Part 09
-
ESP-NOW

Version: 2021-01-09

Getting Started with ESP-NOW

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.
The communication range between the two boards is up to 200 meters in open field with antennas were pointing to each other.

Introducing ESP-NOW
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 ESP32 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 ESP32 board sending data to another ESP32 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.

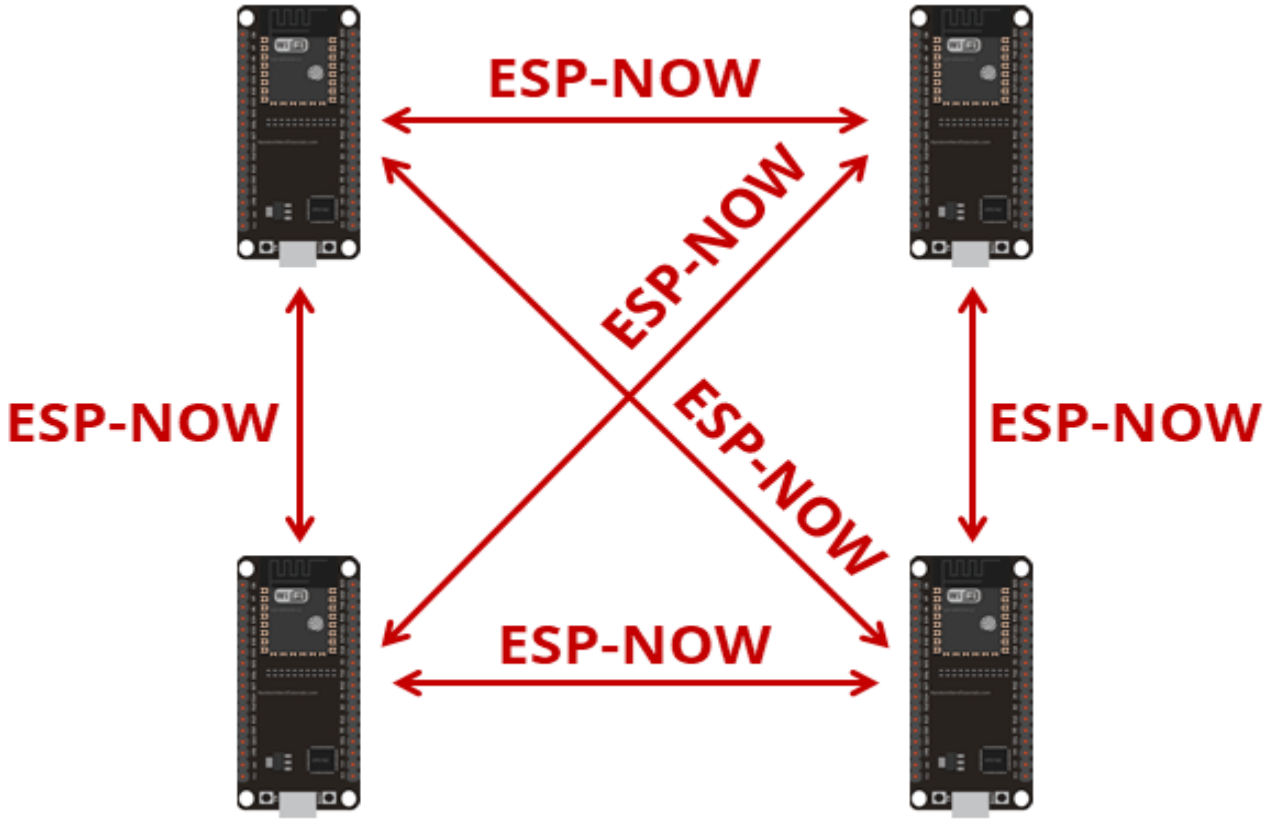
A "master" ESP32 sending data to multiple ESP32 "slaves"
One ESP32 board sending the same or different commands to different ESP32 boards. This configuration is ideal to build something like a remote control. You can have several ESP32 boards around the house that are controlled by one main ESP32 board.

One ESP32 "slave" receiving data from multiple "masters"
This configuration is ideal if you want to collect data from several sensors nodes into one ESP32 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 ESP32 boards communicate with each other).



In summary, ESP-NOW is ideal to build a network in which you can have several ESP32 boards exchanging data with each other.

ESP-NOW Libray and Useful Functions
There is no need to install a libray to have ESP-NOW available in your code. It is by default installed when you install ESP library and tools in Arduino IDE.

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

- `esp_now_init()`
  Initializes ESP-NOW. You must initialize Wi-Fi before initializing ESP-NOW.

- `esp_now_add_peer()`
  Call this function to pair a device and pass as argument the peer MAC address.

- `esp_now_send()`
  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 read the ESP-NOW documentation : https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html

