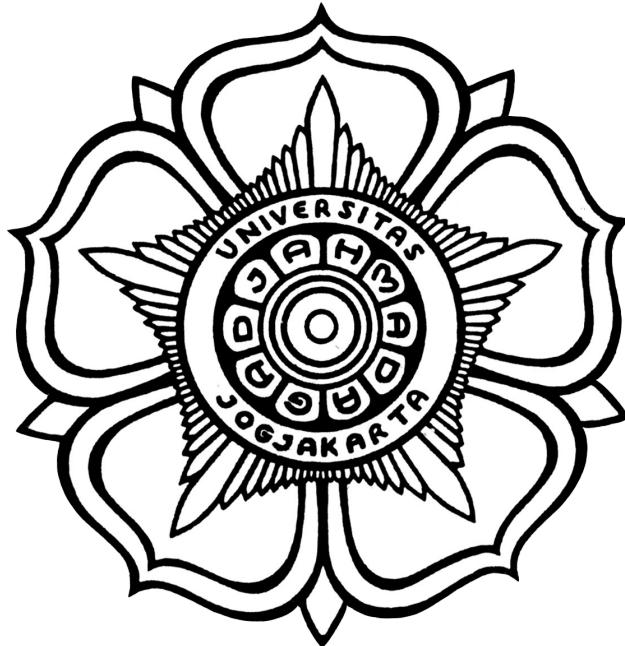


**LAPORAN PRAKTIKUM ELEKTRONIKA MESIN LISTRIK DAN
TEKNIK KENDALI**
32 bit Microcontroller

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



Disusun Oleh:
Kelompok 9

Pintien Wicaksono (19/447122/SV/16841)

Sidiq Maulana Abdulloh (19/447128/SV/16847)

Kelas: ARM 2

TEKNOLOGI REKAYASA MESIN
DEPARTEMEN TEKNIK MESIN SEKOLAH VOKASI
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2022

BAB I

Spesifikasi

Spesifikasi chip ESP32 :

- ESP32 memiliki dual core processor, artinya memiliki 2 prosesor.
- Memiliki Wifi dan bluetooth built-in, kondisi ini tentu memiliki kelebihan dalam penggunaannya yaitu kita tidak perlu menambah modul lain ketika akan melakukan pekerjaan kontrol yang membutuhkan bluetooth dan Wifi
- ESP32 dapat menjalankan program 32 bit.
- Frekuensi clock bisa mencapai 240 MHz dan memiliki RAM 512 kB.
- Modul chip ESP32 ini memiliki 30 atau 36 pin, 15 di setiap baris
- 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.
- Terdapat 15 pin channel ADC (*Analog to Digital Converter*)
- Terdapat 25 pin PWM (Pulse Width Modulation)
- Terdapat 2 pin channel DAC (*Digital to Analog Converter*)

Spesifikasi Chip ESP32 DEVKIT V1 DOIT

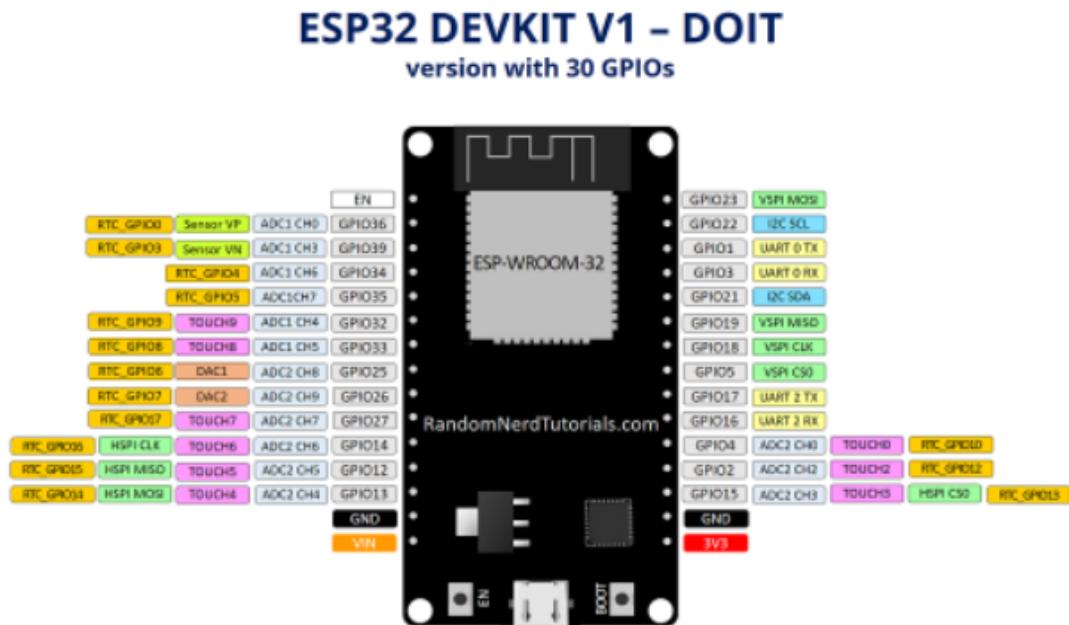
Number of cores	2 (dual core)
Wi-Fi	2.4 GHz up to 150 Mbits/s
Bluetooth	BLE (Bluetooth Low Energy) and legacy Bluetooth
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30 or 36 (depends on the model)
Peripherals	Capacitive touch, ADC (analog to digital converter), DAC (digital to analog 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.

BAB II

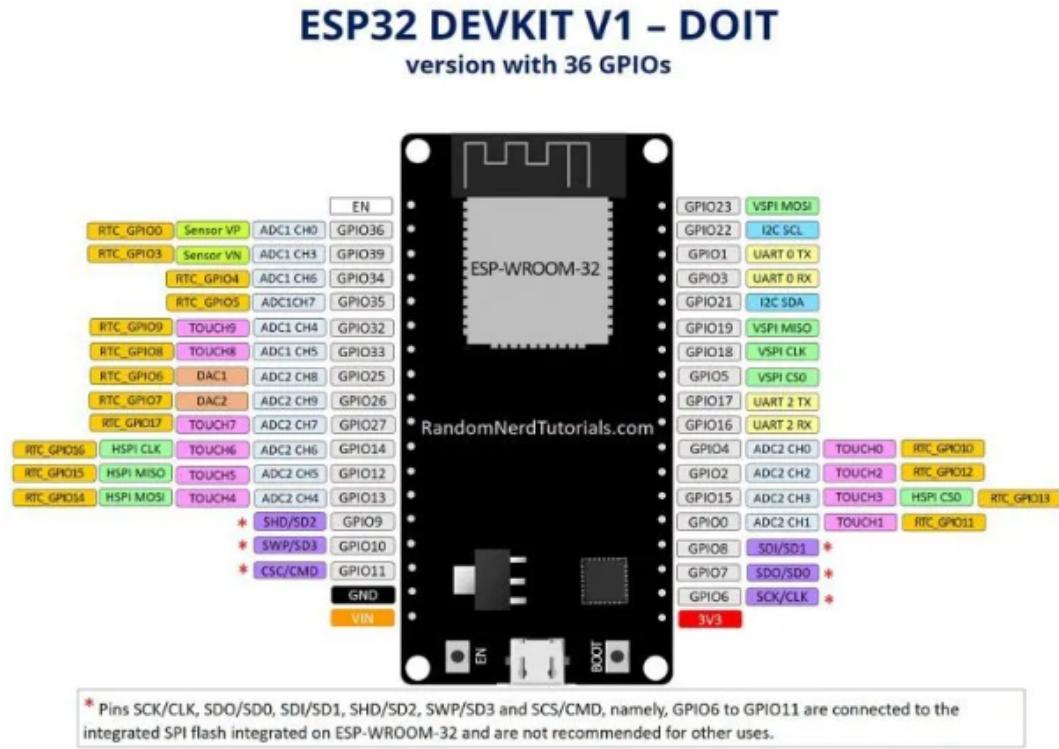
PinOut Beserta Fungsinya.

ESP32 adalah mikrokontroler yang dikenalkan oleh Espressif System dan 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 juga memiliki lebih banyak GPIO sehingga lebih banyak fungsi yang dapat dilakukan dibandingkan dengan ESP826. GPIO adalah singkatan dari General Purpose Input Output yaitu sebuah pin yang terdapat pada sebuah kontroler yang berfungsi untuk mengalirkan data ataupun tegangan listrik di mana pin tersebut bisa digunakan sebagai media penyalur input atau output, input dan output-nya dapat berupa data atau tegangan listrik. Umumnya juga, pin-pin ini disusun dengan multiplexing, 1 pin bisa memiliki berbagai fungsi, bisa menjadi input, output, PWM, interface communication seperti UART, SPI, dan i2c, bahkan sebagai media fisik dari capacitive sensor sesuai dengan pendefinisian yang kita buat pada kode arduinonya.

a. ESP32 DEVKIT V1 dengan 30 GPIO



b. ESP32 DEVKIT V1 dengan 30 GPIO



Pada board ESP32 DevKit terdapat 25 pin GPIO (*General Purpose Input Output*) dengan masing – masing pin mempunyai karakteristik sendiri – sendiri.

Pin hanya sebagai INPUT :

- GPIO 34
- GPIO 35
- GPIO 36
- GPIO 39

Pin dengan internal pull up, dapat disetting melalui program :

- GPIO14
- GPIO16
- GPIO17
- GPIO18
- GPIO19
- GPIO21
- GPIO22
- GPIO23

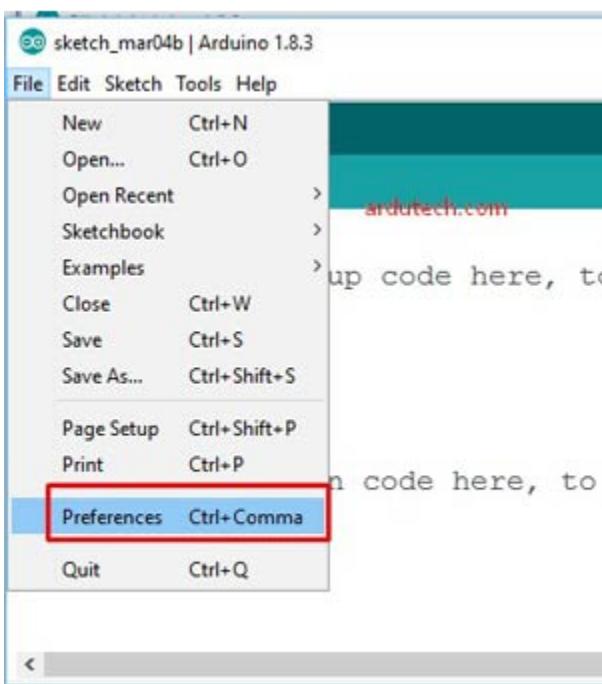
Pin tanpa internal pull up (dapat ditambahkan pull up eksternal sendiri) :

- GPIO13
- GPIO25
- GPIO26
- GPIO27
- GPIO32
- GPIO33

BAB III

Penggunaan Arduino IDE (*setting/konfigurasi*)

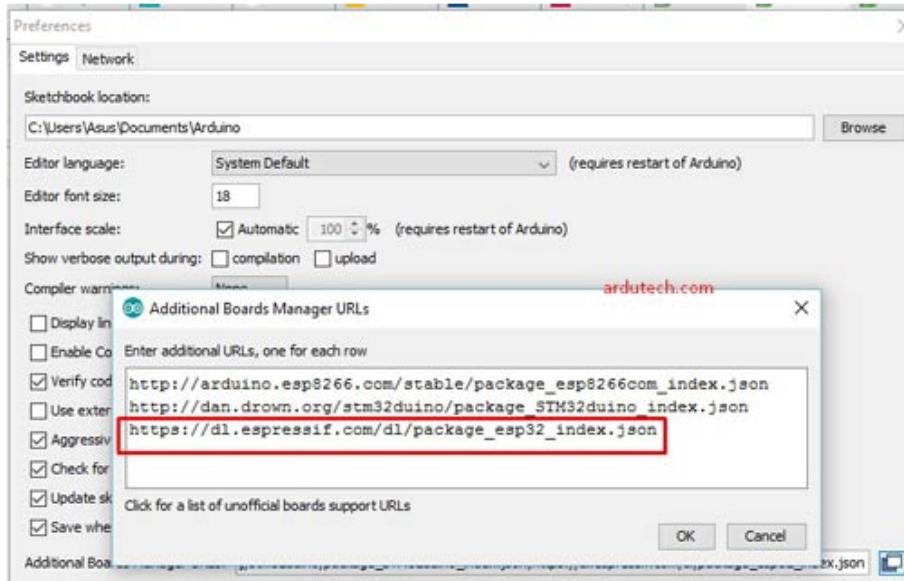
1. Menyiapkan Arduino IDE, kemudian buka/jalankan Arduino IDE.
2. Dari menu **File > Preferences** akan tampil jendela **Preferences**.



