SMART HOME AUTOMATION USING IOT

A Project Report Submitted And Partial Fulfilment Of The Requirements For The Award Of Degree Of

POLYTECHNIC

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

K.VIKASH	**G.V.V.S.S.S CHAITANYA**	**M.YESHWANTH**
(21296-EC-037)	**(21296-EC-025)**	**(21296-EC-055)**

Under The Esteemed Guidance of

Ms. N. SIVALEELA, M. Tech

Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CHAITANYA ENGINEERING COLLEGE

(Approved by AICTE) (Affiliated to SBTET, AP)

Madhurawada, Visakhapatnam- 530048, A.P

(2021-2024)

CHAITANYA ENGINEERINF COLLEGE

(Approved By AICTE) (Affiliated To SBTET, AP)

Madhurawada, Visakhapatnam – 530048, A.P

<u>CERTIFICATE</u>

This is to certify that the project titled "**SMART HOME AUTOMATION USING IOT"** is a work done by **K. VIKASH (21296-EC-037), G.V.V.S.S.S CHAITANYA (21296-EC-025), M.YESHWANTH (21296-EC-055)** during the academic year 2021-2024 under our guidance. The work submitted to the "Department Of Electronics And Communication Engineering" in partial fulfillment for the award of the polytechnic in "Electronics And Communication Engineering".

PROJECT GUIDE	**HEAD OF SECTION**
Mrs. S. SIVALEELA, M.Tech	**Mrs. S. SIVALEELA, M.Tech**
Assistant professor	Assistant professor
ECE Department, CEC	ECE Department, CEC

ACKNOWLEDGEMENT

We wish to express our deep sense of gratitude to our guide **Ms. S. SIVALEELA, MTech.** Assistant Professor, Chaitanya Engineering College, for giving us an opportunity to work under him by giving valuable suggestions and kind cooperation, guidance throughout the time.

With a great sense of pleasure and privilege, we extended our gratitude to **Ms.S.SIVALEELA, M. Tech** Assistant Professor & Head of The Section, Electronics and Communication Engineering, Chaitanya Engineering College, for her cooperation and encouragement.

We would like to express our sincere thanks to Diploma principal **Dr. G. Bhanu Praveen, PH.D** and Management members, Chaitanya Engineering College, Visakhapatnam for their encouragement.

Our sincere thanks to all faculty members and non-Teaching staff for their valuable suggestions and creative criticism on the project.

Finally, we wish to take this opportunity to express our deep gratitude to all our friends and family members who have extended their cooperation in various ways during the project work. It is our pleasure to acknowledge the help of all those individuals.

K. VIKASH

(21296-EC-037)

G.V.V.S.S.SCHAITANYA
(21296-EC-025)

M.YESHWANTH

(21296-EC-055)

ABSTRACT

The "Smart Home Automation Using IOT" project aims to revolutionize traditional home management by integrating Internet of Things (IOT) technologies to create an intelligent and automated home environment. This project leverages the power of interconnected devices and sensors to enhance the overall efficiency, security, and comfort of residential spaces.

Home automation or smart home is basically a wireless smart controlling or operating of all home appliances like lights, fans, doors, refrigerators, washing machines etc… where all the devices which are connected to the internet , that they are the part of internet of things. These thing helps people to monitor their home from anywhere by these we can save electricity, water, soon. We can monitor our home from different place or different country too. It is a basic ESP32 (ESP-01) WIFI module .These helps to easy human efforts and to maintain comfortable life within a home, by monitoring their home and automation utilizing. And home automation topic in IOT gaining more popularity in nowadays, because of many advantages. Everyone can achieve home automation by simply connecting their electrical devices to the internet or cloud storage and it is simplicity and comparable affordability. It is used in user friendly manner using custom defined portals, where cloud act as a front end to access IOT. we can easily automate our home by using phone from anywhere. By using the gas sensor, if any gas is leake’s in our house all the electronic appliances are automatically turn off and we get the email notification alert.

CONTENTS

LIST OF THE FIGURES

LIST OF THE TABLES

LIST OF THE ABBRIVATIONS

- TTL Logic - Transistor–transistor logic
- AO – Analog Output
- DO – Digital Output
- LED – Light Emitting Diode
- SIG – Signal
- ULP – Ultra – Low Power
- RAM – Random Access Memory
- RTC – Real-Time Clock
- PWM – Pulse-Width Modulation
- UART - Universal Asynchronous Receiver-Transmitter
- GPIO - General-Purpose Input/Output
- ADC – Analog to Digital Converter
- HTML – Hyper Text Markup Language
- CSS - Cascading Style Sheets
- JS – Java Script
- HTTP - Hyper Text Transfer Protocol
- SMTP - Simple Mail Transfer Protocol

CHAPTER-1

INTRODUCTON

The "Smart Home Automation Using IOT" project represents a significant advancement in home management systems by incorporating Internet of Things (IOT) technologies. This innovative system aims to enhance convenience, efficiency, and safety within residential spaces by integrating interconnected devices and sensors with cloud-based communication.

Working Principle:

The system's core functionality revolves around a web interface that allows users to control various home appliances such as LEDs and motors using switches on a webpage. These switches serve as virtual controls that communicate with the IoT-enabled microcontroller, which in turn triggers actions in response to user commands. For instance, when a user toggles a switch for an LED or motor on the webpage, the corresponding signal is sent to the microcontroller via the IoT network. The microcontroller then activates the appropriate relay or actuator to turn on/off the LED or motor accordingly. One of the key features of this smart home system is its ability to detect hazardous situations, such as the presence of flammable gas in the house. When a gas sensor detects flammable gas above a certain threshold, the system initiates an automatic shutdown of all connected electronics to prevent any potential hazards.

Simultaneously, it sends an alert notification to the user's email address, informing them of the situation. Additionally, a buzzer is activated to provide an audible warning, ensuring immediate attention to the safety concern. This intelligent automation not only provides convenience in everyday tasks but also prioritizes safety by proactively responding to potential dangers within the home environment. By leveraging IoT technologies and cloud communication, the system offers a comprehensive and efficient solution for modern smart homes, enhancing overall comfort, security, and peace of mind for homeowners.

1.1 PROJECT OVERVIEW

The "Smart Home Automation Using IOT" project aims to modernize and optimize home management through the integration of Internet of Things (IOT) technologies. The project focuses on leveraging interconnected devices and sensors to create an intelligent and automated home environment. By utilizing IOT capabilities, homeowners can remotely control and monitor various aspects of their homes such as lighting, temperature, appliances, and security systems. The goal is to enhance efficiency, security, and comfort while also promoting resource conservation through smart energy and water usage. The project emphasizes scalability, affordability, and user-friendly interfaces to ensure accessibility to a wide

Key Components and Features:

IOT-enabled Microcontrollers: Utilizing microcontrollers such as Arduino or ESP32 allows for the integration of sensors and actuators, enabling intelligent decision-making and automation. range of users. Overall, the project seeks to revolutionize traditional home living by harnessing the power of IOT for improved quality of life and sustainable practices.

Sensors: Different sensors including temperature, motion, gas, and light sensors are deployed to gather data about the home environment, detect anomalies, and trigger appropriate actions.

Actuators/Relays: Actuators or relays are used to control home appliances such as lights, fans, doors, and motors based on commands received from the microcontroller or user inputs.

WiFi Modules: WiFi module such as ESP32 facilitate wireless communication between the microcontroller and the internet, enabling remote monitoring and control capabilities.

Cloud Platform Integration: The system integrates with cloud-based platforms to store data, enable remote access, and provide real-time notifications and alerts to users.

Web Interface/Mobile App: A user-friendly web interface or mobile application allows homeowners to monitor home status, control devices remotely, set automation schedules, and receive alerts on their smartphones or computers.

-

CHAPTER-2

PROBLEM DEFINITION

The "Smart Home Automation Using IOT" project addresses the limitations of traditional home management systems by integrating IOT technologies. Traditional systems lack centralized control, remote monitoring capabilities, and efficient resource utilization, leading to inconvenience and inefficiencies for homeowners. By leveraging IOT devices such as sensors and interconnected appliances, the project aims to streamline home automation, enhance energy efficiency, improve security measures, and enable remote monitoring and control via cloud-based platforms. The objective is to create a seamless and user-friendly experience that allows homeowners to monitor and manage their homes from anywhere, leading to cost savings, enhanced security, and overall comfort within the home environment.

2.1 BLOCK DIAGRAM

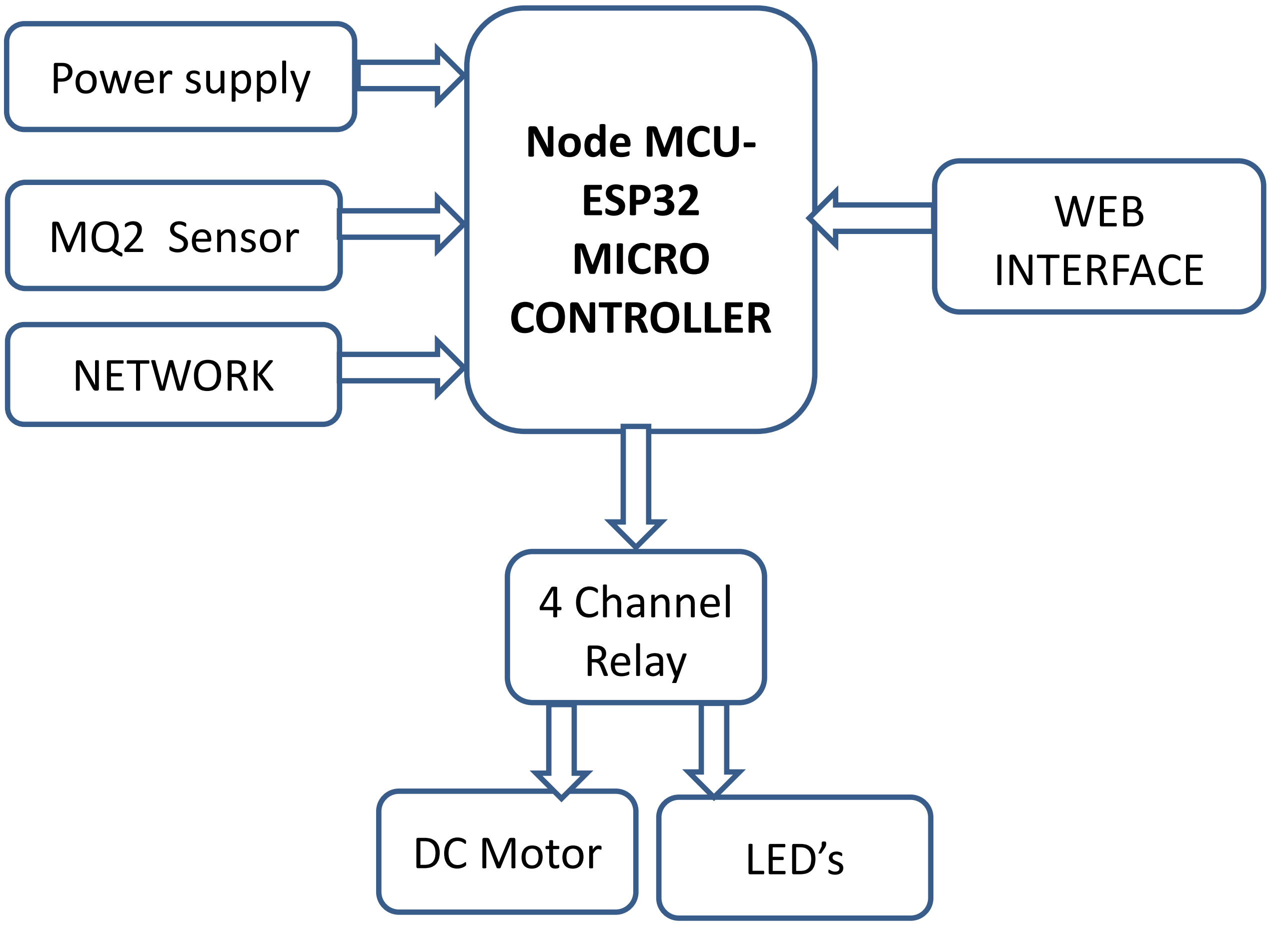


Fig 2.1 Block Diagram of SMART HOME AUTOMATION USING IOT

Here, ESP32 acts as a heart of our project, in the above block diagram we can see that MQ2 gas sensor is acts as input interface to the micro controller and 4 channel relay, DC motor and LEDs are acts as output interface to the micro controller and web interface used to control the output interfaced devices like motor and led. , here the input and output interface can be indicated with the arrow lines with respective to the micro controller performs with the respective commands and delay which is programmed on Arduino IDE software.

2.2 CIRCUIT DIAGRAM OF THE SYSTEM

The circuit diagram features a Node MCU ESP32 microcontroller at its core, serving as the central processing unit. The Node MCU is connected to the MQ2 gas sensor for detecting the gas leakage in the home, Buzzer is used for when gas is leaked the buzzer will sounds and the 4 channel relay is also connected to motor and led for control the motor and led by using the web interface. A 5v power supply is essential to power the microcontroller, sensors, WiFi module, and other components in the circuit. The WiFi module facilitates communication between the microcontroller and the cloud platform. The clo ud platform acts as a bridge between the user interface and the IOT devices, allowing users to control appliances, and receive notifications remotely.

