**Device to Device Communication with ESP-NOW**

Learn how to use the ESP-NOW protocol to send data between ESP based boards.

AuthorHannes Siebeneicher

Last revision01/05/2024

Introduction

ESP-NOW is a wireless communication protocol developed by Espressif, the company behind the ESP32 microcontroller (MCU). Since the Arduino Nano ESP32 is equipped with that MCU it also supports the ESP-NOW protocol out of the box. It's designed for efficient and low-latency communication between devices, capable of sending up to 250 bytes.

Hardware Requirements

ESP-NOW is supported on the following microcontrollers:

* [**Arduino Nano ESP32**](https://store.arduino.cc/products/nano-esp32)
* generic ESP8266
* generic ESP32
* generic ESP32-S
* ESP32-C series of SoCs

You need two boards or more to communicate via ESP-NOW. Minimum one sender and one receiver.

How It Works

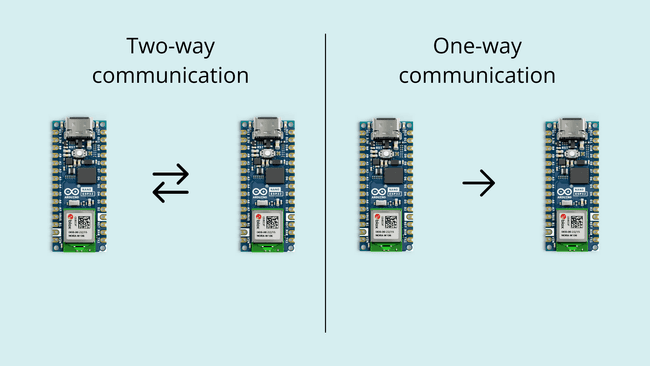
**Communication Protocol**

ESP-NOW operates as a peer-to-peer (P2P) protocol, meaning it allows direct communication between two ESP8266 or ESP32 devices without the need for a central server or access point, e.g. a Wi-Fi® router. Each ESP device has a unique [**MAC address**](https://en.wikipedia.org/wiki/MAC_address) which is used to identify the receiving board.

ESP-NOW can be set up in different ways:

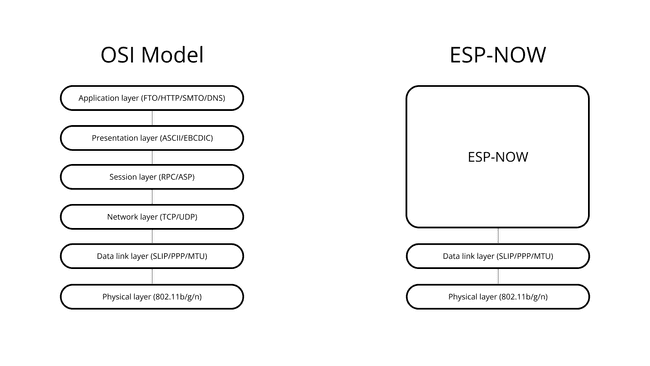
**One-way communication**: In one-way communication mode, one device (the sender) can send data to another device (the receiver) without expecting a response. This mode is often used for scenarios where one device provides data or commands to another device, such as remote sensor readings or control commands.

**Two-way communication**: In two-way communication mode, both devices can exchange data bidirectionally. This mode enables a back-and-forth exchange of information between the devices, allowing for more interactive and responsive communication. It's suitable for applications where devices need to send and receive data from each other, such as remote control systems or interactive IoT devices.

**[](https://docs.arduino.cc/static/e8137f39673a6d0b41afa5fc0567c559/29114/esp-now-communication.png)**ESP-NOW Communication

**Protocol Stack**

ESP-NOW operates primarily at the data-link layer of the [**OSI model**](https://en.wikipedia.org/wiki/OSI_model). In contrast to traditional networking protocols that involve multiple layers, ESP-NOW condenses the communication stack, streamlining the process. By reducing the protocol stack to a single layer, ESP-NOW eliminates the need for complex [**packet headers**](https://www.cloudflare.com/en-gb/learning/network-layer/what-is-a-packet/) and unpackers on each layer. This simplicity results in quicker response times, reducing delays caused by packet loss in congested networks.

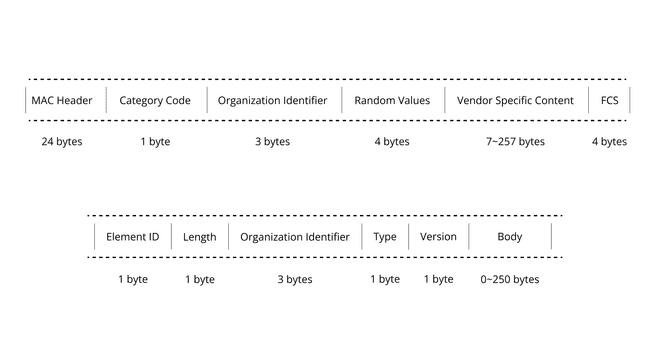
**[](https://docs.arduino.cc/static/9f804c5922fccea427a2e15c53c4e1c0/29114/protocol-stack.png)**TCP/IP and ESP-NOW Protocol Stack

**Lightweight Design**

The protocol minimizes communication overhead, making it more efficient for resource-constrained devices. It also doesn’t perform [**handshakes**](https://en.wikipedia.org/wiki/Handshake_(computing)) as devices are preconfigured to allow for faster data transmission and reduced power consumption.

**Limited Data Quantities**

ESP-NOW is useful when transmitting smaller amounts of data (max 250 bytes). It utilizes action frames for communication. Action frames are specialized data frames that contain information about the action being performed. These frames allow ESP-NOW devices to efficiently exchange data without the overhead associated with more generic data frames.

**[](https://docs.arduino.cc/static/c468f6fc7ce55abf3b78f39c44486f3c/b3931/action-frame.png)**Action Frames

**Low Latency**

As a result of the limited data quantities and its lightweight design ESP-NOW introduces a low latency, which means that devices can quickly exchange data making it suitable for remote controlling other devices, e.g. a radio car.

Limitations

**Limited Range**

ESP-NOW has a limited signal range, typically around 220 meters under ideal conditions. The actual range can vary depending on factors like environmental interference, antenna design, and obstacles in the communication path.

**Interference**

Like other wireless communication technologies operating in the 2.4 GHz band, ESP-NOW can be susceptible to interference from other devices and Wi-Fi networks. It's essential to choose communication channels carefully to minimize interference and ensure reliable communication.

**No Network Infrastructure**

ESP-NOW is designed for point-to-point and point-to-multipoint communication but doesn't provide the infrastructure for building complex network topologies. If your application requires a network with multiple interconnected devices or Internet connectivity, you may need to complement ESP-NOW with additional networking solutions.

**Limited Data Payload**

ESP-NOW is optimized for transmitting smaller amounts of data, with a maximum payload of around 250 bytes. If your application requires high-bandwidth data transfer or large file exchanges, other services such as the [**Arduino Cloud**](https://cloud.arduino.cc/) might be more suitable.

**Security Considerations**

While ESP-NOW offers some level of data privacy, it may not be as secure as other communication protocols like HTTPS or MQTT with robust encryption. If your project involves sensitive data, consider implementing additional security measures to protect against eavesdropping and unauthorized access.

**No Message Acknowledgment**

ESP-NOW does not provide built-in acknowledgment mechanisms for confirming successful message delivery. If message reliability is crucial for your application, you'll need to implement your own acknowledgment and error-handling mechanisms.

**Compatibility**

ESP-NOW is primarily designed for use with ESP8266 and ESP32 microcontrollers. While it can be compatible with other ESP32-based devices, it may not work seamlessly with other non-ESP platforms. Compatibility should be considered when designing a system with multiple types of devices.

Arduino Cloud

Arduino provides their own service for handling wireless communication called Arduino Cloud. You can read more about it [**here**](https://docs.arduino.cc/arduino-cloud/).

While the Arduino Cloud offers similar features to the ESP-NOW protocols they are vastly different and should be used for specific use cases. Check the comparison table below for more information.

|  | **Arduino Cloud** | **ESP-NOW** |
| --- | --- | --- |
| Range | Depends on Internet connectivity, suitable for global reach. | Approx. 220 m. |
| Security | Secure communication with encryption and authentication. | Basic security, may need additional measures. |
| Delay | Network-related delays due to Internet communication. | Low latency for local communication. |
| Data Size | Unlimited data (min. Entry plan) | max 250 bytes. |
| Device Compatibility | Compatible with various Arduino boards and IoT devices. | Primarily used with ESP8266 and ESP32 microcontrollers. |
| Protocol | Uses MQTT for communication. | Uses a custom ESP-NOW protocol. |
| Power Consumption | May consume more power, especially when connected to the internet. | Known for low power consumption. |
| Data Processing | Allows for Cloud-based data processing and analytics. | Primarily for direct device-to-device communication. |
| Use Cases | IoT projects requiring global connectivity and Cloud-based data management. | Applications needing low-latency, local communication. |

Check out the different Arduino Cloud plans [**here**](https://cloud.arduino.cc/plans).

Code

Below you can find examples for sending and receiving data via the ESP-NOW protocol. You first need to identify the MAC address of your receiving board to let the sender know where to send the data. To identify the MAC address of your board upload the following code:

COPY

1// Complete Instructions to Get and Change ESP MAC Address: https://RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-arduino/

2

3*#include "WiFi.h"*

4

5void setup(){

6 Serial.begin(115200);

7 WiFi.mode(WIFI\_MODE\_STA);

8 Serial.println(WiFi.macAddress());

9}

10

11void loop(){

12

13}

**Sender Sketch**

COPY

1/\*

2 Rui Santos

3 Complete project details at https://RandomNerdTutorials.com/esp-now-esp32-arduino-ide/

4

5 Permission is hereby granted, free of charge, to any person obtaining a copy

6 of this software and associated documentation files.

7

8 The above copyright notice and this permission notice shall be included in all

9 copies or substantial portions of the Software.

10\*/

11

12*#include <esp\_now.h>*

13*#include <WiFi.h>*

14

15// REPLACE WITH YOUR RECEIVER MAC Address

16uint8\_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

17

18// Structure example to send data

19// Must match the receiver structure

20typedef struct struct\_message {

21 char a[32];

22 int b;

23 float c;

24 bool d;

25} struct\_message;

26

27// Create a struct\_message called myData

28struct\_message myData;

29

30esp\_now\_peer\_info\_t peerInfo;

31

32// callback when data is sent

33void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

34 Serial.print("\r\nLast Packet Send Status:\t");

35 Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");

36}

37

38void setup() {

39 // Init Serial Monitor

40 Serial.begin(115200);

41

42 // Set device as a Wi-Fi Station

43 WiFi.mode(WIFI\_STA);

44

45 // Init ESP-NOW

46 if (esp\_now\_init() != ESP\_OK) {

47 Serial.println("Error initializing ESP-NOW");

48 return;

49 }

50

51 // Once ESPNow is successfully Init, we will register for Send CB to

52 // get the status of Transmitted packet

53 esp\_now\_register\_send\_cb(OnDataSent);

54

55 // Register peer

56 memcpy(peerInfo.peer\_addr, broadcastAddress, 6);

57 peerInfo.channel = 0;

58 peerInfo.encrypt = false;

59

60 // Add peer

61 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

62 Serial.println("Failed to add peer");

63 return;

64 }

65}

66

67void loop() {

68 // Set values to send

69 strcpy(myData.a, "THIS IS A CHAR");

70 myData.b = random(1,20);

71 myData.c = 1.2;

72 myData.d = false;

73

74 // Send message via ESP-NOW

75 esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData));

76

77 if (result == ESP\_OK) {

78 Serial.println("Sent with success");

79 }

80 else {

81 Serial.println("Error sending the data");

82 }

83 delay(2000);

84}

**Receiver Sketch**

COPY

1/\*

2 Rui Santos

3 Complete project details at https://RandomNerdTutorials.com/esp-now-esp32-arduino-ide/

4

5 Permission is hereby granted, free of charge, to any person obtaining a copy

6 of this software and associated documentation files.

7

8 The above copyright notice and this permission notice shall be included in all

9 copies or substantial portions of the Software.

10\*/

11

12*#include <esp\_now.h>*

13*#include <WiFi.h>*

14

15// Structure example to receive data

16// Must match the sender structure

17typedef struct struct\_message {

18 char a[32];

19 int b;

20 float c;

21 bool d;

22} struct\_message;

23

24// Create a struct\_message called myData

25struct\_message myData;

26

27// callback function that will be executed when data is received

28void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

29 memcpy(&myData, incomingData, sizeof(myData));

30 Serial.print("Bytes received: ");

31 Serial.println(len);

32 Serial.print("Char: ");

33 Serial.println(myData.a);

34 Serial.print("Int: ");

35 Serial.println(myData.b);

36 Serial.print("Float: ");

37 Serial.println(myData.c);

38 Serial.print("Bool: ");

39 Serial.println(myData.d);

40 Serial.println();

41}

42

43void setup() {

44 // Initialize Serial Monitor

45 Serial.begin(115200);

46

47 // Set device as a Wi-Fi Station

48 WiFi.mode(WIFI\_STA);

49

50 // Init ESP-NOW

51 if (esp\_now\_init() != ESP\_OK) {

52 Serial.println("Error initializing ESP-NOW");

53 return;

54 }

55

56 // Once ESPNow is successfully Init, we will register for recv CB to

57 // get recv packer info

58 esp\_now\_register\_recv\_cb(OnDataRecv);

59}

60

61void loop() {

62

63}

Read More

For a more detailed explanation on how the code works check out the article at [**randomnerdtutorials.com**](https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/).

You can also read more about the

esp\_now

 library over at [**https://docs.espressif.com/**](https://docs.espressif.com/).

Summary

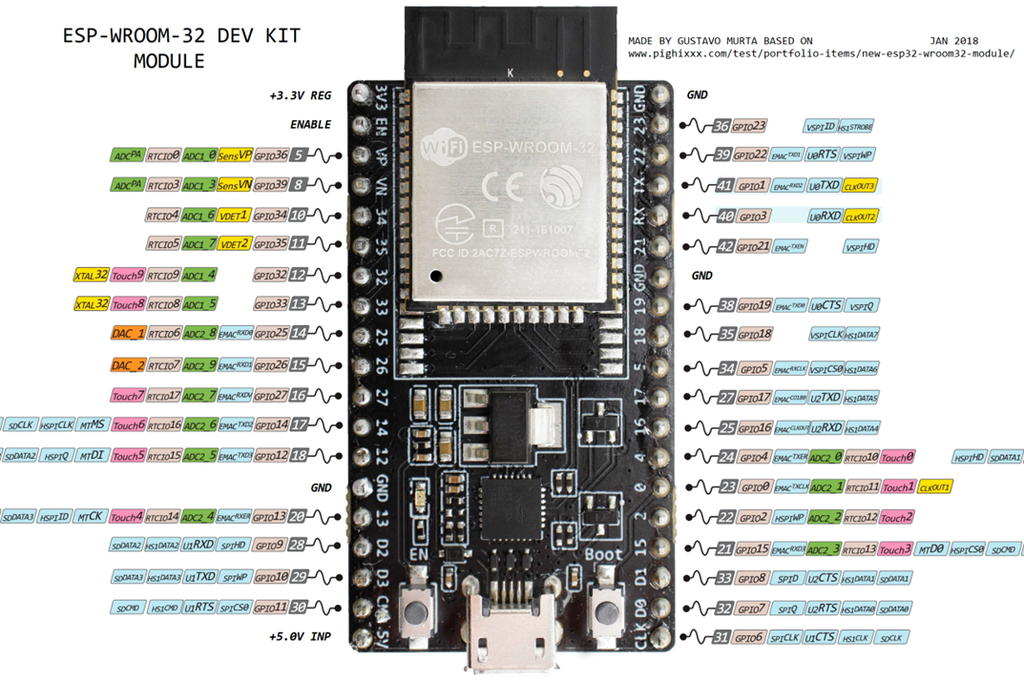
In this article we covered the basics of the ESP-NOW protocol and how it works. We also provided example code serving as a starting point for your project. Overall, ESP-NOW is a powerful communication protocol for local and low-latency applications, particularly when using ESP8266 and ESP32 microcontrollers.

However, it comes with certain limitations that you should be aware of when designing projects.

The ESP-Now is a very special, high-speed network, making it perfect for residential and industrial automation. It is another protocol developed by Espressif. We’ll be talking about this network today, which allows several devices to communicate without using a WiFi network made by a router. I'll show you an introduction to the subject and make several ESPs32 communicate through this scheme. Therefore, an ESP32 will read the pins and transmit their values, while the other devices will receive these values and change the output of the pins according to those numbers.

This network is reliable and is 2.4GHz. This network also is with the same frequency and channels as your WiFi router. The highlight, however, is that it goes away from WiFi, as it is instant.

**Step 1: ESP32 Pinout**

[](https://content.instructables.com/FKZ/5Y5T/JFIIR5XE/FKZ5Y5TJFIIR5XE.png?auto=webp&frame=1&width=1024&fit=bounds&md=7270061a2186ef65002fa222a0b0b5ac)

I show here the pinning of the ESP32.

**Step 2: About ESP-Now**

• Communication protocol created by Espressif.

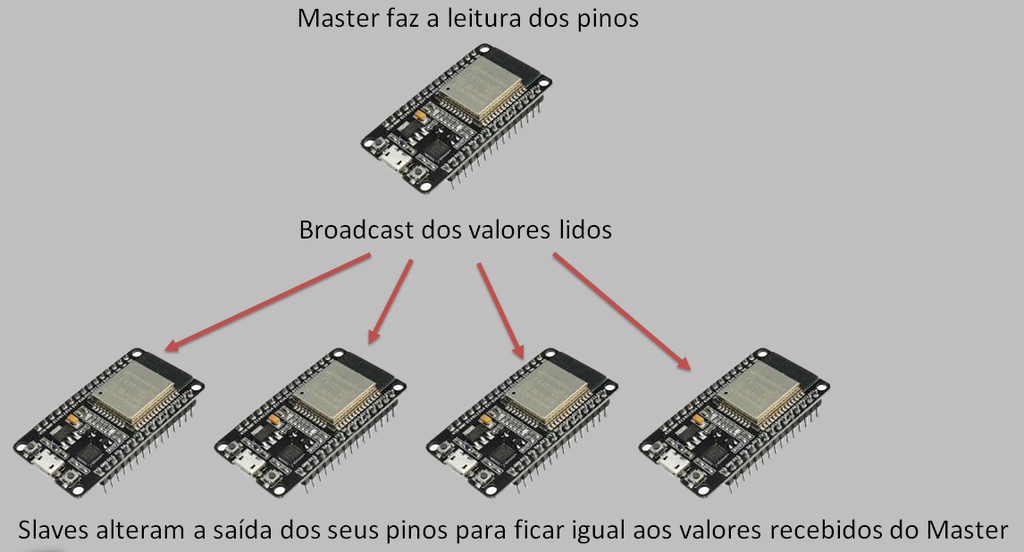
• You don’t need a WiFi network.

• Similar to the low-power protocol used on a 2.4GHz wireless mouse.

• Initial pairing required.

• After pairing, the connection is persistent peer-to-peer.

**Step 3: Tutorial Example**

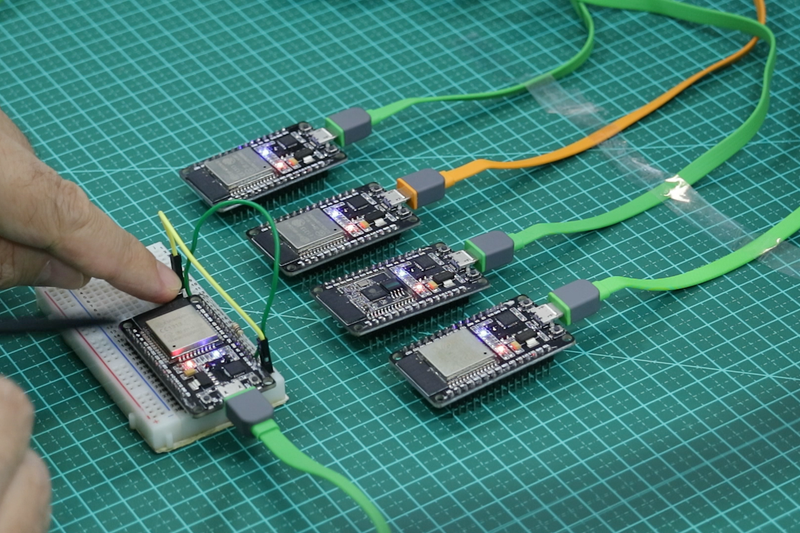
[](https://content.instructables.com/FDL/XGOC/JFIIR5Y8/FDLXGOCJFIIR5Y8.png?auto=webp&frame=1&width=1024&fit=bounds&md=a0bf7f45807b84559dd02197b499dc08)

1. Master reads pins

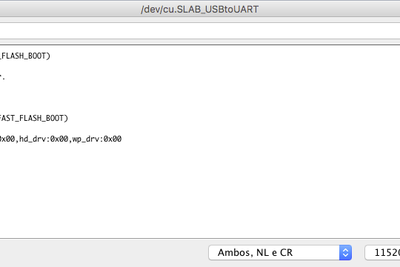
2. Broadcast of the values read

3. Slaves change the output of their pins to equal the values received from the Master

**Step 4: Demonstration**

[](https://content.instructables.com/FFA/HPIA/JFIIR5YV/FFAHPIAJFIIR5YV.png?auto=webp&frame=1&width=1024&height=1024&fit=bounds&md=a38af58cf88b63030fe9ae7043af7033)

[](https://content.instructables.com/F6J/CZK9/JFIIR5YW/F6JCZK9JFIIR5YW.png?auto=webp&frame=1&fit=bounds&md=21cc8141f10731a1a038c30338d4b277)

[](https://content.instructables.com/FLI/HYQ4/JFIIR5YX/FLIHYQ4JFIIR5YX.png?auto=webp&frame=1&width=1024&fit=bounds&md=f44bfadc4c8639b8c7b5f7b13a7a04e8)

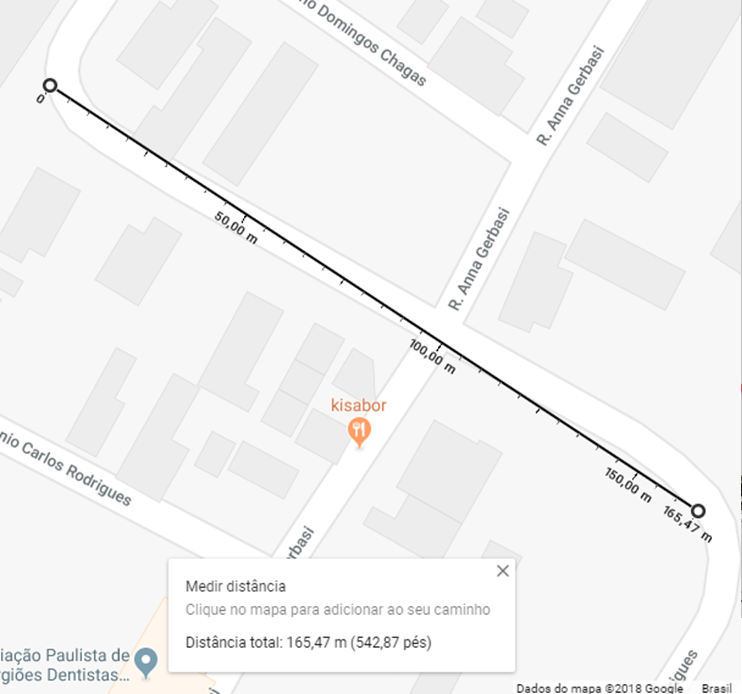
In our assembly, we have an ESP32 isolated, which is configured as Master. It is important to remember that there is no Master device, and that they all function as Stations. However, to facilitate the identification, I point to this first ESP as Master, in which I set up a button in the GPIO02.