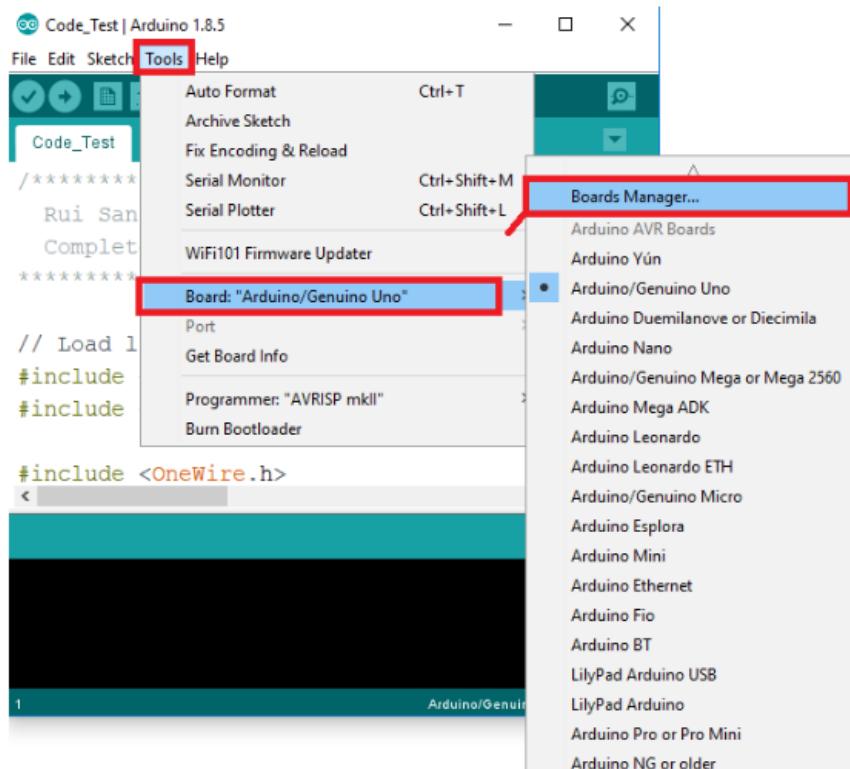
3. Pada kolom “Additional Boards Manager URLs:” mengisikan :

```
https://raw.githubusercontent.com/espressif/arduino  
-esp32/gh-pages/package_esp32_index.json
```

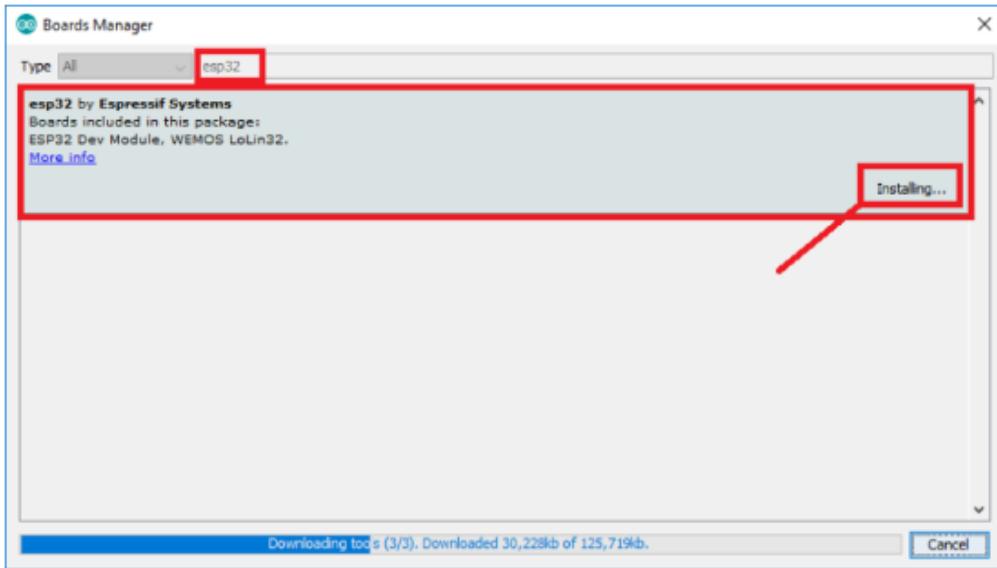
Jika sebelumnya sudah ada link lain seperti esp8266 klik kotak disebelah kanan kemudian letakkan link yang baru tadi (esp32) dibagian bawahnya atau dapat juga dengan tanda koma. Kemudian klik “OK”.



4. Membuka Boards Manager dari menu Tools > Board > Boards Manager ...



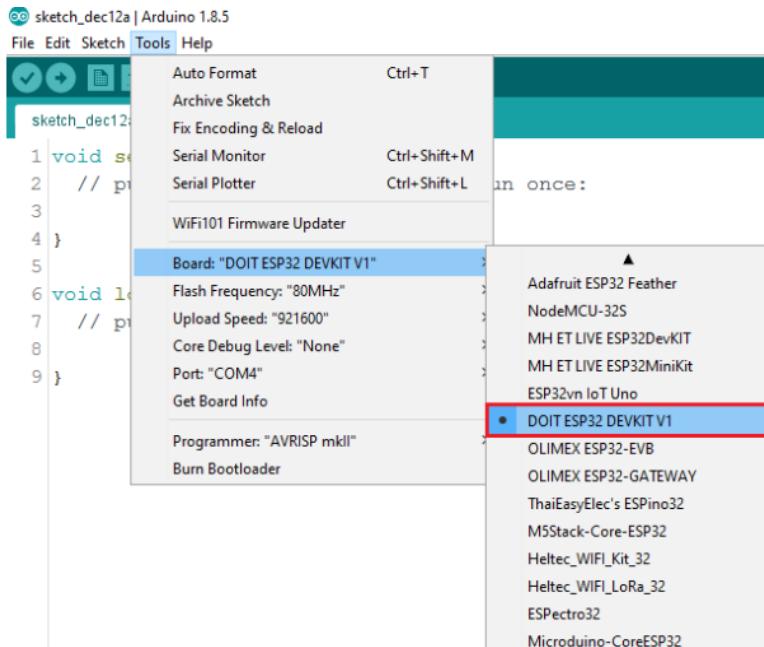
5. Mencari **ESP32** pada kolom *search* kemudian memilih “**ESP32 by Espressif Systems**“lalu mengklik “Install”.



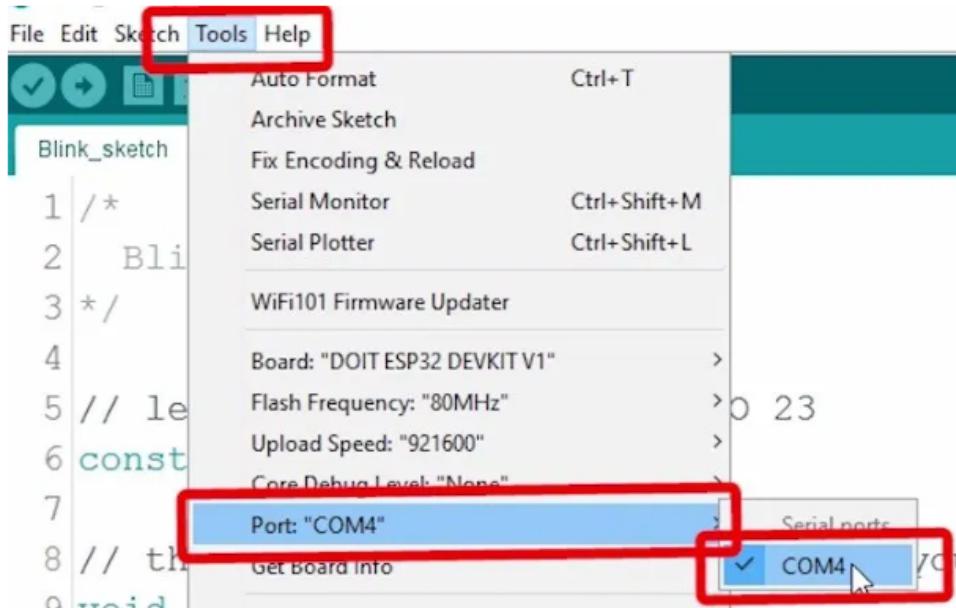
Kemudian pastikan komputer/laptop terhubung dengan internet. Lalu ditunggu sampai proses *downloading* selesai.

Untuk mengetes kinerja dari aplikasi yang sudah diinstal maka dapat dilakukan langkah-langkah sebagai berikut.

1. Menghubungkan ESP32 DevKit ke port USB komputer. Kemudian memilih jenis boardnya dari menu **Tools > Board > ESP32 Dev Module**.



2. Memilih nomor Port-nya yaitu dengan mengecek di *Device Manager*. Jika port belum terdeteksi, disarankan untuk menginstal driver usb-nya (ESP32 DevKit memakai driver USB CP210x). Lalu menyesuaikan nomor port, dari menu **Tools > Port**.

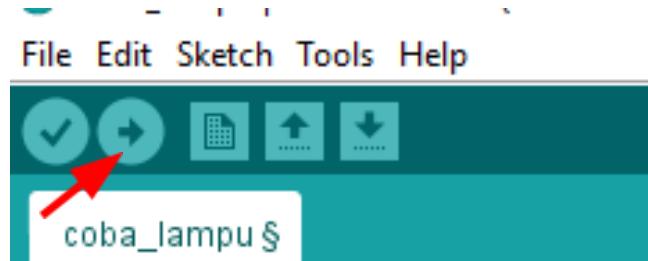


3. Kemudian coding dapat dimasukan dan wiring pada sistem dirakit sesuai dengan terminal yang telah didefinisikan.

CODING

```
int ledPin = 2;  
void setup()  
{  
    pinMode(ledPin, OUTPUT);  
}  
void loop()  
{  
    digitalWrite(ledPin, HIGH);  
    delay(1000);  
    digitalWrite(ledPin, LOW);  
    delay(1000);  
}
```

4. Setelah itu file/coding disimpan (**Save**) kemudian **Upload**. Tunggu proses *compiling* selesai dan sukses (tidak ada error) kemudian tunggu proses *uploading* selesai dan sukses (mengisikan program dari komputer ke memori ESP32)



5. Jika sudah maka akan terlihat LED di board ESP32 DevKit akan nyala berkedip (LED biru) dengan delay 1 detik

BAB IV

ESP32 Input/Output

4.1.1 ESP32 Output Digital

Untuk mengkonfigurasi output pada ESP 32 dapat dilakukan dengan langkah sebagai berikut :

Pertama, dengan mengatur GPIO yang ingin dikontrol sebagai **OUTPUT**. Dalam mengontrol output sendiri menggunakan fungsi `pinMode()` sebagai berikut:

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

Untuk mengontrol output digital kami cukup menggunakan fungsi `digitalWrite()`, yang menerima sebuah argument, GPIO (nomor int) dengan statusnya yaitu HIGH atau LOW.

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

Semua pin GPIO dapat digunakan sebagai output kecuali pada GPIO 6 sampai 11 (karena terhubung ke flash SPI yang terintegrasi) serta GPIO 34, 35, 36 dan 39 (hanya input GPIO);

4.1.2 ESP32 Input Digital

Untuk mengkonfigurasi input pada ESP 32 dapat dilakukan dengan langkah sebagai berikut :

Pertama, dengan mengatur GPIO yang ingin dibaca sebagai INPUT, menggunakan fungsi `pinMode()` sebagai berikut:

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

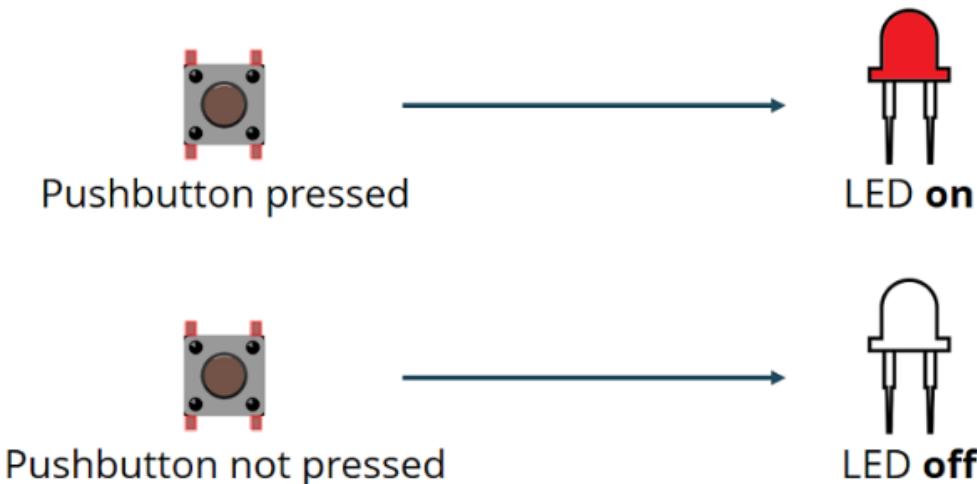
Untuk membaca input digital, seperti tombol, kita bisa menggunakan fungsi `digitalRead()`, yang mana dapat menerima argumen pada GPIO (nomor int) yang diatur.

```
digitalRead(GPIO);
```

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

4.1.3 Contoh Project Sederhana

Untuk mengetahui bagaimana cara menggunakan input digital dan output digital, kami membuat contoh proyek sederhana menggunakan tombol saklar dan lampu LED sebagai berikut.



