berikut untuk menginstal library di Arduino IDE:

1. Klik di sini untuk mengunduh [ESP32_Arduino_Servo_Library](#). Kami harus memiliki folder .zip di folder Unduhan.
2. Buka zip folder .zip dan Kami akan mendapatkan folder *ESP32-Arduino-Servo-Library-Master*.
3. Ganti nama folder Kami dari *ESP32-Arduino-Servo-Library-Master* ke *ESP32_Arduino_Servo_Library*
4. Pindahkan folder *ESP32_Arduino_Servo_Library* ke folder library instalasi Arduino IDE.
5. Terakhir, buka kembali Arduino IDE.

Contoh kasus menggerakan servo :



The screenshot shows the Arduino IDE interface with the following code:

```
modul_1_no_8 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_8
*****
ARDUINO
Complete project details at https://randomnerdtutorials.com
Written by BARRAGAN and modified by Scott Fitzgerald
*****/
#include <Servo.h>

Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0; // variable to store the servo position

void setup() {
  myservo.attach(13); // attaches the servo on pin 13 to the servo object
}

void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
}
```

The code initializes a servo object and attaches it to pin 13. It then enters a loop where it rotates the servo from 0 to 180 degrees in increments of 1 degree, waits 15ms, and then rotates it back from 180 to 0 degrees in decrements of 1 degree, also waiting 15ms. This creates a continuous back-and-forth motion.

e. Menguji Sketsa

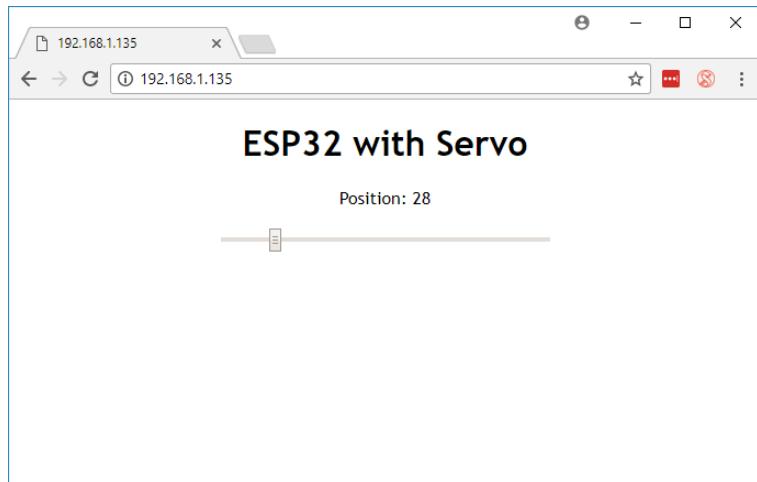
Unggah kode ke ESP32 yang telah dibuat. Setelah mengupload kode, akan terlihat poros motor berputar ke satu sisi dan kemudian ke sisi lainnya.



f. Membuat Server Web ESP32

Setelah mengetahui cara mengontrol servo dengan ESP32, selanjutnya membuat server web untuk mengontrolnya. Spesifikasi server web yang akan kami buat adalah sebagai berikut:

- i. Berisi slider dari 0 hingga 180, yang dapat Kami sesuaikan untuk mengontrol posisi poros servo;
- ii. Nilai sliding saat ini diperbarui secara otomatis di halaman web, serta posisi poros, tanpa perlu menyegarkan halaman web.
Untukini, kami menggunakan AJAX untuk mengirim permintaan HTTPke ESP32 di latar belakang;
- iii. Menyegarkan halaman web tidak mengubah nilai sliding, begitupula posisi poros.



g. Membuat Halaman HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <link rel="icon" href="data:, ">
    <style>
        body {
            text-align: center;
            font-family: "Trebuchet MS", Arial;
            margin-left: auto;
            margin-right: auto;
        }
        .slider {
            width: 300px;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.
3.1/jquery.min.js"></script>
</head>
<body>
    <h1>ESP32 with Servo</h1>
    <p>Position: <span id="servoPos"></span></p>
        <input type="range" min="0" max="180"
class="slider" id="servoSlider"
onchange="servo(this.value)"/>
    <script>
        var slider =
document.getElementById("servoSlider");
        var servoP =
document.getElementById("servoPos");
        servoP.innerHTML = slider.value;
        slider.oninput = function() {
```

```
    slider.value = this.value;
    servoP.innerHTML = this.value;
}
$.ajaxSetup({timeout:1000});
function servo(pos) {
    $.get("/?value=" + pos + "&");
    {Connection: close};
}
```

```
</script>
</body>
</html>
```

h. Membuat Slider

Halaman HTML untuk proyek ini melibatkan pembuatan slider. Untuk membuat slider di HTML Kami menggunakan tag <input>. Tag <input> menentukan bidang tempat pengguna dapat memasukkan data. Ada berbagai macam jenis input. Untuk menentukan sliding, gunakan atribut “**type**” dengan “**range**” value. Di sliding, Kami juga perlu menentukan rentang minimum dan maksimum menggunakan atribut “**min**” dan “**max**”.

```
<input type="range" min="0" max="180"
class="slider" id="servoSlider"
onchange="servo(this.value)"/>
```

Selain itu juga perlu mendefinisikan atribut lain seperti:

- i. kelas untuk **menata** bilah geser
- ii. **id** untuk memperbarui posisi saat ini yang ditampilkan di halaman web
- iii. Dan terakhir, atribut **onchange** untuk memanggil fungsi servo untuk mengirim permintaan HTTP ke ESP32 saat slider bergerak.

i. Menambahkan JavaScript ke File HTML

Selanjutnya, Kami perlu menambahkan beberapa kode JavaScript ke file HTML Kami menggunakan tag <script> dan </script>. Cuplikan kode ini memperbarui halaman web dengan posisi sliding saat ini:

```
var slider = document.getElementById("servoSlider");
var servoP = document.getElementById("servoPos");
servoP.innerHTML = slider.value;
slider.oninput = function() { slider.value = this.value; servoP.innerHTML = this.value; }
```

Dan baris berikutnya membuat permintaan HTTP GET pada alamat IP ESP di jalur URL khusus ini /?value=[SLIDER_POSITION]&.

```
$.ajaxSetup ({ batas waktu : 1000 });fungsi servo ( pos ) {
```

```
$.dapatkan ( "?nilai=" + pos + "&" );
```

```
}
```

Misalnya, saat bilah geser berada di **0**, maka membuat permintaan HTTP GET di URL berikut:

```
http://192.168.1.135/?nilai= 0&
```

Dan ketika sliding berada pada **180** derajat, Kami akan memiliki sesuatu sebagai berikut:

```
http://192.168.1.135/?nilai= 180&
```

Dengan cara ini, ketika ESP32 menerima permintaan GET, ia dapat mengambil parameter nilai di URL dan memindahkan motor servo ke posisi yang tepat.

j. Kode

Sekarang, kita perlu memasukkan teks HTML sebelumnya ke dalam sketsa dan memutar servo sesuai dengannya. Sketsa berikutnya melakukan hal itu dengan tepat.

Salin kode berikut ke Arduino IDE Kami, tetapi jangan mengunggahnya dulu. Pertama, kita akan melihat sekilas cara kerjanya.

```
*****
Rui Santos
Complete project details at https://randomnerdtutorials.com
***** #include
<WiFi.h> #include
<Servo.h>
Servo myservo; // create servo object to control a servo
// twelve servo objects can be created on most boards

// GPIO the servo is attached to
static const int servoPin = 13;
// Replace with your network credentials
const char* ssid      = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
// Set web server port number to 80
WiFiServer server(80);
// Variable to store the HTTP request
String header;
// Decode HTTP GET value String
valueString = String(5);
```

```

int pos1 = 0;
int pos2 = 0;
// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)const
long timeoutTime = 2000;
void setup() {
    Serial.begin(115200);
        myservo.attach(servoPin); // attaches the servo on the servoPinto
the servo object
    // Connect to Wi-Fi network with SSID and password
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    // Print local IP address and start web server
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    server.begin();
}
void loop(){
    WiFiClient client = server.available(); // Listen for incoming clients
    if (client) { // If a new client connects,
        currentTime = millis();
        previousTime = currentTime;
        Serial.println("New Client."); // print a message out in
the serial port
        String currentLine = ""; // make a String to hold
incoming data from the client
        while (client.connected() && currentTime - previousTime <=
timeoutTime) { // loop while the client's connected
            currentTime = millis();
            if (client.available()) { // if there's bytes to read from
the client,
                char c = client.read(); // read a byte, then

```

```

Serial.write(c);           // print it out the serial monitor
header += c;
if (c == '\n') {          // if the byte is a newline character
    // if the current line is blank, you got two newline
characters in a row.
    // that's the end of the client HTTP request, so send a response:
    if (currentLine.length() == 0) {
        // HTTP headers always start with a response code (e.g.
HTTP/1.1 200 OK)
        // and a content-type so the client knows what's coming,
then a blank line:
        client.println("HTTP/1.1 200 OK");
        client.println("Content-type:text/html");
        client.println("Connection: close");
        client.println();
        // Display the HTML web page
        client.println("<!DOCTYPE html><html>");
        client.println("<head><meta name=\"viewport\""
content="width=device-width, initial-scale=1\"");
        client.println("<link rel=\"icon\" href=\"data:,\"");
        // CSS to style the on/off buttons
        // Feel free to change the background-color and font-size
attributes to fit your preferences
        client.println("<style>body { text-align: center; font-
family: \"Trebuchet MS\", Arial; margin-left:auto; margin-right:auto;}");
        client.println(".slider { width: 300px; }</style>");
        client.println("<script"
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
</script>");
        // Web Page
        client.println("</head><body><h1>ESP32 with Servo</h1>");
        client.println("<p>Position: <span"
id="servoPos"></span></p>");
        client.println("<input type=\"range\" min=\"0\""
max="180" class="slider" id="servoSlider"
onchange="servo(this.value)" value="" +valueString+ "/");
        client.println("<script>var slider ="
document.getElementById("servoSlider"));
        client.println("var servoP ="
document.getElementById("servoPos"); servoP.innerHTML = slider.value;" );

```

```

        client.println("slider.oninput = function() { slider.value =
this.value; servoP.innerHTML = this.value; }");

        client.println("$.ajaxSetup({timeout:1000}); function
servo(pos) { ");

            client.println("$.get(\"/?value=" + pos + "&\");
{Connection: close};}</script>");

            client.println("</body></html>");
//GET /?value=180& HTTP/1.1
if(header.indexOf("GET /?value=")>=0) {
    pos1 = header.indexOf('=');
    pos2 = header.indexOf('&');
    valueString = header.substring(pos1+1, pos2);
//Rotate the servo
myservo.write(valueString.toInt());
Serial.println(valueString);
}

// The HTTP response ends with another blank line
client.println();
// Break out of the while loop
break;
} else { // if you got a newline, then clear currentLine
currentLine = "";
}

} else if (c != '\r') { // if you got anything else but a
carriage return character,
currentLine += c;      // add it to the end of the currentLine
}
}
}

// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}

```

k. Cara Kerja Kode

Pertama, kami menyertakan perpustakaan Servo, dan membuat objek servo yang disebut myservo.

```
#include <Servo.h>
```

```
Servo myservo; // create servo object to control a servo
```

Kami juga membuat variabel untuk menyimpan nomor GPIO yang terhubung dengan servo. Pada kasus ini, **GPIO 13**.

```
const int servoPin = 13;
```

Jangan lupa bahwa untuk memodifikasi dua baris berikut untuk menyertakan kredensial jaringan Kami.

```
// Replace with your network credentials const char* ssid = "";
```

```
const char* password = "";
```

Kemudian, buat beberapa variabel yang akan digunakan untuk mengekstrak posisi slider dari permintaan HTTP.

```
// Decode HTTP GET value String valueString =
```

```
String(5);int pos1 = 0;
```

```
int pos2 = 0;
```

- **setup()**

Dalam `setup()`, Kami harus memasang servo ke GPIO yang terhubung dengannya, dengan `myservo.attach()`.

```
myservo.attach(servoPin); // attaches the servoon the servoPin to the servo object
```

- **loop()**

Bagian pertama dari `loop()` membuat server web dan mengirimkan teks HTML untuk menampilkan halaman web. Bagian kode dibawah ini mengambil nilai slider dari permintaan HTTP.

