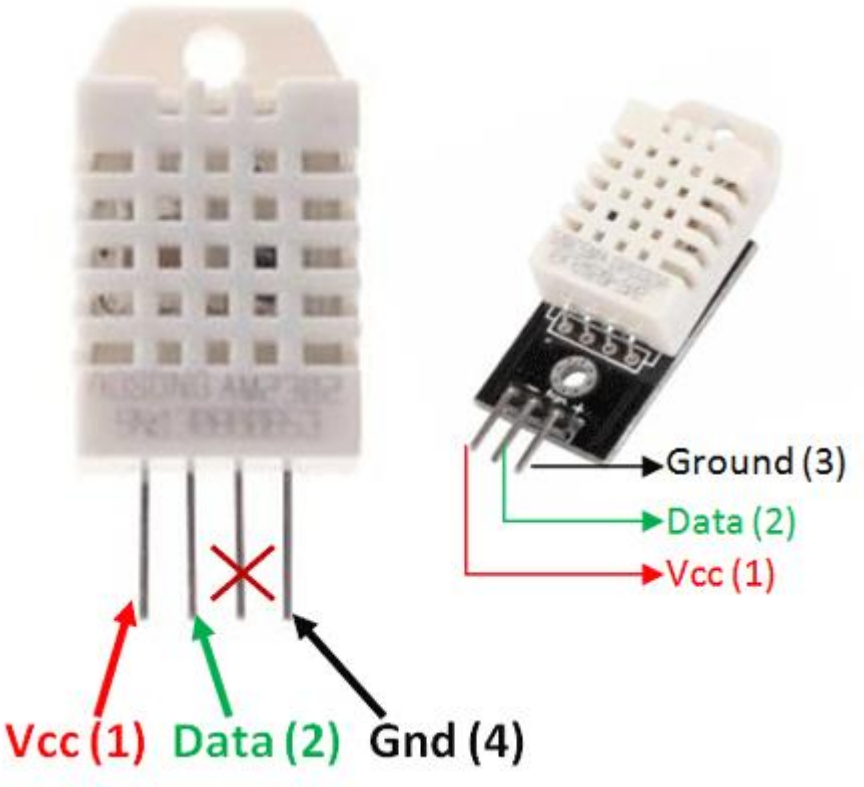
LAMPIRAN

Lampiran-1 Data Shet

DHT22 – Temperature and Humidity Sensor



Pin Identification and Configuration

No:	Pin Name	Description
For DHT22 Sensor		
1	Vcc	Power supply 3.5V to 5.5V

2	Data	Outputs both Temperature and Humidity through serial Data
3	NC	No Connection and hence not used
4	Ground	Connected to the ground of the circuit
For DHT22 Module		
1	Vcc	Power supply 3.5V to 5.5V
2	Data	Outputs both Temperature and Humidity through serial Data
3	Ground	Connected to the ground of the circuit

DHT22 Specifications

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: -40°C to 80°C
- Humidity Range: 0% to 100%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±0.5°C and ±1%

Note: The **DHT22 datasheet** can be found at the bottom of the page

DHT22 Equivalent Temperature Sensors

DHT11, AM2302, SHT71

Other Temperature Sensors

Thermocouple, TMP100, LM75, DS18820, SHT15, LM35DZ, TPA81, D6T

Difference between DHT22 Sensor and Module

The DHT22 sensor is the successor of the DHT11 module, it can either be purchased as a sensor or as a module. Either way the performance of the sensor is same. The sensor will come as a 4-pin package out of which only three pin will be used whereas the module will come with just three pins as shown in the **DHT22 pinout** above.

The only difference between the sensor and module is that the module will have a filtering capacitor and pull-up resistor inbuilt, and for the sensor you have to use them externally if required.

The module is slightly costly than the DHT11, but it has a higher measuring range and slightly better accuracy .

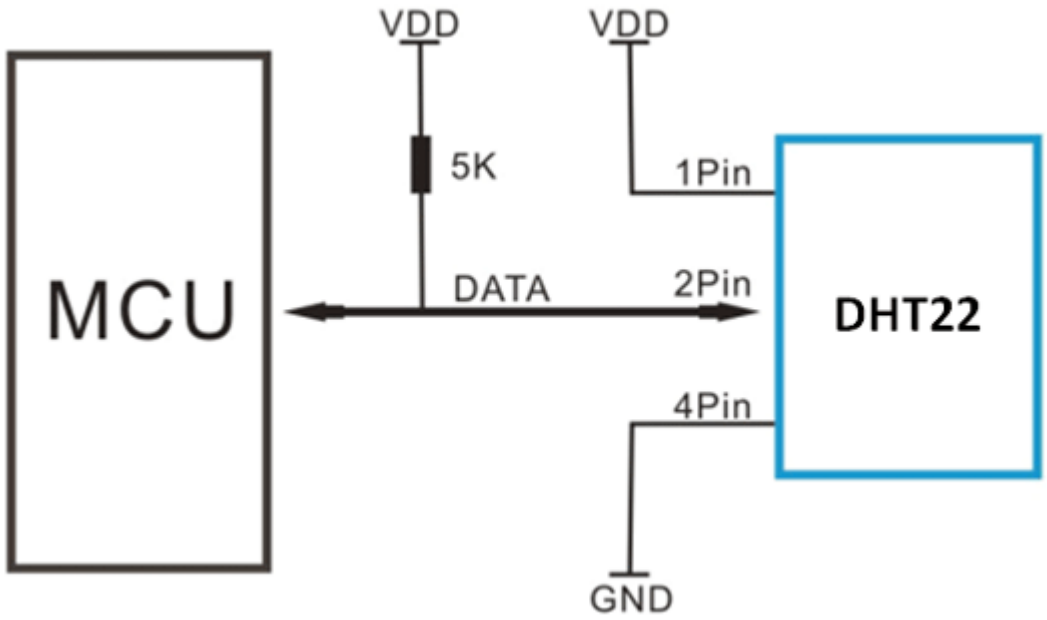
Where DHT22 Sensor is Used

The **DHT22** is a commonly used Temperature and humidity sensor. The sensor comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The sensor is also factory calibrated and hence easy to interface with other microcontrollers.

The sensor can measure temperature from **-40°C to 80°C** and humidity from **0% to 100%** with an accuracy of ±1°C and ±1%. So if you are looking to measure in this range then this sensor might be the right choice for you.

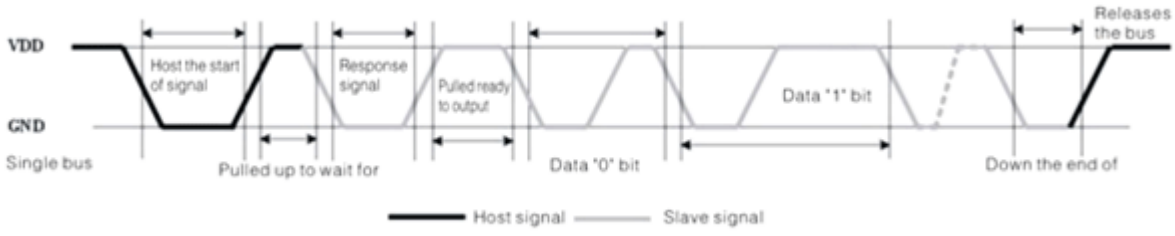
How to use DHT22 Sensor

The **DHT22 Sensor** is factory calibrated and outputs serial data and hence it is highly easy to set it up. The connection diagram for this sensor is shown below.



As you can see the data pin is connected to an I/O pin of the MCU and a 5K pull up resistor is used. This data pin outputs the value of both temperature and humidity as serial data. If you are trying to **interface DHT22 with Arduino Uno** then there are ready made libraries for it which will give you a quick start.

If you are trying to interface it with some other MCU then the datasheet given below will come in handy. The output given out by the data pin will be in the order of 8bit humidity integer data + 8bit the Humidity decimal data +8 bit temperature integer data + 8bit fractional temperature data +8 bit parity bit. To request the DHT11 module to send these data the I/O pin has to be momentarily made low and then held high as shown in the timing diagram below.

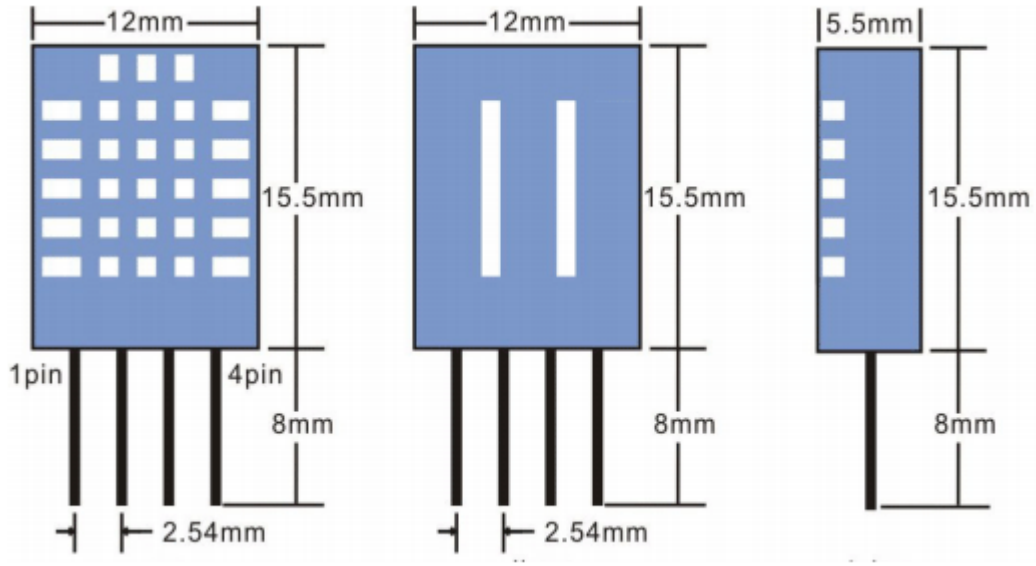


The duration of each host signal is explained in the datasheet, with neat steps and illustrative timing diagrams

Applications

- Measure temperature and humidity
- Local Weather station
- Automatic climate control
- Environment monitoring

2D – model of the sensor



ESP32

ESP32 is an embedded module that supports both WiFi and BT(dual-mode) connectivity and is thus used in **Cloud-based IoT projects**. ESP32 is the upgraded model of the ESP8266 module and is designed by Espressif Systems in China.

The following tables show the main features and technical specifications of the ESP32 module.