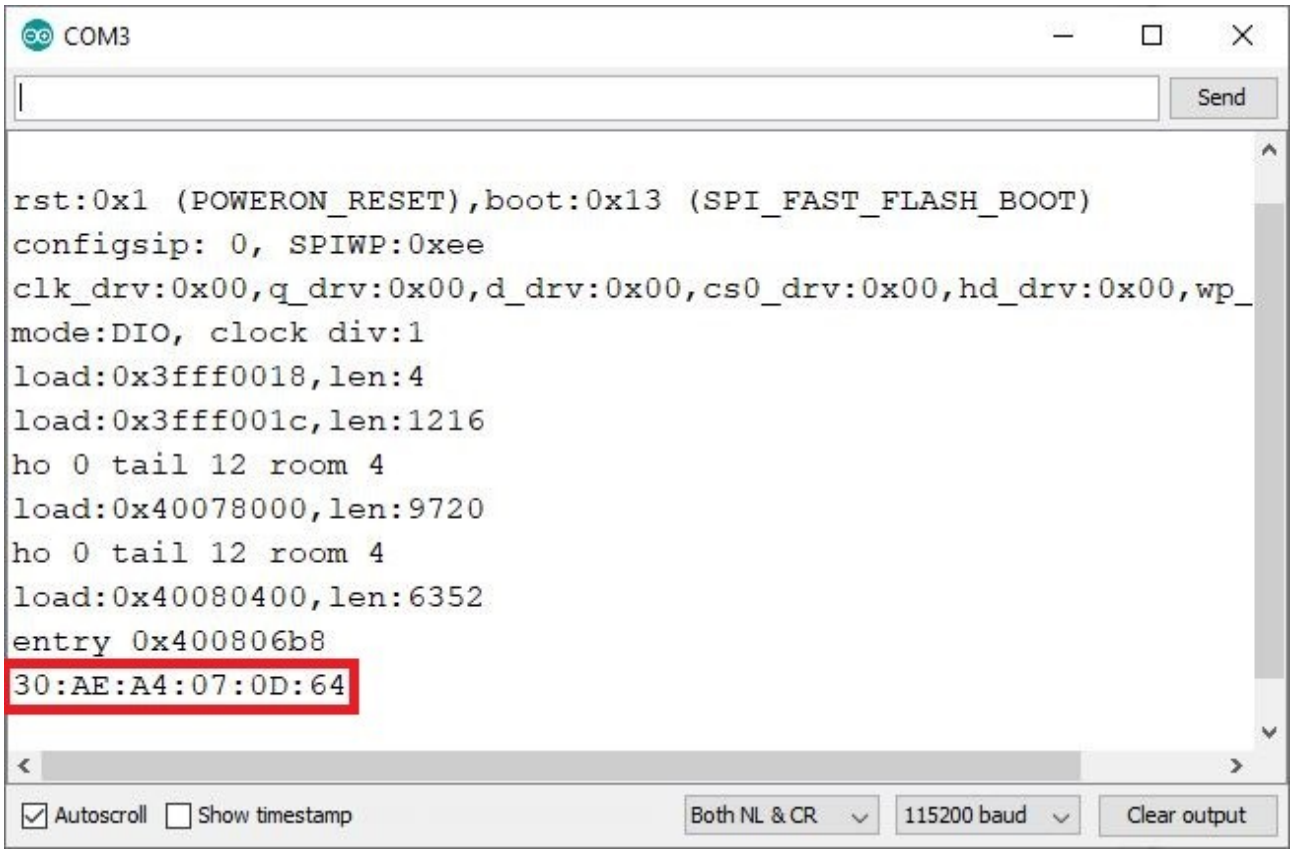
Examples
For a complete examples, in your Arduino IDE, you can go to `File > Examples > ESP32 > ESPNow` and choose one of the example sketches.

Getting Board MAC Address
To communicate via ESP-NOW, you need to know the MAC Address of the ESP32 receiver. That's how you know to which device you'll send the information to.
Each ESP32 has a unique MAC Address and that's how we identify each board to send data to it using ESP-NOW.

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

```
#include "WiFi.h"

void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
}

void loop()
{
}
```

After uploading the code, open the Serial Monitor at a baud rate of 115200 and press the ESP32 RST/EN 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 Point-to-Point One-way Communication

To get you started with ESP-NOW wireless communication, we'll build a simple project that shows how to send a message from one ESP32 to another. One ESP32 will be the "sender" and the other ESP32 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).
We'll use the boolean in the structure to switch on or off the on-board LED.
For better understanding we'll call "sender" to ESP32 #1 and "receiver" to ESP32 #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).

Important note: The slave can not figure out if the master is still active or not.

ESP32 Sender

Here's the code for the ESP32 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.

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

// REPLACE WITH YOUR RECEIVER MAC ADDRESS
uint8_t MACaddress[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00};

// Structure example to send data
// Sender and receiver must use same structure
typedef struct struct_message
{
  char a[32];
  int b;
  float c;
  String d;
  bool e;
} struct_message;

// Create a struct_message called outData
struct_message outData;

//define PINs used - GPIO mode
#define LED 2  // build-in LED

// callback when data is sent
void OnDataSent(const uint8_t *mac, esp_now_send_status_t status)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet send to ");
  // make printable format of MAC address supplied
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.print(sMac);
  // print result status
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? " -> Delivery successful" :
" -> Delivery failed");
  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("Error initializing ESP-NOW");
    return;
  }

  // Register for Send callback to get the status of transmitted packet
```

```
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  memcpy(peerInfo.peer_addr, MACaddress, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // use buildin LED to show activity
  pinMode(LED, OUTPUT);

}

void loop()
{
  static int ledState = LOW;

  // Set values to send
  strcpy(outData.a, "THIS IS A CHAR");
  outData.b = random(1,20);
  outData.c = 1.2;
  outData.d = "Hello";

  // if the LED is off turn it on and vice-versa:
  ledState = (ledState == LOW) ? HIGH : LOW;
  if (ledState == HIGH)
  {
    outData.e = true;
  }
  else
  {
    outData.e = false;
  }
  digitalWrite(LED, ledState);

  // Send message via ESP-NOW
  Serial.println("Sending data");
  esp_err_t result = esp_now_send(MACaddress, (uint8_t *)&outData,
sizeof(outData));
  if (result == ESP_OK)
  {
    Serial.println("Sent data successful");
  }
  else
  {
    Serial.println("Sent data failed");
  }

  delay(5000);
}
```

How the code works
First, include the esp_now.h and WiFi.h libraries.

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

In the next line, you should insert the ESP32 receiver MAC address.

