A black text on a white background

Description automatically generated

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

### A

**Project**

**SYNOPSIS**

### On

# Real-Time Wireless Communication between Two ESP32 Modules using ESP-NOW Protocol

*Submitted by:*

# Rishiraj Rakesh Kumar, 225807062

## ABSTRACT

This project demonstrates real-time wireless communication between two ESP32 microcontrollers using the ESP-NOW protocol. By leveraging the MAC address-based peer-to-peer communication mechanism, the transmitter ESP32 sends string-based data entered through the serial monitor to a paired receiver ESP32, which displays it on its own serial monitor. This low-latency, low-power communication method bypasses traditional Wi-Fi connectivity, making it ideal for IoT-based applications where real-time and efficient data transfer is crucial.

## INTRODUCTION

In modern embedded systems and IoT applications, real-time data communication plays a pivotal role. ESP-NOW, a protocol developed by Espressif, enables device-to-device communication without relying on Wi-Fi routers or internet connectivity. This project involves two ESP32 boards—one acting as a transmitter and the other as a receiver—communicating wirelessly using MAC address-based pairing. The main focus is to showcase real-time, reliable data transmission and reception using the ESP-NOW protocol.

## 3. LITERATURE REVIEW

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No | Title | Authors | Publication | Topics Covered |
| 1 | ESP-NOW Wireless Communication Protocol for Low Power IoT Applications | V. Vignesh, A. Bansal | IJERT, 2020 | Understood how ESP-NOW uses MAC-address-based pairing for data transmission and how it’s more efficient than MQTT or HTTP for small payloads. |
| 2 | Real-Time Sensor Monitoring System Using ESP32 and ESP-NOW Protocol | P. Kumar et al. | IRJET, 2021 | Gained insights on using real-time sensor data transfer between ESP32 devices with minimal latency. |
| 3 | Comparative Study of ESP-NOW and Bluetooth Low Energy | R. Gupta, S. Sharma | IJEEE, 2019 | Reinforced the advantage of ESP-NOW over BLE in terms of power efficiency and broadcast capability for IoT mesh-type communication. |
| 4 | MAC Layer Optimization for Peer-to-Peer Communication in IoT Networks | J. Lee, M. Kim | IEEE Access, 2019 | Provided deeper understanding of MAC-level optimizations and peer information management. |

## 4. NEED FOR THE PROJECT

There is a growing demand for lightweight, peer-to-peer communication in IoT applications, especially where internet infrastructure is absent. This project addresses the need for a simple, efficient communication setup for tasks like telemetry, remote sensing, or device status updates without complex networking stacks or external modules.

**5. OBJECTIVE**

To establish a real-time, wireless communication channel between two ESP32 boards using the ESP-NOW protocol and demonstrate successful data transmission and reception through the Arduino IDE’s serial monitor.

### 6. METHODOLOGY\*\*

* 1. Both ESP32 boards are configured to operate in WIFI\_STA (station mode), which is essential for peer-to-peer communication using ESP-NOW. This mode disables the access point functionality and prepares the ESP32 to act like a Wi-Fi client device. It ensures minimal interference and simplifies the MAC-based transmission setup by eliminating unnecessary SSID discovery and Wi-Fi credentials.

2. The ESP-NOW protocol is initialized using esp\_now\_init(). Following successful initialization, two callback functions are registered: esp\_now\_register\_send\_cb() for handling send status, and esp\_now\_register\_recv\_cb() for receiving data. These callbacks are essential for confirming whether a message was delivered successfully and for processing incoming data, respectively. This makes the communication bi-directional and event-driven.

3. Each ESP32 board needs to recognize its communication partner via a unique MAC address. This is achieved by defining a peer structure esp\_now\_peer\_info\_t and filling in the receiver’s MAC address. The peer is added using esp\_now\_add\_peer(). This step simulates a handshake and ensures data is sent securely to the intended device. It avoids broadcasting to unintended nodes and ensures deterministic communication.

4. The transmitter ESP32 reads real-time input strings from the serial monitor using Serial.readStringUntil('\n'). The input is cleaned using trim() and packed into a struct (myMessage). The struct is then sent using esp\_now\_send() targeting the receiver's MAC address. This showcases manual real-time message initiation, simulating a telemetry or control signal in IoT systems.

5. The receiver ESP32 decodes incoming data in the OnDataRecv() function, copies the content to the incomingMessage structure, and prints it on the serial monitor. This ensures that messages sent from the transmitter are instantly shown on the receiver’s screen, demonstrating an application-level confirmation of communication.

6. Although this setup mainly transmits from one side, both ESP32 boards have send and receive callbacks, making them capable of full duplex communication. This allows for future expansion of the project into more complex two-way messaging or even multi-node networks. Delivery success is printed to the console, ensuring robustness.

\*\*(Refer Appnedix for Source Codes of Transmitter and Receiver ESP-32 modules)

### 7. COMPONENTS REQUIRED

1. 2 × ESP32 Dev Boards

2. USB cables

3. Arduino IDE

4. Serial Monitor for testing

5. Laptop/Desktop for development

### 8. EXPECTED RESULTS

### The expected outcome of this project is the successful implementation of wireless communication between two ESP32 modules using the ESP-NOW protocol. A key result is the successful pairing of the two ESP32 devices through their MAC addresses, allowing for secure and direct communication without the need for an external Wi-Fi network or router. Once paired, the transmitter ESP32 is expected to transmit user-inputted messages in real-time from the serial monitor. These messages, upon reaching the receiver ESP32, should be accurately displayed on its own serial monitor, ensuring the integrity of the transmitted data. Additionally, the system should provide clear confirmation of message delivery success or failure using callback functions, which enhances the reliability and debugging capabilities of the communication system. Overall, the setup should demonstrate a responsive, low-latency, and efficient wireless communication system suitable for small payload transmission in IoT applications.