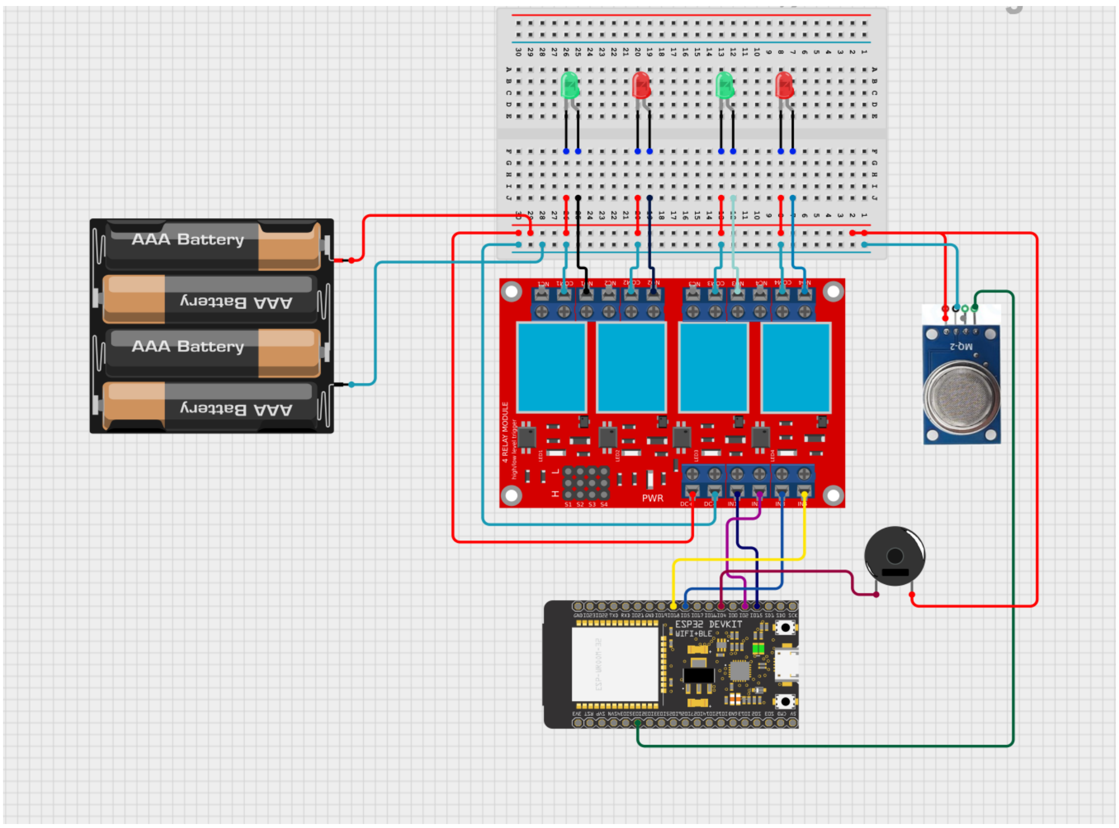


Fig 2.2 CIRCUIT DIAGRAM OF THE SYSTEM

CHAPTER-3

REQUIREMENT SPECIFICATION

3.1 HARDWARE REQUIREMENTS

3.1.1.1MQ2 GAS SENSOR

The MQ2 gas sensor is a small electronic device designed to detect various gases in the air. It operates by measuring the concentration of gases such as methane, propane, carbon monoxide, and smoke.

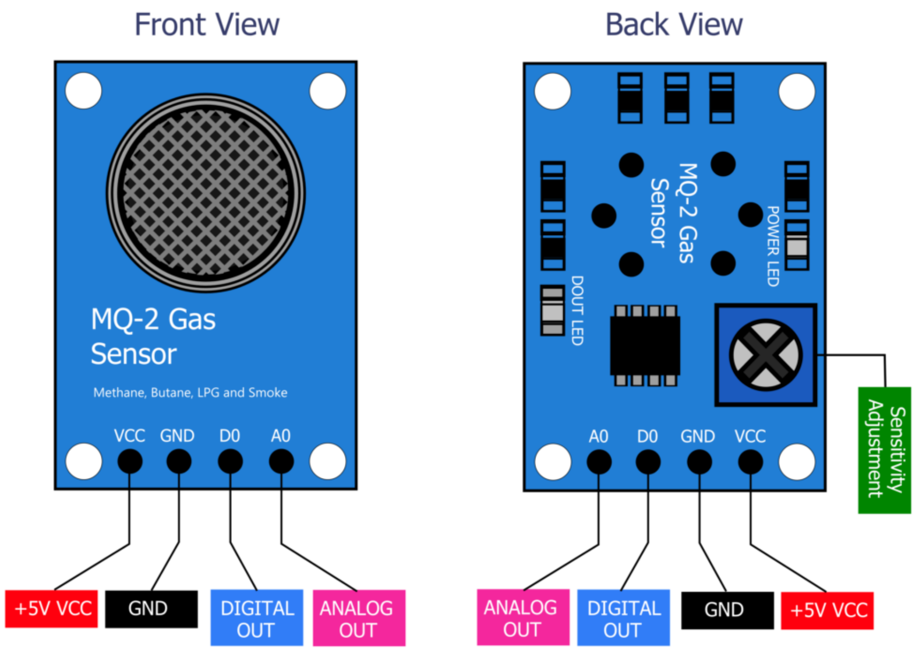


Fig 3.1.1 MQ2 gas sensor module

This sensor is commonly used in applications like gas leakage detectors, smoke alarms, and air quality monitoring systems. The MQ2 sensor works by changing its electrical resistance in response to the presence of different gases, and this change is then converted into a signal that can be interpreted by a microcontroller or other electronic devices. Its compact size and sensitivity make it a valuable tool in ensuring safety and maintaining air quality in various environments.

3.1.1.2Features of MQ2 Gas Sensor

Hardware Features

1. Operating Voltage is +5V
2. Can be used to Measure or detect LPG, Alcohol, Propane, Hydrogen, CO and even methane
3. Analog output voltage: 0V to 5V

4. Digital Output Voltage: 0V or 5V (TTL Logic)
5. Preheat duration 20 seconds
6. Can be used as a Digital or analog sensor
7. The Sensitivity of Digital pin can be varied using the potentiometer

3.1.1.3Applications

1. Detects or measure Gases like LPG, Alcohol, Propane, Hydrogen, CO and even methane
2. Air quality monitor
3. Gas leak alarm
4. Safety standard maintenance
5. Maintaining environment standards in hospitals

3.1.1.4Pin Description

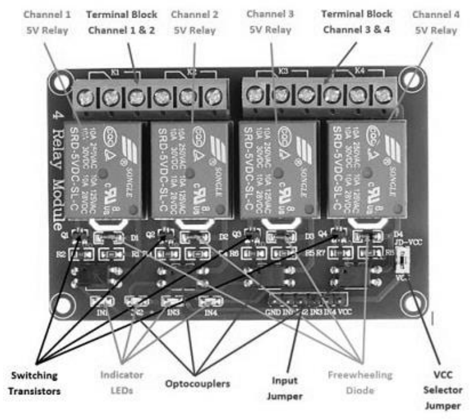
The MQ2 gas sensor features essential pins for seamless integration into electronic systems. The VCC pin connects to the power source, typically 5V, supplying energy to the sensor. The GND pin establishes the ground connection, ensuring a complete circuit and proper sensor functionality. The AO (Analog Output) pin produces an analog signal proportionate to the detected gas concentration, facilitating communication with microcontrollers. The DO (Digital Output) pin provides a digital signal that switches between high and low states based on predetermined gas concentration thresholds. Additionally, the sensor includes a heater element, connected to the power source, which maintains a consistent temperature for the gas-sensitive component, ensuring reliable and accurate gas detection.

Pin Number	Pin Name	Function
1	Vcc (+5v)	This pin powers the module, typically the operating voltage is +5V
2	Ground (GND)	Used to connect the module to system ground
3	Digital Out	You can also use this sensor to get digital output from this pin, by setting a threshold value using the potentiometer
4	Analog Out	This pin outputs 0-5V analog voltage based on the intensity of the gas

Table 3.1.1.4: Pin Description of MQ2 gas sensor

3.1.2.1 4 Channel relay module

A relay module that has four 5V relays with the associated switching, as well as isolating components, is known as a 5V 4-channel relay module. This relay module will make interfacing easy with a sensor or microcontroller. This relay module is a convenient board used mainly for controlling high voltage, high current loads like electrical motors, lamps, AC loads & solenoid valves. This module is mainly designed for interfacing with microcontrollers like PIC, Arduino, etc.



3.1.2.1 : 4 Channel relay module

3.1.2.2 Features & Specifications:

- This module includes four channels.
- This relay module size is L x W x H 75mm X 55mm X 19.3mm.
- Its weight is 61 grams.
- The PCB color of the relay module is Blue.
- This relay module includes four 3.1mm permanent screw holes at every corner of the module.
- It includes one NC contact & one NO contact.
- Its operating voltage ranges from 3.3V to 5V.
- Its trigger current is 5mA.
- The max contact voltage of the relay is 250VAC, 30VDC.
- The Max current of the relay is 10A.
- When the single relay is active then the current is 70mA & 300mA for all four relays.
- It has a high impedance-based controller pin.
- It has a pull-down circuit to avoid malfunction.
- It has an indicator of power supply.
- This relay module is obedient with international security standards, control & load areas isolation channels.
- Alternate relay modules are single-channel, dual-channel, and eight-channel relay modules.

3.1.2.3 Pin Configuration

- **Pin1 (GND):** It is a GND pin of the module.
- **Pin2 (IN1):** It is an i/p to activate the relay.
- **Pin3 (IN2):** It is an i/p to activate relay2.
- **Pin4 (IN3):** It is an i/p to activate relay3.
- **Pin5 (IN4):** It is an i/p to activate relay4.
- **Pin6 (VCC):** It is a power supply pin for the relay module.

- **Pin7 (VCC):** It is a power supply choice jumper.
- **Pin8 (JD-VCC):** It is an alternating power pin of the module.

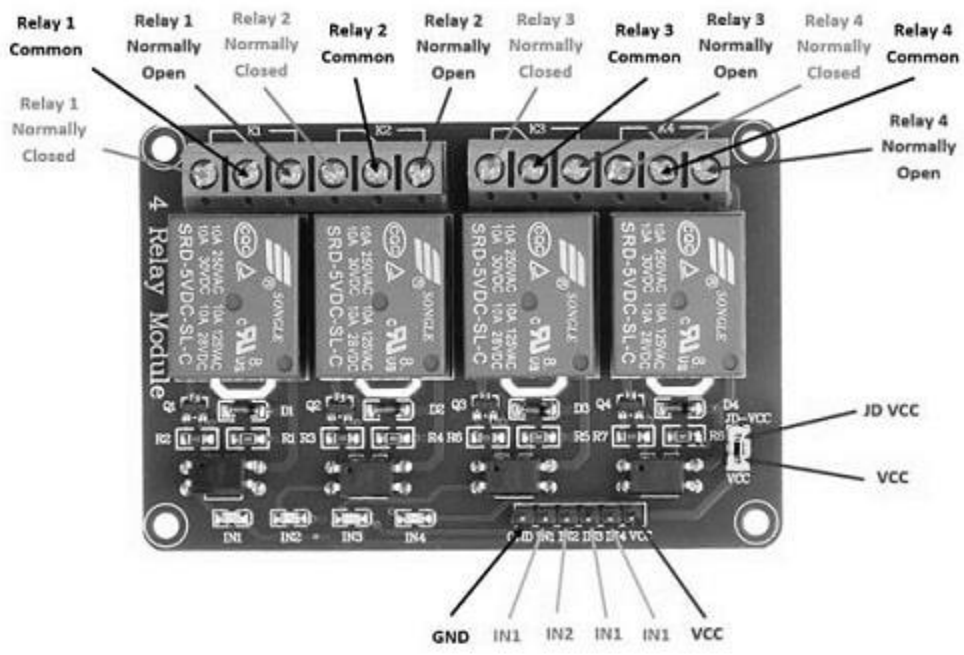


Fig 3.1.2.3: 5V 4 Channel Relay Module Pin Configuration

Output Connections:

Common Connection (COM): The common connection of the relay is the center terminal, used within both NC & NO connections.

Normally Open (NO): The NO connection works like a switch. In this connection, there is no contact between the common connection & NO because it is normally open. However, once we trigger the relay module, it will connect to the COM & provides power to the load, which will activate the light. So the circuit will firstly be in open condition until we activate the condition.

NC (Normally Closed): As compared to the NC connection, NO is quite the opposite connection. So it will stay in contact always with COM even if the module is not activated. Once we activate the relay module, it will open the electrical circuit, thus the connection will be lost.

3.2 NODE MCU ESP32 MICRO-CONTROLLER

The ESP32 module is a highly versatile microcontroller developed by Espressif Systems. It boasts a dual-core Xtensa LX6 processor, providing enhanced processing capabilities for a variety of applications. Equipped with built-in Wi-Fi and Bluetooth support, the ESP32 is ideal for IoT projects, allowing seamless wireless communication. Its extensive set of GPIO pins, including digital and analog options, makes it adaptable for interfacing with sensors and peripherals. Security features, like hardware-accelerated encryption, enhance data protection in IoT applications. With compatibility for programming languages such as Arduino, Micro Python, and the Espressif IoT Development Framework, the ESP32 caters to a broad user base, from hobbyists to professionals.

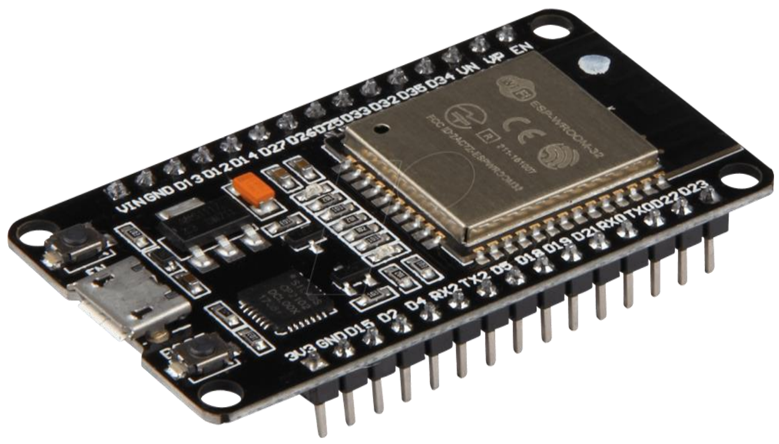


Fig 3.2 (a) ESP32 MICRO CONTROLLER MODULE

In addition to its technical prowess, the ESP32's popularity stems from its compact size, low power consumption, and user-friendly programmability. Its ability to function as a standalone microcontroller or connect to other devices via Wi-Fi and Bluetooth makes it a go-to choice for projects ranging from home automation to industrial applications. The ESP32's robust capabilities, ease of use, and strong community support have solidified its position as a key player in the ever-expanding landscape of microcontrollers and IoT development.