4.3.1. Kode Program

```
//*****
Bissmillahirrohmanirrohim :)
ESP32 Input/Output
*****/
// set pin numbers
const int buttonPin = 5; // the number of the
pushbutton pin
const int ledPin = 4; // 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);
    }
}
```

```
    }  
}
```

4.3.2 Pendefinisian Kerja

Pada 2 baris atas dalam kode program menentukan judul program. Tanda `/*.....*/` menunjukkan judul program, bagian tersebut bebas akan diisi apapun, apakah penjelasan, judul maupun tulisan lainnya karena tidak mempengaruhi program ketika dijalankan. Lalu 2 baris berikutnya yaitu menunjukkan variabel pin mana saja yang kami gunakan dalam program ini

```
const int buttonPin = 5; // the number of the  
pushbutton pin  
const int ledPin = 4; // the number of the  
LED pin
```

Dari variabel diatas menunjukkan bahwa Push Button disalurkan ke pin GPIO 5 sedangkan LED disalurkan ke pin GPIO 4 pada ESP32. Lalu ketika Push Button ditekan maka yang akan terjadi ialah arus akan berhenti sehingga lampu akan mati. Kode ketika lampu mati diwakili oleh angka 0 sesuai dengan default angka digital yaitu 0 = mati

```
int buttonState = 0
```

Kemudian kita juga perlu melakukan pendefinisian pada kedua terminal / pin yaitu mengenai Input dan Outputnya. Dalam program ini Button Pin didefinisikan sebagai Input program kemudian LED didefinisikan sebagai Output program. Untuk mengaktifkan pendefinisian tersebut maka digunakan fungsi `pinMode`

```
// initialize the pushbutton pin as an input  
pinMode(buttonPin, INPUT);  
// initialize the LED pin as an output  
pinMode(ledPin, OUTPUT)
```

Lalu baris berikutnya adalah untuk membaca kondisi tombol dan menyimpannya di button state variable. Untuk pembacaan tersebut seperti yang kita lihat sebelumnya yaitu menggunakan `digitalRead()` function.

```
buttonState = digitalRead(buttonPin)
```

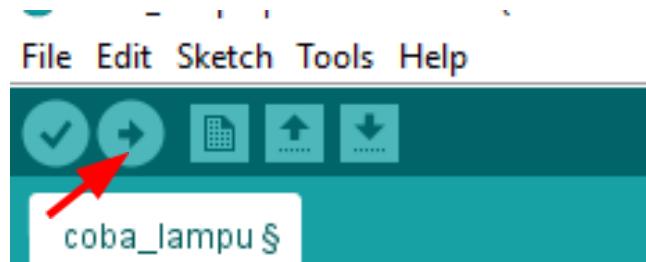
\Kemudian untuk memeriksa apakah status tombol adalah HIGH atau LOW, kita menggunakan fungsi IF. Jika HIGH, maka akan menyalakan LED menggunakan fungsi digitalRead() yang menerima sebagai argumen ledpin, dan HIGH state.

```
if (buttonState == HIGH) {  
    // turn LED on  
    digitalWrite(ledPin, HIGH);  
} else {  
    // turn LED off  
    digitalWrite(ledPin, LOW)
```

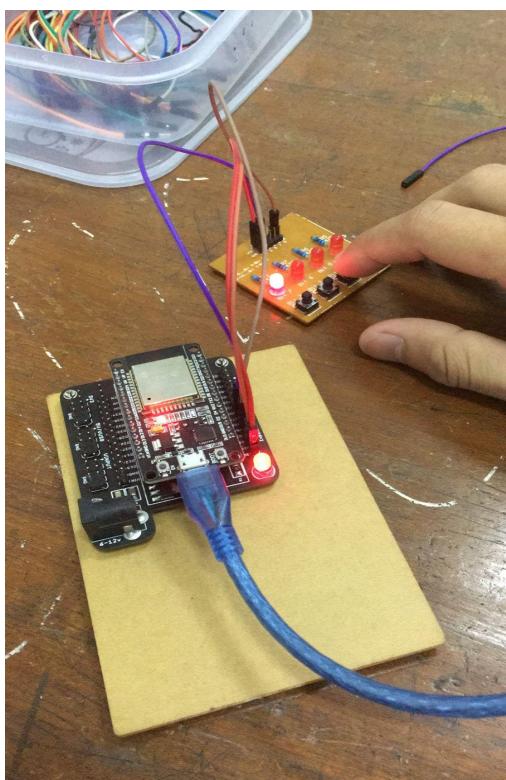
Jika status tombol tidak HIGH atau LOW, maka LED dihidupkan. bisa diberikan pengecualian LED LOW sebagai argumen kedua dalam digitalWrite() function. Langkah selanjutnya adalah mengupload program

4.3.3 Mengupload Program

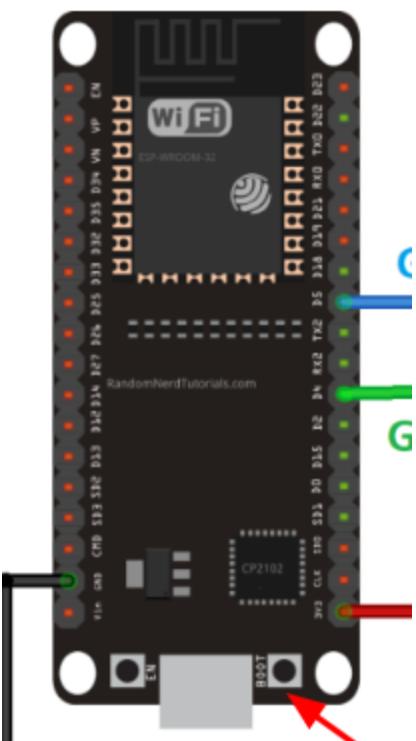
Sebelum mengklik tombol unggah, terlebih dahulu buka **Alat > Papan**, dan pilih papan yang digunakan. Dalam kasus ini adalah papan **DOIT ESP32 DEVKIT V1**. Kemudian buka **Alat > Port** dan pilih port COM yang terhubung dengan ESP32. Kemudian, tekan tombol unggah dan tunggu pesan “**Done Uploading**”.



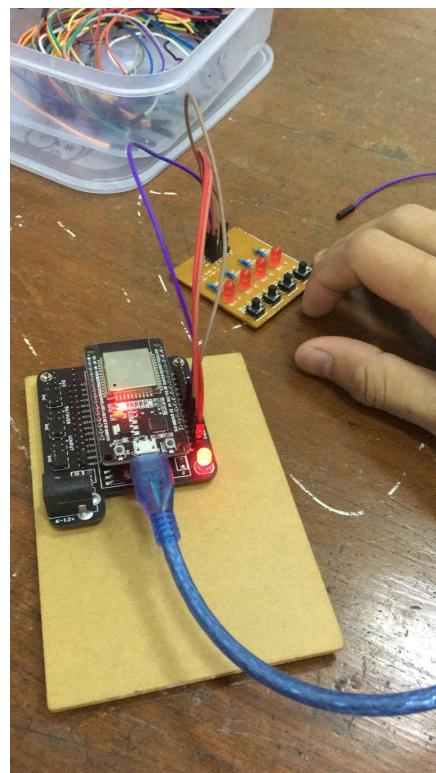
Jika terlihat banyak titik (...__..._) pada jendela debugging dan pesan “Failed to connect to ESP32: Timed out waiting for packet header”, itu berarti perlu menekan tombol BOOT on-board ESP32 setelah titik mulai muncul.



Position On



BOOT



Position Off

BAB V

ESP32 PWM (*Pulse Width Modulation*)

Pada metode ESP32 PWM dilakukan sama dengan menggunakan Arduino IDE. Sebagai contoh dimana kami akan membuat suatu rangkaian sederhana yang hanya meredupkan LED menggunakan pengontrol PWM LED dari ESP32. ESP32 memiliki 16 saluran independent.

Berikut beberapa rangkaian ketika meredupkan LED dengan PWM dan menggunakan Arduino IDE:

1. Kami harus memilih saluran yang ada 16 ada 0-15
2. Kemudian, kami perlu mengatur frekuensi sinyal PWM. Untuk LED, frekuensi 5000 Hz aman saja ketika digunakan.
3. Kami juga perlu mengatur resolusi siklus tugas sinyal: Kami harus memiliki resolusi dari 1-16 bit. Kita menggunakan resolusi 8-bit. Yang berarti kita dapat mengontrol kecerahan LED menggunakan nilai dari 0 - 255.
4. Selanjutnya, perlu menentukan ke GPIO atau GPIO mana sinyal yang akan muncul. Oleh karena itu, kami akan menggunakan fungsi `ledcAttachPin (GPIO, Saluran)`.
5. Akhirnya, untuk mengontrol kecerahan LED menggunakan PWM kami.

5.1.1 Kode Program

```
### CODING
// the number of the LED pin
const int ledPin = 21; // 21 corresponds to GPIO21
const int ledPin2 = 19; // 19 corresponds to GPIO19
const int ledPin3 = 18; // 18 corresponds to GPIO18
const int ledPin4 = 14; // 14 corresponds to GPIO14

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

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

    // attach the channel to the GPIO to be controlled
    ledcAttachPin(ledPin, ledChannel);
    ledcAttachPin(ledPin2, ledChannel);
    ledcAttachPin(ledPin3, ledChannel);
    ledcAttachPin(ledPin4, 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);
    }
}
```

Latihan_PWM | Arduino 1.8.20 Hourly Build 2022/04/25 09:33

```

File Edit Sketch Tools Help
  Auto Format Ctrl+T
  Archive Sketch
  Fix Encoding & Reload
  Manage Libraries... Ctrl+Shift+I
  Serial Monitor Ctrl+Shift+M
  Serial Plotter Ctrl+Shift+L
  WiFi101 / WiFiNINA Firmware Updater
Board: "DOIT ESP32 DEVKIT V1"
Upload Speed: "921600"
Flash Frequency: "80MHz"
Core Debug Level: "None"
Port
Get Board Info
Programmer
  Burn Bootloader
ledcAttachPin(ledPin4, 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);
  }
}

Done uploading.

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

```

Boards Manager...
 Arduino AVR Boards
 ESP32 Arduino
 Adafruit Feather ESP32-S3 No PSRAM
 Adafruit QT Py ESP32-S3 No PSRAM
 NodeMCU-32S
 MH ET LIVE ESP32DevKIT
 MH ET LIVE ESP32MiniKit
 ESP32n IoT Uno
 DOIT ESP32 DEVKIT V1
 DOIT ESPDuino32
 OLIMEX ESP32-EVB
 OLIMEX ESP32-GATEWAY
 OLIMEX ESP32-PoE
 OLIMEX ESP32-PoE-ISO
 OLIMEX ESP32-DevKit-LiPo
 ThaiEasyFle's ESPino32
 M5Stack-Core-ESP32
 M5Stack-FIRE
 M5Stick-C
 M5Stack-ATOM
 M5Stack-Core2
 M5Stack-Timer-CAM
 M5Stack-CoreLnk
 ODROID ESP32
 Heltec WiFi LoRa 32
 Heltec WiFi LoRa 32(V2)

DOIT ESP32 DEVKIT V1, 80MHz, 921600, None on COM3

Pada bagian kasus ESP32 PWM, kami menggunakan Board *DOIT ESP32 DEVKIT V1* dan menggunakan *serial com port* sesuai dengan device yang tersedia di laptop masing-masing.

Contoh kerja praktek 4 lamp sebagai output:



BAB VI

ESP32 Interrupt

Pada sistem ini sebuah ESP32 mendeteksi gerakan menggunakan sensor gerak PIR. Dalam contoh ini, ketika gerakan terdeteksi (interupsi dipicu), ESP32 memulai pengatur waktu dan menyalakan LED selama beberapa detik yang telah ditentukan. Ketika penghitung waktu selesai menghitung mundur, LED mati secara otomatis.

Dengan contoh ini kami juga akan mengeksplorasi dua konsep penting: interupsi dan timer

Untuk dapat melakukan suatu peristiwa dengan sensor gerak PIR, Kami menggunakan interupsi. Interupsi berguna untuk membuat sesuatu terjadi secara otomatis dalam program mikrokontroler, dan dapat membantu memecahkan masalah waktu.

Dengan interupsi, kami tidak perlu terus-menerus memeriksa nilai pin saat ini. Ketika perubahan terdeteksi, suatu peristiwa dipicu (fungsi dipanggil). Untuk mensetting interupsi di Arduino IDE, kami menggunakan fungsi `attachInterrupt()`, yang menerima sebagai argumen adalah pin GPIO, dan nama fungsi yang akan dieksekusi, dan mode:

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

6.1.1 GPIO Interrupt

Argumen pertama adalah nomor GPIO. Biasanya, harus menggunakan `digitalPinToInterrupt(GPIO)` untuk mengatur GPIO yang sebenarnya sebagai pin interupsi. Misalnya, jika ingin menggunakan GPIO 27 sebagai interupsi, gunakan: `digitalPinToInterrupt(27)`

6.1.2 Function to be triggered

Untuk argumen kedua dari fungsi `attachInterrupt()` adalah nama fungsi yang akan dipanggil setiap kali interupsi dipicu.

6.1.3 Mode

Argumen ketiga adalah mode. Ada 5 mode yang berbeda diantaranya adalah:

- Low: untuk memicu interupsi setiap kali pin LOW;
- High: untuk memicu interupsi setiap kali pin HIGH;
- Change: untuk memicu interupsi setiap kali pin mengubah nilai – misalnya dari HIGH ke LOW atau LOW ke HIGH;
- Falling: ketika pin beralih dari TINGGI ke RENDAH;
- Rising: untuk memicu ketika pin berubah dari LOW ke HIGH

6.1.4 Introducing Timers

Dalam contoh ini kami juga akan memperkenalkan timer. Kami ingin LED tetap menyala selama beberapa detik yang telah ditentukan setelah gerakan terdeteksi. Alih-alih menggunakan fungsi delay() yang memblokir kode dan tidak memungkinkan Kami melakukan hal lain selama beberapa detik yang ditentukan, kami harus menggunakan timer.

```
delay(time in milliseconds)
```

Ketika kami melakukan delay(1000) program kami berhenti pada baris itu selama 1 detik. delay() adalah fungsi pemblokiran. Fungsi pemblokiran mencegah program melakukan hal lain sampai tugas tertentu selesai. Jika Anda memerlukan beberapa tugas untuk dilakukan pada saat yang sama, Anda tidak dapat menggunakan delay().

Untuk sebagian besar proyek, Kami harus menghindari penggunaan penundaan dan menggunakan pengatur waktu sebagai gantinya.

6.1.5 The millis() function

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

```
millis()
```

6.1.6 Kode Program

```
### CODING
*****
Bismillahirrohmanirrohim :)
Blinking an LED with Millis
*****/

// constants won't change. Used here to set a pin number :
const int ledPin = 25; // the number of the LED pin

// Variables will change :
int ledState = LOW; // ledState used to set the LED

// Generally, you should use "unsigned long" for variables that hold time
// 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 = 500;          // 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();

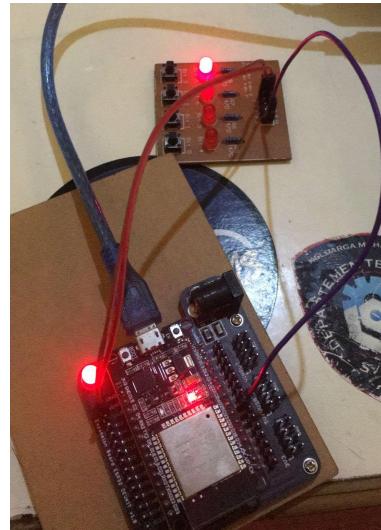
  if (currentMillis - previousMillis >= interval) {
    // save the last time you blinked the LED
    previousMillis = currentMillis;

    // 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);
  }
}

```

Contoh kerja praktek 1 lamp sebagai output dan GPIO di D25:



BAB VII

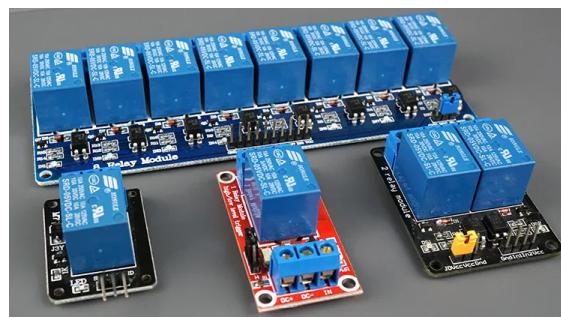
ESP32 Relay

7.1.1 Pengenalan Relay

Relay adalah saklar yang dioperasikan secara listrik dan seperti saklar lainnya, relai dapat dihidupkan atau dimatikan, membiarkan arus mengalir atau tidak. Hal ini dapat dikontrol dengan tegangan rendah, seperti 3.3V yang disediakan oleh GPIO ESP32 dan memungkinkan kita untuk mengontrol tegangan tinggi seperti 12V, 24V atau tegangan listrik (230V di Eropa dan 120V di AS)

7.1.2. Modul Relay 1, 2, 4, 8, 16 Saluran

Ada modul relay yang berbeda dengan jumlah saluran yang berbeda. Dapat menemukan modul relay dengan satu, dua, empat, delapan, dan bahkan enam belas saluran. Jumlah saluran menentukan jumlah keluaran yang dapat kami kendalikan.



