Modul Pelatihan

Microcontroller

ECS Laboratory

Daftar Isi

Daftar Gambar

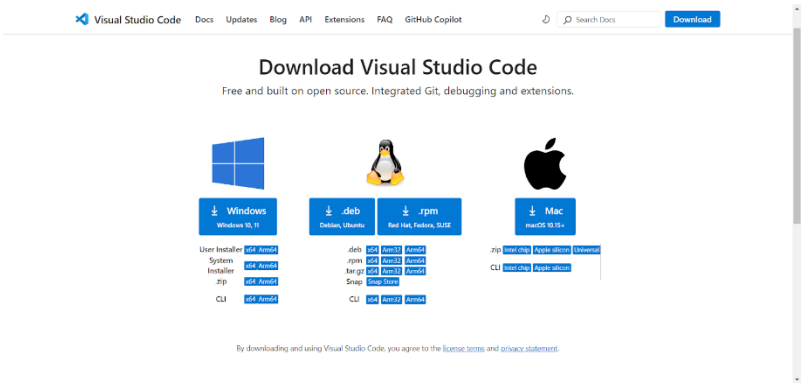
Daftar Tabel

BAB I
TAHAP PERSIAPAN

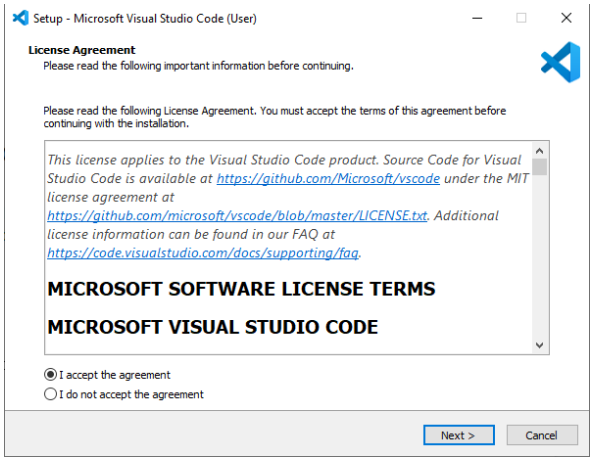
1.1. Instalasi VSCode

Visual Studio Code adalah perangkat lunak penyunting kode-sumber atau *text editor* buatan Microsoft untuk Linux, macOS, dan Windows. VSCode adalah *text editor* paling populer sekarang.

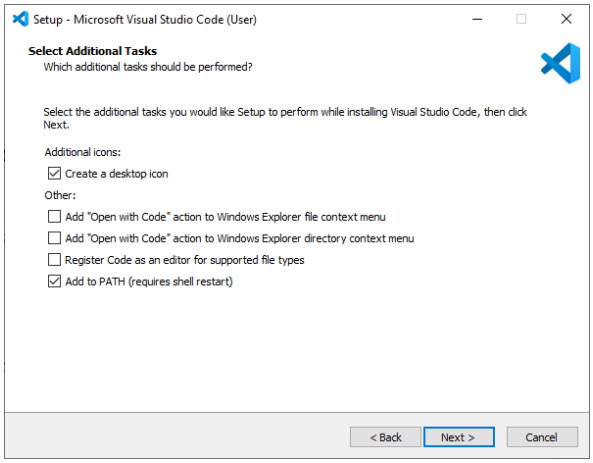
1. Kunjungi https://code.visualstudio.com/ dan unduh versi stabil untuk sistem operasi. (Windows).



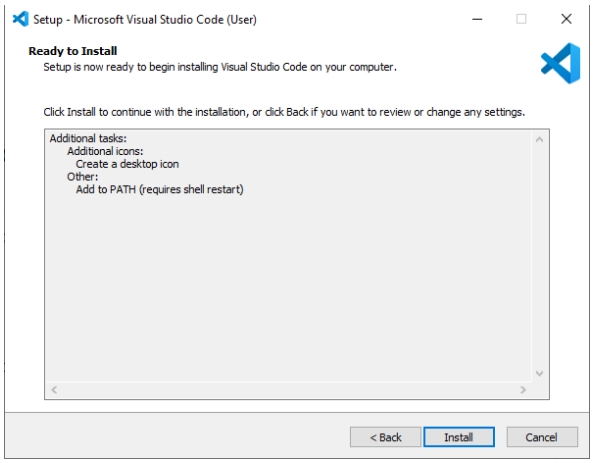
1. Klik pada *wizard* instalasi untuk memulai instalasi dan ikuti semua langkah untuk menyelesaikan instalasi. Terima kesepakatan dan tekan tombol “Next”.



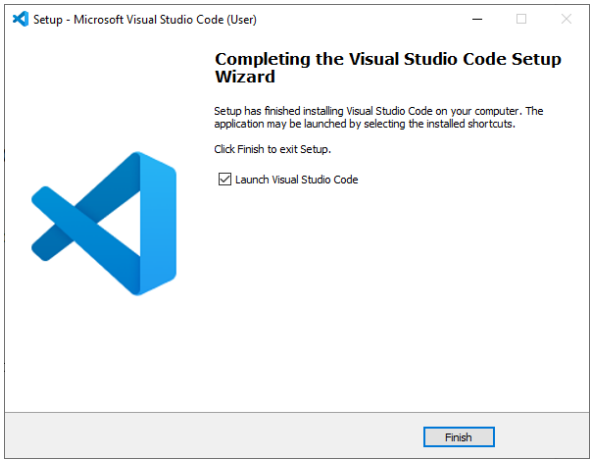
2. Pilih opsi berikut dan klik “Next”.



3. Tekan tombol “Install”.



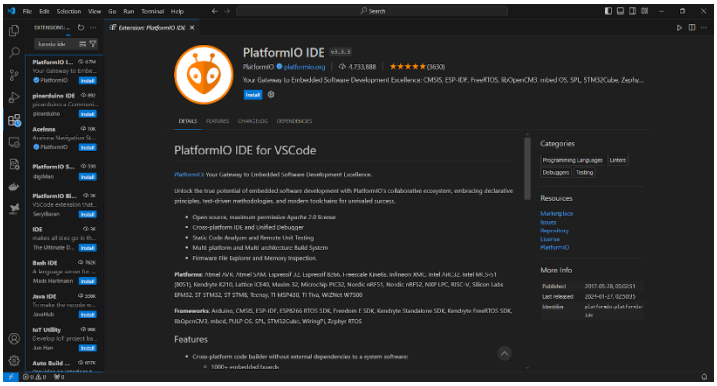
2. klik “Finish” untuk menyelesaikan instalasi.



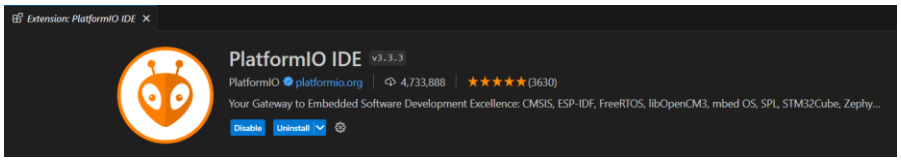
Lihat dokumentasi VSCode https://code.visualstudio.com/docs untuk tata cara penggunaan yang lengkap.

1.2. Instalasi PlatformIO IDE

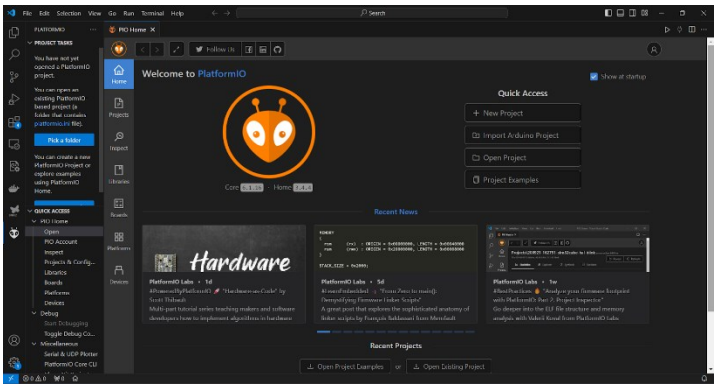
1. Buka VSCode *Extension Manager*.
2. Cari ekstensi resmi platformio ide.
3. Install PlatformIO IDE.



4. Setelah menginstal, pastikan bahwa ekstensi PlatformIO IDE diaktifkan seperti yang ditunjukkan di bawah ini.



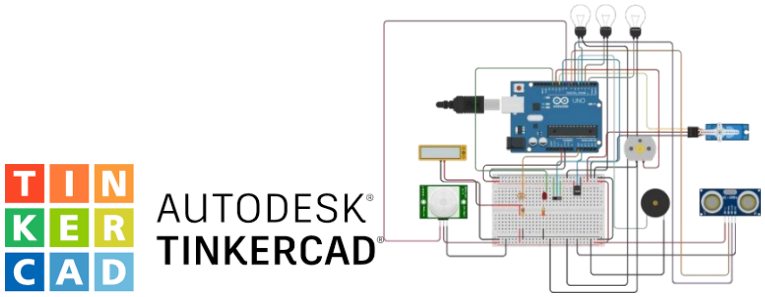
5. ikon PlatformIO harus muncul di *sidebar* kiri serta ikon "Home" yang mengarahkan ke beranda PlatformIO.



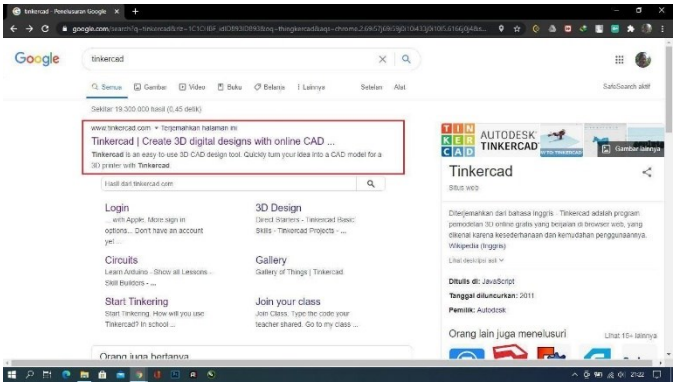
Jika tidak terlihat ikon PIO dan quick tool di bagian bawah, mungkin diperlukan memulai ulang VSCode agar perubahan tersebut berlaku. Lihat dokumentasi PlatformIO IDE https://platformio.org/install/ide?install=vscode untuk tata cara penginstalan yang lengkap.

