# Project Report:

# Title:

## "Hybrid-Mode Appliance Control and Monitoring System: Touch Panel & Wi-Fi App"

## 1) Introduction:

Traditional appliance control systems, often limited by manual switches or wired remotes, fail to offer flexibility, efficiency, and remote monitoring capabilities. These systems are static, require physical proximity, and do not integrate well with modern technological advances, such as wireless connectivity or IoT. As technology continues to evolve, there is a growing demand for more dynamic, user-friendly, and remotely accessible solutions for appliance control and monitoring.

This project aims to address these limitations by developing a **Hybrid-Mode Appliance Control and Monitoring System** using a **touch panel** and a **Wi-Fi app**. The proposed system will allow users to control and monitor appliances both locally and remotely, offering a scalable and flexible solution that can integrate into future IoT ecosystems. The system will enhance the convenience, efficiency, and accessibility of appliance management, making it more adaptable to modern lifestyles.

## 2) Project Components:

### 2.1) Hardware Components:

- **NodeMCU ESP8266** – A Wi-Fi-enabled microcontroller for remote communication.
- **wires** – For connections between components.
- **LED lights** – Used as indicators for system status.
- **Relays** – For controlling the power of appliances.
- **DWIN display** – A touch panel display for user interface and local control.

### 2.2) Software Components:

- **DGUS software** – For designing the graphical user interface (GUI) on the DWIN display.
- **Flutter framework** – Used for developing the Wi-Fi app to control and monitor appliances remotely.
- **Arduino IDE** – For writing and uploading the code to the microcontroller (NodeMCU ESP8266).

## 3) Project:

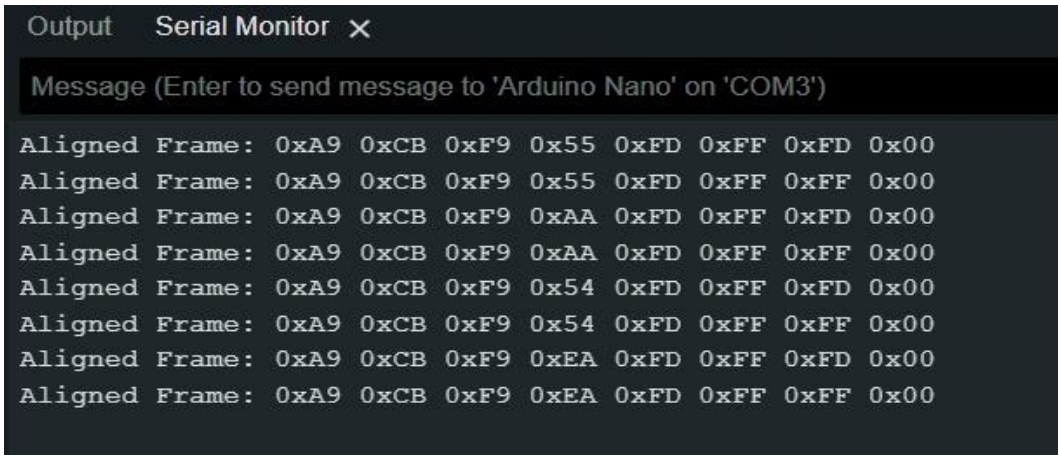
### 3.1) Troubleshooting:

The initial phase of the project involved designing a test **Graphical User Interface (GUI)** for the **DWIN display**. We defined variable pointers (VP) for the touch buttons in the GUI. After

connecting the DWIN display to the **ESP8266 microcontroller**, we began printing the hexadecimal frames received from the display on the **serial monitor**. Initially, the frames appeared random, which required further analysis.

To troubleshoot, we used an **Arduino NANO** and printed the output on the serial monitor. Upon observation, we noticed some consistency in the frames, but the bytes were misaligned because the **first byte** was undefined. We tested each byte as the potential first byte, but encountered another issue: the frame had 8 bytes, while our array size was 9 bytes. After adjusting the array size to match 8 bytes, the frame alignment was corrected, and the first byte was properly defined.