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.

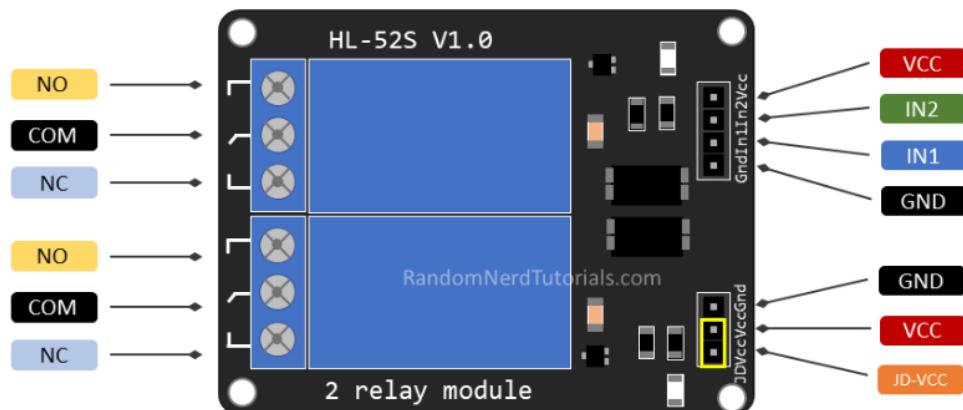
Selain itu, beberapa dilengkapi dengan optocoupler internal yang menambahkan "lapisan" perlindungan ekstra, yang secara optik mengisolasi ESP32 dari sirkuit relay.

Beberapa modul relay:

- [5V 2-channel relay module](#) (with optocoupler)
- [5V 1-channel relay module](#) (with optocoupler)
- [5V 8-channel relay module](#) (with optocoupler)
- [5V 16-channel relay module](#) (with optocoupler)
- [3.3V 1-channel relay module](#) (with optocoupler)

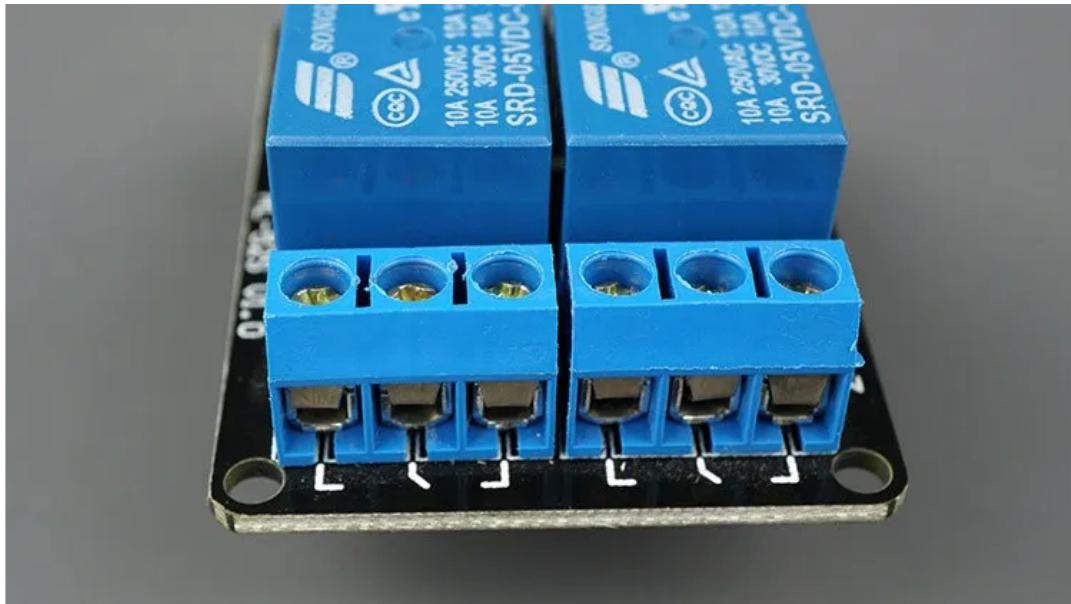
7.1.3 Pinout Relay

Untuk tujuan demonstrasi, mari kita lihat pinout modul relai 2 saluran. Menggunakan modul relay dengan jumlah saluran yang berbeda serupa.



Di sisi kiri, ada dua set tiga soket untuk menghubungkan tegangan tinggi, dan pin di sisi kanan (tegangan rendah) terhubung ke GPIO ESP32.

7.1.4 Koneksi Tegangan Listrik



Modul relay yang ditunjukkan pada foto sebelumnya memiliki dua konektor, masing-masing dengan tiga soket: common (**COM**), Normally Closed (**NC**), and Normally Open (**NO**).

- **COM**: hubungkan arus yang ingin Anda kendalikan (tegangan listrik).
- **NC (Normally Closed)**: konfigurasi yang biasanya tertutup digunakan bila ingin relai ditutup secara default. NC adalah pin COM yang terhubung, artinya arus mengalir kecuali jika Anda mengirim sinyal dari ESP32 ke modul relay untuk membuka rangkaian dan menghentikan aliran arus.
- **NO (Normally Open)**: konfigurasi yang biasanya terbuka bekerja sebaliknya: tidak ada koneksi antara pin NO dan COM, sehingga sirkuit terputus kecuali apabila dilakukan pengiriman sinyal dari ESP32 untuk menutup sirkuit.

7.1.5 Pin Kontrol



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

Jika modul relay hanya memiliki satu saluran, maka hanya akan memiliki satu pin IN. Jika apabila memiliki empat saluran, maka akan memiliki empat pin IN, dan seterusnya.

Sinyal yang dikirim ke pin IN, menentukan apakah relay aktif atau tidak. Relay dipicu ketika input berjalan di bawah sekitar 2V. Ini berarti akan memiliki tahap berikut:

- **Normally Closed configuration (NC) :**

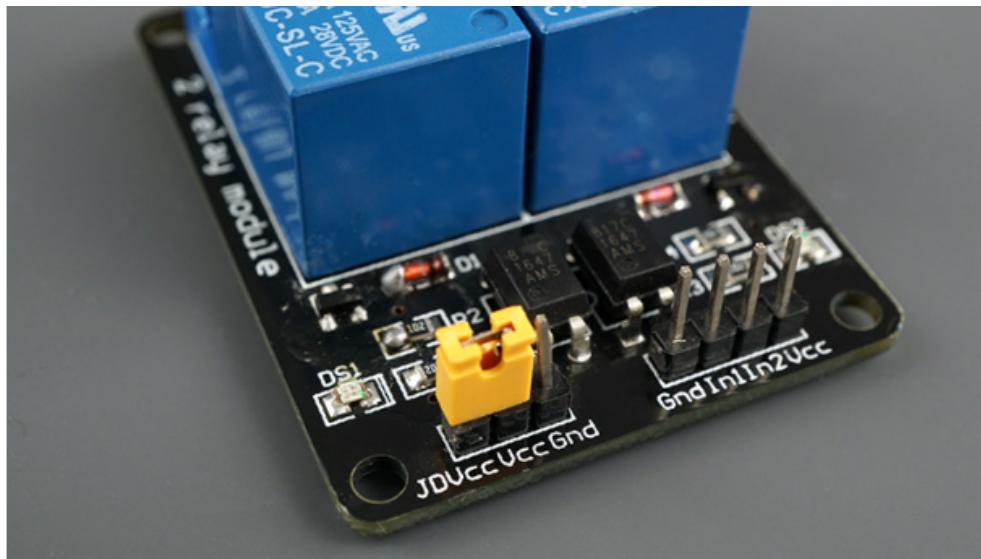
- Sinyal HIGH – arus mengalir
- Sinyal LOW – arus tidak mengalir

- **Normally Open configuration (NO) :**

- Sinyal HIGH – arus tidak mengalir
- Sinyal LOW – arus mengalir

Dalam program ini, digunakan konfigurasi yang biasanya tertutup ketika arus harus mengalir sebagian besar waktu, dan hanya ingin menghentikannya sesekali. Namun, gunakan konfigurasi yang biasanya terbuka ketika ingin arus mengalir sesekali (misalnya, menyalakan lampu sesekali).

7.1.6 Pemilihan Catu Daya



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

Dengan tutup jumper terpasang, **VCC** dan **JD-VCC** pin terhubung. Itu berarti elektromagnet relai dialiri langsung dari pin power ESP32, sehingga modul relay dan rangkaian ESP32 tidak terisolasi secara fisik satu sama lain.

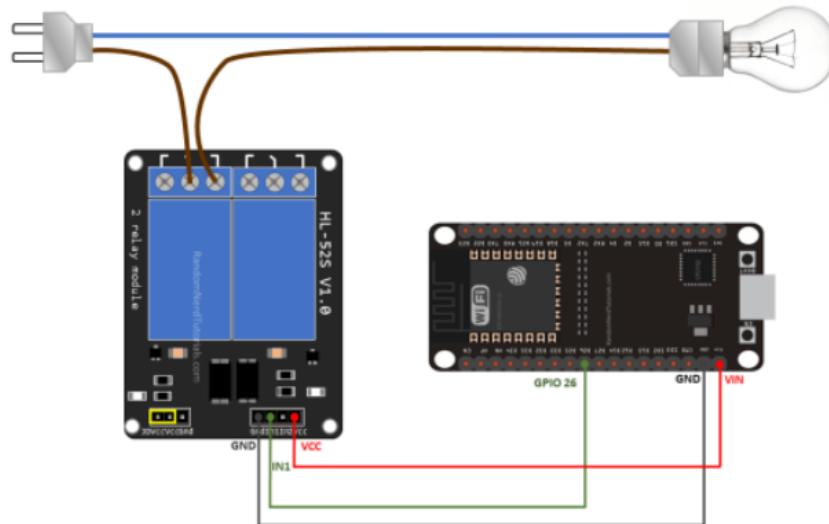
Tanpa tutup jumper, maka kita perlu menyediakan sumber daya independen untuk menyalakan elektromagnet relay melalui **JD-VCC** pin. Konfigurasi itu secara fisik mengisolasi relai dari ESP32 dengan optocoupler bawaan modul, yang mencegah kerusakan pada ESP32 jika terjadi lonjakan listrik.

7.1.7 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.

***Peringatan:** dalam contoh ini, kita berurusan dengan tegangan listrik. Penyalahgunaan dapat mengakibatkan cedera serius. Jika Kami tidak terbiasa dengan tegangan listrik, mintalah seseorang yang akan membantu Kami. Saat memprogram ESP atau memasang kabel sirkuit Kami, pastikan semuanya terputus dari tegangan listrik.*

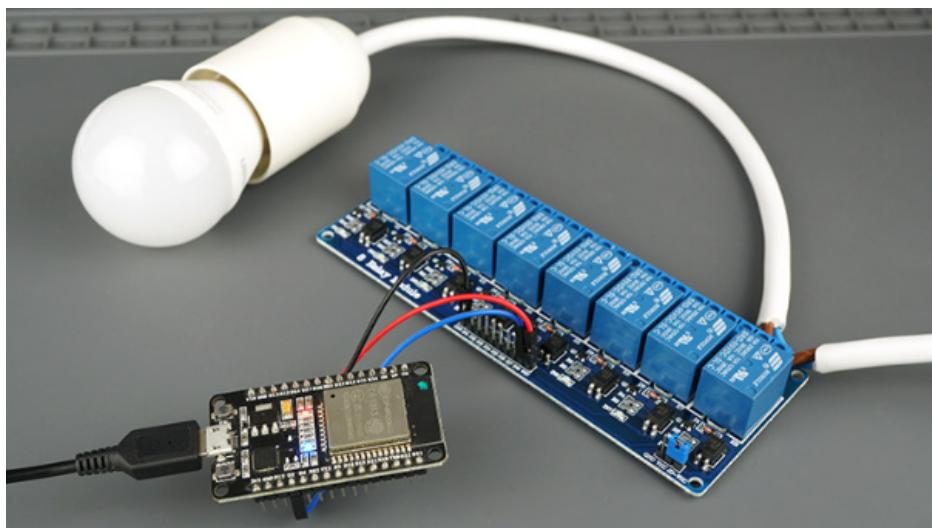
Atau, Kami dapat menggunakan sumber daya 12V untuk mengontrol peralatan 12V.



Untuk contoh yang disuguhkan adalah sistem kontrol sebuah lampu. Tujuannya hanya ingin menyalakan lampu sesekali, jadi lebih baik menggunakan konfigurasi yang biasanya terbuka. Kami menghubungkan pin IN 1 ke **GPIO 26**. Anda dapat menggunakan GPIO lain yang sesuai. Lihat [Panduan Referensi GPIO ESP32](#).

7.1.8 Mengontrol Modul Relay dengan ESP32 – Arduino Sketch

Kode untuk mengontrol relay dengan ESP32 semudah mengontrol LED atau output lainnya. Dalam contoh ini, karena kita menggunakan konfigurasi yang biasanya terbuka, kita perlu mengirim sinyal RENDAH untuk membiarkan arus mengalir, dan sinyal TINGGI untuk menghentikan aliran arus.