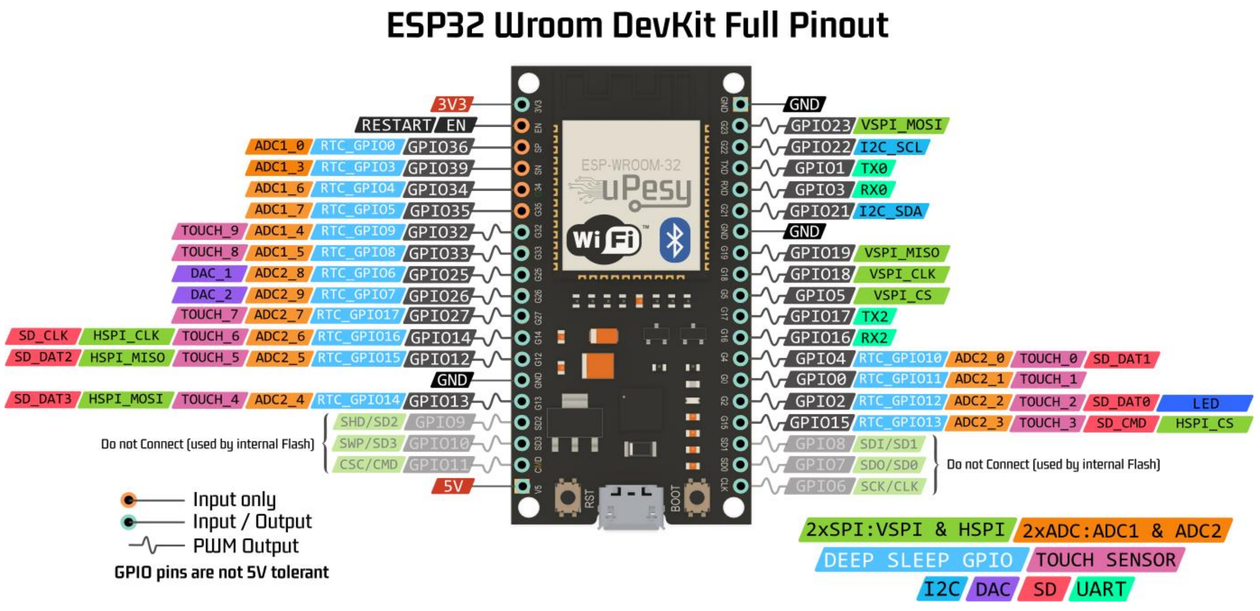

ESP32 Wroom DevKit Full Pinout
3V3
RESTART / EN
ADC1_0 / RTC_GPIO0 / GPIO36
ADC1_3 / RTC_GPIO3 / GPIO39
ADC1_6 / RTC_GPIO4 / GPIO34
ADC1_7 / RTC_GPIO5 / GPIO35
TOUCH_9 / ADC1_4 / RTC_GPIO9 / GPIO32
TOUCH_8 / ADC1_5 / RTC_GPIO8 / GPIO33
DAC_1 / ADC2_8 / RTC_GPIO6 / GPIO25
DAC_2 / ADC2_9 / RTC_GPIO7 / GPIO26
TOUCH_7 / ADC2_7 / RTC_GPIO17 / GPIO27
SD_CLK / HSPI_CLK / TOUCH_6 / ADC2_6 / RTC_GPIO16 / GPIO14
SD_DAT2 / HSPI_MISO / TOUCH_5 / ADC2_5 / RTC_GPIO15 / GPIO12
GND
SD_DAT3 / HSPI_MOSI / TOUCH_4 / ADC2_4 / RTC_GPIO14 / GPIO13
SHD/SD2 / GPIO9
SWP/SD3 / GPIO10
CSC/CMD / GPIO11
Do not Connect (used by internal Flash)
5V
GND
GPIO23 / VSPI_MOSI
GPIO22 / I2C_SCL
GPIO1 / TX0
GPIO3 / RX0
GPIO21 / I2C_SDA
GND
GPIO19 / VSPI_MISO
GPIO18 / VSPI_CLK
GPIO5 / VSPI_CS
GPIO17 / TX2
GPIO16 / RX2
GPIO4 / RTC_GPIO10 / ADC2_0 / TOUCH_0 / SD_DAT1
GPIO0 / RTC_GPIO11 / ADC2_1 / TOUCH_1
GPIO2 / RTC_GPIO12 / ADC2_2 / TOUCH_2 / SD_DAT0 / LED
GPIO15 / RTC_GPIO13 / ADC2_3 / TOUCH_3 / SD_CMD / HSPI_CS
GPIO8 / SDI/SD1
GPIO7 / SDO/SD0
GPIO6 / SCK/CLK
Do not Connect (used by internal Flash)
ESP-WROOM-32
uPesy
Wi Fi
RST
BOOT
Input only
Input / Output
PWM Output
GPIO pins are not 5V tolerant
2xSPI:VSPI & HSPI
2xADC:ADC1 & ADC2
DEEP SLEEP GPIO
TOUCH SENSOR
I2C
DAC
SD
UART

Fig 3.2 (b) PIN OUT OF ESP32 MICRO CONTROLLER MODULE

Applications:

The ESP32 module finds applications in diverse fields, serving as the brain of smart home devices like thermostats and lighting systems. It powers wearable gadgets, enabling fitness trackers and smartwatches to connect seamlessly to other devices. In industrial settings, the ESP32 facilitates automation by controlling machinery and collecting data for monitoring. In agriculture, it aids in creating smart irrigation systems, while its use in educational projects helps students learn about electronics and programming. With its compact size and wireless capabilities, the ESP32 is a versatile choice for projects ranging from simple DIY setups to complex IOT applications.

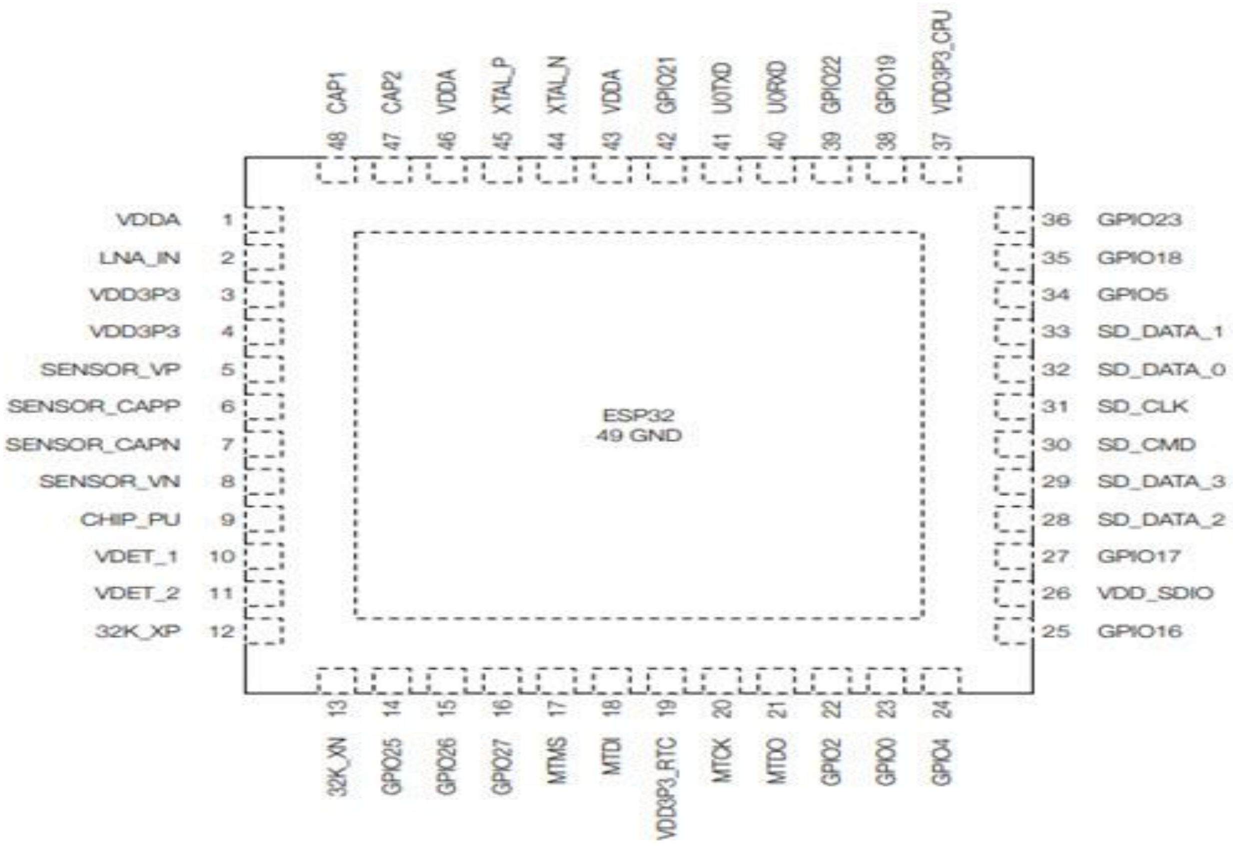
3.3 Features of Micro controller

1. Dual-Core Processor: Dual-core Xtensa LX6 processor for improved processing capabilities.
2. Wireless Connectivity: Built-in Wi-Fi and Bluetooth
3. Peripheral Interfaces: Rich set of digital and analog input/output pins.
   UART, SPI, I2C, PWM
4. 30 × programmable GPIOs
5. Memory: 520 KiB RAM, 448 KiB ROM
6. Ultra-low power (ULP) co-processor
7. IEEE 802.11 standard security features all supported, including WPA, WPA2, WPA3 (depending on version) [5] and WLAN Authentication and Privacy Infrastructure (WAPI)
8. Power management: Individual power domain for RTC
9. I/O pins: 28
10. Timers: Two 32 bit / Four 16-bit timers
11. PWM: 16 Channels
12. UART: Yes
13. 2 x I^2s interfaces

The ESP32 stands out with its dual-core processing power, enabling efficient multitasking and improved overall performance. Its built-in Wi-Fi and Bluetooth capabilities make it a go-to choice for seamless connectivity in Internet of Things (IOT) projects, allowing devices to communicate effortlessly. The microcontroller offers extensive peripheral support, featuring digital and analog pins, along with interfaces like UART, SPI, I2C, and PWM, providing versatility for connecting various sensors and peripherals. Security is a priority, with hardware-accelerated encryption ensuring the safe transmission of data, a critical feature for protecting sensitive information in IOT applications. The ESP32's ease of programmability, supporting popular languages like Arduino and Micro Python, caters to a diverse user base, making it accessible for both beginners and experienced developers. Moreover, its compact size and low power consumption make the ESP32 an ideal choice for a wide range of applications, from wearables to energy-efficient IOT devices.

3.4 PIN DIAGRAM OF Xtensa LX6 processor

The Xtensa LX6 processor in the ESP32 is a dual-core engine designed for efficient and powerful performance. With two processing units, it enables the ESP32 to handle multiple tasks simultaneously, enhancing overall responsiveness. The processor's architecture is optimized for embedded systems and Internet of Things (IoT) applications, providing a balance between performance and energy efficiency. Its dual-core design allows for effective multitasking, making the ESP32 well-suited for a variety of computing needs. The Xtensa LX6 contributes to the ESP32's capability to execute complex instructions, making it a reliable choice for a wide range of projects, from smart home devices to industrial automation.



Description of each pin of esp32 micro-controller module

Pin number	Description	Function
1	VDDA	Analog power supply (2.3 V ~ 3.6 V)
2	LNA_IN	RF input and output
3	VDD3P3	Analog power supply (2.3 V ~ 3.6 V)

4	VDD3P3	Analog power supply (2.3 V ~ 3.6 V)
5	SENSOR_VP	GPIO36, ADC1_CH0, RTC_GPIO0
6	SENSOR_CAPP	GPIO37, ADC1_CH1, RTC_GPIO1
7	SENSOR_CAPN	GPIO38, ADC1_CH2, RTC_GPIO2
8	SENSOR_CN	GPIO39, ADC1_CH3, RTC_GPIO3
9	CHIP_PU	**High**: On; enables the chip **Low**: Off; the chip shuts down
10	VDET_1	GPIO34, ADC1_CH6, RTC_GPIO4
11	VDET_2	GPIO35, ADC1_CH7, RTC_GPIO5
12	32K_XP	GPIO32, ADC1_CH4, RTC_GPIO9, TOUCH9, 32K_XP (32.768 kHz crystal oscillator input)
13	32K_XN	GPIO33, ADC1_CH5, RTC_GPIO8, TOUCH8, 32K_XN (32.768 kHz crystal oscillator output)
14	GPIO25	GPIO25, ADC2_CH8, RTC_GPIO6, DAC_1, EMAC_RXD0
15	GPIO26	GPIO26, ADC2_CH9, RTC_GPIO7, DAC_2, EMAC_RXD1
16	GPIO27	GPIO27, ADC2_CH7, RTC_GPIO17, TOUCH7, EMAC_RX_DV
17	MTMS	GPIO14, ADC2_CH6, RTC_GPIO16, TOUCH6, EMAC_TXD2, HSPICLK, HS2_CLK, SD_CLK
18	MTDI	GPIO12, ADC2_CH5, RTC_GPIO15, TOUCH5,

		EMAC_TXD3, HSPIQ, HS2_DATA2, SD_DATA2
19	VDD3P3_RTC	Input power supply for RTC IO (2.3 V ~ 3.6 V)
20	MTCK	GPIO13, ADC2_CH4, RTC_GPIO14, TOUCH4, EMAC_RX_ER, HSPID, HS2_DATA3, SD_DATA3
21	MTDO	GPIO15, ADC2_CH3, RTC_GPIO13, TOUCH3, EMAC_RXD3, HSPICS0, HS2_CMD, SD_CMD
22	GPIO2	ADC2_CH2, RTC_GPIO12, TOUCH2, HSPIWP, HS2_DATA0, SD_DATA0
23	GPIO0	ADC2_CH1, RTC_GPIO11, TOUCH1, EMAC_TX_CLK, CLK_OUT1,
24	GPIO4	GPIO4, ADC2_CH0, RTC_GPIO10, TOUCH0, EMAC_TX_ER, HSPIHD, HS2_DATA1, SD_DATA1
25	GPIO16	GPIO16, HS1_DATA4, U2RXD,
26	VDD_SDIO	Output power supply: 1.8 V or the same voltage as VDD3P3_RTC
27	GPIO17	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
28	SD_DATA_2	GPIO9, HS1_DATA2, U1RXD, SD_DATA2, SPIHD
29	SD_DATA_3	GPIO10, HS1_DATA3, U1TXD, SD_DATA3, SPIWP
30	SD_CMD	GPIO11, HS1_CMD, U1RTS, SD_CMD, SPICS0
31	SD_CLK	GPIO6, HS1_CLK, U1CTS, SD_CLK, SPICLK