ESP32 datasheet library

ESP32 development kit

ESP32 FEATURES & SPECS

ESP32 TECHNICAL SPECIFICATIONS

No.	Parameter Name	Parameter Value
1	Microprocessor	Tensilica Xtensa single-/dual-core 32-bit LX6 microprocessor(s)
2	CoreMark® score	1 core at 240 MHz: 504.85 CoreMark; 2.10 CoreMark/MHz
		2 cores at 240 MHz: 994.26 CoreMark; 4.14 CoreMark/MHz
3	Operating Voltage	3.3V
4	DC Current on 3.3V Pin	50 mA
5	DC Current on I/O Pins	40 mA
6	Maximum Operating Frequency	240MHz
7	Frequency Oscillators	8MHz (Internal Oscillator)
		Internal RC Oscillatoror
		2MHz ~ 60MHz External Crystal Oscillator(40MHz required for WiFi/BT)
		32kHz External Crystal Oscillator(For RTC)
8	Timers	2 x 64-bit Timers, 1 RTC Timer,

ESP32 PINOUT

1	DAC	2 Channels (8-bit, digital to analog converter)
2	ADC	2 Channels (8-bit, digital to analog converter)
3	Capacitive Touch Sensors	10
4	LED PWM	16 Channels

ESP32 COMMUNICATION PROTOCOLS

1	Wi-Fi	802.11 b/g/n (Speed upto 150Mbps)
2	Bluetooth	Supports Classic Bluetooth v4.2 BR/EDR & Bluetooth Low Energy(BLE)
3	Bluetooth Low Energy	Supports BLE
4	UART Protocol	3 Channels
5	SPI Protocol	4 Channels
6	I2C Protocol	2 Channels
7	I2S Protocol	2 Channels (for digital audio)
8	CAN Protocol	1 Channels

ESP32 BUILTIN MEMORY

No.	Parameter Name	Parameter Value
1	SRAM	520kb
2	ROM(Flash Memory)	448kb
3	RTC SRAM	16kb

technopreneur
THE ENGINEERING PROJECTS

Cloud IoT solutions

ESP32 Technical Specifications		
No.	Parameter Name	Parameter Value
1	Microprocessor	Tensilica Xtensa single-/dual-core 32-bit LX6 microprocessor(s)
2 ESP32 development kit	CoreMark® score	1 core at 240 MHz: 504.85 CoreMark; 2.10 CoreMark/MHz
		2 cores at 240 MHz: 994.26 CoreMark; 4.14 CoreMark/MHz
3	Operating Voltage	3.3V
4	DC Current on 3.3V Pin	50 mA
5	DC Current on I/O Pins	40 mA
6	Maximum Operating Frequency	240MHz Wireless communication hardware
7	Frequency Oscillators	8MHz (Internal Oscillator)
		Internal RC Oscillatoror

ESP32 Technical Specifications		
		2MHz ~ 60MHz External Crystal Oscillator(40MHz required for WiFi/BT)
		32kHz External Crystal Oscillator(For RTC)
8	Timers	2 x 64-bit Timers, 1 RTC Timer,

ESP32 Pinout		
1 ESP32 development kit ESP32 pinout guide	DAC	2 Channels (8-bit, digital to analog converter)
2	ADC	18 Channels (12-bit, analog to digital converter)
3	Capacitive Touch Sensors	10
4	LED PWM	16 Channels

ESP32 Communication Protocols		
1 ESP32 development kit	Wi-Fi	802.11 b/g/n (Speed upto 150Mbps)
2	Bluetooth	Supports Classic Bluetooth v4.2 BR/EDR & Bluetooth Low Energy(BLE)
3	Bluetooth Low Energy	Supports BLE
4	UART Protocol	3 Channels
5	SPI Protocol	4 Channels
6	I2C Protocol	2 Channels
7	I2S Protocol	2 Channels (for digital audio)
8	CAN Protocol	1 Channels

ESP32 Builtin Memory		
No. ESP32 development kit	Parameter Name	Parameter Value
1	SRAM	520kb
2	ROM(Flash Memory)	448kb

ESP32 Builtin Memory		
3	RTC SRAM	16kb

So, let's get started with the Introduction to ESP32:

ESP32 development kit

Where To Buy?			
No.	Components	Distributor	Link To Buy
1	ESP32	Amazon	Buy Now

Introduction to ESP32 Module



- **ESP32** (designed by **Espressif Systems**, a Shanghai-based Chinese Company) is a **48 Pin microprocessor-based** embedded IC(available in QFN package), that supports both **WiFi & BT(dual-mode) connectivity** and is used majorly in **wearable devices, mobile & cloud-based IoT applications**.

- The microprocessor used in the ESP32 chip is the Tensilica Xtensa **LX6 microprocessor** (single-core and dual-core).
- A few LX6 based ESP32 ICs are:
    - ESP32-D0WDQ6 (and ESP32-D0WD)
    - ESP32-D2WD
    - ESP32-S0WD
    - ESP32-PICO-D4

ESP32 vs ESP8266

- Both the ESP32 and ESP8266 are inexpensive WiFi modules with low power consumption.
- Both modules are desirable for DIY projects in the areas of IoT (Internet of Things) and automation.
- The ESP32 is a dual-core 160Mhz 240Mhz CPU, while the ESP8266 has an 80Mhz single-core processor. Therefore, if your primary concern is processor speed, you should prioritize the ESP32 over the ESP8266.
- The ESP32 offers more GPIO than the ESP8266.
- ESP32 supports Bluetooth 4.2 and BLE(Bluetooth Low Energy).
- The ESP32 offers a 12-bit ADC, while the ESP8266 offers only a 10-bit ADC.

ESP32-WROOM-32

- **ESP32-WROOM-32** is a **38-pin** breakout board of ESP32, which is most commonly used in third-party ESP32 modules.
- As ESP32 IC is available in the QFN(Quad Flat No Leads) package, so it's quite difficult to solder the IC in embedded projects.

- So, to ease the process of using ESP32 IC, Espressif Systems designed numerous small modules(using ESP32 chip) that have a built-in antenna and easily usable pinout.
- Other ESP32 modules are ESP32-SOLO and ESP32-WROVER.
- One of the most commonly used breakout boards of ESP32 is **ESP32-WROOM-32**, shown in the below figure:



Third-Party ESP32 Development Modules

- Many embedded companies have used ESP32-WROOM-32 and designed different ESP32 development boards, which are plug-and-play modules and are thus normally used for learning and prototyping purposes.
- One of the most commonly used ESP32 development boards is called **ESP32-DevkitC**.
- **ESP32-DevKitC** is a 30-pin ESP32-based development board, designed by Espressif Systems and is used in embedded and IoT projects.



- All you need to do is plug this device using a USB cable and play with it on the fly.
- **Boot mode** and **Reset** buttons are incorporated on the board.
- USB micro connector and USB-UART Bridge, and LDO regulator are also included in the device.

Types of ESP32 Development Boards

- The following are the five different versions of ESP32-DevKitC.
    1. ESP32-DevKitC-32E
    2. ESP32-DevKitC-32UE
    3. ESP32-DevKitCVE
    4. ESP32-DevKitCVIE
    5. ESP32-DevKitCS1

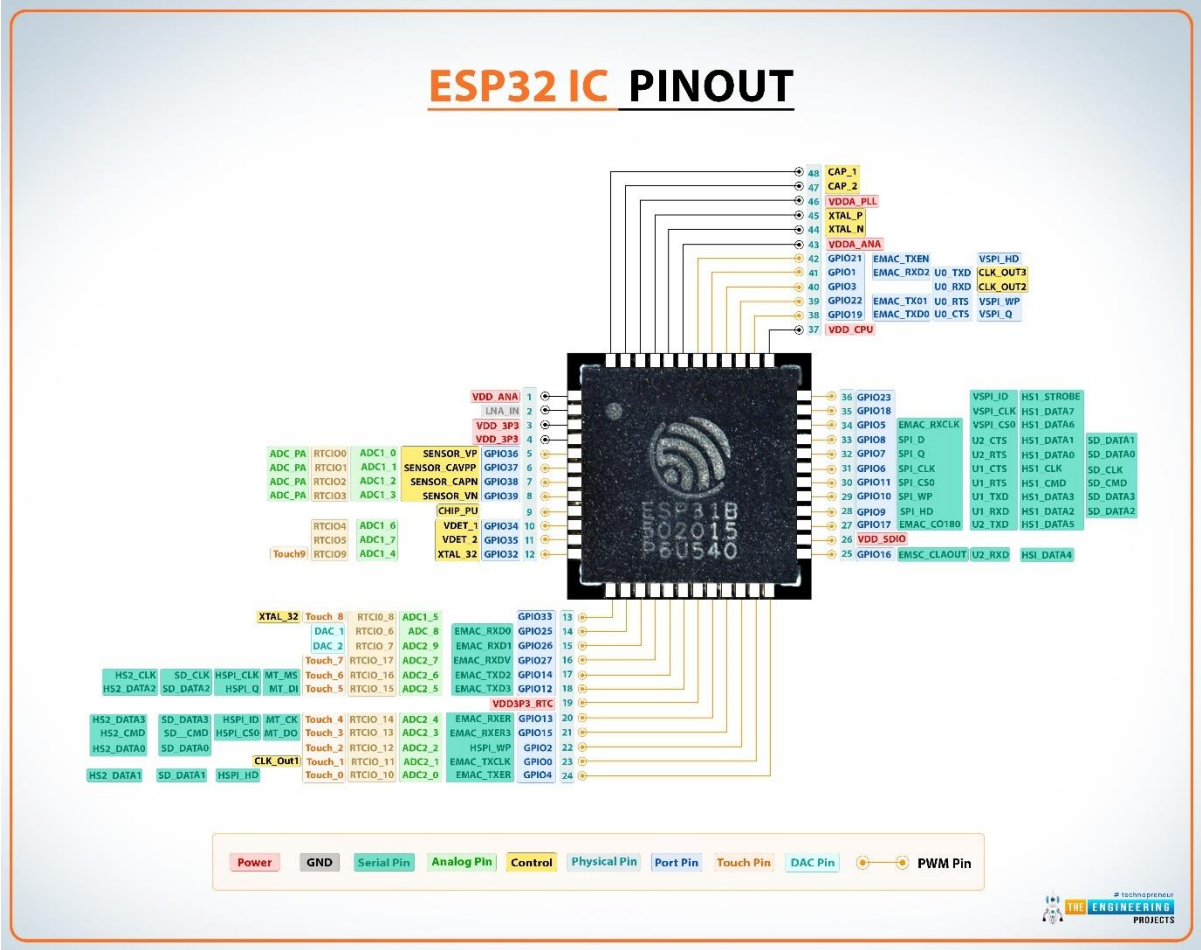
So, that was the evolution of ESP32 from a simple IC to plug & play board. Now let's have a look at the Pinout of the ESP32 microcontroller and modules:

ESP32 Pinout

We have seen above that ESP32 has evolved first into ESP32-WROOM-32 and is further upgraded into ESP32-DevKitC. So, let's have a look at the pinout of all these boards, one by one:

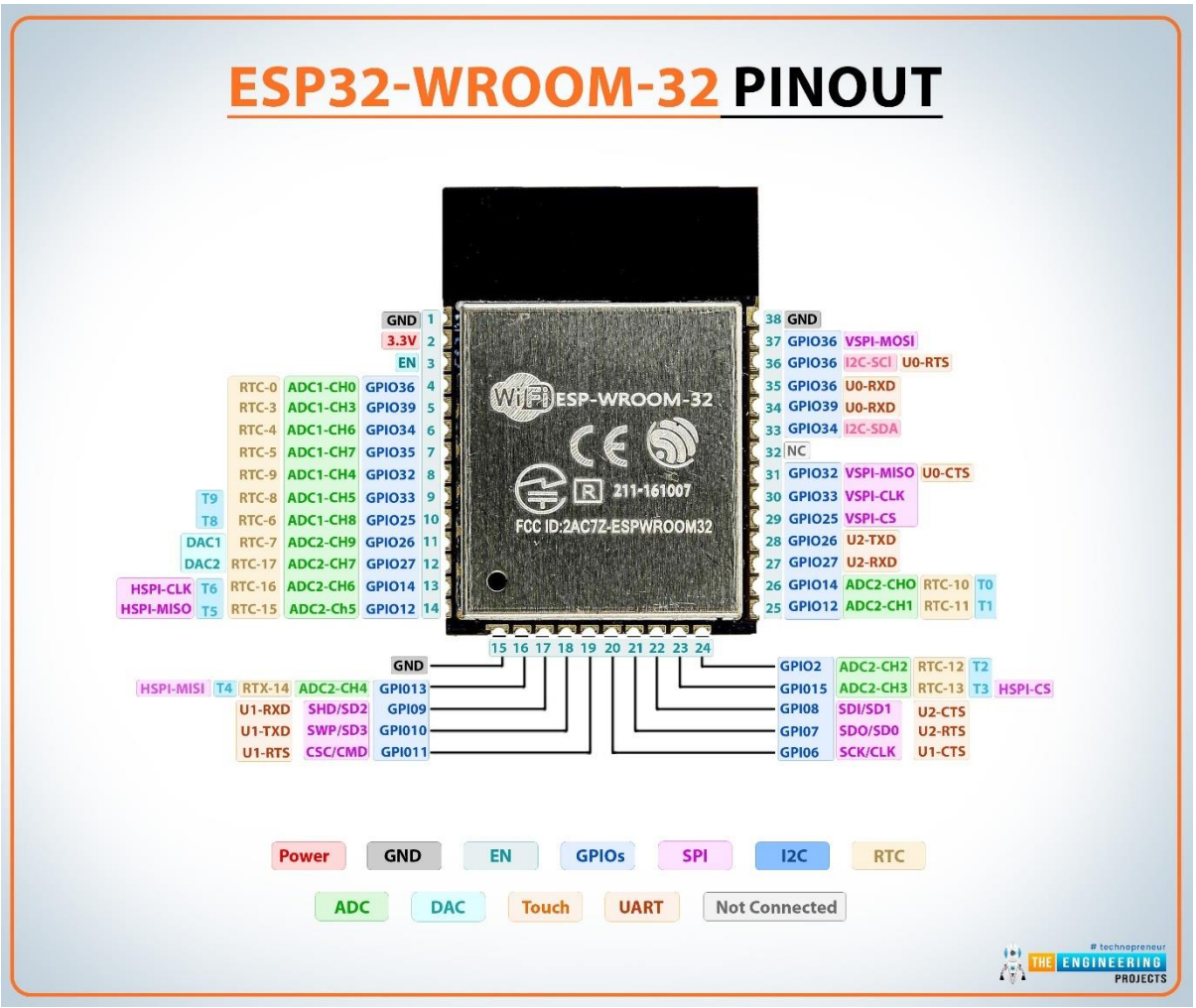
Pinout of ESP32 IC

- ESP32 IC in its pure form consists of **48 pins** in total.
- The following figure shows the labeled **ESP32 Pinout** diagram:



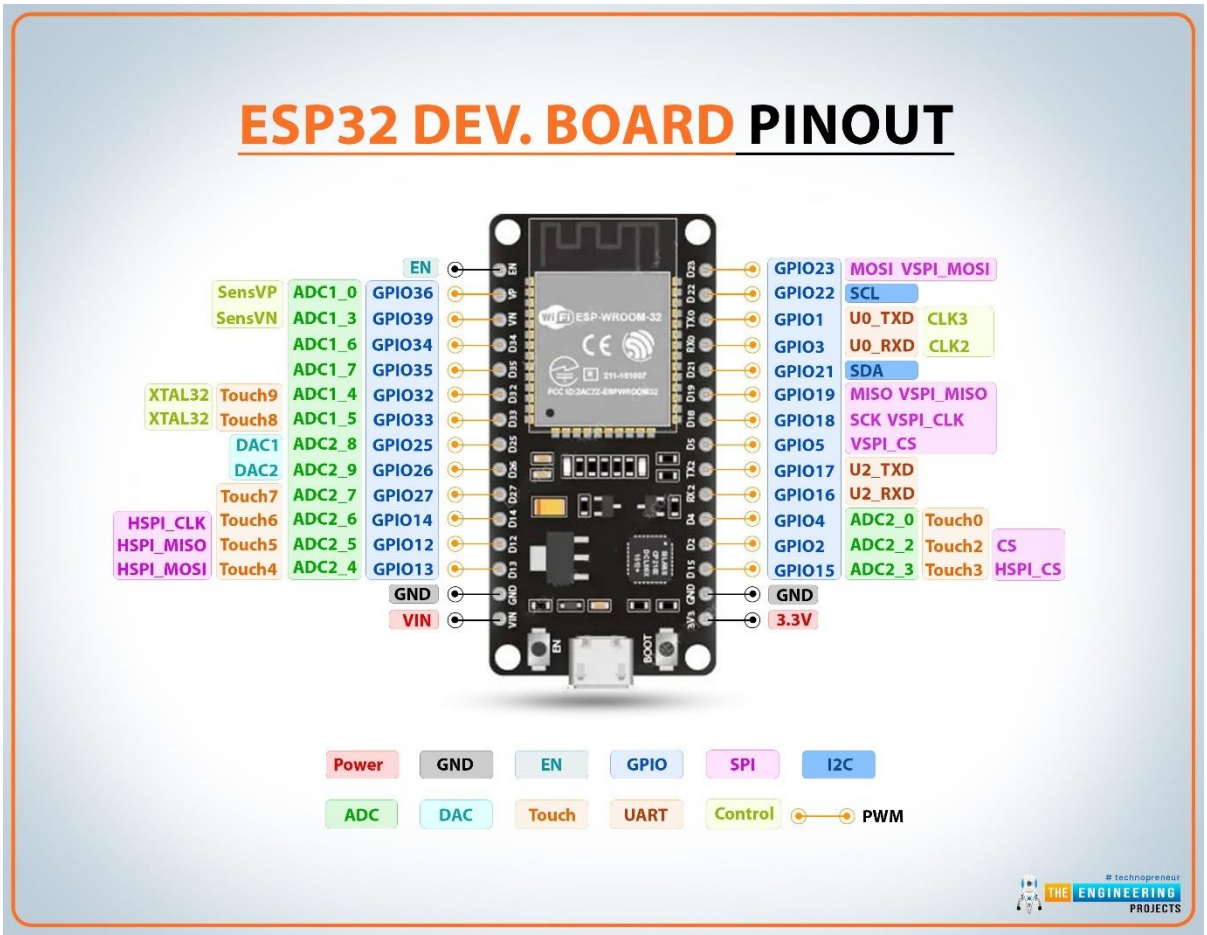
Pinout of ESP32-WROOM-32

- ESP32-WROOM-32 is a breakout board of ESP32 and consists of 38 pins in total.
- Here's the pinout of the ESP-WROOM-32 board:



Pinout of ESP32-DevkitC

- ESP32-DevKitC is a development board based on the ESP32 microcontroller and it has 36 pins in total.
- Here's the pinout diagram of ESP32 DevKitC:



ESP32 Pin Description

Now, let's have a look at the functions of ESP32 Pinout:

Power Pins in ESP32

- **Power:** Power is applied through Micro-USB, 3.3V pin, 5V pin, and GND. Regulated 5V is supplied to this pin which is further regulated to 3.3V to power up the board. And 3.3V pin directly supplies the 3.3V regulated to the board. And the ground is connected to GND.
- **Enable:** The enable pin is represented by ‘En’ on the board and is used to reset the microcontroller.
- **AREF:** It is marked as **AREF** which is used to provide a reference voltage for input voltage.

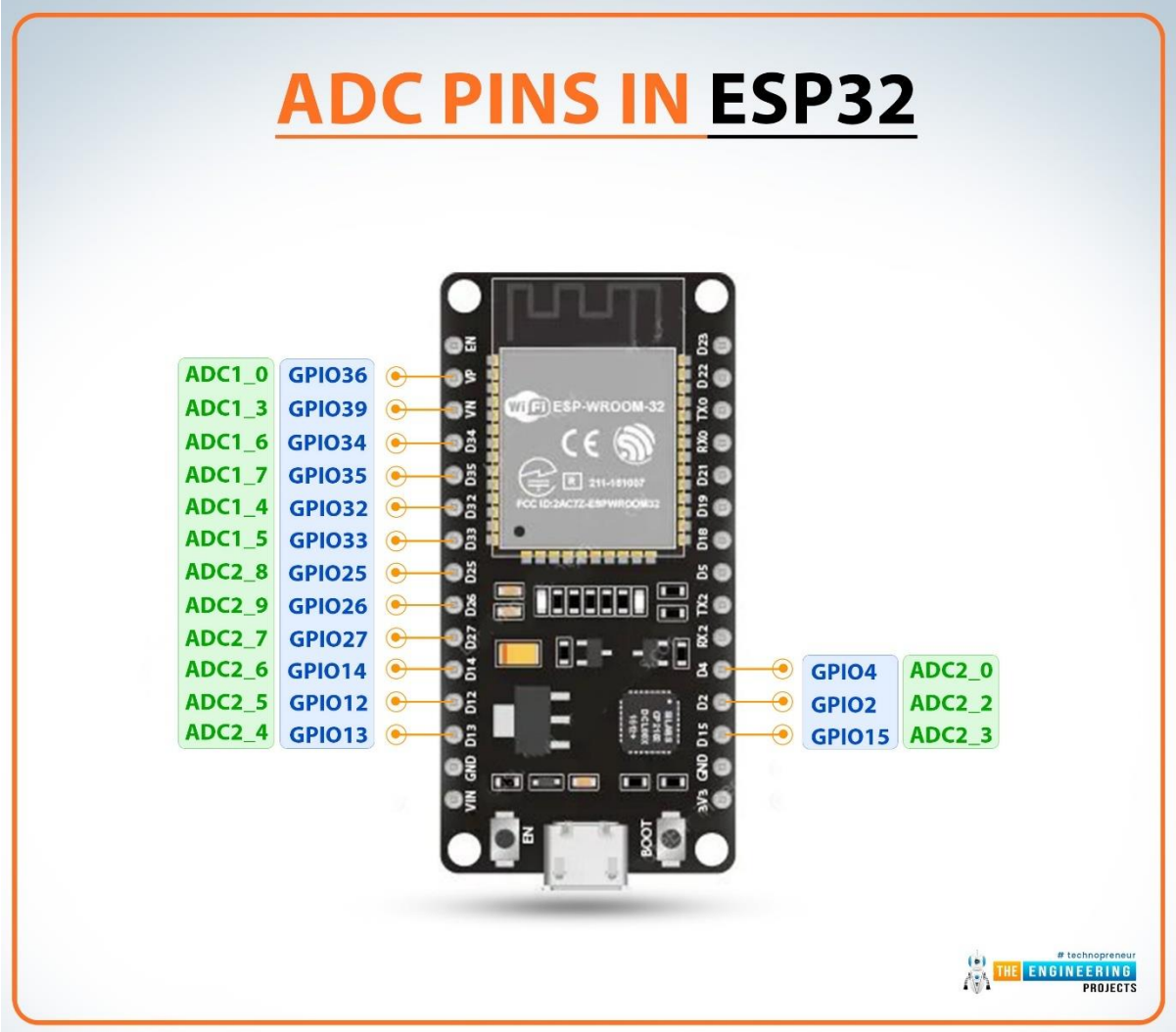
GPIO Pins in ESP32

ESP32 has 36 GPIO(general purpose input/output) pins to perform numerous operations(normally one at a time). Third-party ESP32 modules have different numbers of GPIO pins i.e. ESP32 Dev Kit V1 includes 30 GPIO pins. Let's have a look at the functionality of ESP32 GPIO Pins:

ADC Pins in ESP32

- **ADC Pins:** ESP32 has a total of 18 ADC channels(12-bit each) used to measure the analog voltage within the range of 0-3.3V.

ESP32 is equipped with two SAR analog-to-digital converter modules named ADC1 and ADC2. ADC1 has 10 Channels labeled from **ADC2_1 to ADC2_7**, while ADC2 has 10 Channels labeled from **ADC2_0 to ADC2_9**. The ADC output value ranges from 0 to 4093 at 12-bit resolution.



DAC Pins in ESP32

- **DAC Pins:** ESP32 features 2 distinct 8-bit digital-to-analog converters(DAC1 and DAC2) for translating digital values to analog signals. The DAC function is attached to below two GPIO pins:

1. DAC1-GPIO25
2. DAC2-GPIO26

The DAC employs a power supply as an input reference voltage and features an internal resistance network.



PWM Pins in ESP32

- **PWM Pins:** ESP32's PWM controller has 16 independent PWM channels with configurable frequency and duty cycles. Any **GPIO pin** can be used as a PWM pin.

PWM pulses are used to control the speed of motors or the brightness of LEDs. You can configure the frequency, channel, GPIO pin, and duty cycle of the PWM signal.

