

Introduction

India is an agricultural-based country. Our ancient people completely depended on agricultural harvesting. Agriculture is a source of livelihood of majority Indians and greatly impacts the country's economy. In dry areas or in case of inadequate rainfall, irrigation becomes difficult. So, it needs to be automated for proper yield and handled remotely for farmer safety. Increasing energy costs and decreasing water supplies point out the need for better water management. Irrigation management is a complex decision-making process to determine when and how much water to apply to a growing crop to meet specific management objectives. If the farmer is far from the agricultural land, he will not be noticed of current conditions. So, efficient water management plays an important role in irrigated agricultural cropping systems.

A low-cost alternative solution for efficient water management currently in use is irrigation systems that consist of an automated controller to turn ON & OFF the control valves, which in turn helps the farmers by managing the water supply to the crop fields and helps in better crop production. This project probes into the design of the automated irrigation system based on ESP32-S2. This IoT project is to design and develop a low-cost feature that is based on an IoT platform for the water irrigation system. The project uses an ESP32-S2 which is used as a controller to process the information. The aim of the implementation is to demonstrate that Smart Irrigation can be used to reduce water use.

Literature Review

2.1 General Introduction

A literature survey is fundamental to understanding and developing the idea. A literature survey not only summarizes the knowledge of the area or field but also gives us an idea of what had to be done. In the following section, we discuss the issues and works related to the Smart Irrigation system.

2.2 Literature Survey

2.2.1 Smart irrigation

Smart irrigation to the plants making use ATMEGA 328 microcontrollers tested by Bishnu Deo Kumar Et al. in their work to reduce expenses and optimize the use of water. The entire system is turned ON/OFF based on the measured values from the temperature sensor and the humidity sensor which are connected to the internal ports of the microcontroller via a comparator. Whenever there is a variation in temperature and humidity of the environment these sensors sense the changes and give an interrupt signal to the microcontroller thus the motor is activated, along with a buzzer is used to indicate that the pump is on.

2.2.2 Arduino-Uno based Smart Irrigation

Kavya Monisha K Et al. proposed a smart irrigation system that is being effectively used in the field of agriculture. This smart irrigation system, using Arduino-Uno, checks the moisture level in the soil. If the moisture level in the soil is low, it automatically sends an alert message and turns on the water motor to flow water to the soil. If the moisture level in the soil is sufficient, it switches off the motor. This system helps in agricultural crop growth and soil maintenance. The smart irrigation system reduces the effect caused by insufficient rainfall. This irrigation system prevents an excess of water from flowing into the soil which causes wastage of water, and electricity, and damage to the soil, effectively.

2.2.3 Android and Arduino

A smart irrigation system using Arduino and Android is suggested by Santhiya P Et al. The existing system uses pc and cloud computing to store the data which is read by sensors and the drip is controlled. The proposed system uses Bluetooth to ON/OFF the drip wherever we go. The Microcontroller Arduino is used to control this System and Android is connected to the controller and controls the drip. The sensors are to take values from their surroundings and store them in a controller.

2.3 Summary

The above-discussed papers presented us with a broad perspective on how to tackle what we have aimed at and gave us the different possible approaches to follow to build our project i.e., Smart Irrigation system.

Problem Formulation

The problem statement and objectives of this proposed farmer-friendly Smart Irrigation system project are presented in this section.

3.1 Problem statement

The problem of inefficient water usage in agriculture due to manual irrigation techniques has been a long-standing issue. Traditional methods of irrigation rely on guesswork and human intuition to determine how much water to use, which can result in over or under-watering of crops. This can lead to crop failure, soil erosion and water waste.

3.2 Objectives

- i. To provide an automatic turn ON/OFF facility in the agricultural irrigation system
- ii. To control the water flow on a timely basis using solenoid valves in each irrigation lane

Methodology

4.1 Proposed block diagram of Smart irrigation system:

The proposed irrigation system consists of a microcontroller that is programmed to control the irrigation lanes. Each lane is connected to a solenoid valve. These solenoid valves are controlled by the microcontroller. The microcontroller is programmed in such a way that the pump and valves are turned ON/OFF with the help of web server. The switching of solenoid valves is controlled to cover the entire agricultural land uniformly. The pump will be turned on with the help of a relay. Thus, supplying water to the pipelines whose valves are controlled with the help of solenoid valves.

The block diagram is shown in figure 4.1 depicts the methodology followed for the implementation of Smart irrigation system.

