Első lépések az ESP-NOW-val (ESP32 Arduino IDE-vel)

Fordította: Maczák András

andras@maczak.eu

Tartalomjegyzék

Ismerd meg, hogyan lehet használni az ESP-NOW-t az Arduino IDE-vel programozott ESP32 kártyák közötti adatcserére. Az ESP-NOW egy kapcsolat nélküli kommunikációs protokoll, amelyet az Espressif fejlesztett ki, és amely rövid csomagátvitelt biztosít. Ez a protokoll lehetővé teszi, hogy több eszköz egyszerűen beszélhessen egymással.

Bemutatkozik az ESP-NOW

Az Espressif webhelyén olvasható, hogy az ESP-NOW egy „az Espressif által kifejlesztett protokoll, amely lehetővé teszi, hogy több eszköz kommunikáljon egymással Wi-Fi használata nélkül. A protokoll hasonló az alacsony fogyasztású 2,4 GHz-es vezeték nélküli kapcsolathoz (…). Az eszközök közötti párosításra a kommunikáció előtt van szükség. A párosítás után a kapcsolat biztonságos és peer-to-peer, nincs szükség kézfogásra.”

Ez azt jelenti, hogy az eszközök egymással való párosítása után a kapcsolat állandó. Más szóval, ha az egyik kártyád hirtelen áramszünetet kap vagy újraindul, újrainduláskor automatikusan csatlakozik a társához a kommunikáció folytatásához.

Az ESP-NOW a következő funkciókat támogatja:

- Titkosított és titkosítatlan unicast kommunikáció;
- Vegyes titkosított és titkosítatlan társ eszközök;
- Legfeljebb 250 bájtos hasznos adat hordozható;
- Visszahívási funkció, amely beállítható úgy, hogy tájékoztassa az alkalmazási réteget az átvitel sikeréről vagy sikertelenségéről.

Az ESP-NOW technológiának a következő korlátai is vannak:

- Korlátozottan titkosított társaik. Állomás módban legfeljebb 10 titkosított társ támogatott; SoftAP vagy SoftAP + Station módban legfeljebb 6;
- Több titkosítatlan társ is támogatott, azonban ezek teljes számának 20-nál kevesebbnek kell lennie, beleértve a titkosított partnereket is;
- A hasznos terhelés 250 bájtra korlátozódik.

Egyszerűen fogalmazva, az ESP-NOW egy gyors kommunikációs protokoll, amellyel kis (250 bájtig terjedő) üzeneteket lehet váltani az ESP32 kártyák között.

Az ESP-NOW nagyon sokoldalú, és különböző beállításokkal lehet egy- vagy kétirányú kommunikációt folytatni.

ESP-NOW egyirányú kommunikáció

Például egyirányú kommunikáció esetén a következő forgatókönyvek lehetnek:

- **Egy ESP32 kártya adatokat küld egy másik ESP32 kártyára**

Ez a konfiguráció nagyon könnyen megvalósítható, és nagyszerűen alkalmas adatok küldésére egyik kártyáról a másikra, például érzékelők leolvasásai vagy BE és KI parancsok a GPIO-k vezérléséhez.

- **Egy „főnök” ESP32, amely adatokat küld több ESP32 „szolgának”**

Egy ESP32 kártya ugyanazokat vagy eltérő parancsokat küld a különböző ESP32 kártyáknak. Ez a konfiguráció ideális egy távirányítóhoz hasonló építkezéshez. A ház körül több ESP32 kártya is lehet, amelyeket egy fő ESP32 kártya vezérel.

- **Egy ESP32 „szolga” több „főnök”-től fogad adatokat**

Ez a konfiguráció ideális, ha több érzékelő-csomópontról szeretne adatokat gyűjteni egy ESP32 kártyára. Ez webszerverként konfigurálható például az összes többi kártya adatainak megjelenítéséhez.

Megjegyzés: az ESP-NOW dokumentációban nincs olyan, hogy „küldő/főnök” és „vevő/szolga”. Minden tábla lehet küldő vagy fogadó. Azonban a dolgok tisztán tartása érdekében használjuk a „küldő” és a „vevő” vagy a „főnök” és „szolga” kifejezéseket.

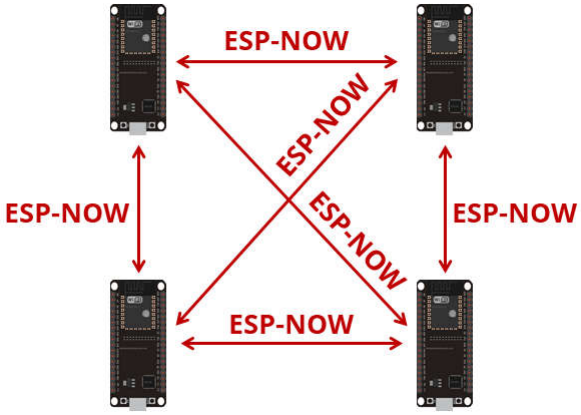
ESP-NOW kétirányú kommunikáció

Az ESP-NOW-val minden tábla egyszerre lehet küldő és vevő is. Így kétirányú kommunikációt létesíthet a táblák között.

Például lehet, hogy két tábla kommunikál egymással.

Ismerd meg a következőket: Az érzékelők leolvasási adatainak cseréje az ESP-NOW kétirányú kommunikációval.

Több kártyát is hozzáadhat ehhez a konfigurációhoz, és valami hálózatnak tűnik (az összes ESP32 kártya kommunikál egymással).



Összefoglalva, az ESP-NOW ideális egy olyan hálózat kiépítéséhez, amelyben több ESP32 kártya is cserélhet egymással adatokat.

ESP32: Az alaplap MAC-címének beszerzése

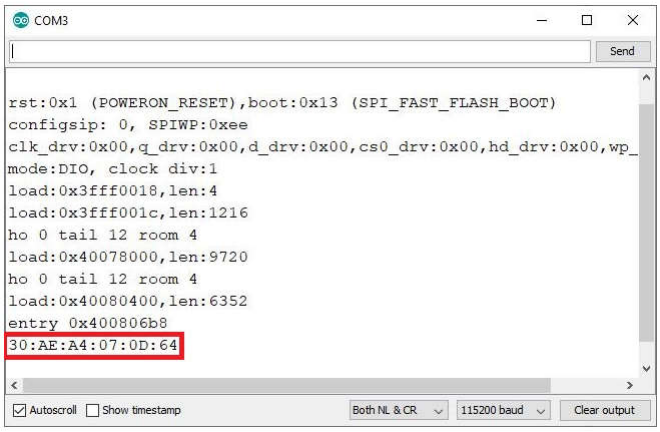
Az ESP-NOW-on keresztüli kommunikációhoz ismerned kell az ESP32 vevő MAC-címét. Így tudja, melyik eszközre küldi az adatokat.

Minden ESP32-nek egyedi MAC-címe van, és így azonosítjuk az egyes kártyákat, hogy adatokat küldhessünk az ESP-NOW használatával (további információ az ESP32 MAC-címének beszerzéséről és módosításáról).

A tábla MAC-címének beszerzéséhez töltsd fel a következő kódot.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/get-change-
esp32-esp8266-mac-address-arduino/
  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(){

}
```

A kód feltöltése után nyisd meg a soros monitort 115200 adatátviteli sebességgel, és nyomd meg az ESP32 RST/EN gombot. A MAC-cím a következőképpen lesz kinyomtatva:



Mentsd el a kártya MAC-címét, mert szükséged lesz rá, hogy adatokat küldj a megfelelő kártyára az ESP-NOW-on keresztül.

1. ESP-NOW Egyirányú pont-pont kommunikáció

Az ESP-NOW vezeték nélküli kommunikáció használatának megkezdéséhez összeállítunk egy egyszerű projektet, amely bemutatja, hogyan küldhetsz üzenetet egyik ESP32-ről a másikra. Az egyik ESP32 lesz a „küldő”, a másik ESP32 pedig a „vevő”.

Elküldünk egy struktúrát, amely egy char, int, float és logikai típusú változót tartalmaz. Ezután módosíthatod a struktúrát, hogy elküld azokat a változótípusokat, amelyek megfelelnek a projektnek (például érzékelők leolvasása vagy logikai változók valami be- vagy kikapcsolásához).

A jobb megértés érdekében az ESP32 #1-nek a „küldőt”, az ESP32 #2-nek pedig a „vevőt” hívjuk.

Íme, mit kell belefoglalnunk a küldő vázlatába:

1. Az ESP-NOW inicializálása;
2. Regisztráljon visszahívási funkciót az adatok küldésekor – az `OnDataSent` funkció üzenet elküldésekor kerül végrehajtásra. Ez megmondhatja nekünk, hogy az üzenetet sikeresen kézbesítettük-e vagy sem;
3. Hozzáadunk egy társeszközt (a vevőt). Ehhez ismerni kell a vevő MAC-címét;
4. Üzenet küldése a társeszköznek.

A vevő oldalon a vázlatnak tartalmaznia kell:

1. Az ESP-NOW inicializálása;
2. Regisztráljon a visszahívási funkcióhoz (`OnDataRecv`). Ez egy olyan funkció, amely üzenet fogadásakor kerül végrehajtásra.
3. A visszahívási függvényen belül az üzenet elmentése egy változóba, hogy az adott információval bármilyen feladatot végrehajthassunk.

Az ESP-NOW olyan visszahívási funkciókkal működik, amelyek akkor hívódnak meg, amikor egy eszköz üzenetet kap, vagy üzenetet küldenek (azt kapja meg, ha az üzenetet sikeresen kézbesítették, vagy ha nem sikerült).

ESP-NOW Hasznos függvények

Íme az ESP-NOW legfontosabb funkcióinak összefoglalása:

Függvény neve és leírása

`esp_now_init()` Inicializálja az ESP-NOW-t. Az ESP-NOW inicializálása előtt inicializálni kell a Wi-Fi-t.

`esp_now_add_peer()` Ezt a függvényt hívjuk egy eszköz párosításához, és argumentumként átadjuk a társ MAC-címét.

`esp_now_send()` Adatok küldése az ESP-NOW segítségével.

`esp_now_register_send_cb()` Regisztrálunk egy visszahívási funkciót, amely az adatok küldésekor aktiválódik. Üzenet elküldésekor egy függvény hívódik meg – ez a függvény visszaadja, hogy a kézbesítés sikeres volt-e vagy sem.

`esp_now_register_recv_cb()` Regisztrálunk egy visszahívási funkciót, amely az adatok fogadásakor aktiválódik. Amikor adatot fogadunk az ESP-NOW-on keresztül, egy függvény hívódik meg.

Ha többet szeretnél megtudni ezekről a funkciókról, olvasd el az ESP-NOW dokumentációját a Dokumentumok olvasása oldalon:
`(https://demo-dijiudu.readthedocs.io/en/latest/api-reference/wifi/esp_now.html).`

ESP32 küldő vázlata (ESP-NOW)

Itt van az ESP32 küldő kártya kódja. Másold ki a kódot az Arduino IDE-be, de még ne töltsd fel. Néhány módosítást kell végrehajtani, hogy az működjön.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-
esp32-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 <esp_now.h>
#include <WiFi.h>

// REPLACE WITH YOUR 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;
```

```
  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, "THIS IS A CHAR");
  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);
}
```

Hogyan működik a kód

Először is felvesszük az `esp_now.h` és a `WiFi.h` könyvtárakat.

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

A következő sorba be kell írni az ESP32 vevő MAC-címét.

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

Esetünkben a vevő MAC-címe: 30:AE:A4:07:0D:64, de ezt a változót le kell cserélned a saját MAC-címedre.

Ezután létrehozunk egy struktúrát, amely tartalmazza a küldeni kívánt típusú adatokat. Ezt a struktúrát `struct_message`-nek hívtuk, és 4 különböző típusú változót tartalmaz. Ezt módosíthatod más változótípusok küldéséhez.

```
typedef struct struct_message {
  char a[32];
  int b;
  float c;
  bool d;
} struct_message;
```

Ezután létrehozunk egy új, `myData` nevű, `struct_message` típusú változót, amely tárolja a változók értékeit.

```
struct_message myData;
```

Létrehozunk létre egy `esp_now_peer_info_t` típusú változót a társ információinak tárolására.

```
esp_now_peer_info_t peerInfo;
```

Ezután meghatározzuk az `OnDataSent()` függvényt. Ez egy visszahívási funkció, amely az üzenet elküldésekor kerül végrehajtásra. Ebben az esetben ez a funkció egyszerűen kinyomtatja, hogy az üzenetet sikeresen kézbesítették-e vagy sem.

```
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");
```

```
}
```

A `setup()`-ban inicializáljuk a soros monitort hibakeresési célból:

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

Beállítjuk az eszközt Wi-Fi állomásként:

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

Az ESP-NOW inicializálása:

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

Az ESP-NOW sikeres inicializálása után regisztráljuk a visszahívási funkciót, amelyet az üzenet elküldésekor hív meg. Ebben az esetben a korábban létrehozott `OnDataSent()` függvényre regisztrálunk.

```
esp_now_register_send_cb(OnDataSent);
```

Ezt követően egy másik ESP-NOW eszközzel kell párosítanunk az adatküldéshez. Ezt tesszük a következő sorokban:

```
//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;
}
```

A `loop()`-ban 2 másodpercenként üzenetet küldünk az ESP-NOW-on keresztül (ez a késleltetési idő módosítható).

Először beállítjuk a változók értékeit a következőképpen:

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

Ne feledd, hogy a `myData` egy struktúra. Itt hozzárendeljük a struktúrán belül küldeni kívánt értékeket. Például az első sor egy karaktert, a második sor egy véletlenszerű Int számot, egy Float és egy logikai változót rendel hozzá.

Ezt a struktúrát azért hozzuk létre, hogy megmutassuk, hogyan kell elküldeni a leggyakoribb változótípusokat. Módosíthatod a szerkezetet, hogy más adattípusokat küldj.

Végül elküldjük az üzenetet az alábbiak szerint:

```
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));
```

Ellenőrizzük, hogy az üzenet sikeresen el lett-e küldve:

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

A `loop()` 2000 ezredmásodpercenként (2 másodpercenként) kerül végrehajtásra.

```
delay(2000);
```

ESP32 vevő vázlat (ESP-NOW)

Töltsd fel a következő kódot az ESP32 vevőkártyára.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-
esp32-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 <esp_now.h>
#include <WiFi.h>

// Structure example to receive data
// Must match the sender 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;

// 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;
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
}

void loop() {

}
```

Hogyan működik a kód

A küldőhöz hasonlóan kezdjük a könyvtárak felvételével:

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

Hozzunk létre egy struktúrát az adatok fogadásához. Ennek a szerkezetnek meg kell egyeznie a küldő vázlatában meghatározottakkal.

```
typedef struct struct_message {
  char a[32];
  int b;
  float c;
  bool d;
} struct_message;
```

Hozzunk létre egy `myData` nevű `struct_message` változót.

```
struct_message myData;
```

Hozzunk létre egy visszahívási függvényt, amelyet akkor hív meg, amikor az ESP32 fogadja az adatokat az ESP-NOW-on keresztül. A függvényt `onDataRecv()` hívják, és több paramétert is el kell fogadnia az alábbiak szerint:

```
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
```

Az `incomingData` adatváltozó tartalmát bemásoljuk a `myData` változóba.

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

Most a `myData` struktúra számos változót tartalmaz az ESP32 küldő által küldött értékekkel. Például az `a` változó eléréséhez csak meg kell hívnunk a `myData.a`-t.

Ebben a példában egyszerűen kinyomtatjuk a kapott adatokat, de egy gyakorlati alkalmazásban kinyomtathatjuk például egy kijelzőre is.

```
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();
```

A `setup()`-ban inicializáljuk a soros monitort.

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

Beállítjuk az eszközt Wi-Fi állomásként.

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

Az ESP-NOW inicializálása:

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

Regisztráljunk egy visszahívási funkcióra, amelyet az adatok fogadásakor hív meg. Ebben az esetben regisztrálunk a korábban létrehozott `OnDataRecv()` függvényre.