```
uint8_t MACaddress[] = {0x30, 0xAE, 0xA4, 0x07, 0x0D, 0x64};
```

In our case, the receiver MAC address is: `30:AE:A4:07:0D:64`, but you need to replace that variable with your own MAC address.

Note: when you set the MAC address to `{0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF}`, which is the broadcast address, any ESP32 with the receiver code loaded and in range will react on the message sent. However, the sender will consider the any message sent as delivered successful even if no receiver is listening nor replying.
Do some testing with this.

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;
```

Then, create a new variable of type struct_message that is called outData that will store the variables values and set the GPIO value to control the onboard LED

```
struct_message outData;

//define PINs used - GPIO mode
#define LED 2 // build-in LED
```

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

```
void OnDataSent(const uint8_t *mac, esp_now_send_status_t status)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet send to ");
  // make printable format of MAC address supplied
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.print(sMac);
  // print result status
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? " -> Delivery successful" :
" -> Delivery failed");
  Serial.println();
}
```

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() != ESP_OK) {
  Serial.println("Error initializing ESP-NOW");
  return;
}
```

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);
```

After that, we need to pair with another ESP-NOW device to send data. That's what we do in the next lines:

```
//Register peer
esp_now_peer_info_t peerInfo;
peerInfo.channel = 0;
peerInfo.encrypt = false;

//Add peer
memcpy(peerInfo.peer_addr, MACaddress, 6);
if (esp_now_add_peer(&peerInfo) != ESP_OK){
  Serial.println("Failed to add peer");
  return;
}
```

And then we set the GPIO pin, as output to allow to control the buildin LED

```
// use buildin LED to show activity
pinMode(LED, OUTPUT);
```

In the loop(), we'll send a message via ESP-NOW every 5 seconds (you can change this delay time).

First, we set the variables values as follows:

```
strcpy(outData.a, "THIS IS A CHAR");
outData.b = random(1,20);
outData.c = 1.2;
outData.d = "Hello";
```

And we set change the state of the LED and adjust the boolean value accordingly

```
// if the LED is off turn it on and vice-versa:
ledState = (ledState == LOW) ? HIGH : LOW;
if (ledState == HIGH)
{
  outData.e = true;
}
else
{
  outData.e = false;
}
digitalWrite(LED, ledState);
```

Remember that `outData` 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_err_t result = esp_now_send(MACaddress, (uint8_t *)&outData,
sizeof(outData));
```

Check if the message was successfully sent:

```
if (result == ESP_OK)
{
  Serial.println("Sent data successful");
}
else
{
  Serial.println("Sent data failed");
}
```

The loop() is executed every 5000 milliseconds (5 seconds).

```
Delay(5000);
```

ESP32 Receiver
Upload the following code to your ESP32 receiver board.

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

// Structure example to recive data
// Sender and receiver must use same structure
typedef struct struct_message
{
    char a[32];
    int b;
    float c;
    String d;
    bool e;
} struct_message;

// Create a struct_message called inData
struct_message inData;

//define PINs used - GPIO mode
#define LED  2 // build-in LED

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet received from ");
  // make printable format of MAC address supplied
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.println(sMac);

  memcpy(&inData, incomingData, sizeof(inData));
  Serial.print("Bytes received: "); Serial.println(len);
  Serial.print("Char: "); Serial.println(inData.a);
  Serial.print("Int: "); Serial.println(inData.b);
  Serial.print("Float: "); Serial.println(inData.c);
  Serial.print("String: "); Serial.println(inData.d);
  Serial.print("Bool: "); Serial.println(inData.e);
  Serial.println();

  // set buildin LED based on boolean received
  if (inData.e)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
}

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;
  }

  // use buildin LED to show activity
  pinMode(LED, OUTPUT);

  // Register for Recv callback to get the data is received
  esp_now_register_recv_cb(OnDataRecv);

}

void loop()
{

}
```

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 inData and define the pin for the onboard LED

```
struct_message inData;

//define PINs used - GPIO mode
#define LED  2 // build-in LED
```

Create a callback function that will be called when the ESP32 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 inData variable.

```
memcpy(&inData, incomingData, sizeof(inData));
```

Now, the `inData` structure contains several variables inside with the values sent by the sender ESP32. To access variable a, for example, we just need to call `inData.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(inData.a);
  Serial.print("Int: "); Serial.println(inData.b);
  Serial.print("Float: "); Serial.println(inData.c);
  Serial.print("String: "); Serial.println(inData.d);
  Serial.print("Bool: "); Serial.println(inData.e);
  Serial.println();
```

And then we set the onboard LED based on the boolean varaible

```
  if (inData.e)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
}
```

In the setup(), intialize the Serial Monitor.

```
Serial.begin(115200);
```

Set the device as a Wi-Fi Station.

```
WiFi.mode(WIFI_STA);
```

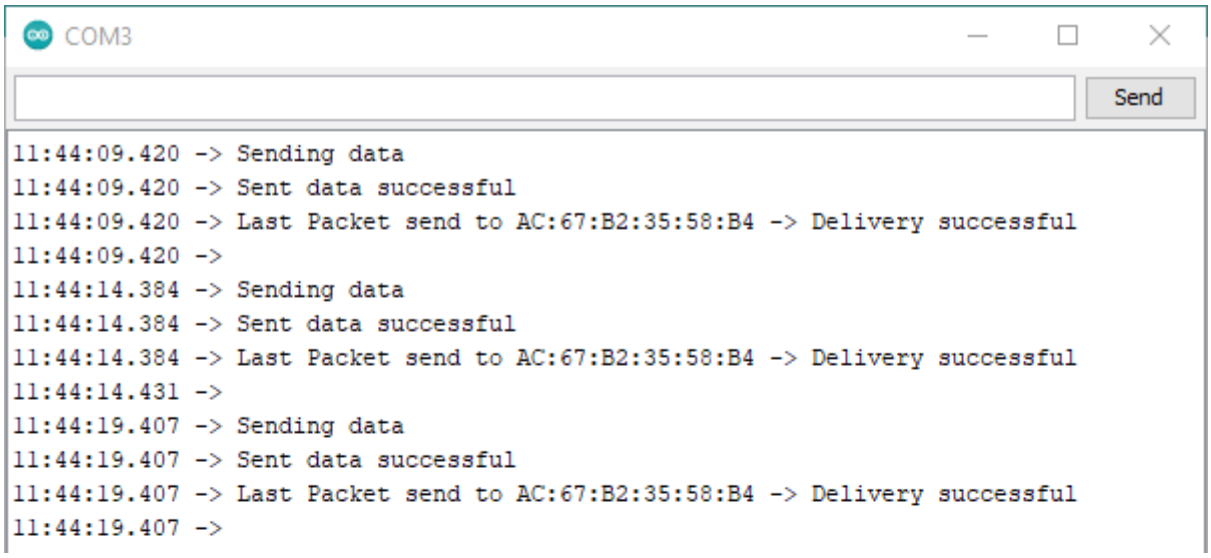
Initialize ESP-NOW:

```
if (esp_now_init() != ESP_OK)
{
  Serial.println("Error initializing ESP-NOW");
  return;
}
```

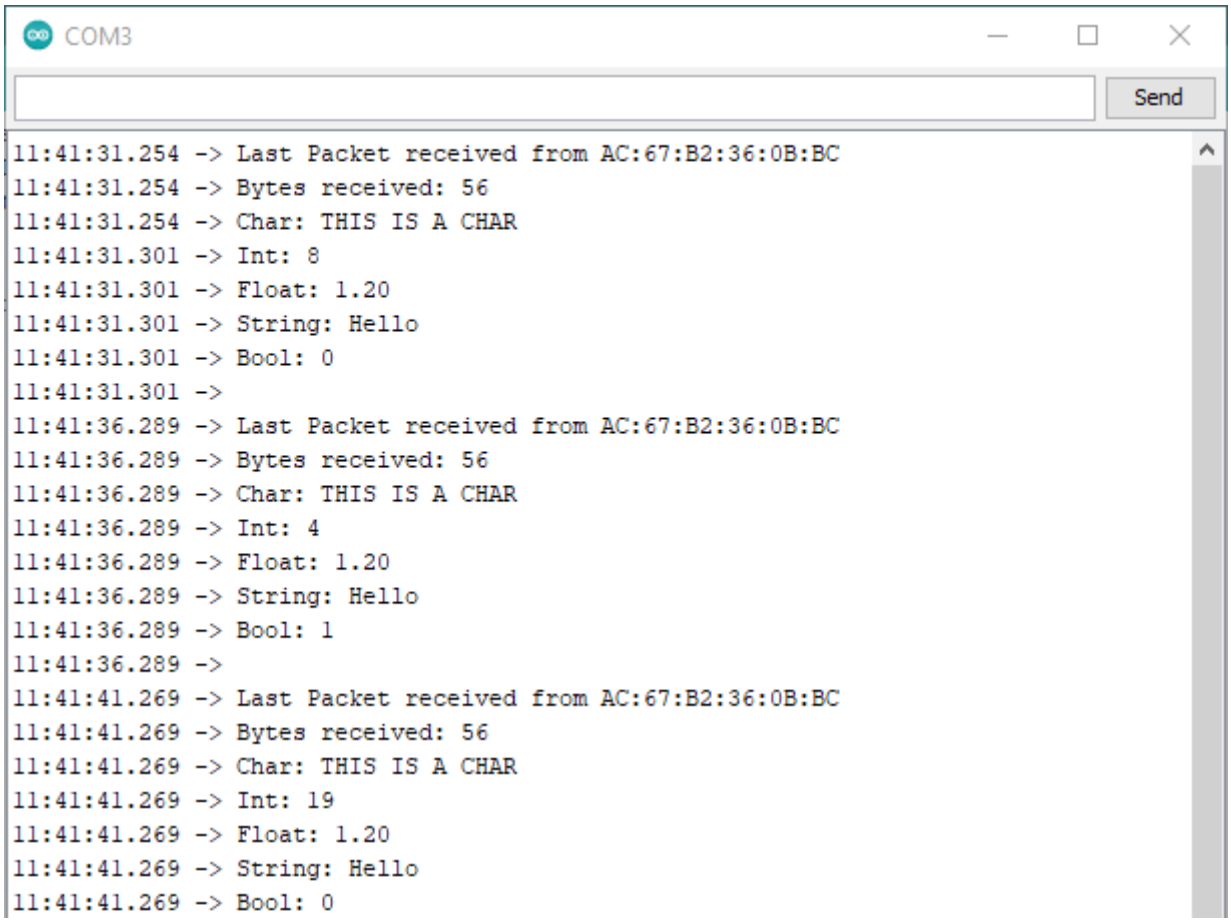
Register for a callback function that will be called when data is received. In this case, we register for the OnDataRecv() function that was created previously.