When this button is pressed, a LED lights up in this microcontroller, and all four other ESPs32 instantly restart the action. Why does this occur? Because the moment the Master sends the information to the station, it is sending a MAC address of Broadcast, which means that everyone on the network receives the data at the same time.

In this example, I compiled the same receiver for all microcontrollers. I copied the code that sends to the Master, and this one also sends the Broadcast to the others. In this scheme, it is still possible to see that we practically don’t have Boot time, because when the ESP is switched off and reconnected, the operation resumes immediately.

In the serial print of Setup, both the sending and the receiving code contain the MAC address values of each of the chips involved. See the example in the pictures.

**Step 5: Distance**

[](https://content.instructables.com/F6X/P67Z/JFIIR602/F6XP67ZJFIIR602.png?auto=webp&frame=1&fit=bounds&md=25d04f2aecb896467b4b74b1bab2996a)

I will show here that in the tests we did with the ESPs32, they managed to communicate up to 165.47 meters in a straight line, using only the internal antennas of the devices. See on the map.

**Step 6: Code - ​ESPNowMaster.ino**

Here, we will work with the libraries ESP\_NOW.h and WiFi.h. Let's define the channel for connection and the pins that will be read, as well as the data that will be sent to Slaves. It is important to highlight that the source code of Slaves has this same array with the same GPIOs. In the Setup part, we will calculate the number of pins and put in this variable, so we won’t need to change every time we change the number of pins. We still work with the MAC Address of the slaves to which we will send the reading.

//Libs for espnow and wifi

#include <esp\_now.h>

#include <WiFi.h>

//Channel used in the connection

#define CHANNEL 1

//Gpios that we are going to read (digitalRead) and send to the Slaves

//It's important that the Slave source code has this same array

//with the same gpios in the same order

uint8\_t gpios[] = {23, 2};

//In the setup function we'll calculate the gpio count and put in this variable,

//so we don't need to change this variable everytime we change

//the gpios array total size, everything will be calculated automatically

//on setup function

int gpioCount;

//Slaves Mac Addresses that will receive data from the Master

//If you want to send data to all Slaves, use only the broadcast address {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}

//If you want to send data to specific Slaves, put their Mac Addresses separeted with comma (use WiFi.macAddress())

//to find out the Mac Address of the ESPs while in STATION MODE)

uint8\_t macSlaves[][6] = {

//To send to specific Slaves

//{0x24, 0x0A, 0xC4, 0x0E, 0x3F, 0xD1}, {0x24, 0x0A, 0xC4, 0x0E, 0x4E, 0xC3}

//Or to send to all Slaves

{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}

};

**Step 7: ESPNowMaster.ino - Setup**

In this part, we will deal with the Master, remembering that we have a Master and four Slaves. Let's calculate the array size of GPIOs that will be read with digitalRead. We will still put ESP in station mode, and display on the Serial Monitor of the MAC Address of this ESP when in station mode, and we’ll call the function that initializes ESPNow.

void setup() {  
 Serial.begin(115200);

//Calculation of gpio array size:

//sizeof(gpios) returns how many bytes "gpios" array points to.

//Elements in this array are of type uint8\_t.

//sizeof(uint8\_t) return how many bytes uint8\_t type has.

//Therefore if we want to know how many gpios there are,

//we divide the total byte count of the array by how many bytes

//each element has.

gpioCount = sizeof(gpios)/sizeof(uint8\_t);

//Puts ESP in STATION MODE

WiFi.mode(WIFI\_STA);

//Shows on the Serial Monitor the STATION MODE Mac Address of this ESP

Serial.print("Mac Address in Station: ");

Serial.println(WiFi.macAddress());

//Calls the function that will initialize the ESP-NOW protocol

InitESPNow();

We also calculate the size of the array with the MAC addresses of the slaves and create a variable that will save the information of each slave. We inform the channel and add the slave.

//Calculation of the size of the slaves array:  
 //sizeof(macSlaves) returns how many bytes the macSlaves array points to.

//Each Slave Mac Address is an array with 6 elements.

//If each element is sizeof(uint8\_t) bytes

//then the total of slaves is the division of the total amount of bytes

//by how many elements each MAc Address has

//by how much bytes each element has.

int slavesCount = sizeof(macSlaves)/6/sizeof(uint8\_t);

//For each Slave

for(int i=0; i

In this step, we record the callback that will inform us about the status of the shipment. The function that will be executed is OnDataSent and is declared below. We put the slave in read mode and call the send function, which deals with sending.

//Registers the callback that will give us feedback about the sent data  
 //The function that will be executed is called OnDataSent

esp\_now\_register\_send\_cb(OnDataSent);

//For each gpio

for(int i=0; i

//Calls the send function

send();

}

**Step 8: ESPNowMaster.ino - InitESPNow**

The InitESPNow function is simple and works here with the possibilities of successful boot, as well as possibilities of error during boot time.

void InitESPNow() {  
 //If the initialization was successful

if (esp\_now\_init() == ESP\_OK) {

Serial.println("ESPNow Init Success");

}

//If there was an error

else {

Serial.println("ESPNow Init Failed");

ESP.restart();

}

}

**Step 9: ESPNowMaster.ino - Send**

Here, we have the function that will read the pins that are in the array GPIOs and send the read values to the other ESPs. This array will store the read values.

//Function that will read the gpios and send  
//the read values to the others ESPs

void send(){

//Array that will store the read values

uint8\_t values[gpioCount];

//For each gpio

for(int i=0; i<gpioCount; i++){

//Reads the value (HIGH or LOW) of the gpio

//and stores the value on the array

values[i] = digitalRead(gpios[i]);

}

The broadcast address will send the information to all ESPs. If you want the information to go to specific ESPs, you should call the esp\_now\_send function for each MAC Address, passing the information that the MAC Address is the first parameter in place of the Broadcast.

//In this example we are going to use the broadcast address {0xFF, 0xFF,0xFF,0xFF,0xFF,0xFF}  
 //to send the values to all Slaves.

//If you want to send to a specific Slave, you have to put its Mac Address on macAddr.

//If you want to send to more then one specific Slave you will need to create

//a "for loop" and call esp\_now\_send for each mac address on the macSlaves array

uint8\_t macAddr[] = {0xFF, 0xFF,0xFF,0xFF,0xFF,0xFF};

esp\_err\_t result = esp\_now\_send(macAddr, (uint8\_t\*) &values, sizeof(values));

Serial.print("Send Status: ");

//If it was successful

if (result == ESP\_OK) {

Serial.println("Success");

}

//if it failed

else {

Serial.println("Error");

}

}

**Step 10: ESPNowMaster.ino - OnDataSent**

Let's now define the function that serves as a callback to let us know about the sending situation that we make. We copy the destination MAC address to a string and show the MAC address that served as the destination of the message. We've also shown whether the status of the upload was successful or not.

//Callback function that gives us feedback about the sent data  
void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

char macStr[18];

//Copies the receiver Mac Address to a string

snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",

mac\_addr[0], mac\_addr[1], mac\_addr[2], mac\_addr[3], mac\_addr[4], mac\_addr[5]);

//Prints it on Serial Monitor

Serial.print("Sent to: ");

Serial.println(macStr);

//Prints if it was successful or not

Serial.print("Status: ");

Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Success" : "Fail");

//Sends again

send();

}

**Step 11: SPNowMaster.ino - Loop**

We don’t have to do anything in the Loop, because whenever we receive feedback from the Send via OnDataSent function, we send the data again. This causes the data to always be sent in sequence.

//We don't do anything on the loop.  
//Every time we receive feedback about the last sent data,

//we'll be calling the send function again,

//therefore the data is always being sent

void loop() {

}

**Step 12: ESPNowSlave.ino**

Now we're off to the Slaves. We work with the libraries ESP\_NOW.h and WiFi.h. The pins that we will write (digitalWrite) will have the values received from the Master. Remember that it is important that the source code of the Master has this same array with the same GPIOs in the same order. In the setup, we will calculate the amount of pins and put in this variable. This way, we don’t have to change here every time we change the number of pins, because everything is calculated in the setup.

//Libs for espnow e wifi

#include <esp\_now.h>

#include <WiFi.h>

//Gpios we'll write the values received from the Master  
//It's important that the Master source code has this same array

//with the same gpios in the same order

uint8\_t gpios[] = {23, 2};

//In the setup function we'll calculate the gpio count and put in this variable,

//so we don't need to change this variable everytime we change

//the gpios array total size, everything will be calculated automatically

//on setup function

int gpioCount;

**Step 13: ESPNowSlave.ino - Setup**

We have in this step the calculation of the array size of GPIOs. We put the ESP in station mode, and we show on the Serial Monitor the Mac Address of this ESP when in station mode. If you want the Master to send to ESPs in specific, make changes in the array of slaves (in the source code of the Master) so that it has only the Mac Addresses printed there. Finally, we call the function that initializes ESPNow.

void setup() {  
 Serial.begin(115200);

//Calculation of gpio array size:

//sizeof(gpios) returns how many bytes "gpios" array points to.

//Elements in this array are of type uint8\_t.

//sizeof(uint8\_t) return how many bytes uint8\_t type has.

//Therefore if we want to know how many gpios there are,

//we divide the total byte count of the array by how many bytes

//each element has.

gpioCount = sizeof(gpios)/sizeof(uint8\_t);

//Puts ESP in STATION MODE

WiFi.mode(WIFI\_STA);

//Shows on the Serial Monitor the STATION MODE Mac Address of this ESP

Serial.print("Mac Address in Station: ");

Serial.println(WiFi.macAddress());

//Calls the function that will initialize the ESP-NOW protocol

InitESPNow();

We register the callback that will inform us when the Master sent something. The function that will be executed is OnDataRecv and is declared below. We put it in output mode.

//Registers the callback function that will be executed when   
 //this Slave receives data from the Master.

//The function in this case is called OnDataRecv

esp\_now\_register\_recv\_cb(OnDataRecv);

//For each gpio on gpios array

for(int i=0; i

**Step 14: ESPNowSlave.ino - InitESPNow**

Once again we work with the InitESPNow function, this time on the Slave. Again, if all is correct, it will print "Success." Otherwise, it will print "Error.”

void InitESPNow() {  
 //If the initialization was successful

if (esp\_now\_init() == ESP\_OK) {

Serial.println("ESPNow Init Success");

}

//If there was an error

else {

Serial.println("ESPNow Init Failed");

ESP.restart();

}

}

**Step 15: ESPNowSlave.ino - OnDataRecv**

We also resumed the function that serves as a callback there at Slave to let us know when something came from the Master. We copy the source MAC address to a string and show the MAC address that was the origin of the message. For each pin, we put the value received from the Master in the respective output.

//Callback function that tells us when data from Master is received  
void OnDataRecv(const uint8\_t \*mac\_addr, const uint8\_t \*data, int data\_len) {

char macStr[18];

//Copies the sender Mac Address to a string

snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",

mac\_addr[0], mac\_addr[1], mac\_addr[2], mac\_addr[3], mac\_addr[4], mac\_addr[5]);

//Prints it on Serial Monitor

Serial.print("Received from: ");

Serial.println(macStr);

Serial.println("");

//For each gpio

for(int i=0; i<gpioCount; i++){

//Sets its output to match the received value

digitalWrite(gpios[i], data[i]);

}

}

**Step 16: ESPNowSlave.ino - Loop**

Here, we don’t have to do anything on the Loop. Whenever something comes from the Master, the OnDataRecv function runs automatically, since we added it as callback using the esp\_now\_register\_recv\_cb function.

//We don't do anything on the loop.  
//Everytime something comes from Master

//the OnDataRecv function is executed automatically

//because we added it as callback using esp\_now\_register\_recv\_cb

void loop() {

}

**Getting Started with ESP-NOW (ESP8266 NodeMCU with Arduino IDE)**

In this article, we’ll show you how you can use ESP-NOW to exchange data between ESP8266 NodeMCU boards programmed with Arduino IDE. ESP-NOW is a connectionless communication protocol developed by Espressif that features short packet transmission and can be used with ESP8266 and [ESP32 boards](https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/).



We have other tutorials about ESP-NOW with the ESP8266:

* [ESP-NOW Two-Way Communication Between ESP8266 NodeMCU Boards](https://randomnerdtutorials.com/esp-now-two-way-communication-esp8266-nodemcu/)
* [ESP-NOW with ESP8266: Send Data to Multiple Boards (one-to-many)](https://randomnerdtutorials.com/esp-now-one-to-many-esp8266-nodemcu/)
* [ESP-NOW with ESP8266: Receive Data from Multiple Boards (many-to-one)](https://randomnerdtutorials.com/esp-now-many-to-one-esp8266-nodemcu/)

**Arduino IDE**

We’ll program the [ESP8266 NodeMCU board](https://makeradvisor.com/tools/esp8266-esp-12e-nodemcu-wi-fi-development-board/) using Arduino IDE, so before proceeding with this tutorial you should have the ESP8266 add-on installed in your Arduino IDE. Follow the next guide:

* [Installing ESP8266 Board in Arduino IDE (Windows, Mac OS X, Linux)](https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/)

**Note:**we have a similar guide for the ESP32: [Getting Started with ESP-NOW (ESP32 with Arduino IDE)](https://randomnerdtutorials.com/esp-now-esp32-arduino-ide/).

**Introducing ESP-NOW**

The following video shows an introduction to ESP-NOW. This video was recorded for the ESP32, but most concepts also apply to the ESP8266 NodeMCU board.

Stating Espressif website, ESP-NOW is a “*protocol developed by Espressif, which enables multiple devices to communicate with one another without using Wi-Fi. The protocol is similar to the low-power 2.4GHz wireless connectivity (…) . The pairing between devices is needed prior to their communication. After the pairing is done, the connection is safe and peer-to-peer, with no handshake being required*.”



This means that after pairing a device with each other, the connection is persistent. In other words, if suddenly one of your boards loses power or resets, when it restarts, it will automatically connect to its peer to continue the communication.

ESP-NOW supports the following features:

* Encrypted and unencrypted unicast communication;
* Mixed encrypted and unencrypted peer devices;
* **Up to 250-byte** payload can be carried;
* Sending callback function that can be set to inform the application layer of  
  transmission success or failure.

ESP-NOW technology also has the following limitations:

* Limited encrypted peers. 10 encrypted peers at the most are supported in Station mode; 6 at the most in SoftAP or SoftAP + Station mode;
* Multiple unencrypted peers are supported, however, their total number should be less than 20, including encrypted peers;
* **Payload is limited to 250 bytes**.

**In simple words, ESP-NOW is a fast communication protocol that can be used to exchange small messages (up to 250 bytes) between ESP8266 boards.**

ESP-NOW is very versatile and you can have one-way or two-way communication in different setups.

**ESP-NOW One-Way Communication**

For example, in one-way communication, you can have scenarios like this:

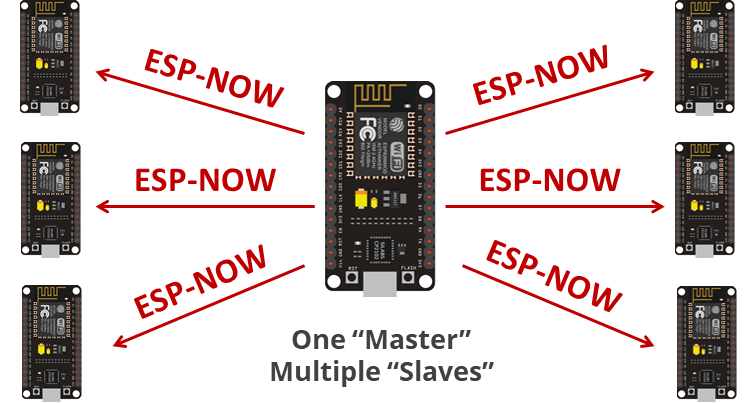
* **One ESP8266 board sending data to another ESP8266 board**



This configuration is very easy to implement and it is great to send data from one board to the other like sensor readings or ON and OFF commands to control GPIOs.

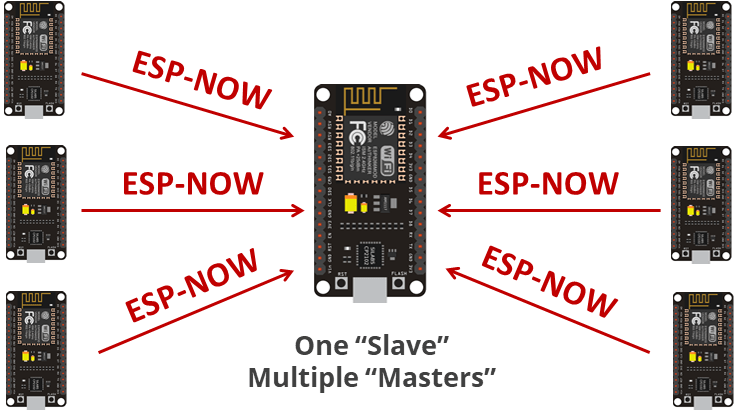
* **A “master” ESP8266 sending data to multiple ESP8266 “slaves”**

One ESP8266 board sending the same or different commands to different ESP8266 boards. This configuration is ideal to build something like a remote control. You can have several ESP8266 boards around the house that are controlled by one main ESP8266 board.



* **One ESP8266 “slave” receiving data from multiple “masters”**

This configuration is ideal if you want to collect data from several sensors nodes into one ESP8266 board. This can be configured as a web server to display data from all the other boards, for example.



**Note:**in the ESP-NOW documentation there isn’t such thing as “sender/master” and “receiver/slave”. Every board can be a sender or receiver. However, to keep things clear we’ll use the terms “sender” and “receiver” or “master” and “slave”.

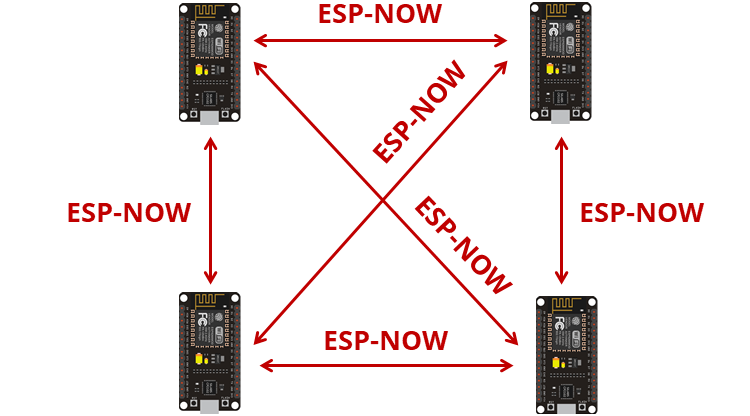
**ESP-NOW Two-Way Communication**

With ESP-NOW, each board can be a sender and a receiver at the same time. So, you can establish a two-way communication between boards.

For example, you can have two boards communicating with each other.



You can add more boards to this configuration and have something that looks like a network (all ESP8266 boards communicate with each other).



In summary, ESP-NOW is ideal to build a network in which you can have several ESP8266 boards exchanging data with each other.

**ESP8266: Getting Board MAC Address**

To communicate via ESP-NOW, you need to know the MAC Address of the ESP8266 receiver. That’s how you know to which device you’ll send the information to.

Each ESP8266 has a [unique MAC Address](https://randomnerdtutorials.com/get-change-esp32-esp8266-mac-address-arduino/) and that’s how we identify each board to send data to it using ESP-NOW (learn how to [Get and Change the ESP8266 MAC Address](https://randomnerdtutorials.com/get-change-esp32-esp8266-mac-address-arduino/)).

To get your board’s MAC Address, upload the following code.

// Complete Instructions to Get and Change ESP MAC Address: https://RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-arduino/

#include <ESP8266WiFi.h>

void setup(){

Serial.begin(115200);

Serial.println();

Serial.print("ESP8266 Board MAC Address: ");

Serial.println(WiFi.macAddress());

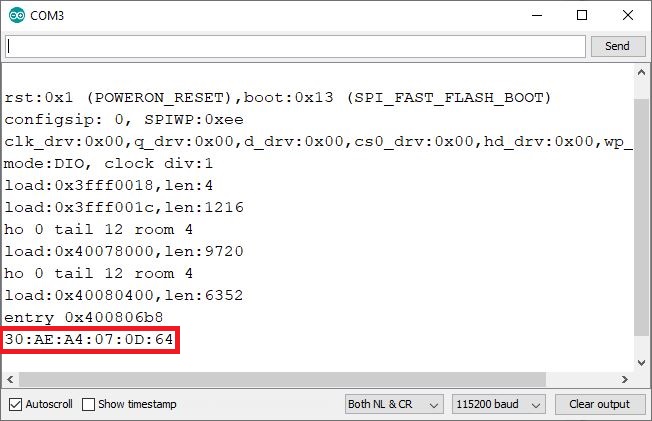
}

void loop(){

}

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP8266/ESP8266_Get_MAC_Address.ino)

After uploading the code, open the Serial Monitor at a baud rate of 115200 and press the ESP8266 RESET button. The MAC address should be printed as follows:



Save your board MAC address because you’ll need it to send data to the right board via ESP-NOW.

**ESP-NOW One-way Point to Point Communication with ESP8266**

To get you started with ESP-NOW wireless communication, we’ll build a simple project that shows how to send a message from one ESP8266 to another. One ESP8266 will act as a “sender” and the other ESP8266 will be the “receiver”.



We’ll send a structure that contains a variable of type *char*, *int*, *float*, *String*and *boolean*. Then, you can modify the structure to send whichever variable types are suitable for your project (like sensor readings, or boolean variables to turn something on or off).

For better understanding we’ll call “sender” to ESP8266 #1 and “receiver” to ESP8266 #2.

Here’s what we should include in the **sender** sketch:

1. Initialize ESP-NOW;
2. Register a callback function upon sending data – the OnDataSent function will be executed when a message is sent. This can tell us if the message was successfully delivered or not;
3. Add a peer device (the receiver). For this, you need to know the the receiver MAC address;
4. Send a message to the peer device.

On the **receiver**side, the sketch should include:

1. Initialize ESP-NOW;
2. Register for a receive callback function (OnDataRecv). This is a function that will be executed when a message is received.
3. Inside that callback function save the message into a variable to execute any task with that information.

ESP-NOW works with callback functions that are called when a device receives a message or when a message is sent (you get if the message was successfully delivered or if it failed).

**ESP-NOW Useful Functions**

Here’s a summary of most essential ESP-NOW functions:

|  |
| --- |
| **Function Name and Description** |
| esp\_now\_init() Initializes ESP-NOW. You must initialize Wi-Fi before initializing ESP-NOW. Returns 0, if succeed. |
| esp\_now\_set\_self\_role(role) the role can be: ESP\_NOW\_ROLE\_IDLE = 0, ESP\_NOW\_ROLE\_CONTROLLER, ESP\_NOW\_ROLE\_SLAVE, ESP\_NOW\_ROLE\_COMBO, ESP\_NOW\_ROLE\_MAX |
| esp\_now\_add\_peer(uint8 mac\_addr, uint8 role, uint8 channel, uint8 key, uint8 key\_len) Call this function to pair a device. |
| esp\_now\_send(uint8 mac\_address, uint8 data, int len) Send data with ESP-NOW. |
| esp\_now\_register\_send\_cb() Register a callback function that is triggered upon sending data. When a message is sent, a function is called – this function returns whether the delivery was successful or not. |
| esp\_now\_register\_rcv\_cb() Register a callback function that is triggered upon receiving data. When data is received via ESP-NOW, a function is called. |

For more information about these functions:

* [ESP-NOW documentation at Read the Docs](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html)
* [ESP8266 API reference from Espressif](https://www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf)

**ESP8266 NodeMCU Sender Sketch (ESP-NOW)**

Here’s the code for the ESP8266 NodeMCU Sender board. Copy the code to your Arduino IDE, but don’t upload it yet. You need to make a few modifications to make it work for you.