Kode berikut akan menyalakan lampu Kami selama 10 detik dan mematikannya selama 10 detik lagi.

```
/*
Bismillahirrohmanirrohim :)
Relay WebServer
***** */

const int relay = 25;

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(3000);

    // 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(3000);
}
```

7.1.9 Cara Kerja Kode

Tentukan pin yang terhubung dengan pin IN relay.

```
const int relay = 25;
```

Dalam `setup()`, tentukan relay sebagai keluaran.

```
pinMode(relay, OUTPUT);
```

Dalam `loop()`, Kirim `LOW` signal untuk membiarkan arus mengalir dan menyalakan lampu.

```
digitalWrite(relay, LOW);
```

Jika Kami menggunakan konfigurasi yang biasanya tertutup, kirim `HIGH` signal untuk menyalakan lampu. Kemudian, tunggu 3 detik.

```
delay(3000);
```

Hentikan aliran arus dengan mengirimkan `HIGH` signal ke pin relay. Jika Kami menggunakan konfigurasi yang biasanya tertutup, kirim `LOW` signal untuk menghentikan aliran arus.

```
digitalWrite(relay, HIGH);
```

7.1.10 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 apakah mereka dikonfigurasi sebagai biasanya dibuka atau ditutup secara normal. Kita hanya perlu mengubah beberapa baris kode untuk menentukan jumlah relay yang ingin Kami kendalikan dan penetapan pin. Untuk referensi dalam pembuatan server web ini, kami menggunakan [ESPAsyncWebServer library](#).

7.1.11 Memasang ESPAsyncWebServer library

Untuk memasang ESPAsyncWebServer library, berikut langkah-langkah selanjutnya untuk menginstal ESPAsyncWebServer library :

- Klik [di sini](#) untuk mengunduh ESPAsyncWebServer library. Kami harus memiliki folder .zip di folder Unduhan Kami
- Buka zip folder .zip dan Kami akan mendapatkan folder *ESPAsyncWebServer-master*
- Ganti nama folder Kami dari *ESPAsyncWebServer-master* menjadi *ESPAsyncWebServer*
- Pindahkan folder *ESPAsyncWebServer* ke folder Arduino IDE installation libraries

Atau, di Arduino IDE, bisa di cek ke **Sketch > Include Library > Add .ZIP library...** dan pilih library yang baru saja diunduh.

Setelah menginstal perlibraryan yang diperlukan, salin kode berikut ke Arduino IDE yang digunakan.

```

    .// Import required libraries #include "WiFi.h"
#include "ESPAsyncWebServer.h"
// Set to true to define Relay as Normally Open (NO)
#define RELAY_NO      true
// Set number of relays
#define NUM_RELAYS   5
// Assign each GPIO to a relay
int relayGPIOs[NUM_RELAYS] = {2, 26, 27, 25, 33};
// Replace with your network credentials const
char* ssid = "Ardikostwifi_Ext";

const char* password = "Ardikost0041"; const char*
PARAM_INPUT_1 = "relay";
const char* PARAM_INPUT_2 = "state";
// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>

    <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}

    .slider {position: absolute; top: 0; left: 0;
width: 100%; height: 100%}
    .slider::before {content: '';
width: 60px; height: 100%; background-color: #ccc;
border-radius: 50%; position: absolute; left: -30px; top: 0;
background-color: white; transition: width 0.4s}
    .switch input:checked + .slider::before {width: 0;
background-color: #ccc; border-radius: 0; left: 50%; top: 0}
    .switch input:checked ~ .label {color: #007bff; font-weight: bold;
font-size: 1.2em; position: absolute; left: 50%; top: 50%; transform: translate(-50%, -50%)}
    .label {position: absolute; left: 50%; top: 50%; transform: translate(-50%, -50%); color: #000; font-size: 0.8em; font-weight: bold; text-align: center; width: 100px; height: 20px; line-height: 20px; background-color: white; border: 1px solid #ccc; border-radius: 10px; padding: 0 5px; transition: all 0.3s}
    .switch input:checked ~ .label {color: #007bff; font-weight: bold;
font-size: 1.2em; position: absolute; left: 50%; top: 50%; transform: translate(-50%, -50%)}
)rawliteral";

```

```

        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?relay="+element.id+"&state=1", true); }
    else { xhr.open("GET",
"/update?relay="+element.id+"&state=0", true); }
    xhr.send();
}</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 ="";
        for(int i=1; i<=NUM_RELAYS; i++) {
            String relayStateValue = relayState(i);
            buttons+= "<h4>Relay #" + String(i) + " - GPIO "
+ relayGPIOs[i-1] + "</h4><label
class=\"switch\"><input type=\"checkbox\""
onchange="toggleCheckbox(this)" id=\"" + String(i) +

```

```

"\" + relayStateValue + "><span
class=\"slider\">" </span> </label>";
}

return buttons;
}

return String();
String relayState(int numRelay) {
    if(RELAY_NO) {
        if(digitalRead(relayGPIOs [numRelay-1])) {
            return "";
        }
        else {
            return "checked";
        }
    }
    else {
        if(digitalRead(relayGPIOs [numRelay-1])) {
            return "checked";
        }
        else {
            return "";
        }
    }
    return "";
}

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

    // Set all relays to off when the program starts - if set
    // to Normally Open (NO), the relay is off when you set the
    // relay to HIGH
    for(int i=1; i<=NUM_RELAYS; i++) {
        pinMode(relayGPIOs[i-1], OUTPUT);
        if(RELAY_NO) {
            digitalWrite(relayGPIOs[i-1], HIGH);
        }
    }
}

```

```

    else{
        digitalWrite(relayGPIOs[i-1], LOW);
    }
}

// Connect to Wi-Fi
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
}

// Print ESP32 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?relay=<inputMessage>&state=<inputMessage2>
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
String inputMessage2;
    String inputParam2;
    // GET input1 value on <ESP_IP>/update?relay=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1) &
request->hasParam(PARAM_INPUT_2)) {

```

```

    inputMessage           =
request->getParam(PARAM_INPUT_1)->value();

    inputParam = PARAM_INPUT_1;

                                inputMessage2      =
request->getParam(PARAM_INPUT_2)->value();
    inputParam2 = PARAM_INPUT_2;
    if(RELAY_NO) {
        Serial.print("NO ");
digitalWrite(relayGPIOs[inputMessage.toInt()-1],
!inputMessage2.toInt());
    }
    else{
        Serial.print("NC ");

digitalWrite(relayGPIOs[inputMessage.toInt()-1],
inputMessage2.toInt());
    }
}
else {
    inputMessage = "No message sent";
    inputParam = "none";
}
Serial.println(inputMessage + inputMessage2);
request->send(200, "text/plain", "OK");
} );
// Start server
server.begin();
}

void loop() {
}

```

7.1.12 Tentukan Konfigurasi Relay

Ubah variabel berikut untuk menunjukkan apakah akan menggunakan relay dalam konfigurasi biasanya terbuka (NO) atau biasanya tertutup (NC). Mengatur RELAY_NO variabel ke BENAR untuk os yang biasanya terbuka disetel ke Salah untuk biasanya tertutup.

```
#define RELAY_NO true
```

7.1.13 Tentukan Jumlah Relay (Saluran)

Dalam penentuan jumlah relay ini dapat ditentukan sesuai yang ingin dikendalikan pada NUM_RELAYS variabel. Untuk tujuan demonstrasi, kami menyetelnya ke 5

```
#define NUM_RELAYS 5
```

7.1.14 Tentukan Tugas Pin Relay

Dalam variabel array berikut, Anda dapat menentukan GPIO ESP32 yang akan mengontrol relay:

```
int relayGPIOs[NUM_RELAYS] = {2, 26, 27, 25, 33};
```

Jumlah relay yang disetel pada NUM_RELAYS variabel harus sesuai dengan jumlah GPIO yang ditetapkan dalam relay GPIO Himpunan.

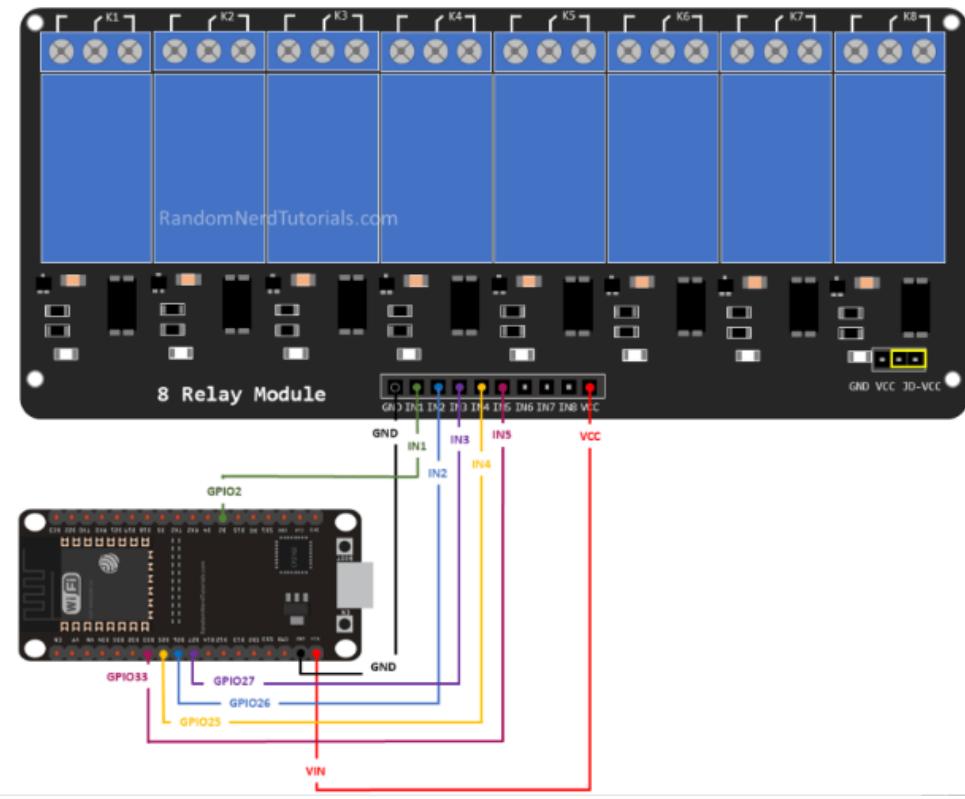
7.1.15 Kredensial Jaringan

Masukkan kredensial jaringan kedalam variabel berikut.

```
const char* ssid      = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";
```

7.1.16 Pengkabelan 8 Saluran Relay ke ESP32

Untuk tujuan demonstrasi, kami mengontrol 5 saluran relai. Sambungkan ESP32 ke modul relay seperti yang ditunjukkan pada diagram skema berikut



7.1.17 Demonstrasi

Setelah melakukan perubahan yang diperlukan, unggah kode ke ESP32. Kemudian buka Serial Monitor pada baud rate 115200 dan tekan tombol ESP32 EN untuk mendapatkan alamat IP-nya. Kemudian, buka browser di jaringan lokal dan ketik alamat IP ESP32 untuk mendapatkan akses ke server web. Output dari program ini yaitu harus mendapatkan sesuatu sebagai laman yang berisi tombol sebanyak jumlah relay yang telah Anda tentukan dalam kode Anda.

BAB VIII

ESP32 Servo

8.1.1 Connecting the Servo Motor to the 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. Saat menggunakan servo kecil seperti S0009 seperti yang ditunjukkan pada gambar di bawah, Anda dapat menyalakannya langsung dari ESP32.



Jika Anda menggunakan servo kecil seperti S0009, Anda perlu menghubungkan:

- GND -> ESP32 **GND** pin;
- Power -> ESP32 **VIN** pin;
- Signal -> **GPIO 13** (or any PWM pin).

8.1.2 Schematic

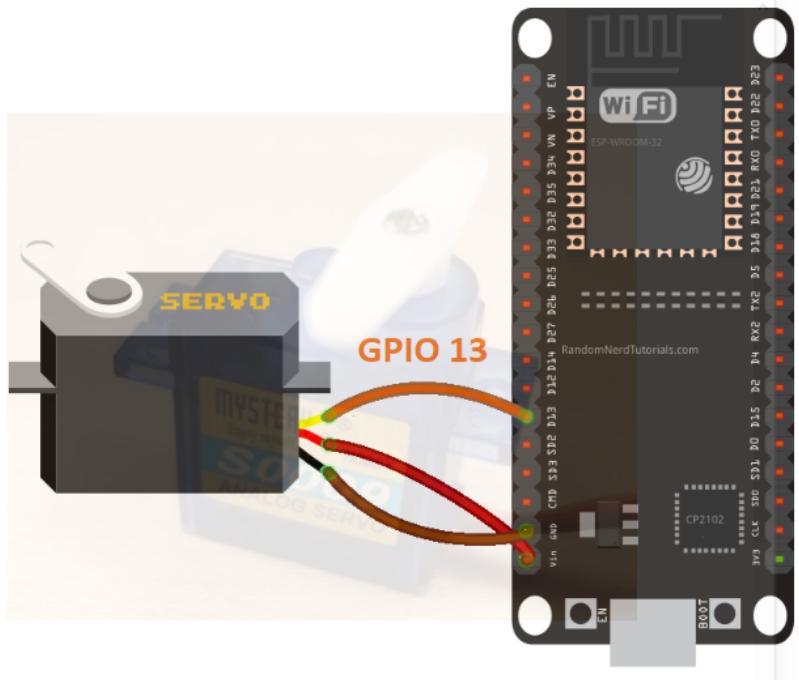
Dalam contoh kami, kami akan menghubungkan kabel sinyal ke GPIO 13. Jadi, dapat mengikuti diagram skema berikutnya untuk menyambungkan motor servo.

8.1.3 Installing the ESP32_Arduino_Servo_Library

Perpustakaan Servo Arduino ESP32 memudahkan untuk mengontrol motor servo dengan ESP32, menggunakan Arduino IDE. Ikuti langkah-langkah selanjutnya untuk menginstal perpustakaan di Arduino IDE:

1. Klik [di sini](#) untuk mengunduh **ESP32_Arduino_Servo_Library**. Anda harus memiliki folder .zip di folder Unduhan Anda

2. Buka zip folder .zip dan Anda akan mendapatkan folder ESP32-Arduino-Servo-Library-Master
3. Ganti nama folder Anda dari ESP32-Arduino-Servo-Library-Master menjadi ESP32_Arduino_Servo_Library
4. Pindahkan folder ESP32_Arduino_Servo_Library ke folder library instalasi Arduino IDE anda
5. Terakhir, buka kembali Arduino IDE Anda.