1.3. Tinker CAD

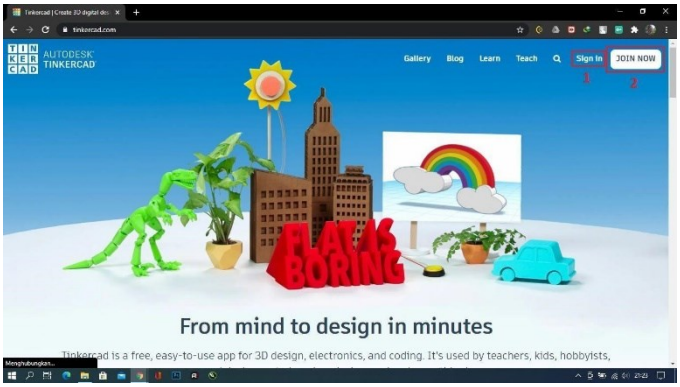
Tinkercad merupakan sebuah platform web yang menyediakan sarana untuk belajar desain 3D, rangkaian elektronika dan codeblock secara online. Pada materi ini akan berfokus membahas salah satu fitur dari Tinkercad yaitu Tinkercad Circuits. Tinkercad Circuits merupakan fitur simulator dimana kita dapat belajar mengenai rangkaian elektronika. Tinkercad Circuits juga support dengan Arduino board serta program Arduino IDE. Dengan adanya fitur ini kita bisa membuat project- project Arduino tanpa harus membeli board Arduino dan perangkat lain yang dibutuhkan.



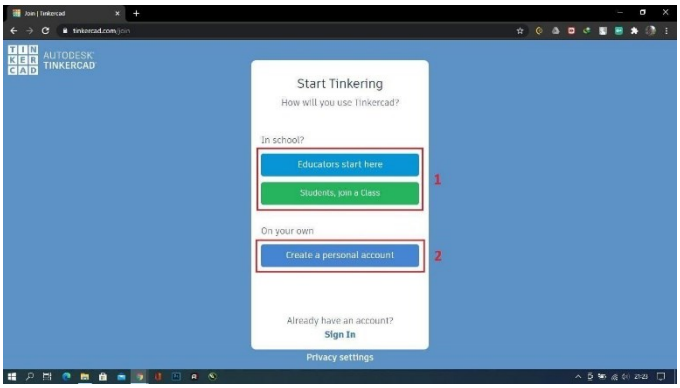
1. Buka laman https://www.tinkercad.com/ atau ketikkan “Tinkercad” pada browser.



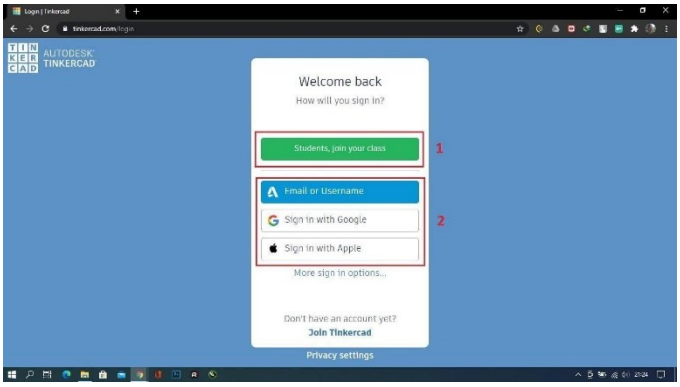
2. Klik (1) “Sign in” untuk masuk jika sudah memiliki akun Tinkercad. Jika belum, klik (2) “JOIN NOW”.



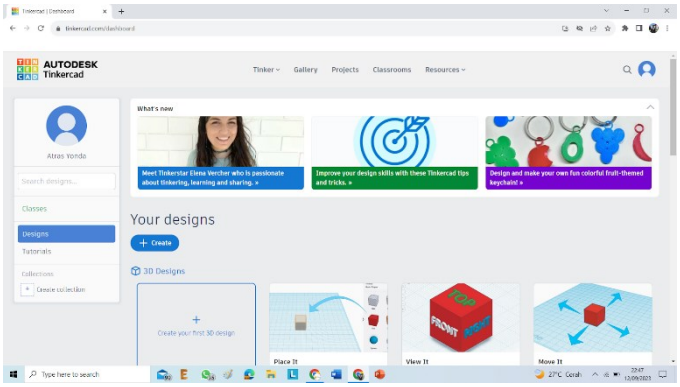
3. Pada menu JOIN NOW pilih jenis akun yang akan dibuat. Educators ‘pengajar’ atau Students ‘Murid’ pada kotak (1). Jika ingin menggunakan secara personal, klik (2) “Create a personal account”.



4. Pada menu “Sign in” dapat login dengan akun “Students”(1) untuk murid atau login dengan akun Email, Google, Apple (2) dan akun lainnya.



5. Berikut ini merupakan tampilan layar setelah berhasil masuk ke dashboard Tinkercad.

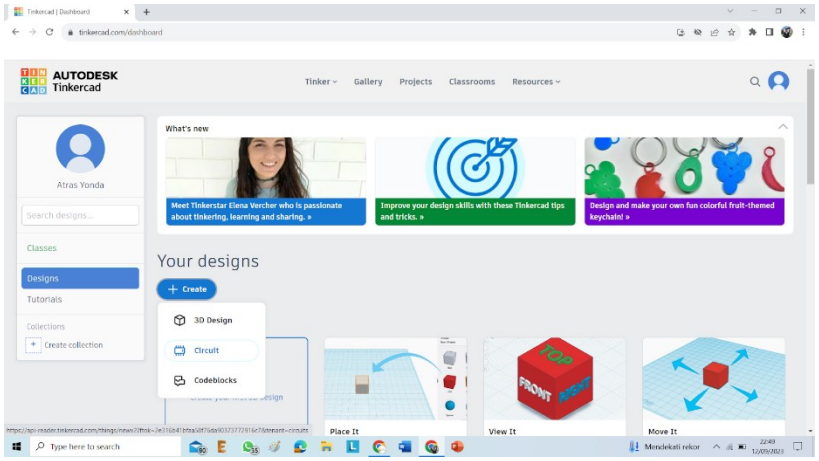

AUTODESK
Tinkercad
Tinker
Gallery
Projects
Classrooms
Resources
What's new
Your designs
Create
3D Designs
Create your first 3D design
Place It
View It
Move It
FRONT
RIGHT
TOP

BAB II
DEVELOPMENT ENVIRONMENT

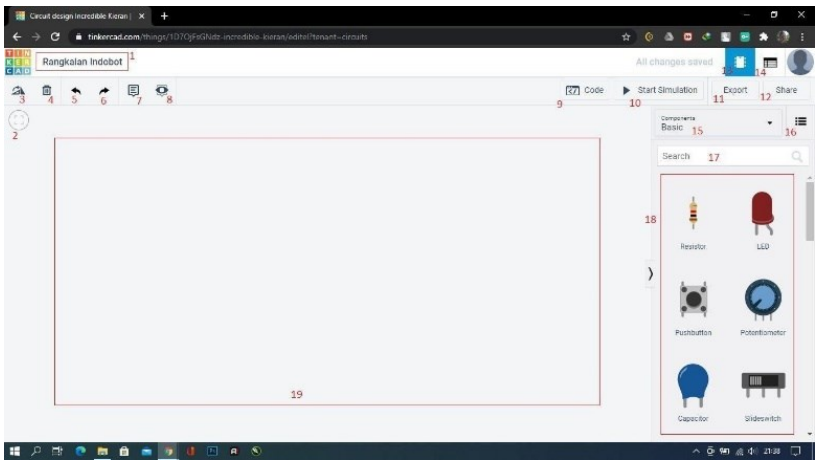
2.1. Getting Started with Autodesk TinkerCAD

Tampilan Awal

Membuat desain klik “Create” kemudian pilih menu “Circuit”.



Selanjutnya, akan dimulai membuat desain. Berikut ini merupakan keterangan bagian-bagian *worksheet* Tinkercad *Circuits*.