```
//GET /?value=180& HTTP/1.1 if ( header.indexOf ( "GET /?value=" )>= 0 ) {  
pos1 = header . indeksDari ( '=' ); pos2 = judul . indeksDari ( '&' ); nilaiString = header  
. substring ( pos1 + 1  
, pos2 );
```

Saat Kami memindahkan sliding, Kami membuat permintaan HTTP pada URL berikut, yang berisi posisi sliding antara tKami = dan &.

```
http://your-esp-ip-address/?value=[SLIDER_POSITION]&
```

Nilai posisi sliding disimpan di variabel valueString. Kemudian, atur servo ke posisi tertentu menggunakan myservo.write() dengan variabel valueString sebagai argumen. Variabel valueString adalah string, jadi kita perlu menggunakan metode TolInt() untuk mengubahnya menjadi bilangan bulat – tipe data yang diterima oleh metode write().

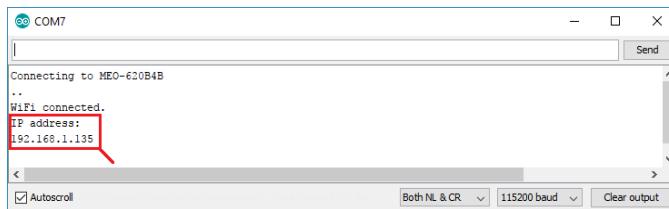
```
myservo.write(valueString.toInt());
```

I. Menguji Server Web

Sekarang kita dapat mengunggah kode ke ESP32, pastikan telah memilih papan dan port COM yang tepat. Juga jangan lupa untuk mengubah kode untuk memasukkan kredensial jaringan. Setelah mengupload kode, buka Serial Monitor dengan baud rate 115200.



Tekan tombol "Enable" ESP32 untuk memulai ulang board, dan salin alamat IP ESP32 yang muncul di Serial Monitor.



Buka browser, rekatkan alamat IP ESP, kemudian akan terlihat halaman web yang telah dibuat sebelumnya. Gerakkan slider untuk mengontrol motor servo.



Di Serial Monitor, Kami juga dapat melihat permintaan HTTP yang Kami kirim ke ESP32 saat Kami menggerakkan slider.

```
137
Client disconnected.

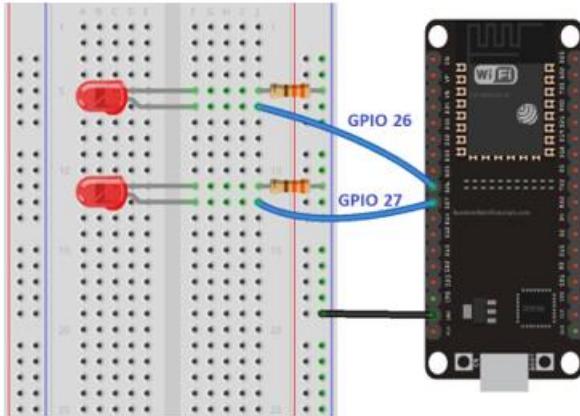
New Client
GET /?value=38s HTTP/1.1
Host: 192.168.1.135
Connection: keep-alive
Accept: /*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36
Referer: http://192.168.1.135/
Accept-Encoding: gzip, deflate
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7

28
Client disconnected.
```

Percobaan dengan server web bertujuan untuk melihat apakah program tersebut telah bekerja dengan benar atau belum.

9. Output WebServer

Pada output web server ditujukan untuk menciptakan server secara fungsional dengan menggunakan ESP32 dengan mengontrol output yaitu sebuah lampu LED menggunakan sistem pemrograman arduino IDE. Dengan webserver sendiri yang memiliki sifat mobile responsive serta dapat diakses pada perangkat apapun yang terhubung dengan jaringan wifi atau jaringan lokal yang pada browser.



a. Kode Server Web ESP32

Berikut merupakan kode untuk membuat web server ESP32. Salin kode berikut ke Arduino IDE, tetapi jangan mengunggahnya dulu karena diperlukan beberapa perubahan sebelum diunggah.

```
*****
Rui Santos
Complete project details at
https://randomnerdtutorials.com
*****
// Load Wi-Fi library#include
<WiFi.h>
// Replace with your network credentials const char* ssid =
"REPLACE_WITH_YOUR_SSID";
const char* password =
"REPLACE_WITH_YOUR_PASSWORD";
// Set web server port number to 80WiFiServer
server(80);

// Variable to store the HTTP requestString header;

// Auxiliar variables to store the currentoutput state
String output26State = "off",String
output27State = "off";

// Assign output variables to GPIO pinsconst int output26 = 26;
const int output27 = 27;
```

```

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds
(example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
    Serial.begin(115200);
    // Initialize the output variables as outputs
    pinMode(output26, OUTPUT);
    pinMode(output27, OUTPUT);
    // Set outputs to LOW
    digitalWrite(output26, LOW);
    digitalWrite(output27, LOW);

    // Connect to Wi-Fi network with SSID and password
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    // Print local IP address and start webserver
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
    server.begin();
}

void loop(){
    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,
        currentTime = millis();

```

```

    previousTime = currentTime;
    Serial.println("New Client."); // print a message out in the serial port
    String currentLine = ""; // make a String to hold incoming data from the client
    while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the client's connected
        currentTime = millis();
        if (client.available()) { //if there's bytes to read from the client,
            char c = client.read(); // read a byte, then
            Serial.write(c); // print it out the serial monitor
            header += c;
            if (c == '\n') { // if the byte is a newline character
                // if the current line is blank, yougot two newline characters in a row.
                // that's the end of the client HTTPRequest, so send a response:
                if (currentLine.length() == 0) {
                    // HTTP headers always start with a response code
                    // (e.g. HTTP/1.1 200 OK)
                    // and a content-type so theclient knows what's coming, then a blank line:
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-type:text/html");
                    client.println("Connection:close");
                    client.println();
                    // turns the GPIOs on and off
                    if (header.indexOf("GET /26/on") >= 0) {
                        Serial.println("GPIO 26 on");
                        output26State = "on";
                        digitalWrite(output26, HIGH);
                    } else if (header.indexOf("GET /26/off") >= 0) {
                        Serial.println("GPIO 26 off");
                    }
                }
            }
        }
    }
}

```

```

        output26State = "off";
        digitalWrite(output26, LOW);
    } else if (header.indexOf("GET
/27/on") >= 0) {
        Serial.println("GPIO 27 on");
        output27State = "on";
        digitalWrite(output27, HIGH);
    } else if (header.indexOf("GET
/27/off") >= 0) {
        Serial.println("GPIO 27 off");
        output27State = "off";
        digitalWrite(output27, LOW);
    }

    // Display the HTML web page
    client.println("<!DOCTYPE
html><html>");
    client.println("<head><meta
name=\"viewport\" content=\"width=device-width,initial-scale=1\">");
    client.println("<link rel=\"icon\" href=\"data:,\">");
    // CSS to style the on/off buttons
    // Feel free to change the background-color
and font-size attributes to fit your preferences
    client.println("<style>html     {   font-family:
Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
    client.println(".button   {   background-
color: #4CAF50; border: none; color: white; padding: 16px 40px;}");
    client.println("text-decoration: none; font-size:
30px; margin: 2px; cursor: pointer;}");
    client.println(".button2
{background-color: #555555;}</style></head>");

    // Web Page Heading
    client.println("<body><h1>ESP32
Web
Server</h1>");
```

```

        // Display current state, and
ON/OFF buttons for GPIO 26
        client.println("<p>GPIO 26 - State" + output26State +
"</p>");
        // If the output26State is off, it displays the ON button
        if (output26State=="off") {
            client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");
        } else {
            client.println("<p><a href=\"/26/off\"><button button2\">OFF</button></a></p>");
        }
    }

        // Display current state, and ON/OFF buttons for
GPIO 27
        client.println("<p>GPIO 27 - State" + output27State +
"</p>");
        // If the output27State is off, it displays the ON button
        if (output27State=="off") {
            client.println("<p><a href=\"/27/on\"><button class=\"button\">ON</button></a></p>");
        } else {
            client.println("<p><a href=\"/27/off\"><button button2\">OFF</button></a></p>");
        }
        client.println("</body></html>");

        // The HTTP response ends with another blank
line
        client.println();
        // Break out of the while loop
        break;
    } else { // if you got a newline, then clear currentLine
        currentLine = "";
    }
}

```

```

        } else if (c != '\r') { // if you got anything
else but a carriage return character,
        currentLine += c; // add it to the end
of the currentLine
    }
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}

```

b. Mengatur Kredensial Jaringan

Kita perlu mengubah baris berikut dengan kredensial jaringan Kami: SSID dan kata sandi. Kode dikomentari dengan baik di mana Kami harus membuat perubahan.

```

// Replace with your network credentials
const char* ssid =
"REPLACE_WITH_YOUR_SSID";
const char* password =
"REPLACE_WITH_YOUR_PASSWORD";

```

c. Mengunggah Kode

Sekarang, Kami dapat mengunggah kode dan server web akan langsung bekerja. Ikuti langkah selanjutnya untuk mengunggah kode ke ESP32:

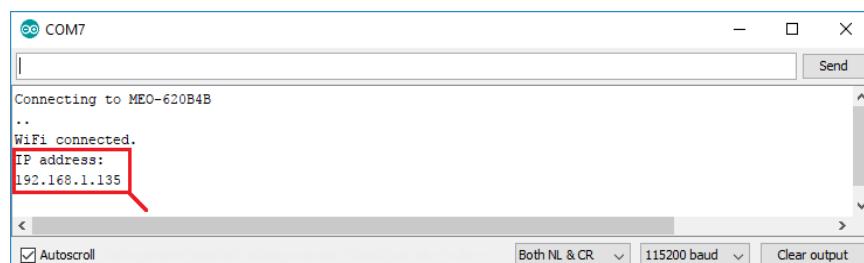
- i. Pasang papan ESP32 di komputer;
- ii. Di Arduino IDE pilih board Kami di **Tools > Board** (dalam kasus kami, kami menggunakan board ESP32 DEVKIT DOIT);

d. Menemukan Alamat IP ESP

Setelah mengupload kode, buka Serial Monitor dengan baud rate 115200.



Tekan tombol ESP32 EN (reset). ESP32 terhubung ke Wi-Fi, dan menampilkan alamat IP ESP di Serial Monitor. Salin alamat IP itu, karena akan diperlukan untuk mengakses server web ESP32.



e. Mengakses Server Web

Untuk mengakses server web, buka browser Kami, tempel alamat IP ESP32, dan Kami akan melihat halaman berikut. Misal pada contoh dibawah ini adalah **192.168.1.135**.



Jika terdapat Serial Monitor, maka dapat melihat apa yang terjadi di latar belakang. ESP menerima permintaan HTTP dari klien baru (browser yang digunakan).



f. Menguji Server Web

Sekarang kita dapat menguji apakah server web berfungsi dengan baik dengan cara klik tombol untuk mengontrol LED.



Pada saat yang sama, kita dapat melihat Serial Monitor untuk melihat apa yang terjadi di latar belakang. Misalnya, ketika mengklik tombol untuk memutar **GPIO 26** ON, ESP32 menerima permintaan pada URL **/26/on**.

```
New Client.  
GET /26/on HTTP/1.1  
Host: 192.168.1.135  
Connection: keep-alive  
Upgrade-Insecure-Requests: 1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.132 Safari/537.36  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8  
Referer: http://192.168.1.135/  
Accept-Encoding: gzip, deflate  
Accept-Language: pt-PT,pt;q=0.9,en-US;q=0.8,en;q=0.7  
  
GPIO 26 on  
Client disconnected.
```

Ketika ESP32 menerima permintaan itu, ternyata LED terpasang ke **GPIO 26** ON dan memperbarui statusnya di halaman web.



Tombol untuk **GPIO 27** bekerja dengan cara yang serupa.

g. Cara Kerja Kode

Pada bagian ini akan melihat lebih dekat pada kode untuk melihat cara kerjanya.

Hal pertama yang perlu Kami lakukan adalah memasukkan WiFi library.

```
#include <WiFi.h>
```

Seperti disebutkan sebelumnya, Kami perlu memasukkan ssid dan kata sandi Kami di baris berikut di dalam tKami kutip gKami.