/\*

Rui Santos

Complete project details at https://RandomNerdTutorials.com/esp-now-esp8266-nodemcu-arduino-ide/

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

\*/

#include <ESP8266WiFi.h>

#include <espnow.h>

// REPLACE WITH RECEIVER MAC Address

uint8\_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// Structure example to send data

// Must match the receiver structure

typedef struct struct\_message {

char a[32];

int b;

float c;

String d;

bool e;

} struct\_message;

// Create a struct\_message called myData

struct\_message myData;

unsigned long lastTime = 0;

unsigned long timerDelay = 2000; // send readings timer

// Callback when data is sent

void OnDataSent(uint8\_t \*mac\_addr, uint8\_t sendStatus) {

Serial.print("Last Packet Send Status: ");

if (sendStatus == 0){

Serial.println("Delivery success");

}

else{

Serial.println("Delivery fail");

}

}

void setup() {

// Init Serial Monitor

Serial.begin(115200);

// Set device as a Wi-Fi Station

WiFi.mode(WIFI\_STA);

// Init ESP-NOW

if (esp\_now\_init() != 0) {

Serial.println("Error initializing ESP-NOW");

return;

}

// Once ESPNow is successfully Init, we will register for Send CB to

// get the status of Trasnmitted packet

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_CONTROLLER);

esp\_now\_register\_send\_cb(OnDataSent);

// Register peer

esp\_now\_add\_peer(broadcastAddress, ESP\_NOW\_ROLE\_SLAVE, 1, NULL, 0);

}

void loop() {

if ((millis() - lastTime) > timerDelay) {

// Set values to send

strcpy(myData.a, "THIS IS A CHAR");

myData.b = random(1,20);

myData.c = 1.2;

myData.d = "Hello";

myData.e = false;

// Send message via ESP-NOW

esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData));

lastTime = millis();

}

}

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP8266/ESP_NOW/ESP_NOW_Sender.ino)

**How the code works**

First, include the ESP8266WiFi.h and espnow.h libraries.

#include <ESP8266WiFi.h>

#include <espnow.h>

In the next line, you should insert the ESP8266 receiver MAC address.

uint8\_t broadcastAddress[] = {0x5C, 0xCF, 0x7F, 0x99, 0x9A, 0xEA};

In our case, the receiver MAC address is: 5C:CF:7F:99:9A:EA, but you need to replace that variable with your own MAC address.

Then, create a structure that contains the type of data we want to send. We called this structure struct\_message and it contains 5 different variable types. You can change this to send whatever variable types you want.

typedef struct struct\_message {

char a[32];

int b;

float c;

String d;

bool e;

} struct\_message;

Create a new variable of type struct\_message that is called myData that will store the variables values.

struct\_message myData;

Next, define the OnDataSent() function. This is a callback function that will be executed when a message is sent. In this case, this message simply prints if the message was successfully sent or not.

void OnDataSent(uint8\_t \*mac\_addr, uint8\_t sendStatus) {

Serial.print("Last Packet Send Status: ");

if (sendStatus == 0){

Serial.println("Delivery success");

}

else{

Serial.println("Delivery fail");

}

}

In the setup(), initialize the serial monitor for debugging purposes:

Serial.begin(115200);

Set the device as a Wi-Fi station:

WiFi.mode(WIFI\_STA);

Initialize ESP-NOW:

if (esp\_now\_init() != 0) {

Serial.println("Error initializing ESP-NOW");

return;

}

Set the ESP8266 role:

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_CONTROLLER);

It accepts the following roles: ESP\_NOW\_ROLE\_CONTROLLER, ESP\_NOW\_ROLE\_SLAVE, ESP\_NOW\_ROLE\_COMBO, ESP\_NOW\_ROLE\_MAX.

After successfully initializing ESP-NOW, register the callback function that will be called when a message is sent. In this case, we register for the OnDataSent() function created previously.

esp\_now\_register\_send\_cb(OnDataSent);

Then, pair with another ESP-NOW device to send data:

esp\_now\_add\_peer(broadcastAddress, ESP\_NOW\_ROLE\_SLAVE, 1, NULL, 0);

The esp\_now\_add\_peer accepts the following arguments, in this order: mac address, role, wi-fi channel, key, and key length.

In the loop(), we’ll send a message via ESP-NOW every 2 seconds (you can change this delay time on the timerDelay variable).

First, we set the variables values as follows:

strcpy(myData.a, "THIS IS A CHAR");

myData.b = random(1,20);

myData.c = 1.2;

myData.d = "Hello";

myData.e = false;

Remember that myData is a structure. Here we assign the values we want to send inside the structure. For example, the first line assigns a char, the second line assigns a random Int number, a Float, a String and a Boolean variable.

We create this kind of structure to show you how to send the most common variable types. You can change the structure to send any other type of data.

Finally, send the message as follows:

esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData));

The loop() is executed every 2000 milliseconds (2 seconds).

if ((millis() - lastTime) > timerDelay) {

// Set values to send

strcpy(myData.a, "THIS IS A CHAR");

myData.b = random(1,20);

myData.c = 1.2;

myData.d = "Hello";

myData.e = false;

// Send message via ESP-NOW

esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData));

lastTime = millis();

}

**ESP8266 NodeMCU Receiver Sketch (ESP-NOW)**

Upload the following code to your [ESP8266 NodeMCU receiver board](https://makeradvisor.com/tools/esp8266-esp-12e-nodemcu-wi-fi-development-board/).

/\*

Rui Santos

Complete project details at https://RandomNerdTutorials.com/esp-now-esp8266-nodemcu-arduino-ide/

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files.

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

\*/

#include <ESP8266WiFi.h>

#include <espnow.h>

// Structure example to receive data

// Must match the sender structure

typedef struct struct\_message {

char a[32];

int b;

float c;

String d;

bool e;

} struct\_message;

// Create a struct\_message called myData

struct\_message myData;

// Callback function that will be executed when data is received

void OnDataRecv(uint8\_t \* mac, uint8\_t \*incomingData, uint8\_t len) {

memcpy(&myData, incomingData, sizeof(myData));

Serial.print("Bytes received: ");

Serial.println(len);

Serial.print("Char: ");

Serial.println(myData.a);

Serial.print("Int: ");

Serial.println(myData.b);

Serial.print("Float: ");

Serial.println(myData.c);

Serial.print("String: ");

Serial.println(myData.d);

Serial.print("Bool: ");

Serial.println(myData.e);

Serial.println();

}

void setup() {

// Initialize Serial Monitor

Serial.begin(115200);

// Set device as a Wi-Fi Station

WiFi.mode(WIFI\_STA);

// Init ESP-NOW

if (esp\_now\_init() != 0) {

Serial.println("Error initializing ESP-NOW");

return;

}

// Once ESPNow is successfully Init, we will register for recv CB to

// get recv packer info

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_SLAVE);

esp\_now\_register\_recv\_cb(OnDataRecv);

}

void loop() {

}

[View raw code](https://github.com/RuiSantosdotme/Random-Nerd-Tutorials/raw/master/Projects/ESP8266/ESP_NOW/ESP_NOW_Receiver.ino)

**How the code works**

Similarly to the sender, start by including the libraries:

#include <esp\_now.h>

#include <WiFi.h>

Create a structure to receive the data. This structure should be the same defined in the sender sketch.

typedef struct struct\_message {

char a[32];

int b;

float c;

String d;

bool e;

} struct\_message;

Create a struct\_message variable called myData.

struct\_message myData;

Create a callback function that will be called when the ESP8266 receives the data via ESP-NOW. The function is called onDataRecv() and should accept several parameters as follows:

void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

We copy the content of the incomingData data variable into the myData variable.

memcpy(&myData, incomingData, sizeof(myData));

Now, the myData structure contains several variables inside with the values sent by the sender ESP8266. To access variable a, for example, we just need to call myData.a.

In this example, we simply print the received data, but in a practical application you can print the data on a display, for example.

Serial.print("Bytes received: ");

Serial.println(len);

Serial.print("Char: ");

Serial.println(myData.a);

Serial.print("Int: ");

Serial.println(myData.b);

Serial.print("Float: ");

Serial.println(myData.c);

Serial.print("String: ");

Serial.println(myData.d);

Serial.print("Bool: ");

Serial.println(myData.e);

Serial.println();

}

In the setup(), intialize Serial communication for debugging purposes.

Serial.begin(115200);

Set the device as a Wi-Fi Station.

WiFi.mode(WIFI\_STA);

Initialize ESP-NOW:

if (esp\_now\_init() != ESP\_OK) {

Serial.println("Error initializing ESP-NOW");

return;

}

Set the ESP8266 role:

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_SLAVE);

Register for a callback function that will be called when data is received. In this case, we register for the OnDataRecv() function that was created previously.

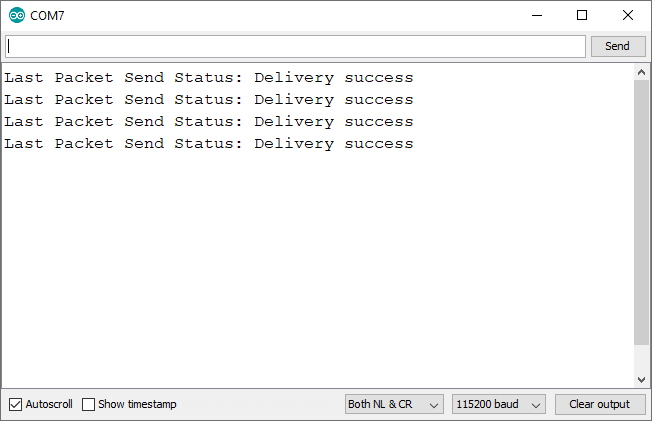
esp\_now\_register\_recv\_cb(OnDataRecv);

**Testing ESP-NOW Communication**

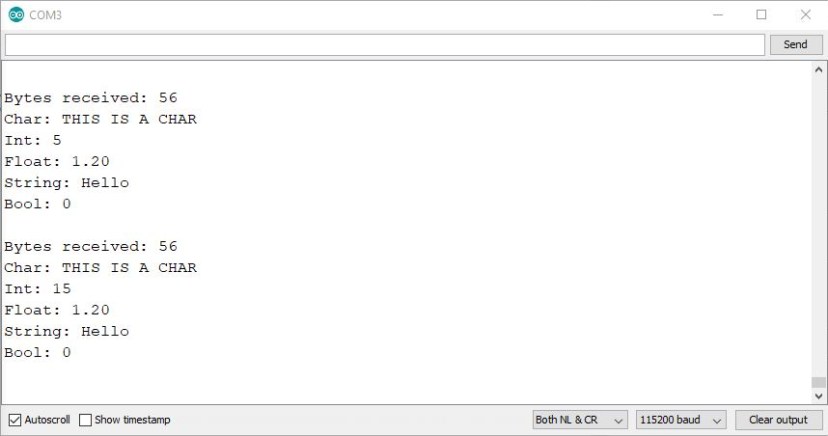
Upload the sender sketch to one board and the receiver sketch to the other board. Don’t forget to insert the receiver MAC address on the sender sketch.

Now, open two Arduino IDE windows. One for the receiver, and another for the sender. Open the Serial Monitor for each board. It should be a different COM port for each board.

This is what you should get on the sender side.



And this is what you should get on the receiver side. Note that the Int variable changes between each reading received (because we set it to a random number in the sender side).



We tested the communication range between the two boards, and we are able to get a stable communication up to 140 meters (approximately 459 feet) in open field. In this experiment both ESP8266 on-board antennas were pointing to each other.



**Wrapping Up**

In this tutorial you’ve learned how to use ESP-NOW with the ESP8266 NodeMCU board. Now, you can combine the sender and receiver sketch so that you have a two-way communication (each board acting as a server and sender at the same time). You can also use more boards to have a communication between multiple boards.

**Getting Started with ESP-NOW**

Will guide you to communicate between ESP32 controllers by using ESP NOW protocol.

[Beginner](https://www.hackster.io/projects?difficulty=beginner)Full instructions provided1,773



**Things used in this project**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Hardware components** | | | | | |
| ESP32 | |  | | --- | | [Espressif ESP32](https://www.hackster.io/Espressif/products/esp32?ref=project-d7d8b2) | |  | | × | 2 |  |  |
| **Software apps and online services** | | | | | |
| Arduino IDE | |  | | --- | | [Arduino IDE](https://www.hackster.io/arduino/products/arduino-ide?ref=project-d7d8b2) | |  | |  | |  |  |

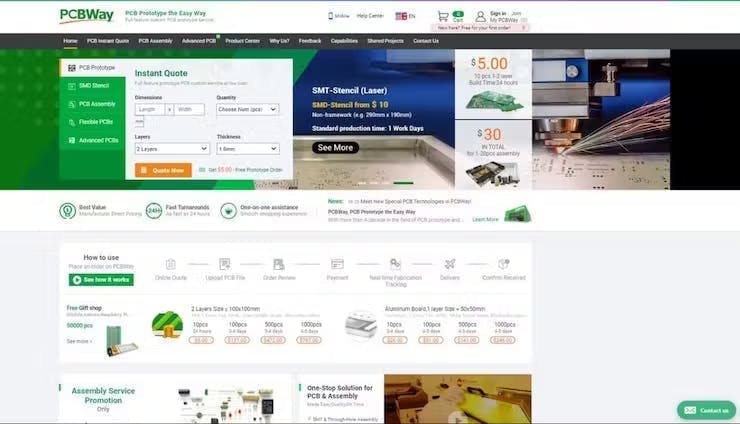
**Story**

ESP-NOW is a wireless communication protocol based on the data-link layer that enables the direct, quick, and low-power control of smart devices without the need for a router. Espressif defines it and can work with Wi-Fi and Bluetooth LE. ESP-NOW provides flexible and low-power data transmission to all interconnected devices. It can also be used as an independent protocol that helps with device provisioning, debugging, and firmware upgrades.



ESP-NOW is a connectionless communication protocol developed by Espressif that features short packet transmission. This protocol enables multiple devices to talk to each other in an easy way. It is a fast communication protocol that can be used to exchange small messages (up to 250 bytes) between ESP32 or ESP8266 boards. ESP-NOW supports the following features: Encrypted and unencrypted unicast communication; Mixed encrypted and unencrypted peer devices; Up to 250-byte payload can be carried; Sending callback function that can be set to inform the application layer of transmission success or failure.

**Get PCBs for Your Projects Manufactured**



You must check out [PCBWAY](https://www.pcbway.com/) for ordering PCBs online for cheap!

You get 10 good-quality PCBs manufactured and shipped to your doorstep for cheap. You will also get a discount on shipping on your first order. Upload your Gerber files onto [PCBWAY](https://www.pcbway.com/) toget them manufactured with good quality and quick turnaround time. PCBWay now could provide a complete product solution, from design to enclosure production. Check out their online Gerber viewer function. With reward points, you can get free stuff from their gift shop.

**How is it different from existing protocols?**

ESP-NOW is a wireless communication protocol that is different from Wi-Fi and Bluetooth in that it reduces the five layers of the OSI model to only one1. Additionally, ESP-NOW occupies fewer CPU and flash resources than traditional connection protocols while co-exists with Wi-Fi and Bluetooth LE.

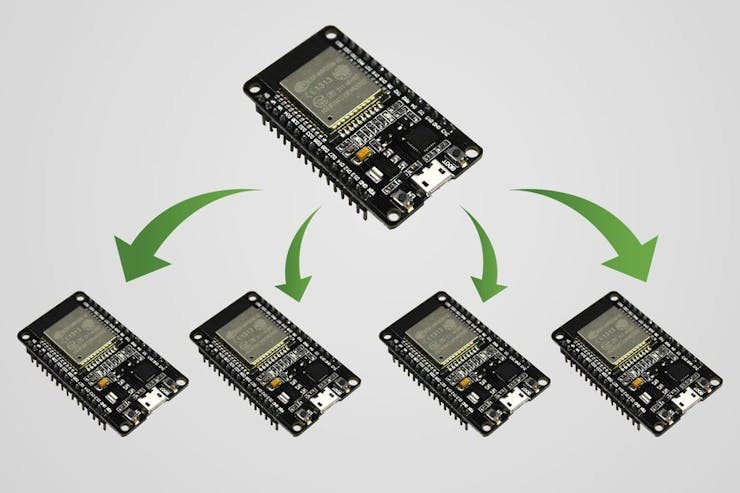
Bluetooth is used to connect short-range devices for sharing information, while Wi-Fi is used for providing high-speed internet access2. Wi-Fi provides high bandwidth because the speed of the internet is an important issue.

**Max Distance:**

The range of ESP-NOW is up to 480 meters when using the ESP-NOW protocol for bridging between multiple ESP32s1. The range can be further increased by enabling long-range ESP-NOW. When enabled, the PHY rate of ESP32 will be 512Kbps or 256Kbps.

**Maximum nodes:**

ESP-NOW supports various series of Espressif chips, providing a flexible data transmission that is suitable for connecting “one-to-many” and “many-to-many” devices.



**Applications:**

ESP-NOW is widely used in

* smart-home appliances,
* remote controlling,
* sensors, etc.

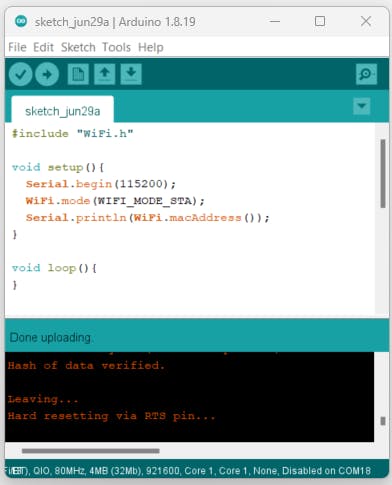
In this tutorial, will see how to implement a basic ESP NOW communication between ESP32 Microcontrollers.

**Step: 1**

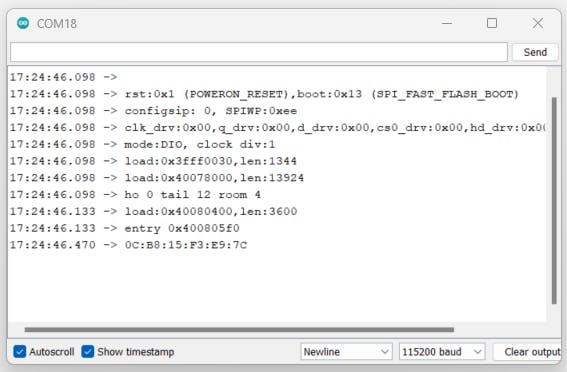
ESPNOW communication works based on the MAC address of the nodes. So, we need to find the Mac address of our slave or receiver node.

]For that just upload the following sketch to the ESP32 board and look for the Mac address in the serial monitor.

#include "WiFi.h"  
   
void setup(){  
 Serial.begin(115200);  
 WiFi.mode(WIFI\_MODE\_STA);  
 Serial.println(WiFi.macAddress());  
}  
   
void loop(){  
}



Once you uploaded the code, press the EN button and wait for the serial monitor results. It will show you the Mac address. Note that.



**Step-2:**

Next, we need to prepare the transmitter, for that use this example sketch which can send multiple data types of data to the particular slave node.

#include <esp\_now.h>  
#include <WiFi.h>  
  
// REPLACE WITH YOUR RECEIVER MAC Address  
uint8\_t broadcastAddress[] = {0x94, 0xB5, 0x55, 0x26, 0x27, 0x34};  
  
// Must match the receiver structure  
typedef struct struct\_message {  
 char a[32];  
 int b;  
 float c;  
 bool d;  
} struct\_message;  
  
// Create a struct\_message called myData  
struct\_message myData;  
  
esp\_now\_peer\_info\_t peerInfo;  
  
// callback when data is sent  
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 setup() {  
 // Init Serial Monitor  
 Serial.begin(115200);  
   
 // Set device as a Wi-Fi Station  
 WiFi.mode(WIFI\_STA);  
  
 // Init ESP-NOW  
 if (esp\_now\_init() != ESP\_OK) {  
 Serial.println("Error initializing ESP-NOW");  
 return;  
 }  
  
 // Once ESPNow is successfully Init, we will register for Send CB to  
 // get the status of Trasnmitted packet  
 esp\_now\_register\_send\_cb(OnDataSent);  
   
 // Register peer  
 memcpy(peerInfo.peer\_addr, broadcastAddress, 6);  
 peerInfo.channel = 0;   
 peerInfo.encrypt = false;  
   
 // Add peer   
 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){  
 Serial.println("Failed to add peer");  
 return;  
 }  
}  
   
void loop() {  
 // Set values to send  
 strcpy(myData.a, "I'm alive");  
 myData.b = random(1,20);  
 myData.c = 1.2;  
 myData.d = false;  
   
 // Send message via ESP-NOW  
 esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData));  
   
 if (result == ESP\_OK) {  
 Serial.println("Sent with success");  
 }  
 else {  
 Serial.println("Error sending the data");  
 }  
 delay(2000);  
}

Note: Change the Mac Address here

Here are the serial monitor results, it show sent success but not delivered. Because we don't have the receiver.



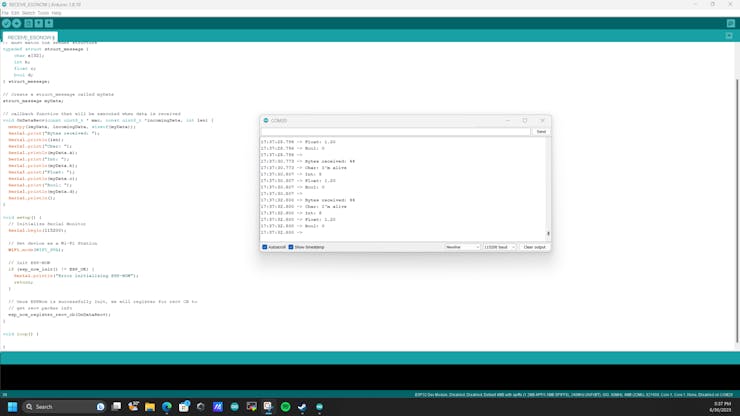
Let's try to implement the receiver.

**Step-3:**

Step-3:d example sketch which can receive the data from the master and it will print that into the serial monitor.

#include <esp\_now.h>  
#include <WiFi.h>  
  
// Structure example to receive data  
typedef struct struct\_message {  
 char a[32];  
 int b;  
 float c;  
 bool d;  
} struct\_message;  
  
// Create a struct\_message called myData  
struct\_message myData;  
  
// callback function that will be executed when data is received  
void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {  
 memcpy(&myData, incomingData, sizeof(myData));  
 Serial.print("Bytes received: ");  
 Serial.println(len);  
 Serial.print("Char: ");  
 Serial.println(myData.a);  
 Serial.print("Int: ");  
 Serial.println(myData.b);  
 Serial.print("Float: ");  
 Serial.println(myData.c);  
 Serial.print("Bool: ");  
 Serial.println(myData.d);  
 Serial.println();  
}  
   
void setup() {  
 // Initialize Serial Monitor  
 Serial.begin(115200);  
   
 // Set device as a Wi-Fi Station  
 WiFi.mode(WIFI\_STA);  
  
 // Init ESP-NOW  
 if (esp\_now\_init() != ESP\_OK) {  
 Serial.println("Error initializing ESP-NOW");  
 return;  
 }  
   
 // get recv packer info  
 esp\_now\_register\_recv\_cb(OnDataRecv);  
}  
   
void loop() {  
  
}

Serial monitor results.