```
esp_now_register_recv_cb(OnDataRecv);
```

Testing
Upload the sender sketch to the sender ESP32 board and the receiver sketch to the receiver ESP32 board.
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.

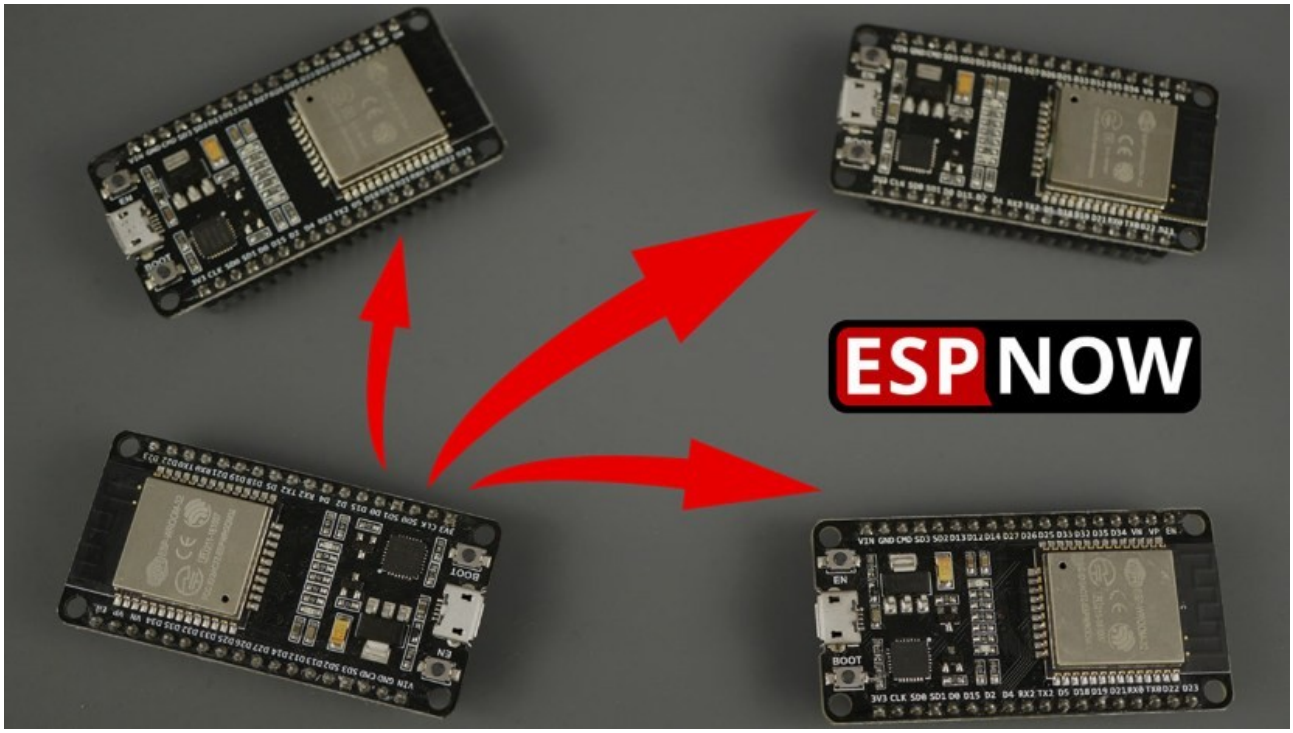


And this is what you should get on the receiver side. Note that the `int` and `boolean` variable changes between each reading received because we set it to a random number or toggle it in the sender side.



ESP-NOW One-To-Many One-way Communication

Use ESP-NOW communication protocol to send data from one ESP32 to multiple ESP32 (one-to-many configuration).



Important note: No slave can not figure out if the master is still active or not.

One ESP32 acts as a sender, multiple ESP32 boards act as receivers. We tested this setup with three ESP32 boards simultaneously. You should be able to add more boards to your setup. The ESP32 sender receives an acknowledge message if the messages are successfully delivered. You know which boards received the message and which boards didn't.

Two scenarios:

- sending the same message to all boards
- sending a different message to each board

ESP32 Sender
The following code sends data to multiple (three) ESP boards via ESP-NOW. You should modify the code with your receiver boards' MAC address. You should also add or delete lines of code depending on the number of receiver boards.
We'll use about the same code as before but make slight changes

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

// REPLACE WITH YOUR RECEIVER MAC Address
uint8_t MACaddress1[] = {0xAC, 0x67, 0xB2, 0x35, 0x58, 0xB4};
uint8_t MACaddress2[] = {0xAC, 0x67, 0xB2, 0x37, 0x8E, 0x3C};
uint8_t MACaddress3[] = {0xAC, 0x67, 0xB2, 0x37, 0x1B, 0xC4};

// 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 outData
struct_message outData;

//define PINs used - GPIO mode
#define LED 2  // build-in LED

// callback when data is sent
void OnDataSent(const uint8_t *mac, esp_now_send_status_t status)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet send to ");
  // make printable format of MAC address supplied
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.print(sMac);
  // print result status
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? " -> Delivery successful" :
" -> Delivery failed");
  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("Error initializing ESP-NOW");
```

```
    return;
  }

  // Register for Send callback to get the status of transmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer 1
  memcpy(peerInfo.peer_addr, MACaddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // Add peer 2
  memcpy(peerInfo.peer_addr, MACaddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // Add peer 3
  memcpy(peerInfo.peer_addr, MACaddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // use bluidinLED to show activity
  pinMode(LED, OUTPUT);

}

void loop()
{
  static int ledState = LOW;

  // Set values to send
  strcpy(outData.a, "THIS IS A CHAR");
  outData.b = random(1,20);
  outData.c = 1.2;
  outData.d = "Hello";

  // if the LED is off turn it on and vice-versa:
  ledState = (ledState == LOW) ? HIGH : LOW;
  if (ledState == HIGH)
  {
    outData.e = true;
  }
  else
  {
    outData.e = false;
  }
  digitalWrite(LED, ledState);

  // Send message via ESP-NOW to all
  Serial.println("Sending data");
```

```
  esp_err_t result = esp_now_send(0, (uint8_t *)&outData, sizeof(outData));
  if (result == ESP_OK)
  {
    Serial.println("Sent with success");
  }
  else
  {
    Serial.println("Error sending the data");
  }

  delay(5000);

}
```

How the code works
I'll just go over the changes as the rest is the same as above

We create receivers' MAC address. In our example, we're sending data to three boards. Make sure you chage these addresses to match with yours.
Note: you could also create an array of array but for improve reading and simplicity, I have chosen for extra code lines.

```
uint8_t MACaddress1[] = {0xAC, 0x67, 0xB2, 0x35, 0x58, 0xB4};
uint8_t MACaddress2[] = {0xAC, 0x67, 0xB2, 0x36, 0x0B, 0xBC};
uint8_t MACaddress3[] = {0xAC, 0x67, 0xB2, 0x37, 0x1B, 0xC4};
```

In the `setup` we add our ESP-NOW devices to send data. That's what we do in the next lines – register peers:

```
// Register peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer 1
  memcpy(peerInfo.peer_addr, MACaddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // Add peer 2
  memcpy(peerInfo.peer_addr, MACaddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // Add peer 3
  memcpy(peerInfo.peer_addr, MACaddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }
```

In the `loop()`, we'll send a message via ESP-NOW every 5 seconds. Send the same data to multiple boards as follows:

```
esp_err_t result = esp_now_send(0, (uint8_t *)&outData, sizeof(outData));
```

The first argument of the `esp_now_send()` function is the receiver's MAC address. If you pass 0 as an argument, it will send the same message to all registered peers.

Send different data to each board

The code to send a different message to each board is very similar with the previous one. So, we'll just take a look at the differences.

Change your data for each ESP device and send it to that specific device by calling the `esp_now_send()` function for each receiver.