**Results:**

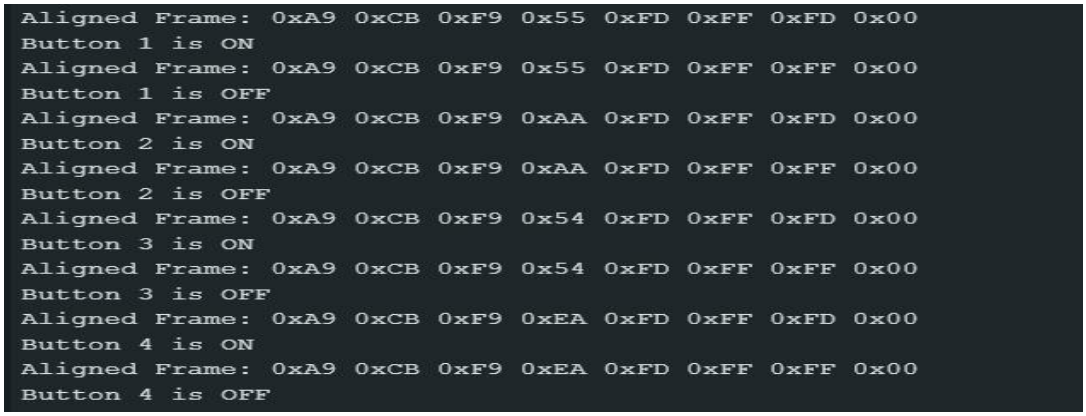



**Upon observing the corrected frame, the structure was as follows:**

| Byte 1 | Byte 2 | Byte 3 | Button | Byte 4 | Byte 5 | State | Null |
|---|---|---|---|---|---|---|---|

Based on this observation, we reprogrammed the system, implementing a **switch-case structure** to handle the data correctly according to the frame output. This ensured the system's behaviour was aligned with the intended functionality.

As a result, the system is now able to provide the correct output based on each input received via the DWIN display, confirming that the interface is working as expected.

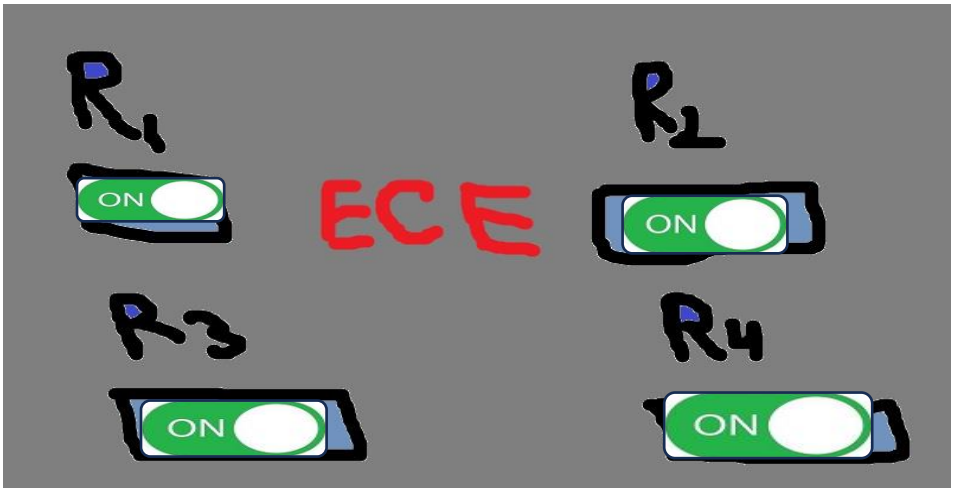



So, we are getting the proper output based on each input given via DWIN display.

Then we connected the LEDs to it and tested, each LED represent a connection for a relay in real project.