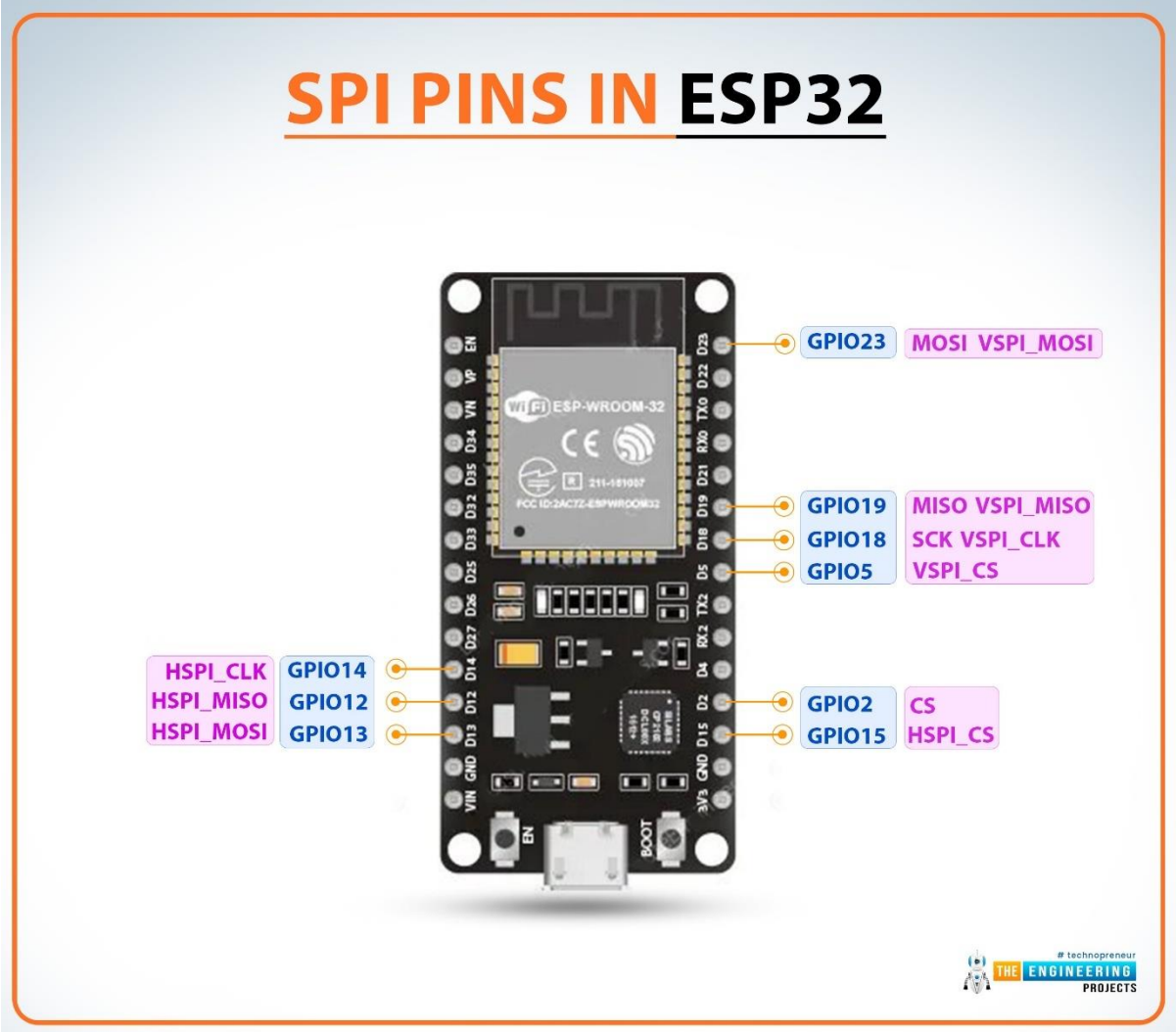


SPI Pins in ESP32

- **SPI Pins:** ESP32 has three SPI blocks that operate in both master and slave modes, named SPI, HSPI, and VSPI.

Among these 3 blocks, SPI is used as an interface to flash memory. So, we are left with HSPI and VSPI for normal use:

1. **VSPI:** ESP32 VSPI Pins are **GPIO23 (MOSI), GPIO19 (MISO), GPIO18 (CLK) and GPIO5 (CS)** used for SPI-1 communication.
2. **HSPI:** ESP32 HSPI Pins are **GPIO13 (MOSI), GPIO12 (MISO), GPIO14 (CLK) and GPIO15 (CS)** used for SPI-2 communication.



I2C Pins in ESP32

The ESP32 has two I2C interfaces. The SCL and SDA pins of both I2C interfaces can be assigned by a user in the program. The default I2C pins are:

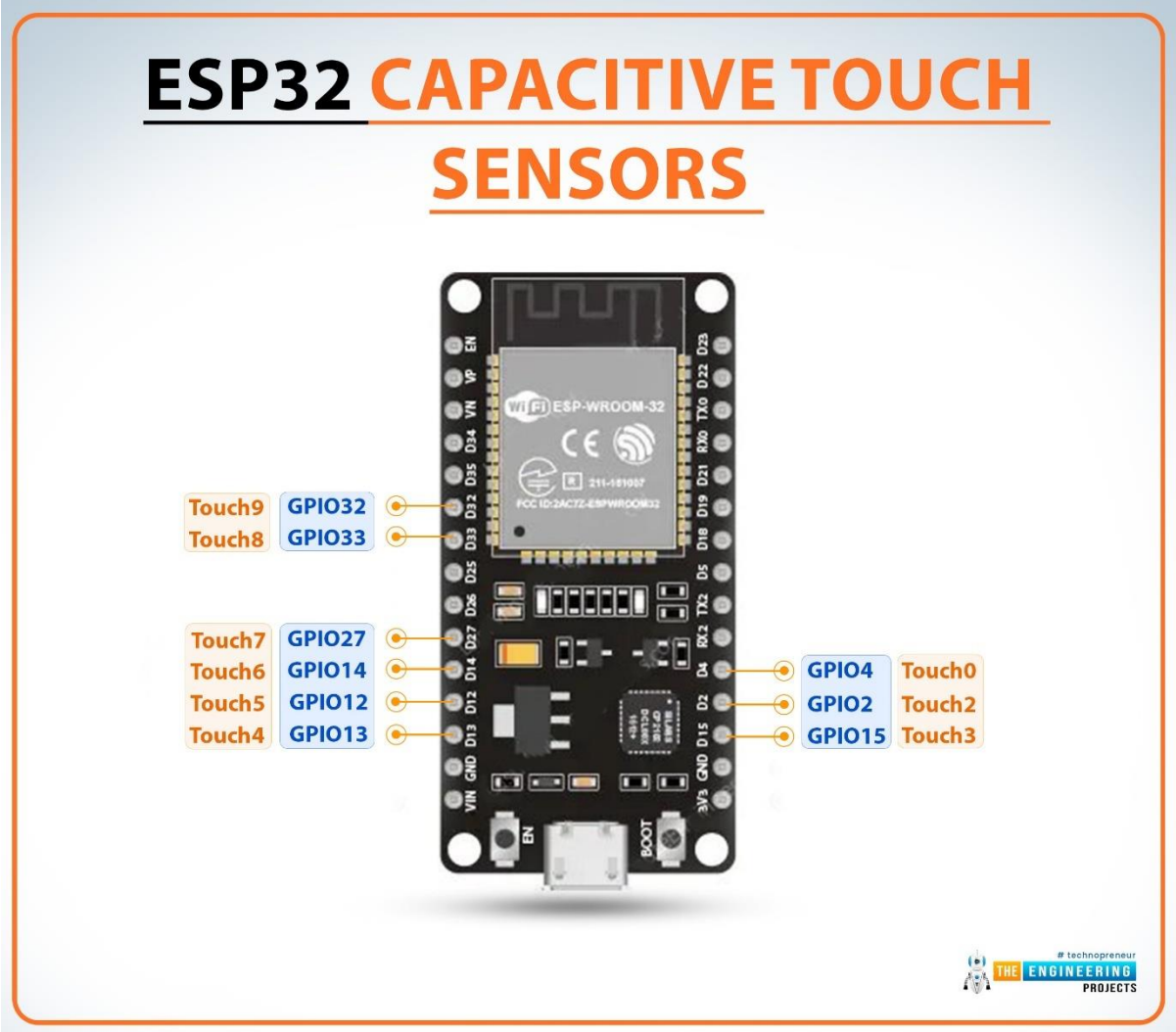
- SDA-GPIO21
- SCL-GPIO22



ESP32 Capacitive Touch Sensors

- ESP32 has 10 capacitive touch-sensing GPIO Pins(**T0 to T9**), which get electrostatically charged when a finger touches the respective GPIO pin.

Without any additional hardware, these touch GPIOs can be utilized to make capacitive touchpads. Variations in capacitance are evident.

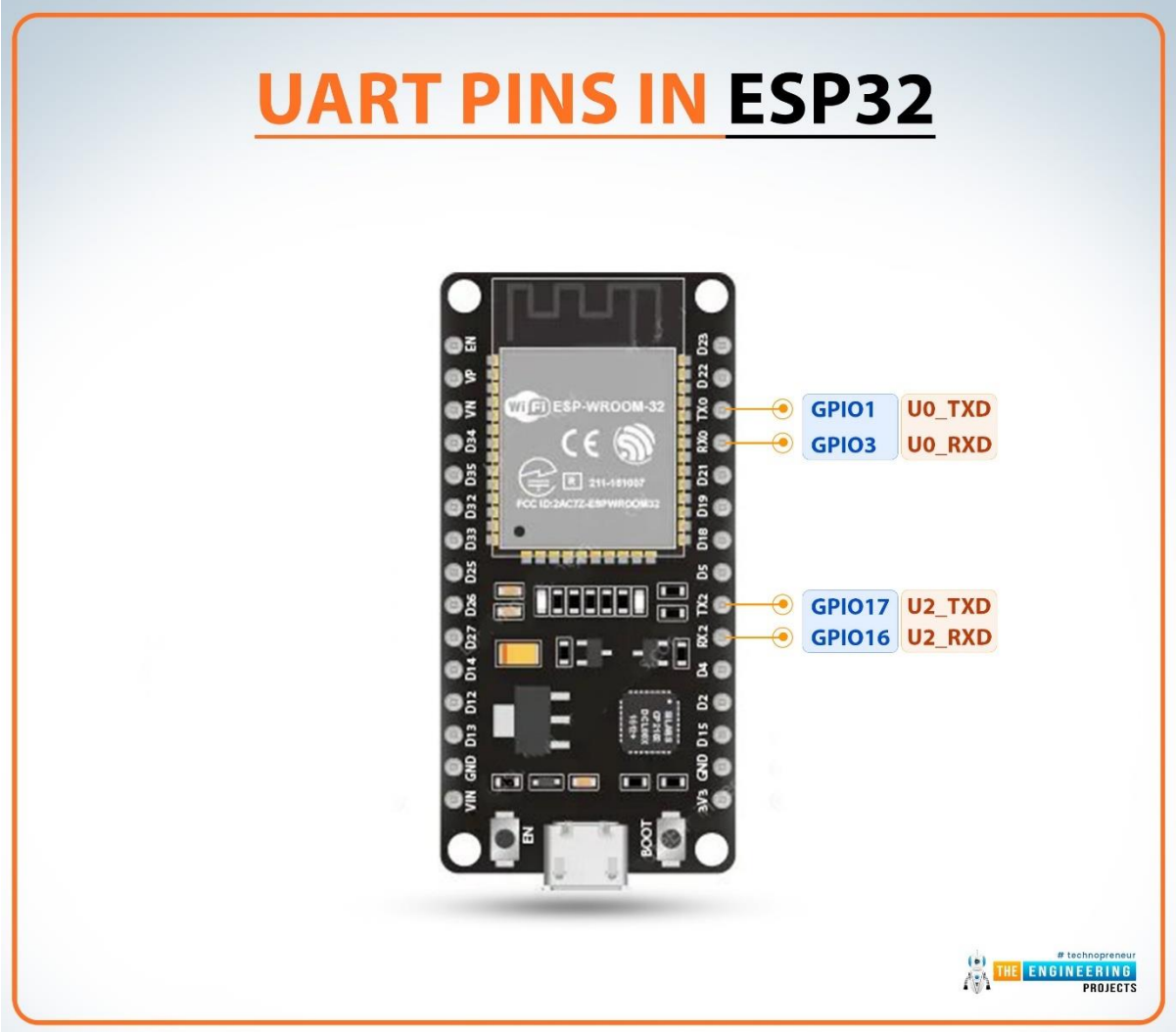


RTC GPIO

- ESP32 has 18 Low-Power RTC GPIO Pins(**RTCIO0 to RTCIO17**) used to wake up the ESP32 board from deep sleep mode.



