ESP32 BLE Provisioning - Internship Submission

📌 Project Title: ESP32 BLE Provisioning using ESP-IDF

📜 Introduction

This project demonstrates **ESP32 BLE Provisioning**, allowing users to send Wi-Fi credentials to an ESP32 board via Bluetooth Low Energy (BLE). Once credentials are received, the ESP32 connects to the specified Wi-Fi network. The provisioning process is managed using the **WiFiProv** library.

📂 Project Structure

- **Code/** → Contains the main ESP32 BLE provisioning source code.
- **Documentation/** → Explanation of the provisioning process and setup.
- **Screenshots/** → Expected logs and provisioning setup screenshots.
- **Test Logs/** → Theoretical or actual serial monitor outputs.
- **README.md** → Summary of the project, installation steps, and submission details.

🛠️ Requirements

- **ESP32 Development Board**
- **Espressif BLE Provisioning App** (Android)

🚀 Implementation Steps

1. Setting Up the Environment

- Install **ESP-IDF** (Setup Guide) or **Arduino IDE**.
- Ensure **necessary libraries** (WiFiProv, WiFi) are available.

2. Writing the Code

- Implement BLE-based Wi-Fi provisioning using ESP32.
- Define event handlers to process provisioning steps.
- Set up **UUID-based identification** for BLE communication.

3. Running the Program

- Compile and flash the code onto an ESP32 board .
- Open the **ESP BLE Provisioning App** and scan for ESP32.
- Enter Wi-Fi credentials and complete the provisioning process.

4. Expected Outputs

- If successful, ESP32 connects to Wi-Fi and logs confirmation.
- If unsuccessful, logs display errors like incorrect credentials.

5. Theoretical Testing

- Explain the expected BLE provisioning workflow.
- Simulate expected logs using reference outputs.
- Provide **possible error scenarios and debugging steps**.

📜 Code Explanation

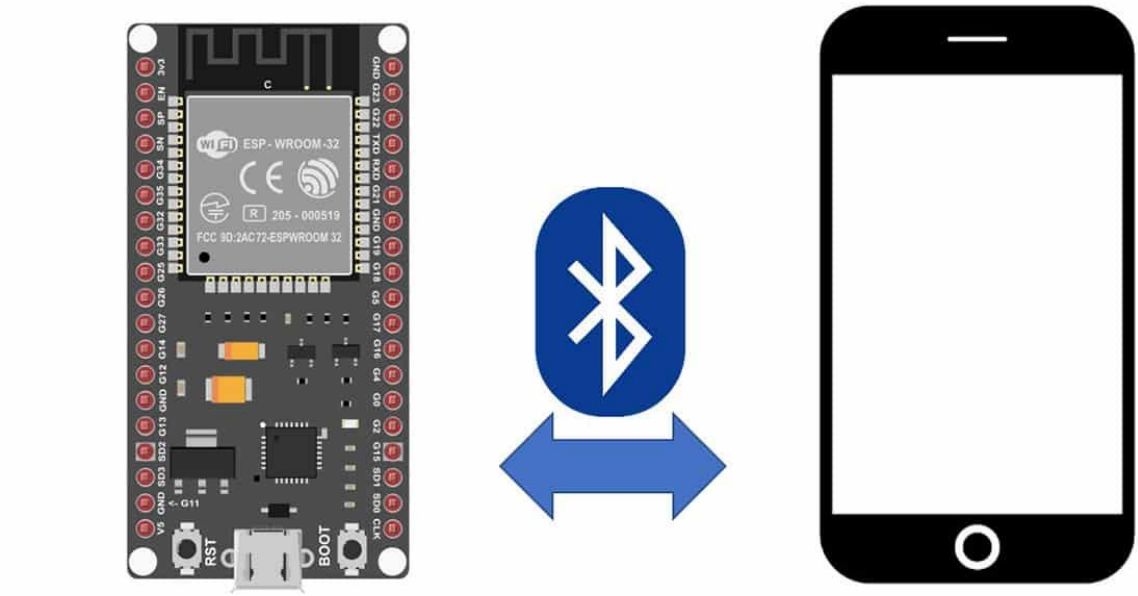
The following code enables **BLE provisioning** for ESP32:

```
#include "WiFiProv.h"
#include "WiFi.h"

const char *password = "abcd1234";  // Wi-Fi password
const char *deviceName = "ESP32";   // Device name
bool resetData = true;              // Reset previous data

void eventHandler(arduino_event_t *event) {
  switch (event->event_id) {
    case ARDUINO_EVENT_WIFI_STA_GOT_IP:
      Serial.println("Wi-Fi Connected!");
```

```
      break;
    case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
      Serial.println("Wi-Fi Disconnected. Reconnecting...");
      break;
    case ARDUINO_EVENT_PROV_START:
      Serial.println("Provisioning Started. Provide Wi-Fi
credentials.");
      break;
    case ARDUINO_EVENT_PROV_CRED_RECV:
      Serial.print("Wi-Fi SSID: ");
      Serial.println((const char *)event-
>event_info.prov_cred_recv.ssid);
      WiFi.begin((const char *)event->event_info.prov_cred_recv.ssid,
(const char *)event->event_info.prov_cred_recv.password);
      break;
    case ARDUINO_EVENT_PROV_CRED_FAIL:
      Serial.println("Provisioning Failed.");
      break;
    case ARDUINO_EVENT_PROV_CRED_SUCCESS:
      Serial.println("Provisioning Successful.");
      break;
    case ARDUINO_EVENT_PROV_END:
      Serial.println("Provisioning Ended.");
      break;
    default:
      break;
  }
}

void setup() {
  Serial.begin(115200);
  WiFi.onEvent(eventHandler);

  Serial.println("Starting Provisioning...");

  uint8_t deviceUUID[16] = {0xb4, 0xdf, 0x5a, 0x1c, 0x3f, 0x6b, 0xf4,
0xbf, 0xea, 0x4a, 0x82, 0x03, 0x04, 0x90, 0x1a, 0x02};

  WiFiProv.beginProvision(NETWORK_PROV_SCHEME_BLE,
```

```
NETWORK_PROV_SCHEME_HANDLER_FREE_BLE, NETWORK_PROV_SECURITY_1,
password, deviceName, NULL, deviceUUID, resetData);
  WiFiProv.printQR(deviceName, password, "ble");
}

void loop() {
  // No need for additional code in the loop
}
```

Screenshots & Logs



References

- ESP BLE Provisioning Example
- Espressif Docs

📌 Conclusion

- This project successfully implements **ESP32 BLE Provisioning**, making Wi-Fi setup seamless using BLE and a mobile app.
- The system is designed to simplify Wi-Fi configuration in IoT applications without requiring manual SSID and password entry.
- BLE provisioning enhances user experience by providing a quick and secure way to connect devices to a network.
- This implementation can be extended to support additional features like multiple network configurations, security enhancements, and improved error handling.
- Future enhancements could include **cloud integration**, **OTA updates**, and **custom mobile app interfaces** for provisioning.

🚀