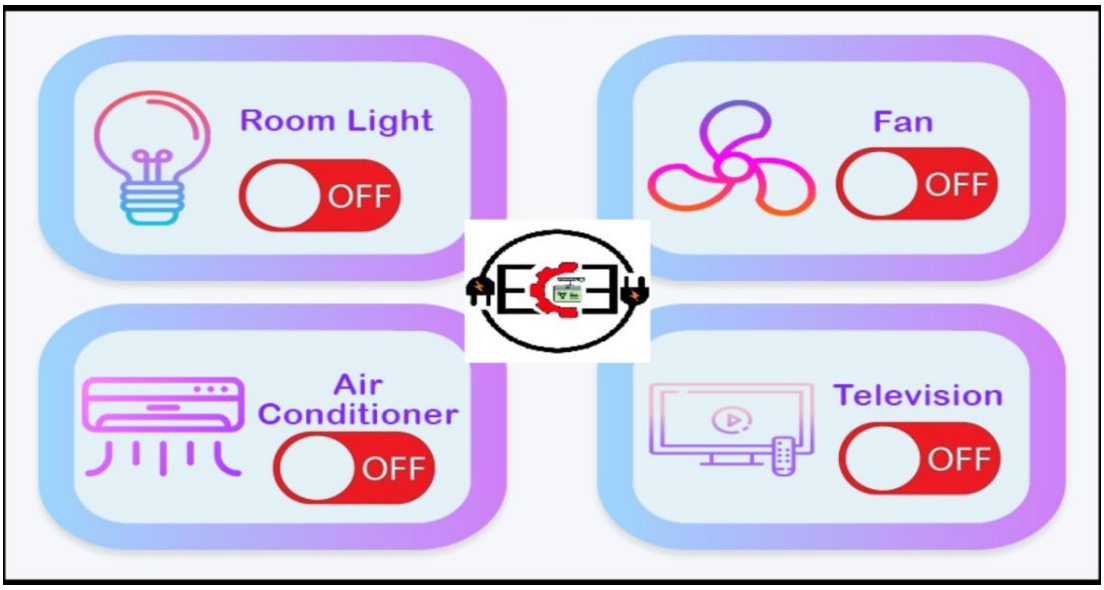
After confirming the accuracy of the display and control mechanism, we connected **LEDs** to represent the relays in the real-world application. Each LED simulates the connection for a relay, enabling us to test the system's response. The LEDs light up or turn off based on the inputs provided via the DWIN display, which corresponds to controlling the connected relays, verifying that the control system functions as intended.