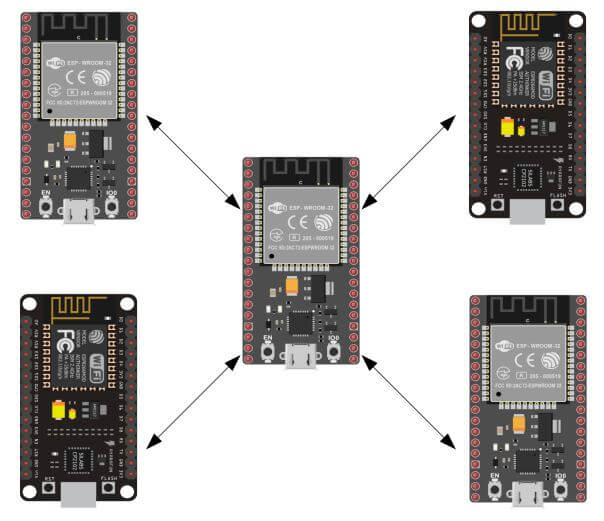


**Wrap Up:**

We have seen how to implement the ESP NOW in ESP32 microcontroller, in upcoming tutorials will see how to transmit sensor data via ESPNOW.

Creating an ESP32 network with ESP-NOW

by [Xukyo](https://www.aranacorp.com/en/author/xwadm/) | 24 Nov 2023 | [Tutorials](https://www.aranacorp.com/en/tutorials/) | [0 comments](https://www.aranacorp.com/en/creating-an-esp32-network-with-esp-now/#respond)



Tags: [ESP32](https://www.aranacorp.com/en/tag/esp32-en/), [ESP8266](https://www.aranacorp.com/en/tag/esp8266-en/), [Wireless communication](https://www.aranacorp.com/en/tag/wireless-com/)

[Facebook](https://www.aranacorp.com/#facebook)[Twitter](https://www.aranacorp.com/#twitter)[Pinterest](https://www.aranacorp.com/#pinterest)[Share](https://www.addtoany.com/share#url=https%3A%2F%2Fwww.aranacorp.com%2Fen%2Fcreating-an-esp32-network-with-esp-now%2F&title=Creating%20an%20ESP32%20network%20with%20ESP-NOW)

0

(0)

In this tutorial, we’ll look at how to set up a network of several ESP32s using the ESP-NOW protocol. The ESP32 is a Wifi-enabled development board. It can therefore connect and exchange data with devices connected to the same network.

Hardware

* Computer
* NodeMCU ESP32 or NodeMCU ESP8266 or Wemos x3
* USB A Male to Mini B Male cable x3

Description ESP-NOW

ESP-NOW is a communication protocol developed by Espressif, enabling wireless communication between several devices without the need for a specific network. It enables high-speed exchange of small data packets over 2.4GHz frequency bands, up to 200m away. It requires an initial pairing, but once this is done, communication is persistent and establishes itself on start-up. One of the great advantages, apart from the fact that it uses a dedicated network, is that one or more ESP32 or ESP8266 stations can connect to the Wifi in parallel.

Thanks to this protocol, it’s possible to create a network of NodeMCUs that communicate with each other.

Code

We’re going to set up bidirectional communication between several ESP32s, using the [espnow.h](https://github.com/espressif/esp-idf/blob/master/components/esp_wifi/include/esp_now.h) library available when you install the board manager. To simplify the code, we’ll define and send an identical data structure for all the cards.

Master code

For the sending card to be able to communicate with another card, it needs its MAC address. You can retrieve this address when the receiving card starts up from messages sent using the setup() function (Serial.println(WiFi.macAddress());). We’re going to define a function that runs after a message has been sent to check that the transmission was successful.

Don’t forget to change the slave card addresses to match the cards you’re using.

#include <esp\_now.h>

#include <**WiFi**.h>

const char nom[10]="Master";

uint8\_t broadcastAddress[2][6] = {

 {0x2C, 0xF4, 0x32, 0x15, 0x52, 0x22}, //station0

 {0xA0, 0x20, 0xA6, 0x08, 0x20, 0xD9} //station1

};// REPLACE WITH RECEIVER MAC ADDRESS

// Structure example to send data

// Must match the receiver structure

typedef struct struct\_message {

 char a[32];

 int b;

 float c;

 String d;

 bool e;

} struct\_message;

struct\_message myData;

struct\_message dataRcv;

unsigned long previousTime=0;

// callbacks for sending and receiving data

void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

**Serial**.print(F("\r\nMaster packet sent:\t"));

**Serial**.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");

}

void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

 memcpy(&dataRcv, incomingData, sizeof(dataRcv));

**Serial**.print("\r\nBytes received: ");

**Serial**.println(len);

**Serial**.print("From slave: ");

**Serial**.println(dataRcv.a);

**Serial**.println();

}

void setup() {

 // Init Serial Monitor

**Serial**.begin(115200);

 // Set device as a Wi-Fi Station

**WiFi**.mode(WIFI\_STA);

 // Init ESP-NOW

 if (esp\_now\_init() != ESP\_OK) {

**Serial**.println(F("Error initializing ESP-NOW"));

   return;

 }

**Serial**.print(F("Reciever initilized : "));

**Serial**.println(**WiFi**.macAddress());

 // Define callback functions

 esp\_now\_register\_send\_cb(OnDataSent);

 esp\_now\_register\_recv\_cb(OnDataRecv);

 // Register peer

 esp\_now\_peer\_info\_t peerInfo;

 peerInfo.channel = 0;

 peerInfo.encrypt = false;

 memcpy(peerInfo.peer\_addr, broadcastAddress[0], 6);

 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

**Serial**.println("Failed to add peer");

   return;

 }

 memcpy(peerInfo.peer\_addr, broadcastAddress[1], 6);

 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

**Serial**.println("Failed to add peer");

   return;

 }

}

void loop() {

 if((millis() -previousTime)>5000){

   // Set values to send

   strcpy(myData.a, nom);

   myData.b = random(1, 20);

   myData.c = 1.2;

   myData.e = false;

   // Send message via ESP-NOW

   myData.d = "Slave0";

   esp\_err\_t result0 = esp\_now\_send(broadcastAddress[0], (uint8\_t \*) &myData, sizeof(myData));

   myData.d = "Slave1";

   esp\_err\_t result1 = esp\_now\_send(broadcastAddress[1], (uint8\_t \*) &myData, sizeof(myData));

   previousTime=millis();

 }

}

Copy

Slave code

In the SLAVE code, we’ll create a function that runs when a message is received. This function is used to process the information received. In this example, we display the data contained in the structure.

You will need to modify the Slave code for each slave card so that the data and identifier are different.

#include <esp\_now.h>

#include <**WiFi**.h>

const char nom[10]="Slave0";

uint8\_t broadcastAddress[] = {0x3C, 0x61, 0x05, 0x30, 0x0A, 0x28};// REPLACE WITH MASTER MAC ADDRESS

//{0xDC, 0x4F, 0x22, 0x58, 0xD2, 0xF5} //station0

// Structure example to send data

// Must match the receiver structure

typedef struct struct\_message {

 char a[32];

 int b;

 float c;

 String d;

 bool e;

} struct\_message;

struct\_message dataSent;

struct\_message dataRcv;

unsigned long previousTime=0;

// callbacks for sending and receiving data

void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

**Serial**.print("\r\n"+String(nom)+" packet sent:\t");

**Serial**.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");

}

void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

 memcpy(&dataRcv, incomingData, sizeof(dataRcv));

**Serial**.print("\r\nBytes received: ");

**Serial**.println(len);

**Serial**.print("From: ");

**Serial**.println(dataRcv.a);

**Serial**.print("To: ");

**Serial**.println(dataRcv.d);

**Serial**.print("Sensor: ");

**Serial**.println(dataRcv.b);

**Serial**.print("Status: ");

**Serial**.println(dataRcv.c);

**Serial**.println();

}

void setup() {

 // Init Serial Monitor

**Serial**.begin(115200);

 // Set device as a Wi-Fi Station

**WiFi**.mode(WIFI\_STA);

 // Init ESP-NOW

 if (esp\_now\_init() != ESP\_OK) {

**Serial**.println(F("Error initializing ESP-NOW"));

   return;

 }

**Serial**.print(F("Reciever initialized : "));

**Serial**.println(**WiFi**.macAddress());

 // Define callback functions

 esp\_now\_register\_send\_cb(OnDataSent);

 esp\_now\_register\_recv\_cb(OnDataRecv);

 // Register peer

 esp\_now\_peer\_info\_t peerInfo;

 memcpy(peerInfo.peer\_addr, broadcastAddress, 6);

 peerInfo.channel = 0;

 peerInfo.encrypt = false;

 // Add peer

 if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK) {

**Serial**.println(F("Failed to add peer"));

   return;

 }

}

void loop() {

 if((millis() -previousTime)>1500){

   // Set values to send

   strcpy(dataSent.a, nom);

   dataSent.b = random(100, 200);

   dataSent.c = false;

   // Send message via ESP-NOW

   esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*) &dataSent, sizeof(dataSent));

   previousTime=millis();

 }

}

Copy

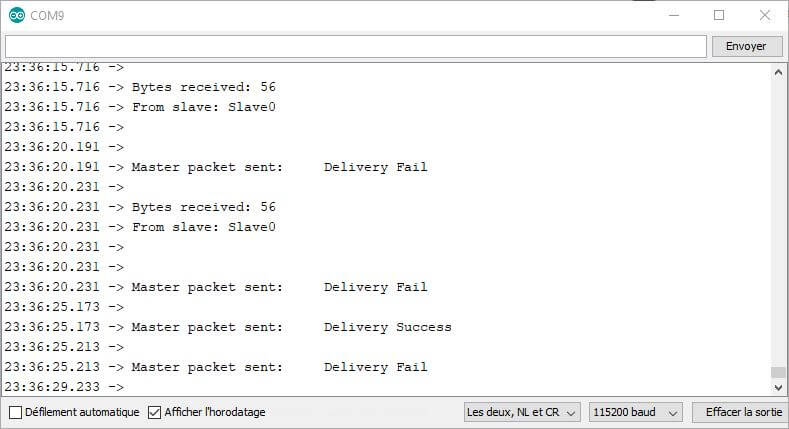
(Code for ESP8266 below)

When you upload the code for the first time, pay close attention to the setup message containing the receiver’s MAC address. You need to place it in the transmitter code to ensure successful pairing and communication.

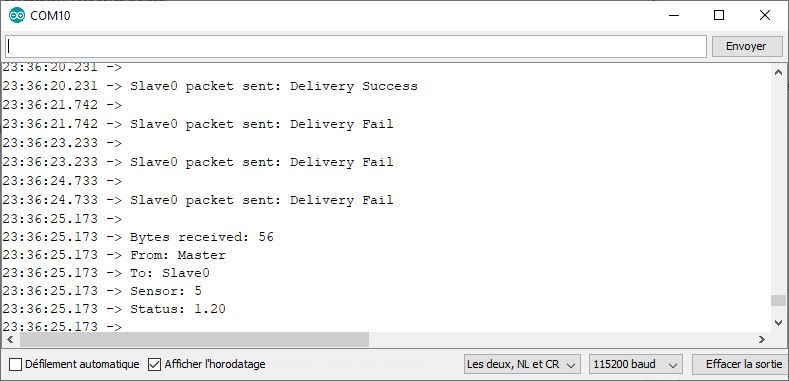
Results

Once the MAC address of the receiving card has been defined and the codes uploaded to each card, communication is established and the structure is sent and decrypted correctly.

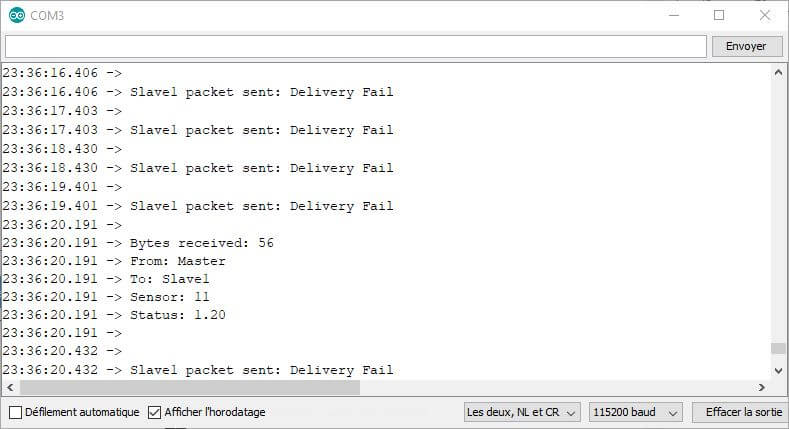
* Master



* Slave0



* Slave1



ATTENTION: during this tutorial, one of the cards seemed to be defective and was not sending messages to the master, even though it was receiving them. To be tested and validated with another card

Bonus: Communication between ESP32 and ESP8266

To integrate ESP8266 cards into your ESP-NOW network, simply modify a few lines of code.

I invite you to review the tutorial on [ESP-NOW and ESP8266](https://www.aranacorp.com/fr/communication-entre-deux-esp8266-avec-esp-now/)

To change from an ESP32 code to an ESP8266 code, you need to modify:

* Includes at the start of codeAdd an esp\_now\_set\_self\_role function with the correct role defined in each case
* Modify the arguments of the esp\_now\_add\_peer function
* Modifier les types des arguments des fonctions OnDataRecv et OnDataSent

#include <espnow.h>//https://github.com/esp8266/Arduino/blob/master/tools/sdk/include/espnow.h

#include <ESP8266WiFi.h>

const char nom[10]="Slave0";

uint8\_t broadcastAddress[] = {0x3C, 0x61, 0x05, 0x30, 0x0A, 0x28};// REPLACE WITH MASTER MAC ADDRESS

//{0xDC, 0x4F, 0x22, 0x58, 0xD2, 0xF5} //station0

// Structure example to send data

// Must match the receiver structure

typedef struct struct\_message {

 char a[32];

 int b;

 float c;

 String d;

 bool e;

} struct\_message;

struct\_message dataSent;

struct\_message dataRcv;

unsigned long previousTime=0;

// callbacks for sending and receiving data

void OnDataSent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

**Serial**.print("\r\n"+String(nom)+" packet sent:\t");

**Serial**.println(status == 0 ? "Delivery Success" : "Delivery Fail");

}

void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

 memcpy(&dataRcv, incomingData, sizeof(dataRcv));

**Serial**.print("\r\nBytes received: ");

**Serial**.println(len);

**Serial**.print("From: ");

**Serial**.println(dataRcv.a);

**Serial**.print("To: ");

**Serial**.println(dataRcv.d);

**Serial**.print("Sensor: ");

**Serial**.println(dataRcv.b);

**Serial**.print("Status: ");

**Serial**.println(dataRcv.c);

**Serial**.println();

}

void setup() {

 // Init Serial Monitor

**Serial**.begin(115200);

 // Set device as a Wi-Fi Station

**WiFi**.mode(WIFI\_STA);

 // Init ESP-NOW

 if (esp\_now\_init() != 0) {

**Serial**.println(F("Error initializing ESP-NOW"));

   return;

 }

**Serial**.print(F("Reciever initialized : "));

**Serial**.println(**WiFi**.macAddress());

 // Define callback functions

 esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_SLAVE);

 esp\_now\_register\_send\_cb(OnDataSent);

 esp\_now\_register\_recv\_cb(OnDataRecv);

 // Register peer

 esp\_now\_add\_peer(broadcastAddress, ESP\_NOW\_ROLE\_CONTROLLER, 1, NULL, 0);

}

void loop() {

 if((millis() -previousTime)>1500){

   // Set values to send

   strcpy(dataSent.a, nom);

   dataSent.b = random(100, 200);

   dataSent.c = false;

   // Send message via ESP-NOW

   uint8\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*) &dataSent, sizeof(dataSent));

   previousTime=millis();

 }

}

Copy

Bonus: Display MAC address for copying

The MAC address of the Maestro or Slaves can be displayed in a format that allows it to be copied directly into broadcastAddress.

void getMacAdress(const uint8\_t \* mac){

 /\*for (int i=0; i<6; i++){

   if (mac[i]<10) Serial.print(0,HEX),Serial.print(mac[i],HEX); // FF:FF:FF:FF:FF:FF

   else Serial.print(mac[i],HEX);

   if(i<5) Serial.print(",");

 } \*/

**Serial**.print("{");

 for (int i=0; i<6; i++){

**Serial**.print("0x");

   if (mac[i]<10) **Serial**.print(0,HEX),**Serial**.print(mac[i],HEX);  // {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF}

   else **Serial**.print(mac[i],HEX);

   if(i<5) **Serial**.print(",");

 }

**Serial**.print("}");

}

Copy

Sources

* [Communication between two ESP32s with ESP-NOW](https://www.aranacorp.com/fr/communication-entre-deux-esp32-avec-esp-now/)
* [Communication between two ESP8266 with ESP-NOW](https://www.aranacorp.com/fr/communication-entre-deux-esp8266-avec-esp-now)
* [ESP-NOW Documentation Espressive](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html)

# [espnow](https://docs.micropython.org/en/latest/library/espnow.html#module-espnow) — support for the ESP-NOW wireless protocol[¶](https://docs.micropython.org/en/latest/library/espnow.html#module-espnow)

This module provides an interface to the [ESP-NOW](https://www.espressif.com/en/products/software/esp-now/overview) protocol provided by Espressif on ESP32 and ESP8266 devices ([API docs](https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/network/esp_now.html)).

**Table of Contents:**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#table-of-contents)

* [Introduction](https://docs.micropython.org/en/latest/library/espnow.html#introduction)
* [Configuration](https://docs.micropython.org/en/latest/library/espnow.html#configuration)
* [Sending and Receiving Data](https://docs.micropython.org/en/latest/library/espnow.html#sending-and-receiving-data)
* [Peer Management](https://docs.micropython.org/en/latest/library/espnow.html#peer-management)
* [Callback Methods](https://docs.micropython.org/en/latest/library/espnow.html#callback-methods)
* [Exceptions](https://docs.micropython.org/en/latest/library/espnow.html#exceptions)
* [Constants](https://docs.micropython.org/en/latest/library/espnow.html#constants)
* [Wifi Signal Strength (RSSI) - (ESP32 Only)](https://docs.micropython.org/en/latest/library/espnow.html#wifi-signal-strength-rssi-esp32-only)
* [Supporting asyncio](https://docs.micropython.org/en/latest/library/espnow.html#supporting-asyncio)
* [Broadcast and Multicast](https://docs.micropython.org/en/latest/library/espnow.html#broadcast-and-multicast)
* [ESPNow and Wifi Operation](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-wifi-operation)
* [ESPNow and Sleep Modes](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-sleep-modes)

**Introduction**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#introduction)

ESP-NOW is a connection-less wireless communication protocol supporting:

* Direct communication between up to 20 registered peers:
  + Without the need for a wireless access point (AP),
* Encrypted and unencrypted communication (up to 6 encrypted peers),
* Message sizes up to 250 bytes,
* Can operate alongside Wifi operation ([network.WLAN](https://docs.micropython.org/en/latest/library/network.WLAN.html)) on ESP32 and ESP8266 devices.

It is especially useful for small IoT networks, latency sensitive or power sensitive applications (such as battery operated devices) and for long-range communication between devices (hundreds of metres).

This module also supports tracking the Wifi signal strength (RSSI) of peer devices.

A simple example would be:

**Sender:**

**import** **network**

**import** **espnow**

*# A WLAN interface must be active to send()/recv()*

sta = network.WLAN(network.STA\_IF) *# Or network.AP\_IF*

sta.active(**True**)

sta.disconnect() *# For ESP8266*

e = espnow.ESPNow()

e.active(**True**)

peer = b'**\xbb\xbb\xbb\xbb\xbb\xbb**' *# MAC address of peer's wifi interface*

e.add\_peer(peer) *# Must add\_peer() before send()*

e.send(peer, "Starting...")

**for** i **in** range(100):

e.send(peer, str(i)\*20, **True**)

e.send(peer, b'end')

**Receiver:**

**import** **network**

**import** **espnow**

*# A WLAN interface must be active to send()/recv()*

sta = network.WLAN(network.STA\_IF)

sta.active(**True**)

sta.disconnect() *# Because ESP8266 auto-connects to last Access Point*

e = espnow.ESPNow()

e.active(**True**)

**while** **True**:

host, msg = e.recv()

**if** msg: *# msg == None if timeout in recv()*

print(host, msg)

**if** msg == b'end':

**break**

**class ESPNow**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#class-espnow)

**Constructor**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#constructor)

***class*espnow.ESPNow**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow)

Returns the singleton ESPNow object. As this is a singleton, all calls to [**espnow.ESPNow()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow) return a reference to the same object.

**Note**

Some methods are available only on the ESP32 due to code size restrictions on the ESP8266 and differences in the Espressif API.

**Configuration**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#configuration)

**ESPNow.active([*flag*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.active)

Initialise or de-initialise the ESP-NOW communication protocol depending on the value of the flag optional argument.

**Arguments:**

* *flag*: Any python value which can be converted to a boolean type.
  + True: Prepare the software and hardware for use of the ESP-NOW communication protocol, including:
    - initialise the ESPNow data structures,
    - allocate the recv data buffer,
    - invoke esp\_now\_init() and
    - register the send and recv callbacks.
  + False: De-initialise the Espressif ESP-NOW software stack (esp\_now\_deinit()), disable callbacks, deallocate the recv data buffer and deregister all peers.

If *flag* is not provided, return the current status of the ESPNow interface.

**Returns:**

True if interface is currently *active*, else False.

**ESPNow.config(*param=value*, *...*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.config)

**ESPNow.config(*'param')   (ESP32 only*)**

Set or get configuration values of the ESPNow interface. To set values, use the keyword syntax, and one or more parameters can be set at a time. To get a value the parameter name should be quoted as a string, and just one parameter is queried at a time.

**Note:** *Getting* parameters is not supported on the ESP8266.

**Options:**

*rxbuf*: (default=526) Get/set the size in bytes of the internal buffer used to store incoming ESPNow packet data. The default size is selected to fit two max-sized ESPNow packets (250 bytes) with associated mac\_address (6 bytes), a message byte count (1 byte) and RSSI data plus buffer overhead. Increase this if you expect to receive a lot of large packets or expect bursty incoming traffic.

**Note:** The recv buffer is allocated by [**ESPNow.active()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.active). Changing this value will have no effect until the next call of [**ESPNow.active(True)**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.active).

*timeout\_ms*: (default=300,000) Default timeout (in milliseconds) for receiving ESPNow messages. If *timeout\_ms* is less than zero, then wait forever. The timeout can also be provided as arg to [**recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv)/[**irecv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv)/[**recvinto()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recvinto).

*rate*: (ESP32 only, IDF>=4.3.0 only) Set the transmission speed for ESPNow packets. Must be set to a number from the allowed numeric values in [enum wifi\_phy\_rate\_t](https://docs.espressif.com/projects/esp-idf/en/v4.4.1/esp32/api-reference/network/esp_wifi.html#_CPPv415wifi_phy_rate_t).

**Returns:**

None or the value of the parameter being queried.

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* ValueError() on invalid configuration options or values.

**Sending and Receiving Data**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#sending-and-receiving-data)

A wifi interface (network.STA\_IF or network.AP\_IF) must be [**active()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) before messages can be sent or received, but it is not necessary to connect or configure the WLAN interface. For example:

**import** **network**

sta = network.WLAN(network.STA\_IF)

sta.active(**True**)

sta.disconnect() *# For ESP8266*

**Note:** The ESP8266 has a *feature* that causes it to automatically reconnect to the last wifi Access Point when set [**active(True)**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) (even after reboot/reset). This reduces the reliability of receiving ESP-NOW messages (see [ESPNow and Wifi Operation](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-wifi-operation)). You can avoid this by calling [**disconnect()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.disconnect) after [**active(True)**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active).