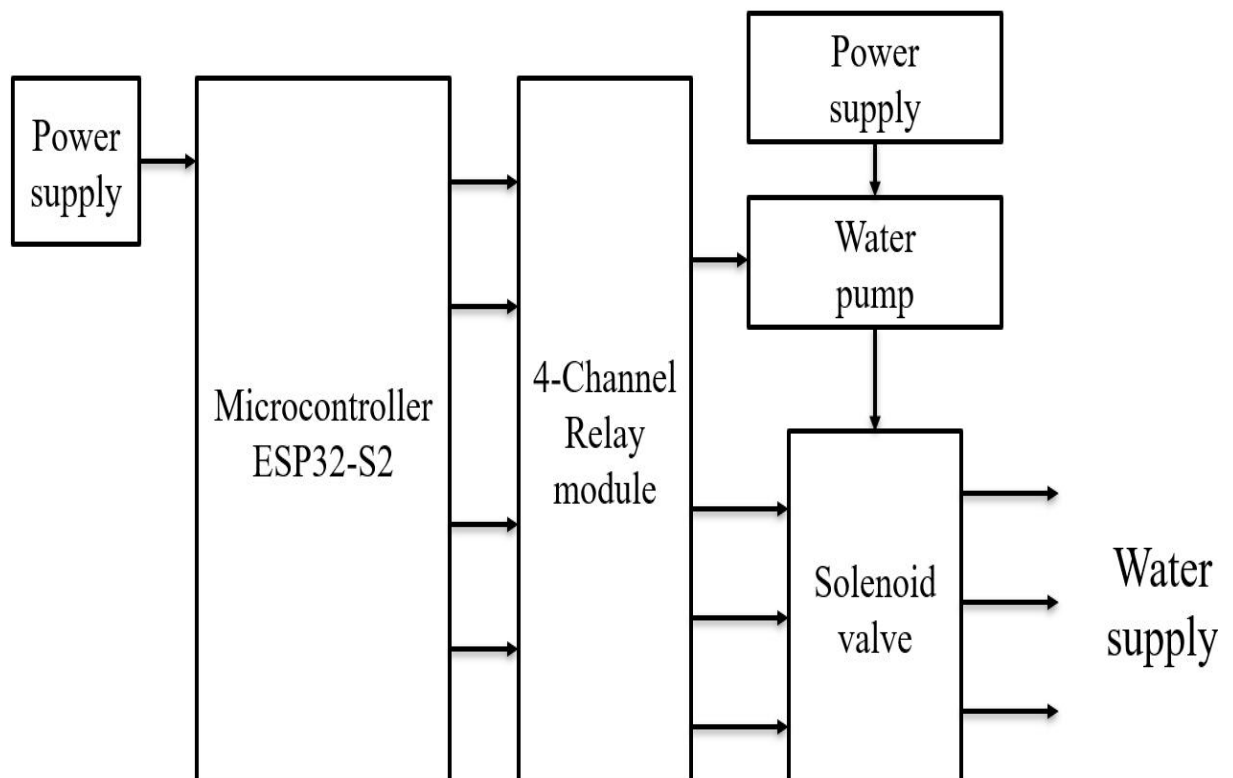


Figure 4.1: Block diagram of Smart irrigation system

4.2 Flow chart of Smart irrigation system:

The Smart irrigation system is an ESP32-S2 based web server. It allows the users to control the pump and the solenoid valves remotely through a web interface. It establishes the Wi-Fi connections which listens for client request and processes them accordingly. It reads the requested parameters to determine the desired state of the pump and valves. Based on the received commands, it updates the states of the corresponding GPIO pins, turning them ON or OFF. The program also generates an HTML webpage with buttons to control the pump and the valves, displaying their current states. It continuously loops waiting for the client request and updating the system accordingly. The flowchart of Smart irrigation system is shown in figure 4.2

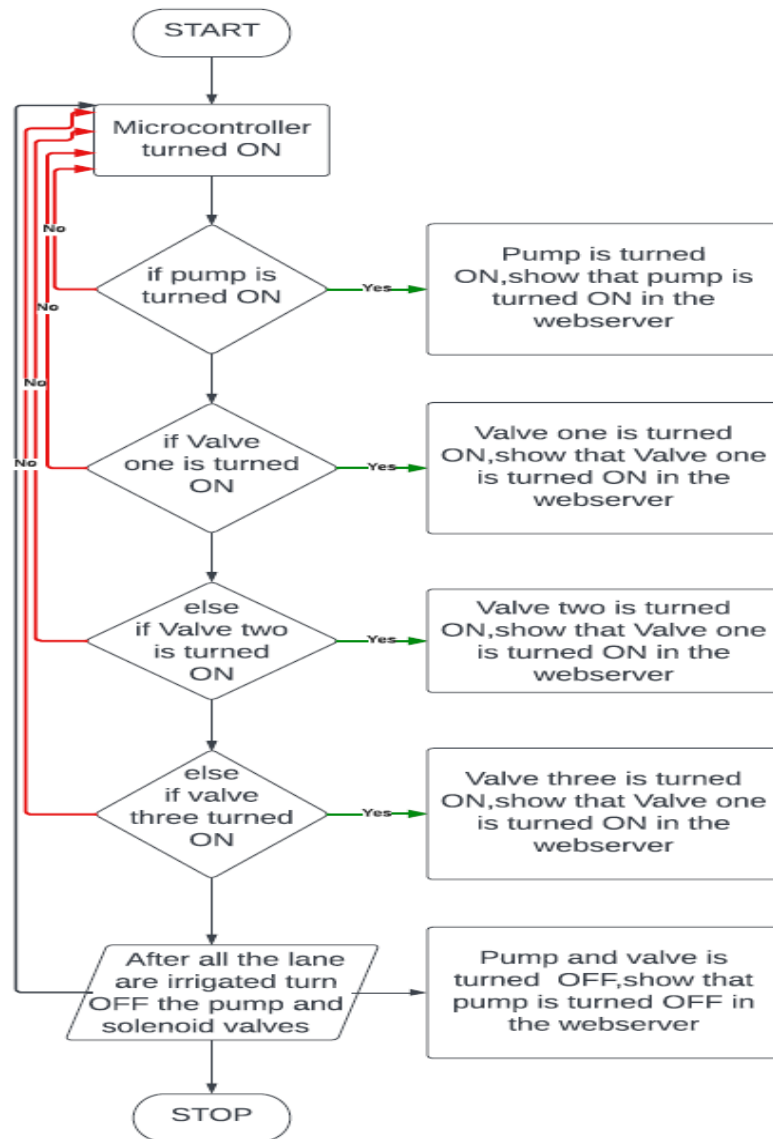


Figure 4.2: Functional flowchart of Smart irrigations system

Chapter-5

Hardware and Software description

5.1 Hardware description

5.1.1 ESP32-S2:

The ESP32-S2 is a highly integrated, low-power, Wi-Fi-enabled microcontroller designed by Espressif Systems. It is the successor to the popular ESP32 chip and offers enhanced

security and performance. The ESP32-S2 is equipped with a 240 MHz single-core Xtensa processor and supports both 2.4 GHz Wi-Fi and BLE 5.0 communication protocols.

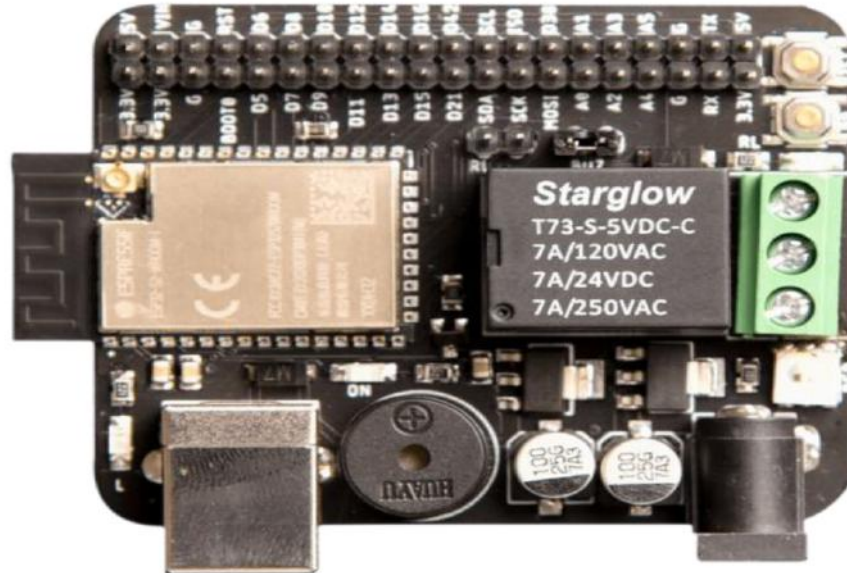


Figure 5.1: ESP32-S2

Specifications:

The ESP32-S2 is a powerful and low-cost microcontroller module designed by Espressif Systems, which is ideal for developing Internet of Things (IoT) devices. Here are some key specifications.

1. **CPU:** The ESP32-S2 is powered by a single-core Xtensa LX7 processor that runs at up to 240 MHz.
2. **Memory:** The module comes with 128 KB of ROM, 320 KB of SRAM, and 16 KB of RTC memory.
3. **Wi-Fi:** The ESP32-S2 supports 802.11 b/g/n Wi-Fi with up to 150 Mbps data rate, making it ideal for IoT applications that require wireless connectivity.
4. **Security:** The module includes a hardware encryption engine that supports AES, SHA-2, RSA, and ECC algorithms, which can help protect your device and data.
5. **Peripherals:** The ESP32-S2 comes with a range of peripherals, including UART, SPI, I2C, I2S, PWM, ADC, and DAC, which make it easy to interface with sensors, actuators, and other devices.