**3.2) For testing the following GUI was used:**

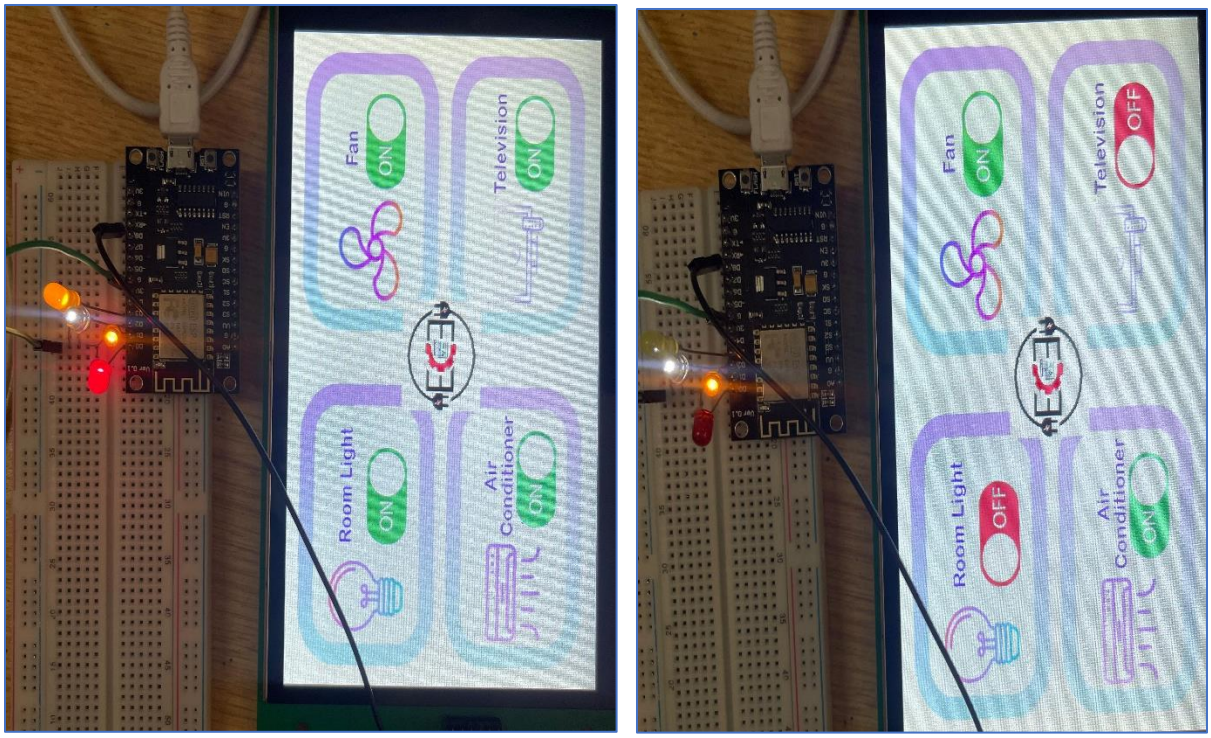



**3.3) Project Final Interface and Code:**

**The DWIN Control Interface is as follows:**




*Control based on DWIN interface*

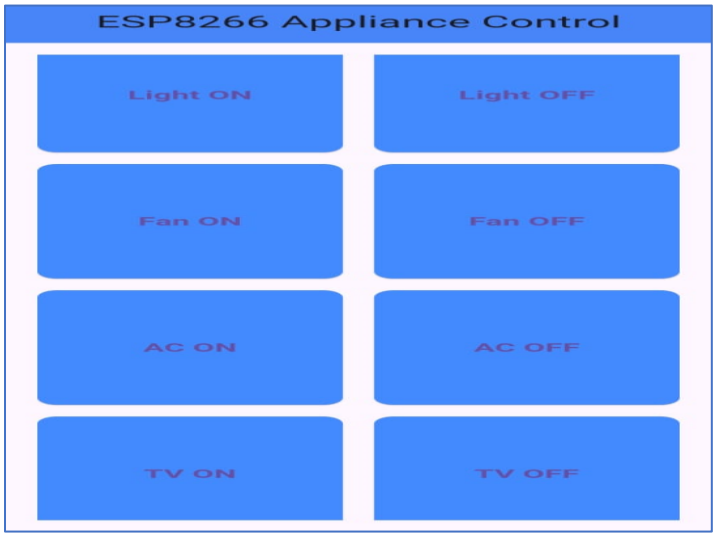



*Project Output*

## 3.4) Flutter App Interface:

For the app development, we chose the **Flutter framework** because of its cross-platform capabilities, enabling us to design the app for multiple platforms, including **Web**, **iOS**, and **Android**. This provides flexibility and wider accessibility for users to control and monitor the appliance system on different devices.