```
esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
```

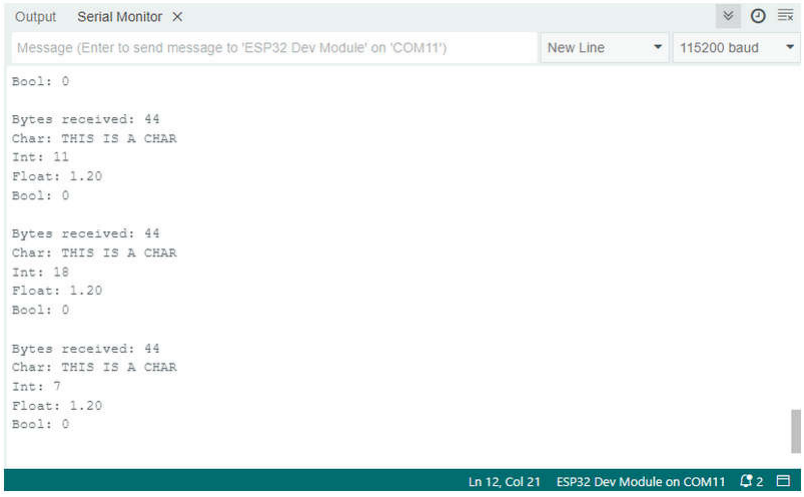
Az ESP-NOW kommunikáció tesztelése

Töltsd fel a küldő vázlatot a küldő ESP32 kártyára és a vevő vázlatot a vevő ESP32 kártyájára.

Most nyiss meg két Arduino IDE ablakot. Egyet a vevőnek, egy másikat a küldőnek. Nyisd meg az egyes kártyák soros monitorát. Külön COM portnak kell lennie minden kártyához.

Ezt kell kapnod a küldő oldalon.

És ez az, amit a vevőoldalon meg kell kapnod. Vedd figyelembe, hogy az Int változó minden fogadott leolvasás között változik (mivel a küldő oldalon véletlen számra állítottuk be).



Kipróbáltuk a két tábla közötti kommunikációs hatótávolságot, és nyílt terepen akár 220 méterig (kb. 722 láb) is stabil kommunikációt tudunk elérni. Ebben a kísérletben mindkét ESP32 fedélzeti antenna egymás felé nézett.

Lezárás

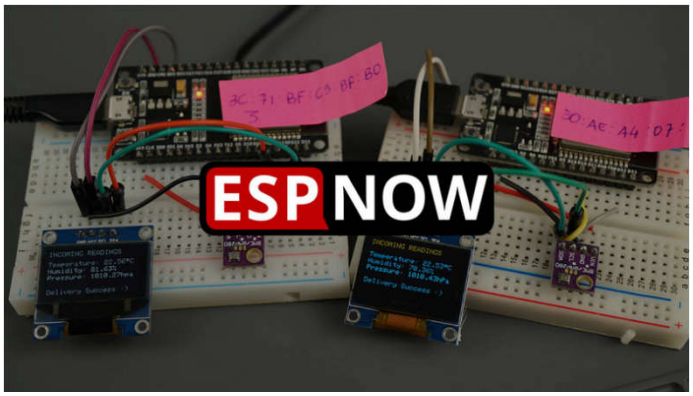
Példáinkat igyekeztünk a lehető legegyszerűbbé tenni, hogy jobban megértsd, hogyan működik minden. További, az ESP-NOW-hoz kapcsolódó függvények is hasznosak lehetnek a projektekben, mint például: társak kezelése, társak törlése, szolga eszközök keresése stb… Egy teljes példa az Arduino IDE-ben a Fájl > Példák > menüpontban. ESP32 > ESPNow, és válassz egyet a példavázlatok közül.

Reméljük, hogy hasznosnak találtad az ESP-NOW bemutatását. Az első lépések egyszerű példájaként bemutattuk, hogyan küldhetsz adatokat struktúraként egyik ESP32-ről a másikra. Az ötlet az, hogy a szerkezeti értékeket helyettesítsd például az érzékelők leolvasásával vagy a GPIO állapotokkal.

Ezenkívül az ESP-NOW-val minden kártya egyszerre lehet küldő és fogadó. Egy tábla több kártyára is küldhet adatokat, és több kártyáról is fogadhat adatokat.

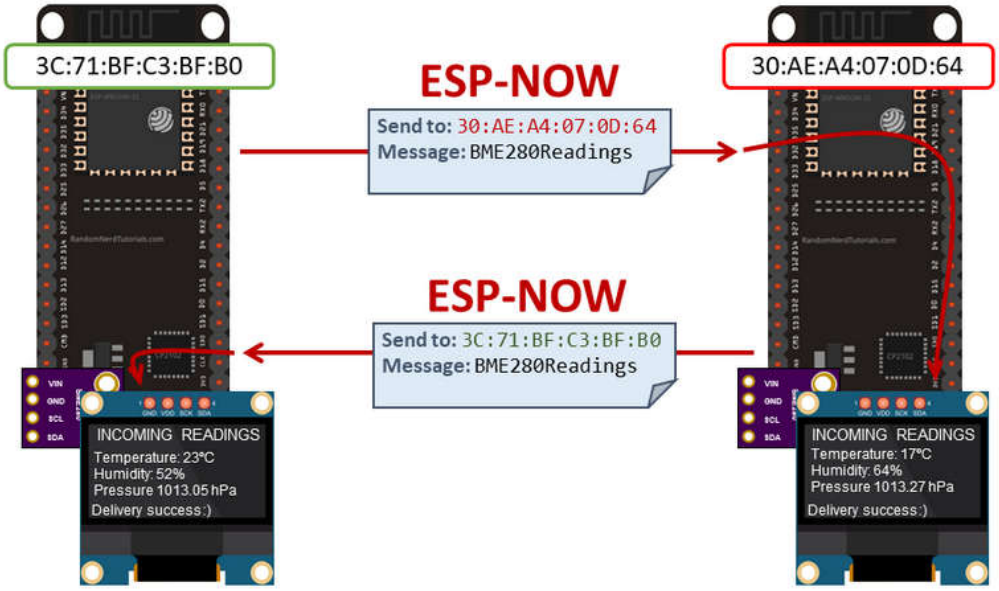
2. ESP-NOW kétirányú kommunikáció az ESP32 kártyák között

Ebben az útmutatóban bemutatjuk, hogyan létesíthetsz kétirányú kommunikációt két ESP32 kártya között az ESP-NOW kommunikációs protokoll használatával. Például két ESP32 kártya cseréli ki az érzékelők leolvasását (nyílt területen akár 220 méter ~ 722 láb hatótávolsággal).



A projekt áttekintése.

Az alábbi diagram egy magas szintű áttekintést nyújt az általunk megépítendő projektről.



- Ebben a projektben két ESP32 kártyánk lesz. Mindegyik kártya OLED kijelzőhöz és BME280 érzékelőhöz csatlakozik;
- Minden tábla hőmérséklet-, páratartalom- és nyomásértékeket kap a megfelelő érzékelőktől;
- Mindegyik tábla az ESP-NOW-on keresztül küldi a leolvasást a másik táblának;
- Amikor egy tábla megkapja a leolvasásokat, megjeleníti azokat az OLED-kijelzőn;
- A leolvasások elküldése után a tábla megjelenítik az OLED-en, ha az üzenetet sikeresen kéz-besítették;
- Az üzenet elküldéséhez minden táblának ismernie kell a másik kártya MAC-címét.

Ebben a példában kétirányú kommunikációt használunk a két tábla között, de további kártyákat is hozzáadhatsz ehhez a beállításhoz, és az összes tábla kommunikál egymással.

Előfeltételek

Mielőtt folytatnád ezt a projektet, győződj meg arról, hogy ellenőrizted a következő előfeltételeket.

Telepítsd a könyvtárakat

Telepítsd a következő könyvtárakat az Arduino IDE-be. Ezek a könyvtárak az Arduino Könyvtár-kezelő keresztül telepíthetők. Lépjen a **Vázlat > Könyvtár tartalmazása> Könyvtárak kezelése** menüpontra, és keresse meg a könyvtár nevét.

- OLED-könyvtárak (ESP32 OLED útmutató): Adafruit_SSD1306 könyvtár és Adafruit_GFX könyvtár
- BME280 könyvtárak (ESP32 BME280 útmutató): Adafruit_BME280 könyvtár és Adafruit Unified Sensor könyvtár

Szükséges alkatrészek

Ehhez az oktatóanyaghoz a következő alkatrészekre van szükség:

- 2x ESP32 fejlesztő kártya (olvasd el: A legjobb ESP32 kártyák)
- 2x BME280 érzékelő (BME280 teljes útmutató)
- 2x 0,96 hüvelykes OLED-kijelző (OLED teljes útmutató)
- Bekötőkártya
- Jumper vezetékek

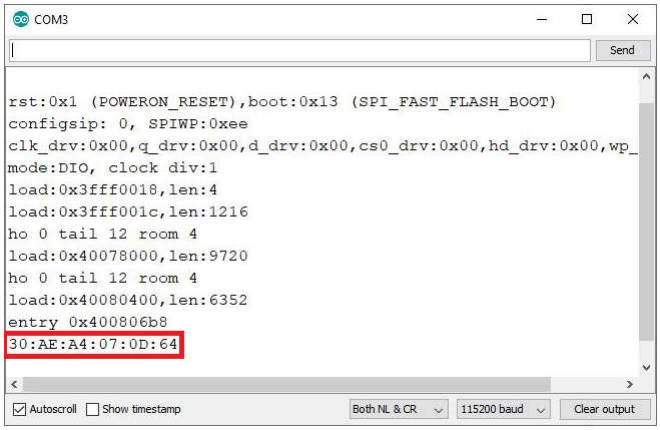
A táblák MAC-címének beszerzése

Az egyes táblák közötti üzenetküldéshez tudnunk kell a MAC-címüket. Minden kártyának egyedi MAC-címe van (további információ az ESP32 MAC-cím beszerzéséről és módosításáról).

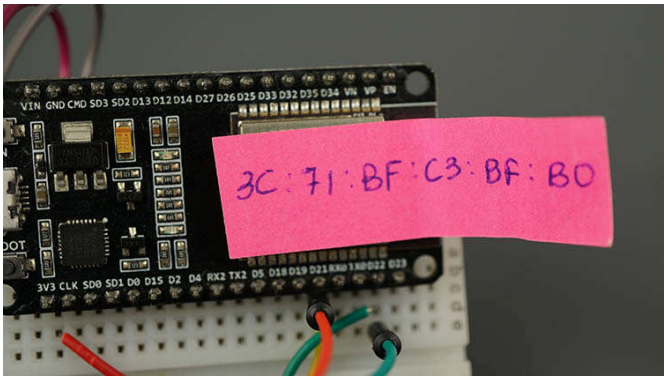
Töltsd fel a következő kódot minden kártyádra, hogy megkapd a MAC-címüket.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/get-change-
esp32-esp8266-mac-address-arduino/
  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(){

}
```

A kód feltöltése után nyomd meg az RST/EN gombot, és a MAC címnek meg kell jelennie a soros monitoron.

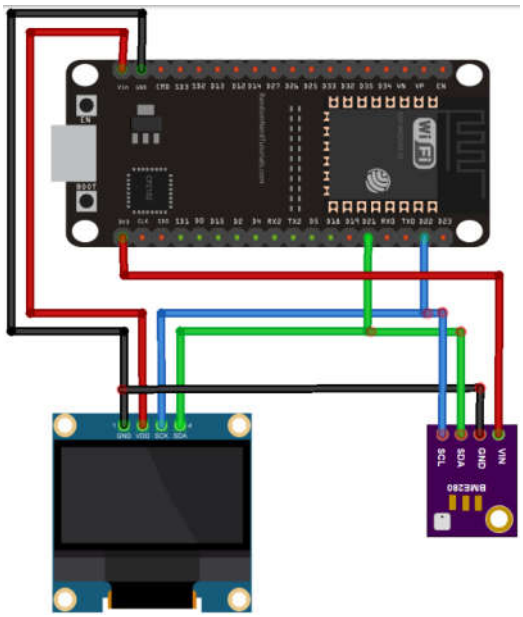


Írd le az egyes kártyák MAC-címét, hogy egyértelműen azonosítsd őket.



Vázlatos diagram

Csatlakoztass egy OLED kijelzőt és egy BME280 érzékelőt minden ESP32 kártyához. Kövesd a következő vázlatos diagramot.



A következő táblázatot referenciaként használhatod a BME280 érzékelő bekötéséhez.

BME280	ESP32
VIN	3,3V
GND	GND
SCL	GPIO 22
SDA	GPIO 21

Kövesd a következő táblázatot az OLED-kijelző ESP32-hez történő csatlakoztatásához.

OLED kijelző	ESP32
GND	GND
VCC	VIN
SCL	GPIO 22
SDA	GPIO 21

ESP32 kétirányú kommunikáció ESP-NOW kód

Töltsd fel a következő kódot minden tábládra. A kód feltöltése előtt meg kell adnod a másik kártya MAC-címét (a tábláét, amelyre adatokat küldesz).

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-two-
way-communication-esp32/
  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 <esp_now.h>
#include <WiFi.h>

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 64  // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

```
Adafruit_BME280 bme;

// REPLACE WITH THE MAC Address of your receiver
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

// Define variables to store BME280 readings to be sent
float temperature;
float humidity;
float pressure;

// Define variables to store incoming readings
float incomingTemp;
float incomingHum;
float incomingPres;

// Variable to store if sending data was successful
String success;

//Structure example to send data
//Must match the receiver structure
typedef struct struct_message {
    float temp;
    float hum;
    float pres;
} struct_message;

// Create a struct_message called BME280Readings to hold sensor readings
struct_message BME280Readings;

// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;

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");
  if (status ==0){
    success = "Delivery Success :)";
  }
  else{
    success = "Delivery Fail :(";
  }
}

// Callback when data is received
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
  Serial.print("Bytes received: ");
  Serial.println(len);
```

```
  incomingTemp = incomingReadings.temp;
  incomingHum = incomingReadings.hum;
  incomingPres = incomingReadings.pres;
}

void setup() {
  // Init Serial Monitor
  Serial.begin(115200);

  // Init BME280 sensor
  bool status = bme.begin(0x76);
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }

  // Init OLED display
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }

  // 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;
  }
  // Register for a callback function that will be called when data is
received
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
}

void loop() {
  getReadings();

  // Set values to send
```

```
  BME280Readings.temp = temperature;
  BME280Readings.hum = humidity;
  BME280Readings.pres = pressure;

  // Send message via ESP-NOW
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&BME280Readings, sizeof(BME280Readings));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
  updateDisplay();
  delay(10000);
}
void getReadings(){
  temperature = bme.readTemperature();
  humidity = bme.readHumidity();
  pressure = (bme.readPressure() / 100.0F);
}

void updateDisplay(){
  // Display Readings on OLED Display
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  display.println("INCOMING READINGS");
  display.setCursor(0, 15);
  display.print("Temperature: ");
  display.print(incomingTemp);
  display.cp437(true);
  display.write(248);
  display.print("C");
  display.setCursor(0, 25);
  display.print("Humidity: ");
  display.print(incomingHum);
  display.print("%");
  display.setCursor(0, 35);
  display.print("Pressure: ");
  display.print(incomingPres);
  display.print("hPa");
  display.setCursor(0, 56);
  display.print(success);
  display.display();

  // Display Readings in Serial Monitor
  Serial.println("INCOMING READINGS");
  Serial.print("Temperature: ");
  Serial.print(incomingReadings.temp);
  Serial.println(" ºC");
  Serial.print("Humidity: ");
```

```
  Serial.print(incomingReadings.hum);
  Serial.println(" %");
  Serial.print("Pressure: ");
  Serial.print(incomingReadings.pres);
  Serial.println(" hPa");
  Serial.println();
}
```

Hogyan működik a kód

Az OLED-kijelzővel és a BME280-as érzékelővel való interakciót a korábbi oktatóanyagokban részletesen ismertettük. Itt csak az ESP-NOW vonatkozó részeit tekintjük át.