- **Serial:** Two serial pins are represented on boards as **Tx** and **Rx**. The Tx is used to transmit serial data while Rx is used to receive serial data.



- **External Interrupts:** All **GPIO pins** can be used as external interrupts.

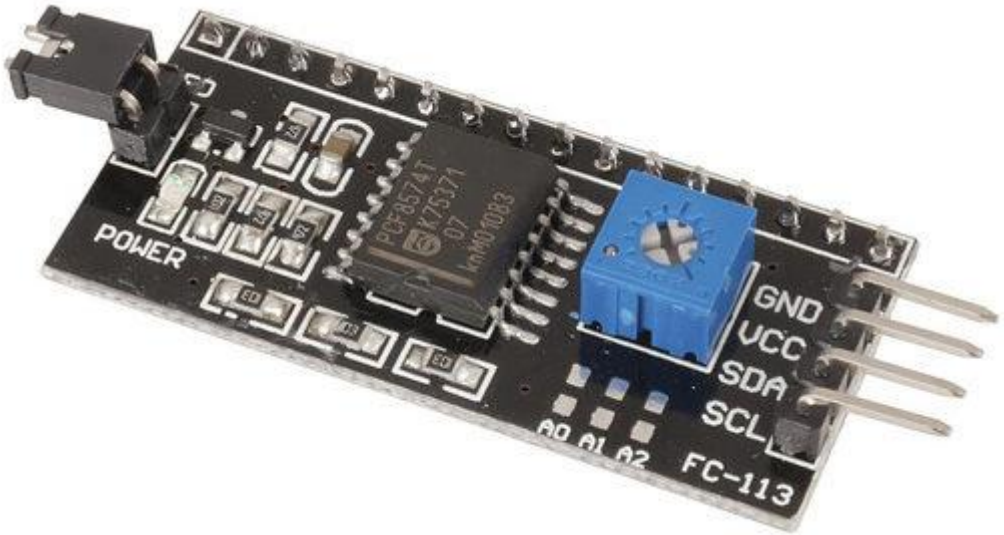
ESP32 Datasheet

Before you incorporate this device into your electrical project, it's wise to go through the datasheet of the component that features the main characteristics of the device. You can click the link given below to download the ESP32 datasheet.

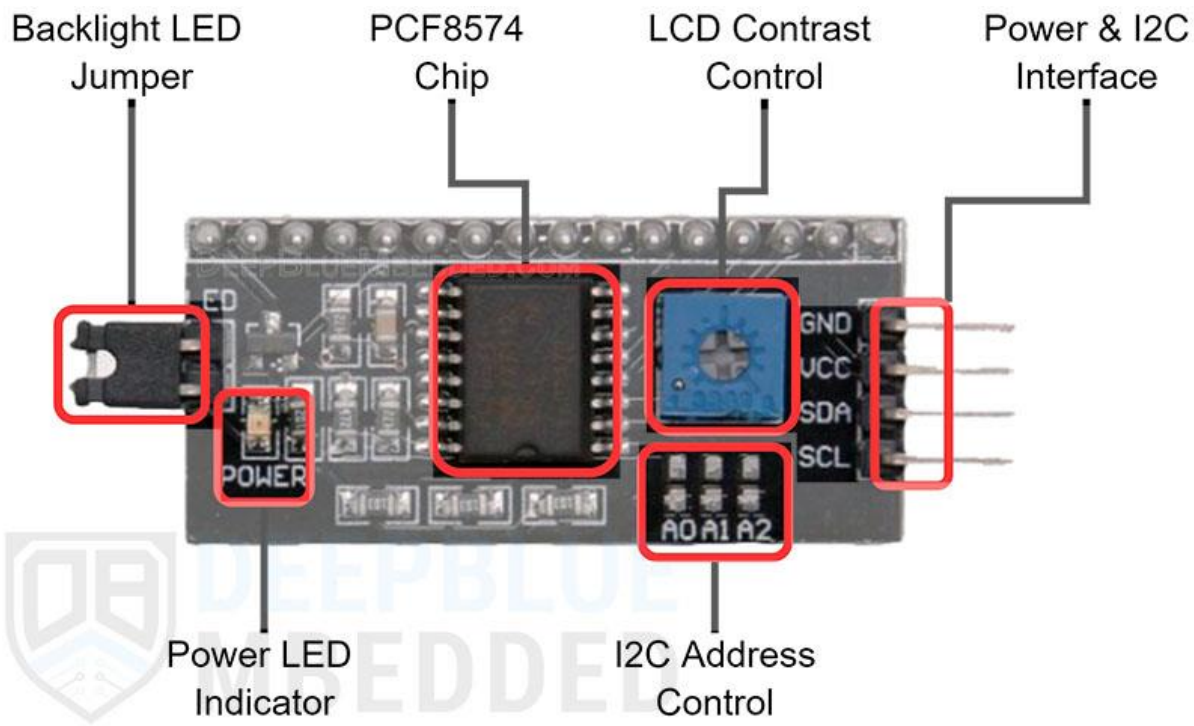
I2C LCD Module (PCF8574 IO Expander)

The I2C IO expander IC (PCF8574) is commonly used as a cheap solution to implement an I2C LCD interface that replaces the classic parallel connection to the LCDs (at least 6 pins) with an easy-to-use

I2C bus (only 2 pins). Using it saves a lot of the STM32 microcontroller's IO pins that would have been consumed to create a generic LCD parallel communication.



Moreover, it's shared with all I2C devices on the bus, so you can still have so many other modules/sensors connected on the same bus.
In **this previous article**, We've demonstrated everything about this IC using its datasheet and build a driver library (in Embedded-C) for this IC. You can check it out if you're interested in learning more about it.



In this tutorial, we'll only discuss the important things that you need to know in order to get started with I2C LCD interfacing with STM32.

I2C LCD Address (Default)

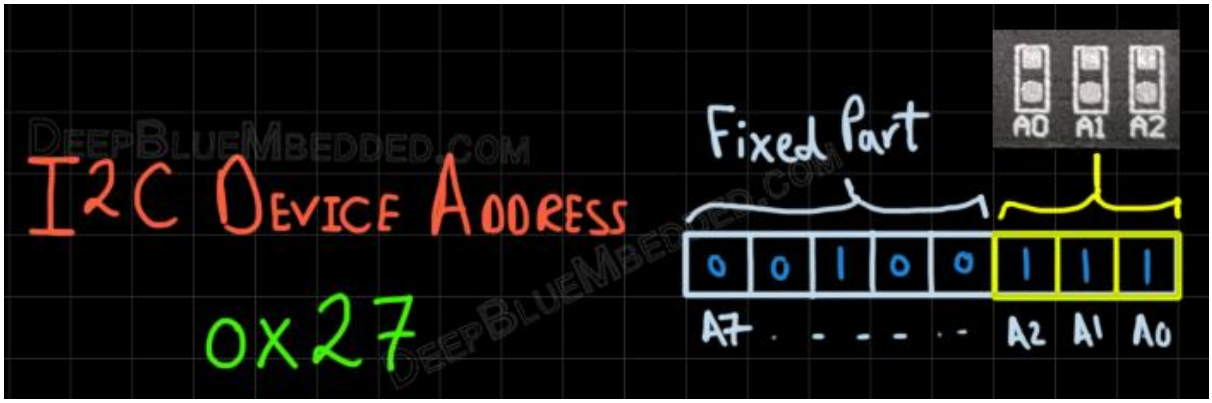
You need to refer to the PCF8574 chip manufacturer's datasheet to make sure what's the I2C device address for the chip you've got around. If you've got an I2C LCD with Ti (**Texas Instruments**) PCF8574 chip, the default I2C address is **0x27** which is the 7-Bit device address with the three least significant bits (A0 – A1 – A2) are pulled up to HIGH.

If you've got an I2C LCD with an **NXP** PCF8574 chip, the default I2C address is therefore **0x3F**. The question is what if we'd like to add multiple I2C LCDs to the I2C bus and control them with an STM32 microcontroller, how is that possible?

First of all, any I2C device on the bus must have a unique address. Therefore, we need to change the I2C address of the PCF8574 module if we'd like to add multiple units of it on the same I2C bus. Let's next see how to do it!

Change I2C LCD Address

The I2C LCD interface (PCF8574) has 3 solder pads on the module's board which control the value of the last 3 digits in the 7-Bit address of the device (A0 – A1 – A2). The pins are by default (internally) pulled up to HIGH (1), but if we short the solder pads together, this will drive the corresponding address bit pin to LOW (0). And that's how we can change the device address.



You can use the interactive tool below to check the I2C LCD device address after soldering any of the solder pads (A0, A1, or A2). There are 8 different combinations, which means we can connect up to 8 different I2C LCDs on the same bus with a single STM32 board and control all of them at the same time.

I2C LCD Address (Texas Instruments' PCF8574)

Address Value:

I2C LCD Address (NXP's PCF8574)

The I2C LCD module has a default I2C device address of either **0x27** or **0x3F** depending on the hardware manufacturer. If you're not quite sure about the device address, you can use this **STM32 I2C Scanner** application to detect the exact device address.

Using the solder pads on the PCD8574 module will enable you to set the low 3 address bits (A0-A1-A2). This means we can have up to 8

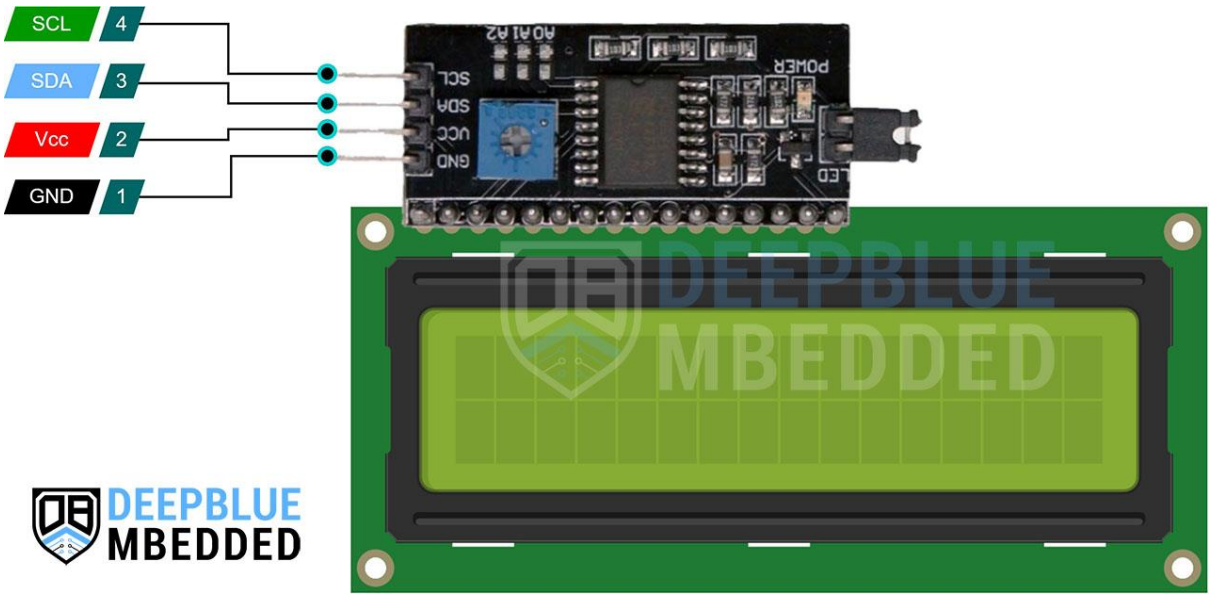
different address combinations and consequently be able to connect up to 8 I2C LCD units on the same I2C bus and control them with only one STM32 microcontroller.