32	SD_DATA_0	GPIO7, HS1_DATA0, U2RTS, SD_DATA0, SPIQ
33	SD_DATA_1	GPIO8, HS1_DATA1, U2CTS, SD_DATA1, SPID
34	GPIO5	GPIO5, HS1_DATA6, VSPICS0, EMAC_RX_CLK
35	GPIO18	GPIO18, HS1_DATA7, VSPICLK
36	GPIO23	GPIO23, HS1_STROBE, VSPID
37	VDD3P3_CPU	Input power supply for CPU IO (1.8 V ~ 3.6 V)
38	GPIO19	GPIO19, U0CTS, VSPIQ
39	GPIO22	GPIO22, U0RTS, VSPIWP, EMAC_TXD1
40	U0RXD	GPIO3, U0RXD, CLK_OUT2
41	U0TXD	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
42	GPIO21	GPIO21, VSPIHD, EMAC_TX_EN
43	VDDA	Analog power supply (2.3 V ~ 3.6 V)
44	XTAL_N	External crystal output
45	XTAL_P	External crystal input
46	VDDA	Analog power supply (2.3 V ~ 3.6 V)
47	CAP2	Connects to a 3.3 nF (10%) capacitor and 20 k Ω resistor in parallel to CAP1
48	CAP1	Connects to a 10 nF series capacitor to ground

Table 3.4: Each Pin Description

3.5 BASIC TERMINOLOGIES IN ESP32

3.5.1 Analog to Digital Converter (ADC)

1. **ADC Channels:** The ESP32 has multiple ADC channels, typically labeled as ADC1_CH0, ADC1_CH1, ..., ADC2_CH0, ADC2_CH1, etc. Each channel corresponds to a specific pin on the ESP32 that can be used for analog input.
2. **Voltage Range:** The ADC in the ESP32 typically operates in the range of 0 to 3.3 volts. Make sure that the analog signal you're measuring falls within this range to avoid damaging the ADC.
3. **Resolution:** The ADC in the ESP32 has a configurable resolution, usually ranging from 9 to 12 bits. Higher resolution provides more precise measurements but requires more processing time.
4. **Reference Voltage:** The ESP32 allows you to set the reference voltage for the ADC. The default reference voltage is usually the supply voltage (VDD), but you can also use an external reference voltage for more accurate measurements.
5. **Functionality:** The ESP32 ADC can be used in both single-ended and differential modes. In single-ended mode, the voltage is measured with respect to ground, while in differential mode, the voltage difference between two channels is measured.
6. **Reading ADC Values:** To read analog values using the ADC in the ESP32, you typically use functions provided by the ESP32 SDK or the Arduino IDE. In Arduino, you may use functions like **analogRead( )** to obtain the ADC values.

3.5.2 Pulse Width Modulation (PWM)

1. **PWM Channels:** The ESP32 provides multiple PWM channels, each associated with a specific GPIO pin. The number of PWM channels available depends on the specific ESP32 module or development board you are using.
2. **Frequency and Duty Cycle:** PWM allows you to control the frequency and duty cycle of the output signal. The frequency determines how fast the PWM signal oscillates, while the duty cycle represents the percentage of time the signal is high compared to the total period. Higher duty cycles correspond to higher average voltages.
3. **Pins with PWM Support:** Not all GPIO pins on the ESP32 support PWM. You need to refer to the documentation or pinout diagram of your specific ESP32 module to identify which pins are PWM-enabled.
4. **Using PWM in Arduino IDE:** If you're using the Arduino IDE to program the ESP32, you can use the **analogWrite()** function to generate PWM signals. The function takes a pin number and a duty cycle value (ranging from 0 to 255).

3.6 Language Reference

The ESP32, a tiny computer, can be programmed using two main languages: C and C++. If you're using the official Espressif IoT Development Framework (ESP-IDF), you'll mainly work with C. There's a helpful documentation on the Espressif website that explains how to use different functions and features for ESP32. On the other hand, if you're using the Arduino framework, you'll write code in a simpler version of C++ that is more beginner-friendly. The Arduino framework hides some of the complicated details, making it easier for people who are just starting. You can find information about ESP32 functions and libraries in the Arduino reference. Whether you choose ESP-IDF or Arduino depends on your project needs and how comfortable you are with each approach - whether you want more control (ESP-IDF) or an easier way to get started (Arduino).

3.7 SOFTWARE REQUIREMENTS

3.7.1 Embedded C

Embedded C programming for the ESP32 involves using the C programming language to develop software for the microcontroller. The ESP32 is a versatile microcontroller with integrated Wi-Fi and Bluetooth capabilities, and it's commonly programmed using the Espressif IOT Development Framework (ESP-IDF). Embedded C, in the context of ESP32, involves writing code that directly interacts with the hardware features of the microcontroller, such as GPIO pins, peripherals (like ADC and PWM), and communication interfaces (Wi-Fi and Bluetooth). The ESP-IDF provides a set of APIs, libraries, and documentation that allow developers to harness the full potential of the ESP32. The code written in Embedded C for ESP32 is typically more low-level and hardware-centric compared to using higher-level frameworks like Arduino, providing greater control and optimization possibilities. Understanding the ESP-IDF documentation and APIs is crucial for efficiently utilizing the ESP32's features in Embedded C programming.

3.7.2 Different between C and Embedded C

C and Embedded C are essentially the same language, but their usage contexts distinguish them. C is a versatile, general-purpose programming language employed in diverse applications, from desktop software to server development. On the other hand, Embedded C is a subset of C crafted specifically for programming embedded systems, such as microcontrollers and IOT devices, which operate under resource constraints. Embedded C focuses on efficiency in managing limited memory and system resources, often providing specialized libraries and features for low-level programming and hardware interfacing. While C assumes sample resources, Embedded C is tailored to optimize code for size and performance in the embedded system environment, requiring specific tools and compilers provided by microcontroller manufacturers for development.

3.7.3 HTML

In your project, HTML (Hypertext Markup Language) plays a crucial role in creating the structure and layout of the web interface. HTML is like the blueprint of a webpage, defining elements such as titles, paragraphs, and links. In simple terms, it helps organize and present information on the monitoring system's display. For example, it specifies where to show temperature, humidity, and other sensor data. Additionally, by embedding HTML code in your Arduino sketch, you're able to dynamically update and showcase real-time information, making it accessible and visually appealing for users monitoring the environmental conditions through a web browser.

3.7.4 CSS

In your project, CSS (Cascading Style Sheets) is employed to enhance the visual presentation and appearance of the web interface created with HTML. CSS acts like a stylist, determining the colors, fonts, and layout of the displayed elements. It allows you to make the monitoring system interface not only informative but also visually engaging and user-friendly. For instance, CSS is used to define the background colors, text styles, and animations, giving a polished look to the temperature, humidity, and other sensor readings. By incorporating CSS into your project, you ensure a pleasant and well-designed user experience, making the monitoring system visually appealing and easy to navigate.

3.7.5 JAVA SCRIPT

In your project, JavaScript is utilized to dynamically update and refresh sensor data on the web interface without requiring the user to manually reload the page. Acting as the interactive engine, JavaScript constantly communicates with the server, fetching and displaying real-time information such as temperature, humidity, and other sensor readings. This enhances the user experience by providing timely and automatic updates, ensuring that the displayed data is always current. JavaScript's role is like a behind-the-scenes coordinator, making the monitoring system responsive and efficient in delivering live sensor information to users as it becomes available.

3.8 ARDUINO IDE

Fig 3.8: Arduino IDE Logo

The Arduino IDE (Integrated Development Environment) is a user-friendly software tool that helps people create and program projects using Arduino microcontrollers. It serves as a virtual workspace where you write, edit, and upload code to control electronic components. Think of it like a digital workshop where you can easily design, experiment, and bring your ideas to life, even if you're not an expert in programming. The IDE simplifies the process of coding for Arduino, making it accessible for beginners and experienced makers alike, providing a platform to turn creative concepts into functioning electronic projects.

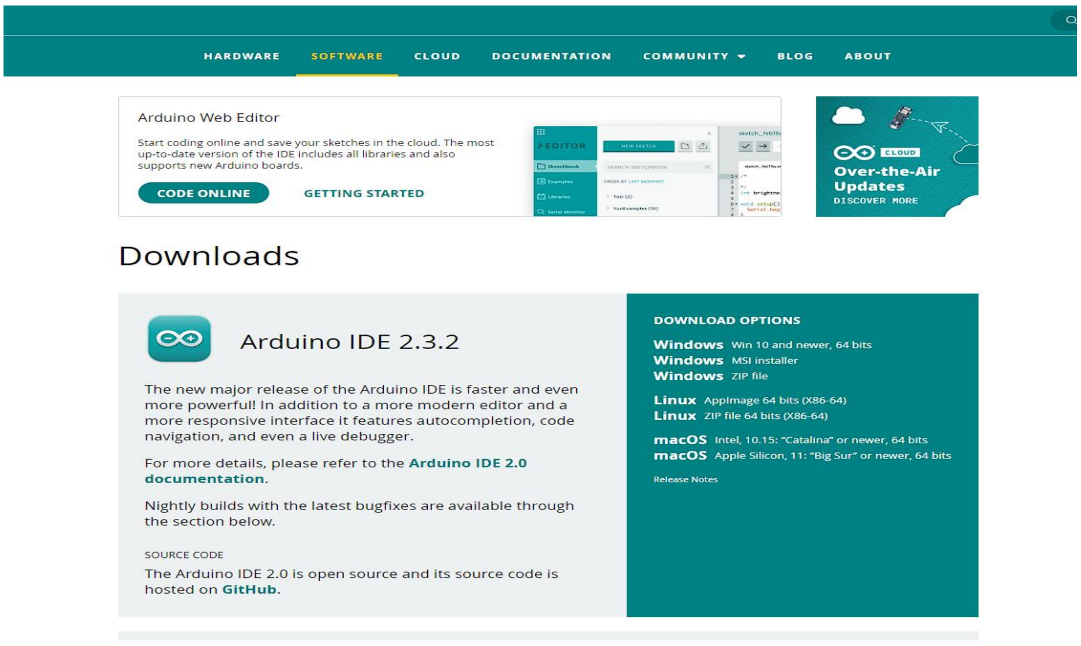
3.8.1 UPLOADING OF SOFTWARE TO ESP32 MICRO-CONTROLLER

Uploading software to an ESP32 board can be accomplished through the Arduino IDE or the Espressif IoT Development Framework (ESP-IDF). In the Arduino IDE, after installing the software and ESP32 board support, select the appropriate board, connect the ESP32 via USB, choose the correct COM port, and click "Upload" to transfer your code. Alternatively, when using the ESP-IDF, install and configure the framework, set up environment variables, navigate to your project directory, configure project settings, and finally build and flash the code to the ESP32 using the command line. Monitoring the serial output can be done optionally to view debug information. Specific details may vary, so consulting the respective documentation for the chosen development environment is recommended for precise instructions.

3.9 Steps to Upload Code to ESP32 using Arduino IDE

3.9.1 <u>Install Arduino IDE</u>

- Visit the official Arduino website at https://www.arduino.cc/en/software. Click on the "Software" tab, and then select the appropriate version for your operating system (Windows, macOS, or Linux).



Arduino Web Editor

Start coding online and save your sketches in the cloud. The most up-to-date version of the IDE includes all libraries and also supports new Arduino boards.

Downloads

Arduino IDE 2.3.2

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

DOWNLOAD OPTIONS

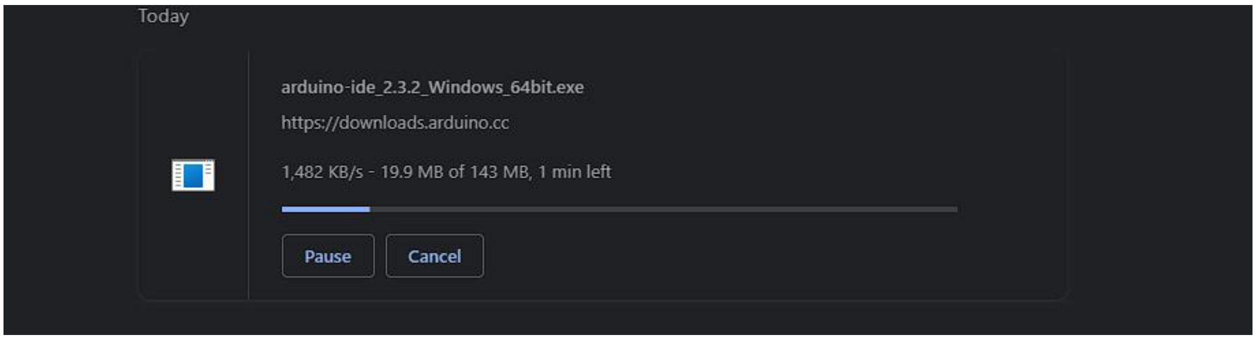
Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

Release Notes

- Click on Windows win10 and newer,64bit download
- And click just download and it will download, press ctrl+j for download conformation


Today

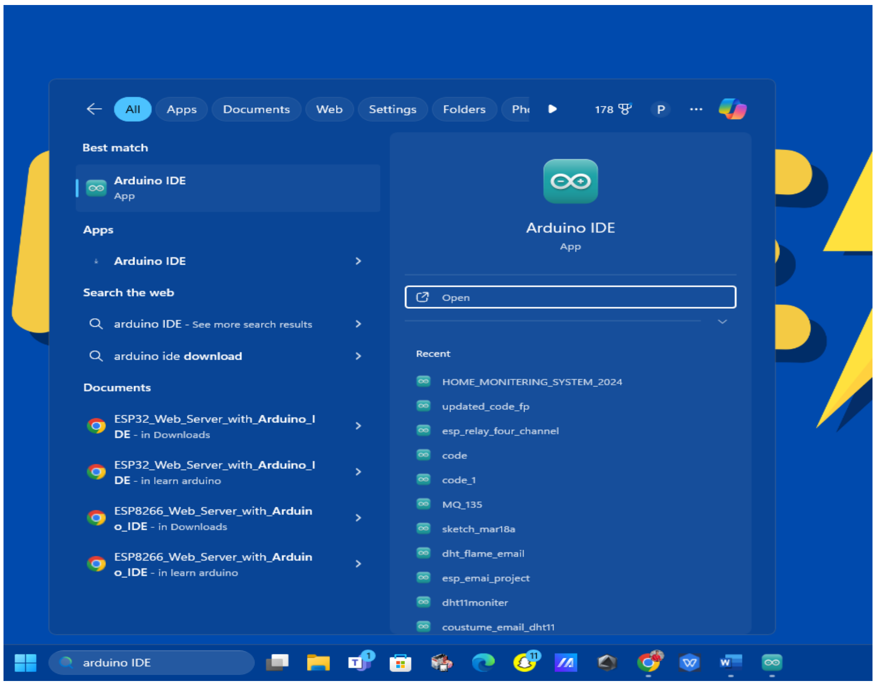
arduino-ide_2.3.2_Windows_64bit.exe

https://downloads.arduino.cc

1,482 KB/s - 19.9 MB of 143 MB, 1 min left

3.9.2 <u>Run Installer</u>

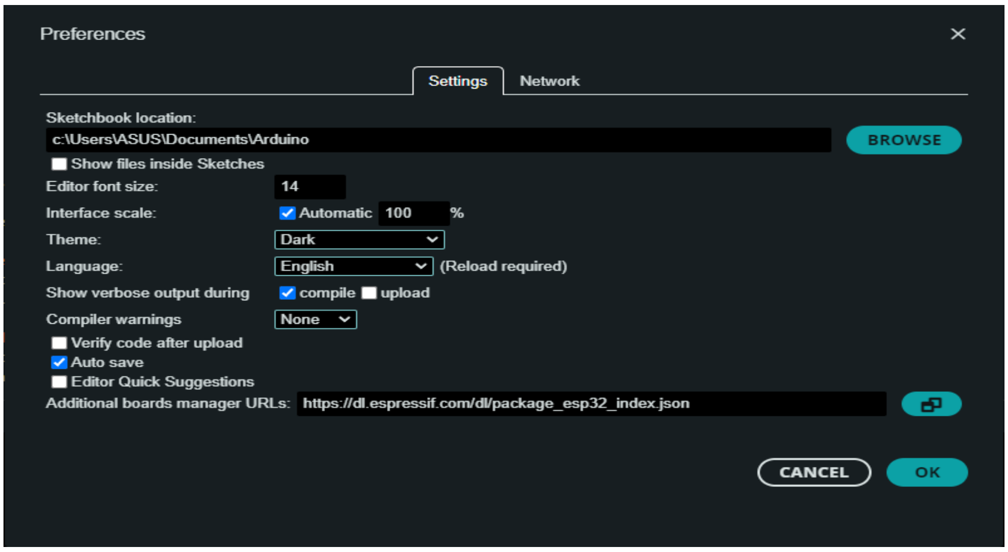
Once the download is complete, run the installer file that you obtained in the previous step.