A kód jól kommentálva van, hogy megértsd, mit csinálnak az egyes kódsorok.

Az ESP-NOW használatához fel kell venni a következő könyvtárakat.

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

A következő sorba írd be a vevőkártya MAC-címét:

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

Létrehozunk változókat a BME280 érzékelő hőmérséklet-, páratartalom- és nyomásértékeinek tárolásához. Ezeket a leolvasásokat küldjük el a másik táblának:

```
uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0x07, 0x0D, 0x64};
// Define variables to store BME280 readings to be sent
float temperature;
float humidity;
float pressure;
```

Létrehozunk változókat a másik kártyáról érkező szenzorleolvasások tárolására:

```
// Define variables to store incoming readings
float incomingTemp;
float incomingHum;
float incomingPres;
```

A következő változó eredményes üzenetet fog tárolni, ha a leolvasások sikeresen elküldésre kerültek a másik táblára.

```
// Variable to store if sending data was successful
String success;
```

Létrehozunk egy szerkezetet, amely tárolja a páratartalom, hőmérséklet és nyomás értékeket.

```
typedef struct struct_message {
  float temp;
  float hum;
  float pres;
} struct_message;
```

Ezután létre kell hoznunk a szerkezet két példányát. Az egyik a leolvasások fogadására, a másik pedig az elküldendő leolvasások tárolására.

A `BME280Readings` tárolja az elküldendő értékeket.

```
// Create a struct_message called BME280Readings to hold sensor readings
struct_message BME280Readings;
```

Az `incomingReadings` tárolja a másik tábláról érkező adatokat.

```
// Create a struct_message to hold incoming sensor readings
struct_message incomingReadings;
```

Létrehozunk egy `esp_now_peer_info_t` típusú változót a társ információinak tárolására.

```
esp_now_peer_info_t peerInfo;
```

Ezután két visszahívási függvényt kell létrehoznunk. Az egyiket az adatok küldésekor hívjuk meg, a másikat pedig az adatok fogadásakor.

Az OnDataSent() visszahívási függvény

Az `OnDataSent()` függvény az új adatok küldésekor kerül meghívásra. Ez a függvény egyszerűen kinyomtatja, hogy az üzenetet sikeresen kézbesítették-e vagy sem. Ha az üzenetet sikeresen kézbesítettük, az állapotváltozó 0-t ad vissza, így az eredményüzenetünket „Sikeres kézbesítés”-re állíthatjuk:

```
Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
if (status ==0) {
  success = "Delivery Success :)";
}
```

Ha az eredményüzenet 1-et ad vissza, az azt jelenti, hogy a kézbesítés sikertelen volt:

```
else {
  success = "Delivery Fail :(";
}
```

Az OnDataRecv() visszahívási függvény

Az `OnDataRecv()` függvény akkor kerül meghívásra, ha új csomag érkezik.

```
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
```

Az új csomagot a korábban létrehozott `incomingReadings` struktúrába mentjük:

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

Az üzenet hosszát a soros monitorra nyomtatjuk. Egy csomagban csak 250 bájtot küldhet.

```
Serial.print("Bytes received: ");
Serial.println(len);
```

Ezután tároljuk a bejövő leolvasásokat a megfelelő változókban. Az `incomingReadings` struktúrán belüli hőmérsékleti változó eléréséhez egyszerűen meg kell hívni az `incomingReadings.temp` fájlt az alábbiak szerint:

```
incomingTemp = incomingReadings.temp;
```

Ugyanezt a folyamatot hajtják végre a többi változó esetében is:

```
incomingHum = incomingReadings.hum;
incomingPres = incomingReadings.pres;
```

setup()

A `setup()`-ban inicializáljuk az ESP-NOW-t.

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

Ezután regisztráljuk az `OnDataSent` visszahívási funkciót.

```
esp_now_register_send_cb(OnDataSent);
```

Ahhoz, hogy adatokat küldjön egy másik kártyára, párosítania kell azt társként. A következő sorok regisztrálnak és új társat adnak hozzá.

```
// 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;
}
```

Regisztráljuk az `OnDataRecv` visszahívási funkciót.

```
esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
```

loop()

A `loop()`-ban meghívjuk a `getReadings()` függvényt, amely az érzékelőtől új hőmérsékleti értékekért felelős. Ez a függvény a `loop()` után jön létre.

Miután megkaptuk az új hőmérséklet-, páratartalom- és nyomásértékeket, frissítjük a `BME280Reading` szerkezetünket a következő értékekkel:

```
BME280Readings.temp = temperature;
BME280Readings.hum = humidity;
BME280Readings.pres = pressure;
```

Ezután elküldhetjük a `BME280Readings` struktúrát az ESP-NOW-on keresztül:

```
// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)
&BME280Readings, sizeof(BME280Readings));

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

Végül meghívjuk az `updateDisplay()` függvényt, amely frissíti az OLED kijelzőt a másik ESP32 kártyáról érkező adatokkal.

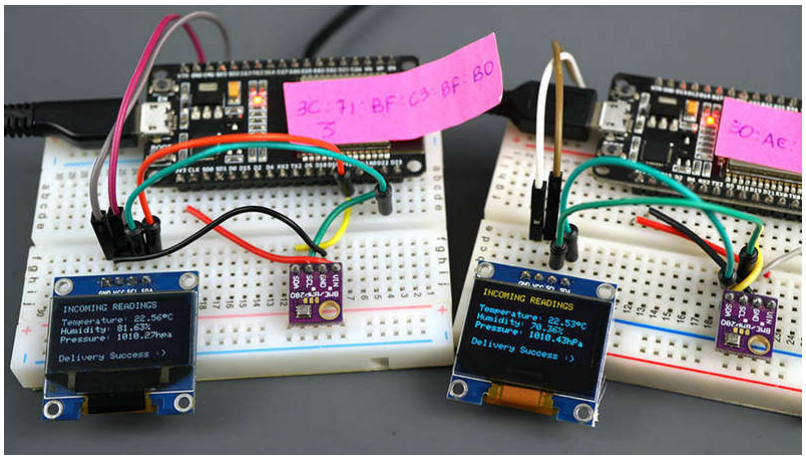
```
updateDisplay();
```

A `loop()` 10 másodpercenként kerül végrehajtásra.

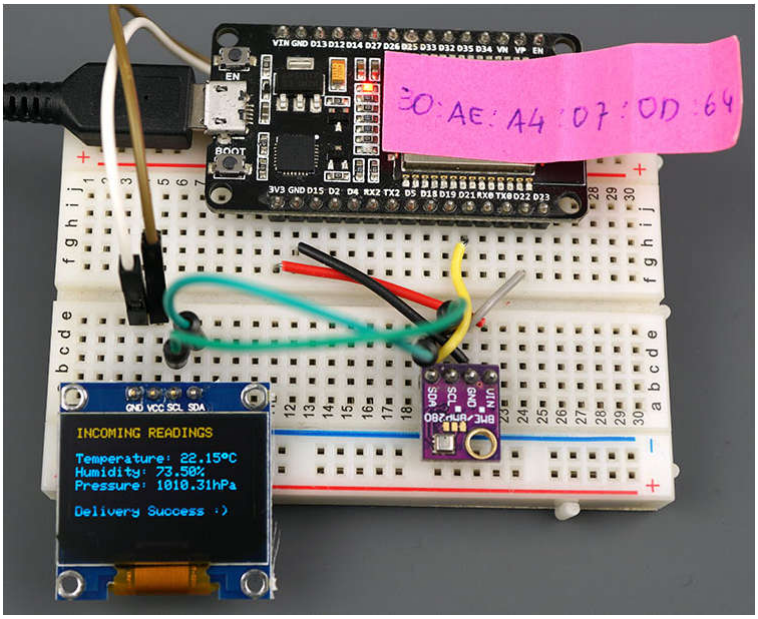
Nagyjából így működik a kód. Fel kell töltened a kódot mindkét táblára. Csak módosítanod kell a kódot annak a kártyának a MAC-címével, amelyre adatokat küldöd.

Demonstráció

Miután feltöltötted a kódot mindkét kártyára, látnod kell az OLED-et, amely megjeleníti a másik kártya érzékelőjeit, valamint egy sikeres kézbesítési üzenetet.



Mint látható, a vártnak megfelelően működik:



Kipróbáltuk a két tábla közötti kommunikációs hatótávolságot, és nyílt terepen akár 220 méterig (kb. 722 láb) is stabil kommunikációt tudunk elérni. Ebben a kísérletben mindkét ESP32 fedélzeti antenna egymás felé mutatott.

Lezárás

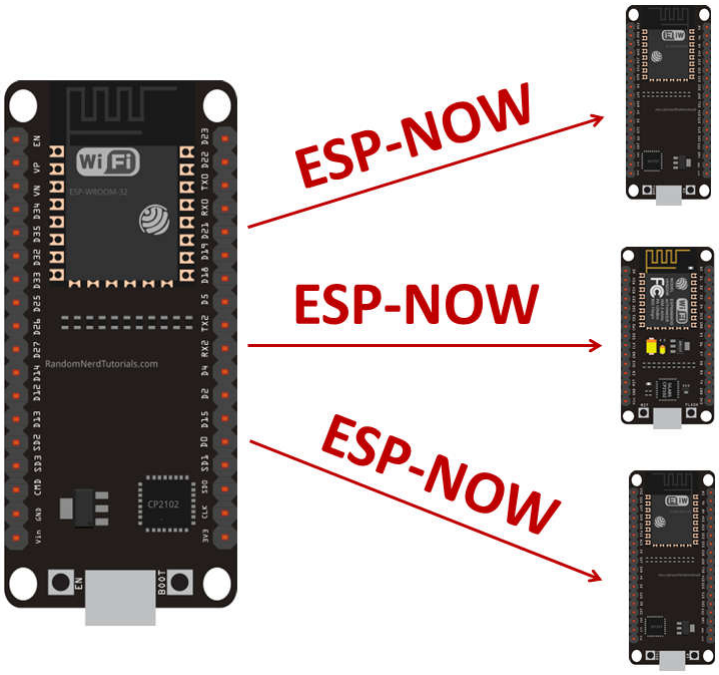
Ebben az oktatóanyagban megmutattuk, hogyan létesíthetsz kétirányú kommunikációt két ESP32 kártyával az ESP-NOW használatával. Ez egy nagyon sokoldalú kommunikációs protokoll, amely akár 250 bájtos csomagok küldésére is használható. Az ESP-NOW kommunikációs protokoll ESP8266 kártyákkal is használható: Első lépések az ESP-NOW-val (ESP8266 NodeMCU Arduino IDE-vel).

Példaként bemutattuk a két tábla közötti interakciót, de sok táblát is hozzáadhatsz a beállításhoz. Csak tudnod kell annak a táblának a MAC-címét, amelyre adatokat küldöd.

3. ESP-NOW ESP32-vel: Adatok küldése több táblára (egy a többhöz)

Ebből az oktatóanyagból megtudhatod, hogyan használhatod az ESP-NOW kommunikációs protokollt adatok küldésére egy ESP32-ről több ESP32 vagy ESP8266 kártyára (egy a többhöz konfiguráció). A kártyák programozása Arduino IDE segítségével történik.

A projekt áttekintése



- Egy ESP32 küldőként működik;
- Több ESP32 vagy ESP8266 kártya működik vevőként. Ezt a beállítást két ESP32 kártyával és egy ESP8266 kártyával teszteltük egyszerre. Képesnek kell lenni további táblák hozzáadására a beállításhoz;
- Az ESP32 küldő nyugtázó üzenetet kap, ha az üzeneteket sikeresen kézbesítették. Tudja, mely táblák kapták meg az üzenetet, és melyek nem;
- Kicsit eltérő vevőkódot kell feltölteni attól függően, hogy ESP32-t vagy ESP8266-ot használunk;
- Példaként véletlenszerű értékeket cserélünk a táblák között. Módosítanod kell ezt a példát, hogy parancsokat vagy érzékelő-leolvasásokat küldj (az érzékelők leolvasását az ESP-NOW használatával cseréljük ki).

Ez az oktatóanyag a következő két forgatókönyvet fedi le:

- ugyanazt az üzenetet küldjük minden táblának;
- minden táblának más üzenetet küldünk.

Előfeltételek

Az ESP32/ESP8266 kártyákat Arduino IDE használatával programozzuk, ezért mielőtt folytatnád ezt az oktatóanyagot, győződj meg arról, hogy ezek a kártyák telepítve vannak az Arduino IDE-ben.

- Az ESP32 tábla telepítése Arduino IDE-ben (Windows, Mac OS X és Linux)
- Az ESP8266 tábla telepítése Arduino IDE-ben (Windows, Mac OS X, Linux)

Szükséges alkatrészek

Az oktatóanyag követéséhez több ESP32 kártyára és/vagy ESP8266 kártyára van szükség.

- ESP32 (olvasd: a legjobb ESP32 fejlesztőkártyák)
- ESP8266 (olvasd: a legjobb ESP8266 fejlesztőkártyák)

A táblák MAC-címének beszerzése

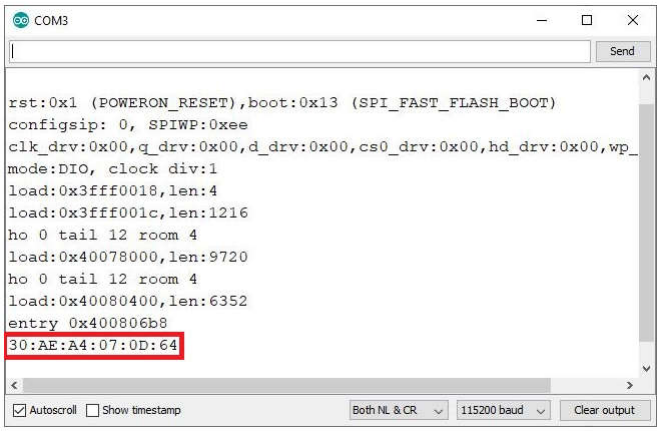
Az ESP-NOW-on keresztüli üzenetek küldéséhez ismerni kell a fogadó kártyák MAC-címét. Minden kártyának egyedi MAC-címe van (további információ az ESP32 MAC-cím beszerzéséről és módosításáról).

Töltsd fel a következő kódot mindegyik vevőkártyádra, hogy megkapd a MAC-címét.

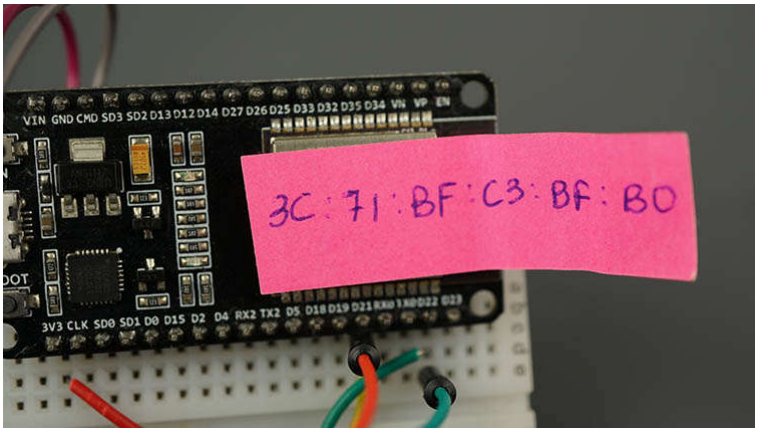
```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  A projekt részleteit itt találod:
  https://RandomNerdTutorials.com/get-change-esp32-esp8266-mac-address-arduino/
  Ezúton ingyenes engedélyt adunk minden olyan személynek, aki másolatot
    szerez a szoftverről és a kapcsolódó dokumentációs fájlokról.
  A fenti szerzői jogi megjegyzést és ezt az engedélyt tartalmazó megjegy-
    zést a Szoftver minden másolatának vagy jelentős részének tartalmaznia
    kell.
*/
#ifdef ESP32
  #include <WiFi.h>
  #include <esp_wifi.h>
#else
  #include <ESP8266WiFi.h>
#endif

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

  Serial.print("ESP Board MAC Address: ");
  #ifdef ESP32
    WiFi.mode(WIFI_STA);
    WiFi.STA.begin();
    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");
    }
  #else
    Serial.println(WiFi.macAddress());
  #endif
}

void loop(){

}
```

A kód feltöltése után nyomd meg az RST/EN gombot, és a MAC címnek meg kell jelennie a soros monitoron.



Felírhatod a táblák MAC-címét egy címkére, hogy egyértelműen azonosítsd az egyes táblákat.



ESP32 küldőkód (ESP-NOW)

A következő kód több (három) ESP-kártyára küld adatokat az ESP-NOW-on keresztül. Módosítsd a kódot a vevőkártyák MAC-címével. A vevőkártyák számától függően kódsorokat is kell hozzáadni vagy törölni.

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-one-
to-many-esp32-esp8266/
  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 <esp_now.h>
#include <WiFi.h>

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS
uint8_t broadcastAddress1[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
uint8_t broadcastAddress2[] = {0xFF, , , , , };
uint8_t broadcastAddress3[] = {0xFF, , , , , };

typedef struct test_struct {
  int x;
  int y;
} test_struct;

test_struct test;

esp_now_peer_info_t peerInfo;

// callback when data is sent
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet to: ");
  // 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]);
  Serial.print(macStr);
  Serial.print(" send 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(OnDataSent);

  // register peer
  peerInfo.channel = 0;
```

```
  peerInfo.encrypt = false;
  // register first peer
  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  // register second peer
  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
  /// register third peer
  memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  test.x = random(0,20);
  test.y = random(0,20);

  esp_err_t result = esp_now_send(0, (uint8_t *) &test,
sizeof(test_struct));

  if (result == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
  delay(2000);
}
```

Hogyan működik a kód

Először is felvesszük az `esp_now.h` és a `WiFi.h` könyvtárakat.