8.1.4 Kode Program

```
*****
Bismillahirrohmanirrohim :)
ESP32 Servo
*****/

#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(5); // attaches the servo on pin 5 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(20); // 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(20); // waits 15ms for
the servo to reach the position
    }
}
```

BAB IX

ESP32 Output WebServer

9.1.1 Project Overview

Sebelum langsung ke proyek, penting untuk menguraikan apa yang akan dilakukan server web, sehingga lebih mudah untuk mengikuti langkah-langkahnya nanti.

- Server web yang akan Anda buat mengontrol dua LED yang terhubung ke ESP32 GPIO 12 dan GPIO 13;
- Selanjutnya mengakses server web ESP32 dengan mengetikkan alamat IP ESP32 pada browser di jaringan lokal;
- Dengan mengklik tombol di server web, Anda dapat langsung mengubah status setiap LED.

9.1.2 Schematic

Mulailah dengan membangun sirkuit. Hubungkan dua LED ke ESP32 yaitu satu LED terhubung ke GPIO 12, dan lainnya ke GPIO 13.

9.1.3 Kode Program

```
*****
Bismillahirrohmanirrohim :)
*****  
  
// Load Wi-Fi library
#include <WiFi.h>  
  
// Replace with your network credentials
const char* ssid = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";  
  
// Set web server port number to 80
WiFiServer server(80);  
  
// Variable to store the HTTP request
String header;  
  
// Auxiliar variables to store the current output state
String output12State = "off";
String output13State = "off";  
  
// Assign output variables to GPIO pins
```

```
const int output12 = 12;
const int output13 = 13;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 5000ms = 5s)
const long timeoutTime = 5000;

void setup() {
    Serial.begin(115200);
    // Initialize the output variables as outputs
    pinMode(output12, OUTPUT);
    pinMode(output13, OUTPUT);
    // Set outputs to LOW
    digitalWrite(output12, LOW);
    digitalWrite(output13, 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 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();

                // turns the GPIOs on and off
                if (header.indexOf("GET /12/on") >= 0) {
                    Serial.println("GPIO 12 on");
                    output12State = "on";
                    digitalWrite(output12, HIGH);
                } else if (header.indexOf("GET /12/off") >= 0) {
                    Serial.println("GPIO 12 off");
                    output12State = "off";
                    digitalWrite(output12, LOW);
                } else if (header.indexOf("GET /13/on") >= 0) {
                    Serial.println("GPIO 13 on");
                    output13State = "on";
                    digitalWrite(output13, HIGH);
                } else if (header.indexOf("GET /13/off") >= 0) {
                    Serial.println("GPIO 13 off");
                    output13State = "off";
                    digitalWrite(output13, 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 12
client.println("<p>GPIO 12 - State " + output12State + "</p>");
        // If the output12State is off, it displays the ON button
if (output12State=="off") {
                client.println("<p><a href=\"/12/on\"><button
class=\"button\">ON</button></a></p>");
} else {
                client.println("<p><a href=\"/12/off\"><button class=\"button
button2\">OFF</button></a></p>");
}

        // Display current state, and ON/OFF buttons for GPIO 13
client.println("<p>GPIO 13 - State " + output13State + "</p>");
        // If the output13State is off, it displays the ON button
if (output13State=="off") {
                client.println("<p><a href=\"/13/on\"><button
class=\"button\">ON</button></a></p>");
} else {
                client.println("<p><a href=\"/13/off\"><button class=\"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("");
}
}

```

9.1.4 Hasil Praktek

Not secure | 192.168.0.177/12/on

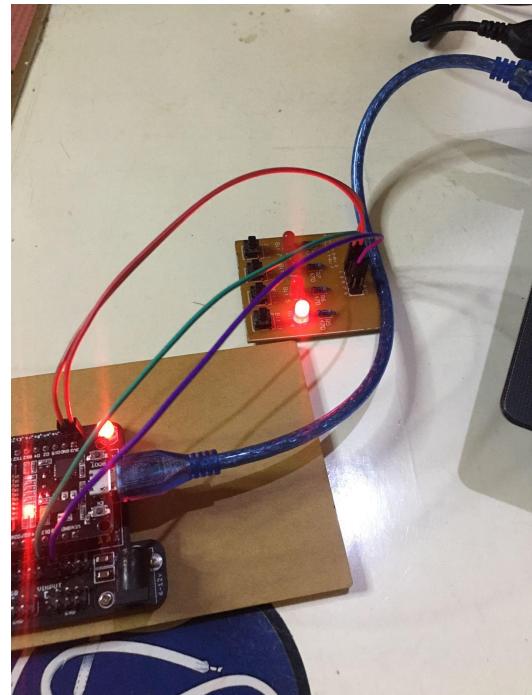
ESP32 Web Server

GPIO 12 - State on

OFF

GPIO 13 - State off

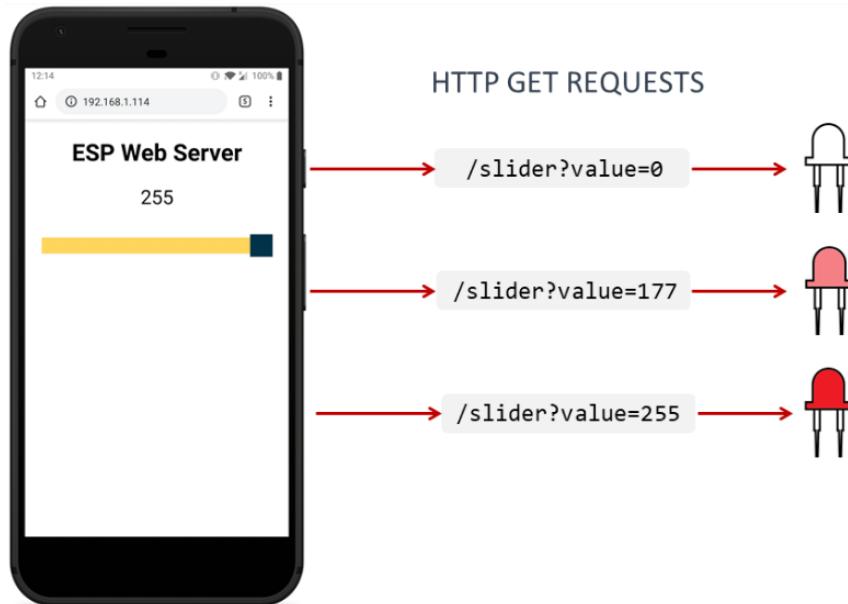
ON



BAB X

ESP32 Web Server with Slider

10.1 Ulasan Web Server dengan Slider



- ESP32 menghosting server web yang menampilkan halaman web dengan penggeser;
- Saat Anda memindahkan penggeser, Anda membuat permintaan HTTP ke ESP32 dengan nilai penggeser baru;
- Permintaan HTTP datang dalam format berikut: DAPATKAN/penggeser?nilai=SLIDERVALUE, di mana NILAI SLIDER adalah angka antara 0 dan 255. Anda dapat memodifikasi slider Anda untuk memasukkan rentang lainnya;
- Dari permintaan HTTP, ESP32 mendapatkan nilai slider saat ini;
- 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.

10.2 Prasarat

Sebelum melanjutkan dengan proyek ini, pastikan Anda memeriksa prasyarat berikut.

10.2.1.Arduino IDE

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

10.2.2 Pustaka Server Web Async

Kami akan membangun server web menggunakan pustaka berikut:

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

Library ini tidak tersedia untuk diinstal melalui Arduino Library Manager, jadi untuk menggunakannya perlu menyalin file library ke folder Library Instalasi Arduino. Atau, di Arduino IDE,buka **Sketch > Include Library > Add .zip Library** dan pilih library yang baru saja diunduh.

10.2.3 Kode

Kode berikut mengontrol kecerahan LED bawaan ESP32 menggunakan penggeser di server web. Dengan kata lain, user dapat mengubah siklus tugas PWM dengan penggeser. Ini dapat berguna untuk mengontrol kecerahan LED atau mengontrol motor servo, misalnya.

```
*****
Bismillahirrohmanirrohim :)
*****  
  
// Import required libraries
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>
// Replace with your network credentials
const char* ssid = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";  
  
const int output = 22;  
  
String sliderValue = "0";  
  
// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
```

```

const int resolution = 8;

const char* PARAM_INPUT = "value";

// Create AsyncWebServer object on port 80
AsyncWebServer 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: #5CF4FF;
              outline: none; -webkit-transition: .2s; transition: opacity .2s;}
    .slider::-webkit-slider-thumb { -webkit-appearance: none; appearance: none; width: 35px; height: 35px; background: #490000; cursor: pointer;}
    .slider::-moz-range-thumb { width: 35px; height: 35px; background: #490000; 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 section in 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 functionalities
    ledcSetup(ledChannel, freq, resolution);

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

    ledcWrite(ledChannel, sliderValue.toInt());

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.println("Connecting to WiFi..");
    }

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

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

10.2.4 Cara Kerja Kode

- ❖ Mengimpor perpustakaan

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

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

- ❖ Mengatur kredensial jaringan Anda

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