```
const char* ssid = ""; const char*  
password = "";
```

Kemudian, Kami mengatur server web Kami ke port 80.

```
WiFiServer server(80);
```

Baris berikut membuat variabel untuk menyimpan header permintaan HTTP:

```
String header;
```

Selanjutnya, Kami membuat variabel tambahan untuk menyimpan status keluaran Kami saat ini. Jika Kami ingin menambahkan lebih banyak output dan menyimpan statusnya, Kami perlu membuat lebih banyak variabel.

```
String output26State = "off", String  
output27State = "off";
```

Kita juga perlu menetapkan GPIO untuk setiap output Kami. Di sini kami menggunakan **GPIO 26** dan **GPIO 27**. Kami dapat menggunakan GPIOs lain yang sesuai.

```
const int output26 = 26;  
const int output27 = 27;
```

● Setup()

Sekarang, mari kita masuk ke **setup()**. Pertama, kita memulai komunikasi serial pada baud rate 115200 untuk keperluan debugging.

```
Serial.begin(115200);
```

Kita juga menentukan GPIO sebagai OUTPUT dan menyetelnya ke LOW.

```

// Initialize the output variables as outputs
pinMode(output26, OUTPUT);
pinMode(output27, OUTPUT);

// Set outputs to LOW
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);

```

Baris berikut memulai koneksi Wi-Fi dengan WiFi.begin(ssid, kata sandi), tunggu koneksi berhasil dan cetak alamat IP ESP di Serial Monitor.

```

// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();

```

- **loop()**

Dalam `loop()` kami memprogram apa yang terjadi ketika klien baru membuat koneksi dengan server web.

ESP32 selalu merespon klien yang masuk dengan baris berikut:

```
WiFiClient client = server.available(); //Listen for incoming clients
```

Ketika permintaan diterima, selanjutnya menyimpan data yang masuk. Kemudian salin kode berikut ini.

```
if (client) { // If a new client connects,
```

```
Serial.println("New Client."); // print a
message out in the serial port
String currentLine = ""; // make a String to hold incoming data from the
client
while (client.connected()) { // loop while the client's connected
    if (client.available()) { // if there's bytes to read from the client,
        char c = client.read(); // read a byte,
then
        Serial.write(c); // print it out the
serial monitor
        header += c;
        if (c == '\n') { // if the byte is a newline character
            // if the current line is blank, you got two newline characters in a
row.
            // that's the end of the client HTTP request, so send a
response:
            if (currentLine.length() == 0) {
                // HTTP headers always start with a response code (e.g.
HTTP/1.1 200 OK)
                // and a content-type so the client knows what's coming,
then a blank line:
                client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println("Connection: close");
client.println();
// turns the GPIOs on and off
if (header.indexOf("GET /26/on") >= 0) {
    Serial.println("GPIO 26 on");
    output26State = "on";
    digitalWrite(output26, HIGH);
} else if (header.indexOf("GET /26/off") >= 0)
{
    Serial.println("GPIO 26 off");
}
```

```

        output26State = "off";
        digitalWrite(output26, LOW);
    } else if (header.indexOf("GET /27/on") >= 0)
    {
        Serial.println("GPIO 27 on");
        output27State = "on";
        digitalWrite(output27, HIGH);
    } else if (header.indexOf("GET /27/off") >= 0)
    {
        Serial.println("GPIO 27 off");
        output27State = "off";
        digitalWrite(output27, LOW);
    }
}

```

Misalnya, jika telah menekan tombol GPIO 26 ON, dan ESP32 menerima permintaan pada **URL /26/ON** (kita dapat melihat informasi tersebut pada header HTTP di Serial Monitor). Jadi, kita dapat memeriksa apakah header berisi ekspresi **GET /26/on**. Jika berisi, kami mengubah variabel `output26state` ke ON, dan ESP32 menyalaikan LED.

h. Menampilkan halaman web HTML

Hal berikutnya yang perlu dilakukan adalah membuat halaman web. ESP32 akan mengirimkan respons ke browser dengan beberapa kode HTML untuk membangun halaman web. Halaman web dikirim ke klien menggunakan ekspresi `klien.println()`.

Hal pertama yang harus kita kirim selalu baris berikut, yang menunjukkan bahwa kita mengirim HTML.

```
<!DOCTYPE HTML><html>
```

Kemudian, baris berikut membuat halaman web responsive di browser web apa pun.

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
```

Dan berikut ini digunakan untuk mencegah request pada favicon. - Kami tidak perlu khawatir tentang baris ini.

```
client.println("<link rel=\"icon\" href=\"data:;\">");
```

i. Menata Halaman Web

Selanjutnya, kita memiliki beberapa teks CSS untuk menata tombol dan tampilan halaman web. Kami memilih font Helvetica, menentukan konten yang akan ditampilkan sebagai blok dan disejajarkan di tengah.

```
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}</style>");
```

Kami menata tombol kami dengan warna #4CAF50, tanpa batas, teks dalam warna putih, dan dengan padding ini: 16px 40px. Kami juga mengatur dekorasi teks ke none, menentukan ukuran font, margin, dan kursor ke pointer.

```
client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;}");
```

```
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;});
```

Kami juga mendefinisikan gaya untuk tombol kedua, dengan semua properti tombol yang telah kami definisikan sebelumnya, tetapi dengan warna yang berbeda. Ini akan menjadi gaya untuk tombol off.

```
client.println(".button2 {background-color:#555555;}</style></head>");
```

j. **Mengatur Judul Pertama Halaman Web**

Di baris berikutnya Kami dapat mengatur judul pertama halaman web Kami. Di sini kami memiliki "**ESP32 Web Server**", tetapi Kami dapat mengubah teks ini menjadi apa pun yang Kami suka.

```
// Web Page Heading client.println("<h1>ESP32 Web Server</h1>");
```

k. **Menampilkan Tombol dan Status yang Sesuai**

Kemudian, Kami menulis paragraf untuk menampilkan **GPIO 26** kondisi saat ini. Seperti yang Kami lihat, kami menggunakan **output26State** variabel, sehingga status diperbarui secara instan ketika variabel ini berubah.

```
client.println("<p>GPIO 26 - State " + output26State + "</p>");
```

Kemudian, kami menampilkan tombol on atau off, tergantung pada kondisi GPIO saat ini. Jika keadaan GPIO saat ini mati, kami menunjukkan tombol ON, jika tidak, kami menampilkan tombol OFF.

```
if (output26State=="off") {
```

```

client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
} else {
    client.println("<p><a
href=\"/26/off\"><button
class=\"button
button2\">OFF</button></a></p>");
}

```

Gunakan prosedur yang sama untuk **GPIO 27**.

I. Menutup Koneksi

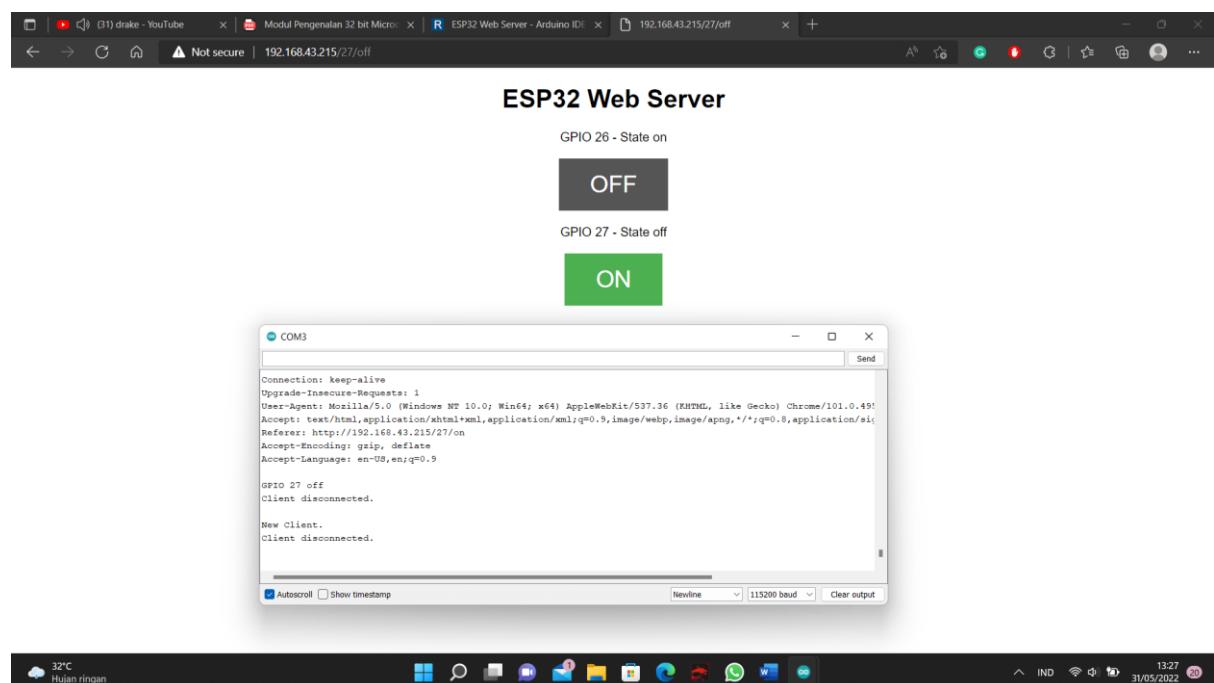
Akhirnya, ketika respons berakhir, kami menghapus header variabel, dan hentikan koneksi dengan klien dengan `client.stop()`.

```

// Clear the header variableheader = "";
// Close the connectionclient.stop();

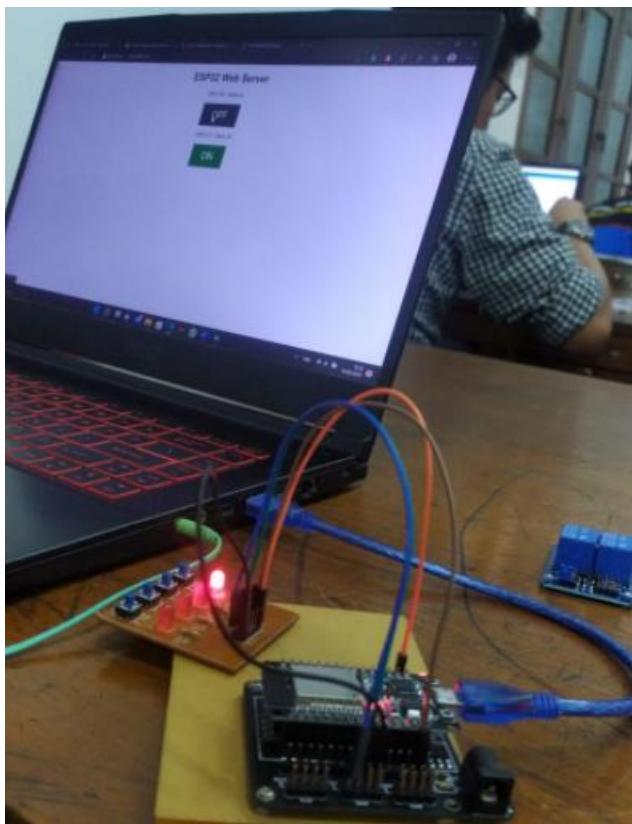
```

Contoh kasus :





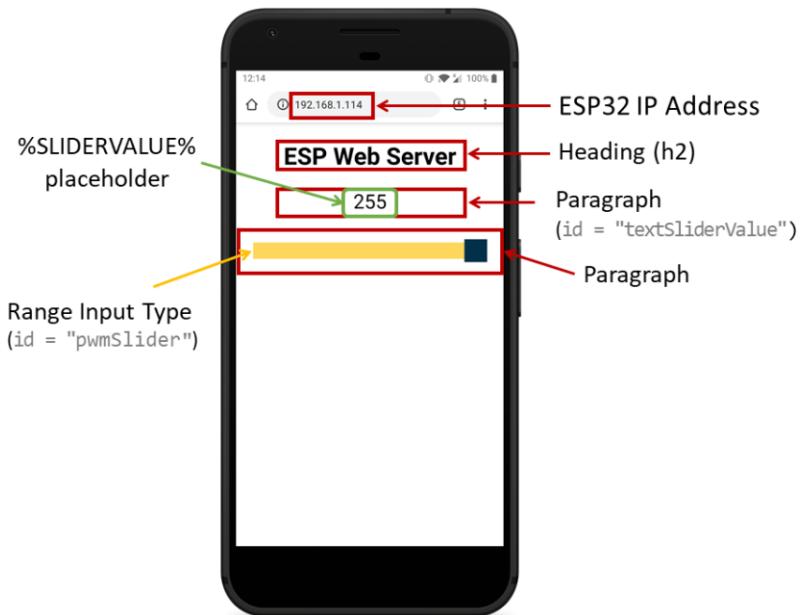
Saat 1 button off



10. WebServer dengan Slider

Webserver with Slider merupakan sistem pemrograman web server ESP32 menggunakan Arduino IDE, bertujuan untuk melakukan kontrol kecerahan pada lampu LED dengan cara menggeser. Variabel yang digunakan merupakan ESP32 yang nantinya terdapat suatu program yang dapat mengontrol sinyal, program tersebut dapat digunakan untuk melakukan kontrol pada sinyal PWM dan sistem lainnya yaitu mengubah kecerahan pada lampu LED. Selain penggunaannya pada mengubah kecerahan lampu LED, dapat juga digunakan untuk mengontrol motor servo dan lainnya.

