ESP 32 NOW

ESP-NOW est un protocole d'échange de données entre ESP-32 (ou ESP8266) que l'on peut programmer avec l'IDE Arduino. C'est un protocole développé par Espressif, le fabricant des puces ESP, et qui permet d'échanger de courts paquets de données directement et simplement. Il est possible de réaliser des échanges entre plusieurs ESP dans les deux sens (émission et réception) sans intermédiaire central (sans routeur).

Le protocole ESP-NOW

Tel qu'il est décrit sur le site d'Espressif, le protocole ESP-NOW permet d'échanger des paquets sans routeur Wifi, s'approchant en cela du système utilisé par les objets connectés en 2.4Ghz (souris ou claviers sans fil, en particulier). L'appairage entre modules est nécessaire (nous verrons plus loin qu'il faut utiliser l'adresse MAC des ESP), mais une fois que cet appairage est réalisé la connexion s'effectue très rapidement, sans “handshake”. Dis plus simplement, lorsque l'appairage est effectué, on peut éteindre ou redémarrer un module, la reconnexion sera automatique et immédiate.

Il est possible de réaliser un schéma où un module central envoie des informations à de nombreux autres modules (one to many), ou au contraire de nombreux modules envoient à un module central (many to one), mais aussi un réseau maillé ou chaque module peut envoyer à tous les autres. Chaque module, au sein d'un réseau, peut-être à le fois émetteur ET récepteur.

Il est possible de chiffrer les communications, et de mélanger des communications chiffrées ou non dans un même réseau. Le nombre de modules doit rester en dessous de 10 lorsque l'on utilise le chiffrement et 20 lorsque l'on échange en clair.

Il n'est possible d'échanger que 250 octets au maximum à chaque envoi. Une fonction de rappel peut être déclenchée pour confirmer la bonne réception ou l'envoi des données. Connaître l'adresse MAC

La première étape va consister à noter les adresses MAC de chaque appareil que nous voulons faire communiquer. Pour mémoire, l'adresse MAC (Media Access Control) est un identifiant unique matériel qui identifie chaque appareil sur un réseau. Chaque ESP32 possède, en sortie d'usine, une adresse MAC différente, composée de 6 octets. Il est possible de changer de manière logicielle l'adresse MAC de l'ESP, mais cela ne survit pas à un reboot, il faut donc l'inclure dans le code exécuté à chaque fois (tuto ici).

Téléverser le code suivant dans chaque ESP, et **noter soigneusement le résultat qui va s'afficher dans la console.**

exemple001b.ino

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at
https://RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-ar
duino/
  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 <WiFi.h>
#include <esp_wifi.h>

void readMacAddress(){
  uint8_t baseMac[6];
  esp_err_t ret = esp_wifi_get_mac(WIFI_IF_STA, baseMac);
  if (ret == ESP_OK) {
    Serial.printf("%02x:%02x:%02x:%02x:%02x:%02x\n",
                  baseMac[0], baseMac[1], baseMac[2],
                  baseMac[3], baseMac[4], baseMac[5]);
  } else {
    Serial.println("Failed to read MAC address");
  }
}

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

  WiFi.mode(WIFI_STA);
  WiFi.STA.begin();

  Serial.print("[DEFAULT] ESP32 Board MAC Address: ");
  readMacAddress();
}

void loop(){

}
```

Cette étape manuelle permettra d'appairer facilement chaque module. Néanmoins, il est à noter qu'une méthode très astucieuse permet de se passer de cette manip préalable, et de connecter de manière automatique chaque module. Néanmoins, elle est un peu plus complexe.

L'idée est la suivante: les “récepteurs” émettent un réseau wifi dont le SSID comporte une chaîne prédéfinie (par exemple, “RECEPTION”. L'émetteur scanne les réseaux Wifi environnants, et lorsqu'il détecte un SSID commençant par “RECEPTION”, il en récupère l'adresse MAC, avant de créer l'appairage. C'est de cette manière que fonctionnent les deux exemples Master et Slave que l'on trouvera dans les exemples ESP32>ESP-NOW de l'IDE Arduino.

Principe de communication

Pour commencer, nous allons simplement envoyer des informations d'un ESP vers un autre. Par commodité, nous les appellerons donc “Émetteur” et “Récepteur”.

Côté émetteur:

1. -Initialiser ESP-NOW.
2. -Enregistrer une fonction de rappel, qui sera exécutée quant un message est envoyé. Cela nous permettra de vérifier la bonne transmission du message.
3. -On ajoute l'adresse MAC du récepteur, pour l'appairage.
4. -On envoie le message.

Côté récepteur:

1. -Initialiser ESP-NOW.
2. -Enregistrer une fonction de rappel, qui sera exécutée quant un message est reçu.
3. -Dans cette fonction de rappel, on sauve le contenu du message dans une variable pour en faire quelque chose.

À Chaque étape listée ci-dessus va correspondre une fonction spécifique à l'utilisation du protocole:

- **esp_now_init()** Initialiser ESP-NOW. Il faut initialiser le wifi avant d'initialiser ESP-NOW.
- **esp_now_add_peer()** On appelle cette fonction pour appairer un ESP, on passe son adresse MAC en argument.
- **esp_now_send()** Envoie des données avec ESP-NOW.
- **esp_now_register_send_cb()** Enregistre une fonction de rappel qui sera déclenchée lorsque l'on envoie des données.
- **esp_now_register_rcv_cb()** Enregistre une fonction de rappel qui sera déclenchée lorsque l'on reçoit des données.

Code de l'émetteur

Ci-dessous, le code commenté:

```
// Référence technique:
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html


// Inclure les librairies
#include <esp_now.h>
#include <WiFi.h>

// Stockage de l'adresse MAC du récepteur pour usage ultérieur. Remplacer les 'FF' par les valeur notées plus avant.
uint8_t MAC_recepteur[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// La variable qui sera envoyée au récepteur (nous générerons une valeur aléatoire pour l'exemple)

float maValeurEnvoyee;

// La fonction de rappel qui nous assurera de la bonne livraison du message
```

```
void quand_donnees_Envoyees(const uint8_t *mac_addr, esp_now_send_status_t
status) {
  Serial.print("\r\nDernier paquet envoyé:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Succès" : "Échec");
}

// Une variable qui servira à stocker les réglages concernant le récepteur
esp_now_peer_info_t infosRecepteur;


void setup() {
  // On initie la comm série à 115200 Bauds
  Serial.begin(115200);

  // On démarre le Wifi en mode Station
  WiFi.mode(WIFI_STA);

  // Puis on initialise ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Erreur d'initialisation ESP-NOW");
    return;
  }

  // Si ESP-NOW a correctement démarré, il est temps d'enregistrer la
fonction de rappel:
  esp_now_register_send_cb(quand_donnees_Envoyees);

  // Tout est prêt pour l'appairage avec notre récepteur:
  memcpy(infosRecepteur.peer_addr, MAC_recepteur, 6);

  // On définit un canal (0 utilisera automatiquement le même canal que
celui utilisé par le wifi)
  infosRecepteur.channel = 0;

  // On ne chiffre pas les échanges
  infosRecepteur.encrypt = false;

  // Appairage
  if (esp_now_add_peer(&infosRecepteur) != ESP_OK){
    Serial.println("Échec de l'appairage");
    return;
  }
}

void loop() {

  // On définit la valeur de la variable à envoyer à l'aide d'un générateur
aléatoire
  maValeurEnvoyee = random(1,20);
```

```
  // On envoie le message
  esp_err_t resultat = esp_now_send(MAC_recepteur, (uint8_t *)
&maValeurEnvoyee, sizeof(maValeurEnvoyee));

  if (resultat == ESP_OK) {
    Serial.println("Envoi OK");
  }
  else {
    Serial.println("Erreur envoi");
  }

  // On effectue cette opération toutes les secondes
  delay(1000);
}
```

Code du récepteur

Voici le code du récepteur, commenté en détail:

```
// Inclure les librairies
#include <esp_now.h>
#include <WiFi.h>

// La variable qui sera envoyée au récepteur (nous générerons une valeur
aléatoire pour l'exemple)
float maValeurRecue;

// La fonction de rappel qui nous assurera de la bonne livraison du message
void quand_donnees_Recues(const uint8_t * mac, const uint8_t
*data_reception, int taille) {
  memcpy(&maValeurRecue, data_reception, sizeof(maValeurRecue));
  Serial.print("Bytes received: ");
  Serial.println(taille);
  Serial.print("valeur reçue: ");
  Serial.println(maValeurRecue);
  Serial.println();
}

void setup() {
  // On initie la comm série à 115200 Bauds
  Serial.begin(115200);

  // On démarre le Wifi en mode Station
  WiFi.mode(WIFI_STA);

  // Puis on initialise ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("Erreur d'initialisation ESP-NOW");
```

```
    return;
  }

  // Si ESP-NOW a correctement démarré, il est temps d'enregistrer la
fonction de rappel:
  esp_now_register_recv_cb(quand_donnees_Recues);

}

void loop() {

}
```

Tester et aller plus loin

Lorsque vous regarderez les console série des deux ESP, vous constaterez que les messages sont bien expédiés, bien reçus et leur contenu correctement interprété côté récepteur. Les différentes sources trouvées sur le net parlent d'une portée en extérieur supérieur à 200 mètres, avec les deux antennes pointant l'une vers l'autre. À tester, les essais réalisés autour de notre projet de compteur ont plutôt été concluants jusqu'à 100 mètres environ.

Nous vous renvoyons aux exemples disponibles dans la bibliothèque Arduino pour aller plus loin !

arduino-esp:esp-now_programmes.zip