6. **Operating Voltage:** The module operates at a voltage range of 2.0V to 3.6V, making it compatible with a wide range of power sources.
7. **Operating Temperature:** The module can operate in a temperature range of -40°C to 125°C, making it suitable for harsh environments.

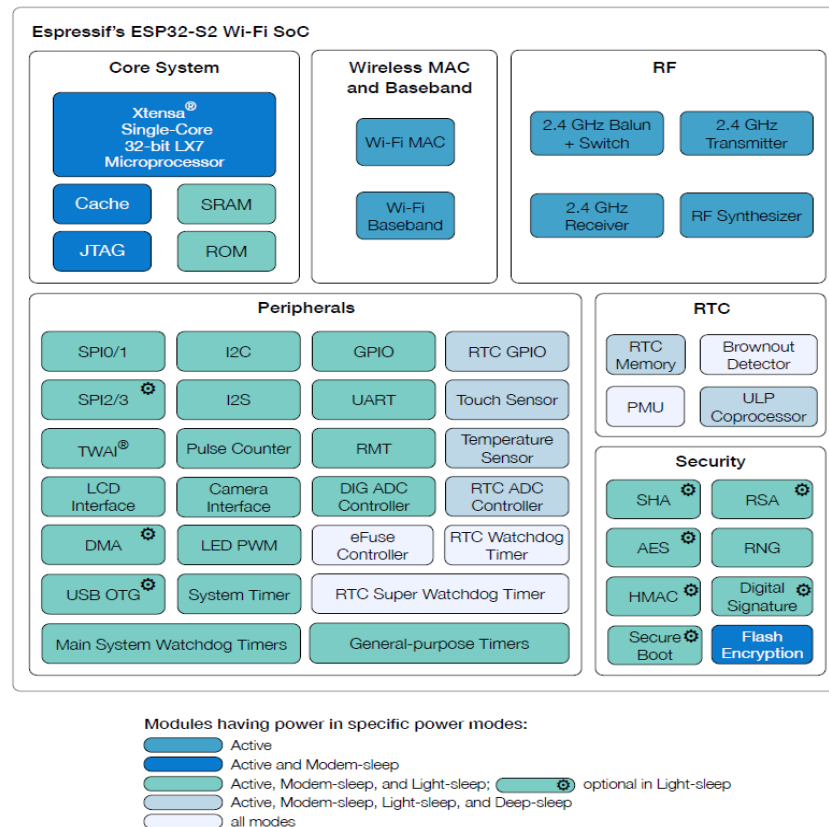


Figure 5.2: Block diagram of ESP32-S2

The block diagram of ESP32-S2 is shown in the figure 5.2. ESP32-S2 is a highly-integrated, low-power, 2.4 GHz Wi-Fi System-on-Chip (SOC) solution. It includes a Wi-Fi subsystem that integrates a Wi-Fi MAC, Wi-Fi radio and baseband, power amplifier, low noise amplifier (LNA) etc. The chip is fully compliant with the IEEE 802.11b/g/n protocol and offers a complete Wi-Fi solution.

Pin configurations:

The ESP32-S2 comes in different package options, including QFN and SOIC. The pinout varies slightly depending on the package, but in general, the ESP32-S2 has the following pins:

1. **Ground (GND)** - This is the ground pin.

2. **Power (VCC)** - This is the power supply pin, which should be connected to a regulated 3.3V power source.
3. **GPIO pins** - The ESP32-S2 has a number of general-purpose input/output (GPIO) pins that can be used for various purposes, such as controlling LEDs, reading sensors, or communicating with other devices. These pins are labeled GPIO0 through GPIO21.
4. **Analog input pins** - The ESP32-S2 has two analog input pins, labeled ADC1 and ADC2.
5. **UART pins** - The ESP32-S2 has two UART interfaces, labeled UART0 and UART1. These interfaces can be used for serial communication with other devices.
6. **I2C pins** - The ESP32-S2 has two I2C interfaces, labeled I2C0 and I2C1. These interfaces can be used for communicating with I2C-compatible devices, such as sensors or LCD displays.
7. **SPI pins** - The ESP32-S2 has two SPI interfaces, labeled SPI0 and SPI1. These interfaces can be used for communicating with SPI-compatible devices, such as flash memory or digital-to-analog converters.
8. **SDIO pins** - The ESP32-S2 has an SDIO interface that can be used for interfacing with SD cards or other SDIO-compatible devices.
9. **Ethernet pins** - The ESP32-S2 supports Ethernet connectivity and has pins for connecting to an Ethernet jack.
10. **USB pins** - The ESP32-S2 has a USB interface that can be used for programming and debugging the chip, as well as for connecting to USB-compatible devices.
11. **JTAG pins** - The ESP32-S2 has pins for connecting to a JTAG interface for debugging and programming purposes.

Overall, the ESP32-S2 is a highly capable microcontroller with a wide range of communication and interface options. Its flexible pinout and support for various communication protocols make it suitable for a wide range of applications, from IoT devices to industrial control systems.

5.1.2 Relay

A relay is an electrically operated switch (5v). A relay is used to control a circuit by a low power signal or where several circuits must be controlled by one signal.



Figure 5.3: Relay with the driver circuit

The figure 5.3 shows the relay used to realize the proposed project. This is a 5V 4-channel relay interface board, and each channel needs a 15-20mA driver current. It can be used to control various appliances and equipment with a large current. It is equipped with high-current relays that work under AC250V 10A or DC30V 10A. It has a standard interface that can be controlled directly by the microcontroller.

Relay can reduce the need for high amperage wiring and switches, which are expensive and take up space. Therefore, switching to relays in your electronic systems can reduce the size or weight of a casing, for instance, or allow manufacturers to fit more functionality into a space of the same size.

PIN Descriptions:

Supply voltage - 3.75V to 6V

Trigger current – 5mA

Current when the relay is active - ~70mA (single), ~300mA (all four)

Relay maximum constant voltage – 250V AC, 30V DC

Relay maximum current – 10A

VCC: Positive supply voltage

IN1--IN4: Relay control port

5.1.3 Solenoid Valve

The figure 5.4 is an electromechanically operated solenoid valve. Solenoid valves differ in the characteristics of the electric current they use, the strength of the magnetic field they generate, the mechanism they use to regulate the fluid, and the type and characteristics of fluid they control. The mechanism varies from linear action and

plunger-type actuators to pivoted-armature actuators and rocker actuators. The valve can use a two-port design to regulate a flow or use a three or more port design to switch flows between ports. Multiple solenoid valves can be placed together on a manifold.