### 9. PROJECT SCHEDULE

| Week | Task |
| --- | --- |
| 1 | Introduction to ESP32 and ESP-NOW protocol |
| 2 | Setting up development environment |
| 3 | Writing transmitter code and testing |
| 4 | Writing receiver code and pairing devices |
| 5 | Testing real-time data transmission |
| 6 | Final integration and validation |
| 7 | Documentation and report preparation |

**10. REFERENCES**

1. Espressif Systems, “ESP-NOW User Guide”,

https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp\_now.html

2. “ESP-NOW Communication Between ESP32 Boards”, Random Nerd Tutorials, https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/

3. Book: Simon Monk, Programming ESP32 with Arduino IDE, McGraw-Hill, 2020

**11**. **APPENDIX**

### Transmitter ESP32 Code :-

#include <esp\_now.h>  
 #include <WiFi.h>  
  
 uint8\_t broadcastAddress[] = {0x8C, 0x4F, 0x00, 0x2F, 0xAD, 0x94};  
  
 typedef struct struct\_message {  
 char message[32];  
 } struct\_message;  
  
 struct\_message myMessage;  
 struct\_message incomingMessage;  
  
 void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {  
 Serial.print("\r\nLast Packet Send Status:\t");  
 Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");  
 }  
  
 void OnDataRecv(const esp\_now\_recv\_info\_t \*recv\_info, const uint8\_t \*incomingData, int len) {  
 memcpy(&incomingMessage, incomingData, sizeof(incomingMessage));  
 Serial.print("Received Message from Receiver: ");  
 Serial.println(incomingMessage.message);  
 }  
  
 void setup() {  
 Serial.begin(115200);  
 WiFi.mode(WIFI\_STA);  
  
 if (esp\_now\_init() != ESP\_OK) {  
 Serial.println("Error initializing ESP-NOW");  
 return;  
 }  
  
 esp\_now\_register\_send\_cb(OnDataSent);  
 esp\_now\_register\_recv\_cb(OnDataRecv);  
  
 esp\_now\_peer\_info\_t peerInfo;  
 memset(&peerInfo, 0, sizeof(peerInfo));  
 memcpy(peerInfo.peer\_addr, broadcastAddress, 6);  
 peerInfo.channel = 0;  
 peerInfo.encrypt = false;  
  
 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK) {  
 Serial.println("Failed to add peer");  
 return;  
 }  
 Serial.println("Peer added successfully");  
 }  
  
 void loop() {  
 if (Serial.available() > 0) {  
 String userInput = Serial.readStringUntil('\n');  
 userInput.trim();  
 strncpy(myMessage.message, userInput.c\_str(), sizeof(myMessage.message));  
  
 esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*)&myMessage, sizeof(myMessage));  
  
 if (result == ESP\_OK) {  
 Serial.println("Message sent successfully");  
 } else {  
 Serial.println("Message failed to send");  
 }  
 }  
}

### Receiver ESP32 Code :-

#include <esp\_now.h>  
#include <WiFi.h>  
  
uint8\_t broadcastAddress[] = {0x8C, 0x4F, 0x00, 0x2F, 0xAD, 0x94};  
typedef struct struct\_message {  
 char message[32];  
} struct\_message;  
  
struct\_message myMessage;  
struct\_message incomingMessage;  
  
void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {  
 Serial.print("\r\nLast Packet Send Status:\t");  
 Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");  
}  
  
void OnDataRecv(const esp\_now\_recv\_info\_t \*recv\_info, const uint8\_t \*incomingData, int len) {  
 memcpy(&incomingMessage, incomingData, sizeof(incomingMessage));  
 Serial.print("Received Message from Transmitter: ");  
 Serial.println(incomingMessage.message);  
}  
  
void setup() {  
 Serial.begin(115200);  
 WiFi.mode(WIFI\_STA);  
  
 if (esp\_now\_init() != ESP\_OK) {  
 Serial.println("Error initializing ESP-NOW");  
 return;  
 }  
  
 esp\_now\_register\_send\_cb(OnDataSent);  
 esp\_now\_register\_recv\_cb(OnDataRecv);  
  
 esp\_now\_peer\_info\_t peerInfo;  
 memset(&peerInfo, 0, sizeof(peerInfo));  
 memcpy(peerInfo.peer\_addr, broadcastAddress, 6);  
 peerInfo.channel = 0;  
 peerInfo.encrypt = false;  
  
 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK) {  
 Serial.println("Failed to add peer");  
 return;  
 }  
 Serial.println("Peer added successfully");  
}  
  
void loop() {  
 if (Serial.available() > 0) {  
 String userInput = Serial.readStringUntil('\n');  
 userInput.trim();  
 strncpy(myMessage.message, userInput.c\_str(), sizeof(myMessage.message));  
  
 esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*)&myMessage, sizeof(myMessage));  
  
 if (result == ESP\_OK) {  
 Serial.println("Message sent successfully");  
 } else {  
 Serial.println("Message failed to send");  
 }  
 }  
}