- Building the Web Page



10.1 Arduino IDE

Kami akan memprogram papan ESP32 menggunakan Arduino IDE, jadi sebelum melanjutkan dengan tutorial ini, pastikan telah menginstal papan ESP32 di Arduino.

10.2 Async Web Server Libraries

Kami akan membangun server web menggunakan libraries berikut:

- [ESPAsyncWebServer](#)
- [AsyncTCP](#)

Library tersebut tidak tersedia untuk diinstal melalui Arduino Library Manager, kita perlu menyalin file library ke folder Library Instalasi Arduino. Atau di Arduino IDE dengan cara **Sketch > Include Library > Add .zip Library** dan pilih library yang baru saja Kami unduh.

10.3 Kode

Kode berikut mengontrol kecerahan LED bawaan ESP32 menggunakan slider di server web. Dengan kata lain, Kami dapat mengubah siklus tugas PWM dengan slider. Ini dapat berguna untuk mengontrol kecerahan LED atau mengontrol motor servo, misalnya salin kode ke Arduino IDE Kami. Masukkan kredensial jaringan Kami dan kode akan langsung berfungsi.

```
*****
Rui Santos
Complete      project      details      at
https://RandomNerdTutorials.com/esp32-web-server-slider-pwm/

Permission is hereby granted, free of charge, to any person
obtaining a copy
of this software and associated documentation files.

The above copyright notice and this permission notice shall
be included in all
copies or substantial portions of the software.

*****/
```

```

// Import required libraries#include <WiFi.h>
#include <AsyncTCP.h> #include
<ESPAAsyncWebServer.h>

// Replace with your network credentials const char* ssid =
"REPLACE_WITH_YOUR_SSID";
const char* password =
"REPLACE_WITH_YOUR_PASSWORD";

const int output = 2; String sliderValue =
"0";

// setting PWM propertiesconst int freq
= 5000; const int ledChannel = 0;const
int resolution = 8;

const char* PARAM_INPUT = "value";

// Create AsyncWebServer object on port 80AsyncWebServer
server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
    <meta name="viewport"
content="width=device-width, initial-scale=1">
    <title>ESP Web Server</title>
    <style>
        html {font-family: Arial; display:inline-block; text-align:
center;}
        h2 {font-size: 2.3rem;}
        p {font-size: 1.9rem;}
        body {max-width: 400px; margin:0px auto; padding-bottom:
25px;}
        .slider { -webkit-appearance: none; margin: 14px; width:
360px; height: 25px; background: #FFD65C;
outline: none; -webkit-transition:
.2s; transition: opacity .2s;}
```

```

        .slider::-webkit-slider-thumb
{-webkit-appearance: none; appearance: none; width: 35px; height: 35px;
background: #003249; cursor: pointer;}
        .slider::-moz-range-thumb { width: 35px; height: 35px;
background: #003249; cursor: pointer; }
    </style>
</head>
<body>
    <h2>ESP Web Server</h2>
    <p><span id="textSliderValue">%SLIDERVALUE%</span></p>
        <p><input type="range" onchange="updateSliderPWM(this)" id="pwmSlider" min="0" max="255" value="%SLIDERVALUE%" step="1" class="slider"></p>
    <script>
        function updateSliderPWM(element) {
            var sliderValue = document.getElementById("pwmSlider").value;
            document.getElementById("textSliderValue").innerHTML = sliderValue;
            console.log(sliderValue);
            var xhr = new XMLHttpRequest();
            xhr.open("GET",
                    "/slider?value=" + sliderValue, true);
            xhr.send();
        }
    </script>
</body>
</html>
)rawliteral";

// Replaces placeholder with button sectionin your web page
String processor(const String& var){
    //Serial.println(var);
    if (var == "SLIDERVALUE"){
        return sliderValue;
    }
    return String();
}

```

```

}

void setup(){
    // Serial port for debugging purposes
    Serial.begin(115200);

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

    // attach the channel to the GPIO to becontrolled
    ledcAttachPin(output, ledChannel);

        ledcWrite(ledChannel,
sliderValue.toInt());

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi..");
    }

    // Print ESP Local IP Address
    Serial.println(WiFi.localIP());

    // Route for root / web page
    server.on("/", HTTP_GET,
[])(AsyncWebServerRequest *request){
    request->send_P(200, "text/html",index_html,
processor);
});

        //      Send      a      GET      request      to
<ESP_IP>/slider?value=<inputMessage>
    server.on("/slider", HTTP_GET, [])
(AsyncWebServerRequest *request) {
    String inputMessage;
        //      GET      input1      value      on
<ESP_IP>/slider?value=<inputMessage>
    if (request->hasParam(PARAM_INPUT)) {
        inputMessage =
request->getParam(PARAM_INPUT)->value();
}
}

```

```

        sliderValue = inputMessage;
        ledcWrite(ledChannel,
sliderValue.toInt());
    }
    else {
        inputMessage = "No message sent";
    }
    Serial.println(inputMessage);
    request->send(200, "text/plain", "OK");
}

// Start server
server.begin();
}

void loop() {
}

```

10.4 Cara Kerja Kode

- **Mengimpor perpustakaan**

Pertama, impor perpustakaan yang diperlukan. `WiFi`, `ESPAsyncWebServer` dan `ESPAsyncTCP` diperlukan untuk membangun server web.

```
#include <WiFi.h> #include
<AsyncTCP.h>
#include <ESPAsyncWebServer.h>
```

- **Mengatur kredensial jaringan Kami**

Masukkan kredensial jaringan Kami di variabel berikut, sehingga ESP32 dapat terhubung ke jaringan lokal Kami.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const           char*           password      =
"REPLACE_WITH_YOUR_PASSWORD";
```

- **Definisi variabel**

Digunakan untuk mengontrol kecerahan LED bawaan ESP32. LED bawaan sesuai dengan `GPIO 2`. Simpan GPIO yang ingin dikontrol output variabel. Variabel `sliderValue` akan menahan nilai slider. Pada awalnya, itudiatur ke nol.

```
String sliderValue = "0";
```

- **Setel Properti PWM**

Baris berikut mendefinisikan properti PWM untuk mengontrol LED.

```
// setting PWM propertiesconst int freq  
= 5000; const int ledChannel = 0;  
  
const int resolution = 8;
```

Pada kasus kali ini akan digunakan resolusi 8-bit, yang berarti dapat mengontrol kecerahan LED menggunakan nilai dari 0 hingga 255.

- **Input Parameters**

Variabel `PARAM_INPUT` akan digunakan untuk "mencari" nilai slider pada permintaan yang diterima oleh ESP32 saat slider dipindahkan. (Ingat: ESP32 akan menerima permintaan seperti ini: `GET/slider?value=SLIDERVALUE`)

```
const char* PARAM_INPUT = "value";
```

Kode tersebut akan mencari `value` pada URL dan mendapatkan nilai yang diberikan padanya.

- **Membuat Halaman Web**

Halaman web untuk proyek ini cukup sederhana yaitu berisi satu judul, satu paragraf dan satu input dari rentang jenis. Semua teks HTML dengan gaya yang disertakan disimpan di variabel `index_html`.

Tag `<meta>` berikut membuat halaman web Kami responsif di browser apapun.

```
<meta name="viewport" content="width=device-width,initial-scale=1">
```

Di antara tag `<title> </title>` ada judul server web yang dibuat. Judul adalah teks yang muncul di tab browser web.

- **Styles**

Di antara tag `<style></style>`, kita menambahkan beberapa CSS untuk menata halaman web.

```
<style>
    html {font-family: Arial; display: inline-block;text-align: center;}
    h2 {font-size: 2.3rem;}
    p {font-size: 1.9rem;}
    body {max-width: 400px; margin:0px auto;padding-bottom: 25px;}
    .slider { -webkit-appearance: none; margin: 14px; width: 360px; height: 25px;
background: #FFD65C;
outline: none; -webkit-transition: .2s;transition: opacity .2s;}
    .slider::-webkit-slider-thumb
{-webkit-appearance: none; appearance: none; width: 35px; height: 35px;
background: #003249; cursor: pointer;}
    .slider::-moz-range-thumb { width: 35px; height:35px; background: #003249;
cursor: pointer; }

</style>
```

Pada dasarnya, kami mengatur halaman HTML untuk menampilkan teks dengan font Arial di blok tanpa margin, dan rata di tengah.

```
html {font-family: Arial; display: inline-block; text-align: center;}
```

Baris berikut mengatur ukuran font untuk heading (h2) dan paragraf (p).

```
h2 {font-size: 2.3rem;}p {font-size:
1.9rem;}
```

Setel properti body HTML.

```
body { max-width: 400px; margin: 0px auto;  
padding-bottom: 25px;}
```

Baris berikut menyesuaikan slider:

```
.slider { -webkit-appearance: none; margin: 14px; width: 360px; height: 25px;  
background: #FFD65C;  
outline: none; -webkit-transition: .2s; transition: opacity .2s;}  
.slider::-webkit-slider-thumb {-webkit-appearance:none; appearance: none; width:  
35px; height: 35px; background: #003249; cursor: pointer;}  
.slider::-moz-range-thumb { width: 35px; height: 35px; background: #003249;  
cursor: pointer; }
```

● HTML Body

Di dalam tag `<body></body>` adalah tempat kita menambahkan konten halaman web.

Tag `<h2></h2>` menambahkan heading ke halaman web. Dalam hal ini, teks "ESP Web Server", tetapi Kami dapat menambahkan teks lainnya.

```
<h2>ESP Web Server</h2>
```

Paragraf pertama akan berisi nilai slider saat ini. Tag HTML tertentu memiliki id `textSliderValue` yang ditetapkan padanya, sehingga kita dapat merujuknya nanti.

```
<p><span id="textSliderValue">%SLIDERVALUE%</span></p>
```

`%SLIDERVALUE%` adalah pengganti untuk nilai slider. Ini akan digantikan oleh ESP32 dengan nilai aktual saat mengirimkannya ke browser. Ini berguna untuk menunjukkan nilai saat ini saat Kami mengakses browser untuk pertama kalinya.

● Membuat Slider

Untuk membuat slider di HTML digunakan tag `<input>`. Tag `<input>` menentukan bidang tempat pengguna dapat memasukkan data. Ada berbagai macam jenis input. Untuk menentukan slider, gunakan atribut `"type"` dengan nilai `"range"`. Dalam slider, Kami juga perlu menentukan rentang

minimum dan maksimum menggunakan atribut “**min**” dan “**max**” (dalam hal ini, Odan255, masing-masing).

```
<p><input type="range"
onchange="updateSliderPWM(this)" id="pwmSlider" min="0" max="255"
value="%SLIDERVALUE%" step="1" class="slider"></p>
```

Kasus ini juga perlu mendefinisikan atribut lain seperti:

1. Atribut **step** menentukan interval antara angka yang valid . Dalam kasus kami, ini diatur ke 1;
2. **Class** untuk mengatur gaya slider (class="slider");
3. **id** untuk memperbarui posisi saat ini yang ditampilkan di halaman web;
4. Atribut **onchange** untuk memanggil fungsi (updateSliderPWM(this)) untuk mengirim permintaan HTTP ke ESP32 saat slider bergerak.

● Menambahkan JavaScript ke File HTML

Selanjutnya, perlu menambahkan beberapa kode JavaScript ke file HTML dengan menggunakan tag `<script>` dan `</script>`. Kita perlu menambahkan fungsi `updateSliderPWM()` yang akan membuat permintaan ke ESP32 dengan nilai slider saat ini.