STM32 I2C LCD Interfacing

Now, let's move to interfacing the I2C LCD 16×2 display with STM32. Let's check the pinout, wiring diagram, LCD contrast control, and the I2C LCD device address.

I2C LCD Pinout

The I2C LCD Display has only four pins. The pinout is shown below:



GND is the ground pin.

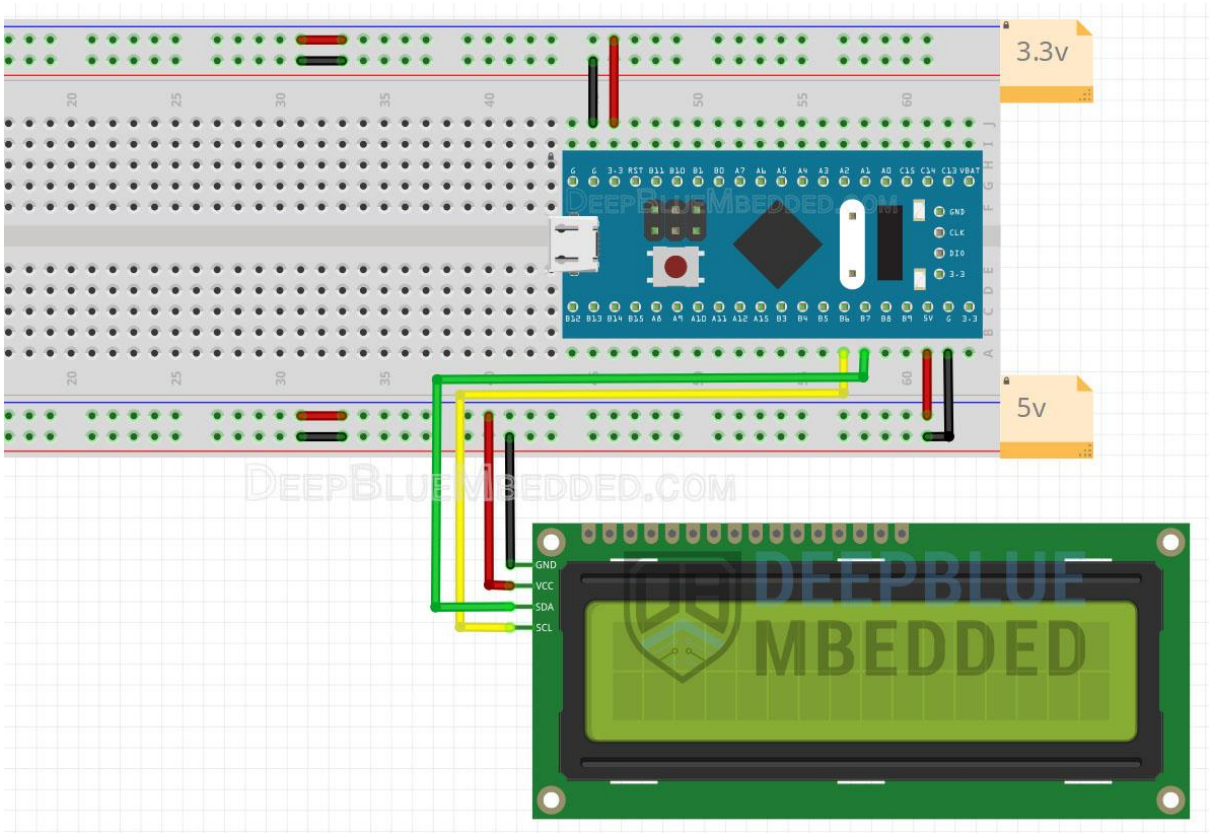
Vcc is the LCD's power supply input pin (connects to +5v).

SDA is the serial data line for the I2C LCD interface.

SCL is the serial clock line for the I2C LCD interface.

Wiring I2C LCD With STM32 Blue Pill

Here is the wiring diagram for the I2C LCD display with STM32 Blue Pill board that we'll be using in the examples hereafter in this tutorial.



Contrast Adjustment For I2C LCD

After connecting the I2C LCD module, you'll be able to control the LCD contrast by using the PCF8574 module's on-board potentiometer. Turn it to the right and to the left until you feel satisfied with the current contrast level.

Get The I2C LCD Address

If you're not quite sure about the I2C LCD's device address, you can use the code example below and run it on your STM32 board after connecting the I2C LCD display to your board. It'll automatically

detect and print out the I2C device address for the LCD over the serial monitor.

```
/*
 * File: I2C_Scanner
 * Author:     Khaled Magdy
 * -----------------------------------------
 * For More Information, Tutorials, etc.
 * Visit Website: www.DeepBlueMbedded.com
 *
 */
#include "main.h"
#include "stdio.h"

I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart1;

uint8_t Buffer[25] = {0};
uint8_t Space[] = " - ";
uint8_t StartMSG[] = "Starting I2C Scanning: \r\n";
uint8_t EndMSG[] = "Done! \r\n\r\n";

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART1_UART_Init(void);
```

```
int main(void)
{
  uint8_t i = 0, ret;

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART1_UART_Init();

  HAL_Delay(1000);

  /*-[ I2C Bus Scanning ]-*/
  HAL_UART_Transmit(&huart1, StartMSG, sizeof(StartMSG), 10000);
  for(i=1; i<128; i++)
  {
    ret = HAL_I2C_IsDeviceReady(&hi2c1, (uint16_t)(i<<1), 3, 5);
    if (ret != HAL_OK) /* No ACK Received At That Address */
    {
      HAL_UART_Transmit(&huart1, Space, sizeof(Space), 10000);
    }
    else if(ret == HAL_OK)
    {
      sprintf(Buffer, "0x%X", i);
```

```
            HAL_UART_Transmit(&huart1, Buffer, sizeof(Buffer), 10000);
        }
    }
    HAL_UART_Transmit(&huart1, EndMSG, sizeof(EndMSG), 10000);
    /*--[ Scanning Done ]--*/
    while (1)
    {
    }
}
```

For more information about how to use this I2C scanner script, check its full tutorial linked below.



STM32 I2C Scanner

This article will give you more in-depth information about the STM32 I2C Scanner application and how to use it to detect various I2C devices' addresses.

STM32 I2C LCD Library (Driver)

Let's now move on to the STM32 I2C LCD library that I've developed for this STM32 series of tutorials which is probably what

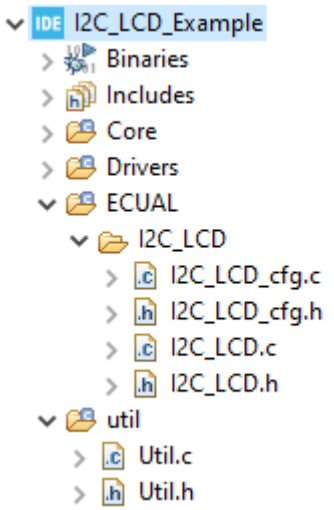
we've searched for before landing on this article. In this section, we'll explore the library files, functions, and configurations. So you can make the best use of it and also know how to modify or add any extra functionalities that your project needs.

In the next section, we'll discuss how to integrate this STM32 I2C LCD library into your STM32 CubeIDE projects step-by-step.

STM32 I2C_LCD Library Directory & Files

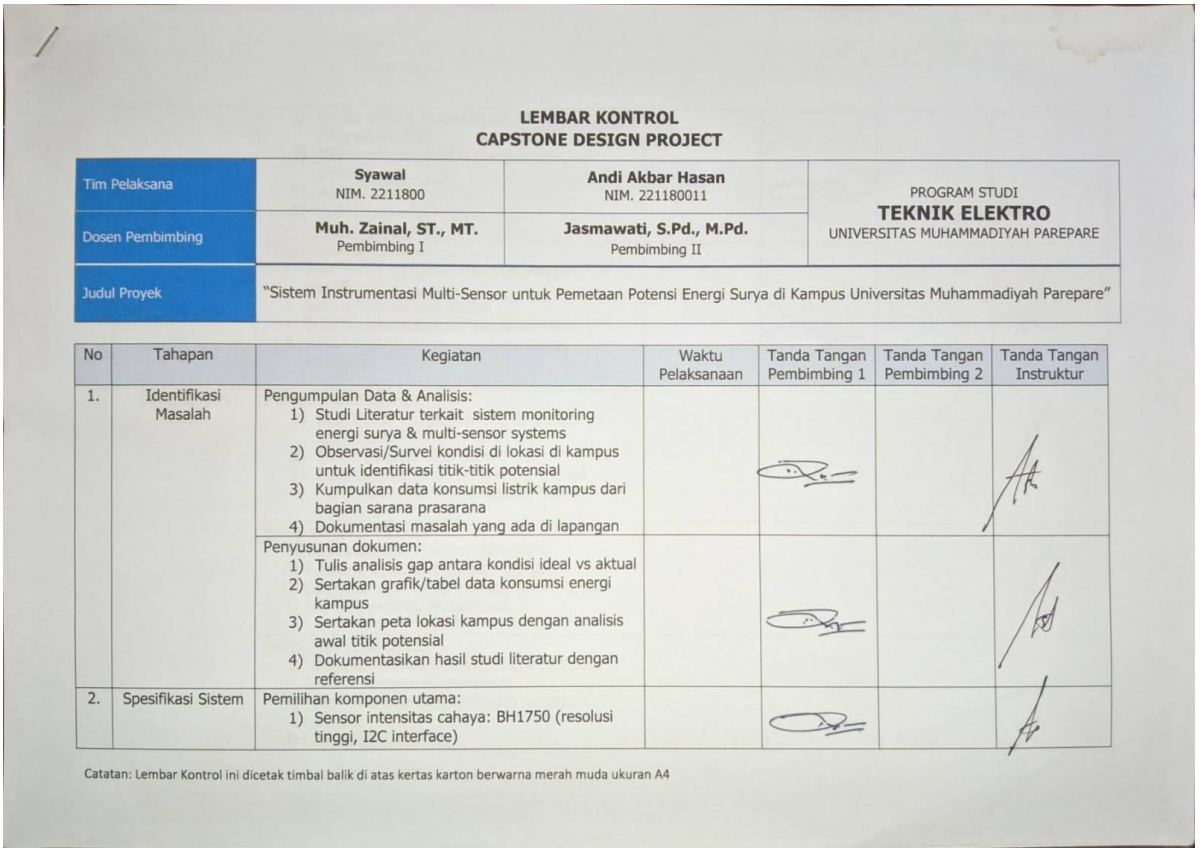
The STM32 **I2C_LCD** library is part of the **ECUAL** (ECU abstraction layer) that we're continuously building through this STM32 Series of Tutorials. The driver consists of 4 files: 2 core files (a header file & a source file) + 2 configuration files (a header & a source).

- I2C_LCD.h
- I2C_LCD.c
- I2C_LCD_cfg.h
- I2C_LCD_cfg.c



The STM32 I2C_LCD library needs you to also include the **util** library as a dependency because it uses the **delay** utility built into the util driver. The util library is also provided in the download section & code examples near the end of this tutorial.