## The interface looks like follows




*Mobile Interface for App*

**3.5) Code:**

```
#include <ESP8266WiFi.h>
#include
<ESP8266WebServer.h>

// Wi-Fi credentials for
Access Point
const char* ssid =
"ESP8266_AP";
const char* password =
"12345678";

// Create a web server on
port 80
ESP8266WebServer
server(80);

// Appliance pins
#define LIGHT_PIN D0  //
Controlled by Button 1
#define FAN_PIN D1    //
Controlled by Button 2
#define AC_PIN D2     //
Controlled by Button 3
#define TV_PIN D3     //
Controlled by Button 4

unsigned char
Buffer[8];  // For DWIN
display frames

void setup() {
  Serial.begin(115200);
  Serial.println("Initiali
zing...");

  // Configure appliance
pins as output
  pinMode(LIGHT_PIN,
OUTPUT);
  pinMode(FAN_PIN,
OUTPUT);
  pinMode(AC_PIN, OUTPUT);
  pinMode(TV_PIN, OUTPUT);
```

```
   server.on("/ac/off", []() {
    digitalWrite(AC_PIN, LOW);
    server.send(200, "text/plain",
"AC OFF");
  });

  server.on("/tv/on", []() {
    digitalWrite(TV_PIN, HIGH);
    server.send(200, "text/plain",
"TV ON");
  });

  server.on("/tv/off", []() {
    digitalWrite(TV_PIN, LOW);
    server.send(200, "text/plain",
"TV OFF");
  });

// Start the server
  server.begin();
  Serial.println("Web Server
Started");
}

void loop() {
  // Handle Wi-Fi client requests
  server.handleClient();

  // Handle DWIN display commands
  handleDWINInput();
}

// Function to handle DWIN display
input
void handleDWINInput() {
  if (Serial.available() >= 8) {
    // Read the DWIN frame
    while (Serial.available() > 0)
{
      unsigned char byte =
Serial.read();
      if (byte == 0xA9) { // Start
byte found
        Buffer[0] = byte;
```

```
  // Turn all appliances OFF initially
  digitalWrite(LIGHT_PIN, LOW);
  digitalWrite(FAN_PIN, LOW);
  digitalWrite(AC_PIN, LOW);
  digitalWrite(TV_PIN, LOW);

  // Set up ESP8266 as Access Point
  WiFi.softAP(ssid, password);
  Serial.println("Access Point Started");
  Serial.print("IP Address: ");
  Serial.println(WiFi.softAPIP());

  // Define server routes for Wi-Fi control
  server.on("/light/on", []() {
    digitalWrite(LIGHT_PIN, HIGH);
    server.send(200, "text/plain", "Light ON");
  });

  server.on("/light/off", []() {
    digitalWrite(LIGHT_PIN, LOW);
    server.send(200, "text/plain", "Light OFF");
  });

  server.on("/fan/on", []() {
    digitalWrite(FAN_PIN, HIGH);
    server.send(200, "text/plain", "Fan ON");
        for (int i = 1; i < 8; i++) {
          while (!Serial.available()); // Wait for each byte
          Buffer[i] = Serial.read();
        }

        // Identify and process button and state
        identifyButtonAndControlPins(Buffer[3], Buffer[6]);
        break;
      }
    }
  }
}

// Function to identify button and control GPIO pins
void identifyButtonAndControlPins(unsigned char buttonByte, unsigned char stateByte) {
  int pinToControl;

  // Map the button to the corresponding pin
  switch (buttonByte) {
    case 0x55: pinToControl = LIGHT_PIN; break; // Button 1
    case 0xAA: pinToControl = FAN_PIN; break;   // Button 2
    case 0x54: pinToControl = AC_PIN; break;    // Button 3
    case 0xEA: pinToControl = TV_PIN; break;    // Button 4
    default:
      Serial.println("Unknown Button");
      return;
  }

  // Determine ON/OFF state
  bool isOn = (stateByte == 0xFD); // ON if 0xFD, OFF if 0xFF
```

```
  });

  server.on("/fan/off",
[]() {
    digitalWrite(FAN_PIN,
LOW);
    server.send(200,
"text/plain", "Fan OFF");
  });

  server.on("/ac/on", []()
{
    digitalWrite(AC_PIN,
HIGH);
    server.send(200,
"text/plain", "AC ON");
  });
```

```
  // Control the pin
  digitalWrite(pinToControl, isOn ?
HIGH : LOW);

  // Print the status
  Serial.print("Button ");
  Serial.print(buttonByte, HEX);
  Serial.print(" -> ");
  Serial.println(isOn ? "ON" :
"OFF");
}
```

## 4) Project Working:

The project operates as follows:

1. **Touch Panel Input:**
   When a touch button on the DWIN display is tapped, the display sends a UART frame to the ESP8266 microcontroller.

   - The button address in the frame identifies which button was pressed.
   - The state address determines whether the button corresponds to an ON or OFF state.

   Based on this information, the appropriate output pin on the microcontroller is activated or deactivated, which controls the connected load relay.

2. **Wi-Fi Connection with the App:**
   The ESP8266 is configured as an Access Point (AP), enabling a smartphone to connect to it via Wi-Fi.

   - Users connect to the ESP by selecting its SSID and entering the specified password.
   - Once connected, the communication between the smartphone app and the ESP relies on a request-response mechanism.

The app sends a request to update or retrieve the state of a specific button or load.The ESP processes this request and sends a response back to the app, reflecting the updated state.

3. **Load Control and State Update:**
   Based on the input received from either the touch panel or the Wi-Fi-based app, the state of the connected load is updated. The loads, represented by relays (or LEDs during testing), respond to the ON/OFF commands in real time.

This hybrid approach allows the system to be controlled locally through the touch panel and remotely through the Wi-Fi app, offering both flexibility and convenience. The integration of the UART frame mechanism and the Wi-Fi request-response phenomenon ensures seamless communication and control across all components.

# 5) Future work:

## 5.1) Adding More Controls and Features:

In the future, the project can be expanded by adding more touch buttons to the DWIN display, allowing for greater flexibility in controlling additional loads. This enhancement would make the system more versatile and capable of managing multiple devices simultaneously, addressing diverse user requirements. Customization options for specific appliances could also be introduced, improving the system's overall utility and adaptability.