```
<script>
function updateSliderPWM(element) {
    var sliderValue =
document.getElementById("pwmSlider").value;

document.getElementById("textSliderValue").innerHTML =
sliderValue;
console.log(sliderValue);
var xhr = new XMLHttpRequest();
xhr.open("GET", "/slider?value="+sliderValue,true);
xhr.send();
}

</script>
```

Baris berikutnya ini mendapatkan nilai slider saat ini dengan id-nya dan menyimpannya di variabel `sliderValue` JavaScript. Sebelumnya, kami telah menetapkan id slider ke `pwmSlider`. Jadi, kita mendapatkannya sebagai berikut:

```
var sliderValue =  
document.getElementById("pwmSlider").value;
```

Setelah itu, kita atur label slider (yang id-nya adalah `teksSliderValue`) ke nilai yang disimpan pada variabel `sliderValue`.

Terakhir, buat [HTTP GET request](#).

```
var xhr = new XMLHttpRequest();  
xhr.open("GET", "/slider?value="+sliderValue, true);xhr.send();
```

Misalnya, saat slider berada di **0**, kita membuat request HTTP GET di URL berikut:

`http://ESP-IP-ADDRESS/slider?value=0`

Dan ketika nilai slider adalah 200, kita akan memiliki request di URL berikut:

`http://ESP-IP-ADDRESS/slider?value=200`

Dengan cara ini, ketika ESP32 menerima permintaan GET, ESP32 dapat mengambil parameter nilai di URL dan mengontrol sinyal PWM yang sesuai seperti yang akan kita lihat di bagian selanjutnya.

● Prosesor

Sekarang, kita perlu membuat `prosesor()` function, yang akan menggantikan placeholder dalam teks HTML kita dengan nilai slider saat ini saat Kami mengaksesnya untuk pertama kali di browser.

```
// Replaces placeholder with button section in your web page  
String processor(const String& var){  
    //Serial.println(var);  
    if (var == "SLIDERVALUE"){  
        return sliderValue;  
    }  
    return String();  
}
```

Saat halaman web diminta, kami memeriksa apakah HTML memiliki placeholder. Jika menemukan `%SLIDERVALUE%` placeholder, kami mengembalikan nilai yang disimpan di variabel `sliderValue`.

- **Setup()**

Dalam `setup()`,inisialisasi Serial Monitor untuk keperluan debugging.

```
Serial.begin(115200);
```

Konfigurasikan properti PWM LED yang ditentukan sebelumnya.

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

Lampirkan saluran ke GPIO yang ingin Kami kontrol.

```
ledcAttachPin(output, ledChannel);
```

Atur siklus tugas sinyal PWM ke nilai yang disimpan di `sliderValue` (ketika ESP32 dimulai, itu diatur ke 0).

```
ledcWrite(ledChannel, sliderValue.toInt());
```

Hubungkan ke jaringan lokal Kami dan cetak alamat IP ESP32.

```
// Connect to Wi-Fi WiFi.begin(ssid,  
password);  
while (WiFi.status() != WL_CONNECTED) {  
    delay(1000);  
    Serial.println("Connecting to WiFi..");  
}  
  
// Print ESP Local IP Address  
Serial.println(WiFi.localIP());
```

- **Handle Requests**

Terakhir, tambahkan baris kode berikutnya untuk menangani server web.

```
// Route for root / web page  
server.on("/", HTTP_GET, [](AsyncWebServerRequest  
*request){
```

```

        request->send_P(200, "text/html", index_html,
processor);
});

// Send a GET request to
<ESP_IP>/slider?value=<inputMessage>
server.on("/slider", HTTP_GET, []]
(AsyncWebServerRequest *request) {
    String inputMessage;
    // GET input1 value on
<ESP_IP>/slider?value=<inputMessage>
    if (request->hasParam(PARAM_INPUT)) {
        inputMessage = request->getParam(PARAM_INPUT)->value();
        sliderValue = inputMessage;
        ledcWrite(ledChannel, sliderValue.toInt());
    }
    else {
        inputMessage = "No message sent";
    }
    Serial.println(inputMessage);
    request->send(200, "text/plain", "OK");
});

```

Saat membuat permintaan pada URL root, harus mengirim teks HTML yang disimpan di variabel `index_html`. Kita juga harus melewati `prosesor` function, yang akan menggantikan semua placeholder dengan nilai yang tepat.

```

// Route for root / web page
server.on("/", HTTP_GET, []](AsyncWebServerRequest
*request){
    request->send_P(200, "text/html", index_html,processor);
});

```

Kami membutuhkan penangan lain yang akan menyimpan nilai slider saat ini dan mengatur kecerahan LED yang sesuai.

```

server.on("/slider", HTTP_GET, []]
(AsyncWebServerRequest *request) {

```

```

String inputMessage;
//      GET      input1      value      on
<ESP_IP>/slider?value=<inputMessage>
if (request->hasParam(PARAM_INPUT)) {
    inputMessage = request->getParam(PARAM_INPUT)->value();
    sliderValue = inputMessage;
    ledcWrite(ledChannel, sliderValue.toInt());
}
else {
    inputMessage = "No message sent";
}
Serial.println(inputMessage);
request->send(200, "text/plain", "OK");
});

```

Pada dasarnya, kami mendapatkan nilai slider pada baris berikut:

```

if (request->hasParam(PARAM_INPUT)) {
    inputMessage = request->getParam(PARAM_INPUT)->value();
    sliderValue = inputMessage;
}

```

Kemudian, perbarui kecerahan LED (siklus tugas PWM) menggunakan `ledcWrite()` fungsi yang menerima sebagai argumen saluran yang ingin Kami kontrol dan nilainya.

```
ledcWrite(ledChannel, sliderValue.toInt());
```

Terakhir, mulai server.

```
server.begin();
```

Karena ini adalah server web asinkron, kami tidak perlu menulis apa pun `dilingkaran()`.

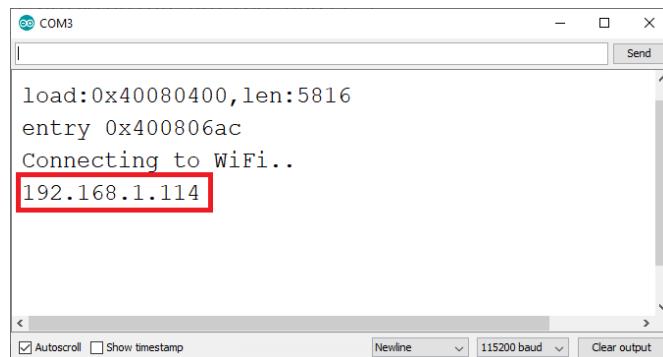
```
void loop(){
```

```
}
```

Kurang lebih begitulah cara kerja kode.

● Unggah Kode

Setelah mengunggah, buka Serial Monitor dengan baud rate 115200. Tekan tombol reset ESP32. Alamat IP ESP32 harus dicetak di monitor serial.



```
load:0x40080400,len:5816
entry 0x400806ac
Connecting to WiFi...
192.168.1.114
```

The screenshot shows the Serial Monitor interface with the title bar 'COM3'. The main window displays the ESP32's boot process, including 'load' and 'entry' messages, followed by 'Connecting to WiFi...' and the IP address '192.168.1.114'. The IP address is highlighted with a red rectangular box. At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', and dropdown menus for 'Newline', '115200 baud', and 'Clear output'.



Penjelasan singkat :

- ESP32 melakukan hosting web server yang menampilkan halaman web dengan penggeser yang nantinya berguna untuk mengatur tingkat kecerahan cahaya.
- Dengan melakukan perpindahan terhadap penggeser, permintaan HTTP diajukan kepada ESP32 dengan menggunakan nilai penggeser yang baru.
- Permintaan pada HTTP nantinya akan berbentuk format sebagai berikut: "GET/slайдер?value=SLIDERVALUE", where value merupakan rentang angka antara 0 dan 255.
- Sehingga setelah melakukan permintaan pada HTTP, ESP32 sudah mendapatkan nilai. - ESP32 menyesuaikan siklus tugas PWM sesuai dengan nilai slider.
- Ini berguna untuk mengontrol kecerahan LED (seperti yang akan kita lakukan dalam contoh ini), motor servo, pengaturan nilai ambang batas, atau aplikasi lain.

- Nantinya dapat digunakan untuk melakukan kontrol kecerahan pada lampu LED, motor servo,pengaturan nilai batas, serta aplikasi lainnya.

Contoh kasus slider :

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** modul_1_no_11 | Arduino 1.8.18
- Menu Bar:** File Edit Sketch Tool Help
- Code Area:** The main code area contains C++ code for an ESP32 web server. It includes includes for WiFi.h, AsyncTCP.h, and ESPAsyncTCP.h, as well as a specific header for ESP32. It defines network credentials (ssid "Redmi" and password "qwerty123"). A variable output is set to 2. The code then defines an HTML string for index.html with a title "ESP Pushbutton Web Server", a viewport meta tag, and a CSS style block for the body.
- Status Bar:** Shows system information: ESP32 Dev Module, Disabled, Default SPIFFS (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), OIO, 80MHz, 4MB (2Mb), 115200, Core 1, Core 1, None on COM3. It also shows a temperature of 26°C, a battery level of Berawan, and a build date of 01/06/2022.



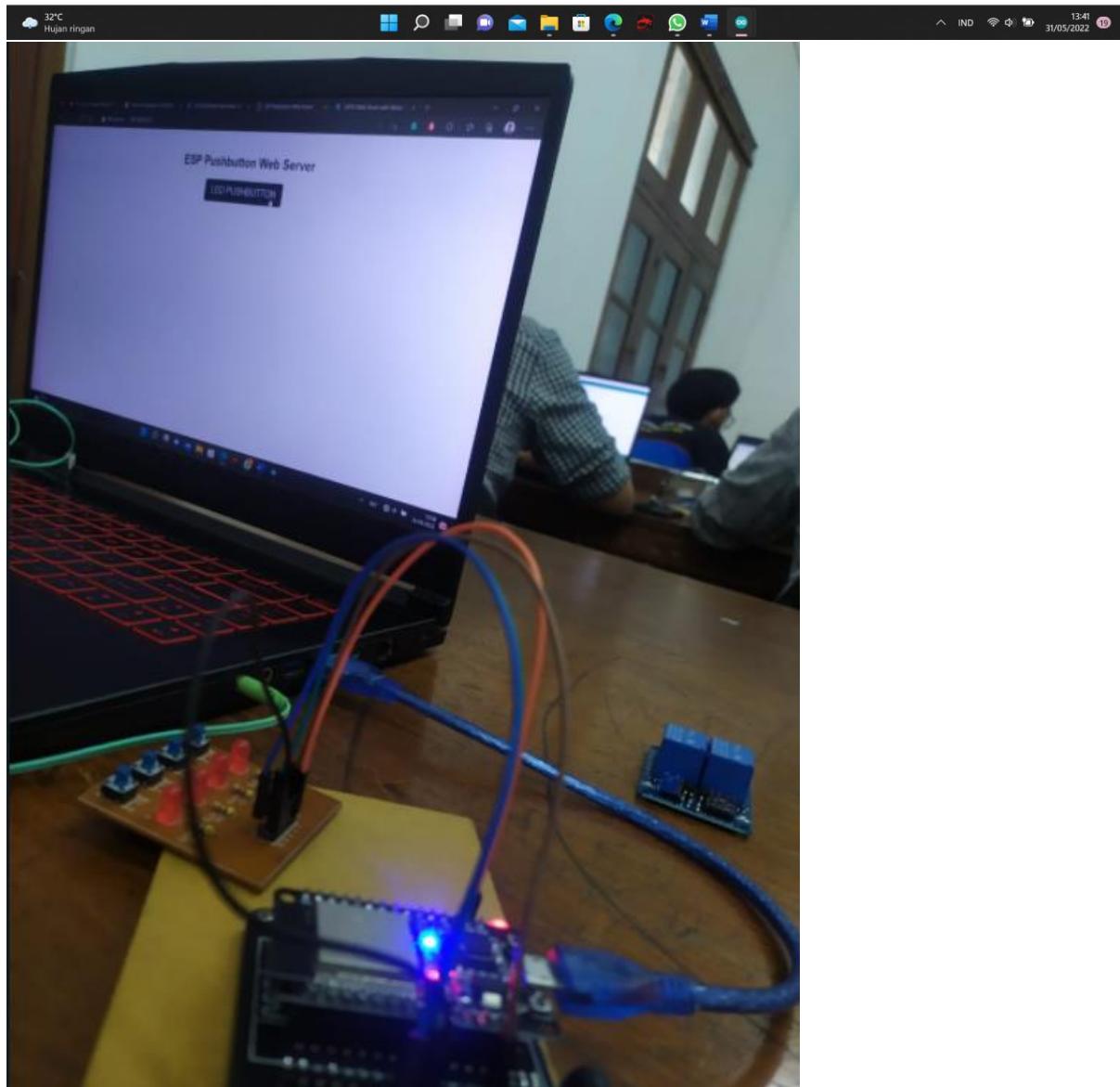
ESP Web Server

164