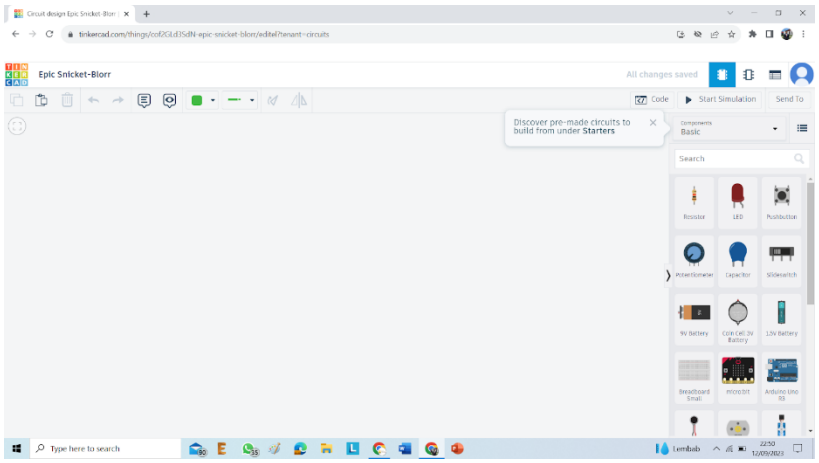
1. **Title** : Judul project
2. **Zoom to fit** : Membesarkan ukuran workspace sesuai desain
3. **Rotate** : Memutar komponen
4. **Delete** : Menghapus komponen
5. **Undo** : Membatalkan suatu perintah yang sudah dilakukan sebelumnya
6. **Redo** : Mengulang suatu perintah yang telah dibatalkan sebelumnya
7. **Annotation** : Memberi komentar

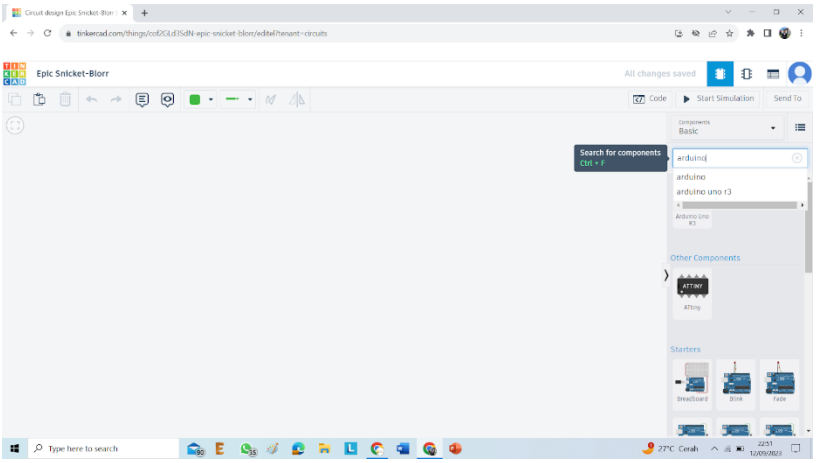
8. **View/Hide** : Memunculkan/menyembunyikan komentar
9. **Toggle code editor** : Memprogram mikrokontroller
10. **Start/Stop Simulation** : Memulai/menghentikan simulasi
11. **Export** : Mengubah skema ke dalam bentuk ".BRD" software EAGLE
12. **Share** : Mendownload skema / membagikan skema
13. **Circuit View** : Tampilan sirkuit
14. **Component List** : Tampilan daftar komponen
15. **Group Components** : Tampilan grup komponen
16. **Type Components View** : Mengubah tampilan komponen
17. **Search** : Mencari komponen
18. **Components** : Menambahkan komponen ke workspace
19. **Workspace** : Tempat membuat skema/rangkaian

Memulai Proyek Baru

1. Add *Component*

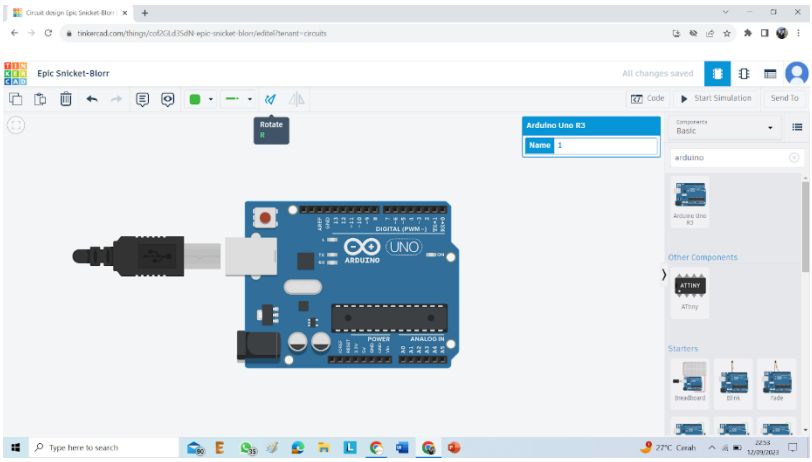
   Berikut ini adalah tampilan komponen yang ada pada Tinkercad. Untuk menambahlan komponen pada *workspace*, ketik komponen yang ingin dicari pada kolom "Search". Sebagai contoh "Arduino".





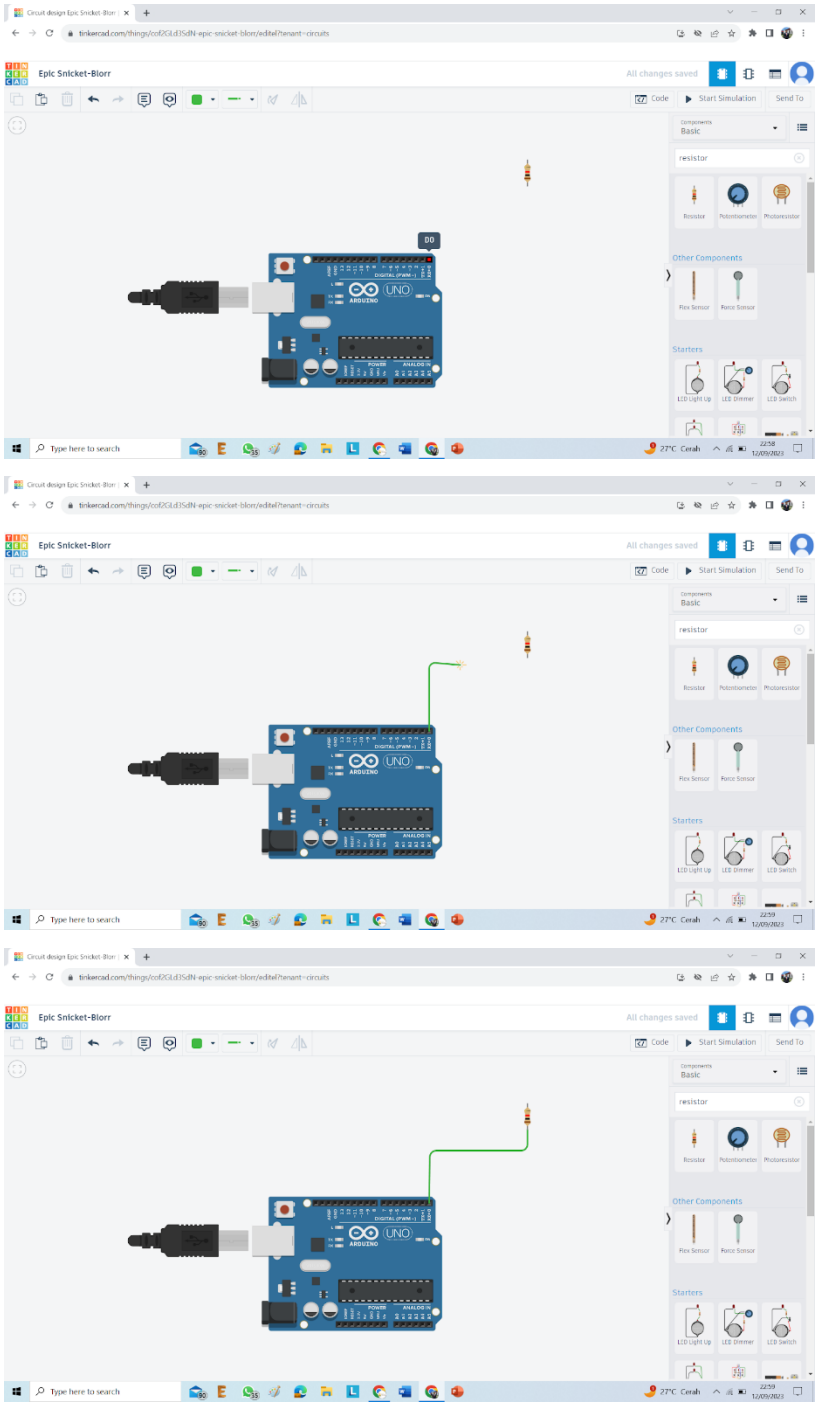
2. Edit *Component*

   Tekan komponen yang diinginkan kemudian tahan dan arahkan ke *workspace* yang diinginkan (*drag*). Komponen bisa diputar, di-*copy*, atau dipindahkan sesuai kebutuhan.



3. *Wiring*

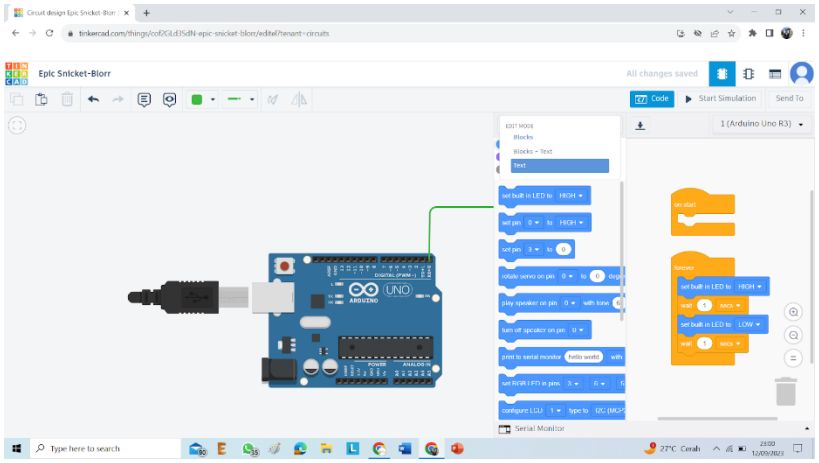
   Wiring adalah proses menyambungkan komponen dengan kabel. Jika kita ingin menyambungkan pin D0 dengan resistor, maka tekan pin D0 kemudian akan muncul kabel berwarna hijau lalu tarik kabel dan arahkan ke resistor. Tap untuk membuat kabel berhenti bergerak lurus.