Solenoid valves are the most frequently used control elements in fluidics. Their tasks are to shut off, release, dose, distribute or mix fluids. They are found in many application areas. Solenoids offer fast and safe switching, high reliability, long service life, good medium compatibility of the materials used, low control power, and compact design.



Figure 5.4: Solenoid valve

In this system, a solenoid valve was used for controlling the water flow to start or stop the irrigation process according to the control signals received from the microcontroller.

Specifications:

Model: G 1/180 1/2'' x 1/2''

Water solenoid Valve 220V DC 500mA, 0.2 to 10 Bar

Tu=25° C, Tm=60° C

Suitable for commercial Water purifiers, Industrial & food Industry use.

There are normally closed, wet armature types and used for dispensing cold water.

5.1.4 Water Pump



Figure 5.5: Single Phase Motor

The figure 5.5 shows A water pump is an essential tool to pump out water from the garden, pool, or under the ground. It controls the speed of the water and is incredibly useful in conserving water. The pumps come with various designs and capacities to cater to different needs of pumping.

Specifications:

Material: Metal

Package contents: 1 Pump

Head: 6 – 26 m, Capacity: 33 – 6 Rpm, Flow Rate: 17 Lpm.

Thermal Overload protector: No; Adjustable Speed: No

Brand	Kirloskar
Material	Metal
Item Dimensions L x W x H	20 x 22 x 22 Centimeters
Power Source	Corded Electric
Item Weight	7 Kilograms
Maximum Flow Rate	4500 Liters Per Hour

5.2 Software description:

5.2.1 Arduino IDE software:

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any supported controller board.

Arduino is an open-source hardware and software company, project, and user community that designs and manufactures single-board microcontrollers and microcontroller kits for building digital devices. Arduino board designs use a variety of microprocessors and controllers. The boards are equipped with sets of Digital and Analog input/output (I/O) pins that may be interfaced to various expansion boards or breadboards and other circuits. The boards feature serial communications interfaces, including Universal Serial Bus (USB) on some models, which are also used for loading programs. The microcontrollers can be programmed using the C and C++ programming languages, using a standard API which is also known as the Arduino language, inspired by the Processing language, and used with a modified version of the Processing IDE. In addition to using traditional compiler toolchains, the Arduino project provides an integrated development environment (IDE) and a command line tool developed in Go.

5.2.2 Introduction to web server:

When we talk about web server in the context of the Internet of Things. The web server is nothing but the location where our website files stores and Web server process these files to clients based on the request of a client. Website files are web pages of any site. The webpage is stored in a server of our hosting company. So, when we click on the link to open, we request a client to a server of our hosting company.

After receiving the particular page request, the hosting company server processes the request to display the webpage. Similarly, we can use ESP32 as a host for storing web pages, and when someone requests a web page through a local network, esp32 will serve this webpage. The picture below provides a view of client-server-based communication. Figure 5.6 explains client-server-based communication.

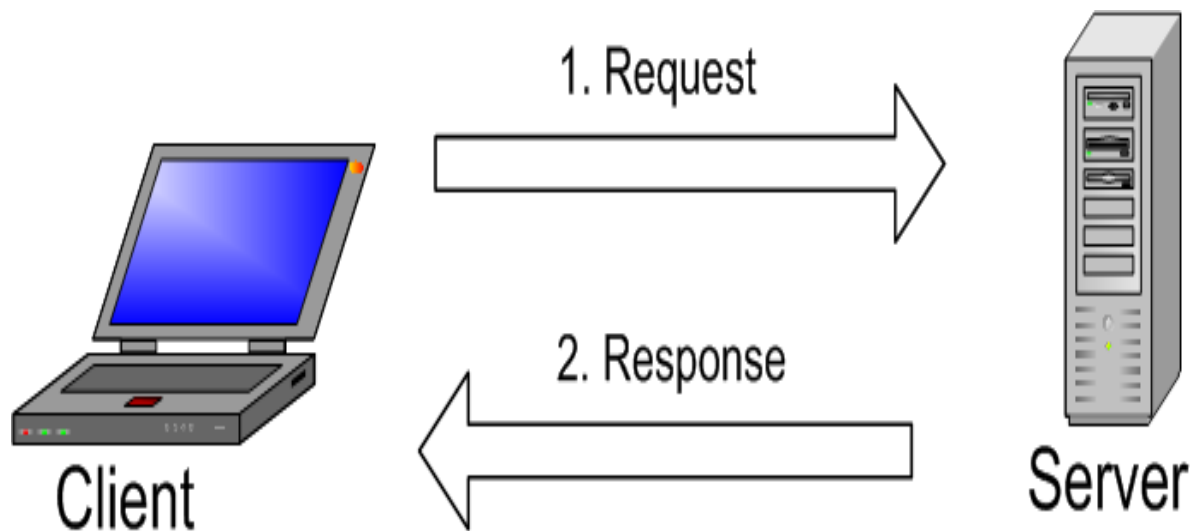


Figure 5.6: Client-Server based communication

Web Server and Client Communication:

This is a basic idea of server and client-based communication over the internet. HTTP, which is also known as Hypertext Transfer control is a protocol that is used to transfer data between the web client and Web server. The web server in this project will be ESP32 and the web client will be any web browser or Android application. Whenever a web client needs to access any web files, it starts an HTTP GET request to the web server.

ESP 32 modes of operation:

It can be used as station mode only where it can connect to an existing network. But this Wi-Fi module has very important features to use a soft access point mode, station mode and both modes at the same time.

Station Mode or STA:

In this mode, ESP32 board connects to our Wi-Fi network through a router. So, the medium of communication between the web client and ESP32 is the router. It gets the IP address from the Wi-Fi router. With this IP address, web clients can access the Web server through an existing local network.

The figure 5.7 explains the station mode of operation in web server communication.

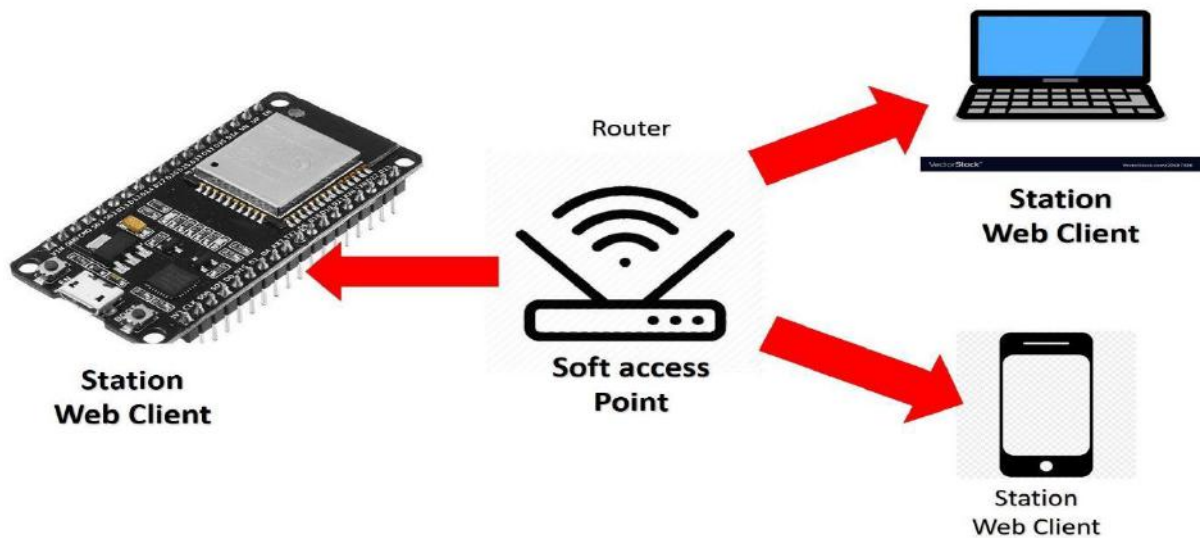


Figure 5.7: Station Mode

Soft Access Point mode:

In this mode, ESP32 is used to create its own wireless Wi-Fi network similar to our existing Wi-Fi router. In this mode, we do not need to connect ESP2 to Wi-Fi network. Up to 5 devices can connect to this Wi-Fi network created by this Wi-Fi board.

Figure 5.8 explains the soft access point mode if communication in web server communication.

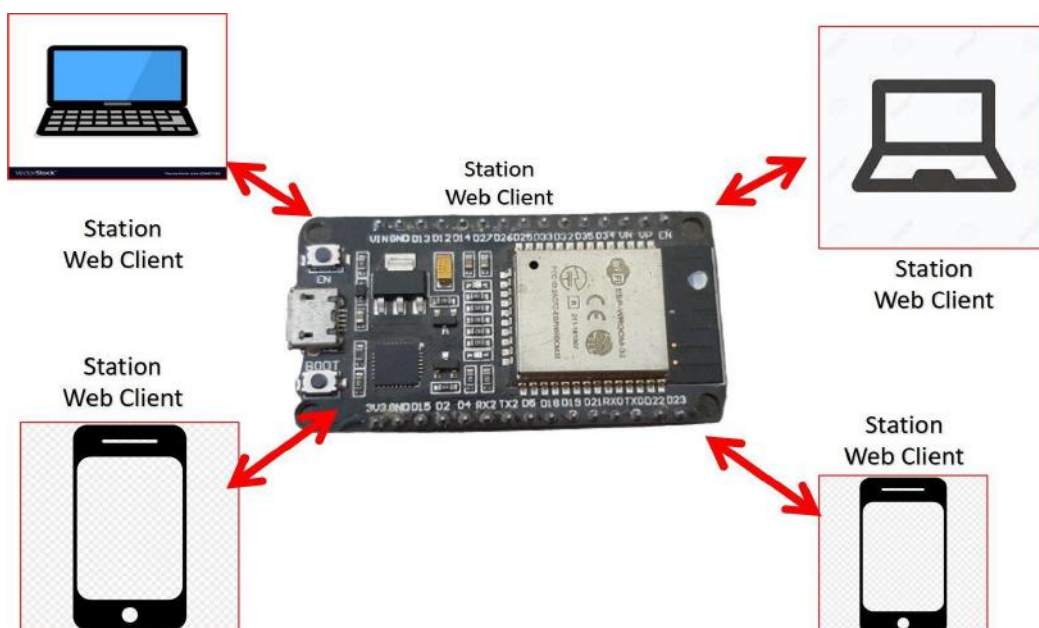


Figure 5.8: Soft access point mode

Use of ESP32 as a Web server in station mode:

We can turn any computer windows or Linux into a web server. But our computer should remain connected to the internet all the time. Similarly, we can use the ESP32 development board as a host server.

The figure 5.9 explains how ESP32 can be used as web server in station mode.

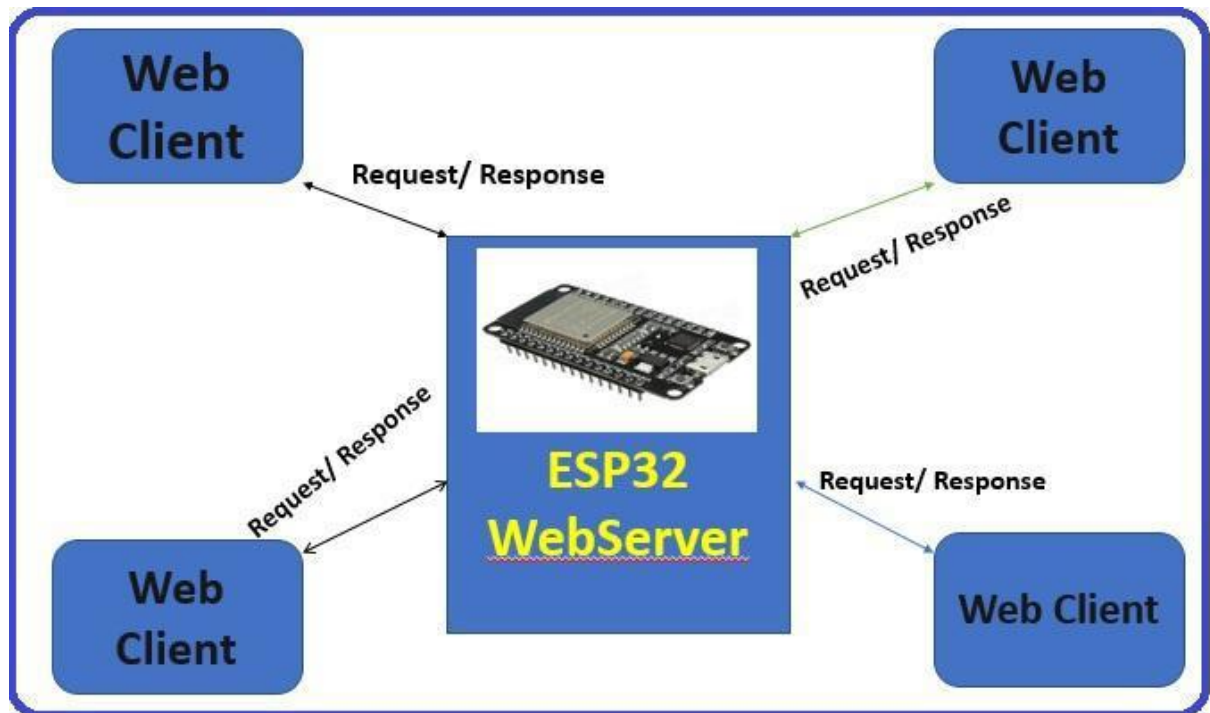


Figure 5.9: ESP32 used as a Web server

Getting IP Address:

We can have multiple clients and a single host server like ESP32 or any IOT development board. When we connect any device to the network, the IP address is assigned to it. So, we can use this assigned IP address to refer to this device. For example, when we connect ESP32 development board to network, the IP address will be given to it. We will use this IP address to access the ESP32 server over the web browser.