3.9.3 Install ESP32 Board Support

- Open the Arduino IDE, go to "File" > "Preferences," and add the ESP32 board support URL to the Additional Boards Manager URLs. The URL is: <u>https://dl.espressif.com/dl/package_esp32_index.json</u>

- Then, go to "Tools" > "Board" > "Boards Manager," search for "esp32," and install the ESP32 board support.

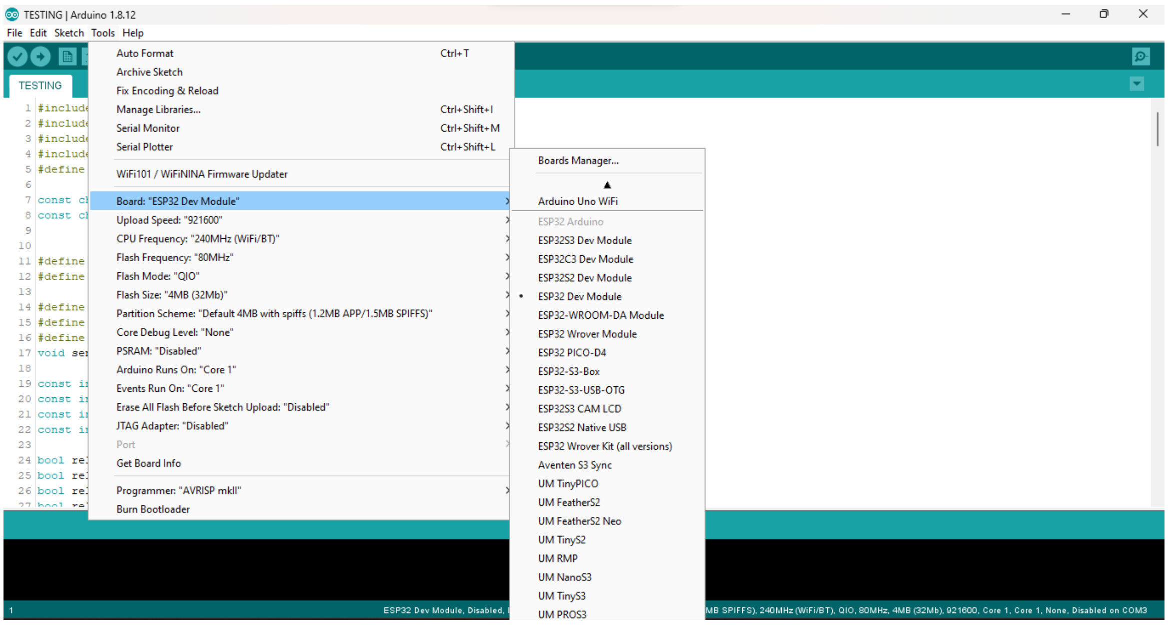

Preferences

Settings | Network

Sketchbook location:
c:\Users\ASUS\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Dark
Language: English (Reload required)
Show verbose output during ☑ compile ☐ upload
Compiler warnings None
☐ Verify code after upload
☑ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://dl.espressif.com/dl/package_esp32_index.json

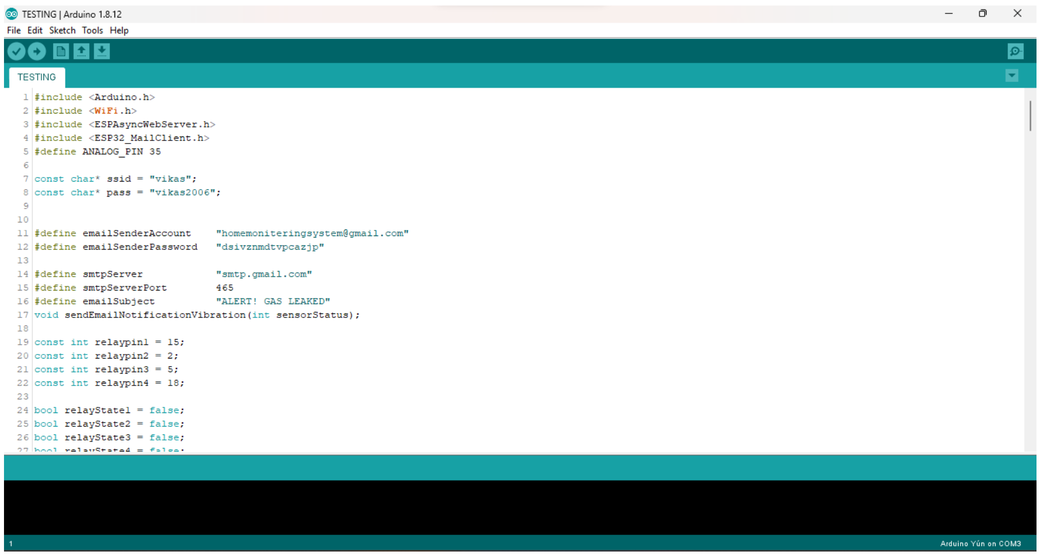
3.9.4 Select ESP32 Board

- In the Arduino IDE, go to "Tools" > "Board" and select your specific ESP32 board from the list. Choose the appropriate variant and upload speed.
- After following these steps, you should have a fully installed Arduino IDE on your computer, ready to be used for programming Arduino boards, including the ESP32.



3.9.5 WRITE CODE

Write your Arduino sketch or open an existing one. Ensure that your code is compatible with the ESP32 and contains the necessary configurations.

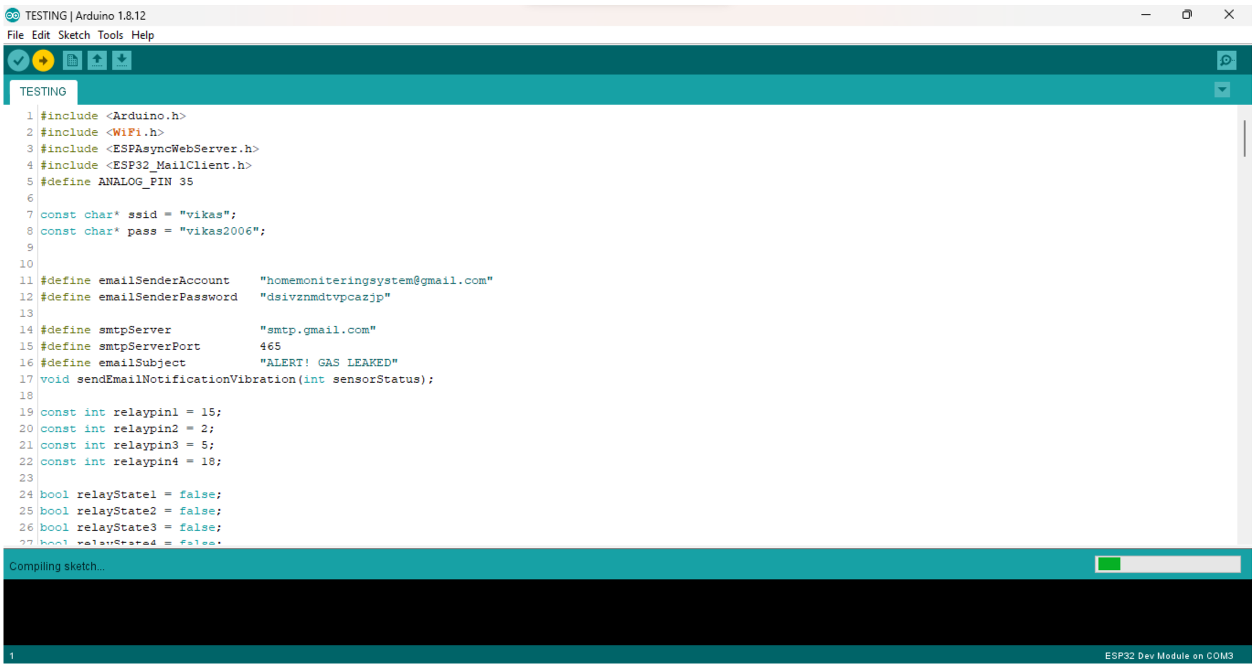

TESTING | Arduino 1.8.12

```
#include <Arduino.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESP32_MailClient.h>
#define ANALOG_PIN 35

const char* ssid = "vikas";
const char* pass = "vikas2006";


#define emailSenderAccount    "homemoniteringsystem@gmail.com"
#define emailSenderPassword   "dsivznmdtvpcazjp"

#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ALERT! GAS LEAKED"
void sendEmailNotificationVibration(int sensorStatus);

const int relaypin1 = 15;
const int relaypin2 = 2;
const int relaypin3 = 5;
const int relaypin4 = 18;

bool relayState1 = false;
bool relayState2 = false;
bool relayState3 = false;
```

1 Arduino Yún on COM3

3.9.6 Upload Code

Click the "Upload" button (right arrow icon) in the Arduino IDE. This will compile your code and upload it to the ESP32. The status will be displayed in the bottom console, and any errors will be shown if there are issues during the upload process.


TESTING | Arduino 1.8.12

```
#include <Arduino.h>
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <ESP32_MailClient.h>
#define ANALOG_PIN 35

const char* ssid = "vikas";
const char* pass = "vikas2006";


#define emailSenderAccount    "homemoniteringsystem@gmail.com"
#define emailSenderPassword   "dsivznmdtvpcazjp"

#define smtpServer            "smtp.gmail.com"
#define smtpServerPort        465
#define emailSubject          "ALERT! GAS LEAKED"
void sendEmailNotificationVibration(int sensorStatus);

const int relaypin1 = 15;
const int relaypin2 = 2;
const int relaypin3 = 5;
const int relaypin4 = 18;

bool relayState1 = false;
bool relayState2 = false;
bool relayState3 = false;
```

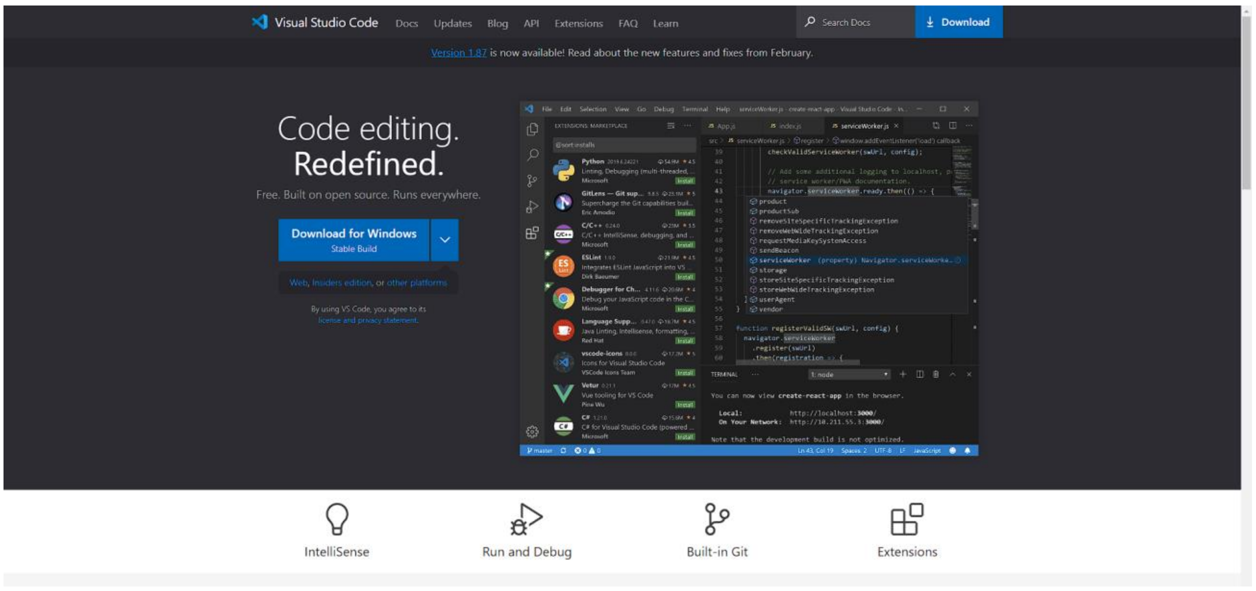
Compiling sketch...