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

A vevők MAC-címei

Írd be a vevőkészülékek MAC-címeit. Példánkban három táblára küldünk adatokat.

```
uint8_t broadcastAddress1[] = {0x3C, 0x71, 0xBF, 0xC3, 0xBF, 0xB0};
uint8_t broadcastAddress2[] = {0x24, 0x0A, 0xC4, 0xAE, 0xAE, 0x44};
uint8_t broadcastAddress3[] = {0x80, 0x7D, 0x3A, 0x58, 0xB4, 0xB0};
```

Ezután létrehozunk egy struktúrát, amely tartalmazza a küldeni kívánt adatokat. Ezt a struktúrát `test_struct`-nak neveztük el, és két egész változót tartalmaz. Ezt megváltoztathatod, hogy bármilyen típusú változót küldhess.

```
typedef struct test_struct {
  int x;
  int y;
} test_struct;
```

Létrehozunk egy `test_struct` típusú új változót, amelyet `test`-nek hívnak, és amely tárolja a változók értékeit.

```
test_struct test;
```

Létrehozunk egy `esp_now_peer_info_t` típusú változót a társ információinak tárolására.

```
esp_now_peer_info_t peerInfo;
```

Az OnDataSent() visszahívási függvény

Ezután meghatározuk az `OnDataSent()` függvényt. Ez egy visszahívási függvény, amely üzenet elküldésekor kerül végrehajtásra. Ebben az esetben ez a függvény kinyomtatja, hogy az üzenetet sikeresen kézbesítették-e vagy sem, és melyik MAC-címhez. Tehát tudja, mely táblák kapták meg az üzenetet, és melyek nem.

```
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet from: ");
  // 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]);
  Serial.print(macStr);
  Serial.print(" send status:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" :
"Delivery Fail");
}
```

setup()

A `setup()`-ban inicializálja a soros monitort hibakeresési célból:

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

Állítsuk be az eszközt Wi-Fi állomásként:

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

Az ESP-NOW inicializálása:

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

Az ESP-NOW sikeres inicializálása után regisztráljuk a visszahívási funkciót, amelyet az üzenet elküldésekor hív meg. Ebben az esetben regisztráljunk a korábban létrehozott `OnDataSent()` függvényre.

```
esp_now_register_send_cb(OnDataSent);
```

Ezt követően az adatok küldéséhez párosítanunk kell más ESP-NOW eszközökkel. Ezt tesszük a következő sorokban – partnerek regisztrálása:

```
// register peer
peerInfo.channel = 0;
peerInfo.encrypt = false;
// register first peer
memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
if (esp_now_add_peer(&peerInfo) != ESP_OK){
  Serial.println("Failed to add peer");
  return;
}
// register second peer
memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
if (esp_now_add_peer(&peerInfo) != ESP_OK){
  Serial.println("Failed to add peer");
  return;
}
/// register third peer
memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
if (esp_now_add_peer(&peerInfo) != ESP_OK){
  Serial.println("Failed to add peer");
  return;
}
```

Ha további társakat szeretnél hozzáadni, csak át kell másolnod ezeket a sorokat, és át kell adna a társ MAC-címét:

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

loop()

A `loop()`-ban 2 másodpercenként üzenetet küldünk az ESP-NOW-on keresztül (ez a késleltetési idő módosítható).

Minden változóhoz értéket rendelünk:

```
test.x = random(0,20);
test.y = random(0,20);
```

Ne feledd, hogy a teszt egy struktúra. Itt rendelheted hozzá azokat az értékeket, amelyeket el szeretnél küldeni a struktúrán belül. Ebben az esetben csak véletlenszerű értékeket küldünk. Gyakorlati alkalmazásban ezeket például parancsokkal vagy érzékelők leolvasásával kell helyettesíteni.

Végül elküldjük az üzenetet az alábbiak szerint:

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

Az `esp_now_send()` függvény első argumentuma a vevő MAC-címe. Ha 0-t ad meg argumentumként, akkor ugyanazt az üzenetet küldi el minden regisztrált társnak. Ha más üzenetet szeretnél küldeni minden társnak, kövesd a következő részt.

Ellenőrizzük, hogy az üzenet sikeresen el lett-e küldve:

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

A `loop()` 2000 ezredmásodpercenként (2 másodpercenként) kerül végrehajtásra.

```
delay(2000);
```

Az egyes táblákra más üzenetet küldő kód nagyon hasonló az előzőhöz. Tehát csak vessünk egy pillantást a különbségekre.

Ha más üzenetet szeretnél küldeni minden táblának, létre kell hoznod egy adatstruktúrát minden táblához, például:

```
test_struct test;
test_struct test2;
test_struct test3;
```

Ebben az esetben ugyanazt a szerkezettípust küldjük el (`test_struct`). Más típusú szerkezetet is küldhetsz, ha a vevő kódja fel van készítve az adott típusú szerkezet fogadására.

Ezután rendeljünk különböző értékeket az egyes struktúrák változóihoz. Ebben a példában csak véletlenszerű számokra állítjuk őket.

```
test.x = random(0,20);
test.y = random(0,20);
test2.x = random(0,20);
test2.y = random(0,20);
test3.x = random(0,20);
test3.y = random(0,20);
```

Végül minden vevőnél meg kell hívni az `esp_now_send()` függvényt.

Például küldjük el a `test` struktúrát arra a kártyára, amelynek MAC-címe `broadcastAddress1`.

```
esp_err_t result1 = esp_now_send(
  broadcastAddress1,
  (uint8_t *) &test,
  sizeof(test_struct));

if (result1 == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

Ugyanezt tedd a többi táblával is. A második táblához küld el a `test2` szerkezetet:

```
esp_err_t result2 = esp_now_send(
  broadcastAddress2,
  (uint8_t *) &test2,
  sizeof(test_struct));

if (result2 == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

És végül a harmadik táblához küld el a `test3` szerkezetet:

```
esp_err_t result3 = esp_now_send(
  broadcastAddress3,
  (uint8_t *) &test3,
  sizeof(test_struct));

if (result3 == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

Íme a teljes kód, amely más üzenetet küld az egyes tábláknak.

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-one-
to-many-esp32-esp8266/
  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 <esp_now.h>
```

```
#include <WiFi.h>

// REPLACE WITH YOUR ESP RECEIVER'S MAC ADDRESS
uint8_t broadcastAddress1[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
uint8_t broadcastAddress2[] = {0xFF, , , , , };
uint8_t broadcastAddress3[] = {0xFF, , , , , };

typedef struct test_struct {
  int x;
  int y;
} test_struct;

esp_now_peer_info_t peerInfo;

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  Serial.print("Packet to: ");
  // 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]);
  Serial.print(macStr);
  Serial.print(" send 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(OnDataSent);

  // register peer
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  memcpy(peerInfo.peer_addr, broadcastAddress1, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }

  memcpy(peerInfo.peer_addr, broadcastAddress2, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
```

```
  }
  memcpy(peerInfo.peer_addr, broadcastAddress3, 6);
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  test_struct test;
  test_struct test2;
  test_struct test3;
  test.x = random(0,20);
  test.y = random(0,20);
  test2.x = random(0,20);
  test2.y = random(0,20);
  test3.x = random(0,20);
  test3.y = random(0,20);

  esp_err_t result1 = esp_now_send(
    broadcastAddress1,
    (uint8_t *) &test,
    sizeof(test_struct));

  if (result1 == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }
  delay(500);
  esp_err_t result2 = esp_now_send(
    broadcastAddress2,
    (uint8_t *) &test2,
    sizeof(test_struct));

  if (result2 == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
  }

  delay(500);
  esp_err_t result3 = esp_now_send(
    broadcastAddress3,
    (uint8_t *) &test3,
    sizeof(test_struct));

  if (result3 == ESP_OK) {
    Serial.println("Sent with success");
  }
  else {
    Serial.println("Error sending the data");
```

```
  }
  delay(2000);
}
```

ESP32 vevőkód (ESP-NOW)

Töltsd fel a következő kódot a vevőkártyákra (példánkban három vevőkártyát használtunk).

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-one-to-many-esp32-esp8266/
  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 <esp_now.h>
#include <WiFi.h>

//Structure example to receive data
//Must match the sender structure
typedef struct test_struct {
  int x;
  int y;
} test_struct;

//Create a struct_message called myData
test_struct 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("x: ");
  Serial.println(myData.x);
  Serial.print("y: ");
  Serial.println(myData.y);
  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;
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
}

void loop() {

}
```

Hogyan működik a kód

Az adóhoz hasonlóan kezdjük a könyvtárak felvételével:

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

Létrehozunk egy struktúrát az adatok fogadásához. Ennek a szerkezetnek meg kell egyeznie a küldő vázlatában meghatározottakkal.

```
typedef struct test_struct {
  int x;
  int y;
} test_struct;
```

Hozzunk létre egy `test_struct` változót `myData` néven.

```
test_struct myData;
```

Létrehozunk egy visszahívási funkciót, amely akkor kerül meghívásra, amikor az ESP32 fogadja az adatokat az ESP-NOW-on keresztül. A függvényt `onDataRecv()`-nek hívják, és több paramétert is el kell fogadnia az alábbiak szerint:

```
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
```

Másoljuk be az `incomingData` adatváltozó tartalmát a `myData` változóba.

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

Most a `myData` struktúra több változót tartalmaz az ESP32 küldő által küldött értékekkel. Például az x változó eléréséhez hívjuk meg `myData.x` névvel.

Ebben a példában kinyomtatjuk a kapott adatokat, de gyakorlati alkalmazásban például OLED kijelző-re is kinyomtathatjuk az adatokat.

```
Serial.print("Bytes received: ");
Serial.println(len);
Serial.print("x: ");
Serial.println(myData.x);
```

```
Serial.print("y: ");
Serial.println(myData.y);
Serial.println();
```

A `setup()`-ban inicializáljuk a soros monitort.

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

Beállítjuk az eszközt Wi-Fi állomásként.

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

Az ESP-NOW inicializálása:

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

Regisztráljunk egy visszahívási funkcióra, amelyet az adatok fogadásakor hív meg. Ebben az esetben regisztrálunk a korábban létrehozott `OnDataRecv()` függvényre.

```
esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
```

ESP8266 vevőkód (ESP-NOW)

Ha ESP8266 kártyát használ vevőként, töltsd fel az előző helyett a következő kódot.

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-one-
to-many-esp32-esp8266/
  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 test_struct {
  int x;
  int y;
} test_struct;

//Create a struct_message called myData
test_struct 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: ");
```

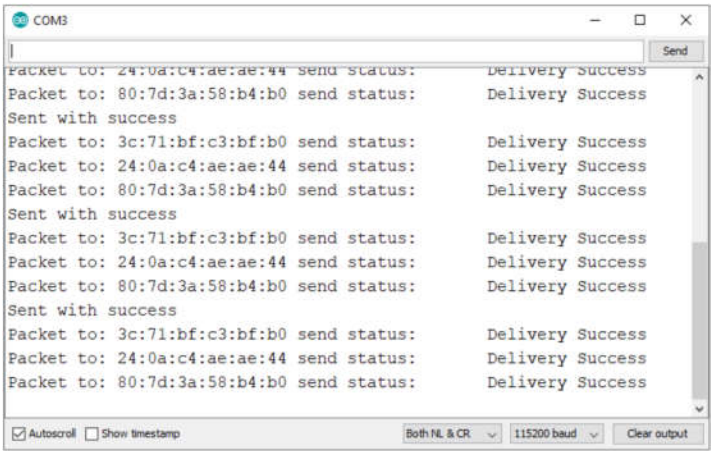
```
  Serial.println(len);
  Serial.print("x: ");
  Serial.println(myData.x);
  Serial.print("y: ");
  Serial.println(myData.y);
  Serial.println();
}

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() {

}
```

Az apró részletektől eltekintve ez a kód nagyon hasonlít az ESP32 vevőkódhoz. Tehát nem magyarázzuk el, hogyan működik. További információért olvassa el az ESP-NOW Kezdő lépések útmutatóját az ESP8266 NodeMCU-val.

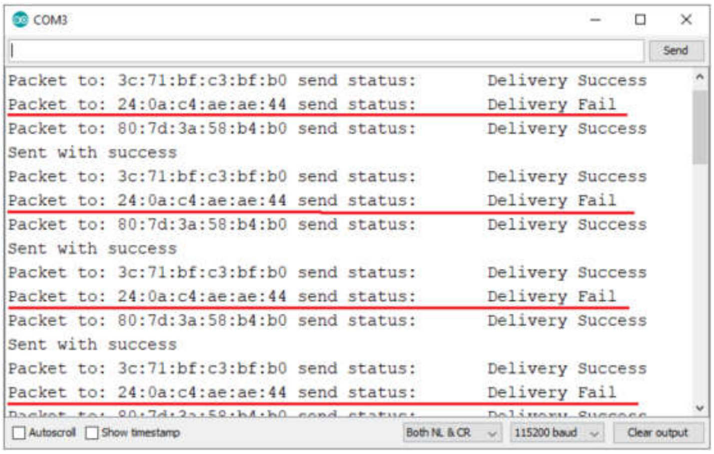
Demonstráció

Ha az összes kártya be van kapcsolva, nyisd meg az Arduino IDE soros monitort ahhoz a COM-porthoz, amelyhez a küldő csatlakozik.

Meg kell kezdenie a „Sikeres kézbesítés” („Delivery Success”)üzenetek fogadását a megfelelő vevő MAC-címével a soros monitoron.

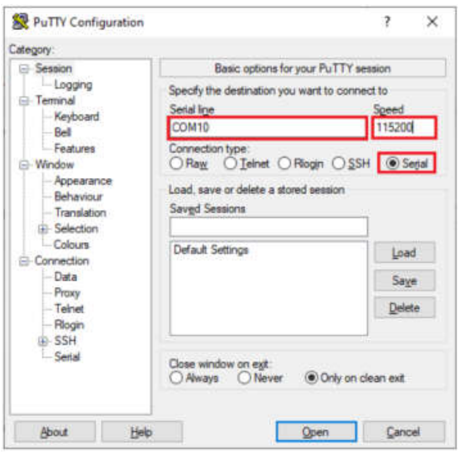


Ha megszakítod az áramellátást az egyik táblánál, akkor egy „Kézbesítési hiba” („Delivery Fail”) üzenetet fogsz kapni az adott táblára vonatkozóan. Így gyorsan azonosíthatod, melyik fórum nem kapta meg az üzenetet.