The figure 5.10 explains how to get a IP address of ESP32.



Figure 5.10: Getting IP Address

How to access the Web page:

To access ESP32 web pages, we need to get its IP address which is assigned to it when it connects to the internet. In the programming part, how to get its IP address has been show in the programming part.

After getting the IP address, simply place it in a web browser, we will get the response from the ESP32 host server. For example, the web files which we stored in ESP32 are like the one shown below. It will show this web file in our web browser.

ESP32 Web server SMART IRRIGATION 2023

SMART IRRIGATION 2023

**Press on button to turn on PUMP
and VALVES and off button to turn
off PUMP and VALVES**

PUMP is off



VALVE one is off



VALVE two is off



VALVE three is off



Figure 5.11: ESP32 Web page

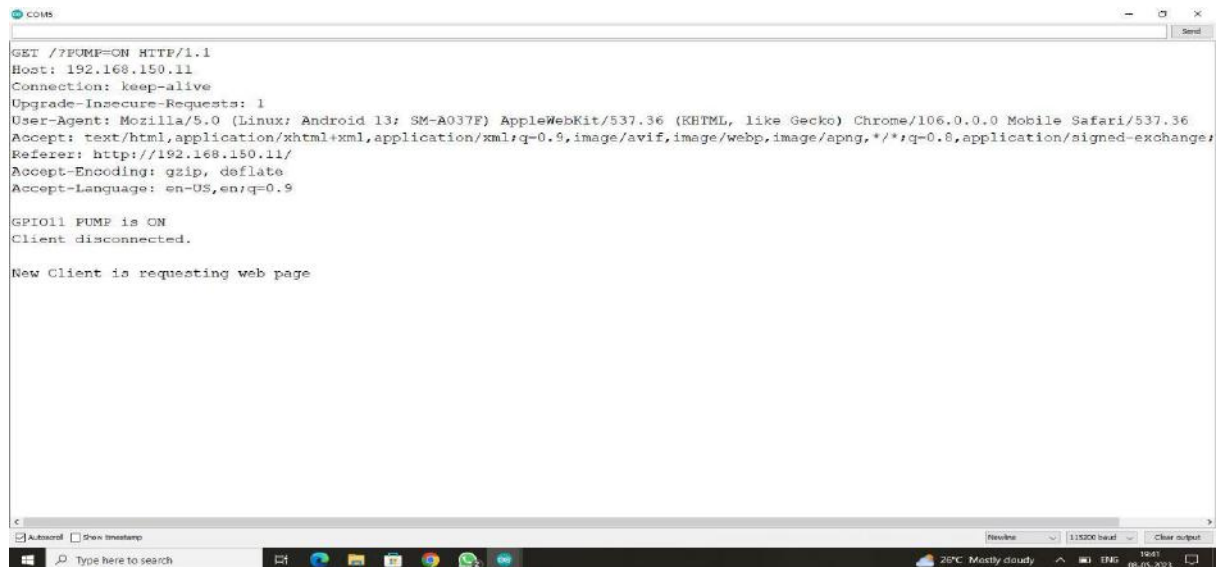
Figure 5.11 represents the ESP32 web server user interface of Smart irrigation System.

This web server is designed using HyperText Markup Language (HTML) through which the farmer takes control of the whole system. To turn ON and OFF pump and solenoid valve there are separate buttons so that user can take control of the system easily and remotely. Through this web-based user interface, users can easily manage and customize their smart irrigation system, ensuring efficient water usage and maintain optimal conditions for plant growth.

Results and Discussion

6.1 Results obtained through Serial Monitor:

The figure 6.1 shows that the pump is turned ON



```

COM5
GET /?PUMP=ON HTTP/1.1
Host: 192.168.150.11
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO11 PUMP is ON
Client disconnected.

New Client is requesting web page
  
```

Figure 6.1: Pump ON

The figure 6.2 shows the station mode of operation in web server communication:



```

COM5
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO11 PUMP is ON
Client disconnected.

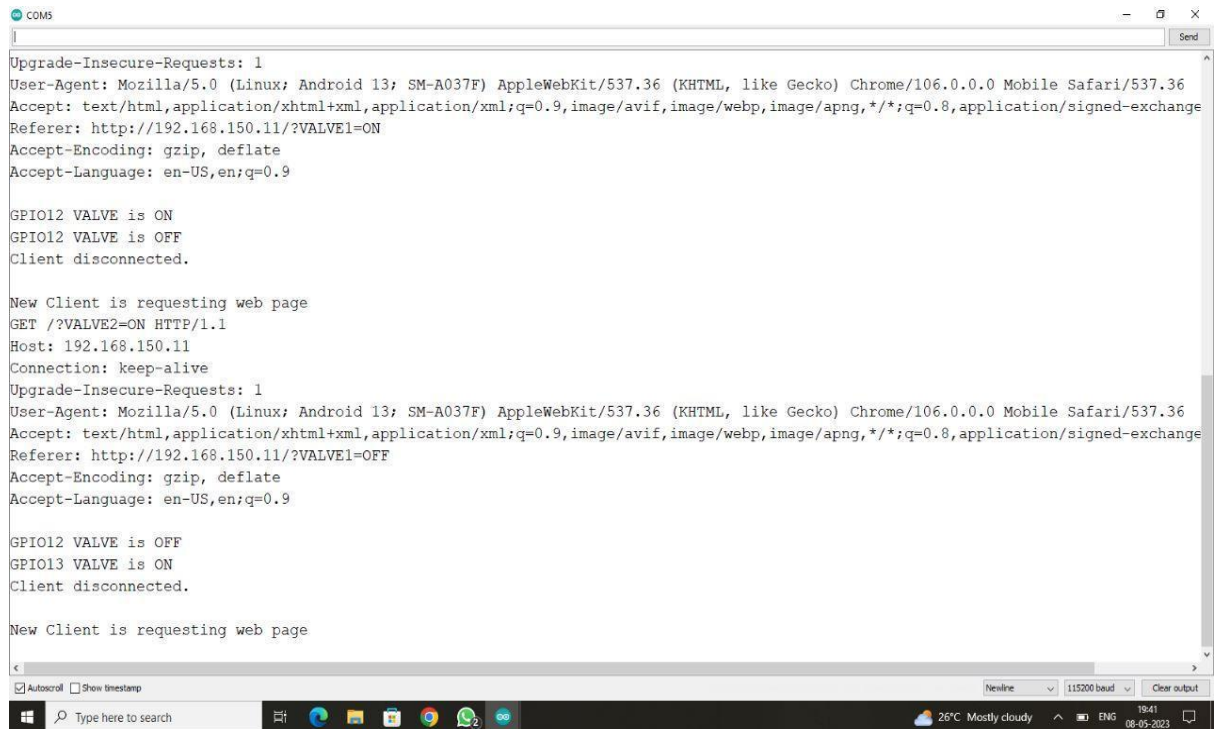
New Client is requesting web page
GET /?VALVE1=ON HTTP/1.1
Host: 192.168.150.11
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.0
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO11 PUMP is ON
GPIO12 VALVE is ON
Client disconnected.

New Client is requesting web page
  