## 5.2) Integration of Sensor Data:

The integration of sensors for environmental monitoring is another key area of improvement. Sensors for pressure, temperature, and humidity can be incorporated to collect real-time data, which can then be displayed on both the DWIN display and the smartphone app. This feature would enable users to monitor environmental conditions alongside controlling their appliances. Furthermore, the sensor data could be transmitted wirelessly to the smartphone app, ensuring accessibility and convenience.

## 5.3) Cloud Storage and Data Analytics:

To enhance the project's scalability, cloud storage capabilities can be introduced to store sensor data for long-term use. By uploading data to the cloud, users can access historical information and perform advanced analytics, such as trend identification, energy consumption optimization, and failure prediction. These capabilities would transform the system into a more intelligent and predictive solution, offering significant benefits for both residential and industrial applications.

## 5.4) Industrial Applications and SCADA Integration:

The project can also be scaled for industrial environments, evolving into a Supervisory Control and Data Acquisition (SCADA) system. By leveraging wireless monitoring and control capabilities, the system can manage industrial machinery and processes more efficiently. This would provide a flexible and cost-effective solution for industrial automation, ensuring real-time control and monitoring with minimal infrastructure requirements.

### 5.5) IoT Integration:

Another critical area of expansion is the integration of the system into an Internet of Things (IoT) ecosystem. By enabling internet-based remote control, the project can support features such as device scheduling, automated alerts, and remote diagnostics. Using IoT communication protocols like MQTT or HTTP would ensure seamless and reliable communication between devices, cloud services, and users, enhancing the system's functionality and usability.

### 5.6) Cross-Platform Scalability:

Finally, the smartphone app developed using the Flutter framework can be refined further to improve its performance across multiple platforms. This cross-platform scalability would ensure a consistent user experience on Android, iOS, and web devices. With these enhancements, the project has the potential to evolve into a comprehensive solution for home automation, industrial monitoring, and IoT-based applications, meeting the growing demands of modern automation and connectivity.

## 6) Learning Outcomes:

### 6.1)Hardware Integration and System Design:

We gained valuable insights into interfacing hardware components such as the DWIN display, NodeMCU ESP8266, relays, and LEDs. This included configuring UART communication and ensuring seamless interaction between the microcontroller and peripheral devices. These skills improved our understanding of embedded system design and hardware integration.

### 6.2) Software Development Skills:

Working with DGUS software, the Flutter framework, and the Arduino IDE provided hands-on experience in software development. We learned to design user-friendly graphical interfaces, program microcontrollers, and create cross-platform applications, enhancing our ability to develop efficient and scalable systems.

### 6.3) Debugging and Problem-Solving:

Throughout the project, we encountered challenges such as debugging UART frames, aligning data bytes, and configuring Wi-Fi-based communication. Overcoming these obstacles sharpened our troubleshooting and problem-solving skills, teaching us how to address real-world integration issues effectively.

### 6.4) IoT and Wireless Communication:

We gained practical experience in implementing IoT concepts by setting up the NodeMCU ESP8266 as an Access Point. Understanding the request-response mechanism for controlling loads wirelessly gave us a solid foundation for designing IoT-enabled systems, preparing us for future projects in this domain.

### 6.5) Collaboration and Project Management:

The project required teamwork, effective communication, and task coordination. By collaborating to design, test, and refine the system, we developed essential project management skills and learned to approach multidisciplinary engineering problems holistically.

### 6.6) Practical Application of Automation Concepts:

This project provided a comprehensive learning experience in automation, demonstrating how to integrate hardware, software, and wireless communication to create a hybrid system for appliance control and monitoring.

## 7) Conclusion:

The Hybrid-Mode Appliance Control and Monitoring System successfully integrates modern technologies to address the limitations of traditional appliance control systems. By combining a touch panel interface with Wi-Fi-based app control, the project provides a flexible, scalable, and user-friendly solution for managing and monitoring loads both locally and remotely. The system demonstrates effective integration of hardware, software, and wireless communication, showcasing the potential for IoT-based automation in both residential and industrial environments. With opportunities for future enhancements, such as sensor integration, cloud storage, and IoT scalability, this project lays the foundation for more advanced and efficient automation solutions tailored to modern lifestyles and industrial needs