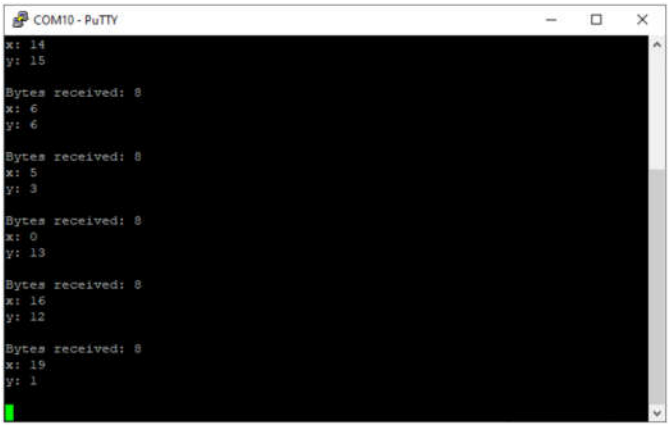


Ha ellenőrizni szeretnéd, hogy a kártyák valóban fogadják-e az üzeneteket, megnyithatod a soros monitort a COM-porthoz, amelyhez csatlakoztatva vannak, vagy a PuTTY segítségével soros kommunikációt létesíthetsz a kártyáiddal.

Ha PuTTY-t használsz, válaszd a Soros kommunikáció lehetőséget, írja be a COM-port számát és az adatátviteli sebességet (115200) az alábbiak szerint, majd kattintson az Open gombra.



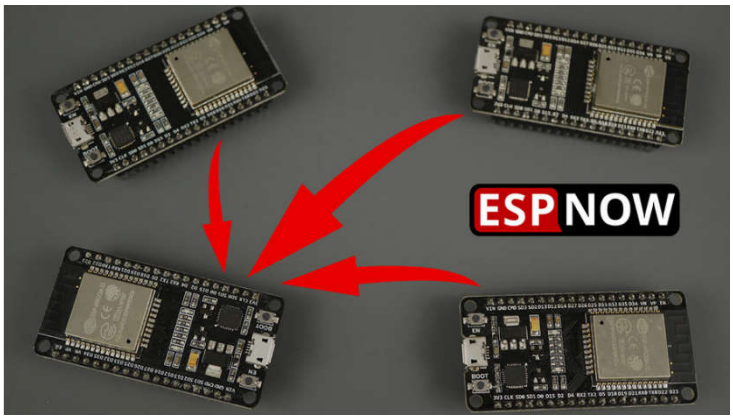
Ezután látnod kell az üzenetek fogadását.



Nyiss meg egy soros kommunikációt minden táblához, és ellenőrizd, hogy megkapják-e az üzeneteket.

4. ESP-NOW ESP32-vel: Adatok fogadása több kártyáról (több az egyhez)

Ez az oktatóanyag bemutatja, hogyan állíthatsz be egy ESP32 kártyát több ESP32 kártya adatának fogadására az ESP-NOW kommunikációs protokollon keresztül (több az egyhez konfiguráció). Ez a konfiguráció ideális, ha több érzékelő-csomópontról szeretnél adatokat gyűjteni egy ESP32 kártyára. A kártyák programozása Arduino IDE segítségével történik.



A projekt áttekintése

Ez az oktatóanyag bemutatja, hogyan állíthat be egy ESP32 kártyát több ESP32 kártya adat fogadására az ESP-NOW kommunikációs protokollon keresztül (több az egyhez konfiguráció), a következő ábrán látható módon.

- Egy ESP32 kártya vevőként/szolgaként működik;
- Több ESP32 kártya működik küldőként/mesterként. Ezt a példát 5 ESP32 küldőkártyával teszteltük, és jól működött. Képesnek kell lenni további táblák hozzáadására a beállításhoz;
- Az adó tábla egy nyugtázó üzenetet kap, amely jelzi, hogy az üzenetet sikeresen kézbesítették-e vagy sem;
- Az ESP32 vevőkártya fogadja az üzeneteket az összes adótól, és azonosítja, hogy melyik kártya küldte az üzenetet;
- Példaként véletlenszerű értékeket cserélünk a táblák között. Módosítania kell ezt a példát, hogy parancsokat vagy érzékelőleolvasásokat küldjön (az érzékelők leolvasását az ESP-NOW használatával cserélje ki).

Szükséges alkatrészek

Az oktatóanyag követéséhez több ESP32 kártyára van szükség. Minden ESP32 modellnek működnie kell. Kísérleteztünk az ESP32 kártyák különböző modelljeivel, és mindegyik jól működött (ESP32 DOIT kártya, TTGO T-Journal, ESP32 OLED kártyával és ESP32-CAM).

- ESP32 (olvasd: a legjobb ESP32 fejlesztőkártyák)

A vevőkártya MAC-címének lekérése

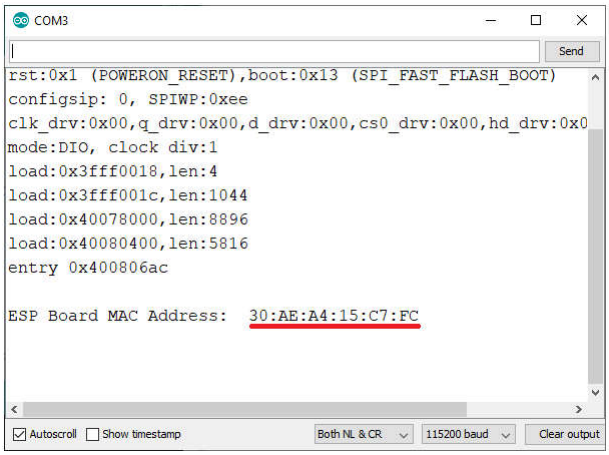
Ha üzeneteket szeretnél küldeni az ESP-NOW-on keresztül, ismerned kell a vevőkártya MAC-címét. Minden kártyának egyedi MAC-címe van (további információ az ESP32 MAC-cím beszerzéséről és módosításáról).

Töltsd fel a következő kódot az ESP32 vevőkártyára, hogy megkapd a MAC-címet.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/get-change-
esp32-esp8266-mac-address-arduino/
  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.
*/
#ifdef ESP32
  #include <WiFi.h>
  #include <esp_wifi.h>
#else
  #include <ESP8266WiFi.h>
#endif

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

  Serial.print("ESP Board MAC Address: ");
  #ifdef ESP32
    WiFi.mode(WIFI_STA);
    WiFi.STA.begin();
    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");
    }
  #else
    Serial.println(WiFi.macAddress());
  #endif
}
```

```
void loop(){

}
```

A kód feltöltése után nyomd meg az RST/EN gombot, és a MAC címnek meg kell jelennie a soros monitoron.

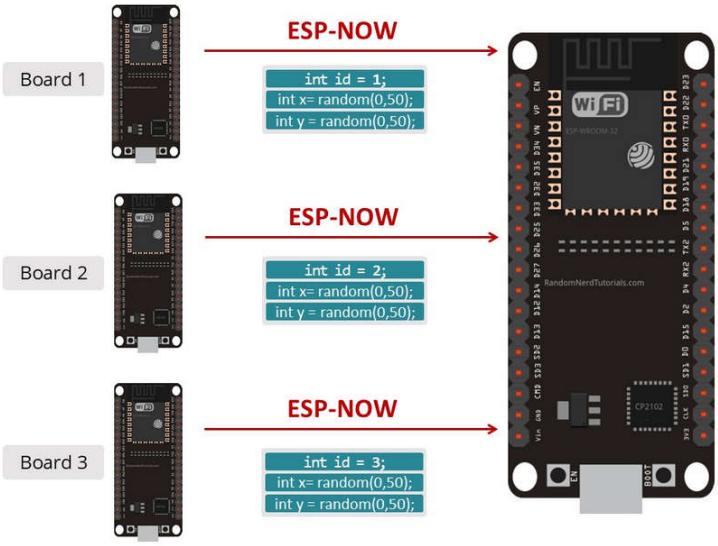


ESP32 küldőkód (ESP-NOW)

A vevő minden feladót egyedi MAC-címe alapján azonosítani tud. Azonban a különböző MAC-címek kezelése a vevő oldalon annak azonosítása érdekében, hogy melyik kártya melyik üzenetet küldte, bonyolult lehet.

Így a dolgok megkönnyítése érdekében minden táblát egyedi számmal (`id`) azonosítunk, amely 1-től kezdődik. Ha három táblája van, az egyik azonosítószáma 1, a másik 2-es, végül pedig 3-as, a többi változóval együtt elküldésre kerül a vevőnek.

Példaként felvázolunk egy struktúrát, amely tartalmazza a tábla azonosító számát, `id`-t és két véletlenszámot, `x` és `y`, ahogy az alábbi ábrán látható.



Töltsd fel a következő kódot mindegyik küldő táblára. Ne felejtsd el növelni az egyes küldőtáblák azonosítószámát (id).

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-many-
to-one-esp32/
  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 <esp_now.h>
#include <WiFi.h>

// REPLACE WITH THE RECEIVER'S 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 {
    int id; // must be unique for each sender board
    int x;
    int y;
} struct_message;

// Create a struct_message called myData
struct_message myData;

// Create peer interface
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
  myData.id = 1;
  myData.x = random(0,50);
  myData.y = random(0,50);

  // 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(10000);
}
```

Hogyan működik a kód

Felvesszük a WiFi és az esp_now könyvtárakat.

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

Írd be a vevő MAC-címét a következő sorba.

```
uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0x15, 0xC7, 0xFC};
```

Ezután létrehozunk egy struktúrát, amely tartalmazza a küldeni kívánt adatokat. Ezt a struktúrát `struct_message`-nek neveztük el, és három egész változót tartalmaz: a tábla azonosítóját, `id`, az `x`-et és az `y`-t. Ezt megváltoztathatod, hogy bármilyen típusú változót küldj (de ne felejtsd el megváltoztatni a fogadó oldalon sem).

```
typedef struct struct_message {
  int id; // must be unique for each sender board
  int x;
```

```
  int y;
} struct_message;
```

Létrehozunk egy új `struct_message` típusú változót `myData` néven, amely tárolja a változók értékeit.

```
struct_message myData;
```

Létrehozunk egy `esp_now_peer_info_t` típusú változót a társ információinak tárolására.

```
esp_now_peer_info_t peerInfo;
```

Ezután meghatározzuk az `OnDataSent()` függvényt. Ez egy visszahívási funkció, amely az üzenet elküldésekor kerül végrehajtásra. Ebben az esetben ez a funkció kinyomtatja, hogy az üzenetet sikeresen kézbesítették-e vagy sem.

```
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");
}
```

A `setup()`-ban inicializáljuk a soros monitort hibakeresési célból:

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

Beállítjuk az eszközt Wi-Fi állomásként:

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

Az ESP-NOW inicializálása:

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

Az ESP-NOW sikeres inicializálása után regisztráljuk a visszahívási funkciót, amelyet az üzenet elküldésekor hív meg. Ebben az esetben regisztráljunk a korábban létrehozott `OnDataSent()` függvényre.

```
esp_now_register_send_cb(OnDataSent);
```

Ahhoz, hogy adatokat küldjünk egy másik kártyára (a vevőre), párosítani kell azt társként. A következő sorok regisztrálnak, és új társat adnak hozzá.

```
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;
}
```

A `loop()`-ban 10 másodpercenként üzenetet küldünk az ESP-NOW-on keresztül (ez a késleltetési idő módosítható). Minden változóhoz rendeljünk értéket.

```
myData.id = 1;
myData.x = random(0,50);
myData.y = random(0,50);
```

Ne feledd, hogy a `myData` egy struktúra. Itt rendelheted hozzá azokat az értékeket, amelyeket el szeretnél küldeni a struktúrán belül. Ebben az esetben csak az azonosítót, `id`-t és a véletlenszerű `x` és `y` értékeket küldjük el. Gyakorlati alkalmazásban ezeket például parancsokkal vagy érzékelők leolvasásával kell helyettesíteni.

Végül elküldjük az üzenetet az ESP-NOW-on keresztül.

```
// 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");
}
```

ESP32 vevőkód (ESP-NOW)

Töltsd fel a következő kódot az ESP32 vevőkártyára. A kód három különböző kártyáról származó adatok fogadására készült. Könnyedén módosíthatod a kódot, hogy különböző számú tábláról fogadjon adatokat.

```
/*********
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp-now-many-
to-one-esp32/
  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 <esp_now.h>
#include <WiFi.h>

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
  int id;
  int x;
```

```
  int y;
}struct_message;

// Create a struct_message called myData
struct_message myData;

// Create a structure to hold the readings from each board
struct_message board1;
struct_message board2;
struct_message board3;

// Create an array with all the structures
struct_message boardsStruct[3] = {board1, board2, board3};

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int
len) {
  char macStr[18];
  Serial.print("Packet received from: ");
  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]);
  Serial.println(macStr);
  memcpy(&myData, incomingData, sizeof(myData));
  Serial.printf("Board ID %u: %u bytes\n", myData.id, len);
  // Update the structures with the new incoming data
  boardsStruct[myData.id-1].x = myData.x;
  boardsStruct[myData.id-1].y = myData.y;
  Serial.printf("x value: %d \n", boardsStruct[myData.id-1].x);
  Serial.printf("y value: %d \n", boardsStruct[myData.id-1].y);
  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;
  }

  // Once ESPNow is successfully Init, we will register for recv CB to
  // get recv packer info
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
}

void loop() {
  // Acess the variables for each board
  /*int board1X = boardsStruct[0].x;
```

```
  int board1Y = boardsStruct[0].y;
  int board2X = boardsStruct[1].x;
  int board2Y = boardsStruct[1].y;
  int board3X = boardsStruct[2].x;
  int board3Y = boardsStruct[2].y;*/

  delay(10000);
}
```

Hogyan működik a kód

A küldőhöz hasonlóan kezdjük a könyvtárak felvételével:

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

Létrehozunk egy struktúrát az adatok fogadásához. Ennek a szerkezetnek meg kell egyeznie a küldő vázlatában meghatározottakkal.

```
typedef struct struct_message {
  int id;
  int x;
  int y;
} struct_message;
```

Létrehozunk egy `myData` nevű `struct_message` változót, amely tárolja a kapott adatokat.

```
struct_message myData;
```

Ezután minden táblához létrehozunk egy `struct_message` változót, hogy a kapott adatokat hozzárendelhessük a megfelelő táblához. Itt három küldő táblához hozunk létre struktúrákat. Ha több küldő táblád van, több struktúrát kell létrehoznod.

```
struct_message board1;
struct_message board2;
struct_message board3;
```

Létrehozunk egy tömböt, amely tartalmazza az összes táblaszerkezetet. Ha eltérő számú táblát használsz, módosítanod kell.

```
struct_message boardsStruct[3] = {board1, board2, board3};
```

Létrehozunk egy visszahívási funkciót, amely akkor kerül meghívásra, amikor az ESP32 fogadja az adatokat az ESP-NOW-on keresztül. A függvényt `onDataRecv()`-nek hívják, és több paramétert is el kell fogadnia az alábbiak szerint:

```
void OnDataRecv(const uint8_t * mac, const uint8_t *incomingData, int len)
{
```

Szerezzük meg a tábla MAC-címét:

```
char macStr[18];
Serial.print("Packet received from: ");
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]);
Serial.println(macStr);
```

Másoljuk be `az incomingData` adatváltozó tartalmát a `myData` változóba.

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

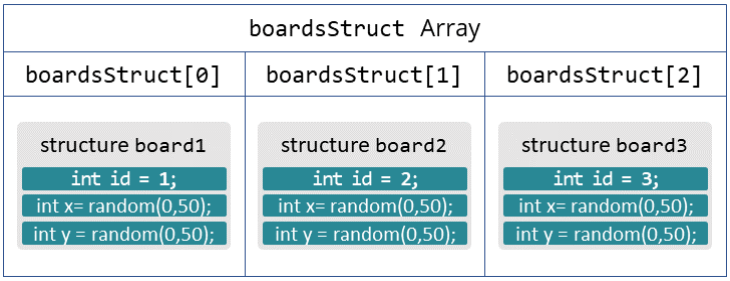
Most a `myData` struktúra több olyan változót tartalmaz, amelyek értékeit az egyik ESP32 küldő küldte. Azonosítható, hogy melyik kártya küldte a csomagot az azonosítója (`id`) alapján: `myData.id`.