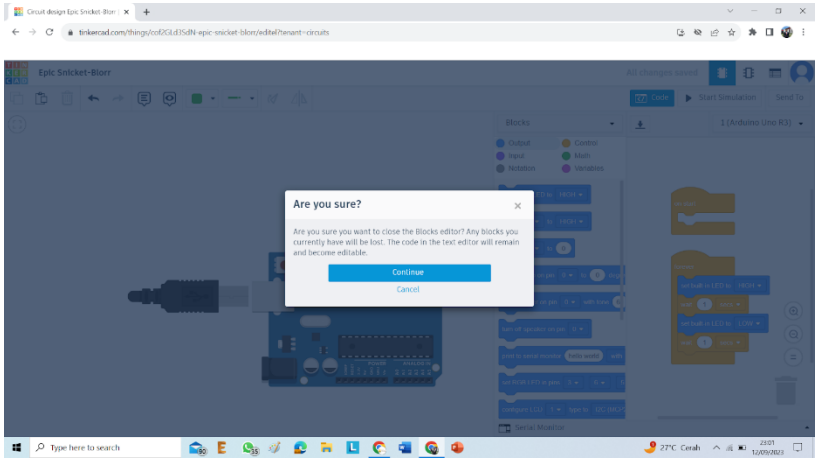
Warna kabel dan bentuk kabel bisa diubah sesuai kebutuhan pada menu bar.

4. Tulis Code

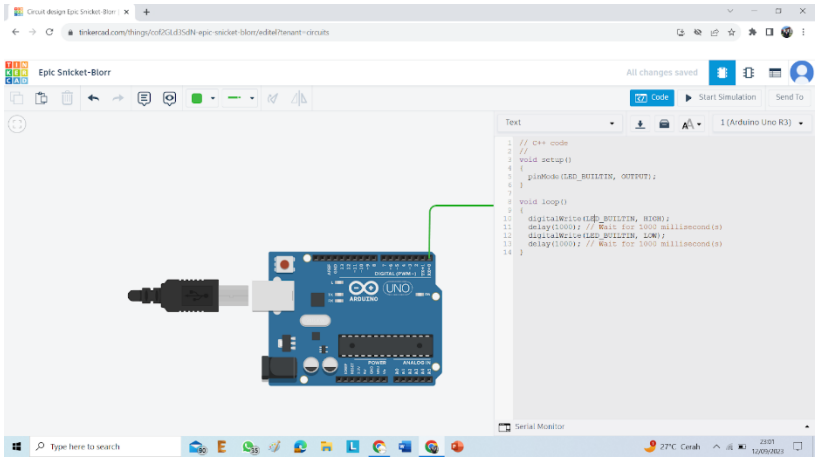
Terdapat menu “Code” yang berfungsi untuk memasukkan program yang ingin kita jalankan. Pada sebuah komponen Arduino Uno R3 ini akan dimasukkan sebuah kode berbentuk teks. Tekan menu "Code" kemudian pilih “Text”.



Tekan "*Continue*" jika muncul perintah seperti dibawah.

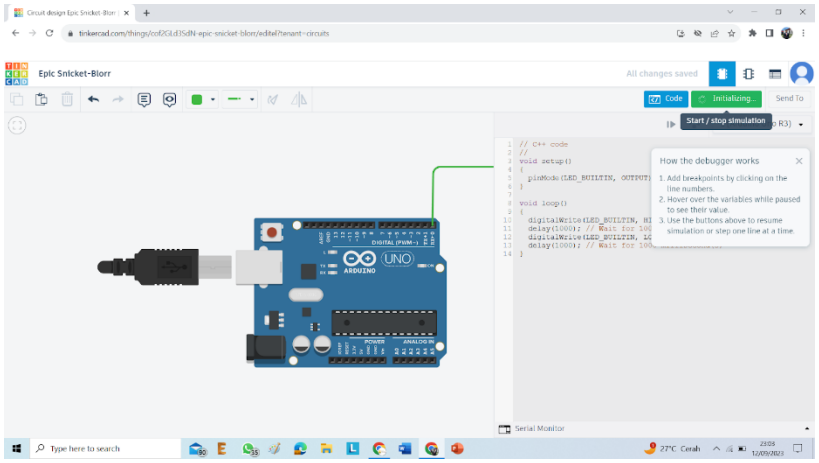


Pada blok kode, kita bisa memasukkan program yang kita inginkan sesuai kebutuhan. Pada contoh dibawah, LED akan berkedip setiap 1000 milisecond atau 1 detik dan terus berulang sampai program di *end*.

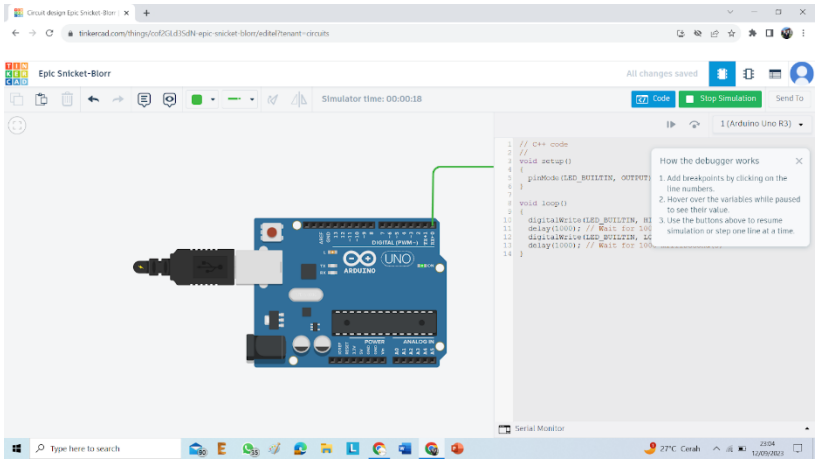


5. Start Simulasi

   Tekan "Start Simulation" untuk melakukan *run* pada kode yang telah dibuat.



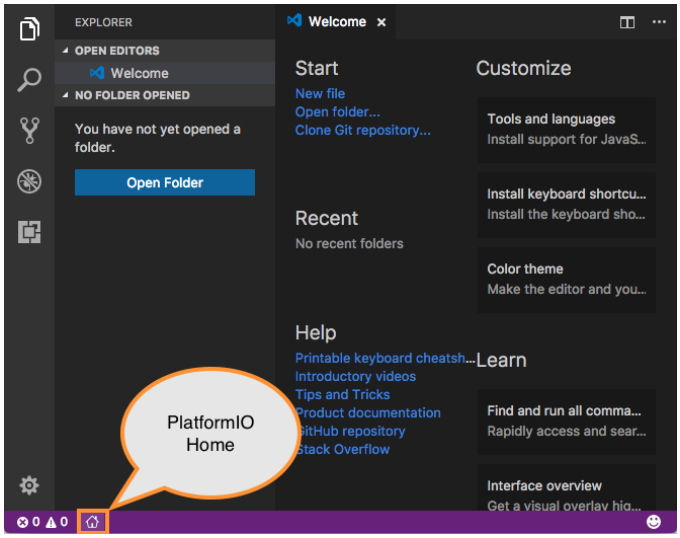
Selesai, kode telah berhasil berjalan ditunjukkan dengan lampu LED dalam simbol "L" yang telah menyala setiap 1000 *milisecond.*



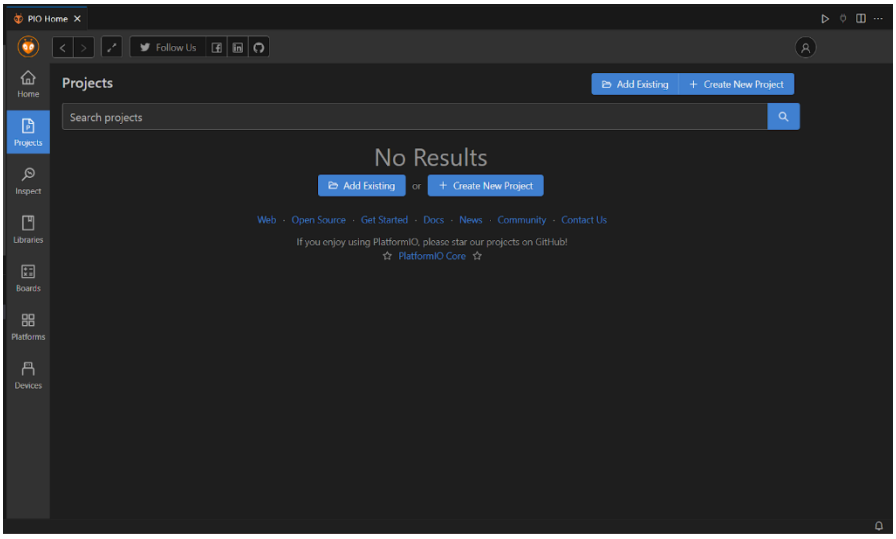
2.2. Getting Started with PlatformIO IDE