Lampiran-2 Logbook

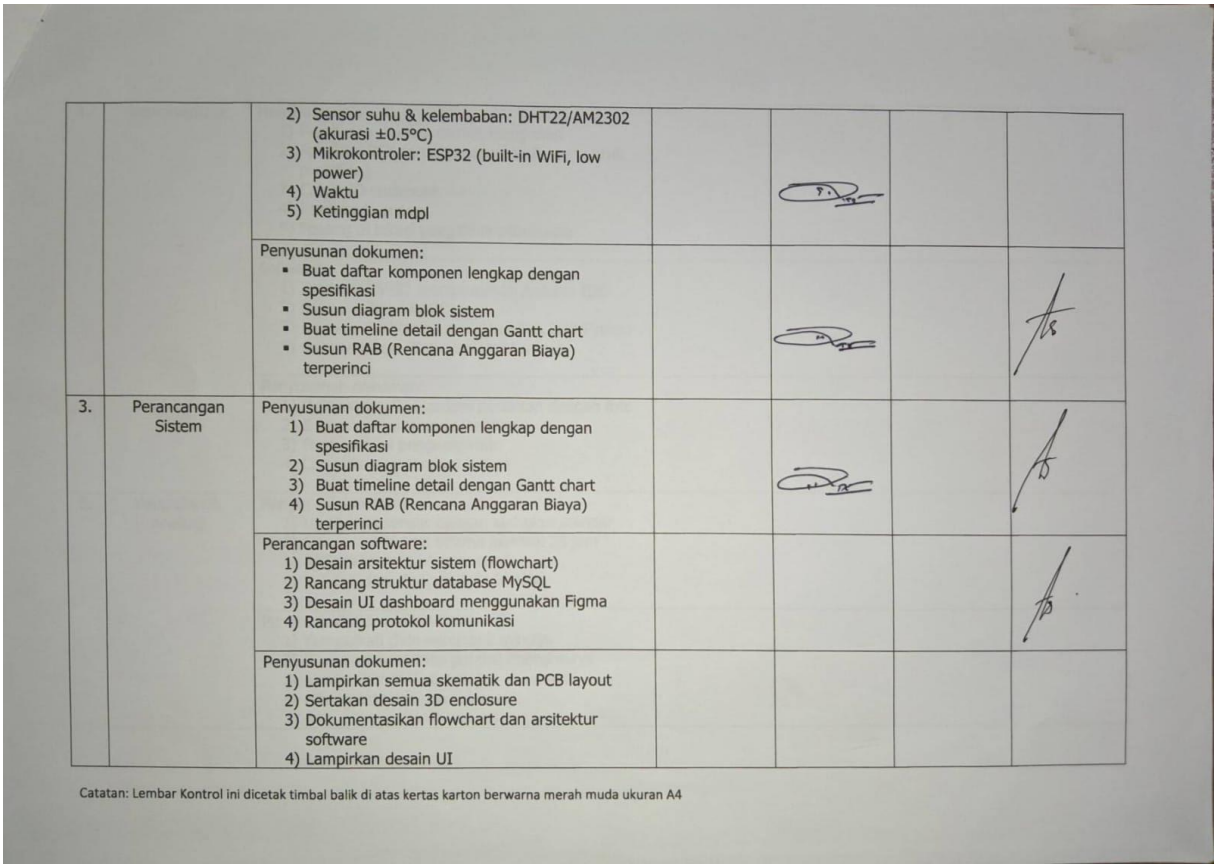


LEMBAR KONTROL
CAPSTONE DESIGN PROJECT

Tim Pelaksana	**Syawal** NIM. 2211800	**Andi Akbar Hasan** NIM. 221180011	PROGRAM STUDI **TEKNIK ELEKTRO** UNIVERSITAS MUHAMMADIYAH PAREPARE
Dosen Pembimbing	**Muh. Zainal, ST., MT.** Pembimbing I	**Jasmawati, S.Pd., M.Pd.** Pembimbing II	
Judul Proyek	"Sistem Instrumentasi Multi-Sensor untuk Pemetaan Potensi Energi Surya di Kampus Universitas Muhammadiyah Parepare"		

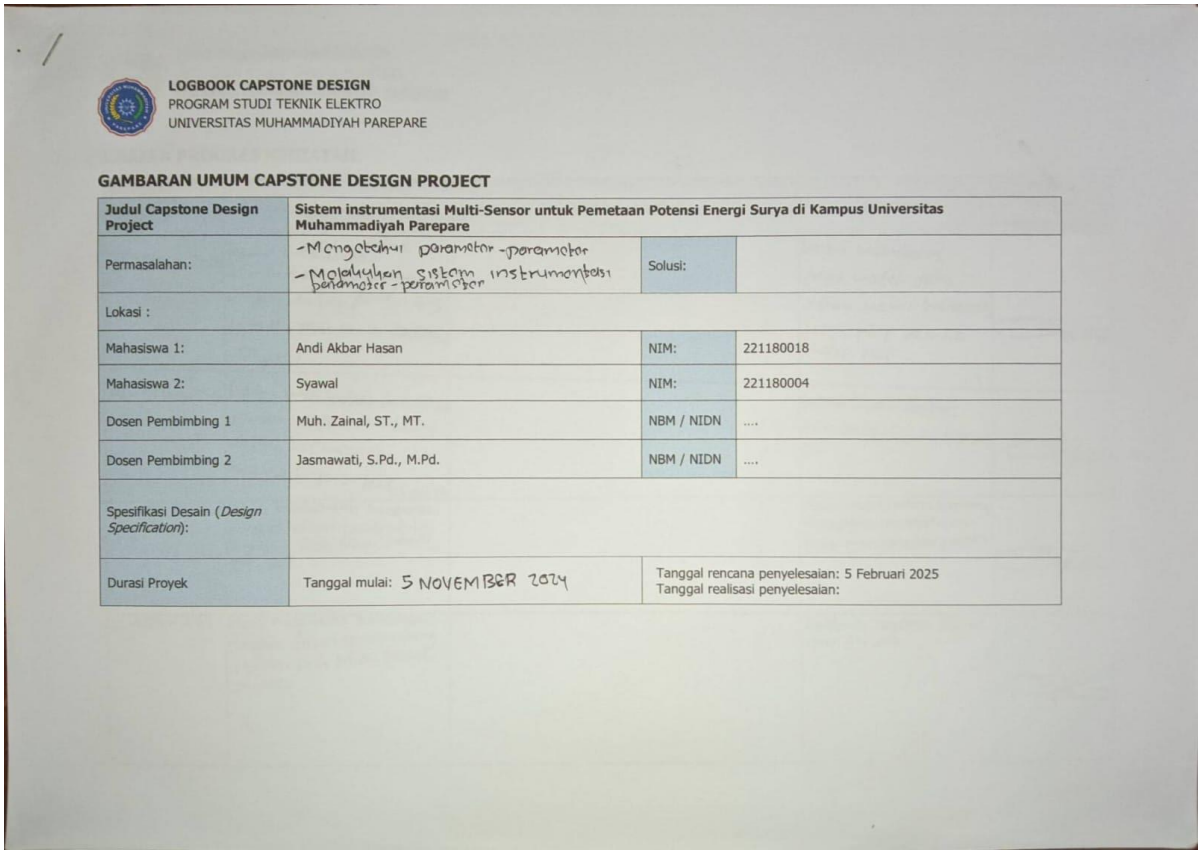
No	Tahapan	Kegiatan	Waktu Pelaksanaan	Tanda Tangan Pembimbing 1	Tanda Tangan Pembimbing 2	Tanda Tangan Instruktur
1.	Identifikasi Masalah	Pengumpulan Data & Analisis: 1) Studi Literatur terkait sistem monitoring energi surya & multi-sensor systems 2) Observasi/Survei kondisi di lokasi di kampus untuk identifikasi titik-titik potensial 3) Kumpulkan data konsumsi listrik kampus dari bagian sarana prasarana 4) Dokumentasi masalah yang ada di lapangan				
		Penyusunan dokumen: 1) Tulis analisis gap antara kondisi ideal vs aktual 2) Sertakan grafik/tabel data konsumsi energi kampus 3) Sertakan peta lokasi kampus dengan analisis awal titik potensial 4) Dokumentasikan hasil studi literatur dengan referensi				
2.	Spesifikasi Sistem	Pemilihan komponen utama: 1) Sensor intensitas cahaya: BH1750 (resolusi tinggi, I2C interface)				

Catatan: Lembar Kontrol ini dicetak timbal balik di atas kertas karton berwarna merah muda ukuran A4



		2) Sensor suhu & kelembaban: DHT22/AM2302 (akurasi ±0.5°C) 3) Mikrokontroler: ESP32 (built-in WiFi, low power) 4) Waktu 5) Ketinggian mdpl				
		Penyusunan dokumen: ▪ Buat daftar komponen lengkap dengan spesifikasi ▪ Susun diagram blok sistem ▪ Buat timeline detail dengan Gantt chart ▪ Susun RAB (Rencana Anggaran Biaya) terperinci				
3.	Perancangan Sistem	Penyusunan dokumen: 1) Buat daftar komponen lengkap dengan spesifikasi 2) Susun diagram blok sistem 3) Buat timeline detail dengan Gantt chart 4) Susun RAB (Rencana Anggaran Biaya) terperinci				
		Perancangan software: 1) Desain arsitektur sistem (flowchart) 2) Rancang struktur database MySQL 3) Desain UI dashboard menggunakan Figma 4) Rancang protokol komunikasi				
		Penyusunan dokumen: 1) Lampirkan semua skematik dan PCB layout 2) Sertakan desain 3D enclosure 3) Dokumentasikan flowchart dan arsitektur software 4) Lampirkan desain UI				

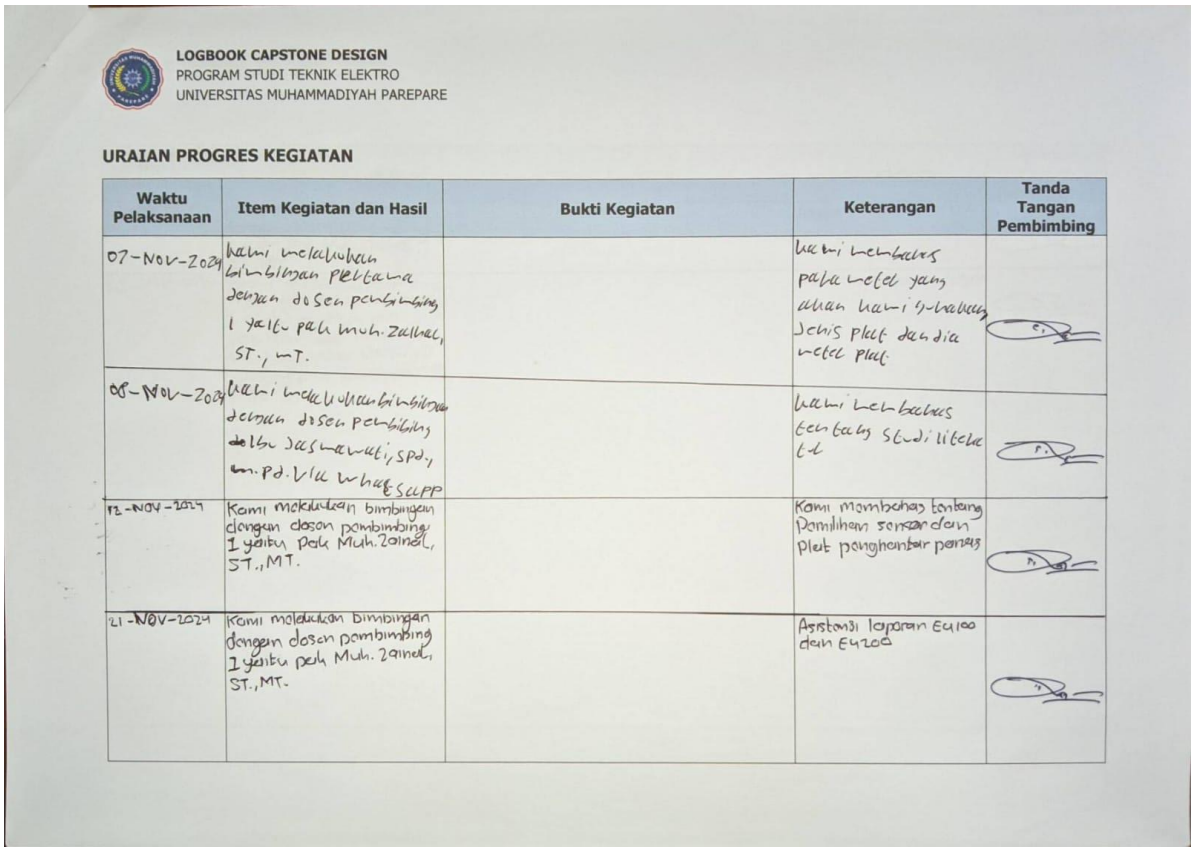
Catatan: Lembar Kontrol ini dicetak timbal balik di atas kertas karton berwarna merah muda ukuran A4