Így a kapott értékeket hozzárendelhetjük a `boardsStruct` tömb megfelelő táblájához:

```
boardsStruct[myData.id-1].x = myData.x;
boardsStruct[myData.id-1].y = myData.y;
```

Képzeld el például, hogy egy csomagot kapsz a táblától 2-es azonosítóval. A `myData.id` értéke 2.

Tehát frissíteni szeretnéd a `board2` struktúra értékeit. A `board2` szerkezet a `boardsStruct` tömb 1 indexű eleme. Ezért vonunk ki 1-et, mert a C-beli tömbök indexelése 0-tól indul. Segíthet, ha megnézed a következő képet.



A `setup()`-ban inicializáljuk a soros monitort.

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

Beállítjuk az eszközt Wi-Fi állomásként.

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

Az ESP-NOW inicializálása:

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

Regisztrálunk egy visszahívási funkcióra, amelyet az adatok fogadásakor hív meg. Ebben az esetben regisztrálunk a korábban létrehozott `OnDataRecv()` függvényre.

```
esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
```

A ciklushoz kommentált következő sorok azt mutatják be, hogy mit kell tenni, ha hozzá szeretnénk férni az egyes táblaszerkezetek változóihoz. Például a `board1` `x` értékének eléréséhez:

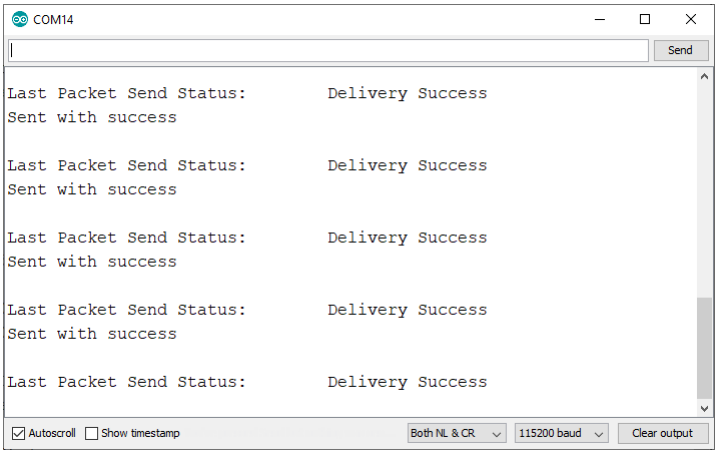
```
int board1X = boardsStruct[0].x;
```

Demonstráció

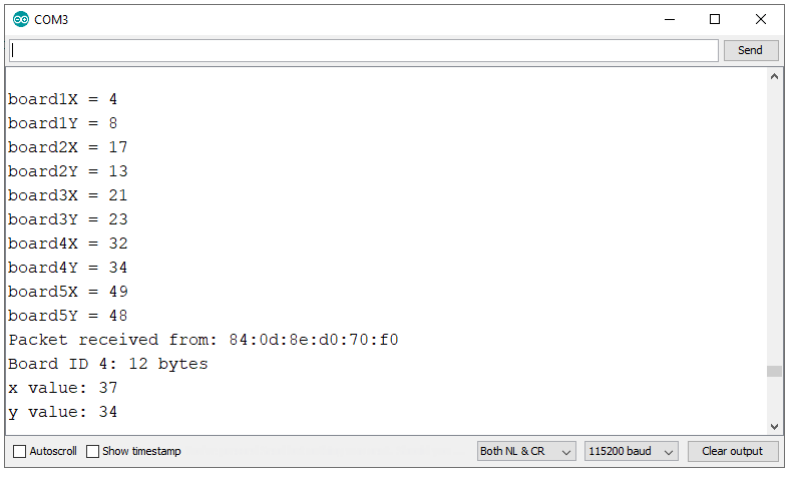
Töltsd fel a küldőkódot minden küldő táblára. Ne felejts el minden táblához más azonosítót adni.

Töltse fel a vevő kódját az ESP32 vevőkártyára. Ne felejtsd el módosítani a szerkezetet, hogy megfeleljen a küldőtáblák számának.

A küldő soros monitorán a "Delivery Success" („Kézbesítés sikeres”) üzenetet kell kapni, ha az üzeneteket megfelelően kézbesítették.



A vevőkártyán meg kell kapni a csomagokat az összes többi táblától. Ebben a tesztben 5 különböző tábláról kaptunk adatokat.



Lezárás

Ebben az oktatóanyagban megtanultad, hogyan állíthatsz be egy ESP32-t több ESP32 kártyáról történő adatfogadásra az ESP-NOW (több az egyhez konfiguráció) használatával.

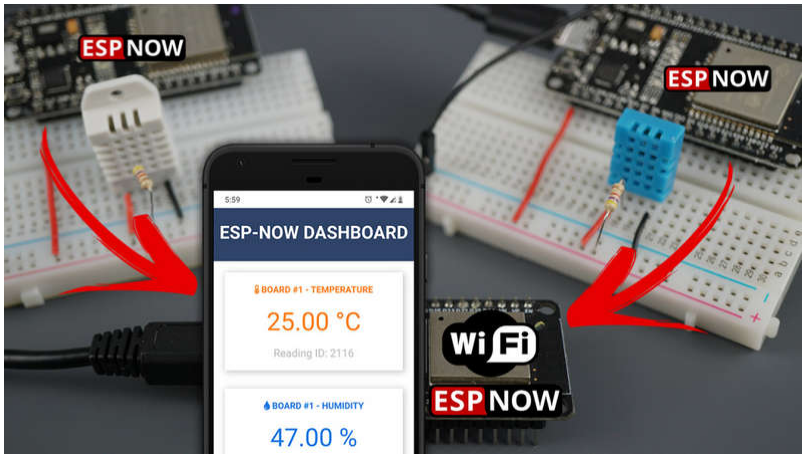
Példaként véletlenszerű számokat küldtünk. Valós alkalmazásban ezek helyettesíthetők tényleges szenzorleolvasásokkal vagy parancsokkal. Ez ideális, ha több érzékelő csomópontról szeretnél adatokat gyűjteni. Tovább viheted ezt a projektet, és létrehozhatsz egy webszervert a fogadó táblán, amely megjeleníti a fogadott üzeneteket.

A Wi-Fi használatához, a webszerver létrehozásához és az ESP-NOW egyidejű használatához be kell állítani az ESP32-t Wi-Fi állomásként és hozzáférési pontként. Ezenkívül be kell állítani egy másik Wi-Fi-csatornát: az egyiket az ESP-NOW-hoz, a másikat az állomáshoz. Követheted a következő oktatóanyagot:

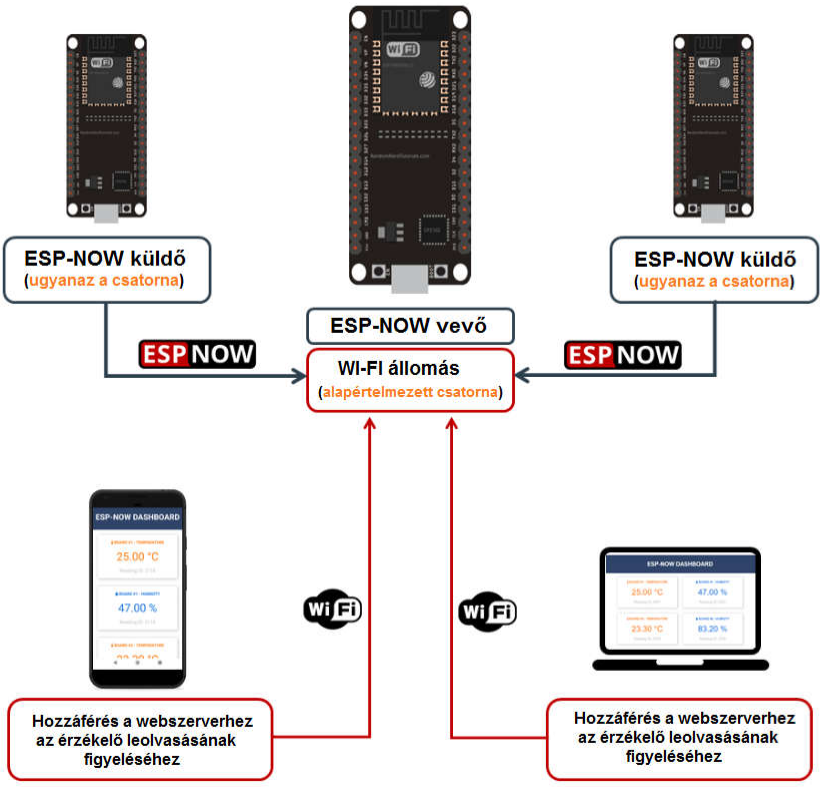
- ESP32: ESP-NOW webszerver érzékelő műszerfala (ESP-NOW + Wi-Fi)

5. ESP32: ESP-NOW webszerver érzékelő műszerfala (ESP-NOW + Wi-Fi)

Ebben a projektben megtanulhatod, hogyan üzemeltethetsz ESP32 webszervert, és hogyan használhatod egyszerre az ESP-NOW kommunikációs protokollt. Több ESP32 kártya is elküldheti az érzékelő leolvasásait az ESP-NOW-on keresztül egyetlen ESP32 vevőnek, amely megjeleníti az összes leolvasást a webszerveren. A kártyák programozása Arduino IDE segítségével történik.



Az ESP-NOW és a Wi-Fi egyidejű használata

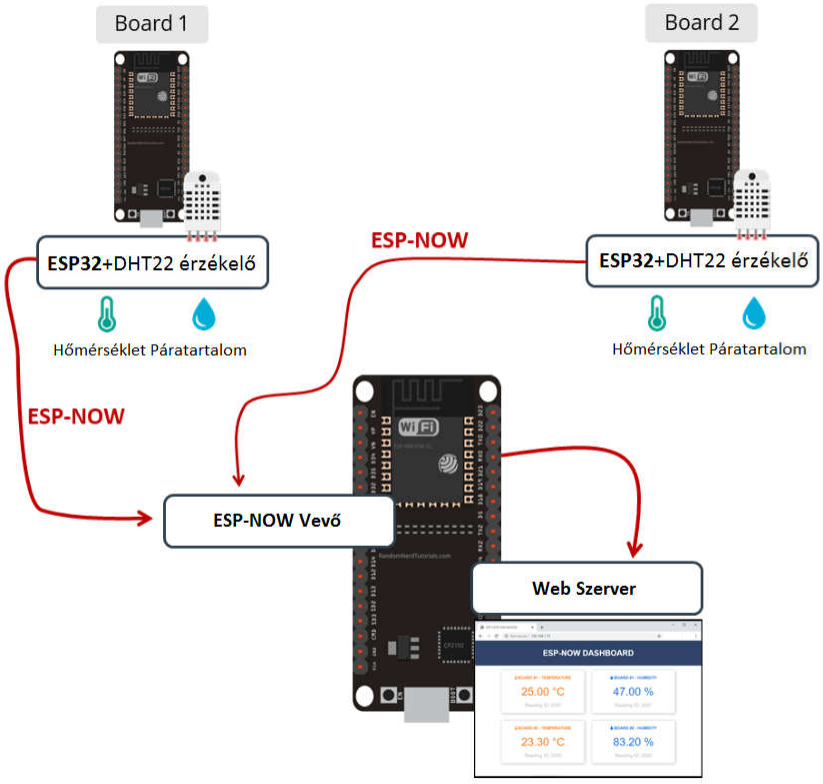


Néhány dolgot figyelembe kell venned, ha Wi-Fi-t szeretnél használni egy webszerver hosztolására, és egyidejűleg az ESP-NOW-t az érzékelők leolvasásának fogadására más kártyákról:

- Az ESP32 küldőkártyáknak ugyanazt a Wi-Fi csatornát kell használniuk, mint a vevőkártyának.
- A vevőkártya Wi-Fi csatornáját a Wi-Fi útválasztó automatikusan hozzárendeli.
- A vevőkártya Wi-Fi üzemmódjának hozzáférési pont és állomás (`WIFI_AP_STA`) kell lennie.
- Ugyanazt a Wi-Fi-csatornát beállíthatod manuálisan is, vagy hozzáadhatsz egy egyszerű kódpörgést a küldőhöz, hogy a Wi-Fi-csatornáját a vevőkártyával azonosra állítsd.

A projekt áttekintése

A következő diagram egy magas szintű áttekintést nyújt az általunk megépítendő projektről.



- Két ESP32 küldőkártyát használunk, amelyek DHT22 hőmérséklet- és páratartalom-leolvasásokat küldenek az ESP-NOW-on keresztül egy ESP32 vevőkártyára (ESP-NOW több az egy konfigurációhoz);
- Az ESP32 vevőkártya fogadja a csomagokat, és megjeleníti a leolvasásokat egy webszerveren;
- A webszerver automatikusan frissül minden alkalommal, amikor új olvasást kap a szerver által küldött események (SSE) használatával.

Előfeltételek

Mielőtt folytatnád ezt a projektet, győződj meg arról, hogy ellenőrizted a következő előfeltételeket.

Arduino IDE

Az ESP32 kártyákat Arduino IDE segítségével programozzuk, ezért mielőtt folytatnád ezt az oktatóanyagot, győződj meg arról, hogy az ESP32 kártya telepítve van az Arduino IDE-ben.

- Az ESP32 tábla telepítése Arduino IDE-ben (Windows, Mac OS X és Linux)

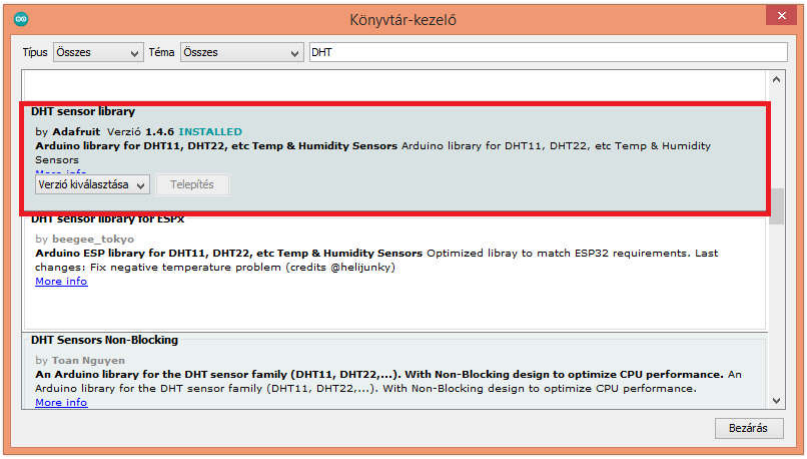
DHT könyvtárak

Az ESP32 küldőkártya hőmérséklet- és páratartalom-leolvasásokat küld a DHT22 érzékelőről.

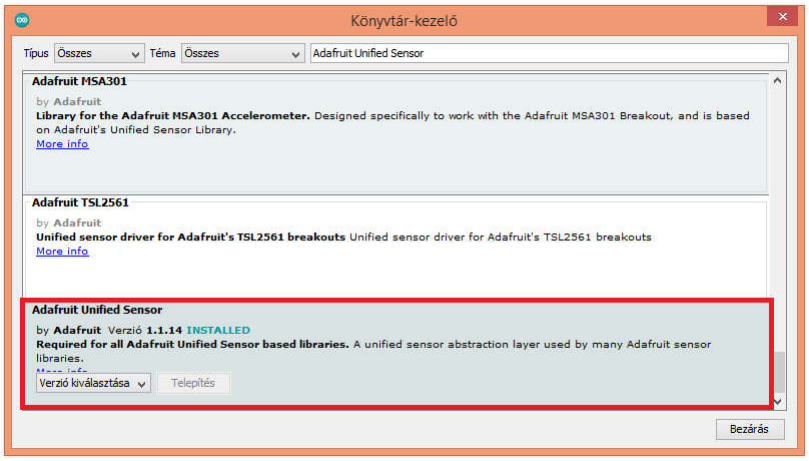
A DHT-érzékelőről való olvasáshoz az Adafruit DHT-könyvtárát használjuk. A könyvtár használatához telepíteni kell az Adafruit Unified Sensor könyvtárat is. Kövesd a következő lépéseket a könyvtárak telepítéséhez.