**ESPNow.send(*mac*, *msg*[, *sync*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send)

**ESPNow.send(*msg)   (ESP32 only*)**

Send the data contained in msg to the peer with given network mac address. In the second form, mac=None and sync=True. The peer must be registered with [**ESPNow.add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer) before the message can be sent.

**Arguments:**

* *mac*: byte string exactly espnow.ADDR\_LEN (6 bytes) long or None. If *mac* is None (ESP32 only) the message will be sent to all registered peers, except any broadcast or multicast MAC addresses.
* *msg*: string or byte-string up to espnow.MAX\_DATA\_LEN (250) bytes long.
* *sync*:
  + True: (default) send msg to the peer(s) and wait for a response (or not).
  + False send msg and return immediately. Responses from the peers will be discarded.

**Returns:**

True if sync=False or if sync=True and *all* peers respond, else False.

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_FOUND") if peer is not registered.
* OSError(num, "ESP\_ERR\_ESPNOW\_IF") the wifi interface is not [**active()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active).
* OSError(num, "ESP\_ERR\_ESPNOW\_NO\_MEM") internal ESP-NOW buffers are full.
* ValueError() on invalid values for the parameters.

**Note**: A peer will respond with success if its wifi interface is [**active()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) and set to the same channel as the sender, regardless of whether it has initialised it’s ESP-NOW system or is actively listening for ESP-NOW traffic (see the Espressif ESP-NOW docs).

**ESPNow.recv([*timeout\_ms*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv)

Wait for an incoming message and return the mac address of the peer and the message. **Note**: It is **not** necessary to register a peer (using [**add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer)) to receive a message from that peer.

**Arguments:**

* *timeout\_ms*: (Optional): May have the following values.
  + 0: No timeout. Return immediately if no data is available;
  + > 0: Specify a timeout value in milliseconds;
  + < 0: Do not timeout, ie. wait forever for new messages; or
  + None (or not provided): Use the default timeout value set with [**ESPNow.config()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.config).

**Returns:**

* (None, None) if timeout is reached before a message is received, or
* [mac, msg]: where:
  + mac is a bytestring containing the address of the device which sent the message, and
  + msg is a bytestring containing the message.

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* OSError(num, "ESP\_ERR\_ESPNOW\_IF") if the wifi interface is not [**active()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active).
* ValueError() on invalid *timeout\_ms* values.

[**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv) will allocate new storage for the returned list and the peer and msg bytestrings. This can lead to memory fragmentation if the data rate is high. See [**ESPNow.irecv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv) for a memory-friendly alternative.

**ESPNow.irecv([*timeout\_ms*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv)

Works like [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv) but will reuse internal bytearrays to store the return values: [mac, msg], so that no new memory is allocated on each call.

**Arguments:**

*timeout\_ms*: (Optional) Timeout in milliseconds (see [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv)).

**Returns:**

* As for [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv), except that msg is a bytearray, instead of a bytestring. On the ESP8266, mac will also be a bytearray.

**Raises:**

* See [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv).

**Note:** You may also read messages by iterating over the ESPNow object, which will use the [**irecv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv) method for alloc-free reads, eg:

**import** **espnow**

e = espnow.ESPNow(); e.active(**True**)

***for*** mac, msg **in** e:

print(mac, msg)

***if*** mac **is** **None**: *# mac, msg will equal (None, None) on timeout*

***break***

**ESPNow.recvinto(*data*[, *timeout\_ms*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recvinto)

Wait for an incoming message and return the length of the message in bytes. This is the low-level method used by both [**recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv) and [**irecv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv) to read messages.

**Arguments:**

*data*: A list of at least two elements, [peer, msg]. msg must be a bytearray large enough to hold the message (250 bytes). On the ESP8266, peer should be a bytearray of 6 bytes. The MAC address of the sender and the message will be stored in these bytearrays (see Note on ESP32 below).

*timeout\_ms*: (Optional) Timeout in milliseconds (see [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv)).

**Returns:**

* Length of message in bytes or 0 if *timeout\_ms* is reached before a message is received.

**Raises:**

* See [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv).

**Note:** On the ESP32:

* It is unnecessary to provide a bytearray in the first element of the data list because it will be replaced by a reference to a unique peer address in the **peer device table** (see [**ESPNow.peers\_table**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.peers_table)).
* If the list is at least 4 elements long, the rssi and timestamp values will be saved as the 3rd and 4th elements.

**ESPNow.any()**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.any)

Check if data is available to be read with [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv).

For more sophisticated querying of available characters use [**select.poll()**](https://docs.micropython.org/en/latest/library/select.html#select.poll):

**import** **select**

**import** **espnow**

e = espnow.ESPNow()

poll = select.poll()

poll.register(e, select.POLLIN)

poll.poll(timeout)

**Returns:**

True if data is available to be read, else False.

**ESPNow.stats(*) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.stats)

**Returns:**

A 5-tuple containing the number of packets sent/received/lost:

(tx\_pkts, tx\_responses, tx\_failures, rx\_packets, rx\_dropped\_packets)

Incoming packets are *dropped* when the recv buffers are full. To reduce packet loss, increase the rxbuf config parameters and ensure you are reading messages as quickly as possible.

**Note**: Dropped packets will still be acknowledged to the sender as received.

**Peer Management**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#peer-management)

On ESP32 devices, the Espressif ESP-NOW software requires that other devices (peers) must be *registered* using [**add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer) before we can [**send()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send) them messages (this is *not* enforced on ESP8266 devices). It is **not** necessary to register a peer to receive an un-encrypted message from that peer.

**Encrypted messages**: To receive an *encrypted* message, the receiving device must first register the sender and use the same encryption keys as the sender (PMK and LMK) (see [**set\_pmk()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.set_pmk) and [**add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer).

**ESPNow.set\_pmk(*pmk*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.set_pmk)

Set the Primary Master Key (PMK) which is used to encrypt the Local Master Keys (LMK) for encrypting messages. If this is not set, a default PMK is used by the underlying Espressif ESP-NOW software stack.

**Note:** messages will only be encrypted if *lmk* is also set in [**ESPNow.add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer) (see [Security](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html#security) in the Espressif API docs).

**Arguments:**

*pmk*: Must be a byte string, bytearray or string of length [**espnow.KEY\_LEN**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.KEY_LEN) (16 bytes).

**Returns:**

None

**Raises:**

ValueError() on invalid *pmk* values.

**ESPNow.add\_peer(*mac*[, *lmk*][, *channel*][, *ifidx*][, *encrypt*])**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer)

**ESPNow.add\_peer(*mac*, *param=value*, *...)   (ESP32 only*)**

Add/register the provided *mac* address as a peer. Additional parameters may also be specified as positional or keyword arguments (any parameter set to None will be set to it’s default value):

**Arguments:**

* *mac*: The MAC address of the peer (as a 6-byte byte-string).
* *lmk*: The Local Master Key (LMK) key used to encrypt data transfers with this peer (unless the *encrypt* parameter is set to False). Must be:
  + a byte-string or bytearray or string of length espnow.KEY\_LEN (16 bytes), or
  + any non True python value (default= b''), signifying an *empty* key which will disable encryption.
* *channel*: The wifi channel (2.4GHz) to communicate with this peer. Must be an integer from 0 to 14. If channel is set to 0 the current channel of the wifi device will be used. (default=0)
* *ifidx*: (ESP32 only) Index of the wifi interface which will be used to send data to this peer. Must be an integer set to network.STA\_IF (=0) or network.AP\_IF (=1). (default=0/network.STA\_IF). See [ESPNow and Wifi Operation](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-wifi-operation) below for more information.
* *encrypt*: (ESP32 only) If set to True data exchanged with this peer will be encrypted with the PMK and LMK. (default = True if *lmk* is set to a valid key, else False)

**ESP8266**: Keyword args may not be used on the ESP8266.

**Note:** The maximum number of peers which may be registered is 20 ([**espnow.MAX\_TOTAL\_PEER\_NUM**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.MAX_TOTAL_PEER_NUM)), with a maximum of 6 ([**espnow.MAX\_ENCRYPT\_PEER\_NUM**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.MAX_ENCRYPT_PEER_NUM)) of those peers with encryption enabled (see [ESP\_NOW\_MAX\_ENCRYPT\_PEER\_NUM](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html#c.ESP_NOW_MAX_ENCRYPT_PEER_NUM) in the Espressif API docs).

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* OSError(num, "ESP\_ERR\_ESPNOW\_EXIST") if *mac* is already registered.
* OSError(num, "ESP\_ERR\_ESPNOW\_FULL") if too many peers are already registered.
* ValueError() on invalid keyword args or values.

**ESPNow.del\_peer(*mac*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.del_peer)

Deregister the peer associated with the provided *mac* address.

**Returns:**

None

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_FOUND") if *mac* is not registered.
* ValueError() on invalid *mac* values.

**ESPNow.get\_peer(*mac) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.get_peer)

Return information on a registered peer.

**Returns:**

(mac, lmk, channel, ifidx, encrypt): a tuple of the “peer info” associated with the given *mac* address.

**Raises:**

* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_INIT") if not initialised.
* OSError(num, "ESP\_ERR\_ESPNOW\_NOT\_FOUND") if *mac* is not registered.
* ValueError() on invalid *mac* values.

**ESPNow.peer\_count(*) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.peer_count)

Return the number of registered peers:

* (peer\_num, encrypt\_num): where
  + peer\_num is the number of peers which are registered, and
  + encrypt\_num is the number of encrypted peers.

**ESPNow.get\_peers(*) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.get_peers)

Return the “peer info” parameters for all the registered peers (as a tuple of tuples).

**ESPNow.mod\_peer(*mac, lmk, [channel], [ifidx], [encrypt]) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.mod_peer)

**ESPNow.mod\_peer(*mac*, *'param'=value*, *...) (ESP32 only*)**

Modify the parameters of the peer associated with the provided *mac* address. Parameters may be provided as positional or keyword arguments (see [**ESPNow.add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer)). Any parameter that is not set (or set to None) will retain the existing value for that parameter.

**Callback Methods**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#callback-methods)

**ESPNow.irq(*callback) (ESP32 only*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irq)

Set a callback function to be called *as soon as possible* after a message has been received from another ESPNow device. The callback function will be called with the [**ESPNow**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow) instance object as an argument. For more reliable operation, it is recommended to read out as many messages as are available when the callback is invoked and to set the read timeout to zero, eg:

***def*** recv\_cb(e):

***while*** **True**: *# Read out all messages waiting in the buffer*

mac, msg = e.irecv(0) *# Don't wait if no messages left*

***if*** mac **is** **None**:

***return***

print(mac, msg)

e.irq(recv\_cb)

The [**irq()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irq) callback method is an alternative method for processing incoming messages, especially if the data rate is moderate and the device is *not too busy* but there are some caveats:

* The scheduler stack *can* overflow and callbacks will be missed if packets are arriving at a sufficient rate or if other MicroPython components (eg, bluetooth, machine.Pin.irq(), machine.timer, i2s, …) are exercising the scheduler stack. This method may be less reliable for dealing with bursts of messages, or high throughput or on a device which is busy dealing with other hardware operations.
* For more information on *scheduled* function callbacks see: [**micropython.schedule()**](https://docs.micropython.org/en/latest/library/micropython.html#micropython.schedule).

**Constants**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#constants)

**espnow.MAX\_DATA\_LEN(*=250*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.MAX_DATA_LEN)

**espnow.KEY\_LEN(*=16*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.KEY_LEN)

**espnow.ADDR\_LEN(*=6*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.ADDR_LEN)

**espnow.MAX\_TOTAL\_PEER\_NUM(*=20*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.MAX_TOTAL_PEER_NUM)

**espnow.MAX\_ENCRYPT\_PEER\_NUM(*=6*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.espnow.MAX_ENCRYPT_PEER_NUM)

**Exceptions**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#exceptions)

If the underlying Espressif ESP-NOW software stack returns an error code, the MicroPython espnow module will raise an OSError(errnum, errstring) exception where errstring is set to the name of one of the error codes identified in the [Espressif ESP-NOW docs](https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/network/esp_now.html#api-reference). For example:

**try**:

e.send(peer, 'Hello')

**except** OSError **as** err:

**if** len(err.args) < 2:

**raise** err

**if** err.args[1] == 'ESP\_ERR\_ESPNOW\_NOT\_INIT':

e.active(**True**)

**elif** err.args[1] == 'ESP\_ERR\_ESPNOW\_NOT\_FOUND':

e.add\_peer(peer)

**elif** err.args[1] == 'ESP\_ERR\_ESPNOW\_IF':

network.WLAN(network.STA\_IF).active(**True**)

**else**:

**raise** err

**Wifi Signal Strength (RSSI) - (ESP32 only)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#wifi-signal-strength-rssi-esp32-only)

The ESPNow object maintains a **peer device table** which contains the signal strength and timestamp of the last received message from all hosts. The **peer device table** can be accessed using [**ESPNow.peers\_table**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.peers_table) and can be used to track device proximity and identify *nearest neighbours* in a network of peer devices. This feature is **not** available on ESP8266 devices.

**ESPNow.peers\_table**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.peers_table)

A reference to the **peer device table**: a dict of known peer devices and rssi values:

{peer: [rssi, time\_ms], ...}

where:

* peer is the peer MAC address (as [**bytes**](https://docs.micropython.org/en/latest/library/builtins.html#bytes));
* rssi is the wifi signal strength in dBm (-127 to 0) of the last message received from the peer; and
* time\_ms is the time the message was received (in milliseconds since system boot - wraps every 12 days).

Example:

**>>>** e.peers\_table

{b'\xaa\xaa\xaa\xaa\xaa\xaa': [-31, 18372],

b'\xbb\xbb\xbb\xbb\xbb\xbb': [-43, 12541]}

**Note**: the mac addresses returned by [**recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv) are references to the peer key values in the **peer device table**.

**Note**: RSSI and timestamp values in the device table are updated only when the message is read by the application.

**Supporting asyncio**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#supporting-asyncio)

A supplementary module ([**aioespnow**](https://docs.micropython.org/en/latest/library/espnow.html#module-aioespnow)) is available to provide [asyncio](https://docs.micropython.org/en/latest/library/asyncio.html) support.

**Note:** Asyncio support is available on all ESP32 targets as well as those ESP8266 boards which include the asyncio module (ie. ESP8266 devices with at least 2MB flash memory).

A small async server example:

**import** **network**

**import** **aioespnow**

**import** **asyncio**

*# A WLAN interface must be active to send()/recv()*

network.WLAN(network.STA\_IF).active(**True**)

e = aioespnow.AIOESPNow() *# Returns AIOESPNow enhanced with async support*

e.active(**True**)

peer = b'**\xbb\xbb\xbb\xbb\xbb\xbb**'

e.add\_peer(peer)

*# Send a periodic ping to a peer*

**async** **def** heartbeat(e, peer, period=30):

**while** **True**:

**if** **not** **await** e.asend(peer, b'ping'):

print("Heartbeat: peer not responding:", peer)

**else**:

print("Heartbeat: ping", peer)

**await** asyncio.sleep(period)

*# Echo any received messages back to the sender*

**async** **def** echo\_server(e):

**async** **for** mac, msg **in** e:

print("Echo:", msg)

**try**:

**await** e.asend(mac, msg)

**except** OSError **as** err:

**if** len(err.args) > 1 **and** err.args[1] == 'ESP\_ERR\_ESPNOW\_NOT\_FOUND':

e.add\_peer(mac)

**await** e.asend(mac, msg)

**async** **def** main(e, peer, timeout, period):

asyncio.create\_task(heartbeat(e, peer, period))

asyncio.create\_task(echo\_server(e))

**await** asyncio.sleep(timeout)

asyncio.run(main(e, peer, 120, 10))

***class*aioespnow.AIOESPNow**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#aioespnow.AIOESPNow)

The [**AIOESPNow**](https://docs.micropython.org/en/latest/library/espnow.html#aioespnow.AIOESPNow) class inherits all the methods of [**ESPNow**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow) and extends the interface with the following async methods.

**async AIOESPNow.arecv()**

Asyncio support for [**ESPNow.recv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.recv). Note that this method does not take a timeout value as argument.

**async AIOESPNow.airecv()**

Asyncio support for [**ESPNow.irecv()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.irecv). Note that this method does not take a timeout value as argument.

**async AIOESPNow.asend(mac, msg, sync=True)**

**async AIOESPNow.asend(msg)**

Asyncio support for [**ESPNow.send()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send).

**AIOESPNow.\_aiter\_\_(*) / async AIOESPNow.\_\_anext\_\_(*)**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#aioespnow.AIOESPNow._aiter__)

[**AIOESPNow**](https://docs.micropython.org/en/latest/library/espnow.html#aioespnow.AIOESPNow) also supports reading incoming messages by asynchronous iteration using async for; eg:

e = AIOESPNow()

e.active(**True**)

***async*** ***def*** recv\_till\_halt(e):

***async*** ***for*** mac, msg **in** e:

print(mac, msg)

***if*** msg == b'halt':

***break***

asyncio.run(recv\_till\_halt(e))

**Broadcast and Multicast**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#broadcast-and-multicast)

All active ESPNow clients will receive messages sent to their MAC address and all devices (**except ESP8266 devices**) will also receive messages sent to the *broadcast* MAC address (b'\xff\xff\xff\xff\xff\xff') or any multicast MAC address.

All ESPNow devices (including ESP8266 devices) can also send messages to the broadcast MAC address or any multicast MAC address.

To [**send()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send) a broadcast message, the broadcast (or multicast) MAC address must first be registered using [**add\_peer()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.add_peer). [**send()**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send) will always return True for broadcasts, regardless of whether any devices receive the message. It is not permitted to encrypt messages sent to the broadcast address or any multicast address.

**Note**: [**ESPNow.send(None, msg)**](https://docs.micropython.org/en/latest/library/espnow.html#espnow.ESPNow.send) will send to all registered peers *except* the broadcast address. To send a broadcast or multicast message, you must specify the broadcast (or multicast) MAC address as the peer. For example:

bcast = b'**\xff**' \* 6

e.add\_peer(bcast)

e.send(bcast, "Hello World!")

**ESPNow and Wifi Operation**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-wifi-operation)

ESPNow messages may be sent and received on any [**active()**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) [**WLAN**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN) interface (network.STA\_IF or network.AP\_IF), even if that interface is also connected to a wifi network or configured as an access point. When an ESP32 or ESP8266 device connects to a Wifi Access Point (see [ESP32 Quickref](https://docs.micropython.org/en/latest/esp32/quickref.html#networking)) the following things happen which affect ESPNow communications:

1. Wifi Power-saving Mode ([**network.WLAN.PM\_PERFORMANCE**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.PM_PERFORMANCE)) is automatically activated and
2. The radio on the esp device changes wifi channel to match the channel used by the Access Point.

**Wifi Power-saving Mode:** (see [Espressif Docs](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#esp32-wi-fi-power-saving-mode)) The power saving mode causes the device to turn off the radio periodically (typically for hundreds of milliseconds), making it unreliable in receiving ESPNow messages. This can be resolved by either of:

1. Disabling the power-saving mode on the STA\_IF interface;
   * Use sta.config(pm=sta.PM\_NONE)
2. Turning on the AP\_IF interface, which will disable the power saving mode. However, the device will then be advertising an active wifi access point.
   * You **may** also choose to send your messages via the AP\_IF interface, but this is not necessary.
   * ESP8266 peers must send messages to this AP\_IF interface (see below).
3. Configuring ESPNow clients to retry sending messages.

**Receiving messages from an ESP8266 device:** Strangely, an ESP32 device connected to a wifi network using method 1 or 2 above, will receive ESPNow messages sent to the STA\_IF MAC address from another ESP32 device, but will **reject** messages from an ESP8266 device!!!. To receive messages from an ESP8266 device, the AP\_IF interface must be set to active(True) **and** messages must be sent to the AP\_IF MAC address.

**Managing wifi channels:** Any other ESPNow devices wishing to communicate with a device which is also connected to a Wifi Access Point MUST use the same channel. A common scenario is where one ESPNow device is connected to a wifi router and acts as a proxy for messages from a group of sensors connected via ESPNow:

**Proxy:**

**import** **network**, **time**, **espnow**

sta, ap = wifi\_reset() *# Reset wifi to AP off, STA on and disconnected*

sta.connect('myssid', 'mypassword')

**while** **not** sta.isconnected(): *# Wait until connected...*

time.sleep(0.1)

sta.config(pm=sta.PM\_NONE) *# ..then disable power saving*

*# Print the wifi channel used AFTER finished connecting to access point*

print("Proxy running on channel:", sta.config("channel"))

e = espnow.ESPNow(); e.active(**True**)

**for** peer, msg **in** e:

*# Receive espnow messages and forward them to MQTT broker over wifi*

**Sensor:**

**import** **network**, **espnow**

sta, ap = wifi\_reset() *# Reset wifi to AP off, STA on and disconnected*

sta.config(channel=6) *# Change to the channel used by the proxy above.*

peer = b'0**\xaa\xaa\xaa\xaa\xaa**' *# MAC address of proxy*

e = espnow.ESPNow(); e.active(**True**);

e.add\_peer(peer)

**while** **True**:

msg = read\_sensor()

e.send(peer, msg)

time.sleep(1)

Other issues to take care with when using ESPNow with wifi are:

* **Set WIFI to known state on startup:** MicroPython does not reset the wifi peripheral after a soft reset. This can lead to unexpected behaviour. To guarantee the wifi is reset to a known state after a soft reset make sure you deactivate the STA\_IF and AP\_IF before setting them to the desired state at startup, eg.:
* **import** **network**, **time**
* **def** wifi\_reset(): *# Reset wifi to AP\_IF off, STA\_IF on and disconnected*
* sta = network.WLAN(network.STA\_IF); sta.active(**False**)
* ap = network.WLAN(network.AP\_IF); ap.active(**False**)
* sta.active(**True**)
* **while** **not** sta.active():
* time.sleep(0.1)
* sta.disconnect() *# For ESP8266*
* **while** sta.isconnected():
* time.sleep(0.1)
* **return** sta, ap
* sta, ap = wifi\_reset()

Remember that a soft reset occurs every time you connect to the device REPL and when you type ctrl-D.

* **STA\_IF and AP\_IF always operate on the same channel:** the AP\_IF will change channel when you connect to a wifi network; regardless of the channel you set for the AP\_IF (see [Attention Note 3](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html#_CPPv419esp_wifi_set_config16wifi_interface_tP13wifi_config_t) ). After all, there is really only one wifi radio on the device, which is shared by the STA\_IF and AP\_IF virtual devices.
* **Disable automatic channel assignment on your wifi router:** If the wifi router for your wifi network is configured to automatically assign the wifi channel, it may change the channel for the network if it detects interference from other wifi routers. When this occurs, the ESP devices connected to the wifi network will also change channels to match the router, but other ESPNow-only devices will remain on the previous channel and communication will be lost. To mitigate this, either set your wifi router to use a fixed wifi channel or configure your devices to re-scan the wifi channels if they are unable to find their expected peers on the current channel.
* **MicroPython re-scans wifi channels when trying to reconnect:** If the esp device is connected to a Wifi Access Point that goes down, MicroPython will automatically start scanning channels in an attempt to reconnect to the Access Point. This means ESPNow messages will be lost while scanning for the AP. This can be disabled by sta.config(reconnects=0), which will also disable the automatic reconnection after losing connection.
* Some versions of the ESP IDF only permit sending ESPNow packets from the STA\_IF interface to peers which have been registered on the same wifi channel as the STA\_IF:
* ESPNOW: Peer channel is not equal to the home channel, send fail!