For example, send the structure to the board whose MAC address is `MACaddress1`.

```
// Set values to send
strcpy(outData.a, "ESP32-1");
outData.b = random(1,20);
outData.c = 1.2;
outData.d = "Hello 1";

// if the LED is off turn it on and vice-versa:
ledState = (ledState == LOW) ? HIGH : LOW;
if (ledState == HIGH)
{
  outData.e = true;
}
else
{
  outData.e = false;
}
digitalWrite(LED, ledState);

esp_err_t result = esp_now_send(MACaddress1, (uint8_t *)&outData,
  sizeof(outData));
if (result == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

Do the same for the other board

```
// Set values to send
strcpy(outData.a, "ESP32-2");
outData.b = random(1,20);
outData.c = 1.2;
outData.d = "Hello 2";

// if the LED is off turn it on and vice-versa:
ledState = (ledState == LOW) ? HIGH : LOW;
if (ledState == HIGH)
{
  outData.e = true;
}
else
{
  outData.e = false;
}
digitalWrite(LED, ledState);

esp_err_t result = esp_now_send(MACaddress2, (uint8_t *)&outData,
```

```
  sizeof(outData));
if (result == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

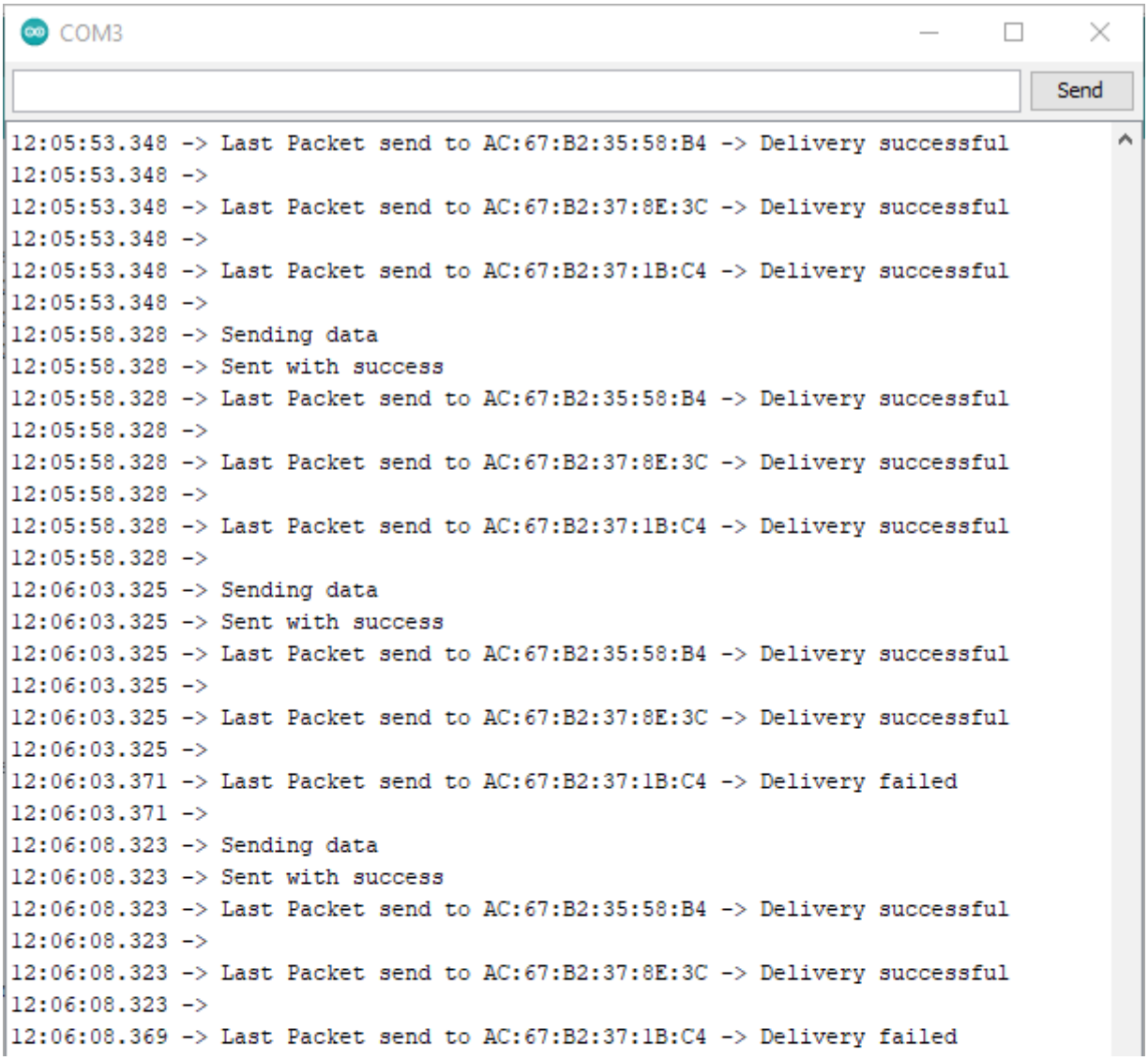
And finally, for the third board:

```
// Set values to send
strcpy(outData.a, "ESP32-3");
outData.b = random(1,20);
outData.c = 1.2;
outData.d = "Hello 3";

// if the LED is off turn it on and vice-versa:
ledState = (ledState == LOW) ? HIGH : LOW;
if (ledState == HIGH)
{
  outData.e = true;
}
else
{
  outData.e = false;
}
digitalWrite(LED, ledState);

esp_err_t result = esp_now_send(MACaddress3, (uint8_t *)&outData,
  sizeof(outData));
if (result == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

Note: As already said before putting all things into arrays will make the code easy to expand or reduce as well as less extensive if more nodes are added.

ESP32 Receiver
The code for the receiver is exactly the same as in the 1-to-1 set-up

Testing
Having all your boards powered on, open the Arduino IDE Serial Monitor for the COM port the sender is connected to. You should start receiving "`Delivery Success`" messages with the corresponding receiver's MAC address in the Serial Monitor.
If you remove power from one of the boards, you'll receive a "`Delivery Fail`" message for that specific board. So, you can quickly identify which board didn't receive the message.



If you want to check if the boards are actually receiving the messages, you can open the Serial Monitor for the COM port they are connected to, or you can use PuTTY to establish a serial communication with your boards.

ESP-NOW Point-to-Point Two-Way Communication

We'll show how to establish a two-way communication between two ESP32 boards using ESP-NOW communication protocol.

ESP32 code

Upload the following sender and receiver code to each of your boards. Before uploading the code, you need to enter the MAC address of the other board (the board you're sending data to).

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

// REPLACE WITH YOUR RECEIVER MAC Address
//uint8_t MACaddress[] = {0xAC, 0x67, 0xB2, 0x35, 0x58, 0xB4};
uint8_t MACaddress[] = {0xAC, 0x67, 0xB2, 0x36, 0x0B, 0xBC};

// Structure example to send data
typedef struct struct_message
{
  char a[32];
  int b;
  float c;
  String d;
  bool e;
} struct_message;

// Create a struct_message called outData = data going out
struct_message outData;

// Create a struct_message called inData = data coming in
struct_message inData;

//define PINs used - GPIO mode
#define LED 2  // build-in LED

// ledState used to set the LED
int ledState = LOW;

// Callback when data is sent
void OnDataSent(const uint8_t *mac, esp_now_send_status_t status)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet send to ");
  // make printable format of MAC address supplied
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.print(sMac);
  // print result status
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? " -> Delivery successful" :
" -> Delivery failed");
  Serial.println();
}

// Callback when data is received
void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len)
{
  char macStr[18];
  String sMac;

  Serial.print("Last Packet received from ");
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
mac[1], mac[2], mac[3], mac[4], mac[5]);
```

```
  macStr[17] = '\0';
  sMac = String(macStr);
  sMac.toUpperCase();
  Serial.println(sMac);

  memcpy(&inData, incomingData, sizeof(inData));
  Serial.print("  Bytes received: "); Serial.println(len);
  Serial.print("  Char  : "); Serial.println(inData.a);
  Serial.print("  Int   : "); Serial.println(inData.b);
  Serial.print("  Float : "); Serial.println(inData.c);
  Serial.print("  String: "); Serial.println(inData.d);
  Serial.print("  Bool  : "); Serial.println(inData.e);
  Serial.println();

  // set buildin LED based on boolean received
  if (inData.e)
  {
    digitalWrite(LED, HIGH);
  }
  else
  {
    digitalWrite(LED, LOW);
  }
}

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;
  }

  // Register for Send callback to get the status of transmitted packet
  esp_now_register_send_cb(OnDataSent);

  // Register for Recv callback to get the data is received
  esp_now_register_recv_cb(OnDataRecv);

  // Register peer
  esp_now_peer_info_t peerInfo;
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  // Add peer
  memcpy(peerInfo.peer_addr, MACaddress, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("Failed to add peer");
    return;
  }

  // use bluidinLED to show activity
  pinMode(LED, OUTPUT);

}
```