```
rst:0x1 (POWERON_RESET),boot:0x12 (SPI_FAST_FLASH_BOOT)
configip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13516
load:0x40080400,len:3604
entry 0x40080400
Connecting to WiFi...
192.168.43.215
176
50
25
0
164
```

Autoscroll Show timestamp Newline



11. Mempelajari Momentary Switch WebServer

- a. ESP32 atau ESP8266 host server web yang dapat Kami akses untuk mengontrol output;
- b. Status default output adalah LOW, tetapi Kami dapat mengubahnya tergantung pada aplikasi proyek Kami;
- c. Ada tombol yang berfungsi seperti saklar sesaat:
 - i. jika kita menekan tombol, output berubah statusnya menjadi TINGGI selama Kami terus menahan tombol;
 - ii. setelah tombol dilepaskan, status output kembali ke LOW.

Momentary Switch Web Server dapat digunakan untuk membuat server dengan button sehingga momentary dapat melakukan kontrol terhadap output ESP32 atau ESP8266. Jika diambil dengan asumsi sebuah lampu LED yang ditekan lama pada jarak kendali jauh maka lampu tersebut akan menyala terang, namun jika tombol kendali dilepaskan maka akan kembali redup. Namun nantinya momentary dapat mengontrol aspek lainnya.

11.1 Kode

Salin kode berikut ke Arduino IDE

```
*****
Rui Santos
Complete project details at
https://RandomNerdTutorials.com/esp32-esp8266-web-server-outputs-
momentary-switch/
```

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

```
*****/
```

```
#ifdef ESP32
    #include <WiFi.h>
    #include <AsyncTCP.h>#else
    #include <ESP8266WiFi.h>
    #include <ESPAsyncTCP.h>#endif
#include <ESPAsyncWebServer.h>

// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const           char*           password      =
"REPLACE_WITH_YOUR_PASSWORD";

const int output = 2;
```

```
// HTML web page
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
  <head>
    <title>ESP Pushbutton Web Server</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
      body { font-family: Arial; text-align:center; margin:0px auto; padding-top: 30px;}
      .button {
        padding: 10px 20px;
        font-size: 24px;
        text-align: center;
        outline: none;
        color: #fff;
        background-color: #2f4468;
        border: none;
        border-radius: 5px;
        box-shadow: 0 6px #999;
        cursor: pointer;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
        -webkit-tap-highlight-color: rgba(0,0,0,0);
      }
      .button:hover {background-color: #1f2e45}
      .button:active {
        background-color: #1f2e45;
        box-shadow: 0 4px #666;
        transform: translateY(2px);
      }
    </style>
  </head>
  <body>
    <h1>ESP Pushbutton Web Server</h1>
    <button class="button" onmousedown="toggleCheckbox('on');" ontouchstart="toggleCheckbox('on');">

```

```

onmouseup="toggleCheckbox('off');"
ontouchend="toggleCheckbox('off');" >LED
PUSHBUTTON</button>
<script>
function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/" + x, true);
    xhr.send();
}
</script>
</body>
</html>)rawliteral";

void notFound(AsyncWebServerRequest *request) {
    request->send(404, "text/plain", "Not found");
}

AsyncWebServer server(80);void setup() {
    Serial.begin(115200);
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    if (WiFi.waitForConnectResult() != WL_CONNECTED)
    {
        Serial.println("WiFi Failed!");
        return;
    }
    Serial.println();
    Serial.print("ESP IP Address: http://");
    Serial.println(WiFi.localIP());

    pinMode(output, OUTPUT);
    digitalWrite(output, LOW);

    // Send web page to client
    server.on("/", HTTP_GET, [](AsyncWebServerRequest
    *request){
        request->send_P(200, "text/html", index_html);
    });
}

// Receive an HTTP GET request

```

```

        server.on("/on", HTTP_GET, [
(AsyncWebServerRequest *request) {
    digitalWrite(output, HIGH);
    request->send(200, "text/plain", "ok");
});

// Receive an HTTP GET request
server.on("/off", HTTP_GET, [
(AsyncWebServerRequest *request) {
    digitalWrite(output, LOW);
    request->send(200, "text/plain", "ok");
});

server.onNotFound(notFound);
server.begin();
}

void loop() {
}

```

11.2 Kredensial Jaringan

Masukkan kredensial jaringan pada baris berikut:

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const           char*           password      =
"REPLACE_WITH_YOUR_PASSWORD";

```

11.3 Tombol Pengalihan Sesaat

Baris berikut membuat tombol saklar sesaat.

```

<button class="button" onmousedown="toggleCheckbox('on');"
ontouc

```

11.4 HTTP GET Permintaan untuk Mengubah Status Tombol (JavaScript)

Fungsi `toggleCheckbox()`, membuat permintaan HTTP ke ESP32 baik di `/on` atau `/off` URL:

```

function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/" + x, true);
    xhr.send();
}

```

11.5 Handle Request

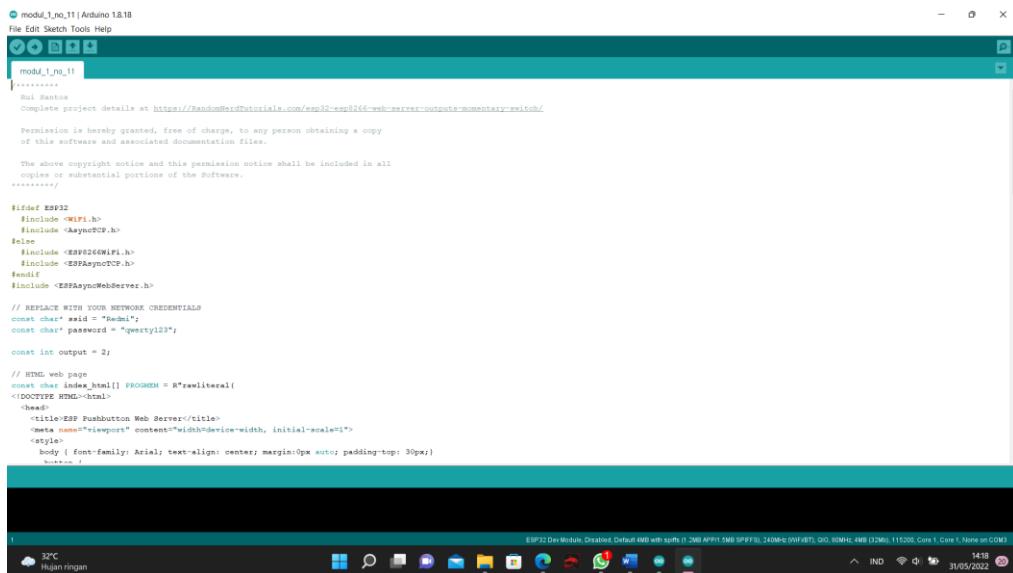
Ketika permintaan diterima di `/on` URL, aktifkan GPIO (HIGH) seperti yang ditunjukkan di bawah ini:

```
server.on("/on", HTTP_GET, [] (AsyncWebServerRequest *request) {
    digitalWrite(output, HIGH);
    request->send(200, "text/plain", "ok");
});
```

Saat permintaan diterima di `/off` URL, kami mematikan GPIO (LOW):

```
server.on("/off", HTTP_GET, [] (AsyncWebServerRequest *request) {
    digitalWrite(output, LOW);
    request->send(200, "text/plain", "ok");
});
```

Contoh kasus :



The screenshot shows the Arduino IDE interface with the file `modul_1_no_11.ino` open. The code implements a pushbutton control system using an ESP32 module connected to an Arduino Uno. It defines two pins for the pushbutton and an LED, initializes them, and sets up an interrupt for the pushbutton. The main loop checks for button presses and toggles the state of the LED. The code also includes a function to handle HTTP requests for turning the LED on or off.

```
#include "ESP32.h"
#include "WiFi.h"
#include "AsyncWebServer.h"
#include "ESPAsyncWebServer.h"

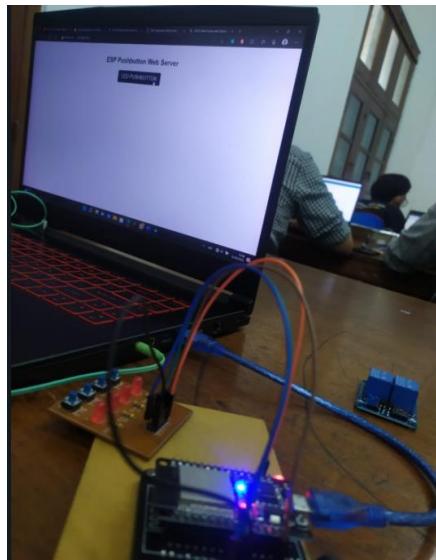
// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "Seamless";
const char* password = "qweerty123";

const int output = 2;

// HTML web page
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
<title>ESP Pushbutton Web Server</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body { font-family: Arial; text-align: center; margin:0px auto; padding-top: 30px; }
</style>
</head>
<body>
<h1>ESP Pushbutton Web Server</h1>
<form action="/" method="post">
<input type="submit" value="ON"/>
<input type="submit" value="OFF"/>
</form>
</body>
</html>
)rawliteral";

void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi!");
  pinMode(output, OUTPUT);
  digitalWrite(output, HIGH);
}

void loop() {
  if (WiFi.status() == WL_CONNECTED) {
    AsyncWebServer server(80);
    server.on("/", HTTP_GET, []() {
      server.send(index_html);
    });
    server.on("/on", HTTP_GET, []() {
      digitalWrite(output, HIGH);
      server.send(index_html);
    });
    server.on("/off", HTTP_GET, []() {
      digitalWrite(output, LOW);
      server.send(index_html);
    });
    server.begin();
  }
}
```



12. Mempelajari Physical Button WebServer

Physical Button Webserver yaitu merupakan mengontrol output pada ESP32 atau ESP8266, dengan menggunakan web server dan tombol secara langsung. Sehingga output yang dihasilkan pada laman web yang terbaru dapat terdeteksi nantinya apakah terdapat perubahan yang dilakukan pada web server atau tombol secara langsung.

Dapat dijelaskan secara lebih ringkas tahapannya seperti :

- ESP32 atau ESP8266 melakukan hosting web server yang memungkinkan untuk user untuk mengontrol status secara keluaran.
- Status yang sudah didapatkan saat ini ditampilkan pada web server.
- ESP nantinya juga terhubung ke tombol secara langsung yang dapat melakukan kontrol pada output yang sama.
- Sehingga nantinya jika akan mengubah status output menggunakan tombol secara langsung, status saat ini juga diperbarui pada web server.