1 ESP32 Dev Module on COM3

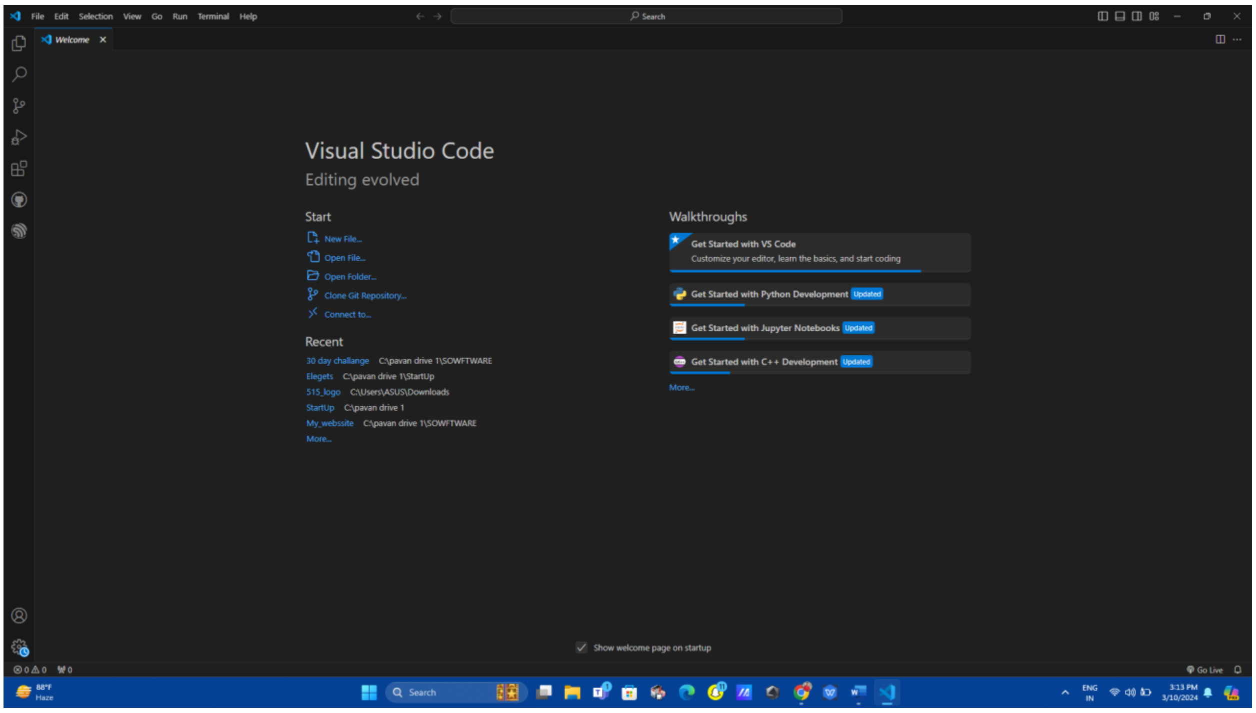
3.10 USER INTERFACE USING HTML AND CSS

3.10.1 Install Visual Studio Code

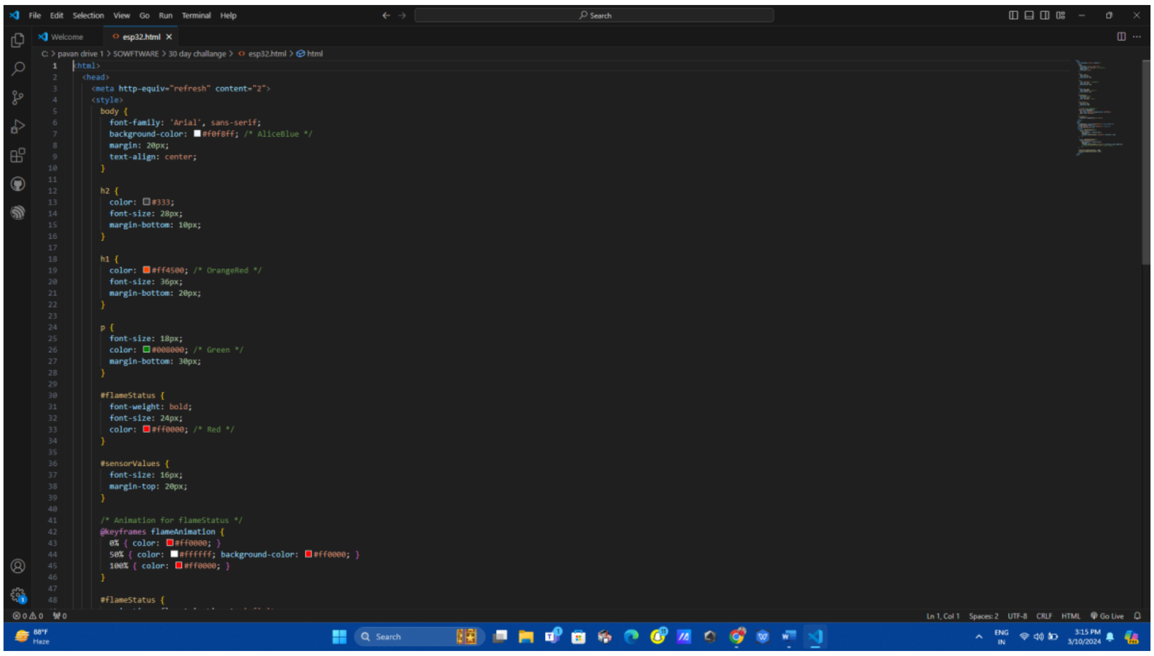
Download and install Visual Studio Code from the official website: Visual Studio Code


Visual Studio Code
Docs
Updates
Blog
API
Extensions
FAQ
Learn
Search Docs
Download
Version 1.87 is now available! Read about the new features and fixes from February.
Code editing.
Redefined.
Free. Built on open source. Runs everywhere.
Download for Windows
Stable Build
Web, Insiders edition, or other platforms
By using VS Code, you agree to its license and privacy statement.
IntelliSense
Run and Debug
Built-in Git
Extensions

Open VS Code and go to the Extensions view by clicking on the Extensions icon in the Activity Bar on the side of the window or press **Ctrl+Shift+X**.

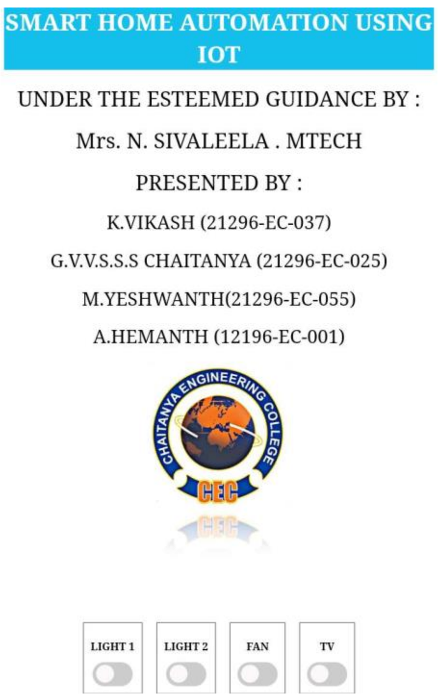

File
Edit
Selection
View
Go
Run
Terminal
Help
Search
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
30 day challange C:\pavan drive 1\SOWFTWARE
Elegets C:\pavan drive 1\StartUp
515_logo C:\Users\ASUS\Downloads
StartUp C:\pavan drive 1
My_website C:\pavan drive 1\SOWFTWARE
More...
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Get Started with Python Development
Updated
Get Started with Jupyter Notebooks
Updated
Get Started with C++ Development
Updated
More...
Show welcome page on startup

- Click on "New Project" and select your board (ESP32). Follow the prompts to set up a new project.
- Open your HTML and CSS files in the VS Code editor and start coding your user interface. You can use standard HTML and CSS syntax.



```
<html>
  <head>
    <meta http-equiv="refresh" content="2">
    <style>
      body {
        font-family: 'Arial', sans-serif;
        background-color: #f0f8ff; /* AliceBlue */
        margin: 20px;
        text-align: center;
      }

      h2 {
        color: #333;
        font-size: 28px;
        margin-bottom: 10px;
      }

      h1 {
        color: #ff4500; /* OrangeRed */
        font-size: 36px;
        margin-bottom: 20px;
      }

      p {
        font-size: 18px;
        color: #008000; /* Green */
        margin-bottom: 30px;
      }

      #flameStatus {
        font-weight: bold;
        font-size: 24px;
        color: #ff0000; /* Red */
      }

      #sensorValues {
        font-size: 16px;
        margin-top: 20px;
      }

      /* Animation for flameStatus */
      @keyframes flameAnimation {
        0% { color: #ff0000; }
        50% { color: #ffffff; background-color: #ff0000; }
        100% { color: #ff0000; }
      }

      #flameStatus {
```

- Check preview by clicking run / Alt+B



SMART HOME AUTOMATION USING IOT

UNDER THE ESTEEMED GUIDANCE BY :

Mrs. N. SIVALEELA . MTECH

PRESENTED BY :

K.VIKASH (21296-EC-037)

G.V.V.S.S.S CHAITANYA (21296-EC-025)

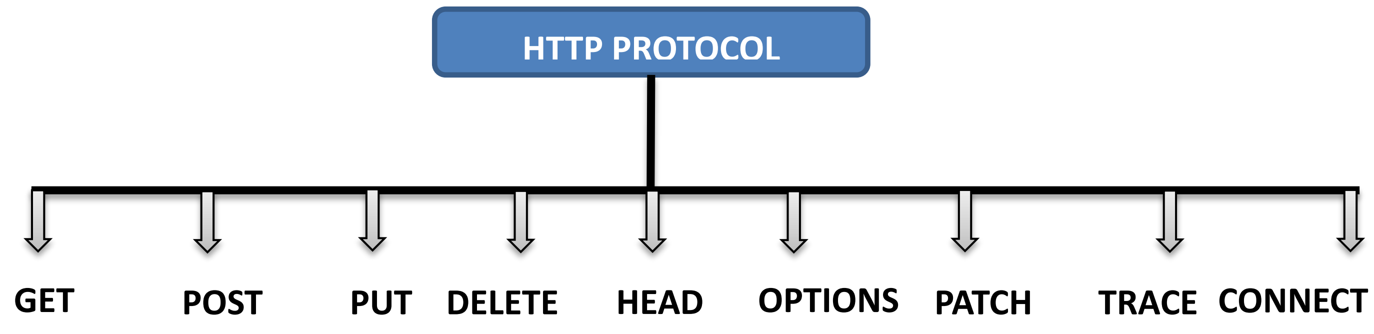
M.YESHWANTH(21296-EC-055)

A.HEMANTH (12196-EC-001)

- Upload code to esp32 module and connect the sensor and appliances as per circuit and control the appliances by using switches in the web interface.

3.10   HTTP PROTOCOL

In your project, the HTTP (Hypertext Transfer Protocol) protocol serves as the communication method between the ESP32-based automation and the web browser. It's like a set of rules that allows the devices to exchange information. When you access the automation through a web browser, the browser uses HTTP to request data from the ESP32. The ESP32 then responds, providing the requested information, such as sensor detects. It's similar to a conversation where one side asks questions (HTTP requests), and the other side provides answers (HTTP responses), enabling seamless and standardized communication between the automation and the user's browser.



TYPES OF HTTP PROTOCOLS

3.10.1 Explanation of commonly used HTTP Protocol

1) GET - Requests data from a specified resource. It should have no side effects on the server.

2) POST - Submits data to be processed to a specified resource. It may lead to a change in state on the server.

3) PUT - Updates a resource or creates a new resource if it does not exist at the specified URI.

4) DELETE - Deletes the specified resource.

5) HEAD - Requests the headers of the specified resource without the actual data. It's often used to check if a resource has changed.

6) OPTIONS - Requests information about the communication options available for the target resource.

7) PATCH -Applies partial modifications to a resource.

8) TRACE - Performs a message loop-back test along the path to the target resource.

9) CONNECT- Establishes a network connection to the target resource.

EXAMPLE:

```
request->send(200, "text/html", html);

            });
```

```
 server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest *request) {
   if (request->hasParam("relay")) {
     int relay = request->getParam("relay")->value().toInt();
     if (relay == 1) {
       relayState1 = !relayState1;
       digitalWrite(relaypin1, relayState1 ? LOW : HIGH);
       request->send(200, "text/plain", relayState1 ? "Relay 1 Turned ON" : "Relay 1
turned OFF");
     } else if (relay == 2) {
       relayState2 = !relayState2;
       digitalWrite(relaypin2, relayState2 ? LOW : HIGH);
       request->send(200, "text/plain", relayState2 ? "Relay 2 turned OFF" : "Relay 2
turned ON");
     } else if (relay == 3) {
       relayState3 = !relayState3;
       digitalWrite(relaypin3, relayState3 ? LOW : HIGH);
       request->send(200, "text/plain", relayState3 ? "Relay 3 turned OFF" : "Relay 3
turned ON");
     } else if (relay == 4) {
       relayState4 = !relayState4;
       digitalWrite(relaypin4, relayState4 ? LOW : HIGH);
       request->send(200, "text/plain", relayState4 ? "Relay 4 turned OFF" : "Relay 4
turned ON");
     } else {
       request->send(400, "text/plain", "Invalid request");
     }
   }
 });
```

3.11 SMTP PROTOCOL

SMTP, or Simple Mail Transfer Protocol, is a widely used network protocol for the transmission of electronic mail (email) between computers. It's the protocol responsible for sending emails from a client to a server or between servers. Here's a brief explanation of how SMTP works:

1. **Communication Flow:**
   - **Client-Server Interaction:** SMTP operates on a client-server model. An email client (like Outlook or Thunderbird) communicates with an SMTP server to send emails.
   - **Server-to-Server Communication:** When an email is sent from one domain to another, the SMTP server of the sender communicates with the SMTP server of the recipient.
2. **Port and Security:**
   - **Port Number:** SMTP typically uses port 25 for unencrypted communication and port 587 for encrypted communication (STARTTLS). Port 465 is also sometimes used for secure SMTP over SSL/TLS.
   - **Security:** In modern email systems, encryption (SSL/TLS) is often used to secure the communication between the email client and the server, preventing eavesdropping.
3. **Commands and Responses:**
   - **Commands:** The client issues commands to the server to send an email, specify recipients, etc. Common commands include EHLO (identify the client), MAIL FROM (specify the sender), RCPT TO (specify the recipient), DATA (start message transmission), etc.
   - **Responses:** The server responds to each command, indicating success or providing error information.
4. **Message Format:**
   - **Header and Body:** The email message is divided into two parts: the header and the body. The header contains metadata like sender, recipient, subject, etc., while the body contains the actual message.
   - **Termination:** The end of the message is indicated by a special sequence (CRLF.CRLF).
5. **Relaying and Forwarding:**
   - **Relaying:** SMTP servers can relay messages to other servers if the recipient is not on the same server. This facilitates email communication across different domains.