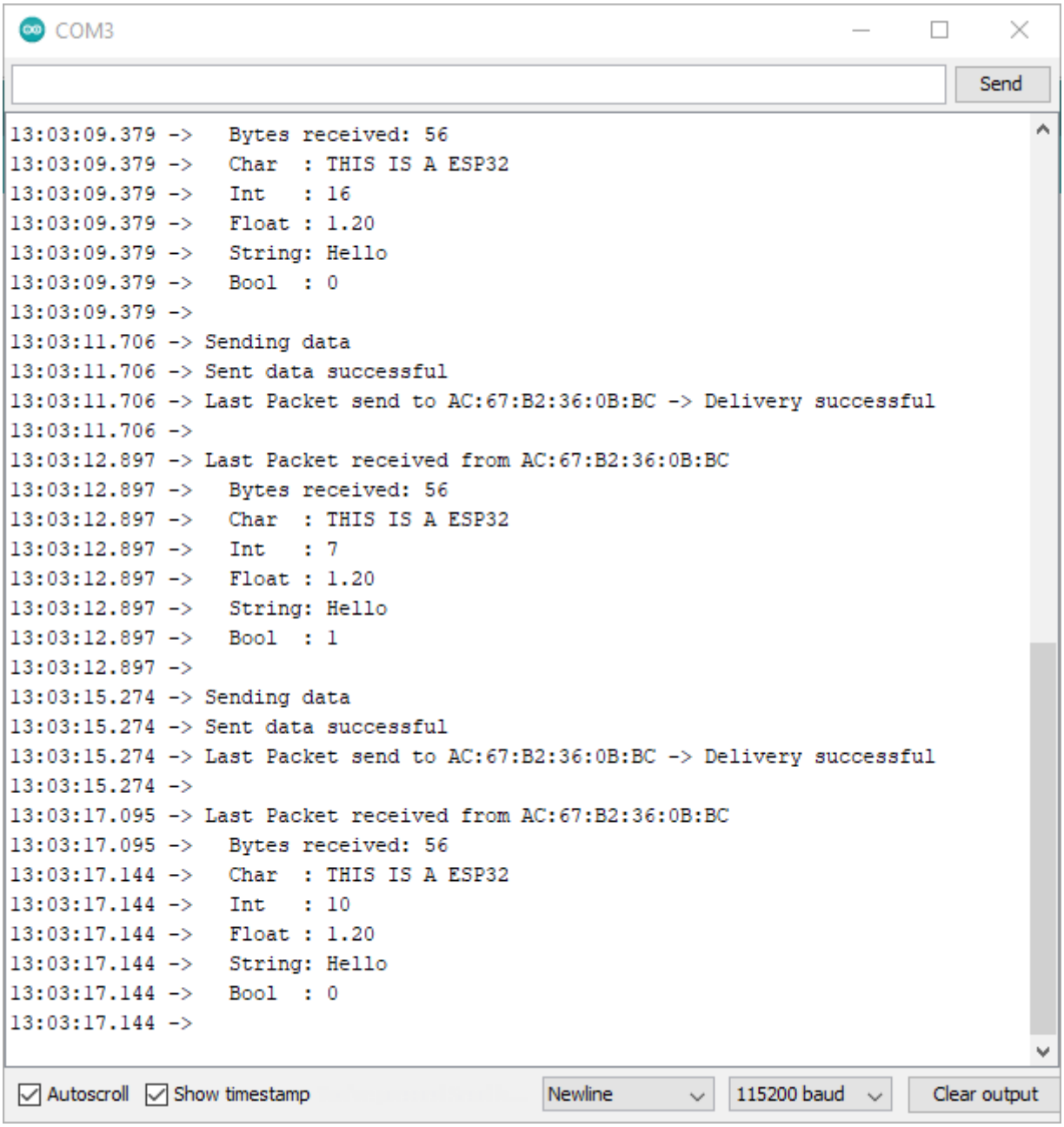
```
void loop()
{
  static int ledState = LOW;

  // Set values to send
  strcpy(outData.a, "THIS IS A ESP32");
  outData.b = random(1,20);
  outData.c = 1.2;
  outData.d = "Hello";

  // toggle state
  ledState = (ledState == LOW) ? HIGH : LOW;
  if (ledState == HIGH)
  {
    outData.e = true;
  }
  else
  {
    outData.e = false;
  }

  // Send message via ESP-NOW
  Serial.println("Sending data");
  esp_err_t result = esp_now_send(MACaddress, (uint8_t *) &outData,
sizeof(outData));
  if (result == ESP_OK)
  {
    Serial.println("Sent data successful");
  }
  else
  {
    Serial.println("Sent data failed");
  }

  delay(random(3000,5000));

}
```