ESP Web Server Code

```
*****
  Rui Santos
  Complete project details at
https://RandomNerdTutorials.com/esp32-esp8266-web-server-physical-button/

  The above copyright notice and this permission notice
shall be included in all
  copies or substantial portions of the Software.
*****
```

```
// Import required libraries
#ifndef ESP32
  #include <WiFi.h>
  #include <AsyncTCP.h>
#else
  #include <ESP8266WiFi.h>
  #include <ESPAsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>
```

```
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

const char* PARAM_INPUT_1 = "state";

const int output = 2;
const int buttonPin = 4;

// Variables will change:
int ledState = LOW; // the current state of the
output pin
int buttonState; // the current reading from
the input pin
int lastButtonState = LOW; // the previous reading
from the input pin

// the following variables are unsigned longs because
the time, measured in
// milliseconds, will quickly become a bigger number
than can be stored in an int.
unsigned long lastDebounceTime = 0; // the last time
the output pin was toggled
unsigned long debounceDelay = 50; // the debounce
time; increase if the output flickers

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
    <title>ESP Web Server</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1">
    <style>
        html {font-family: Arial; display: inline-block;
text-align: center;}
        h2 {font-size: 3.0rem;}
        p {font-size: 3.0rem;}
        body {max-width: 600px; margin: 0px auto;
padding-bottom: 25px;}
        .switch {position: relative; display: inline-block;
width: 120px; height: 68px}
        .switch input {display: none}
)rawliteral";
```

```

        .slider {position: absolute; top: 0; left: 0; right: 0; bottom: 0; background-color: #ccc; border-radius: 34px}
            .slider:before {position: absolute; content: ""; height: 52px; width: 52px; left: 8px; bottom: 8px; background-color: #fff; -webkit-transition: .4s; transition: .4s; border-radius: 68px}
                input:checked+.slider {background-color: #2196F3}
                input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px)}
            </style>
        </head>
    <body>
        <h2>ESP Web Server</h2>
        %BUTTONPLACEHOLDER%
<script>function toggleCheckbox(element) {
    var xhr = new XMLHttpRequest();
    if(element.checked){ xhr.open("GET", "/update?state=1", true);
    } else { xhr.open("GET", "/update?state=0", true); }
    xhr.send();
}

setInterval(function () {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var inputChecked;
            var outputStateM;
            if( this.responseText == 1){
                inputChecked = true;
                outputStateM = "On";
            }
            else {
                inputChecked = false;
                outputStateM = "Off";
            }
            document.getElementById("output").checked =
inputChecked;
            document.getElementById("outputState").innerHTML =
outputStateM;
        }
    };
    xhttp.open("GET", "/state", true);
    xhttp.send();
}, 1000 ) ;

```

```
</script>
</body>
</html>
)rawliteral";

// Replaces placeholder with button section in your web
page
String processor(const String& var){
    //Serial.println(var);
    if(var == "BUTTONPLACEHOLDER"){
        String buttons = "";
        String outputStateValue = outputState();
        buttons+= "<h4>Output - GPIO 2 - State <span
id=\"outputState\"></span></h4><label
class=\"switch\"><input type=\"checkbox\"
onchange=\"toggleCheckbox(this)\" id=\"output\" " +
outputStateValue + "><span
class=\"slider\"></span></label>";
        return buttons;
    }
    return String();
}

String outputState(){
    if(digitalRead(output)){
        return "checked";
    }
    else {
        return "";
    }
    return "";
}

void setup(){
    // Serial port for debugging purposes
    Serial.begin(115200);

    pinMode(output, OUTPUT);
    digitalWrite(output, LOW);
    pinMode(buttonPin, INPUT);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi..");
    }
}
```

```

// Print ESP Local IP Address
Serial.println(WiFi.localIP());

// Route for root / web page
server.on("/", HTTP_GET, [](AsyncWebServerRequest
*request){
    request->send_P(200, "text/html", index_html,
processor);
});

// Send a GET request to
<ESP_IP>/update?state=<inputMessage>
server.on("/update", HTTP_GET, []
(AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on
<ESP_IP>/update?state=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage =
request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
        digitalWrite(output, inputMessage.toInt());
        ledState = !ledState;
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    request->send(200, "text/plain", "OK");
});

// Send a GET request to <ESP_IP>/state
server.on("/state", HTTP_GET, []
(AsyncWebServerRequest *request) {
    request->send(200, "text/plain",
String(digitalRead(output)).c_str());
});
// Start server
server.begin();
}

void loop() {
// read the state of the switch into a local variable
int reading = digitalRead(buttonPin);

```

```
// check to see if you just pressed the button
// (i.e. the input went from LOW to HIGH), and you've
waited long enough
// since the last press to ignore any noise:

// If the switch changed, due to noise or pressing:
if (reading != lastButtonState) {
    // reset the debouncing timer
    lastDebounceTime = millis();
}

if ((millis() - lastDebounceTime) > debounceDelay) {
    // whatever the reading is at, it's been there for
longer than the debounce
    // delay, so take it as the actual current state:

    // if the button state has changed:
    if (reading != buttonState) {
        buttonState = reading;

        // only toggle the LED if the new button state is
HIGH
        if (buttonState == HIGH) {
            ledState = !ledState;
```

```

        }
    }

// set the LED:
digitalWrite(output, ledState);

// save the reading. Next time through the loop, it'll be the
lastButtonState:
lastButtonState = reading;
}

```

How the Code Works

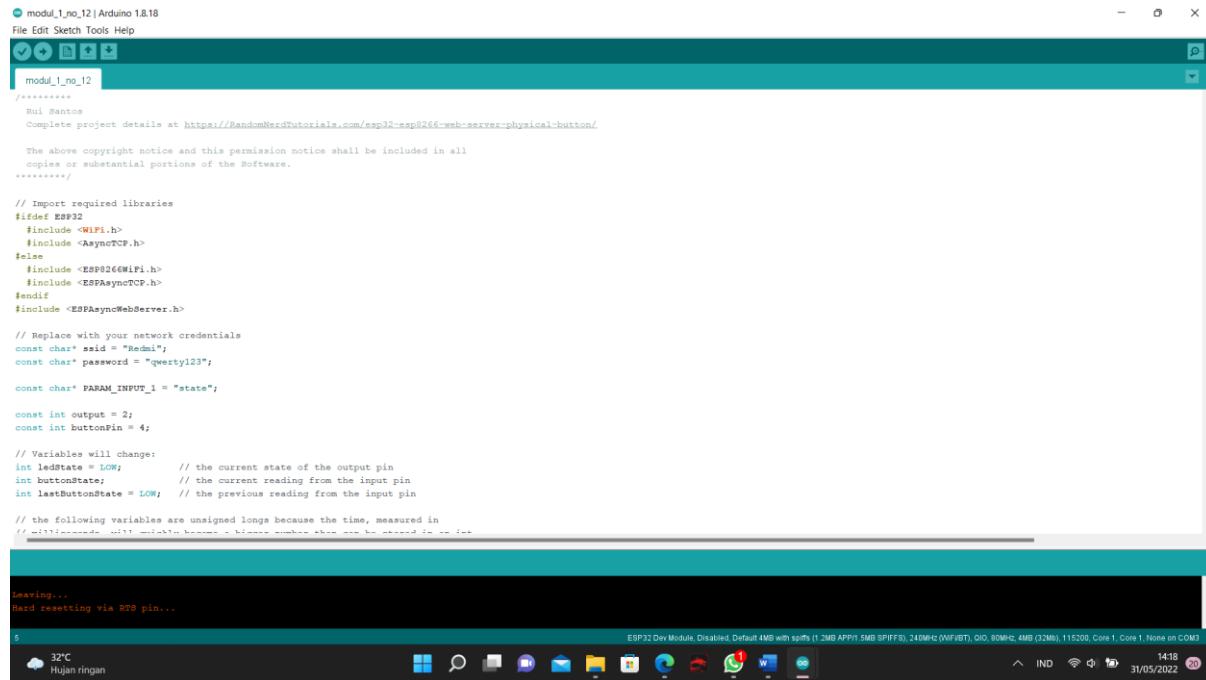
a. Network Credentials

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

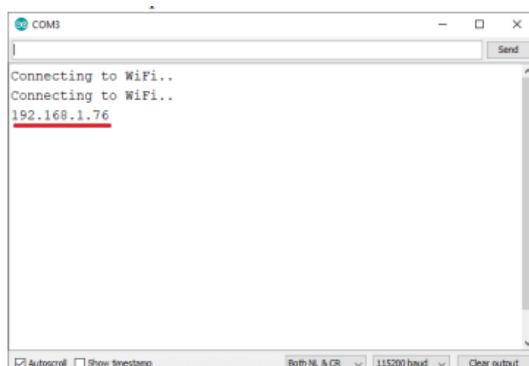
```

Contoh kasus :

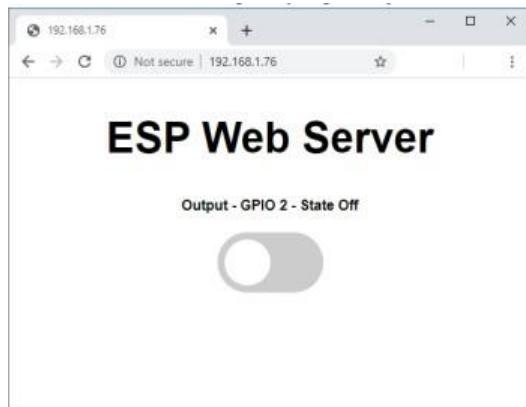


- Demonstration

Menungguh kode ke papan ESP32 atau ESP8266. Kemudian, buka Serial Monitor pada baud rate 115200. Tekan tombol EN/RST on-board untuk mendapatkan alamatIP



Buka browser di jaringan lokal, dan ketik alamat IP ESP. Saya harus memiliki akses ke server web seperti yang ditunjukkan di bawah ini.



Saya dapat mengaktifkan tombol di server web untuk menyalakan LED



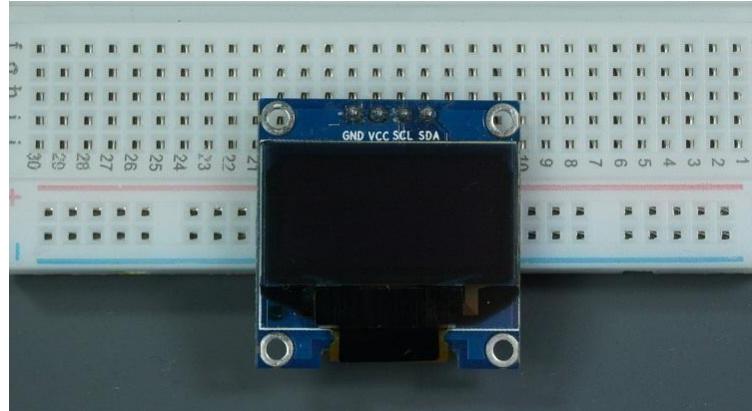
Saya juga dapat mengontrol LED yang sama dengan tombol tekan fisik. Statusnya akan selalu diperbarui secara otomatis di server web.



13. Mengetahui alamat oled

Hal ini dilakukan untuk mengetahui alamat oled yang nantinya akan dimasukan ke kodingan untuk membuat tampilan pada layar Oled

Layar OLED yang akan kita gunakan dalam tutorial ini adalah model SSD1306: layar 0,96 inci monicolor dengan 128x64 piksel seperti yang ditunjukkan pada gambar berikut.



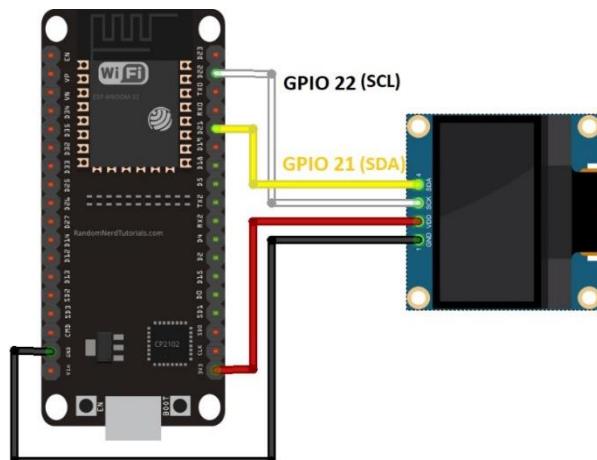
Layar OLED tidak memerlukan lampu latar, yang menghasilkan kontrasyang sangat bagus di lingkungan yang gelap. Selain itu, pikselnya hanya

mengkonsumsi energi saat menyala, sehingga layar OLED mengonsumsi lebih sedikit daya jika dibandingkan dengan layar lainnya. Model yang kami gunakan memiliki empat pin dan berkomunikasi dengan mikrokontroler apapun menggunakan protokol komunikasi I2C. Ada model yang datang dengan pin RESET ekstra atau yang berkomunikasi menggunakan protokol komunikasi SPI.

13.1 OLED Display SSD1306 Pin Wiring

| Pin | ESP32 |
|-----|---------|
| Vin | 3.3V |
| GND | GND |
| SCL | GPIO 22 |
| SDA | GPIO 21 |

Atau ikuti diagram skema berikutnya untuk menyambungkan ESP32 ke layar OLED.



Dalam contoh ini, kami menggunakan protokol komunikasi I2C. Pin yang paling cocok untuk komunikasi I2C di ESP32 adalah GPIO 22 (SCL) dan GPIO 21 (SDA). Jika Kami menggunakan layar OLED dengan protokol komunikasi SPI, gunakan GPIO berikut.

- 1) GPIO 18: CLK
- 2) GPIO 19: MISO
- 3) GPIO 23: MOSI
- 4) GPIO 5: CS

13.2 Installing SSD1306 OLED Library – ESP32

1. Tulis IDE Arduino Kami dan buka Sketch > Include Library > ManageLibraries. Manajer Perpustakaan harus terbuka.
2. Ketik "SSD1306" di kotak pencarian dan instal perpustakaan SSD1306 dari Adafruit.

Contoh kasus :