- **Forwarding:** Servers may also forward emails to the next server in the destination domain until the email reaches its final destination.

6. **Authentication:**

- **Authentication Methods:** To prevent unauthorized access, SMTP servers often require authentication. Common authentication methods include username/password, and more secure methods like OAuth.

CHAPTER-4

INTERFACING AND TESTING

4.1 MICRO-CONTROLLER – MQ2 SENSOR INTERFACING

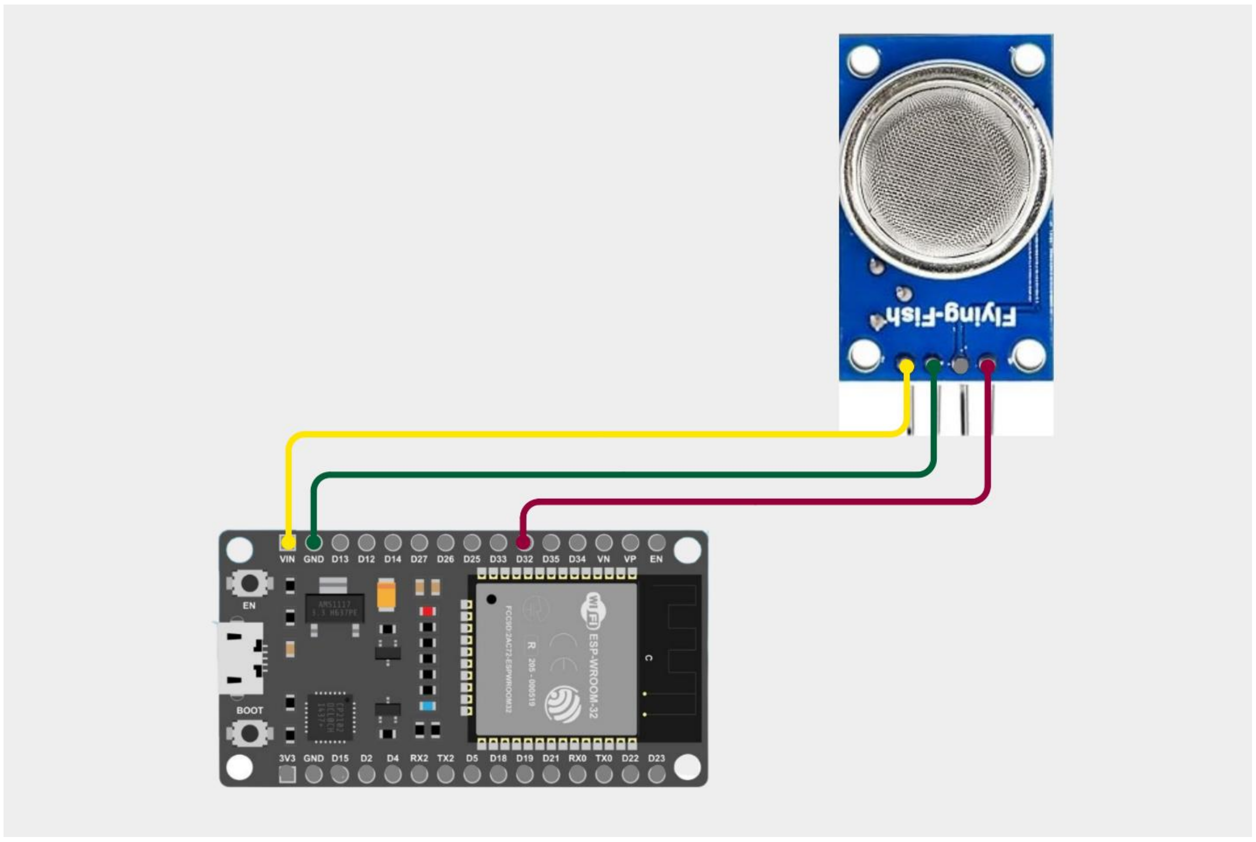

Flying-Fish
VIN GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
EN
BOOT
ESP-WROOM-32
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23

Fig 4.1: MQ2 sensor interfacing with ESP32

The ESP32 microcontroller seamlessly interfaces with the MQ2 gas sensor, utilizing GPIO pins for communication. The MQ2 sensor detects various gases such as LPG, propane, methane, alcohol, and smoke. Its detection range covers concentrations from 300 to 10,000 ppm for LPG, 200 to 10,000 ppm for propane, and 100 to 10,000 ppm for methane, with sensitivity adjustments possible through on board potentiometers. Employing appropriate software libraries, like those available for Arduino, the ESP32 efficiently captures gas concentration data from the MQ2 sensor. This data, vital for ensuring indoor air quality and detecting potential hazards, can be transmitted to a Raspberry Pi for further processing using communication protocols such as MQTT or HTTP. This integration empowers IOT applications to monitor and respond to gas concentrations in real-time, enhancing safety and environmental awareness.

4.2 MICRO-CONTROLLER – 4 CHANNEL RELAY MODULE INTERFACING

Interfacing a 4-channel relay module with a microcontroller involves establishing electrical connections between the relay module and the microcontroller, enabling remote control of high-power devices through low-power signals. Typically, each relay in the module has a control pin that connects to a GPIO pin on the microcontroller. Through software control, the microcontroller sends signals to these pins to activate or deactivate the relays, which, in turn, switch the connected devices on or off. Care must be taken to ensure proper power supply to the relay module and appropriate handling of electrical connections to prevent damage to the components. This interface allows for the automation and remote operation of various appliances, making it useful in applications ranging from home automation to industrial control systems.

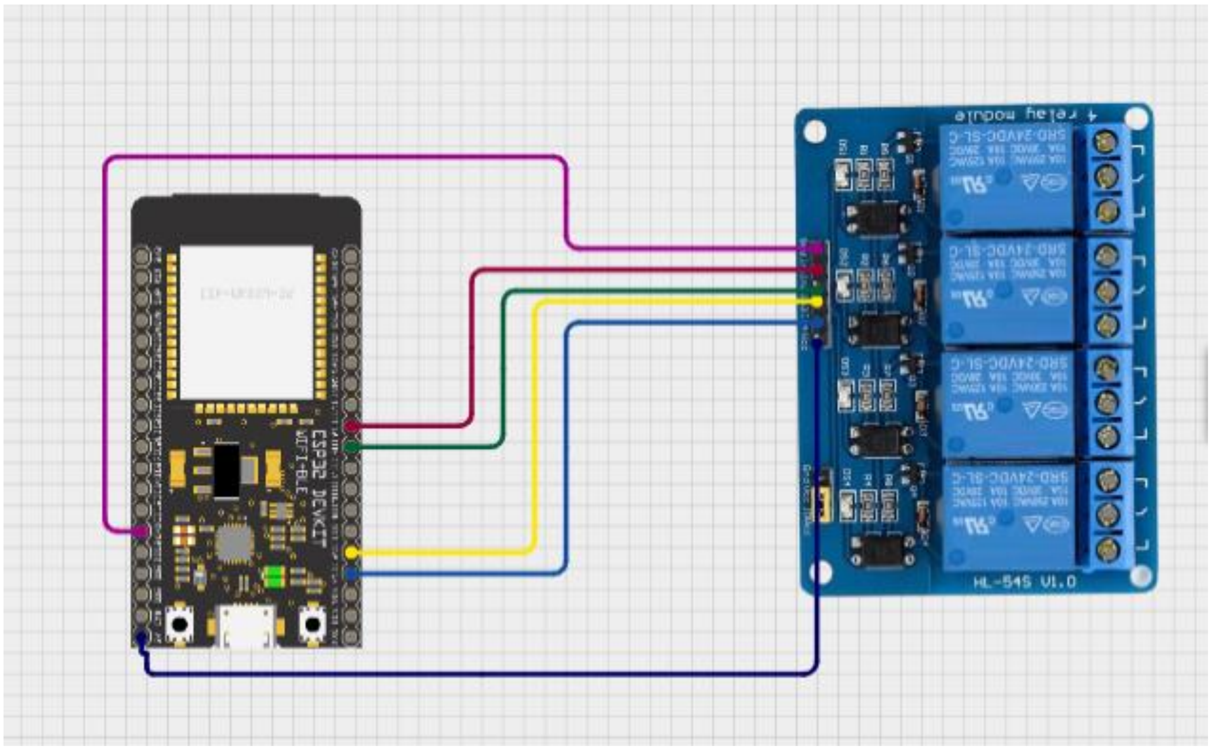


Fig 4.2: 4Channel Relay interfacing with ESP32

4.3 4 CHANNEL RELAY MODULE INTERFACING WITH APPLIANCES

Interfacing a 4-channel relay module with appliances involves connecting the relay module to both the appliances and a microcontroller or a controlling device. Each relay on the module serves as a switch for an individual appliance. By controlling the relay module through the microcontroller, you can effectively switch the connected appliances on or off remotely or based on predefined conditions. For instance, in a smart home application, you could use sensors or timers to trigger the relays, allowing you to automate tasks such as turning lights, fans, or other devices on and off. Proper attention to electrical safety and understanding the power requirements of the appliances is crucial to ensure safe and efficient operation. With the appropriate setup and programming, interfacing a 4-channel relay module with appliances offers a versatile solution for automation and control in various applications.