How the code works

The code is a combination of the previous sender 1-to-1 and the receiver code which we listed and explained above.

A small difference is that we now randomly make a delay between 3 and 5 seconds.

Testing
After uploading the code to both boards, you should see the data sent from the other board, as well as a success delivery message.



ESP-NOW Many-To-One One-Way Communication

How to setup an ESP32 board to receive data from multiple ESP32 boards via ESP-NOW communication protocol (many-to-one configuration). This configuration is ideal if you want to collect data from several sensors nodes into one ESP32 board.

One ESP32 board acts as a receiver/slave. Multiple ESP32 boards act as senders/masters.

ESP32 code
Using the 1-to-1 sender code as well as the reciever code will allow you to set-up this configuration. In the receiver code, you can distungish based on the MACaddress from which device which data was sent.

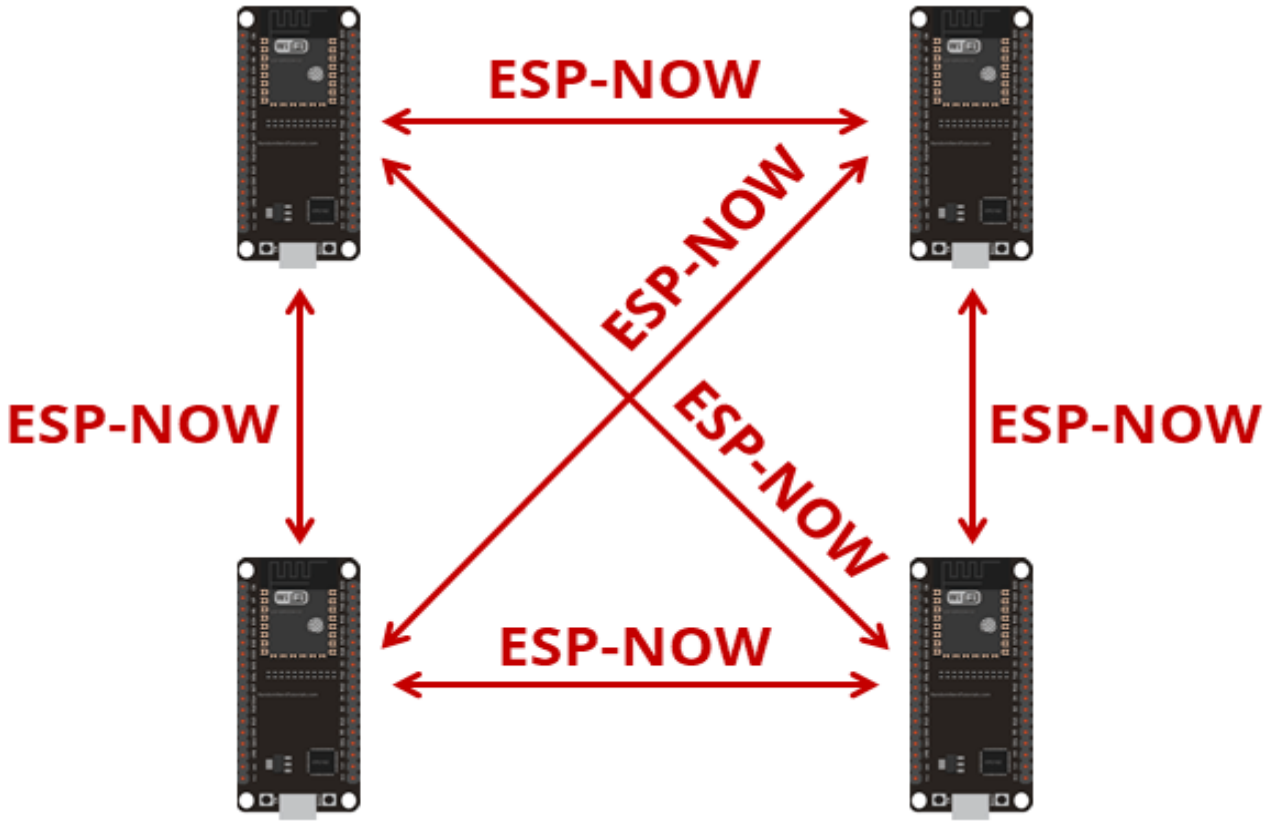
Note: you might consider using device ID's eg: D1, D2, D3, … being sent in the data transmitted to ease the identification of the device

Testing

Upload the sender code to each of your sender boards. Upload the receiver code to the ESP32 receiver board. On the senders' Serial Monitor, you should get a "`Delivery Success`" message if the messages are delivered properly.

On the receiver board, you should be receiving the packets from all the other boards. In this test, we were receiving data from 3 different boards.

ESP-NOW Many-To-Many Two-Way Communication



You can accomplish this by expanding the code of ESP-Now Point-to-Point Two-Way communication.
It will be become more complex as you expand with more ESP boards.
Hence, there is a more suitable and enhanced protocol and library available for this set-up which can also be referred as a MESH set-up.