```

Figure 6.2: Lane 1 ON

Below figure 6.3 shows that 1st lane is OFF, and the second lane has been turned ON:



```
COMS
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://192.168.150.11/?VALVE1=ON
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO12 VALVE is ON
GPIO12 VALVE is OFF
Client disconnected.

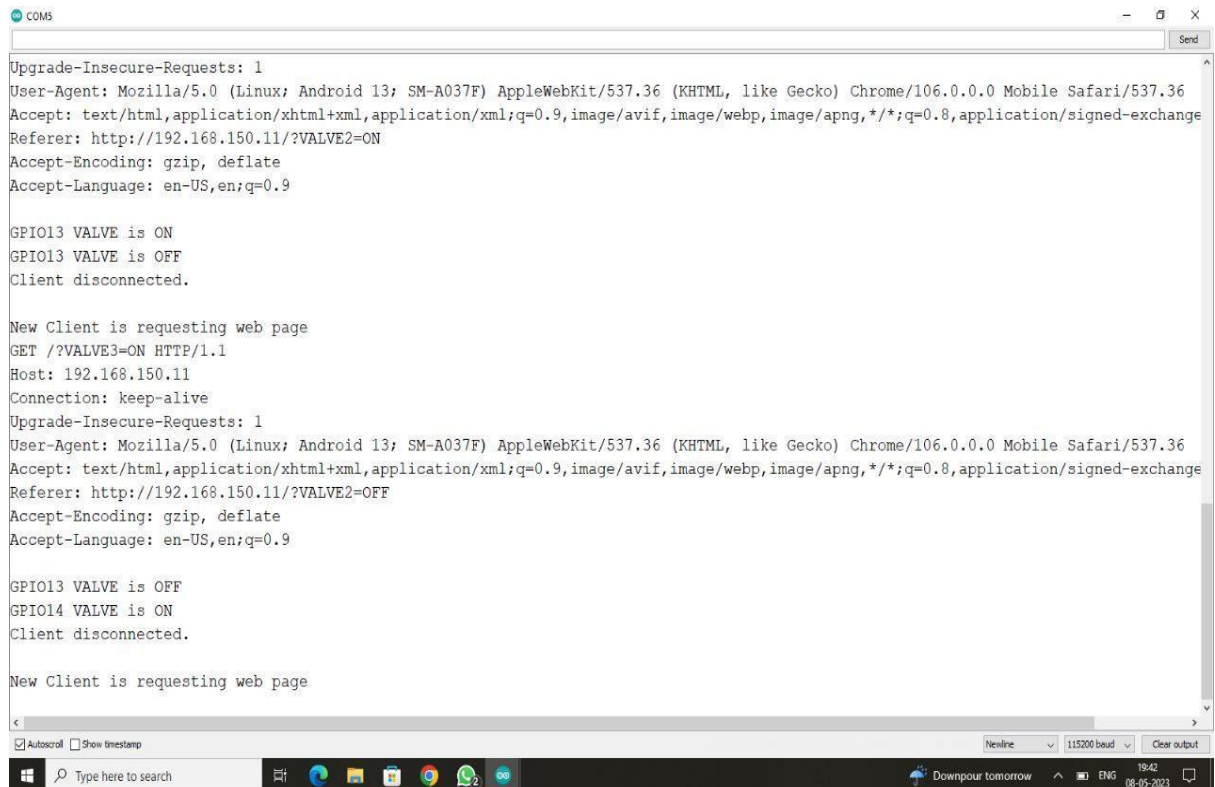
New Client is requesting web page
GET /?VALVE2=ON HTTP/1.1
Host: 192.168.150.11
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://192.168.150.11/?VALVE1=OFF
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO12 VALVE is OFF
GPIO13 VALVE is ON
Client disconnected.

New Client is requesting web page
```

Figure 6.3: Lane 1 OFF, Lane 2 ON

The figure 6.4 shows that the 3rd lane is off:



```
COMS
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://192.168.150.11/?VALVE2=ON
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO13 VALVE is ON
GPIO13 VALVE is OFF
Client disconnected.

New Client is requesting web page
GET /?VALVE3=ON HTTP/1.1
Host: 192.168.150.11
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Linux; Android 13; SM-A037F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Mobile Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange
Referer: http://192.168.150.11/?VALVE2=OFF
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

GPIO13 VALVE is OFF
GPIO14 VALVE is ON
Client disconnected.

New Client is requesting web page
```

Figure 6.4: 3rd lane OFF

6.2 Model of the Project:

The control board is shown in figure 6.5 which includes ESP32-S2 with 4 channel relay modules. The ESP32-S2 has control over the whole system. The relay module consists of 4 channels in which one is connected to a motor and the rest of them are connected to the solenoid valves. The figure 6.7 shows the setup of 3 lanes in which the water flow is controlled by solenoid valve.

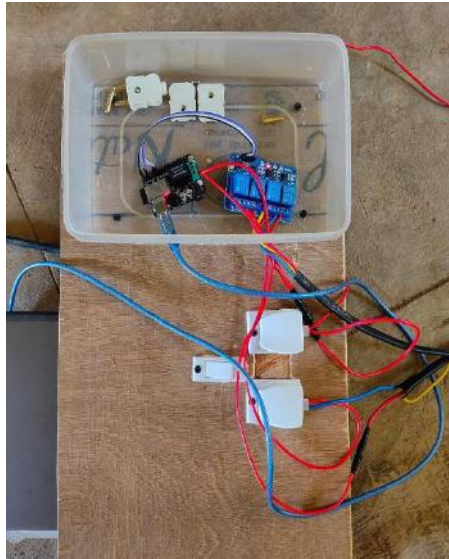


Figure 6.5: Control Board

Figure 6.6: Pump Setup



Figure 6.7: 3-Lane setup

Conclusion

In conclusion, the utilization of ESP32-S2 for smart irrigation systems offers numerous advantages. Its integrated features such as Wi-Fi connectivity, low-power consumption, and real-time data processing capabilities make it an ideal choice for efficient and automated irrigation control. By leveraging plant water requirements, the ESP32-S2 can intelligently adjust irrigation schedules, conserve water, and promote optimal plant growth. Overall, the implementation of smart irrigation using ESP32-S2 has the potential to enhance water management, reduce costs, and contribute to sustainable agricultural practices.