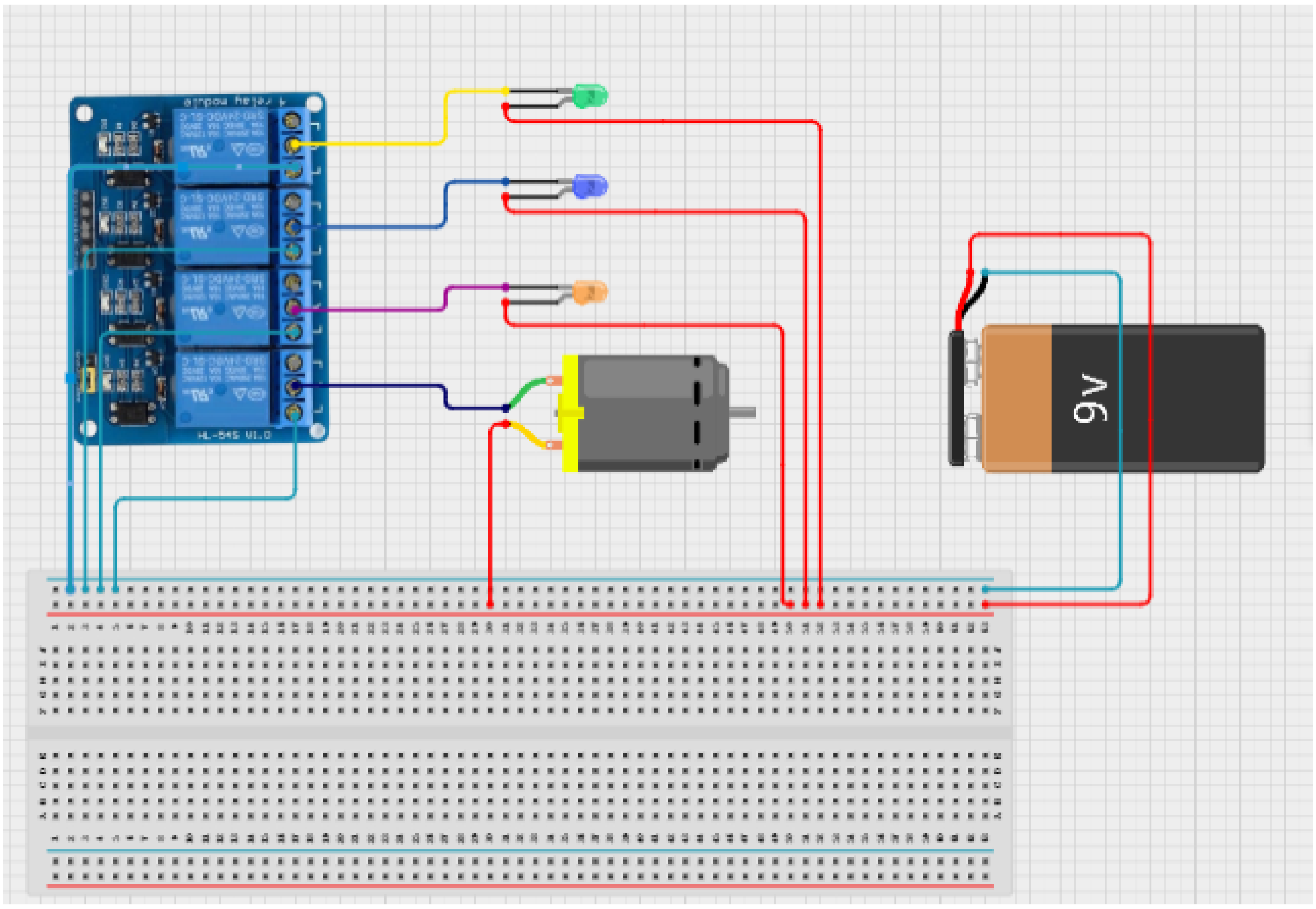


Fig 4.3: 4Channel Relay interfacing with Appliances

CHAPTER-5

INITIALIZATION AND WORKING OF THE MODULE

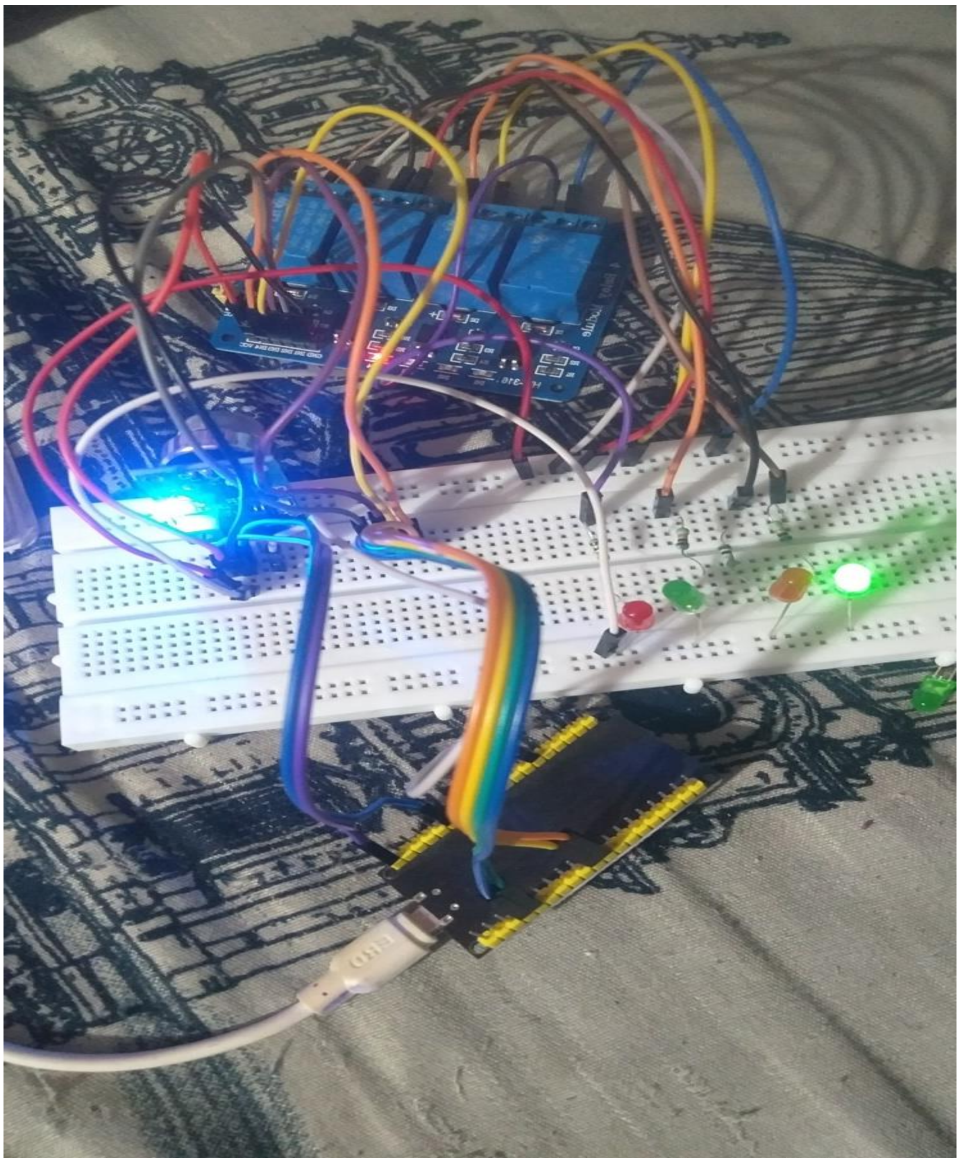


Fig 5.1: INITIALIZATION OF THE MODULE

CHAPTER-6

ADVANTAGES AND APPLICATIONS

6.1 ADVANTAGES

1. Convenience
2. Energy Efficiency
3. Security & Safety
4. Easy to operate
5. Flexibility

6.2 APPLICATIONS

1. Smart Home Appliances
2. Home safety and security
3. Lighting control
4. Industrial Automation
5. Laboratory Automation
6. Vehicle Automation

CHAPTER-7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

In conclusion, the implementation of smart home automation using IOT technology offers a seamless and efficient way to control various devices within a household. By integrating microcontrollers like the ESP32 and connecting them to the internet, users can remotely access and control their home appliances and systems through a web interface or mobile application. The IoT-based smart home system enhances convenience by allowing users to manage devices from anywhere with an internet connection. Whether it's adjusting lighting, regulating temperature, or monitoring security cameras, homeowners have greater flexibility and control over their living environment. Moreover, the inclusion of sensors for detecting hazards such as flammable gas adds an extra layer of safety to the smart home ecosystem. In the event of a gas leak or other emergencies, the system can automatically respond by shutting off electronic devices and issuing alerts to homeowners via email or other notification channels. Overall, smart home automation using IOT not only simplifies daily tasks but also enhances safety and peace of mind for homeowners, offering a modern and efficient solution for managing residential environments.

7.2 FUTURE SCOPE

The future scope for smart home automation using IoT (Internet of Things) is vast and promising.

1. **Integration with AI and Machine Learning:** Future smart homes will likely incorporate more advanced AI algorithms to learn user behavior and preferences. This could lead to predictive actions and adaptive automation, where the system anticipates the user's needs and adjusts settings accordingly.

2. **Enhanced Energy Efficiency:** IOT-enabled devices will continue to evolve to optimize energy usage within the home. This could include features such as smart thermostats that adjust temperature based on occupancy patterns or lighting systems that automatically dim or turn off when natural light is sufficient.
3. **Expansion of IOT Ecosystem:** The range of IOT devices available for smart homes will expand, including everything from appliances and security systems to health monitoring devices and environmental sensors. Interoperability between these devices will improve, allowing for seamless integration and centralized control.

4. **Increased Security Measures:** As smart home devices become more prevalent, cyber security will become an even greater concern. Future developments will focus on enhancing security protocols and implementing measures such as end-to-end encryption, biometric authentication, and secure firmware updates.

5. **Integration with Smart Grids and Renewable Energy Sources:** Smart homes will increasingly interact with the broader energy infrastructure, including smart grids and renewable energy sources like solar panels and battery storage. This will enable optimized energy usage, load balancing, and even selling excess energy back to the grid.

6. **Health and Wellness Monitoring:** IOT-enabled devices will play a larger role in monitoring occupants' health and wellness within the home. This could include wearable sensors, smart mattresses, and air quality monitors that provide insights and recommendations for improving overall well-being.

7. **Voice and Gesture Control:** Natural language processing and gesture recognition technologies will continue to advance, enabling more intuitive control of smart home devices through voice commands or gestures.

8. **Augmented Reality Interfaces:** Augmented reality (AR) interfaces could revolutionize the way users interact with their smart homes, allowing for immersive experiences where virtual overlays provide information and control options within the physical environment.

9. **Autonomous Systems:** Eventually, smart homes may become fully autonomous, where AI-driven systems manage all aspects of home life, from adjusting the thermostat to ordering groceries and scheduling maintenance tasks.

10. **Privacy and Ethical Considerations:** As smart home technology advances, there will be increasing scrutiny on privacy issues and ethical considerations surrounding data collection, usage, and ownership. Future developments will need to prioritize user consent, data protection, and transparent practices.

Overall, the future of smart home automation using IOT holds tremendous potential to revolutionize the way we live, offering increased convenience, efficiency, and safety while also posing new challenges and considerations for society to address.

APPENDIX – A

```
#include <Arduino.h>

#include <WiFi.h>

#include <ESPAsyncWebServer.h>

#include <ESP32_MailClient.h>

#define ANALOG_PIN 35

const char* ssid = "vikas";

const char* pass = "vikas2006";

#define emailSenderAccount    "homemoniteringsystem@gmail.com"

#define emailSenderPassword   "dsivznmdtvpcazjp"

#define smtpServer            "smtp.gmail.com"

#define smtpServerPort        465

#define emailSubject          "ALERT! GAS LEAKED"

void sendEmailNotificationVibration(int sensorStatus);

const int relaypin1 = 15;

const int relaypin2 = 2;

const int relaypin3 = 5;

const int relaypin4 = 18;

bool relayState1 = false;

bool relayState2 = false;

bool relayState3 = false;

bool relayState4 = false;

AsyncWebServer server(80);

void setup() {

  Serial.begin(115200);

  WiFi.begin(ssid, pass);

  pinMode(ANALOG_PIN, INPUT);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.println(WiFi.localIP());
  // set relay pin output
  pinMode(relaypin1, OUTPUT);
  pinMode(relaypin2, OUTPUT);
  pinMode(relaypin3, OUTPUT);
  pinMode(relaypin4, OUTPUT);
  // set relay initially low
  digitalWrite(relaypin1, HIGH);
  digitalWrite(relaypin2, HIGH);
  digitalWrite(relaypin3, HIGH);
  digitalWrite(relaypin4, HIGH);
#define BLYNK_TEMPLATE_ID "TMPL38mBmPH4P"
#define BLYNK_TEMPLATE_NAME "SMART HOME"
#define BLYNK_AUTH_TOKEN "DlaWmSe--nVFOWJIlaYj9DXrAxWU1LW4"
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
  String html = R"html(
    <html>
      <head>
        <style>
          .switch {
            position: relative;
            display: inline-block;
            width: 90px;
            height: 54px;
          }
          .switch input {
            opacity: 0;
            width: 0;
            height: 0;
          }
          .slider {
            position: absolute;
            cursor: pointer;
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            background-color: #ccc;
            -webkit-transition: .4s;
            transition: .4s;
            border-radius: 34px;
          }
          .slider:before {
            position: absolute;
            content: "";
            height: 40px;
            width: 40px;
            left: 8px;
            bottom: 8px;
            background-color: rgb(253, 248, 252);
            -webkit-transition: .4s;
            transition: .4s;
            border-radius: 50%;
          }
          input:checked + .slider {
            background-color: #2196F3;
          }
          input:focus + .slider {
            box-shadow: 0 0 1px #2196F3;
          }
          input:checked + .slider:before {
            -webkit-transform: translateX(26px);
            -ms-transform: translateX(26px);
            transform: translateX(26px);
          }
          section{
            text-align: center;
          }
          section h1{
            background-color: rgb(20, 192, 235);
            color: white;
            font-size: 38px;
          }
          section p{
            font-size: 28px;
            text-transform: capitalize;
```

```
      font-family: Georgia, 'Times New Roman', Times, serif;
    }
    .home{
      display: flex;
      justify-content: center;
      align-items: center;
      height: 0vh;
    }
    .relay{
      border:1px solid #e0d1d1;
      padding:1em;
      text-align: center;
      margin: 1em;
      box-shadow: 0 3px 6px rgba(0,0,0,0.1);
      transition: transform 0.3s;
      border: 1px solid rgb(43, 40, 40);
    }
    .relay:hover{
      transform: translateY(-5px);
    }
  </style>
  <script>
    function toggleRelay(relay) {
      var xhr = new XMLHttpRequest();
      xhr.open('GET', '/toggle?relay=' + relay, true);
      xhr.send();
    }
  </script>
</head>
<body>
  <section>
    <h1>SMART HOME AUTOMATION USING IOT</h1>
    <p>UNDER THE ESTEEMED GUIDANCE BY :</p>
    <p>Mrs. N. SIVALEELA . MTECH</p>
    <p>PRESENTED BY : </p>
    </p>K.VIKASH (21296-EC-037)</p>
    </p>G.V.V.S.S.S CHAITANYA (21296-EC-025)</p>
    </p> M.YESHWANTH(21296-EC-055)</p>
    </p>A.HEMANTH (12196-EC-001)</p>
    <img
src="https://res.cloudinary.com/ddqsy5em1/image/upload/v1709456659/CEC_k6xtgj.j
pg" alt="">
  </section>
   <div class = "HOME">
     <div class="relay">
      <h2>LIGHT 1</h2>
      <label class='switch'>
        <input type='checkbox' onchange='toggleRelay(1)' )" + String(relayState1 ?
"checked" : "") + R"('>
        <span class='slider'></span>
      </label>
    </div>
    <div class="relay">
      <h2>LIGHT 2</h2>
      <label class='switch'>
```

```
              <input type='checkbox' onchange='toggleRelay(2)' )" + String(relayState2 ?
"checked" : "") + R"('>
              <span class='slider'></span>
            </label>
          </div>
          <div class="relay">
            <h2>FAN</h2>
            <label class='switch'>
              <input type='checkbox' onchange='toggleRelay(3)' )" + String(relayState3 ?
"checked" : "") + R"('>
              <span class='slider'></span>
            </label>
          </div>
          <div class="relay">
            <h2>TV</h2>
            <label class='switch'>
              <input type='checkbox' onchange='toggleRelay(4)' )" + String(relayState4 ?
"checked" : "") + R"('>
              <span class='slider'></span>
            </label>
          </div>
        </div>
      </body>
    </html>
  )html";
request->send(200, "text/html", html);
  });
  server.on("/toggle", HTTP_GET, [](AsyncWebServerRequest *request) {
    if (request->hasParam("relay")) {
      int relay = request->getParam("relay")->value().toInt();
      if (relay == 1) {
        relayState1 = !relayState1;
        digitalWrite(relaypin1, relayState1 ? LOW : HIGH);
        request->send(200, "text/plain", relayState1 ? "Relay 1 Turned ON" : "Relay 1
turned OFF");
      } else if (relay == 2) {
        relayState2 = !relayState2;
        digitalWrite(relaypin2, relayState2 ? LOW : HIGH);
        request->send(200, "text/plain", relayState2 ? "Relay 2 turned OFF" : "Relay 2
turned ON");
      } else if (relay == 3) {
        relayState3 = !relayState3;
        digitalWrite(relaypin3, relayState3 ? LOW : HIGH);
        request->send(200, "text/plain", relayState3 ? "Relay 3 turned OFF" : "Relay 3
turned ON");
      } else if (relay == 4) {
        relayState4 = !relayState4;
        digitalWrite(relaypin4, relayState4 ? LOW : HIGH);
        request->send(200, "text/plain", relayState4 ? "Relay 4 turned OFF" : "Relay 4
turned ON");
      } else {
        request->send(400, "text/plain", "Invalid request");
      }
    }
  });
```

```
 server.begin();
}
void loop() {
  int sensorValue = analogRead(ANALOG_PIN);
  int sensorStatus = sensorValue > 2500;
 if (sensorValue > 2500) {
 digitalWrite(relaypin1, HIGH);
 digitalWrite(relaypin2, HIGH);
 digitalWrite(relaypin3, HIGH);
 digitalWrite(relaypin4, HIGH);
 Serial.println("Gas detected!");
 delay(1000);
 Serial.println(sensorValue);
 sendEmailNotificationVibration(sensorStatus);
 }
 else {
   Serial.println("gas not detected");
   delay(1000);
 }
}
 // SMTP STARTS
 void sendEmailNotificationVibration(int sensorStatus ) {
   SMTPData smtpData;
   // Set the SMTP Server Email host, port, account, and password
   smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
   smtpData.setSender("ESP32_GAS_Alert", emailSenderAccount);
```

```
    // Set Email priority or importance High, Normal, Low or 1 to 5 (1 is highest)
    smtpData.setPriority("High");
    // Set the subject
    smtpData.setSubject("ALERT! GAS Detected");
    // Compose the email message
    String emailMessage = "GAS  detected! Check the Kitchen area.";
    // Set the message with HTML format
    smtpData.setMessage(emailMessage, true);
    // Add recipients
    smtpData.addRecipient("karrothuvikas2006@gmail.com");
    // Send the email
    if (!MailClient.sendMail(smtpData)) {
      Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
    }
    else {
      Serial.println("Email sent successfully");
    }
    smtpData.empty(); // Clear the data for the next use
  }
```