```
const char* ssid = "Ardikostwifi_Ext";  
const char* password = "Ardikost0041";
```

- ❖ Definisi variabel

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

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

- ❖ Setel Properti PWM

Baris berikut mendefinisikan properti PWM untuk mengontrol LED.

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

Dalam sistem ini menggunakan resolusi 8-bit, yang berarti kecerahan LED dapat dikontrol menggunakan nilai dari 0 hingga 255.

❖ Parameter Masukan

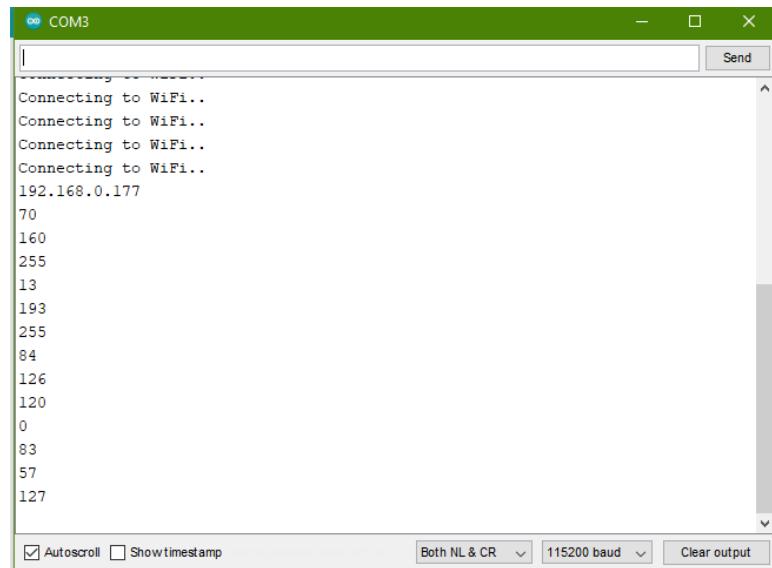
`PARAM_INPUT` variabel akan digunakan untuk "mencari" nilai slider pada permintaan yang diterima oleh ESP32 saat slider dipindahkan. (Ingat: ESP32 akan menerima permintaan seperti ini DAPATKAN/penggeser?nilai=SLIDERVALUE)

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

Ini akan mencari `nilai` pada URL dan dapatkan nilai yang diberikan padanya.

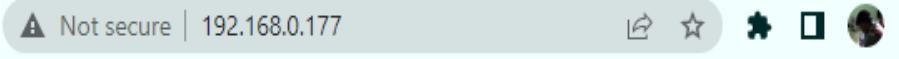
10.2.5 Hasil Running Program

a. Kondisi slider pada posisi 127



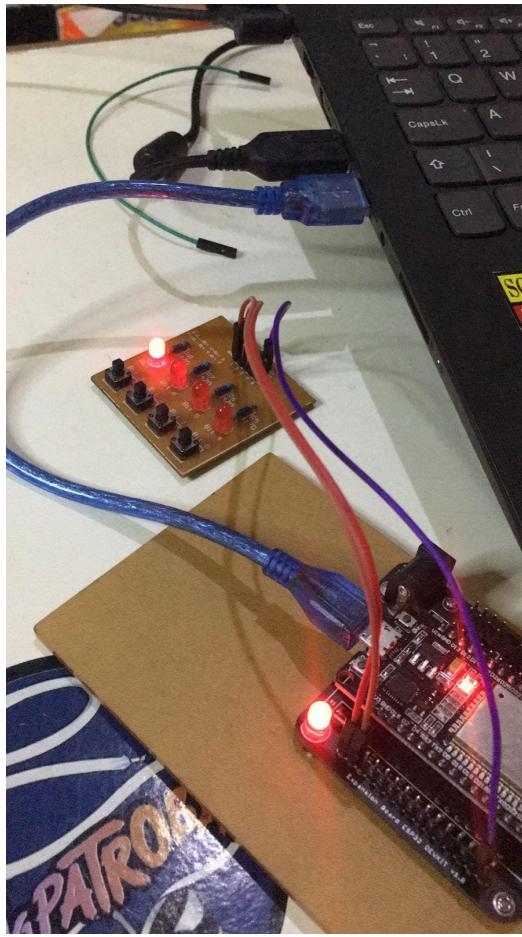
```
Connecting to WiFi...
Connecting to WiFi...
Connecting to WiFi...
Connecting to WiFi...
192.168.0.177
70
160
255
13
193
255
84
126
120
0
83
57
127
```

Autoscroll Showtimestamp Both NL & CR 115200 baud Clear output

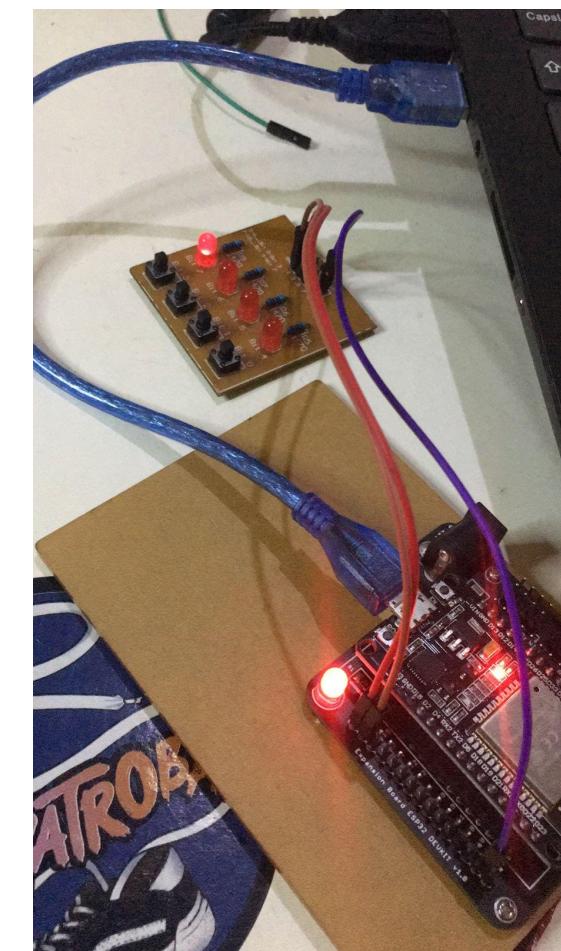


ESP Web Server

127



- b. Kondisi slider pada posisi



BAB XI

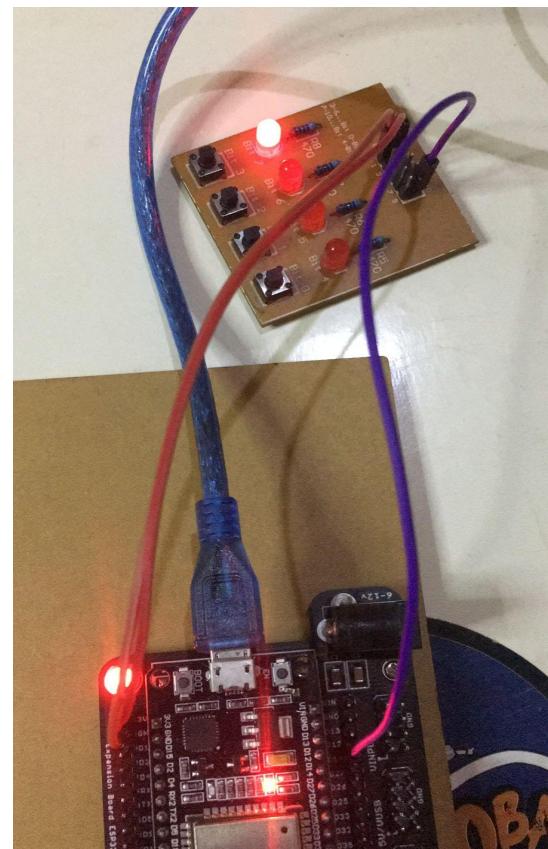
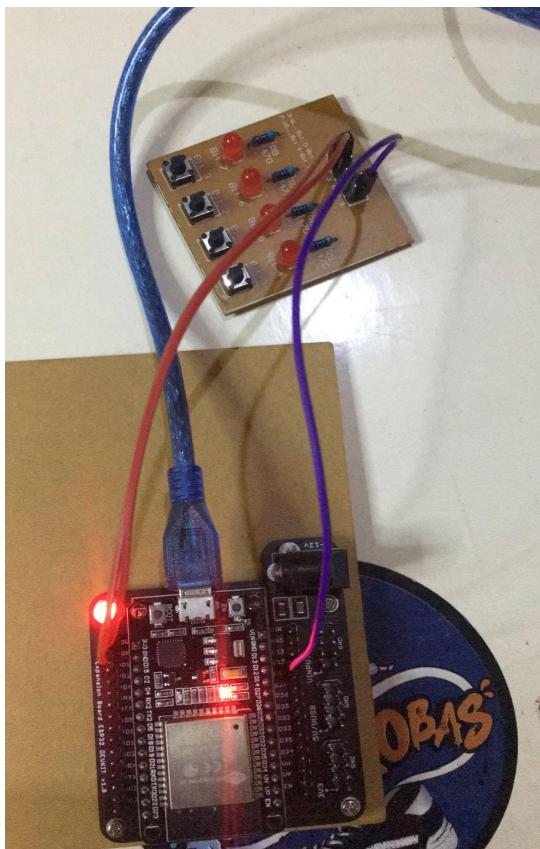
ESP32 Momentary Switch Web Server

11.1.1 Control Outputs with Momentary Switch

Pada sistem ini menunjukkan cara membuat server web dengan tombol yang berfungsi sebagai saklar sesaat (*Momentary*) untuk mengontrol output ESP32 atau ESP8266 dari jarak jauh. Status output TINGGI jika selama button terus ditekan di halaman web. Setelah Anda melepaskannya, itu berubah menjadi RENDAH/mati. Sebagai contoh, kami akan mengontrol LED.

11.1.2 Project Overview

- ESP32 atau ESP8266 meng-hosting server web yang dapat kami akses untuk mengontrol output;
- Status default output adalah RENDAH, tetapi anda dapat mengubahnya tergantung pada aplikasi proyek Anda;
- Ada tombol yang berfungsi seperti saklar sesaat:
 - Jika saat menekan tombol, output berubah statusnya menjadi TINGGI selama terus menahan tombol;
 - Setelah tombol dilepaskan, status output kembali ke LOW.



11.1.3 Kode Program

```
*****
Bismillahirrohmanirrohim :)
*****



#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 = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";

const int output = 12;

// 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: #ba8f18;
border: none;
border-radius: 5px;
box-shadow:#ba8f18 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: #2d2e16}
    .button:active {
        background-color: #2d2e16;
        box-shadow: 0 4px #c7c479;
        transform: translateY(2px);
    }
</style>
</head>
<body>
    <h1>ESP Pushbutton Web Server</h1>
    <button class="button" onmousedown="toggleCheckbox('off');"
ontouchstart="toggleCheckbox('off');" onmouseup="toggleCheckbox('on');"
ontouchend="toggleCheckbox('on');">LED PUSHBUTTON</button>
    <script>
        function toggleCheckbox(x) {
            var xhr = new XMLHttpRequest();
            xhr.open("GET", "/" + x, true);
            xhr.send();
        }
    </script>
</body>
</html>)rawliteral";

void notFound(AsyncWebRequest *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() {
```

BAB XII

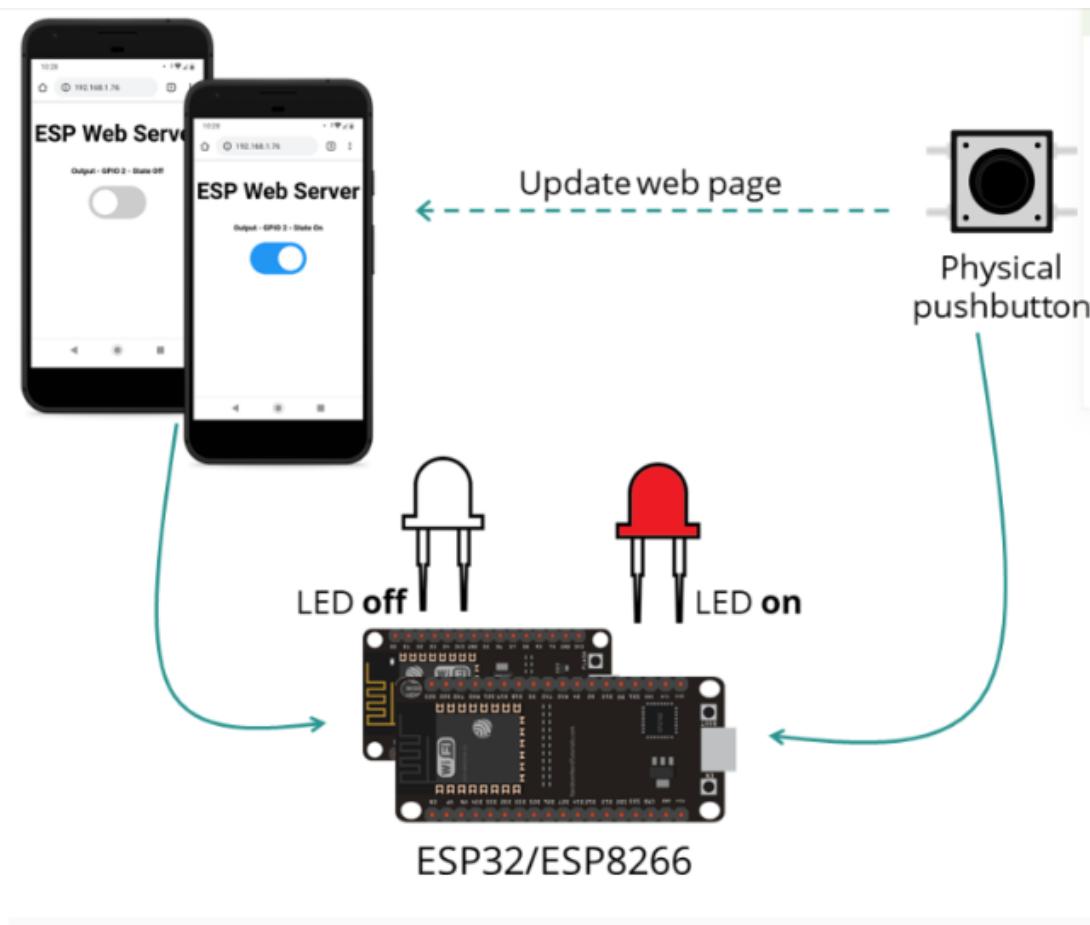
ESP32 Physical Button WebServer

12.1.1 Control Outputs with Web Server and a Physical Button Simultaneously

Pada bab ini kami menunjukkan cara mengontrol output ESP32 atau ESP8266 menggunakan server web dan tombol fisik secara bersamaan. Status keluaran diperbarui pada halaman web apakah itu diubah melalui tombol fisik atau server web.

12.1.2 Project Overview

Mari kita lihat sekilas bagaimana proyek ini bekerja.



- ESP32 atau ESP8266 menghosting server web yang memungkinkan Anda mengontrol status keluaran;
- Status keluaran saat ini ditampilkan di server web;
- ESP juga terhubung ke tombol fisik yang mengontrol output yang sama;

- Jika ingin mengubah status keluaran menggunakan tombol push fisik, status saat ini juga diperbarui di server web.

12.1.3 Kode Program

```

*****
Bismillahirrohmanirrohim :)
*****


// Import required libraries
#ifndef ESP32
    #include <WiFi.h>
    #include <AsyncTCP.h>
#else
    #include <ESP32WiFi.h>
    #include <AsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char* ssid = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";


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}
  .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: #030303;
-webkit-transition: .4s; transition: .4s; border-radius: 68px}
  input:checked+.slider {background-color: #82f238}
  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 == 0){
        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:
    }
}

```

```

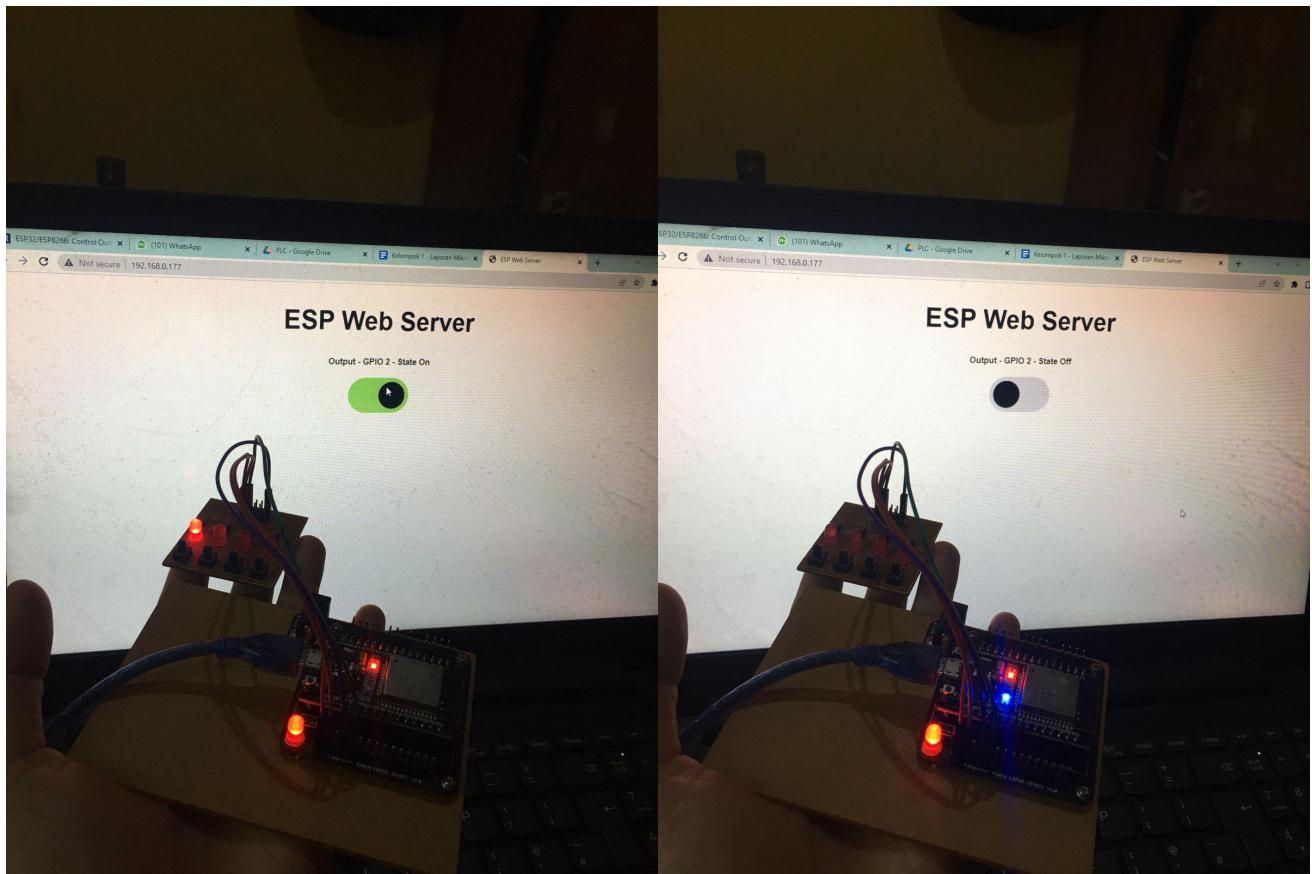
if (reading != buttonState) {
    buttonState = reading;