References

- [1] Bishnu Deo Kumar, Prachi Srivastava, Reetika Agrawal, Vanya Tiwari, Microcontroller Based Automatic Plant Irrigation System, International Research Journal of Engineering and Technology (IRJET), Volume: 04 Issue: 05, May -2017.
- [2] Kavya Monisha K., Aishwarya D., Krupaleni K., Smart irrigation system using Arduino Uno, International Journal of Advance Research, Ideas and Innovations in Technology, 2018.
- [3] Santhiya. P, Lakshmitha. G, Monisha. J, Akshaya. T, 2018, Smart Irrigation System Using Arduino and Android, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ETCAN Volume 6, Issue 05, 2018.
- [4] Ghodake, Rahul & Mulani, Altaf, Microcontroller Based Automatic Drip Irrigation System. 109-115. 10.1007/978-3-319-53556-2_12, 2018.

Appendix

The code is given below:

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

```
const char* WIFI_NAME= "Smart Irrigation";
```

```
const char* WIFI_PASSWORD = "onetwice";
```

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

```
String header;
```

```
String PUMP_ONE_STATE = "off";
```

```
String VALVE_ONE_STATE = "off";
```

```
String VALVE_TWO_STATE = "off";
```

```

String VALVE_THREE_STATE = "off";

const int GPIO_PIN_NUMBER_11 = 5;
const int GPIO_PIN_NUMBER_12 = 7;
const int GPIO_PIN_NUMBER_13 = 8;
const int GPIO_PIN_NUMBER_14 = 9;

void setup() {
  Serial.begin(115200);
  pinMode(GPIO_PIN_NUMBER_11, OUTPUT);
  pinMode(GPIO_PIN_NUMBER_12, OUTPUT);
  pinMode(GPIO_PIN_NUMBER_13, OUTPUT);
  pinMode(GPIO_PIN_NUMBER_14, OUTPUT);

  digitalWrite(GPIO_PIN_NUMBER_11, LOW);
  digitalWrite(GPIO_PIN_NUMBER_12, HIGH);
  digitalWrite(GPIO_PIN_NUMBER_13, HIGH);
  digitalWrite(GPIO_PIN_NUMBER_14, HIGH);

  Serial.print("Connecting to ");
  Serial.println(WIFI_NAME);
  WiFi.begin(WIFI_NAME, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print("Trying to connect to Wifi Network");
  }
  Serial.println("");
  Serial.println("Successfully connected to WiFi network");
  Serial.println("IP address: ");

```

```

Serial.println(WiFi.localIP());
server.begin();
}

void loop(){
WiFiClient client = server.available();
if (client) {
Serial.println("New Client is requesting web page");
String current_data_line = "";
while (client.connected()) {
if (client.available()) {
char new_byte = client.read();
Serial.write(new_byte);
header += new_byte;
if (new_byte == '\n') {

if (current_data_line.length() == 0)
{

client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println("Connection: close");
client.println();

if (header.indexOf("PUMP=ON") != -1)
{
Serial.println("GPIO11 PUMP is ON");
PUMP_ONE_STATE = "on";
digitalWrite(GPIO_PIN_NUMBER_11, HIGH);
}
if (header.indexOf("PUMP=OFF") != -1)

```

```

{
Serial.println("GPIO11 PUMP is OFF");
PUMP_ONE_STATE = "off";
digitalWrite(GPIO_PIN_NUMBER_11, LOW);
}
if (header.indexOf("VALVE1=ON") != -1)
{
Serial.println("GPIO12 VALVE is ON");
VALVE_ONE_STATE = "on";
digitalWrite(GPIO_PIN_NUMBER_12, LOW);
}
if (header.indexOf("VALVE1=OFF") != -1)
{
Serial.println("GPIO12 VALVE is OFF");
VALVE_ONE_STATE = "off";
digitalWrite(GPIO_PIN_NUMBER_12, HIGH);
}
if (header.indexOf("VALVE2=ON") != -1)
{
Serial.println("GPIO13 VALVE is ON");
VALVE_TWO_STATE = "on";
digitalWrite(GPIO_PIN_NUMBER_13, LOW);
}
if(header.indexOf("VALVE2=OFF") != -1) {
Serial.println("GPIO13 VALVE is OFF");
VALVE_TWO_STATE = "off";
digitalWrite(GPIO_PIN_NUMBER_13, HIGH);
}
if (header.indexOf("VALVE3=ON") != -1)
{
Serial.println("GPIO14 VALVE is ON");
VALVE_THREE_STATE = "on";

```

```

digitalWrite(GPIO_PIN_NUMBER_14, LOW);
}
if (header.indexOf("VALVE3=OFF") != -1)
{
Serial.println("GPIO14 VALVE is OFF");
VALVE_THREE_STATE = "off";
digitalWrite(GPIO_PIN_NUMBER_14, HIGH);
}

```

```

client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:;\">");
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px
auto; text-align: center;}");
client.println(".button { background-color: #4CAF50; border: 2px solid #4CAF50;; color:
white; padding: 15px 32px; text-align: center; text-decoration: none; display:
inline-block; font-size: 16px; margin: 4px 2px; cursor: pointer; }");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;});
// Web Page Heading
client.println("</style></head>");
client.println("<body><center><h1>ESP32 Web server SMART IRRIGATION
2023</h1></center>");
client.println("<center><h2>SMART IRRIGATION 2023</h2></center>");
client.println("<center><h2>Press on button to turn on PUMP and VALVES and off
button to turn off PUMP and VALVES</h3></center>");
client.println("<form><center>");
client.println("<p> PUMP is " + PUMP_ONE_STATE + "</p>");
// If the PIN_NUMBER_11State is off, it displays the ON button
client.println("<center> <button class=\"button\" name=\"PUMP\" value=\"ON\"
type=\"submit\">PUMP ON</button>");

```

```

client.println("<button class=\"button\" name=\"PUMP\" value=\"OFF\"
type=\"submit\">PUMP OFF</button><br><br>");

client.println("<p>VALVE one is " + VALVE_ONE_STATE + "</p>");

client.println("<button class=\"button\" name=\"VALVE1\" value=\"ON\"
type=\"submit\">VALVE1 ON</button>");

client.println("<button class=\"button\" name=\"VALVE1\" value=\"OFF\"
type=\"submit\">VALVE1 OFF</button> <br><br>");

client.println("<p>VALVE two is " + VALVE_TWO_STATE + "</p>");

client.println ("<button class=\"button\" name=\"VALVE2\" value=\"ON\"
type=\"submit\">VALVE2 ON</button>");

client.println ("<button class=\"button\" name=\"VALVE2\" value=\"OFF\"
type=\"submit\">VALVE2 OFF</button></center>");

client.println("<p>VALVE three is " + VALVE_THREE_STATE + "</p>");

client.println ("<button class=\"button\" name=\"VALVE3\" value=\"ON\"
type=\"submit\">VALVE3 ON</button>");

client.println ("<button class=\"button\" name=\"VALVE3\" value=\"OFF\"
type=\"submit\">VALVE3 OFF</button></center>");

client.println("</center></form></body></html>");

client.println();

break;
}

else
{
current_data_line = "";
}
}

else if (new_byte != '\r')
{
current_data_line += new_byte;
}
}
}

// Clear the header variable
header = "";

```

```
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}
```