**ESPNow and Sleep Modes**[**¶**](https://docs.micropython.org/en/latest/library/espnow.html#espnow-and-sleep-modes)

The [**machine.lightsleep([time\_ms])**](https://docs.micropython.org/en/latest/library/machine.html#machine.lightsleep) and [**machine.deepsleep([time\_ms])**](https://docs.micropython.org/en/latest/library/machine.html#machine.deepsleep) functions can be used to put the ESP32 and peripherals (including the WiFi and Bluetooth radios) to sleep. This is useful in many applications to conserve battery power. However, applications must disable the WLAN peripheral (using [**active(False)**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active)) before entering light or deep sleep (see [Sleep Modes](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html)). Otherwise the WiFi radio may not be initialised properly after wake from sleep. If the STA\_IF and AP\_IF interfaces have both been set [**active(True)**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) then both interfaces should be set [**active(False)**](https://docs.micropython.org/en/latest/library/network.WLAN.html#network.WLAN.active) before entering any sleep mode.

**Example:** deep sleep:

**import** **network**, **machine**, **espnow**

sta, ap = wifi\_reset() *# Reset wifi to AP off, STA on and disconnected*

peer = b'0**\xaa\xaa\xaa\xaa\xaa**' *# MAC address of peer*

e = espnow.ESPNow()

e.active(**True**)

e.add\_peer(peer) *# Register peer on STA\_IF*

print('Sending ping...')

**if** **not** e.send(peer, b'ping'):

print('Ping failed!')

e.active(**False**)

sta.active(**False**) *# Disable the wifi before sleep*

print('Going to sleep...')

machine.deepsleep(10000) *# Sleep for 10 seconds then reboot*

**Example:** light sleep:

**import** **network**, **machine**, **espnow**

sta, ap = wifi\_reset() *# Reset wifi to AP off, STA on and disconnected*

sta.config(channel=6)

peer = b'0**\xaa\xaa\xaa\xaa\xaa**' *# MAC address of peer*

e = espnow.ESPNow()

e.active(**True**)

e.add\_peer(peer) *# Register peer on STA\_IF*

**while** **True**:

print('Sending ping...')

**if** **not** e.send(peer, b'ping'):

print('Ping failed!')

sta.active(**False**) *# Disable the wifi before sleep*

print('Going to sleep...')

machine.lightsleep(10000) *# Sleep for 10 seconds*

sta.active(**True**)

sta.config(channel=6) *# Wifi loses config after lightsleep()*

**ESP-NOW**

**Overview**

ESP-NOW is a kind of connectionless WiFi communication protocol which is defined by Espressif. In ESP-NOW, application data is encapsulated in vendor-specific action frame and then transmitted from one WiFi device to another without connection. CTR with CBC-MAC Protocol(CCMP) is used to protect the action frame for security. ESP-NOW is widely used in smart light, remote controlling, sensor, etc.

**Frame Format**

ESP-NOW uses vendor-specific action frame to transmit ESP-NOW data. The format of vendor-specific action frame is as follows:

----------------------------------------------------------------------------------------

| MAC Header | Category Code | Organization Identifier | Vendor Specific Content | FCS |

----------------------------------------------------------------------------------------

1 byte 3 bytes 7~255 bytes

* Category Code: The Category field is set to the value(127) indicating the vendor-specific category.
* **Organization Identifier: The Organization Identifier contains a unique identifier(0x18fe34) which is the first three bytes**

of MAC address applied by Espressif.

* Vendor Specific Content: The Vendor Specific Content contains vendor-specific field as follows:

-------------------------------------------------------------------------------

| Element ID | Length | Organization Identifier | Type | Version | Body |

-------------------------------------------------------------------------------

1 byte 1 byte 3 bytes 1 byte 1 byte 0~250 bytes

* Element ID: The Element ID field is set to the value(221) indicating the vendor-specific element.
* Length: The length is the total length of Organization Identifier, Type, Version and Body.
* **Organization Identifier: The Organization Identifier contains a unique identifier(0x18fe34) which is the first three bytes**

of MAC address applied by Espressif.

* Type: The Type field is set to the value(4) indicating ESP-NOW.
* Version: The Version field is set to the version of ESP-NOW.
* Body: The Body contains the ESP-NOW data.

As ESP-NOW is connectionless, the MAC header is a little different from that of standard frames. The FromDS and ToDS bits of FrameControl field are both 0. The first address field is set to the destination address. The second address field is set to the source address. The third address field is set to broadcast address(0xff:0xff:0xff:0xff:0xff:0xff).

**Security**

ESP-NOW use CCMP method which can be referenced in IEEE Std. 802.11-2012 to protect the vendor-specific action frame. The WiFi device maintains a Primary Master Key(PMK) and several Local Master Keys(LMK). The lengths of them are 16 bytes. PMK is used to encrypt LMK with AES-128 algorithm. Call esp\_now\_set\_pmk() to set PMK. If PMK is not set, a default PMK will be used. If LMK of the paired device is set, it will be used to encrypt the vendor-specific action frame with CCMP method. The maximum number of different LMKs is six. Do not support encrypting multicast vendor-specific action frame.

**Initialization and De-initialization**

Call esp\_now\_init() to initialize ESP-NOW and esp\_now\_deinit() to de-initialize ESP-NOW. ESP-NOW data must be transmitted after WiFi is started, so it is recommended to start WiFi before initializing ESP-NOW and stop WiFi after de-initializing ESP-NOW. When esp\_now\_deinit() is called, all of the information of paired devices will be deleted.

**Add Paired Device**

Before sending data to other device, call esp\_now\_add\_peer() to add it to the paired device list first. The maximum number of paired devices is twenty. If security is enabled, the LMK must be set. ESP-NOW data can be sent from station or softap interface. Make sure that the interface is enabled before sending ESP-NOW data. A device with broadcast MAC address must be added before sending broadcast data. The range of the channel of paired device is from 0 to 14. If the channel is set to 0, data will be sent on the current channel. Otherwise, the channel must be set as the channel that the local device is on.

**Send ESP-NOW Data**

Call esp\_now\_send() to send ESP-NOW data and esp\_now\_register\_send\_cb to register sending callback function. It will return *ESP\_NOW\_SEND\_SUCCESS* in sending callback function if the data is received successfully on MAC layer. Otherwise, it will return *ESP\_NOW\_SEND\_FAIL*. There are several reasons failing to send ESP-NOW data, for example, the destination device doesn’t exist, the channels of the devices are not the same, the action frame is lost when transmiting on the air, etc. It is not guaranteed that application layer can receive the data. If necessary, send back ack data when receiving ESP-NOW data. If receiving ack data timeout happens, retransmit the ESP-NOW data. A sequence number can also be assigned to ESP-NOW data to drop the duplicated data.

If there is a lot of ESP-NOW data to send, call esp\_now\_send() to send less than or equal to 250 bytes of data once a time. Note that too short interval between sending two ESP-NOW datas may lead to disorder of sending callback function. So, it is recommended that sending the next ESP-NOW data after the sending callback function of previous sending has returned. The sending callback function runs from a high-priority WiFi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task.

**Receiving ESP-NOW Data**

Call esp\_now\_register\_recv\_cb to register receiving callback function. When receiving ESP-NOW data, receiving callback function is called. The receiving callback function also runs from WiFi task. So, do not do lengthy operations in the callback function. Instead, post necessary data to a queue and handle it from a lower priority task.

**API Reference**

**Header File**

* [esp32/include/esp\_now.h](https://github.com/espressif/esp-idf/blob/f8bda32/components/esp32/include/esp_now.h)

**Functions**

**esp\_err\_t esp\_now\_init(void)**

Initialize ESPNOW function.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_INTERNAL : Internal error

**esp\_err\_t esp\_now\_deinit(void)**

De-initialize ESPNOW function.

**Return**

* ESP\_OK : succeed

**esp\_err\_t esp\_now\_get\_version(uint32\_t \**version*)**

Get the version of ESPNOW.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

* **version**: ESPNOW version

**esp\_err\_t esp\_now\_register\_recv\_cb(**[**esp\_now\_recv\_cb\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv217esp_now_recv_cb_t)***cb*)**

Register callback function of receiving ESPNOW data.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_INTERNAL : internal error

**Parameters**

* **cb**: callback function of receiving ESPNOW data

**esp\_err\_t esp\_now\_unregister\_recv\_cb(void)**

Unregister callback function of receiving ESPNOW data.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized

**esp\_err\_t esp\_now\_register\_send\_cb(**[**esp\_now\_send\_cb\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv217esp_now_send_cb_t)***cb*)**

Register callback function of sending ESPNOW data.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_INTERNAL : internal error

**Parameters**

* **cb**: callback function of sending ESPNOW data

**esp\_err\_t esp\_now\_unregister\_send\_cb(void)**

Unregister callback function of sending ESPNOW data.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized

**esp\_err\_t esp\_now\_send(*const* uint8\_t \**peer\_addr*, *const* uint8\_t \**data*, size\_t *len*)**

Send ESPNOW data.

**Attention**

1. If peer\_addr is not NULL, send data to the peer whose MAC address matches peer\_addr

**Attention**

2. If peer\_addr is NULL, send data to all of the peers that are added to the peer list

**Attention**

3. The maximum length of data must be less than ESP\_NOW\_MAX\_DATA\_LEN

**Attention**

4. The buffer pointed to by data argument does not need to be valid after esp\_now\_send returns

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_INTERNAL : internal error
* ESP\_ERR\_ESPNOW\_NO\_MEM : out of memory
* ESP\_ERR\_ESPNOW\_NOT\_FOUND : peer is not found
* ESP\_ERR\_ESPNOW\_IF : current WiFi interface doesn’t match that of peer

**Parameters**

* **peer\_addr**: peer MAC address
* **data**: data to send
* **len**: length of data

**esp\_err\_t esp\_now\_add\_peer(*const***[**esp\_now\_peer\_info\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv219esp_now_peer_info_t)**\**peer*)**

Add a peer to peer list.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_FULL : peer list is full
* ESP\_ERR\_ESPNOW\_NO\_MEM : out of memory
* ESP\_ERR\_ESPNOW\_EXIST : peer has existed

**Parameters**

* **peer**: peer information

**esp\_err\_t esp\_now\_del\_peer(*const* uint8\_t \**peer\_addr*)**

Delete a peer from peer list.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_NOT\_FOUND : peer is not found

**Parameters**

* **peer\_addr**: peer MAC address

**esp\_err\_t esp\_now\_mod\_peer(*const***[**esp\_now\_peer\_info\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv219esp_now_peer_info_t)**\**peer*)**

Modify a peer.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_FULL : peer list is full

**Parameters**

* **peer**: peer information

**esp\_err\_t esp\_now\_get\_peer(*const* uint8\_t \**peer\_addr*,**[**esp\_now\_peer\_info\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv219esp_now_peer_info_t)**\**peer*)**

Get a peer whose MAC address matches peer\_addr from peer list.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_NOT\_FOUND : peer is not found

**Parameters**

* **peer\_addr**: peer MAC address
* **peer**: peer information

**esp\_err\_t esp\_now\_fetch\_peer(bool *from\_head*,**[**esp\_now\_peer\_info\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv219esp_now_peer_info_t)**\**peer*)**

Fetch a peer from peer list.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument
* ESP\_ERR\_ESPNOW\_NOT\_FOUND : peer is not found

**Parameters**

* **from\_head**: fetch from head of list or not
* **peer**: peer information

**bool esp\_now\_is\_peer\_exist(*const* uint8\_t \**peer\_addr*)**

Peer exists or not.

**Return**

* true : peer exists
* false : peer not exists

**Parameters**

* **peer\_addr**: peer MAC address

**esp\_err\_t esp\_now\_get\_peer\_num(**[**esp\_now\_peer\_num\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv218esp_now_peer_num_t)**\**num*)**

Get the number of peers.

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

* **num**: number of peers

**esp\_err\_t esp\_now\_set\_pmk(*const* uint8\_t \**pmk*)**

Set the primary master key.

**Attention**

1. primary master key is used to encrypt local master key

**Return**

* ESP\_OK : succeed
* ESP\_ERR\_ESPNOW\_NOT\_INIT : ESPNOW is not initialized
* ESP\_ERR\_ESPNOW\_ARG : invalid argument

**Parameters**

* **pmk**: primary master key

**Structures**

***struct*esp\_now\_peer\_info**

ESPNOW peer information parameters.

**Public Members**

**uint8\_t peer\_addr[ESP\_NOW\_ETH\_ALEN]**

ESPNOW peer MAC address that is also the MAC address of station or softap

**uint8\_t lmk[ESP\_NOW\_KEY\_LEN]**

ESPNOW peer local master key that is used to encrypt data

**uint8\_t channel**

Wi-Fi channel that peer uses to send/receive ESPNOW data. If the value is 0, use the current channel which station or softap is on. Otherwise, it must be set as the channel that station or softap is on.

[**wifi\_interface\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_wifi.html#_CPPv216wifi_interface_t)**ifidx**

Wi-Fi interface that peer uses to send/receive ESPNOW data

**bool encrypt**

ESPNOW data that this peer sends/receives is encrypted or not

**void \*priv**

ESPNOW peer private data

***struct*esp\_now\_peer\_num**

Number of ESPNOW peers which exist currently.

**Public Members**

**int total\_num**

Total number of ESPNOW peers, maximum value is ESP\_NOW\_MAX\_TOTAL\_PEER\_NUM

**int encrypt\_num**

Number of encrypted ESPNOW peers, maximum value is ESP\_NOW\_MAX\_ENCRYPT\_PEER\_NUM

**Macros**

**ESP\_ERR\_ESPNOW\_BASE**

ESPNOW error number base.

**ESP\_ERR\_ESPNOW\_NOT\_INIT**

ESPNOW is not initialized.

**ESP\_ERR\_ESPNOW\_ARG**

Invalid argument

**ESP\_ERR\_ESPNOW\_NO\_MEM**

Out of memory

**ESP\_ERR\_ESPNOW\_FULL**

ESPNOW peer list is full

**ESP\_ERR\_ESPNOW\_NOT\_FOUND**

ESPNOW peer is not found

**ESP\_ERR\_ESPNOW\_INTERNAL**

Internal error

**ESP\_ERR\_ESPNOW\_EXIST**

ESPNOW peer has existed

**ESP\_ERR\_ESPNOW\_IF**

Interface error

**ESP\_NOW\_ETH\_ALEN**

Length of ESPNOW peer MAC address

**ESP\_NOW\_KEY\_LEN**

Length of ESPNOW peer local master key

**ESP\_NOW\_MAX\_TOTAL\_PEER\_NUM**

Maximum number of ESPNOW total peers

**ESP\_NOW\_MAX\_ENCRYPT\_PEER\_NUM**

Maximum number of ESPNOW encrypted peers

**ESP\_NOW\_MAX\_DATA\_LEN**

Maximum length of ESPNOW data which is sent very time

**Type Definitions**

***typedefstruct***[**esp\_now\_peer\_info**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv217esp_now_peer_info)**esp\_now\_peer\_info\_t**

ESPNOW peer information parameters.

***typedefstruct***[**esp\_now\_peer\_num**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv216esp_now_peer_num)**esp\_now\_peer\_num\_t**

Number of ESPNOW peers which exist currently.

***typedef*void (\*esp\_now\_recv\_cb\_t)(*const* uint8\_t \*mac\_addr, *const* uint8\_t \*data, int data\_len)**

Callback function of receiving ESPNOW data.

**Parameters**

* **mac\_addr**: peer MAC address
* **data**: received data
* **data\_len**: length of received data

***typedef*void (\*esp\_now\_send\_cb\_t)(*const* uint8\_t \*mac\_addr,**[**esp\_now\_send\_status\_t**](https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html#_CPPv221esp_now_send_status_t)**status)**

Callback function of sending ESPNOW data.

**Parameters**

* **mac\_addr**: peer MAC address
* **status**: status of sending ESPNOW data (succeed or fail)

**Enumerations**

***enum*esp\_now\_send\_status\_t**

Status of sending ESPNOW data .

*Values:*

**ESP\_NOW\_SEND\_SUCCESS = 0**

Send ESPNOW data successfully

**ESP\_NOW\_SEND\_FAIL**

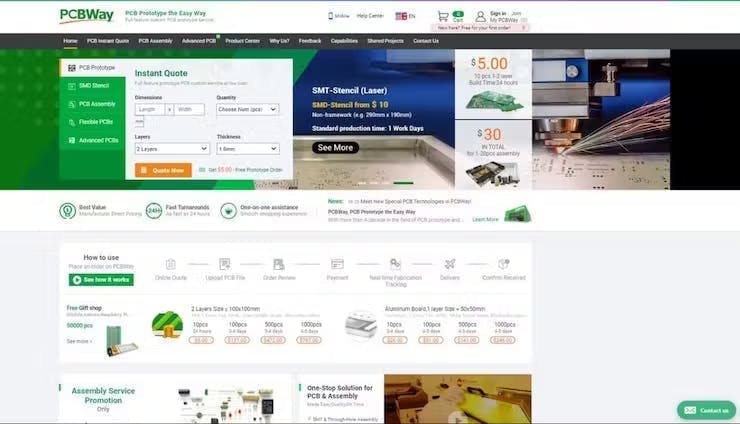
Send ESPNOW data fail

ESP-NOW is a wireless communication protocol based on the data-link layer that enables the direct, quick, and low-power control of smart devices without the need for a router. Espressif defines it and can work with Wi-Fi and Bluetooth LE. ESP-NOW provides flexible and low-power data transmission to all interconnected devices. It can also be used as an independent protocol that helps with device provisioning, debugging, and firmware upgrades.



ESP-NOW is a connectionless communication protocol developed by Espressif that features short packet transmission. This protocol enables multiple devices to talk to each other in an easy way. It is a fast communication protocol that can be used to exchange small messages (up to 250 bytes) between ESP32 or ESP8266 boards. ESP-NOW supports the following features: Encrypted and unencrypted unicast communication; Mixed encrypted and unencrypted peer devices; Up to 250-byte payload can be carried; Sending callback function that can be set to inform the application layer of transmission success or failure.

**Get PCBs for Your Projects Manufactured**



You must check out [PCBWAY](https://www.pcbway.com/) for ordering PCBs online for cheap!

You get 10 good-quality PCBs manufactured and shipped to your doorstep for cheap. You will also get a discount on shipping on your first order. Upload your Gerber files onto [PCBWAY](https://www.pcbway.com/) toget them manufactured with good quality and quick turnaround time. PCBWay now could provide a complete product solution, from design to enclosure production. Check out their online Gerber viewer function. With reward points, you can get free stuff from their gift shop.

**How is it different from existing protocols?**

ESP-NOW is a wireless communication protocol that is different from Wi-Fi and Bluetooth in that it reduces the five layers of the OSI model to only one1. Additionally, ESP-NOW occupies fewer CPU and flash resources than traditional connection protocols while co-exists with Wi-Fi and Bluetooth LE.

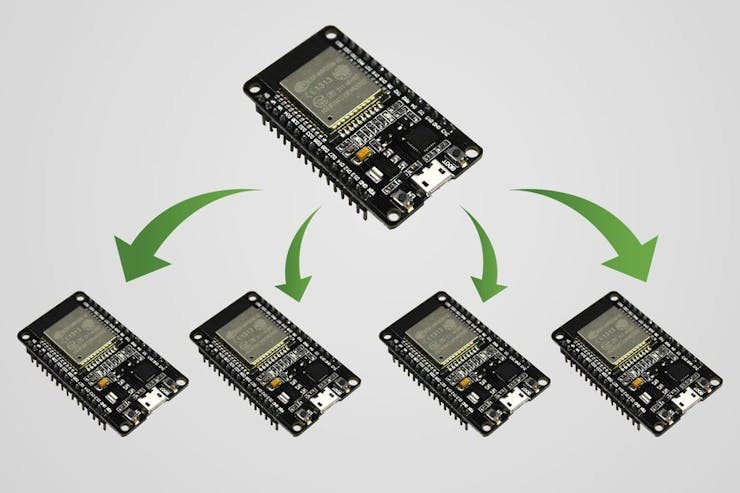
Bluetooth is used to connect short-range devices for sharing information, while Wi-Fi is used for providing high-speed internet access2. Wi-Fi provides high bandwidth because the speed of the internet is an important issue.

**Max Distance:**

The range of ESP-NOW is up to 480 meters when using the ESP-NOW protocol for bridging between multiple ESP32s1. The range can be further increased by enabling long-range ESP-NOW. When enabled, the PHY rate of ESP32 will be 512Kbps or 256Kbps.

**Maximum nodes:**

ESP-NOW supports various series of Espressif chips, providing a flexible data transmission that is suitable for connecting “one-to-many” and “many-to-many” devices.



**Applications:**

ESP-NOW is widely used in

smart-home appliances,remote controlling,sensors, etc.

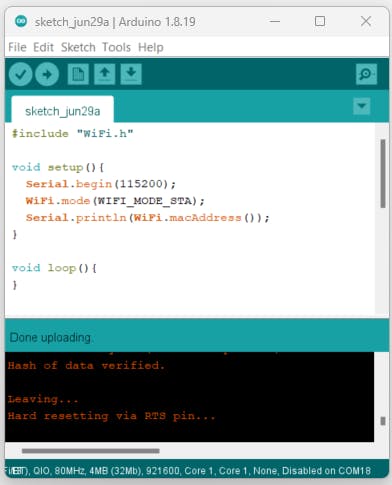
In this tutorial, will see how to implement a basic ESP NOW communication between ESP32 Microcontrollers.

**Step: 1**

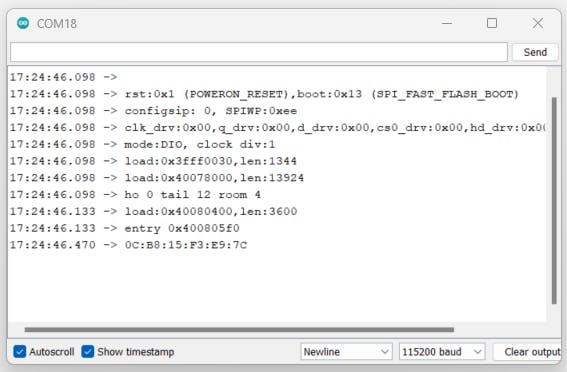
ESPNOW communication works based on the MAC address of the nodes. So, we need to find the Mac address of our slave or receiver node.

]For that just upload the following sketch to the ESP32 board and look for the Mac address in the serial monitor.

|  |  |
| --- | --- |
|  | #include "WiFi.h" |
|  |  |
|  | void setup(){ |
|  | Serial.begin(115200); |
|  | WiFi.mode(WIFI\_MODE\_STA); |
|  | Serial.println(WiFi.macAddress()); |
|  | } |
|  |  |
|  | void loop(){ |
|  | } |



Once you uploaded the code, press the EN button and wait for the serial monitor results. It will show you the Mac address. Note that.