    // 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;
}

```



BAB XIII

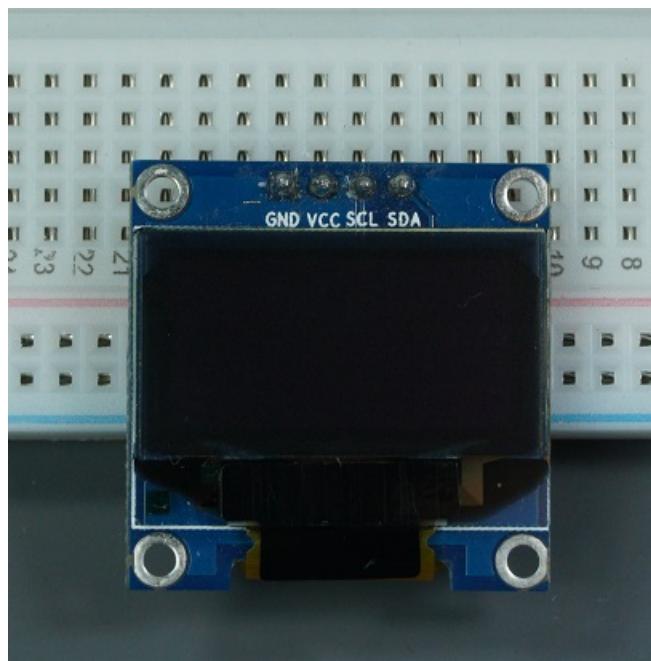
ESP32 Oled Display

13.1.1 ESP32 OLED Display with Arduino IDE

Pada kasus ini kami menunjukkan cara menggunakan layar OLED SSD1306 0,96 inci dengan ESP32 menggunakan Arduino IDE. Kami akan menunjukkan cara menulis teks, mengatur font yang berbeda, menggambar bentuk dan menampilkan gambar bitmap. Namun kami hanya memakai mode gambar bitmap.

13.1.2 Introducing 0.96 inch OLED Display

Layar [OLED](#) yang akan kami gunakan dalam tutorial ini adalah model SSD1306: layar 0,96 inch monicolor dengan 128x64 piksel seperti yang ditunjukkan pada gambar berikut. Layar OLED tidak memerlukan lampu latar, yang menghasilkan kontras yang 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.



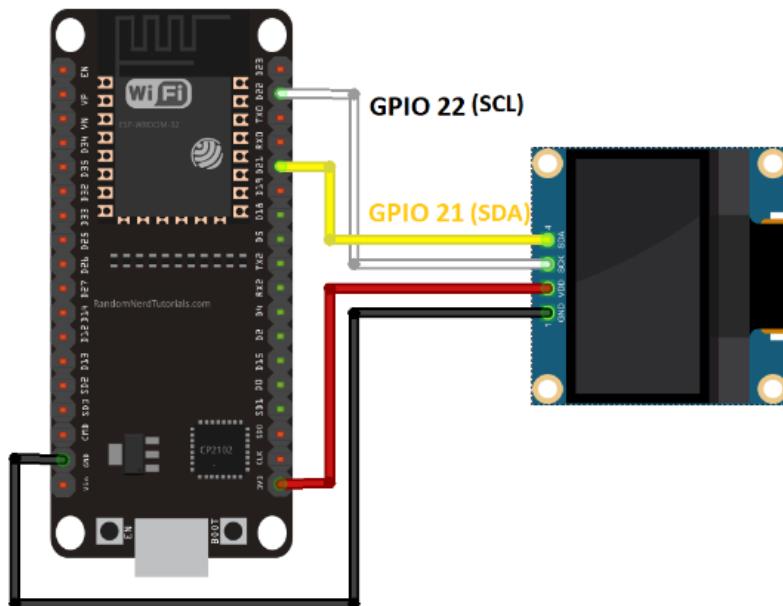
Model yang kami gunakan memiliki empat pin dan berkomunikasi dengan mikrokontroler apapun menggunakan protokol komunikasi I2C seperti pada gambar diatas. Ada model yang datang dengan pin RESET ekstra atau yang berkomunikasi menggunakan protokol komunikasi SPI.

13.1.3 OLED Display SSD1306 Pin Wiring

Karena layar OLED menggunakan protokol komunikasi I2C, pengkabelan menjadi sangat sederhana. Kami menggunakan tabel berikut sebagai referensi.

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

Atau, Anda dapat mengikuti diagram skema berikutnya untuk menyambungkan ESP32 ke layar OLED.



13.1.4 Kode Program

```
*****
 * Bismillahirrohmanirrohim :)
 * ESP32 Oled Display
****/

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

#define SCREEN_WIDTH 231
#define SCREEN_HEIGHT 320

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
static const uint8_t image_data_Image[9280] = { masukkan kode gambar bitmaps
};

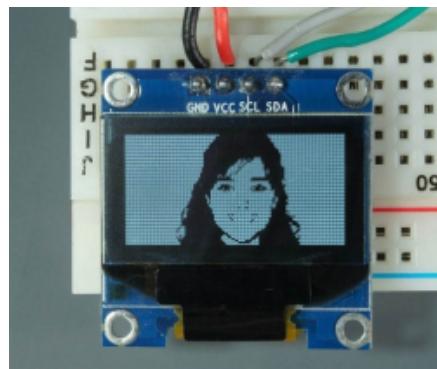
void setup() {
    Serial.begin(115200);

    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000); // Pause for 2 seconds

    // Clear the buffer.
    display.clearDisplay();

    // Draw bitmap on the screen
    display.drawBitmap(0, 0, image_data_Saraarray, 231, 320, 1);
    display.display();
}

void loop() {
}
```



BAB XIV

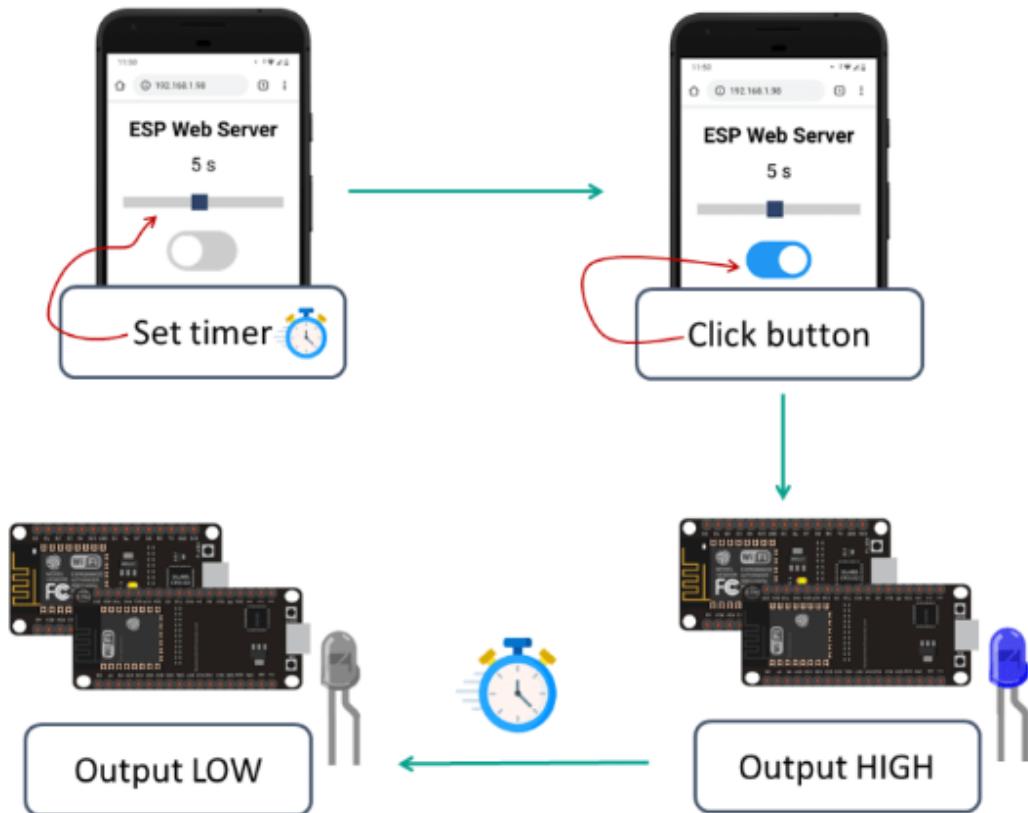
ESP32 Timer/Pulse WebServer

14.1.1 Control Outputs with Timer

Pada kasus ini kami akan membangun server web untuk mengontrol output NodeMCU ESP32 atau ESP8266 dengan pulsa menggunakan Arduino IDE. Lebar pulsa (“timer”) dapat diatur menggunakan slider di halaman web. Saat Anda mengklik tombol ON, ESP menetapkan status output ke TINGGI untuk jumlah detik yang ditentukan di penggeser. Ini dapat berguna secara khusus untuk mengontrol peranti yang membutuhkan sinyal TINGGI selama beberapa detik yang telah ditentukan untuk digerakkan.

14.1.2 Project Overview

Gambar berikut menunjukkan gambaran umum tentang cara kerja proyek ini.



- ESP32/ESP8266 menghosting server web yang memungkinkan Anda mengontrol output dengan pulsa;
- Server web berisi penggeser yang memungkinkan Anda menentukan lebar pulsa (berapa detik output harus TINGGI);
- Ada tombol ON/OFF. Atur 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.

14.1.3 Kode Program

```

*****
Bismillahirrohmanirrohim :)
Timer/Pulse WebServer
*****/

// Import required libraries
#ifndef ESP32
#include <WiFi.h>
#include <AsyncTCP.h>
#else
#include <ESP32WiFi.h>
#include <AsyncTCP.h>
#endif
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char* ssid = "Ardikostwifi_Ext";
const char* password = "Ardikost0041";

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

const int output = 4;

String timerSliderValue = "10";

// Create AsyncWebServer object on port 80

```

```

AsyncWebServer 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: 2.4rem;}
        p {font-size: 2.2rem;}
        body {max-width: 600px; margin:0px auto; padding-bottom: 25px;}
        .switch {position: relative; display: inline-block; width: 120px; height: 68px}
        .switch input {display: none}
        .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: #520638; -webkit-transition: .4s; transition: .4s; border-radius: 68px}
        input:checked+.slider {background-color: #c71a8d}
        input:checked+.slider:before {-webkit-transform: translateX(52px); -ms-transform: translateX(52px); transform: translateX(52px)}
        .slider2 { -webkit-appearance: none; margin: 14px; width: 300px; height: 20px; background: #ccc;
            outline: none; -webkit-transition: .2s; transition: opacity .2s;}
        .slider2::-webkit-slider-thumb {-webkit-appearance: none; appearance: none; width: 30px; height: 30px; background: #c71a8d; cursor: pointer;}
        .slider2::-moz-range-thumb { width: 30px; height: 30px; background: #c71a8d; cursor: pointer; }
    </style>
</head>
<body>
    <h2>ESP Web Server</h2>
    <p><span id="timerValue">%TIMERVALUE%</span> s</p>
    <p><input type="range" onchange="updateSliderTimer(this)" id="timerSlider" min="2" max="15" value="%TIMERVALUE%" step="1" class="slider2"></p>
    %BUTTONPLACEHOLDER%
<script>
function toggleCheckbox(element) {
    var sliderValue = document.getElementById("timerSlider").value;
    var xhr = new XMLHttpRequest();
    if(element.checked){ xhr.open("GET", "/update?state=1", true); xhr.send();
        var count = sliderValue, timer = setInterval(function() {
            count--; document.getElementById("timerValue").innerHTML = count;
    })
}

```

```

        if(count == 0){ clearInterval(timer);
document.getElementById("timerValue").innerHTML =
document.getElementById("timerSlider").value; }
}, 1000);
sliderValue = sliderValue*1000;
setTimeout(function(){ xhr.open("GET", "/update?state=0", true);
document.getElementById(element.id).checked = false; xhr.send(); },
sliderValue);
}
}
function updateSliderTimer(element) {
var sliderValue = document.getElementById("timerSlider").value;
document.getElementById("timerValue").innerHTML = sliderValue;
var xhr = new XMLHttpRequest();
xhr.open("GET", "/slider?value="+sliderValue, true);
xhr.send();
}
</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+= "<p><label class=\"switch\"><input type=\"checkbox\""
onchange="toggleCheckbox(this)" id="output" " + outputStateValue + "><span"
class="slider"></span></label></p>";
return buttons;
}
else if(var == "TIMERVALUE"){
return timerSliderValue;
}
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);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(3000);
        Serial.println("Connecting to WiFi..");
    }

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

    // Route for root / web page
    server.on("/", HTTP_GET, [](AsyncWebRequest *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, [] (AsyncWebRequest *request) {
        String inputMessage;
        // GET input1 value on <ESP_IP>/update?state=<inputMessage>
        if (request->hasParam(PARAM_INPUT_1)) {
            inputMessage = request->getParam(PARAM_INPUT_1)->value();
            digitalWrite(output, inputMessage.toInt());
        }
        else {
            inputMessage = "No message sent";
        }
        Serial.println(inputMessage);
        request->send(200, "text/plain", "OK");
    });

    // Send a GET request to <ESP_IP>/slider?value=<inputMessage>
    server.on("/slider", HTTP_GET, [] (AsyncWebRequest *request) {

```

```
String inputMessage;
// GET input1 value on <ESP_IP>/slider?value=<inputMessage>
if (request->hasParam(PARAM_INPUT_2)) {
    inputMessage = request->getParam(PARAM_INPUT_2)->value();
    timerSliderValue = inputMessage;
}
else {
    inputMessage = "No message sent";
}
Serial.println(inputMessage);
request->send(200, "text/plain", "OK");
});

// Start server
server.begin();
}

void loop() {

}
```

14.1.4 Hasil Pratikum