LOGBOOK CAPSTONE DESIGN
PROGRAM STUDI TEKNIK ELEKTRO
UNIVERSITAS MUHAMMADIYAH PAREPARE

GAMBARAN UMUM CAPSTONE DESIGN PROJECT

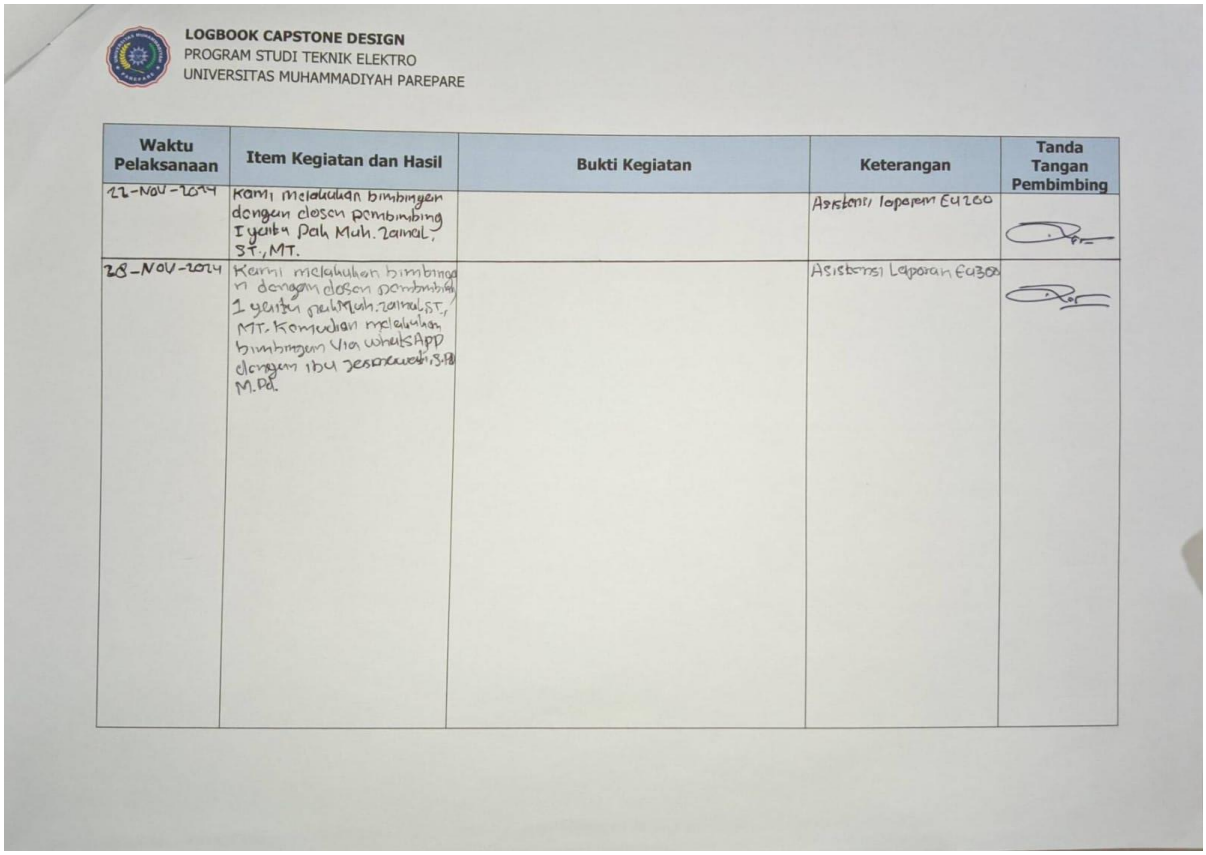
Judul Capstone Design Project	**Sistem instrumentasi Multi-Sensor untuk Pemetaan Potensi Energi Surya di Kampus Universitas Muhammadiyah Parepare**		
Permasalahan:	-Mengetahui parameter-parameter -Melakukan sistem instrumentasi parameter-parameter	Solusi:	
Lokasi :			
Mahasiswa 1:	Andi Akbar Hasan	NIM:	221180018
Mahasiswa 2:	Syawal	NIM:	221180004
Dosen Pembimbing 1	Muh. Zainal, ST., MT.	NBM / NIDN	....
Dosen Pembimbing 2	Jasmawati, S.Pd., M.Pd.	NBM / NIDN	....
Spesifikasi Desain (*Design Specification*):			
Durasi Proyek	Tanggal mulai: 5 NOVEMBER 2024	Tanggal rencana penyelesaian: 5 Februari 2025 Tanggal realisasi penyelesaian:	



LOGBOOK CAPSTONE DESIGN
PROGRAM STUDI TEKNIK ELEKTRO
UNIVERSITAS MUHAMMADIYAH PAREPARE

URAIAN PROGRES KEGIATAN

Waktu Pelaksanaan	**Item Kegiatan dan Hasil**	**Bukti Kegiatan**	**Keterangan**	**Tanda Tangan Pembimbing**
07-NOV-2024	Kami melakukan bimbingan pertama dengan dosen pembimbing 1 yaitu pak Muh. Zainal, ST., MT.		kami membahas para meter yang akan kami gunakan, Jenis plat dan dia meter plat.	
08-NOV-2024	kami melakukan bimbingan dengan dosen pembimbing 2 yaitu Jasmawati, S.Pd., M.Pd. via whatsapp		kami membahas tentang Studi literatur	
12-NOV-2024	Kami melakukan bimbingan dengan dosen pembimbing 1 yaitu pak Muh. Zainal, ST., MT.		Kami membahas tentang pemilihan sensor dan plat penghantar panas	
21-NOV-2024	Kami melakukan bimbingan dengan dosen pembimbing 1 yaitu pak Muh. Zainal, ST., MT.		Asistensi laporan Bab 1 dan Bab 2	



LOGBOOK CAPSTONE DESIGN
PROGRAM STUDI TEKNIK ELEKTRO
UNIVERSITAS MUHAMMADIYAH PAREPARE

Waktu Pelaksanaan	Item Kegiatan dan Hasil	Bukti Kegiatan	Keterangan	Tanda Tangan Pembimbing
22-NOV-2024	Kami melakukan bimbingan dengan dosen pembimbing I yaitu Pak Muh. Zainal, ST., MT.		Asistensi laporan EU260	
28-NOV-2024	Kami melakukan bimbingan dengan dosen pembimbing I yaitu pak Muh. Zainal, ST., MT. Kemudian melakukan bimbingan via whatsApp dengan ibu Jesmawati, S.Pd M.Pd.		Asistensi Laporan EU360	

Lampiran-3 Program

Pengirim

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

#define DHTPIN 4
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

uint8_t gatewayAddress[] = {0x14, 0x2B, 0x2F, 0xEB, 0xC0, 0xF8};

typedef struct struct_message {
  float suhu;
  uint8_t nodeID;
} struct_message;

struct_message dataSuhu;

void OnSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("Status Kirim ke Gateway: ");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "SUKSES" : "GAGAL");
}

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

```
 WiFi.mode(WIFI_STA);
 esp_wifi_set_channel(1, WIFI_SECOND_CHAN_NONE); // Penting: pastikan channel

 dht.begin();

 if (esp_now_init() != ESP_OK) {
  Serial.println("ESP-NOW Init GAGAL");
  return;
 }

 esp_now_register_send_cb(OnSent);

 esp_now_peer_info_t peerInfo = {};
 memcpy(peerInfo.peer_addr, gatewayAddress, 6);
 peerInfo.channel = 1;
 peerInfo.encrypt = false;

 if (esp_now_add_peer(&peerInfo) != ESP_OK) {
  Serial.println("GAGAL tambah peer");
 } else {
  Serial.println("Peer gateway ditambahkan");
 }

 dataSuhu.nodeID = 1; // Ganti sesuai node
}

void loop() {
 float t = dht.readTemperature();
 if (!isnan(t)) {
```

```
    dataSuhu.suhu = t;
    Serial.printf("Mengirim suhu %.1f C dari Node %d\n", t, dataSuhu.nodeID);
    esp_err_t result = esp_now_send(gatewayAddress, (uint8_t *) &dataSuhu,
sizeof(dataSuhu));
    if (result != ESP_OK) {
      Serial.println("Kirim GAGAL");
    }
  } else {
    Serial.println("Gagal baca DHT");
  }
  delay(5000);
}
```

Penerima

```
#include <WiFi.h>
#include <esp_now.h>
#include <esp_wifi.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <SD.h>

// Define SD card chip select pin
#define SD_CS 5

LiquidCrystal_I2C lcd(0x27, 16, 2);  // Ganti ke 0x3F jika LCD tidak tampil

typedef struct struct_message {
```

```
  float suhu;
  uint8_t nodeID;
} struct_message;

struct_message dataSuhu;
float suhuNode[4] = {0, 0, 0, 0}; // Indeks 1-3 untuk node
File dataFile;

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

  // Initialize SD card
  if (!SD.begin(SD_CS)) {
    Serial.println("SD Card Initialization failed!");
    while (1);
  }
  Serial.println("SD card initialized.");

  WiFi.mode(WIFI_STA);
  esp_wifi_set_channel(1, WIFI_SECOND_CHAN_NONE); // Pastikan sama channel-nya
dengan Node

  lcd.begin();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Gateway ESP-NOW");

  if (esp_now_init() != ESP_OK) {
```

```
  Serial.println("ESP-NOW init GAGAL");
  return;
 }

 esp_now_register_recv_cb(onReceiveData);

 // Create data file (overwrite existing file)
 dataFile = SD.open("temperature_data.csv", FILE_WRITE);
 if (!dataFile) {
  Serial.println("Error opening data file");
  while(1); // Stop if file can't be opened.
 } else {
  dataFile.println("Timestamp(ms),NodeID,Suhu(C)"); // Header for the data file
 }
}

void onReceiveData(const uint8_t *mac, const uint8_t *incomingData, int len) {
 char macStr[18];
 snprintf(macStr, sizeof(macStr), "%02X:%02X:%02X:%02X:%02X:%02X",
       mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);

 memcpy(&dataSuhu, incomingData, sizeof(dataSuhu));

 if (dataSuhu.nodeID >= 1 && dataSuhu.nodeID <= 3) {
  suhuNode[dataSuhu.nodeID] = dataSuhu.suhu;

  // Cetak semua data Node ke Serial
  Serial.println("========== DATA MASUK ==========");
  Serial.print("Dari MAC : "); Serial.println(macStr);
```

```
Serial.printf("Node %d update suhu: %.1f C\n", dataSuhu.nodeID, dataSuhu.suhu);
Serial.println(">> Status Suhu Saat Ini:");
Serial.printf("Node 1: %.1f C\n", suhuNode[1]);
Serial.printf("Node 2: %.1f C\n", suhuNode[2]);
Serial.printf("Node 3: %.1f C\n", suhuNode[3]);
Serial.println("================================");

// Update LCD agar sama dengan yang di Serial
lcd.clear();
// Baris 0: Tampilkan Node 1 dan Node 2
lcd.setCursor(0, 0); // Kolom 0, Baris 0
lcd.print("N1:");
lcd.print(suhuNode[1], 1);
lcd.print("C N2:");
lcd.print(suhuNode[2], 1);
lcd.print("C");

// Baris 1: Tampilkan Node 3
lcd.setCursor(0, 1); // Kolom 0, Baris 1
lcd.print("N3:");
lcd.print(suhuNode[3], 1);
lcd.print("C");

// Write data to SD card
if (dataFile) {
  dataFile.print(millis());
  dataFile.print(",");
  dataFile.print(dataSuhu.nodeID);
  dataFile.print(",");
```

```
    dataFile.println(dataSuhu.suhu);
   } else {
    Serial.println("Error writing to SD card");
   }
 }
}

void loop() {
 // Kosong, semua terjadi di callback
}
```

Lampiran-4 dokumentasi

Lokasi 1 (Depan Rektorat)





Lokasi 2 (Parkiran Umpar)





Lokasi 3 (Depan Sekret)





Perbaikan Alat