**Step-2:**

Next, we need to prepare the transmitter, for that use this example sketch which can send multiple data types of data to the particular slave node.

|  |  |
| --- | --- |
|  | #include <esp\_now.h> |
|  | #include <WiFi.h> |
|  |  |
|  | // REPLACE WITH YOUR RECEIVER MAC Address |
|  | uint8\_t broadcastAddress[] = {0x94, 0xB5, 0x55, 0x26, 0x27, 0x34}; |
|  |  |
|  | // Must match the receiver structure |
|  | typedef struct struct\_message { |
|  | char a[32]; |
|  | int b; |
|  | float c; |
|  | bool d; |
|  | } struct\_message; |
|  |  |
|  | // Create a struct\_message called myData |
|  | struct\_message myData; |
|  |  |
|  | esp\_now\_peer\_info\_t peerInfo; |
|  |  |
|  | // callback when data is sent |
|  | 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 setup() { |
|  | // Init Serial Monitor |
|  | Serial.begin(115200); |
|  |  |
|  | // Set device as a Wi-Fi Station |
|  | WiFi.mode(WIFI\_STA); |
|  |  |
|  | // Init ESP-NOW |
|  | if (esp\_now\_init() != ESP\_OK) { |
|  | Serial.println("Error initializing ESP-NOW"); |
|  | return; |
|  | } |
|  |  |
|  | // Once ESPNow is successfully Init, we will register for Send CB to |
|  | // get the status of Trasnmitted packet |
|  | esp\_now\_register\_send\_cb(OnDataSent); |
|  |  |
|  | // Register peer |
|  | memcpy(peerInfo.peer\_addr, broadcastAddress, 6); |
|  | peerInfo.channel = 0; |
|  | peerInfo.encrypt = false; |
|  |  |
|  | // Add peer |
|  | if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){ |
|  | Serial.println("Failed to add peer"); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | void loop() { |
|  | // Set values to send |
|  | strcpy(myData.a, "I'm alive"); |
|  | myData.b = random(1,20); |
|  | myData.c = 1.2; |
|  | myData.d = false; |
|  |  |
|  | // Send message via ESP-NOW |
|  | esp\_err\_t result = esp\_now\_send(broadcastAddress, (uint8\_t \*) &myData, sizeof(myData)); |
|  |  |
|  | if (result == ESP\_OK) { |
|  | Serial.println("Sent with success"); |
|  | } |
|  | else { |
|  | Serial.println("Error sending the data"); |
|  | } |
|  | delay(2000); |
|  | } |

Note: Change the Mac Address here

Here are the serial monitor results, it show sent success but not delivered. Because we don't have the receiver.



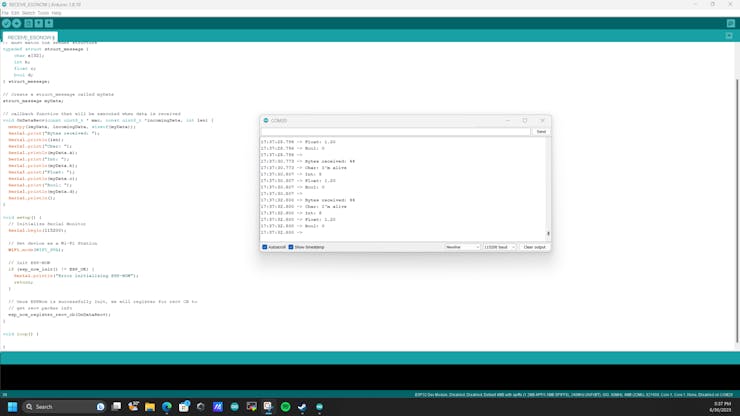
Let's try to implement the receiver.

**Step-3:**

With the help of the below example sketch, you can receive the data from the master and it will print that into the serial monitor.

|  |  |
| --- | --- |
|  | #include <esp\_now.h> |
|  | #include <WiFi.h> |
|  |  |
|  | // Structure example to receive data |
|  | typedef struct struct\_message { |
|  | char a[32]; |
|  | int b; |
|  | float c; |
|  | bool d; |
|  | } struct\_message; |
|  |  |
|  | // Create a struct\_message called myData |
|  | struct\_message myData; |
|  |  |
|  | // callback function that will be executed when data is received |
|  | void OnDataRecv(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) { |
|  | memcpy(&myData, incomingData, sizeof(myData)); |
|  | Serial.print("Bytes received: "); |
|  | Serial.println(len); |
|  | Serial.print("Char: "); |
|  | Serial.println(myData.a); |
|  | Serial.print("Int: "); |
|  | Serial.println(myData.b); |
|  | Serial.print("Float: "); |
|  | Serial.println(myData.c); |
|  | Serial.print("Bool: "); |
|  | Serial.println(myData.d); |
|  | Serial.println(); |
|  | } |
|  |  |
|  | void setup() { |
|  | // Initialize Serial Monitor |
|  | Serial.begin(115200); |
|  |  |
|  | // Set device as a Wi-Fi Station |
|  | WiFi.mode(WIFI\_STA); |
|  |  |
|  | // Init ESP-NOW |
|  | if (esp\_now\_init() != ESP\_OK) { |
|  | Serial.println("Error initializing ESP-NOW"); |
|  | return; |
|  | } |
|  |  |
|  | // get recv packer info |
|  | esp\_now\_register\_recv\_cb(OnDataRecv); |
|  | } |
|  |  |
|  | void loop() { |
|  |  |
|  | } |

Serial monitor results.



**Wrap Up:**

We have seen how to implement the ESP NOW in ESP32 microcontroller, in upcoming tutorials will see how to transmit sensor data via ESPNOW.

# ESP8266 ESP-Now Transceiver

[Tarek Ragab](https://eg.linkedin.com/in/tarek-ragab?trk=article-ssr-frontend-pulse_publisher-author-card)

### Tarek Ragab

#### **IoT Solution Developer | Flutter**

Published Sep 2, 2022

[**+ Follow**](https://www.linkedin.com/signup/cold-join?session_redirect=%2Fpulse%2Fesp8266-esp-now-transceiver-tarek-ragab&trk=article-ssr-frontend-pulse_publisher-author-card)

Hello everyone, I decided to share with you some of my work from time to time, as I hope it may help anyone. Anyways, let's dive into this simple project

As we know IoT has been one of the most trending technologies recently, and ESP chips are considered one of the most commonly WiFi chips that can be easily programmed. One of the features of these ESP chips is that they have the ability to communicate together using a communication protocol developed by "Espressif Systems" called "ESP-Now". And according to the official documentation:

ESP-NOW is yet another protocol developed by Espressif, which enables multiple devices to communicate with one another without using Wi-Fi. The protocol is similar to the low-power 2.4GHz wireless connectivity that is often deployed in wireless mouses. So, the pairing between devices is needed prior to their communication. After the pairing is done, the connection is secure and peer-to-peer, with no handshake being required.

### ESP-NOW supports the following features:

* Encrypted and unencrypted unicast communication.
* Mixed encrypted and unencrypted peer devices.
* **Up to a 250-byte** payload can be carried.
* Sending callback function that can be set to inform the application layer of transmission success or failure.

And as mentioned, it can be implemented on ESP8266 and ESP32 boards. In this article, I am going to explain how to implement two-way communication. So I am going to use "NodeMCU V1.0" with "Platform IO" and "Arduino Framework" to implement ESP-Now communication.

### First, let's explain how does ESP-Now work:

1. Initialization and setting the role of the board. The roles can be either **slave**or **controller**or **combo.**In our case, we will use the "combo" role, in order to make the node capable of both sending and receiving data packets
2. Setting the callback functions of receiving and sending messages. Where these callback functions will get triggered when a packet is received or when a packet is delivered.
3. Registering peers. Connecting to the other node via its mac address.
4. And finally comes your code logic, where you implement when to send messages and how to handle incoming messages.

One important thing to know before implementing the code is that we need to define the format of the message packet using a struct, where this format must be matched between both the transmitter and the receiver or as how they are called in ESP-Now, the controller and the slave.

### Code implementation:

The code begins by including the libraries, where we will find the required APIs.

#include <ESP8266WiFi.h>

#include <espnow.h>

Then we define the struct that defines the data packet format, I defined it as two arrays of characters, where I will be sending info about the message in the header part and the actual message in the body part.

// It must match the struct sent initially

typedef struct struct\_message

{

char header[64];

char body[128];

} struct\_message;

For the receiving callback function: we will just need to implement a function that has 3 arguments which are the mac address of the sender, the incoming data packet and the length of this packet.

// Define an struct to store the received packet at it

struct\_message rec\_packet;

// Callback function that will be executed when data is received

void OnDataRecv(uint8\_t \*mac, uint8\_t \*incomingData, uint8\_t len)

{

memcpy(&rec\_packet, incomingData, sizeof(rec\_packet));

Serial.print("Bytes received: ");

Serial.println(len);

Serial.println("Header: ");

Serial.println(rec\_packet.header);

Serial.println("Body: ");

Serial.println(rec\_packet.body);

}

Now we will implement the sending callback function. We just need to implement a function that takes 2 arguments, which are the mac address of the node will be sending data to and the status of sending this packet.

struct\_message sen\_packet;

// Callback function that will be executed when data is sent

void OnDataSent(uint8\_t \*mac\_addr, uint8\_t sendStatus)

{

Serial.print("Last packet Send Status: ");

if (sendStatus == 0)

{

Serial.println("Delivery success");

}

else

{

Serial.println("Delivery fail");

}

}

For the void setup: We need to initialize esp-now and set the role of the node, and then set the callback functions, and finally peer with the other node

void setup()

{

// Initialize Serial Monitor

Serial.begin(115200);

// Set device as a Wi-Fi Station

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

Serial.print("MAC Address:\t");

Serial.println(WiFi.macAddress());

// Init ESP-NOW

initiallize();

// Set the role of the ESP to be a receiver

esp\_now\_set\_self\_role(ESP\_NOW\_ROLE\_COMBO);

// Set the callback functions

esp\_now\_register\_recv\_cb(OnDataRecv);

esp\_now\_register\_send\_cb(OnDataSent);

esp\_now\_add\_peer(target\_mac\_add, ESP\_NOW\_ROLE\_COMBO, 1, NULL, 0);

}

For testing, I will be just sending a packet periodically.

// Period of data transmitting

unsigned long lastTime = 0; // Last time the data was sent

unsigned long timerDelay = 2000; // Period of two seconds

void send\_packet()

{

// Set values to be sent

strcpy(sen\_packet.header, "This is a Header");

strcpy(sen\_packet.body, "This is a body - data");

// Send a packet via ESP-NOW

esp\_now\_send(target\_mac\_add, (uint8\_t \*)&sen\_packet, sizeof(sen\_packet));

}

Now we just need to call this function in the void loop periodically

void loop()

{

if ((millis() - lastTime) > timerDelay)

{

send\_packet();

lastTime = millis();

}

}

# ESP32 ESP-NOW Send Data to Multiple boards (One to Many Communication)

In this tutorial, we will see how to send data from one ESP32/ESP8266 to multiple ESP32 and ESP8266 boards using ESP-NOW and Arduino IDE. In other words, we will transmit data from one ESP32 to many ESP32 and ESP8266 boards (**One to Many Communication**). Previously, we sent data from one ESP32 to another ESP32 via ESP NOW one way communication. Additionally, we also transmitted sensor readings between two ESP32 boards and displayed the readings on an OLED display via ESP-NOW two-way communication.

You can access both of the user guides below:

* [ESP32 ESP-NOW Getting Started Tutorial with Arduino IDE](https://microcontrollerslab.com/esp32-esp-now-tutorial-arduino-ide/)
* [ESP32 ESP-NOW Two way Communication (Arduino IDE)](https://microcontrollerslab.com/esp32-esp-now-two-way-communication-arduino-ide/)

## ESP-NOW Protocol Introduction

ESP-NOW is a low-power, secure, and direct wireless communication protocol that enables multiple ESP32 devices to communicate with each other without the need for Wi-Fi or a router. Using ESP-NOW, we can perform one-way and even two-way communication between ESP MCU devices without using a Wi-Fi network.

[](https://www.pcbway.com/HighQualityOrderOnline.aspx?from=microcontrollerlab.com2022D)

* It allows low overhead peer-to-peer wireless data transfers but in small packets. A maximum of 250 bytes of data can be transferred. Thus if a larger amount of data needs to be transferred then using this protocol is not useful.
* Using ESP-NOW, the connection protocol is simplified which results in low power consumption as a lesser amount of time is required for the transmission of data.
* Additionally, the ESP-NOW uses the same 2.4 GHz band as the Wi-Fi but does not need to connect or interfere with the local network connection.

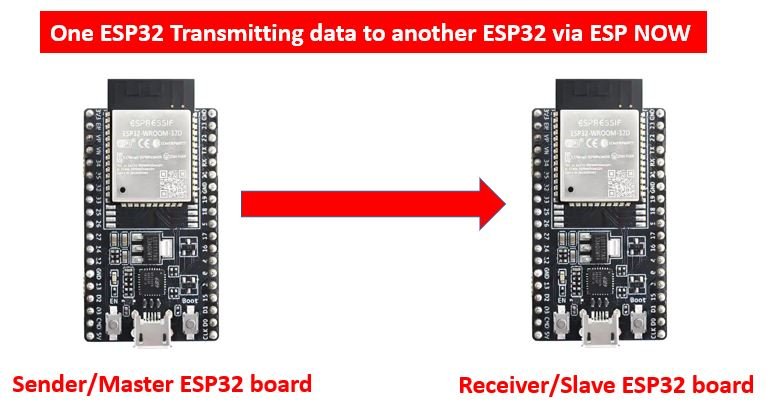
It is a fast and convenient communication protocol for the transmission of a smaller amount of data.

## ESP32 ESP-NOW one-way communication

In one-way communication, one peered device acts as the sender/master and the other as the receiver/slave. We can have multiple configurations of the sender-receiver in this situation.

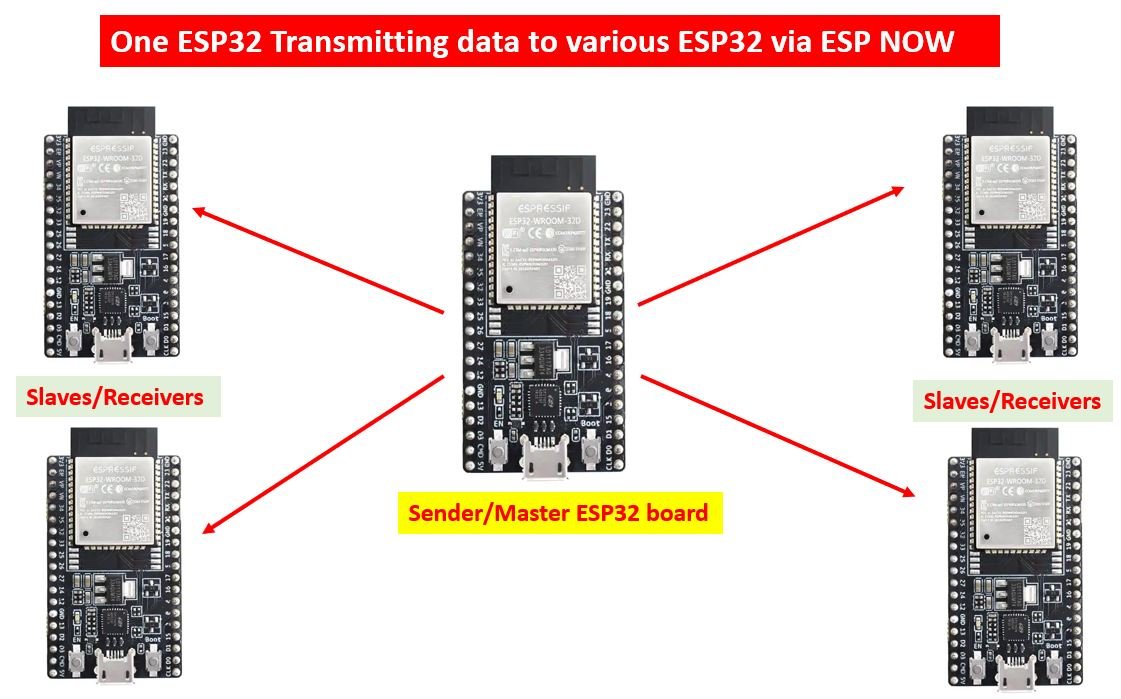
* One ESP32 board sends data to another ESP32 board

As you can view in the picture below, one ESP32 board act as the sender and the other board receives the data and hence acts as the receiver.  
Uses: Sending sensor data, controlling ESP outputs including LEDs, relays, buzzers, etc.



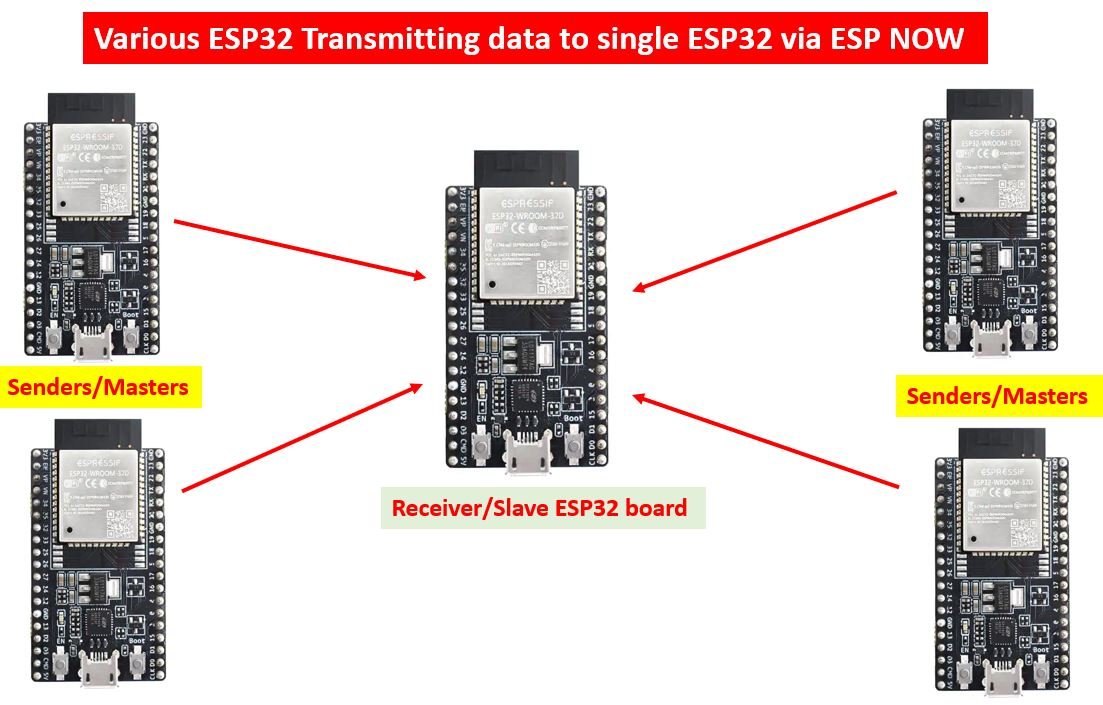
* One ESP32 sender board sends data to various other ESP32 receiver boards

In this scenario, one ESP32 board will act as the sender/master and send data to multiple ESP32 boards that will act as receivers/slaves.  
Uses: remote control



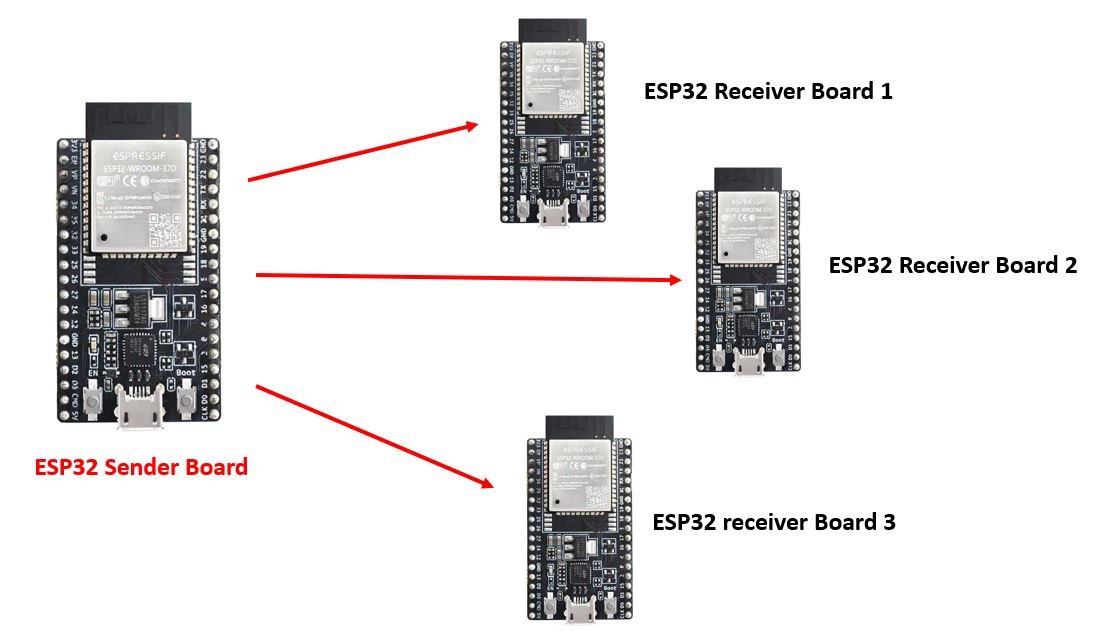
* One ESP32 board receives data from various other ESP32 sender boards

Lastly, in this case, one ESP32 board (receiver/slave) receives data from multiple ESP32 boards (senders/masters).  
Uses: Receiving sensor data from various sensors.



## ESP-NOW One to Many Communication Project Overview

This project consists of four ESP32 boards where one will act as the sender and the other three will be the receivers. Our aim will be to show you how to send data from the sender ESP32 board to the three other receivers’ ESP32/ESP8266 boards. First, we will find the MAC address of each board through an Arduino sketch to differentiate between the four modules. Then, after programming the boards with their respective sender and receiver sketches, receiver boards will start receiving data from the sender ESP32 board through ESP-NOW protocol. You can also use the sketches to transmit useful information between ESP devices including sensor data.

ESP NOW one way communication (one to many configurations)

**Required Hardware:**

4x ESP32 or ESP8266 development boards  
4x power cables

## Setting up Arduino IDE

We will use Arduino IDE to program our ESP32/ESP8266 development board. Thus, you should have the latest version of Arduino IDE. Additionally, you also need to install the ESP32 and the ESP8266 plugin. If your IDE does not have the plugins installed you can visit the links below:

* [Installing ESP32 library in Arduino IDE and upload code.](https://microcontrollerslab.com/install-esp32-arduino-ide/)
* [Installing ESP8266 library in Arduino IDE](https://microcontrollerslab.com/how-to-install-esp8266-board-arduino-ide/)

Now, first, we will introduce you to an Arduino sketch which will help us to identify our ESP board so that we know exactly which board to transmit the data to. It will display our ESP module’s MAC address on the serial monitor which we will later use in another sketch.

## Arduino Sketch for obtaining the MAC Address for the ESP32/ESP8266 board

Open your Arduino IDE and go to **File > New**to open a new file. Copy the code given below in that file and save it.