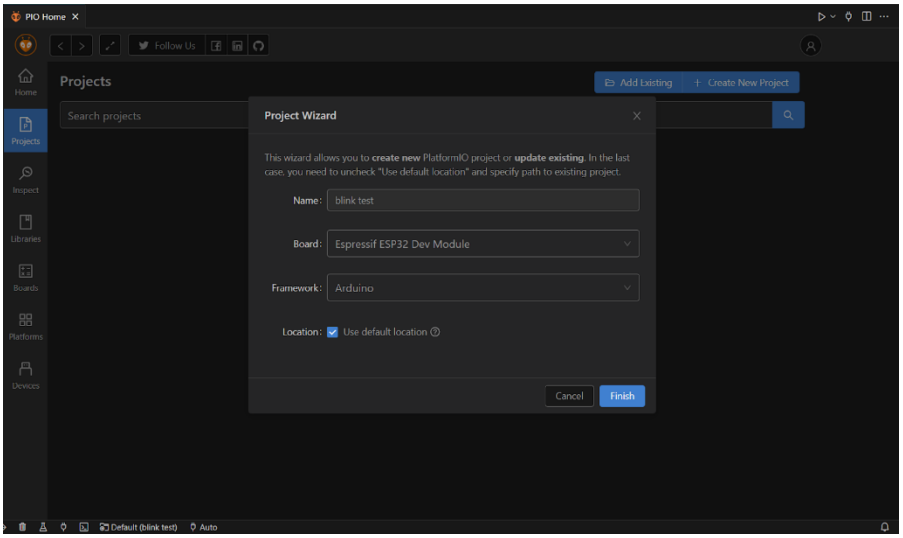
Memulai Proyek Baru

1. Klik tombol "PlatformIO Home" di bagian bawah *Toolbar* PlatformIO.

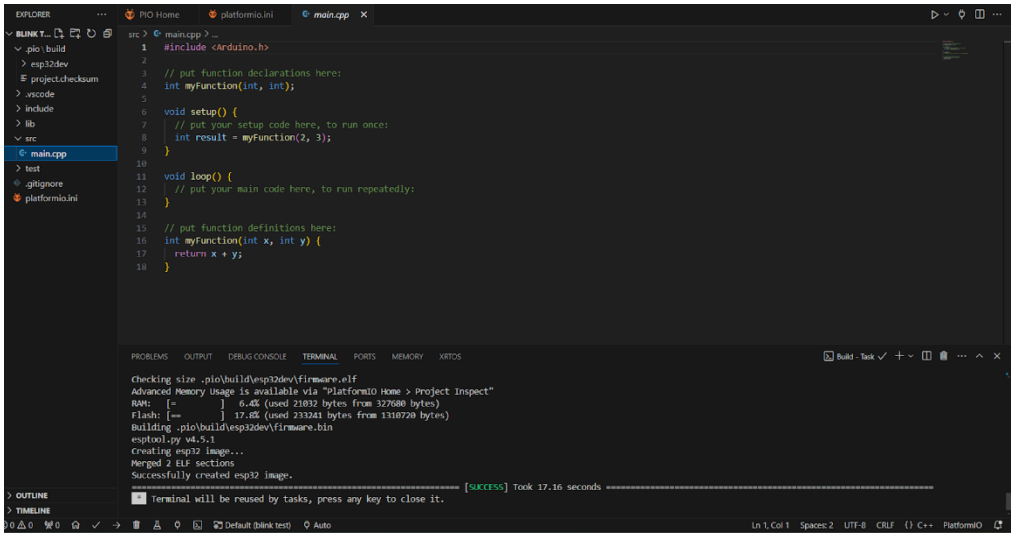


2. Klik pada "Create New Project", pilih jenis mikrokontroler dan buat proyek PlatformIO baru.





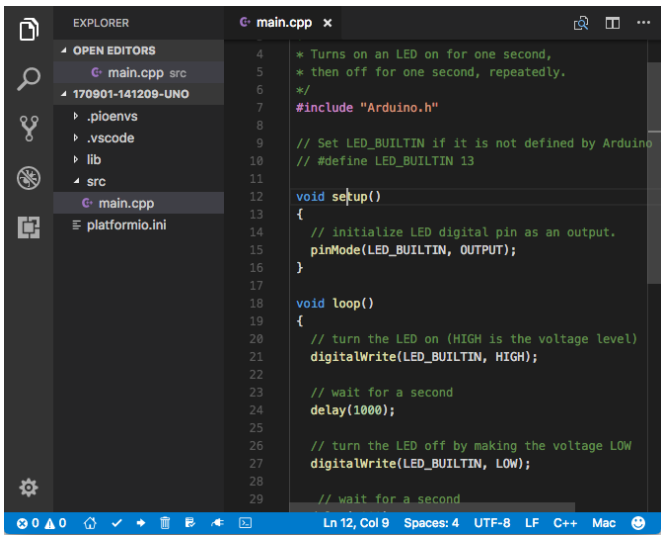
3. Buka file main.cpp dari folder src dan ganti isinya dengan code yang diinginkan.