1. Nyisd meg Arduino IDE-jét, és lépj a **Vázlat > Könyvtár tartalmazása > Könyvtárak kezelése** menüpontra. A Könyvtárkezelőnek meg kell nyílnia.

2. Keresd meg a „**DHT**” kifejezést a Keresés mezőben, és telepítsd a DHT könyvtárat az Adafruittól.



3. Miután telepítetted a DHT-könyvtárat az Adafruittól, írd be az „**Adafruit Unified Sensor**” kifejezést a keresőmezőbe. Görgess le egészen a könyvtár megkereséséhez és telepítéséhez.



A könyvtárak telepítése után indítsd újra az Arduino IDE-t.

Aszinkron webszerver-könyvtárak

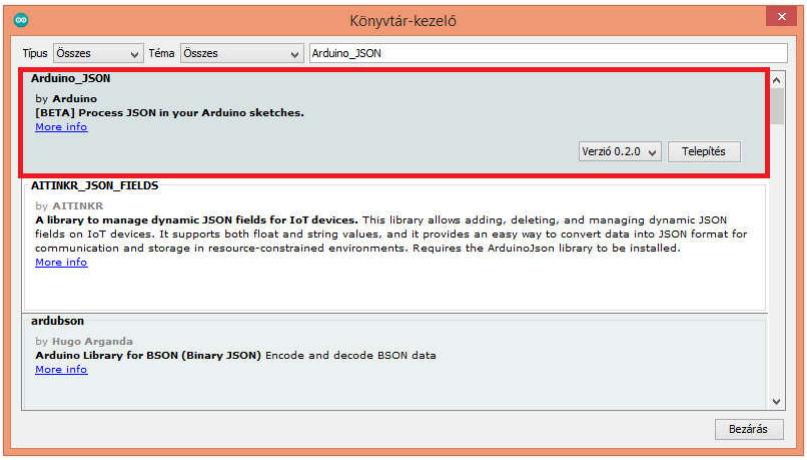
A webszerver felépítéséhez telepíteni kell a következő könyvtárakat:

- ESPAsyncWebServer
- AsyncTCP

Ezek a könyvtárak nem telepíthetők az Arduino Könyvtárkezelőn keresztül, ezért át kell másolni a könyvtár fájljait az Arduino telepítési könyvtárak mappájába. Alternatív megoldásként az Arduino IDE-ben a **Vázlat > Könyvtárak tartalmazása >.zip könyvtár hozzáadása** menüpontra léphet, és kiválaszthatja az éppen letöltött könyvtárakat.

Arduino_JSON könyvtár

Telepíteni kell az Arduino_JSON könyvtárat. Ezt a könyvtárat az Arduino IDE Könyvtárkezelőben telepítheted. Csak lépj a **Vázlat > Könyvtár tartalmazása > Könyvtárak kezelése** menüpontra, és keresd meg a könyvtár nevét az alábbiak szerint:



Szükséges alkatrészek

Az oktatóanyag követéséhez több ESP32 kártyára van szükséged. Három ESP32 kártyát fogunk használni. Szükséged lesz még:

- 3x ESP32 (olvasd el: a legjobb ESP32 fejlesztőkártyák)
- 2x DHT22 hőmérséklet és páratartalom érzékelő – DHT útmutató az ESP32-hez
- 2x 4,7 kOhm-os ellenállás
- Bekötőkártya
- Jumper vezetékek

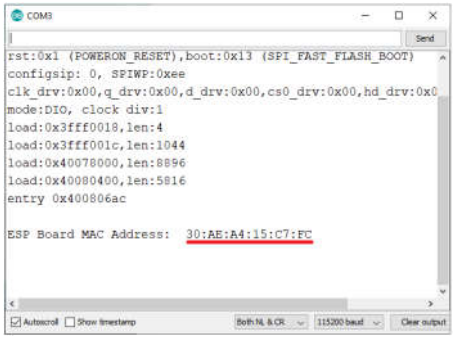
A vevőkártya MAC-címének lekérése

Ha üzeneteket szeretnénk küldeni az ESP-NOW-on keresztül, ismerni kell a vevőkártya MAC-címét. Minden kártyának egyedi MAC-címe van (további információ az ESP32 MAC-cím beszerzéséről és módosításáról).

Töltsd fel a következő kódot az ESP32 vevőkártyára, hogy megkapd a MAC-címét.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/get-change-
esp32-esp8266-mac-address-arduino/
  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.
*/
#ifdef ESP32
  #include <WiFi.h>
  #include <esp_wifi.h>
#else
  #include <ESP8266WiFi.h>
#endif

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

  Serial.print("ESP Board MAC Address: ");
  #ifdef ESP32
    WiFi.mode(WIFI_STA);
    WiFi.STA.begin();
    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");
    }
  #else
    Serial.println(WiFi.macAddress());
  #endif
}

void loop(){

}
```

A kód feltöltése után nyomd meg az RST/EN gombot, és a MAC címnek meg kell jelennie a soros monitoron.



ESP32 vevő (ESP-NOW + webszerver)

Az ESP32 vevőkártya fogadja a csomagokat a küldő kártyáktól, és egy webszervert tárol a legutóbbi fogadott adatok megjelenítéséhez.

Töltsd fel a következő kódot a vevőkártyára – a kód két különböző kártya olvasására készült.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp32-esp-
now-wi-fi-web-server/
  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 <esp_now.h>
#include <WiFi.h>
#include "ESPAsyncWebServer.h"
#include <Arduino_JSON.h>

// Replace with your network credentials (STATION)
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Structure example to receive data
// Must match the sender structure
typedef struct struct_message {
  int id;
  float temp;
  float hum;
  unsigned int readingId;
} struct_message;

struct_message incomingReadings;

JSONVar board;

AsyncWebServer server(80);
AsyncEventSource events("/events");

// callback function that will be executed when data is received
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int
len) {
  // Copies the sender mac address to a string
  char macStr[18];
  Serial.print("Packet received from: ");
  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]);
  Serial.println(macStr);
  memcpy(&incomingReadings, incomingData, sizeof(incomingReadings));
```

```
  board["id"] = incomingReadings.id;
  board["temperature"] = incomingReadings.temp;
  board["humidity"] = incomingReadings.hum;
  board["readingId"] = String(incomingReadings.readingId);
  String jsonString = JSON.stringify(board);
  events.send(jsonString.c_str(), "new_readings", millis());

  Serial.printf("Board ID %u: %u bytes\n", incomingReadings.id, len);
  Serial.printf("t value: %4.2f \n", incomingReadings.temp);
  Serial.printf("h value: %4.2f \n", incomingReadings.hum);
  Serial.printf("readingID value: %d \n", incomingReadings.readingId);
  Serial.println();
}

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <title>ESP-NOW DASHBOARD</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
  <link rel="icon" href="data:,">
  <style>
    html {font-family: Arial; display: inline-block; text-align: center;}
    p {  font-size: 1.2rem;}
    body {  margin: 0;}
    .topnav { overflow: hidden; background-color: #2f4468; color: white;
font-size: 1.7rem; }
    .content { padding: 20px; }
    .card { background-color: white; box-shadow: 2px 2px 12px 1px
rgba(140,140,140,.5); }
    .cards { max-width: 700px; margin: 0 auto; display: grid; grid-gap:
2rem; grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); }
    .reading { font-size: 2.8rem; }
    .packet { color: #bebebe; }
    .card.temperature { color: #fd7e14; }
    .card.humidity { color: #1b78e2; }
  </style>
</head>
<body>
  <div class="topnav">
    <h3>ESP-NOW DASHBOARD</h3>
  </div>
  <div class="content">
    <div class="cards">
      <div class="card temperature">
        <h4><i class="fas fa-thermometer-half"></i> BOARD #1 -
TEMPERATURE</h4><p><span class="reading"><span id="t1"></span>
&deg;C</span></p><p class="packet">Reading ID: <span id="rt1"></span></p>
      </div>
      <div class="card humidity">
```

```
        <h4><i class="fas fa-tint"></i> BOARD #1 - HUMIDITY</h4><p><span
class="reading"><span id="h1"></span> &percnt;</span></p><p
class="packet">Reading ID: <span id="rh1"></span></p>
      </div>
      <div class="card temperature">
        <h4><i class="fas fa-thermometer-half"></i> BOARD #2 -
TEMPERATURE</h4><p><span class="reading"><span id="t2"></span>
&deg;C</span></p><p class="packet">Reading ID: <span id="rt2"></span></p>
      </div>
      <div class="card humidity">
        <h4><i class="fas fa-tint"></i> BOARD #2 - HUMIDITY</h4><p><span
class="reading"><span id="h2"></span> &percnt;</span></p><p
class="packet">Reading ID: <span id="rh2"></span></p>
      </div>
    </div>
  </div>
<script>
if (!!window.EventSource) {
 var source = new EventSource('/events');

 source.addEventListener('open', function(e) {
  console.log("Events Connected");
 }, false);
 source.addEventListener('error', function(e) {
  if (e.target.readyState != EventSource.OPEN) {
    console.log("Events Disconnected");
  }
 }, false);

 source.addEventListener('message', function(e) {
  console.log("message", e.data);
 }, false);

 source.addEventListener('new_readings', function(e) {
  console.log("new_readings", e.data);
  var obj = JSON.parse(e.data);
  document.getElementById("t"+obj.id).innerHTML =
obj.temperature.toFixed(2);
  document.getElementById("h"+obj.id).innerHTML = obj.humidity.toFixed(2);
  document.getElementById("rt"+obj.id).innerHTML = obj.readingId;
  document.getElementById("rh"+obj.id).innerHTML = obj.readingId;
 }, false);
}
</script>
</body>
</html>)rawliteral";

void setup() {
  // Initialize Serial Monitor
  Serial.begin(115200);

  // Set the device as a Station and Soft Access Point simultaneously
  WiFi.mode(WIFI_AP_STA);
```

```
  // Set device as a Wi-Fi Station
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Setting as a Wi-Fi Station..");
  }
  Serial.print("Station IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Wi-Fi Channel: ");
  Serial.println(WiFi.channel());

  // 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 recv CB to
  // get recv packer info
  esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html);
  });

  events.onConnect([](AsyncEventSourceClient *client){
    if(client->lastId()){
      Serial.printf("Client reconnected! Last message ID that it got is:
%u\n", client->lastId());
    }
    // send event with message "hello!", id current millis
    // and set reconnect delay to 1 second
    client->send("hello!", NULL, millis(), 10000);
  });
  server.addHandler(&events);
  server.begin();
}

void loop() {
  static unsigned long lastEventTime = millis();
  static const unsigned long EVENT_INTERVAL_MS = 5000;
  if ((millis() - lastEventTime) > EVENT_INTERVAL_MS) {
    events.send("ping",NULL,millis());
    lastEventTime = millis();
  }
}
```

Hogyan működik a kód

Először is felvesszük a szükséges könyvtárakat.

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

```
#include "ESPAsyncWebServer.h"
#include <Arduino_JSON.h>
```

Az `Arduino_JSON` könyvtárra azért van szükség, mert létrehozunk egy JSON-változót az egyes táblákról kapott adatokkal. Ezt a JSON-változót a rendszer az összes szükséges információ elküldésére fogja használni a weboldalra, amint azt a projektben később látni fogod.

Írd be a hálózat hitelesítő adatait a következő sorokba, hogy az ESP32 csatlakozhasson a helyi hálózathoz.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Adatstruktúra

Ezután létrehozunk egy struktúrát, amely tartalmazza a kapott adatokat. Ezt a struktúrát `struct_message`-nek neveztük el, és ez tartalmazza a kártya azonosítóját, a hőmérséklet és páratartalom értékeket, valamint az olvasási azonosítót.

```
typedef struct struct_message {
    int id;
    float temp;
    float hum;
    int readingId;
} struct_message;
```

Létrehozunk egy új `struct_message` típusú változót, amelyet `incomingReadings`-nek nevezünk, és amely tárolja a változók értékeit.

```
struct_message incomingReadings;
```

Létrehozunk egy `board` nevű JSON-változót.

```
JSONVar board;
```

Létrehozunk egy aszinkron webszervert a 80-as porton.

```
AsyncWebServer server(80);
```

Eseményforrás létrehozása

Az információk automatikus megjelenítéséhez a webszerveren, amikor új leolvasás érkezik, a szerver által küldött eseményeket (SSE) használjuk.

A következő sor új eseményforrást hoz létre az `/events` mappában.

```
AsyncEventSource events("/events");
```

A szerver által küldött események lehetővé teszik, hogy egy weboldal (kliens) frissítéseket kapjon a szervertől. Ezt arra használjuk, hogy automatikusan megjelenítsük az új értékeket a webszerver oldalon, amikor új ESP-NOW csomag érkezik.

Az OnDataRecv() függvény

Az `OnDataRecv()` függvény akkor kerül végrehajtásra, amikor új ESP-NOW csomagot kap.

```
void OnDataRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int
len) {
```

A függvényen belül kinyomtjuk a feladó MAC-címét:

```
// Copies the sender mac address to a string
char macStr[18];
Serial.print("Packet received from: ");
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]);
Serial.println(macStr);
```

Másoljuk az `incomingData` változóban található információkat az `incomingReadings` szerkezeti változóba.

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

Ezután létrehozunk egy JSON String változót a kapott információval (`jsonString` változó):

```
board["id"] = incomingReadings.id;
board["temperature"] = incomingReadings.temp;
board["humidity"] = incomingReadings.hum;
board["readingId"] = String(incomingReadings.readingId);
String jsonString = JSON.stringify(board);
```

Íme egy példa arra, hogyan nézhet ki a `jsonString` változó a kiolvasások fogadása után:

```
board = {
  "id": "1",
  "temperature": "24.32",
  "humidity" = "65.85",
  "readingId" = "2"
}
```

Miután összegyűjtöttük az összes kapott adatot a `jsonString` változóból, elküldjük ezeket az információkat a böngészőnek eseményként („`new_readings`”).

```
events.send(jsonString.c_str(), "new_readings", millis());
```

Később meglátjuk, hogyan kezeljük ezeket az eseményeket az ügyféloldalon.

Végül kinyomtjuk a kapott információkat az Arduino IDE soros monitoron hibakeresési célból:

```
Serial.printf("Board ID %u: %u bytes\n", incomingReadings.id, len);
Serial.printf("t value: %4.2f \n", incomingReadings.temp);
Serial.printf("h value: %4.2f \n", incomingReadings.hum);
```

```
Serial.printf("readingID value: %d \n", incomingReadings.readingId);
Serial.println();
```

Weboldal építése

Az `index_html` változó tartalmazza a weboldal elkészítéséhez szükséges összes HTML-t, CSS-t és JavaScriptet. Nem megyünk bele a HTML és a CSS működésének részleteibe. Csak megnézzük, hogyan kezeljük a szerver által küldött eseményeket.

Létrehozunk egy új `EventSource` objektumot, és megadjuk a frissítéseket küldő oldal URL-jét. Esetünkben ez az `/events`.

```
if (!!window.EventSource) {
 var source = new EventSource('/events');
```

Miután példányosítottunk egy eseményforrást, elkezdhetjük figyelni a szerverről érkező üzeneteket az `addEventListener()` segítségével.