#ifdef ESP32

#include <WiFi.h>

#else

#include <ESP8266WiFi.h>

#endif

void setup(){

Serial.begin(115200);

Serial.println();

Serial.println(WiFi.macAddress());

}

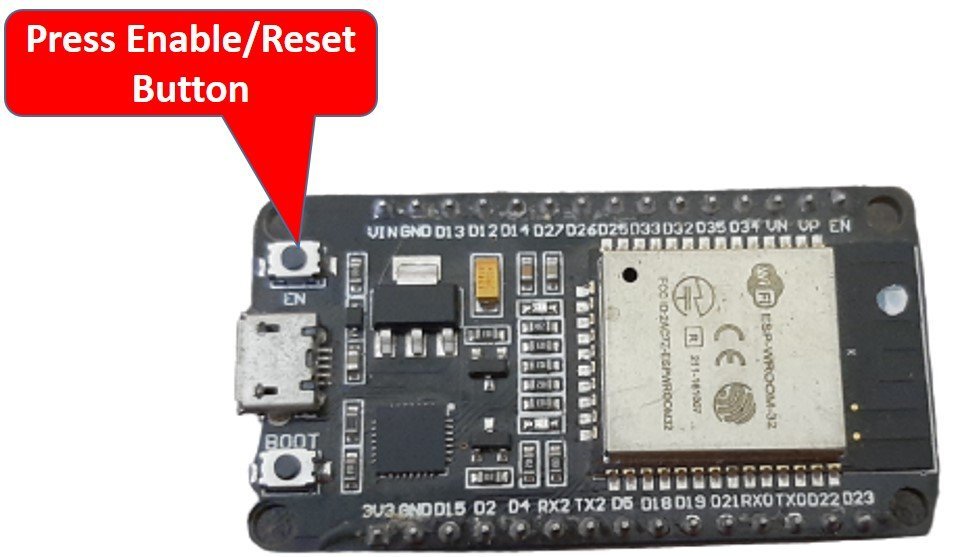
void loop(){

}

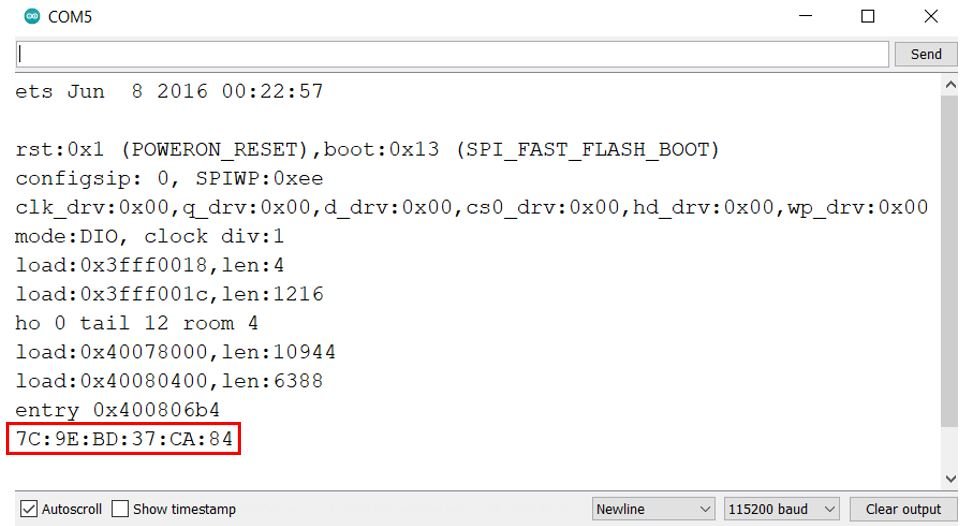
In the setup() function, we are first setting our ESP32/ESP8266 board in station mode. Then by using the WiFi.macAddress() method we will obtain the unique MAC address in our serial monitor.

Make sure you choose the correct board and COM port before uploading your code to the board. Go to Tools > Board and select your board. Next, go to Tools > Port and select the appropriate port through which your board is connected. In this case, we are using the ESP32 development board which is connected to COM5.

Click on the upload button to upload the code into the ESP32 development board. Press its ENABLE button after the sketch has been uploaded.



In your Arduino IDE, open up the serial monitor and you will be able to see the unique MAC address of your ESP32 module.

Serial Monitor

Obtain the MAC address for each ESP board and label it. We will need the unique MAC address of the receiver boards while programming the sender ESP board.

## ESP-NOW ESP32/ESP8266 (One to many) Arduino Sketch

Now, let us learn how to send data from one ESP32/ESP8266 board to another, using the ESP-NOW protocol. We will use the one-to-many configuration, where one ESP32 board acts as the sender and multiple other ESP32 boards act as receivers. You can use the method given below to transmit sensor data or any type of information from one ESP board to another. For this project, we will require two Arduino sketches one for the sender and the other for the receivers.

## ESP-NOW ESP32 Arduino Sketch for Sender Side

Open your Arduino IDE and go to **File > New**to open a new file. Copy the code given below in that file and save it. This sketch follows the points given below:

* Firstly, we will initialize ESP NOW by using esp\_now\_init() function.
* Then we will create a callback function called data\_sent() and register it as a callback function using esp\_now\_register\_send\_cb() . This will return a message in the serial monitor showing whether the data was transmitted successfully or not.
* The next step will be to add the receiver ESP boards by using their unique MAC address.
* We will then send the data to these peer devices that we set up.

#include <esp\_now.h>

#include <WiFi.h>

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS

uint8\_t Receiver\_Address1[] = {0x7C, 0x9E, 0xBD, 0x37, 0x28, 0x4C};,

uint8\_t Receiver\_Address2[] = {0x7C, 0x9E, 0xBD, 0x37, 0xCA, 0x84};

uint8\_t Receiver\_Address3[] = {0x84, 0xCC, 0xA8, 0x5E, 0x52, 0x44};

typedef struct struct\_message {

int integer;

char character[100];

} struct\_message;

struct\_message message;

void data\_sent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

char address[18];

Serial.print("Sent to: ");

snprintf(address, sizeof(address), "%02x:%02x:%02x:%02x:%02x:%02x",

mac\_addr[0], mac\_addr[1], mac\_addr[2], mac\_addr[3], mac\_addr[4], mac\_addr[5]);

Serial.print(address);

Serial.print(" status:\t");

Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");

}

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(data\_sent);

esp\_now\_peer\_info\_t peerInfo;

peerInfo.channel = 0;

peerInfo.encrypt = false;

memcpy(peerInfo.peer\_addr, Receiver\_Address1, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

memcpy(peerInfo.peer\_addr, Receiver\_Address2, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

memcpy(peerInfo.peer\_addr, Receiver\_Address3, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

}

void loop() {

message.integer = random(0,50);

strcpy(message.character, "Welcome to Microcontrollerslab! This is test example.");

esp\_err\_t outcome = esp\_now\_send(0, (uint8\_t \*) &message, sizeof(struct\_message));

if (outcome == ESP\_OK) {

Serial.println("Sent with success");

}

else {

Serial.println("Error sending the data");

}

delay(2000);

}

### How the Code Works?

#### Including Libraries

Firstly, we will include the necessary libraries. For the sender sketch, we are using two of them. These include esp\_now.h for the ESP-NOW communication protocol and WiFi.h will allow our ESP32 board to use the Wi-Fi functionalities.

#include <esp\_now.h>

#include <WiFi.h>

#### Specifying MAC Addresses of Receiver Boards

Secondly, we will specify the MAC Addresses of the three ESP32 boards which will act as the receivers. We will use the same sketch above to find the MAC addresses. Replace the address with the unique MAC addresses of your own ESP32 boards. You can use the sketch which was given previously, to find the MAC addresses of your modules.

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS

uint8\_t Receiver\_Address1[] = {0x7C, 0x9E, 0xBD, 0x37, 0x28, 0x4C};,

uint8\_t Receiver\_Address2[] = {0x7C, 0x9E, 0xBD, 0x37, 0xCA, 0x84};

uint8\_t Receiver\_Address3[] = {0x84, 0xCC, 0xA8, 0x5E, 0x52, 0x44};

#### Defining structure for sending data

Now, we will define a structure named ‘struct\_message.’ Inside the structure, we will initialize the variables which will hold our data that we will transmit to the receiver boards via ESP-NOW. These will be of type int and char.

typedef struct struct\_message {

int integer;

char character[100];

} struct\_message;

Next, we will create a new variable of type struct\_message and call it message. This will be used later on in the sketch to acquire the data and transmit it accordingly.

struct\_message message;

#### data\_sent()

The data\_sent() function acts as the callback function which we will define now. It will be used as a parameter when we will register this callback function for sending messages. This prints whether the message was successfully delivered or not for each ESP board on the serial monitor whenever a message will be sent from the ESP32 sender side.

void data\_sent(const uint8\_t \*mac\_addr, esp\_now\_send\_status\_t status) {

char address[18];

Serial.print("Sent to: ");

snprintf(address, sizeof(address), "%02x:%02x:%02x:%02x:%02x:%02x",

mac\_addr[0], mac\_addr[1], mac\_addr[2], mac\_addr[3], mac\_addr[4], mac\_addr[5]);

Serial.print(address);

Serial.print(" status:\t");

Serial.println(status == ESP\_NOW\_SEND\_SUCCESS ? "Delivery Success" : "Delivery Fail");

}

#### setup()

Inside the setup() function, we will open a serial connection at a baud rate of 115200 and set up the ESP32 board in station mode.

Serial.begin(115200);

WiFi.mode(WIFI\_STA);

The following lines of code will initialize the ESP-NOW protocol. In case of an unsuccessful connection, the serial monitor will display ‘Error initializing ESP-NOW.’

if (esp\_now\_init() != ESP\_OK) {

Serial.println("Error initializing ESP-NOW");

return;

}

Now we will register the data\_sent() function as the callback function. This will make sure that whenever a message will be sent from the sender side, the data\_sent() function will be called.

esp\_now\_register\_send\_cb(data\_sent);

The following lines of code will pair the ESP32 sender board with the three receiver boards.

esp\_now\_peer\_info\_t peerInfo;

peerInfo.channel = 0;

peerInfo.encrypt = false;

memcpy(peerInfo.peer\_addr, Receiver\_Address1, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

memcpy(peerInfo.peer\_addr, Receiver\_Address2, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

memcpy(peerInfo.peer\_addr, Receiver\_Address3, 6);

if (esp\_now\_add\_peer(&peerInfo) != ESP\_OK){

Serial.println("Failed to add peer");

return;

}

#### loop()

Inside the loop() function, we will transmit the message to the receiver ESP32 boards. After every 2 seconds, the following message will be sent:

“Welcome to Microcontrollerslab! This is a test example” as the character and a random integer between 0-50.

You can easily change the structure ‘message’ to send data according to your needs. We have incorporated some of the common data types here.

message.integer = random(0,50);

strcpy(message.character, "Welcome to Microcontrollerslab! This is test example.");

Then we will send the message and monitor if it was sent successfully or not.

esp\_err\_t outcome = esp\_now\_send(0, (uint8\_t \*) &message, sizeof(struct\_message));

if (outcome == ESP\_OK) {

Serial.println("Sent with success");

}

else {

Serial.println("Error sending the data");

}

delay(2000);

## ESP-NOW ESP32 Arduino Sketch for Receiver Side

Open your Arduino IDE and go to **File > New**to open a new file. Copy the code given below in that file and save it. This sketch follows the points given below:

* Again we will first initialize ESP NOW by using esp\_now\_init() function.
* Then, we will create another function called data\_receive() and register it as a callback function using esp\_now\_register\_rcv\_cb(). This callback function will be called whenever the data will be received by the receiver ESP32 board.

#include <esp\_now.h>

#include <WiFi.h>

//Must match the sender structure

typedef struct struct\_message {

int integer;

char character[100];

} struct\_message;

struct\_message message;

void data\_receive(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

memcpy(&message, incomingData, sizeof(message));

Serial.print("Bytes received: ");

Serial.println(len);

Serial.print("Integer: ");

Serial.println(message.integer);

Serial.print("Character: ");

Serial.println(message.character);

Serial.println();

}

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\_recv\_cb(data\_receive);

}

void loop() {

}

### How the Code Works?

#### Including Libraries

Firstly, we will include the necessary libraries. For the receiver sketch, we are also using the same two libraries which we did for the sender sketch. These include esp\_now.h for the ESP-NOW communication protocol and WiFi.h will allow our ESP32 board to use the Wi-Fi functionalities.

#include <esp\_now.h>

#include <WiFi.h>

#### Defining structure for receiving data

Now, we will define the same structure named ‘struct\_message’ which we did for the sender sketch. This structure will be used to receive the data that will be received from the receiver ESP32 board via ESP NOW. Make sure that the structure is the same in both the sketches.

//Must match the sender structure

typedef struct struct\_message {

int integer;

char character[100];

} struct\_message;

Next, we will create a new variable of type struct\_message and call it a message. This will be used later on in the sketch to receive the data.

struct\_message message;

#### data\_receive()

The data\_receive() function acts as the callback function which we will define now. It will be used as a parameter when we will register this callback function for receiving messages. This prints the message on the serial monitor whenever a message is received from the ESP32 sender side.

void data\_receive(const uint8\_t \* mac, const uint8\_t \*incomingData, int len) {

memcpy(&message, incomingData, sizeof(message));

Serial.print("Bytes received: ");

Serial.println(len);

Serial.print("Integer: ");

Serial.println(message.integer);

Serial.print("Character: ");

Serial.println(message.character);

Serial.println();

}

#### setup()

Inside the setup() function, we will open a serial connection at a baud rate of 115200 and set up the ESP32 receiver board in station mode as well.

Serial.begin(115200);

WiFi.mode(WIFI\_STA);

The following lines of code will initialize the ESP-NOW protocol. In case of an unsuccessful connection, the serial monitor will display ‘Error initializing ESP-NOW.’

if (esp\_now\_init() != ESP\_OK) {

Serial.println("Error initializing ESP-NOW");

return;

}

Now we will register the data\_receive() function as the callback function as shown below. This will make sure that whenever a message will be received from the sender side, the data\_receive() function will be called.

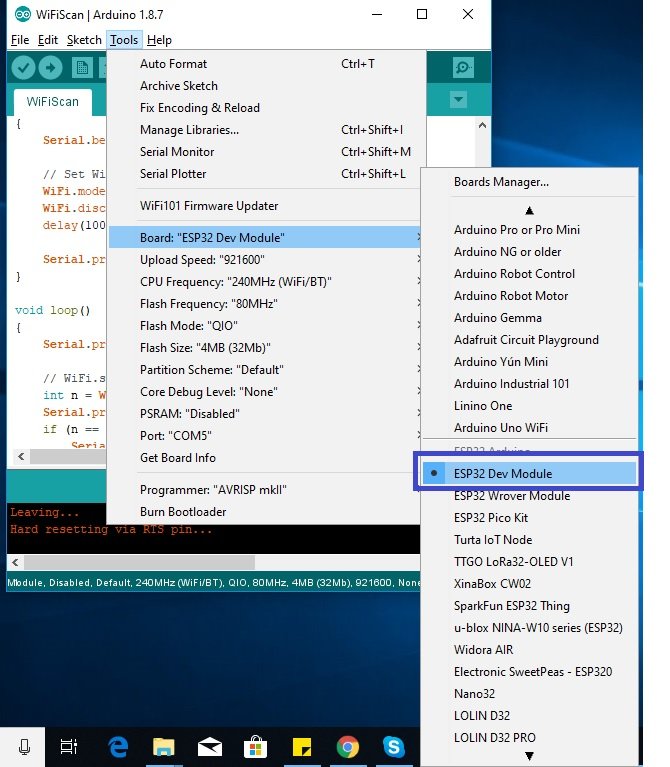
esp\_now\_register\_recv\_cb(data\_receive);

### Demonstration ESP32 ESP-NOW one way communication (one to many)

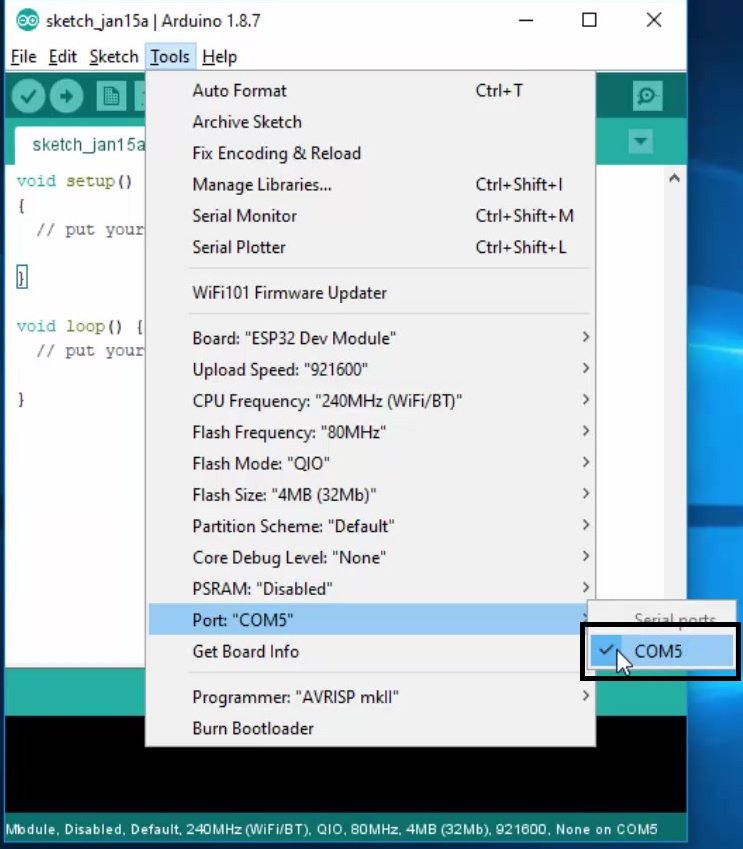
Now after saving both of the sender and receiver sketches upload them onto their respective ESP32 boards. Make sure all the four boards are powered on throughout the demonstration.



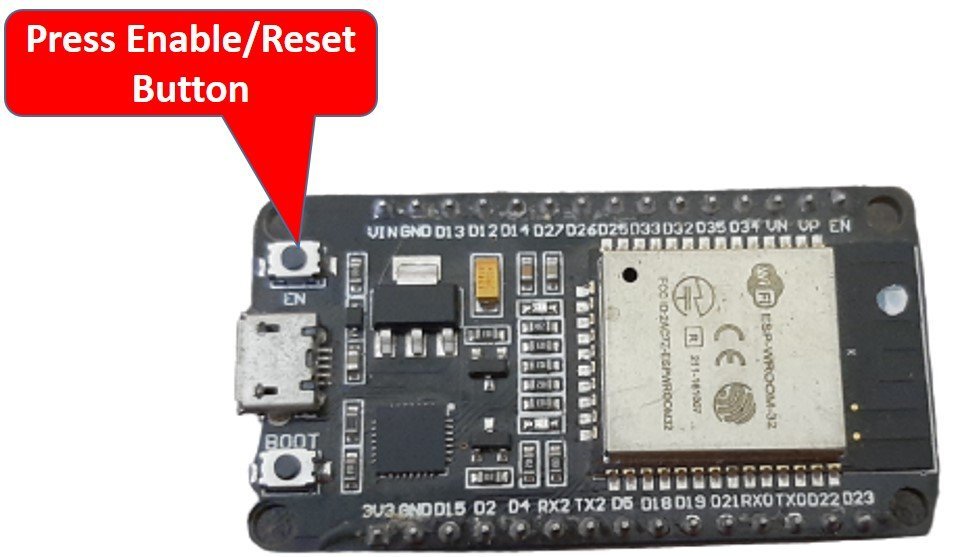
First, open the sender sketch. Choose the correct board and COM port before uploading your code to the sender ESP32 board. Go to Tools > Board and select ESP32 Dev Module.



Next, go to Tools > Port and select the appropriate port through which your board is connected.

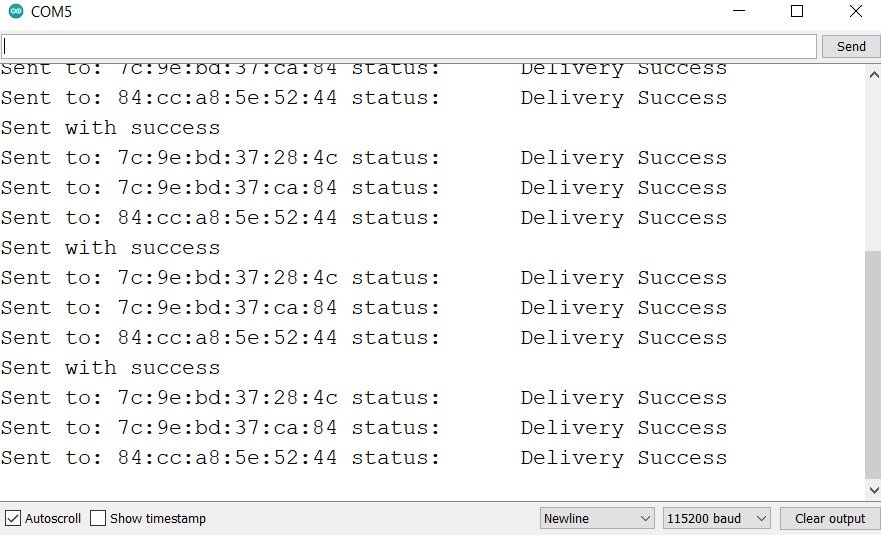


Click on the upload button to upload the code into the ESP32 board (sender). After you have uploaded your code to the board press its ENABLE button.

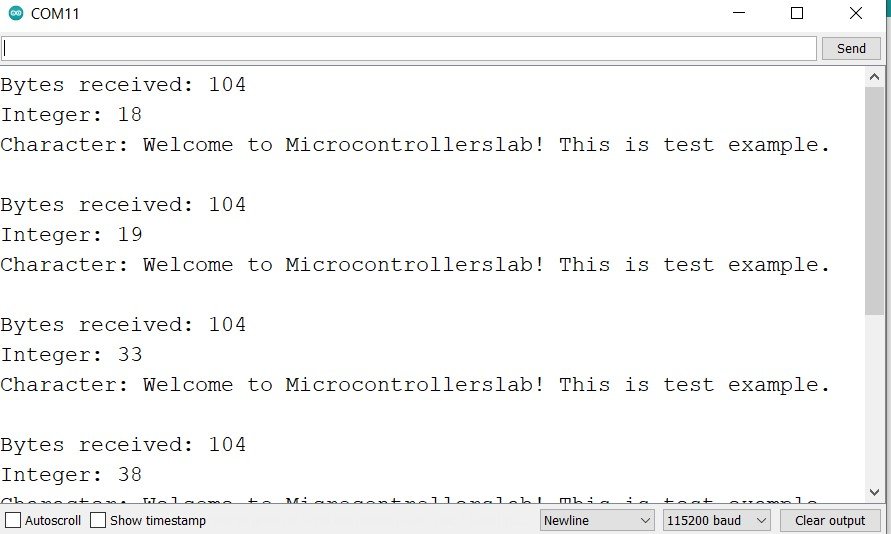


Next, follow the same steps and upload the receiver side sketch to the three receiver ESP32 modules. Make sure you choose the correct COM port through which they are connected. After you have uploaded your code to the board press its ENABLE button.

In the sender side’s serial monitor you will be able to view the text that the message was successfully delivered against each MAC address of the receiver side ESP32 boards.

ESP32 Sender side serial monitor

Now, open one of the receiver side serial monitor. You will be able to see the messages being displayed on the receiver side after every 2 seconds. These include the char and the int parameters which we specified in the program sketch.

ESP32 Receiver side serial monitor

## Conclusion

In conclusion, we have learned about the ESP-NOW one-to-many communication configuration. As an example, we sent a small packet of data from one ESP32 board to multiple ESP32 boards without using any WiFi or internet connection. You can use the same sketch to transfer useful data including sensor readings or even control the output pins of the ESP32 boards through another ESP32 board easily. Although, we have shown you ESP NOW one way communication( one to many configurations). But you can also use ESP NOW two-way communication as well to promptly transfer data two ways.

You may also like to read:

* [ESP32 ESP-NOW Getting Started Tutorial with Arduino IDE](https://microcontrollerslab.com/esp32-esp-now-tutorial-arduino-ide/)
* [ESP32 ESP-NOW Two way Communication (Arduino IDE)](https://microcontrollerslab.com/esp32-esp-now-two-way-communication-arduino-ide/)
* [ESP32 ESP-NOW Receive Data from Multiple boards (Many to One Communication)](https://microcontrollerslab.com/esp32-esp-now-receive-data-from-multiple-boards-many-to-one/)
* [ESP32 ESP-NOW and Wi-Fi Web Server using Arduino IDE](https://microcontrollerslab.com/esp32-esp-now-wi-fi-web-server/)
* [ESP32 ESP-NOW Send and Receive Encrypted Messages](https://microcontrollerslab.com/esp32-esp-now-encrypted-messages/)