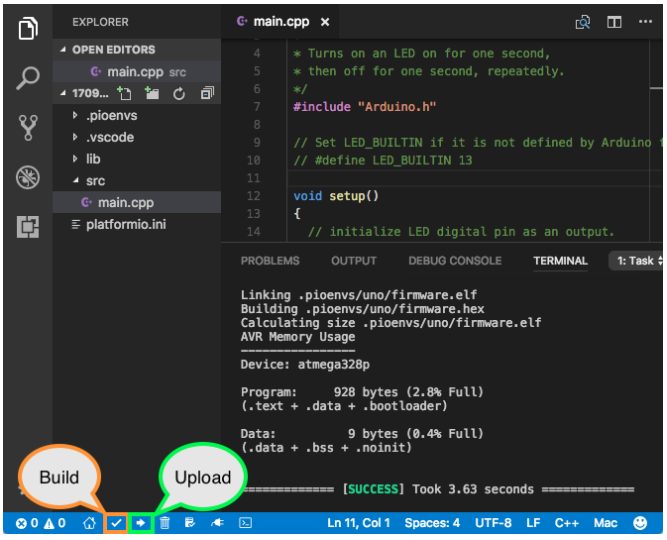
Tempelkan kode di bawah ini.

```
#include <Arduino.h>

#define LED 2

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

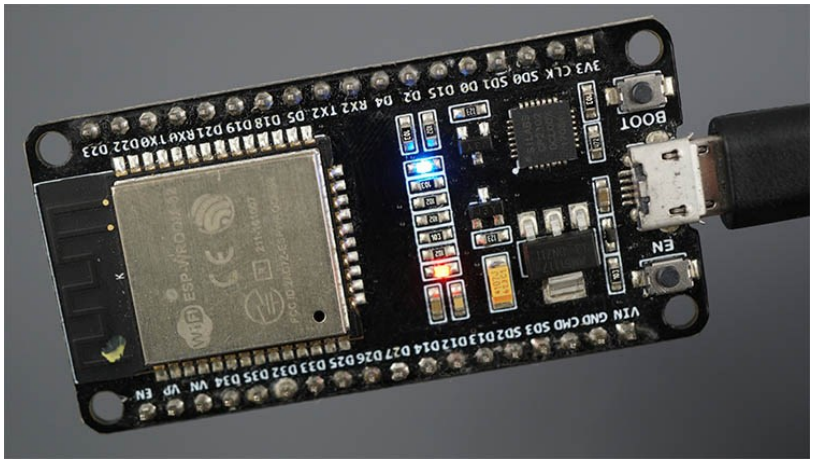
```
void loop() {
 // put your main code here, to run repeatedly:
 digitalWrite(LED, HIGH);
 Serial.println("LED is on");
 delay(1000);
 digitalWrite(LED, LOW);
 Serial.println("LED is off");
 delay(1000);
}
```



4. Bangun proyek dengan menggunakan tombol "Build" di *Toolbar* PlatformIO



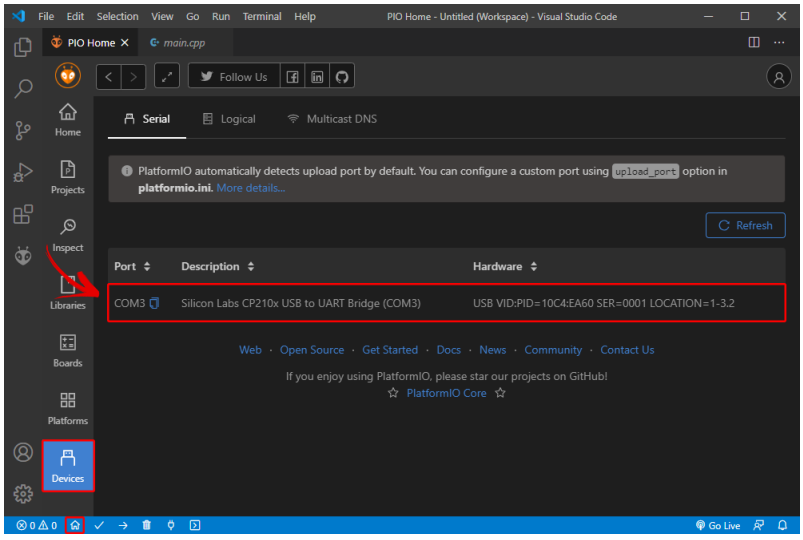
5. Setelah mengunggah kode, ESP32 seharusnya berkedip LED bawaan setiap detik.



Untuk memperdalam cara penggunaan PlatformIO dapat dilihat pada https://docs.platformio.org/en/latest/integration/ide/vscode.html#installation .

Mendeteksi COM Port

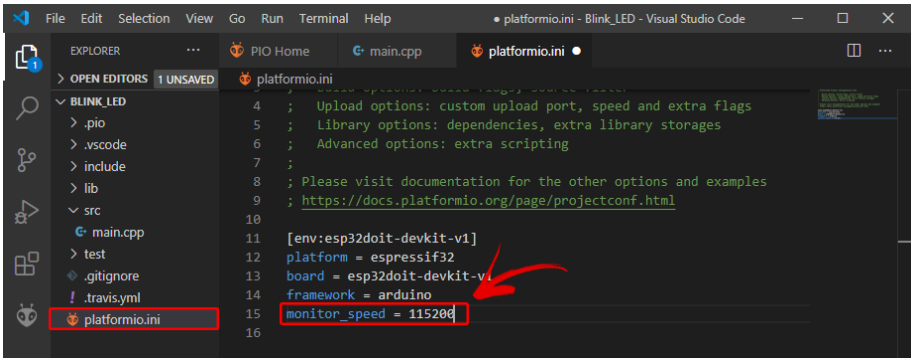
PlatformIO akan secara otomatis mendeteksi port mikrokontroler yang terhubung. Untuk memeriksa perangkat yang terhubung, dapat pergi ke "PIO Home" dan mengklik ikon Perangkat.



Mengubah Baud Rate

Baud rate secara normal yang digunakan oleh PlatformIO adalah 9600. Dimana nilai ini dapat diubah pada file platformio.ini adalah file konfigurasi PlatformIO untuk setiap proyek. Untuk mengubah baud rate perlu menambahkan code.

monitor_speed = baud_rate



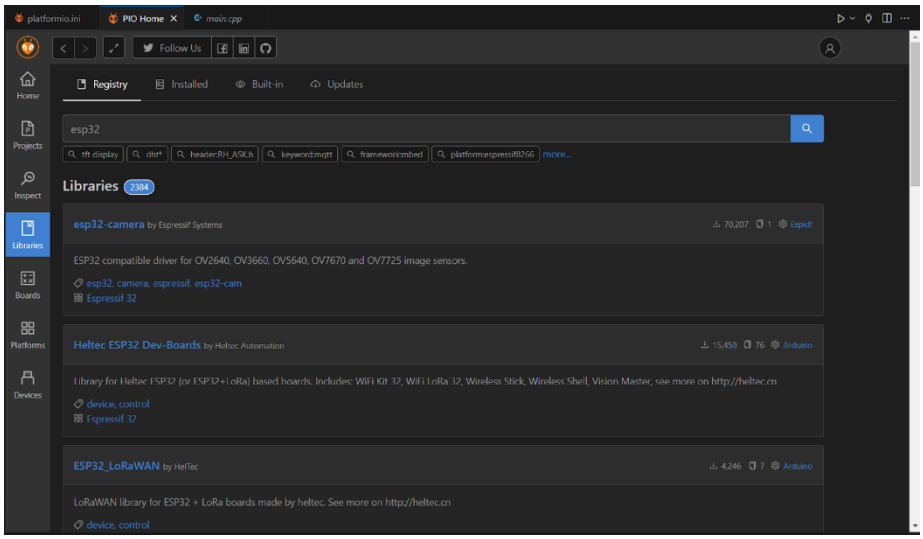
Selain baud rate terdapat opsi lain yang dapat di ubah pada file platformio.ini yaitu.

a. Platform : SoC yang digunakan oleh mikrokontroler.
b. Board : mikrokontroler yang digunakan.
c. Framework : lingkungan software yang akan menjalankan kode proyek.

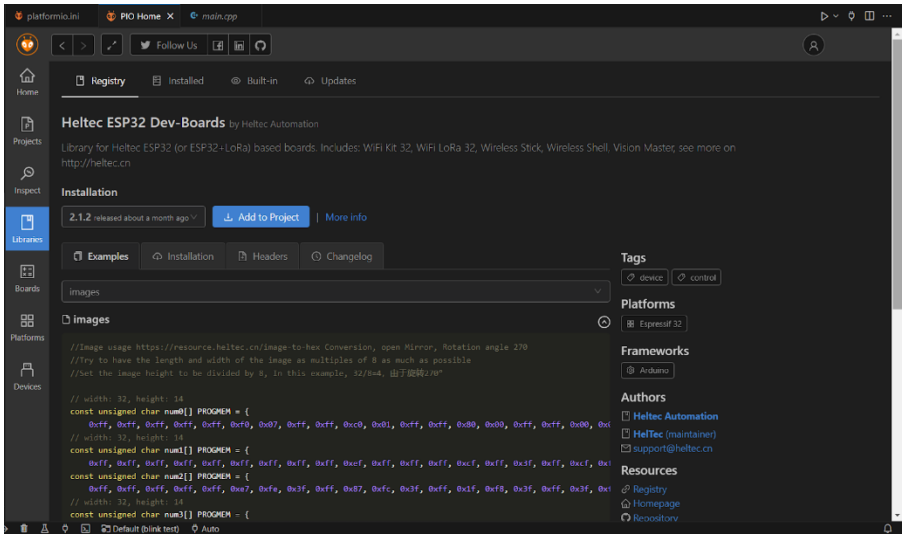
Menambahkan library pada PlatformIO

1. Klik ikon “Home” untuk pergi ke PlatformIO Home. Klik ikon “Libraries” di bilah samping kiri.

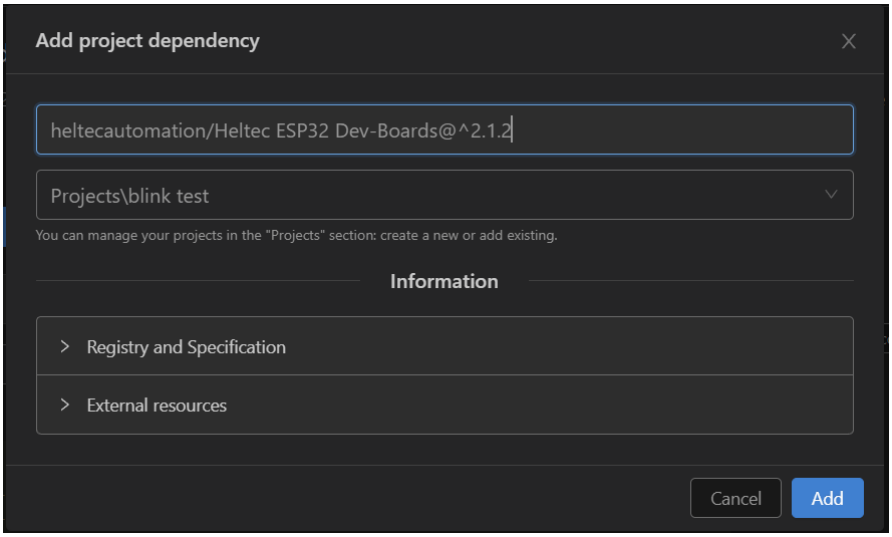
2. Cari *library* yang ingin di instal



3. Klik pada *library* yang ingin disertakan dalam proyek. Kemudian, klik “Add to Project”.



4. Pilih proyek di mana ingin digunakan *library* tersebut.



5. *Library* akan ditambahkan dengan pengidentifikasi pustaka menggunakan direktif lib_deps pada file platformio.ini.

Troubleshooting

1. Jika saat mencoba mengunggah sketch baru ke ESP32 dan mendapatkan pesan kesalahan seperti "*A fatal error occurred: Failed to connect to ESP32: Timed out... Connecting...*" ini berarti ESP32 tidak dalam mode *flashing/uploading*. Setelah memilih jenis mikrokontroler dan COM port yang benar, ikuti langkah-langkah berikut:
   a. Tahan tombol BOOT di papan ESP32
   b. Tekan tombol Upload pada PlatformIO
   c. Setelah melihat pesan "Connecting…." lepaskan jari dari tombol BOOT
   d. Akan muncul pesan "Done uploading" jika berhasil

   Perlu diingat urutan tombol diatas perlu dilakukan setiap kali ingin mengunggah sketch baru. Tetapi, untuk memperbaiki permasalahan ini secara permanen dapat mengikuti tutorial berikut https://randomnerdtutorials.com/solved-failed-to-connect-to-esp32-timed-out-waiting-for-packet-header/ .

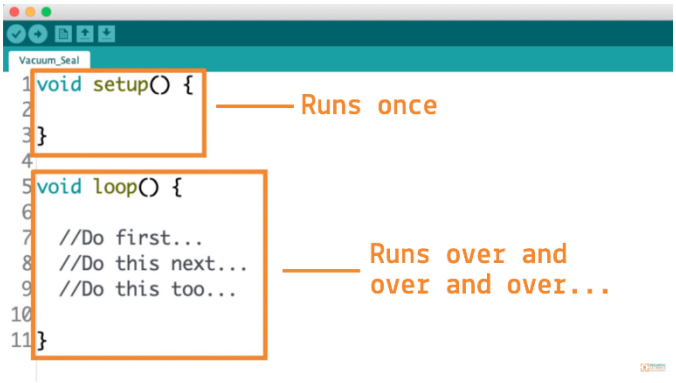
2. Jika mendapatkan pesan kesalahan seperti "COM Port not found/not available", mungkin perlu di instal driver CP210x terlebih dahulu. (windows). https://randomnerdtutorials.com/install-esp32-esp8266-usb-drivers-cp210x-windows/ .

BAB III
PEMROGRAMAN DASAR MIKROKONTROLER

3.1. Sruktur Program

Persyaratan minimum mutlak dari program Arduino based *firmware* adalah penggunaan dua fungsi: void setup() dan void loop() . "Void" menunjukkan bahwa tidak ada yang dikembalikan saat dieksekusi.

1. void setup(), fungsi ini hanya dijalankan sekali, ketika mikrokontroler dinyalakan. Disini didefinisikan hal-hal seperti mode pin (input atau output), baud rate komunikasi serial, atau inisialisasi sebuah library.
2. void loop(), di sinilah tempat menulis kode yang ingin di eksekusi berulang kali, seperti menyalakan/mematikan lampu berdasarkan input, atau melakukan pembacaan sensor setiap X detik.



Gambar 1. Struktur Program Arduino based Firmware

3.2. Tipe Data

Ketika membuat variabel, kompiler perlu mengetahui berapa banyak memori yang harus dialokasikan untuk penyimpanan dan bagaimana data harus ditangani dalam operasi aritmatika dan logika. Ada empat tipe data dasar. Dua dari mereka menangani data tipe integer sementara dua lainnya menangani data floating-point.

Type	Description	Size (bits)
char	Single Character	8
int	Integer	16
float	Single Precision Floating Point Number	32
double	Double Precision Floating Point Number	64

Size (bits) disini tidak distandarisasi, meskipun ukuran yang disajikan merupakan nilai umum. Tipe data int adalah yang paling bervariasi dari satu kompiler ke kompiler lainnya karena ukurannya disesuaikan dengan lebar Arithmetic Logic Unit (ALU) / *data memory word*. Sehingga, pada mikrokontroler 16-bit sebuah int akan berukuran 16-bit, sementara pada mikrokontroler 32-bit sebuah int akan berukuran 32-bit. Ini hampir selalu benar tetapi sering kali tidak berlaku di dunia 8-bit. Banyak kompiler 8-bit mendefinisikan int sebagai 16-bit, sementara beberapa mendefinisikannya sebagai 8-bit. Jadi, sebelum mulai menulis kode, pastikan membaca manual pengguna compiler untuk mengetahui ukuran sebuah int.

Bahkan char, yang di masa lalu hampir selalu diimplementasikan dengan 8-bit untuk mengakomodasi pengkodean karakter ASCII 7-bit, sekarang bisa menjadi 16-bit pada beberapa kompiler yang mendukung pengkodean Unicode.

3.3. Control Flow

Instruksi atau perintah dalam program yang dijalankan, biasanya di struktur secara berurutan tetapi dengan kemampuan untuk mengambil keputusan atau berpindah ke bagian lain dari program berdasarkan kondisi tertentu. Pada dasarnya, *control flow* menentukan urutan eksekusi dari kode program.

1. Conditional Statements

   Struktur yang memungkinkan mikrokontroler untuk mengabil keputusan berdasarkan kondisi tertentu, seperti if, else if, dan else.

Tabel 1. Conditional Statements pada Control Flow