```

modul_1_no_13alamated | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_13alamated
I2C Scanning Example

void loop() {
  byte address;
  int nDevices;
  Serial.println("Scanning...");
  nDevices = 0;
  for(address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if(Wire.endTransmission() == 0) {
      if (Serial.read() == 0x00)
        Serial.print("I2C device found at address 0x");
      if (address<10)
        Serial.print("0");
      Serial.println(address,HEX);
      nDevices++;
    }
    else if (error==4) {
      Serial.print("Unknown error at address 0x");
      if (address<10)
        Serial.print("0");
      Serial.println();
    }
    Serial.println(address,HEX);
  }
  if (nDevices == 0) {
    Serial.println("No I2C devices found\n");
  }
  else {
    Serial.println("done\n");
  }
  delay(5000);
}

Leaving...
Board resetting via RTS pin...

```

The Serial Monitor window shows the output of the I2C scanning code. It lists the addresses of I2C devices found on the bus:

```

entry 0x0000510
I2C Scanner
Scanning...
I2C device found at address 0x30
done
Scanning...
I2C device found at address 0x30
done
Scanning...
I2C device found at address 0x30
done

```

Untuk Kodingan menampilkan display hello world

```

modul_1_no_13 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_13
*****_
Run Sketch
Complete project details at https://randomnerdtutorials.com
*****_

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup() {
  Serial.begin(115200);
  if(!display.begin(SSD1306_SMD_I2C_ADDRESS, 0x3C)) { // Address 0x3C for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  delay(2000);
  display.clearDisplay();

  display.setTextColor(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 10);
  // Display static text
  display.println("Hello, world!");
  display.display();
}

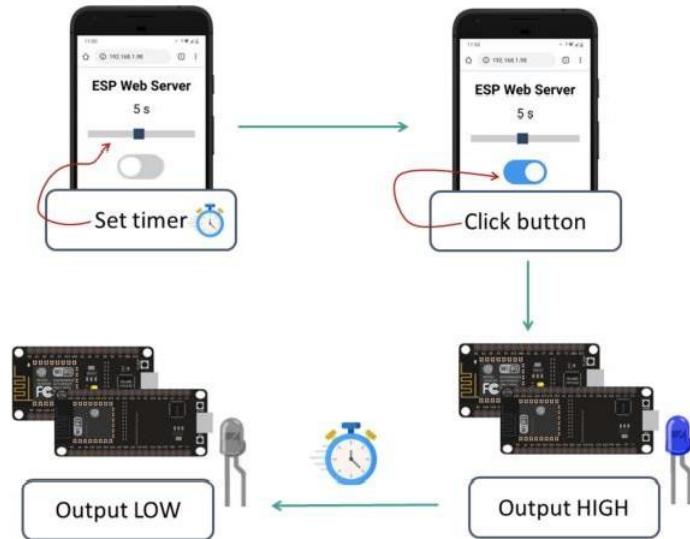
void loop() {

```



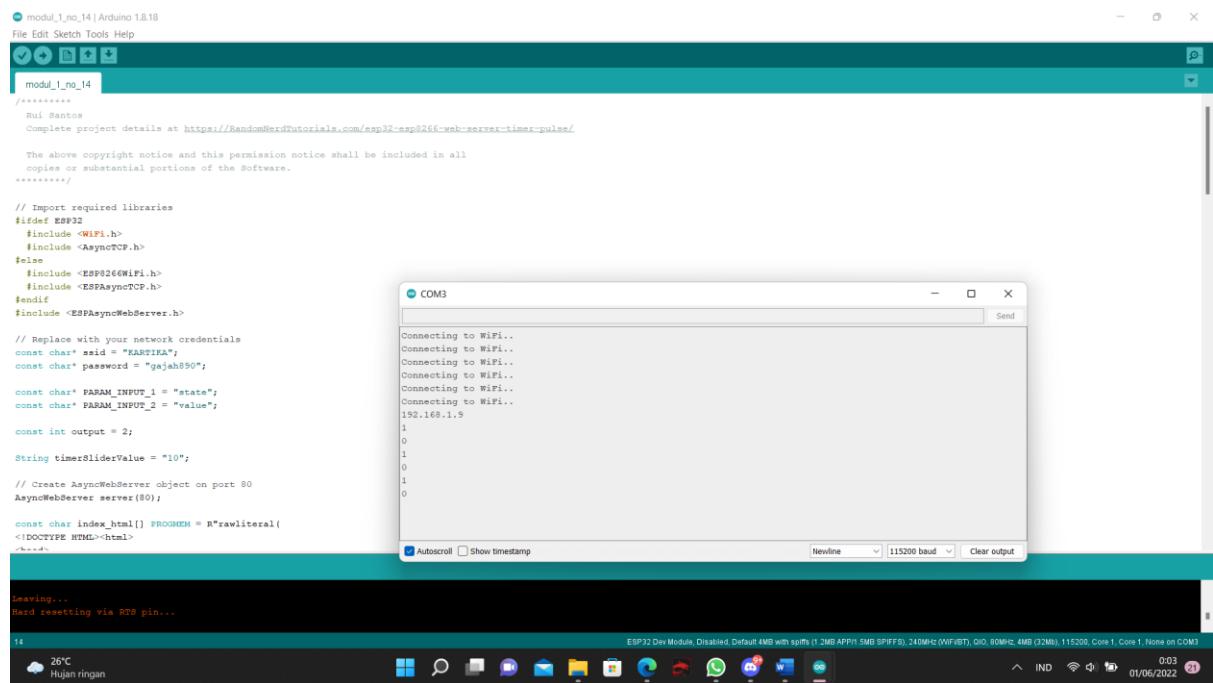
14. Timer/Pulse Web Server

Timer atau Pulse Web Server digunakan untuk menetapkan jumlah detik pada slider, yaitu memiliki cara kerja untuk menggunakan web server untuk melakukan kontrol output NodeMCU ESP32 atau ESP8266 dengan menggunakan Arduino IDE. Sehingga penentuan timer dapat diatur menggunakan slider pada halaman website, tentunya sistem pemrograman sejenis ini sangat berguna untuk melakukan kontrol terhadap yang membutuhkan sinyal tinggi selama beberapa detik yang telah ditentukan untuk saatnya digerakkan.



- ESP32 atau ESP8266 menghosting server web yang memungkinkan untuk mengontrol output dengan sinyal
- Server web berisi penggeser yang memungkinkan untuk menentukan lebar pulsa (berapa detik output harus tinggi)
- Ada tombol ON/OFF. Dapat diatur ke ON untuk mengirim pulsa. Setelah itu, Anda akan melihat timer berkurang selama durasi lebar pulsa;
- Ketika pengatur waktu berakhir, output diatur ke rendah, dan tombol server web kembali ke status OFF;
- Server web ini dapat berguna untuk mengontrol perangkat yang membutuhkan pulsa untuk diaktifkan seperti pembuka pintu garasi, misalnya.

Contoh kasus :



```

modul_1_no_14 | Arduino 1.8.18
File Edit Sketch Tools Help
modul_1_no_14
/*
  Bul Santos
  Complete project details at https://RandomNerdTutorials.com/esp32-esp8266-web-server-timer-pulse/
  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.
*/
// Import required libraries
#ifndef ESP32
#include <WIFI.h>
#include <AsyncTCP.h>
#else
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char* ssid = "MARTINA";
const char* password = "gajah890";

const char* PARAM_INPUT_1 = "state";
const char* PARAM_INPUT_2 = "value";

const int output = 2;
String timerSliderValue = "10";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<h1>ESP32 Web Server</h1>
<input type="range" min="0" max="100" value="10" id="slider">
<p>Slider value: <span>{{value}}</span></p>
</body>
</html>
)rawliteral";

```

COM3

Connecting to WiFi..
 192.168.1.9
 1
 0
 1
 0
 1
 0

Autoscroll Show timestamp Newline 115200 baud Clear output

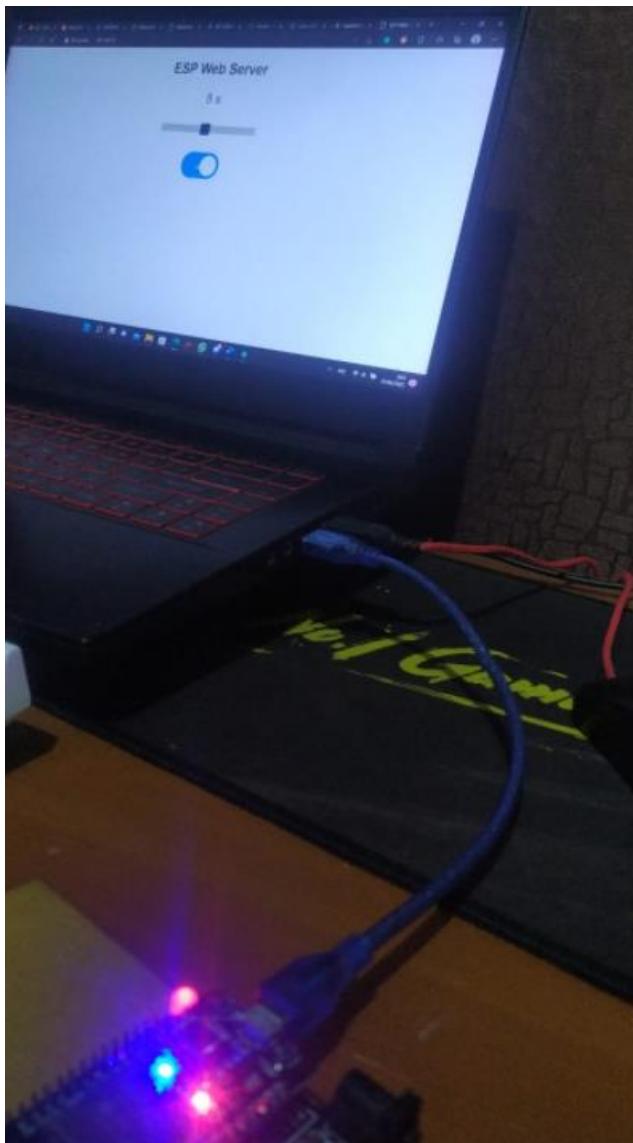
Leaving...
 Hard resetting via RTS pin...

14

ESP32 Dev Module, Disabled, Default 4MB with splits (1.2MB APP/1.5MB SPIFFS), 240MHz (WiFi/BT), QIO, 80MHz, 4MB (32MiB), 115200, Core 1, Core 1, None on COM3

26°C Hujan ringan

IND 003 01/06/2022 21



Cara Kerja : Pada program Arduino IDE, terdapat konfigurasi untuk Esp32 yang bisa terkoneksi dengan WebServer untuk dapat mengatur timer untuk menyalaakan lampu dengan slider. Kita bisa mengatur timer nya sesuai keinginan. Output (dalam hal ini GPIO 2 – built-in LED) akan tetap menyala selama jangka waktu yang telah Anda atur pada slider.