```
if (temperature < 30) {
    // Code to execute if temeprature is less than 30
} else if (temperature < 20) {
    // Code to execute if temperature is less than 20
} else {
    // Code to execute if none of the above conditions are met
}
```

2. Looping

   Looping berguna untuk mengeksekusi blok kode berulang kali. Ini mencakup for, while, dan do-while.

Tabel 2. Looping pada Control Flow

```
for (int i = 0; i < 5; i++) {
    // Code to be repeated 5 times
}

while (isRunning) {
    // Code to be repeated as long as isRunning is true
}
```

3. Functions

   Functions berguna untuk melakukan sesuatu yang spesifik. Ini sangat berguna jika ingin melakukan hal yang sama di beberapa tempat dalam kode.

Tabel 3. Functions pada Control Flow

```
int rectangleArea(int length, int width) {
    int area = length * width;
    return area;
}
```

BAB IV
FITUR KHUSUS ESP32

4.1. Power Management & Sleep Mode

Manajemen daya adalah proses pengelolaan konsumsi daya listrik dalam sistem atau perangkat untuk mengoptimalkan efisiensi dan memperpanjang umur baterai. Manajemen daya pada ESP32 sangat penting karena kebanyakan sistem berbasis ESP32 adalah aplikasi Internet of Things (IoT) yang beroperasi dengan sumber daya baterai. Dalam kasus tersebut, diperlukan suatu cara untuk meminimalkan frekuensi pengisian ulang atau penggantian baterai agar didapatkan *availability* sistem yang tinggi. *Availability* yang tinggi berarti sistem beroperasi sesuai dengan fungsinya dalam waktu operasi yang diharapkan tanpa mengalami kegagalan. Beberapa metode manajemen daya pada ESP32 meliputi:

1. Pengaturan frekuensi sistem

ESP32 memiliki fitur Dynamic Frequency Scaling (DFS) yang berfungsi untuk mengubah frekuensi operasi CPU dan *bus peripheral* secara *realtime* sesuai dengan kebutuhan sistem. Frekuensi yang tinggi berarti sistem mampu mengeksekusi lebih banyak instruksi atau proses setiap detiknya. CPU pada ESP32 beroperasi pada 240 MHz secara *default*, maka CPU dapat melakukan 240 juta instruksi per detik. Namun, semakin tinggi frekuensi berarti semakin banyak daya listrik yang dibutuhkan karena lebih banyak terjadi *switching* dalam sirkuit ESP32. Dengan mengimplementasikan fitur DFS pada program, ESP32 dapat beroperasi pada 160 MHz dan 80 MHz untuk menghemat konsumsi daya listrik saat program tidak memerlukan kecepatan operasi maksimum.

Dynamic Frequency Scaling dapat digunakan menggunakan API Power Management yang tersedia pada dokumentasi ESP32. Berikut contoh pada **Tabel 4** agar ESP32 berjalan dengan clock maksimum 80 MHz, minimum 10 MHz, dan otomatis memasuki mode Light Sleep.

Tabel 4. Source Code fitur DFS dan Light Sleep otomatis pada ESP32

```
#include "esp_pm.h"
void setup() {
  Serial.begin(115200);
  esp_pm_config_t pm_config = {
    .max_freq_mhz = 80,
    .min_freq_mhz = 10,
    .light_sleep_enable = true
  };
  esp_err_t ret = esp_pm_configure(&pm_config);
  if (ret == ESP_OK) {
    Serial.println("Success");
  } else {
    Serial.print("Failed code: ");
    Serial.println(ret);
  }
}
void loop() {
  Serial.println("Running in low-power mode with dynamic
frequency scaling.");
  delay(1000);
}
```

2. Sleep *Mode*

Sleep Mode adalah fitur hemat daya pada ESP32 yang dirancang untuk mematikan komponen tertentu pada perangkat. Beberapa mode yang dimiliki ESP32 adalah sebagai berikut:

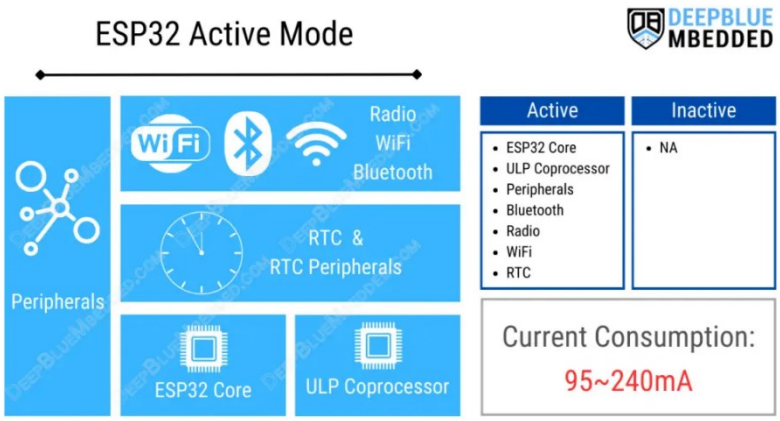
a. Active Mode

Pada mode ini, seluruh perangkat pada *chip* ESP32 menyala sehingga konsumsi arus listrik ESP32 sangat tinggi. Konsumsi daya listrik dapat dihitung dengan mengalikan nominal arus terhadap tegangan operasi ESP32 (3.3V). Berdasarkan datasheet ESP32, konsumsi arus ESP32 dengan penggunaan fungsi Transmit dan Receive modul WiFi

(802.11x) atau Bluetooth/BLE dapat dilihat pada gambar berikut

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	—	240	—	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	—	190	—	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	—	180	—	mA
Receive 802.11b/g/n	—	95 ~ 100	—	mA
Transmit BT/BLE, POUT = 0 dBm	—	130	—	mA
Receive BT/BLE	—	95 ~ 100	—	mA

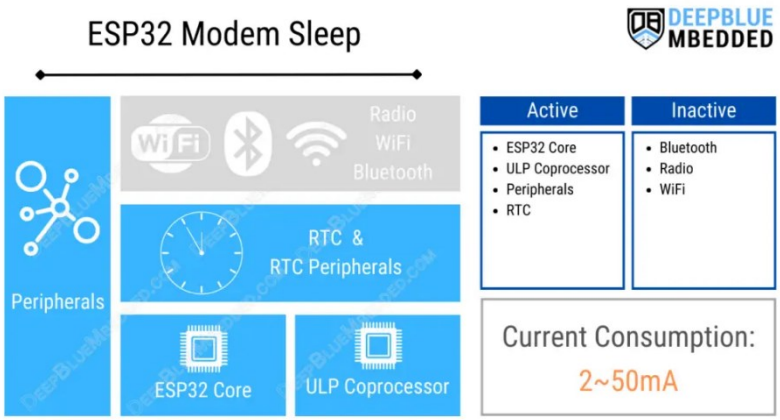
Gambar 2. Konsumsi Arus ESP32 saat Active Mode



Gambar 3. Kondisi ESP32 dalam mode Active

b. Modem Sleep Mode

Pada mode ini modul Bluetooth, Radio, dan WiFi pada ESP32 dimatikan. Namun, tidak menutup kemungkinan digunakannya modul komunikasi tersebut. Dapat dibuat program otomatis untuk mengganti mode ESP32 menjadi Active Mode dengan interval waktu tertentu sehingga ESP32 dapat melakukan proses Transmit atau Receive data dan kemudian kembali ke Modem Sleep Mode. Konsumsi arus tipikal pada mode ini adalah 2 mA sampai 50 mA pada tegangan 3.3 V.



Gambar 4. Kondisi ESP32 dalam mode Modem Sleep

Berikut contoh penggunaan Modem Sleep untuk mendapatkan data pada **https://icanhazdadjoke.com/** menggunakan API. Data diambil (fetch) untuk setiap 1 menit (60000 ms), maka selama periode tersebut akan digunakan `WiFi.setSleep(true)`, dan digunakan `WiFi.setSleep(false)` untuk membangunkan modem setelah 1 menit terlewati. Kode lengkap dapat dilihat pada **Tabel 5.**

Tabel 5. Source Code fitur Modem Sleep untuk mematikan WiFi sementara

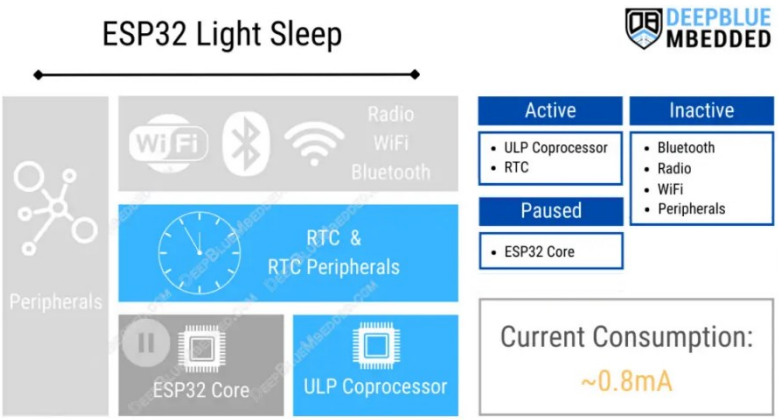
```
#include <WiFi.h>
#include <HTTPClient.h>
#define mS_MODEM_SLEEP_DURATION 60000
#define STA_SSID "SSID"
#define STA_PASS "PASSWORD"
const char* serverUrl = "https://icanhazdadjoke.com/";

void setup() {
    Serial.begin(115200);
    Serial.println("Connecting to WiFi...");
    WiFi.mode(WIFI_STA);
    WiFi.begin(STA_SSID, STA_PASS);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
```

```
    Serial.println("\nWiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
}

void loop() {
    if (WiFi.status() == WL_CONNECTED) {
        fetchDadJoke();
    } else {
        reconnectWiFi();
    }
    WiFi.setSleep(true); // Enable modem sleep
    Serial.println("Modem sleep enabled for 1 minute...");
    delay(mS_MODEM_SLEEP_DURATION);
    WiFi.setSleep(false);  // Disable modem sleep
    Serial.println("Waking up, modem sleep disabled.");
}

void fetchDadJoke() {
    HTTPClient http;
    http.begin(serverUrl);
    http.addHeader("Accept", "application/json");

    int httpResponseCode = http.GET();
    if (httpResponseCode == 200) {
        String response = http.getString();
        int jokeStart = response.indexOf("\"joke\":\"") + 8;
        int jokeEnd = response.indexOf("\",\"status\"");
        String joke = response.substring(jokeStart, jokeEnd);
        joke.replace("\\\"", "\"");
        Serial.println("Here's a dad joke for you: " + joke);
```

```
    } else {
        Serial.print("Error receiving joke, code: ");
        Serial.println(httpResponseCode);
    }
    http.end();
}

void reconnectWiFi() {
    Serial.println("Reconnecting to WiFi...");
    WiFi.begin(STA_SSID, STA_PASS);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("\nReconnected to WiFi");
}
```

c. Light Sleep Mode

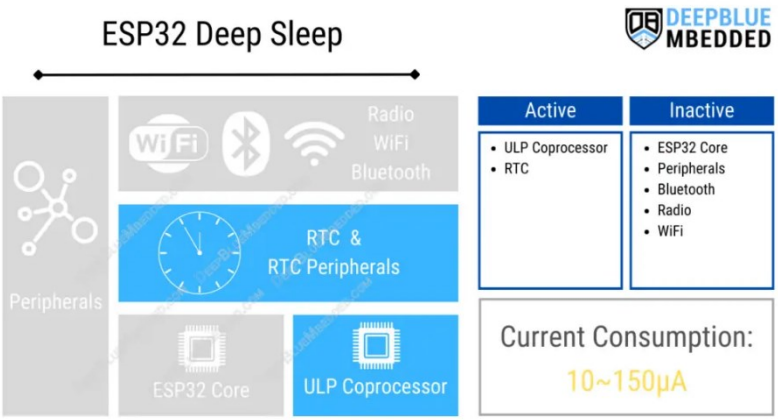
Hampir sama dengan Modem Sleep Mode, tetapi pada mode ini bagian CPU (ESP32 Core), RAM dan peripherals dihentikan sementara (*pause*). *Pause* dilakukan dengan cara *clock-gating*, yaitu metode untuk mengurangi konsumsi daya dengan menghentikan sinyal *clock* di dalamnya. Meskipun dihentikan sementara, CPU, RAM, dan peripherals tetap mempertahankan data yang telah tersimpan karena kondisinya masih menyala hanya saja tidak melakukan instruksi apapun karena tidak ada sinyal *clock*. ESP32 dapat diaktifkan kembali dengan menggunakan sinyal *trigger* dari GPIO dan UART. Konsumsi daya tipikal pada mode ini adalah 0.8 mA pada tegangan 3.3 V. Penggunaan mode Light Sleep secara otomatis terdapat pada **Tabel 4.**



Gambar 5. Kondisi ESP32 dalam mode Light Sleep

d. Deep Sleep Mode

Saat dalam Deep Sleep Mode, seluruh komponen ESP32 dimatikan kecuali komponen Ultra Low Power (ULP) Coprocessor, Real-Time Clock (RTC) Controller, RTC Peripherals, dan RTC fast/slow memory. Pada mode ini, ESP32 tidak menyimpan data yang ada sebelum *sleep* sehingga seluruh data yang ingin dipertahankan harus disimpan ke dalam RTC memory yang mana ukurannya sangat kecil yaitu 8 kiloByte. Selain itu, ESP32 tidak akan pernah menyentuh bagian void loop() dari kode sehingga seluruh instruksi sebelum memasuki Deep Sleep Mode harus dituliskan sebelumnya pada void setup(). ULP Coprocessor dapat digunakan untuk membaca sensor ADC dan I2C dan menyimpan pembacaan pada RTC slow memory. Namun ULP Coprocessor memerlukan pemrograman berbasis *assembly* dan tidak akan dibahas pada modul ini. RTC bertugas untuk membangunkan ESP32 dengan beberapa metode, yaitu Timer, Touchpad, dan External Wakeup. Timer dapat digunakan untuk menyalakan ESP32 karena RTC memiliki fitur penghitung waktu (*timer*) di dalamnya sehingga dapat diatur seberapa lama ESP32 berada dalam Deep Sleep Mode. Pin Touch dan RTC_GPIO dapat digunakan untuk membangunkan ESP32 dari Deep Sleep Mode melalui sinyal eksternal karena termasuk ke dalam RTC Peripherals. Di dalam RTC Peripherals terdapat modul RTC I/O yang mampu menangani sinyal *input* dan *output* selama ESP32 berada dalam mode Deep Sleep. Konsumsi arus tipikal pada mode ini adalah 10 µA pada tegangan 3.3 V (tanpa penggunaan ULP Coprocessor).



Gambar 6. Kondisi ESP32 dalam mode Deep Sleep

Berikut adalah contoh penggunaan Deep Sleep Mode dengan metode Timer untuk membangunkannya secara otomatis. Durasi Deep Sleep diatur menggunakan `esp_sleep_enable_timer_wakeup(uS_SLEEP)` dengan durasi dalam satuan mikrosekon. Setelah itu, memasuki mode Deep Sleep dengan fungsi `esp_deep_sleep_start()`. Terdapat variabel untuk menghitung sudah berapa kali siklus Deep Sleep – bangun dijalankan, disimpan pada variabel `int bootCount` dengan penambahan atribut `RTC_DATA_ATTR` agar variable tersimpan di dalam RTC memory. Kode lebih lengkap dapat dilihat pada **Tabel 6** berikut.

Tabel 6. Source Code fitur Deep Sleep dengan Timer sebagai sinyal Wake Up

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

RTC_DATA_ATTR int bootCount = 0; // RTC memory variable
#define TIME_TO_SLEEP  10 // in seconds
#define uS_TO_S_FACTOR 1000000
#define uS_SLEEP TIME_TO_SLEEP * uS_TO_S_FACTOR

void setup() {
  Serial.begin(115200);
  delay(1000);
  bootCount++;
  Serial.printf("Boot number: %d\n", bootCount);
```

```
  printWakeupReason();

  esp_sleep_enable_timer_wakeup(uS_SLEEP);
  Serial.println("Entering deep sleep... for 10 seconds");
  esp_deep_sleep_start();
}

void loop() {
  //ESP32 will not run this code because it's in deep sleep
}

void printWakeupReason() {
  auto wakeup_reason = esp_sleep_get_wakeup_cause();

  switch(wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:
      Serial.println("Wakeup external signal RTC_IO");
      break;
    case ESP_SLEEP_WAKEUP_EXT1:
      Serial.println("Wakeup external signal RTC_CNTL");
      break;
    case ESP_SLEEP_WAKEUP_TIMER:
      Serial.println("Wakeup timer");
      break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD:
      Serial.println("Wakeup touchpad");
      break;
    case ESP_SLEEP_WAKEUP_ULP:
      Serial.println("Wakeup ULP program");
      break;
    default:
```

```
      Serial.println("Wakeup was not caused by deep sleep");
      break;
  }
}
```

4.2. WiFi & Bluetooth

WiFi dan Bluetooth adalah dua fitur yang tertanam di mikrokontroler ESP32, menjadikannya pilihan paling populer untuk aplikasi Internet of Things (IoT). Kedua hal ini menjadikan ESP32 unggul dalam hal konektivitas nirkabel dibandingkan dengan mikrokontroler lain seperti Arduino dan STM32 yang tidak memiliki fitur ini secara bawaan. Pada Arduino dan STM32, perlu ditambahkan modul eksternal untuk mendapatkan koneksi WiFi dan Bluetooth yang mana menambah biaya dan kompleksitas sistem baik dari segi ukuran mau pun pemrograman. Berikut penjelasan masing-masing fitur tersebut:

1. **WiFi**

ESP32 memiliki integrasi WiFi di dalamnya yang mendukung standar IEEE 802.11 b/g/n. Fitur ini memungkinkan ESP32 terhubung ke jaringan internet atau bisa juga memanfaatkan koneksi jaringan lokal (*access point*) intranet. Dengan jaringan internet, ESP32 dapat bertukar data dengan perangkat lain atau server. Hal ini memungkinkan aplikasi IoT seperti *monitoring real-time* jarak jauh, kontrol perangkat berbasis perintah jarak jauh, dan pengiriman data hasil pembacaan sensor ke *cloud*. Penggunaan fitur WiFi untuk mengakses API dan mengambil data (*fetching*) dapat dilihat pada **Tabel 5.**

2. **Bluetooth**

ESP32 juga dilengkapi dengan Bluetooth Classic dan Bluetooth Low Energy (BLE). Bluetooth Classic memungkinkan transfer data dengan bandwidth lebih besar dibanding BLE, contoh penggunaanya adalah *audio streaming* dan pengiriman data antara ESP32 dengan *smartphone*. Di sisi lain, BLE sangat efisien dalam konsumsi daya listrik karena jangkauan jaraknya pendek dan kapasitas datanya kecil. Membuat fitur BLE cocok untuk pengiriman data dengan *bandwidth* rendah seperti perangkat IoT *smart home devices*, remot kontrol, sistem notifikasi, dan sebagainya.