Ezek az alapértelmezett eseményfigyelők, amint az itt látható az AsyncWebServer dokumentációjában.

```
source.addEventListener('open', function(e) {
  console.log("Events Connected");
}, false);
source.addEventListener('error', function(e) {
 if (e.target.readyState != EventSource.OPEN) {
   console.log("Events Disconnected");
 }
}, false);

source.addEventListener('message', function(e) {
 console.log("message", e.data);
}, false);
```

Ezután hozzáadjuk az eseményfigyelőt az „`new_readings`”-ként.

```
source.addEventListener('new_readings', function(e) {
```

Amikor az ESP32 új csomagot kap, egy JSON-karakterláncot küld a kiolvasásokkal eseményként („`new_readings`”) az ügyfélnek. A következő sorok azzal foglalkoznak, hogy mi történik, amikor a böngésző megkapja az eseményt.

```
console.log("new_readings", e.data);
var obj = JSON.parse(e.data);
document.getElementById("t"+obj.id).innerHTML = obj.temperature.toFixed(2);
document.getElementById("h"+obj.id).innerHTML = obj.humidity.toFixed(2);
document.getElementById("rt"+obj.id).innerHTML = obj.readingId;
document.getElementById("rh"+obj.id).innerHTML = obj.readingId;
```

Alapvetően a böngésző konzolon kinyomtatja az új olvasatokat, és tegye a kapott adatokat a weboldal megfelelő azonosítójával rendelkező elemekbe.

setup()

A `setup()`-ban beállítjuk az ESP32 vevőt hozzáférési pontként és Wi-Fi állomásként:

```
WiFi.mode(WIFI_AP_STA);
```

A következő sorok csatlakoztatják az ESP32-t a helyi hálózathoz, és kinyomtatják az IP-címet és a Wi-Fi csatornát:

```
// Set device as a Wi-Fi Station
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Setting as a Wi-Fi Station..");
}
Serial.print("Station IP Address: ");
Serial.println(WiFi.localIP());
Serial.print("Wi-Fi Channel: ");
Serial.println(WiFi.channel());
```

Inicializáljuk az ESP-NOW-t.

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

Regisztráljunk az `OnDataRecv` visszahívási funkcióra, hogy az új ESP-NOW csomag megérkezésekor végrehajtódjon.

```
esp_now_register_recv_cb(esp_now_recv_cb_t(OnDataRecv));
```

Kezeljük a kéréseket

Amikor elérjük az ESP32 IP-címét a gyökérben/URL-ben, küldje el az `index_html` változóban tárolt szöveget a weboldal felépítéséhez.

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html);
});
```

Szeveresemény forrása

Beállítjuk az eseményforrást a szerveren.

```
events.onConnect([](AsyncEventSourceClient *client){
  if(client->lastId()){
    Serial.printf("Client reconnected! Last message ID that it got is:
%u\n", client->lastId());
  }
  // send event with message "hello!", id current millis
```

```
  // and set reconnect delay to 1 second
  client->send("hello!", NULL, millis(), 10000);
 );
  server.addHandler(&events);
```

Végül indítsuk el a szervert.

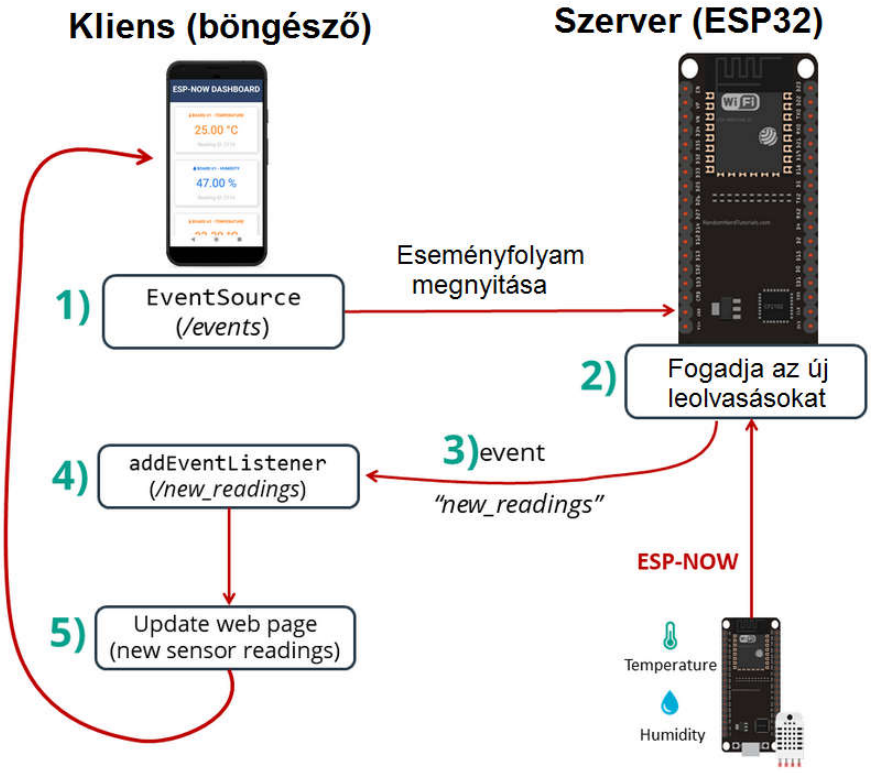
```
server.begin();
```

loop()

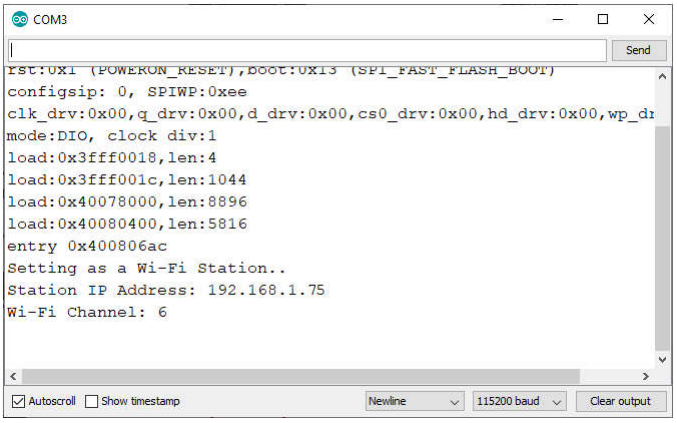
A `loop()`-ban egy pinget küldünk 5 másodpercenként. Ezzel ellenőrizhető a kliens oldalon, hogy a szerver még fut-e.

```
static unsigned long lastEventTime = millis();
static const unsigned long EVENT_INTERVAL_MS = 5000;
if ((millis() - lastEventTime) > EVENT_INTERVAL_MS) {
  events.send("ping",NULL,millis());
  lastEventTime = millis();
}
```

A következő diagram összefoglalja, hogyan működnek a szerver által küldött események ezen a projekten, és hogyan frissíti az értékeket a weboldal frissítése nélkül.

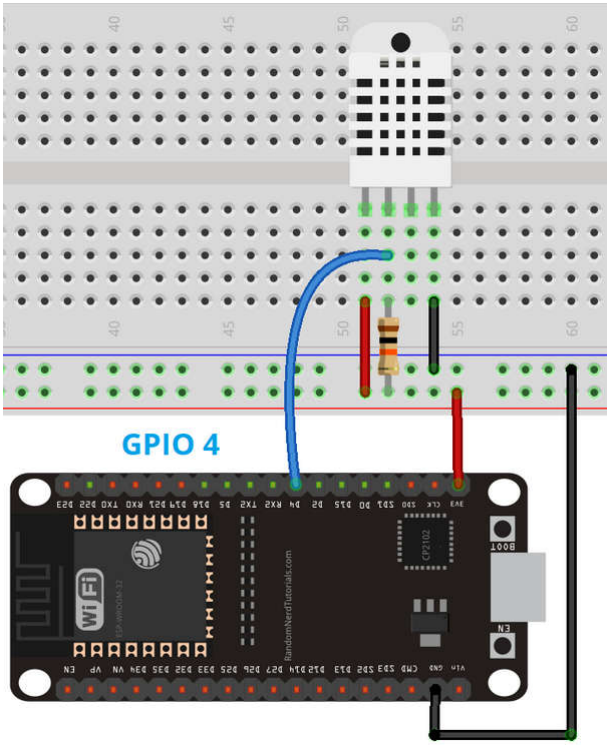


Miután feltöltötted a kódot a vevőkártyára, nyomd meg a fedélzeti EN/RST gombot. Az ESP32 IP-címét ki kell nyomtatni a soros monitorra, valamint a Wi-Fi csatornát.



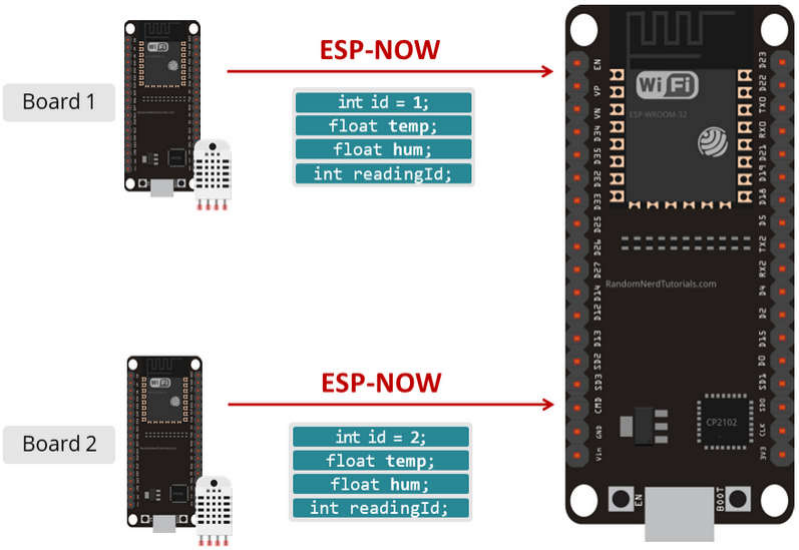
ESP32 küldő áramkör

Az ESP32 küldő kártyák egy DHT22 hőmérséklet- és páratartalom-érzékelőhöz csatlakoznak. Az adattű a GPIO 4-hez csatlakozik. Bármely másik megfelelő GPIO-t választhatsz (lásd az ESP32 Pinout Guide-ot). Kövesd a következő sematikus diagramot az áramkör bekötéséhez.



ESP32 küldőkód (ESP-NOW)

Minden küldő tábla küld egy szerkezetet az ESP-NOW-on keresztül, amely tartalmazza a kártya azonosítóját (hogy azonosítani tudja, melyik kártya küldte a leolvasást), a hőmérsékletet, a páratartalmat és az olvasási azonosítót. Az olvasási azonosító egy int szám, amelyből megtudható, hogy hány üzenetet küldtek el.



Töltsd fel a következő kódot mindegyik küldő táblára. Ne felejtsd el növelni az egyes küldőkártyák azonosítószámát, és beilleszteni az SSID-t a WIFI_SSID változóba.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  Complete project details at https://RandomNerdTutorials.com/esp32-esp-now-wi-fi-web-server/
  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 <esp_now.h>
#include <esp_wifi.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

// Set your Board ID (ESP32 Sender #1 = BOARD_ID 1, ESP32 Sender #2 = BOARD_ID 2, etc)
#define BOARD_ID 1

// Digital pin connected to the DHT sensor
#define DHTPIN 4

// Uncomment the type of sensor in use:
//#define DHTTYPE    DHT11     // DHT 11
#define DHTTYPE    DHT22     // DHT 22 (AM2302)
//#define DHTTYPE    DHT21     // DHT 21 (AM2301)

DHT dht(DHTPIN, DHTTYPE);

//MAC Address of the receiver
uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

//Structure example to send data
```

```
//Must match the receiver structure
typedef struct struct_message {
    int id;
    float temp;
    float hum;
    int readingId;
} struct_message;

esp_now_peer_info_t peerInfo;

//Create a struct_message called myData
struct_message myData;

unsigned long previousMillis = 0;   // Stores last time temperature was
published
const long interval = 10000;        // Interval at which to publish sensor
readings

unsigned int readingId = 0;

// Insert your SSID
constexpr char WIFI_SSID[] = "REPLACE_WITH_YOUR_SSID";

int32_t getWiFiChannel(const char *ssid) {
  if (int32_t n = WiFi.scanNetworks()) {
      for (uint8_t i=0; i<n; i++) {
          if (!strcmp(ssid, WiFi.SSID(i).c_str())) {
              return WiFi.channel(i);
          }
      }
  }
  return 0;
}

float readDHTTemperature() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  //float t = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return 0;
  }
  else {
    Serial.println(t);
    return t;
  }
}

float readDHTHumidity() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
```

```
sensor)
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return 0;
  }
  else {
    Serial.println(h);
    return h;
  }
}

// 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);

  dht.begin();

  // Set device as a Wi-Fi Station and set channel
  WiFi.mode(WIFI_STA);

  int32_t channel = getWiFiChannel(WIFI_SSID);

  WiFi.printDiag(Serial); // Uncomment to verify channel number before
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
  esp_wifi_set_promiscuous(false);
  WiFi.printDiag(Serial); // Uncomment to verify channel change after

  // 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.encrypt = false;

  // Add peer
  if (esp_now_add_peer(&peerInfo) != ESP_OK){
    Serial.println("Failed to add peer");
    return;
```

```
  }
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    // Save the last time a new reading was published
    previousMillis = currentMillis;
    //Set values to send
    myData.id = BOARD_ID;
    myData.temp = readDHTTemperature();
    myData.hum = readDHTHumidity();
    myData.readingId = readingId++;

    //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");
    }
  }
}
```

Hogyan működik a kód

Kezdjük a szükséges könyvtárak importálásával:

```
#include <esp_now.h>
#include <esp_wifi.h>
#include <WiFi.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

Beállítjuk a táblaazonosítót

Határozzuk meg az ESP32 küldőkártya azonosítóját, például állítsuk be a BOARD_ID 1 értéket az ESP32 Küldő #1 számára stb.

```
// Set your Board ID (ESP32 Sender #1 = BOARD_ID 1, ESP32 Sender #2 =
BOARD_ID 2, etc)
#define BOARD_ID 1
```

DHT érzékelő

Határozzuk meg azt a tűt, amelyhez a DHT-érzékelő csatlakoztatva van. Példánkban a -hez csatlakozik.

```
#define DHTPIN 4
```

Kiválasztjuk a használt DHT-érzékelő típusát. A projektben DHT22-t használunk.

```
// Uncomment the type of sensor in use:
//#define DHTTYPE    DHT11     // DHT 11
#define DHTTYPE    DHT22     // DHT 22 (AM2302)
//#define DHTTYPE    DHT21     // DHT 21 (AM2301)
```

Létrehozunk egy DHT objektumot a korábban definiált tűn, és típuson.

```
DHT dht(DHTPIN, DHTTYPE);
```

A vevő MAC-címe

Írjuk be a vevő MAC-címét a következő sorba (példa):

```
uint8_t broadcastAddress[] = {0x30, 0xAE, 0xA4, 0x15, 0xC7, 0xFC};
```

Adatstruktúra

Ezután létrehozunk egy struktúrát, amely tartalmazza a küldeni kívánt adatokat. A `struct_message` tartalmazza az alaplap azonosítóját, a hőmérsékleti leolvasást, a páratartalom leolvasását és az olvasó azonosítót.

```
typedef struct struct_message {
    int id;
    float temp;
    float hum;
    int readingId;
} struct_message;
```

Létrehozunk egy új, `struct_message` típusú változót `myData` néven, amely tárolja a változók értékeit.

```
struct_message myData;
```

Időzítő intervallum

Létrehozunk néhány kiegészítő időzítő változót a leolvasások 10 másodpercenkénti közzétételéhez. Módosíthatod a késleltetési időt az `interval` változón.

```
unsigned long previousMillis = 0;  // Stores last time temperature was
published
const long interval = 10000;       // Interval at which to publish sensor
readings
```

Inicializáljuk a `readingId` változót – nyomon követjük a küldött leolvasások számát.

```
unsigned int readingId = 0;
```

Wi-Fi csatorna váltása

Most megkapjuk a vevő Wi-Fi csatornáját. Ez azért hasznos, mert lehetővé teszi, hogy automatikusan hozzárendeljük ugyanazt a Wi-Fi csatornát a küldő táblához.

Ehhez be kell írnod az SSID-t a következő sorba:

```
constexpr char WIFI_SSID[] = "REPLACE_WITH_YOUR_SSID";
```

Ezután a `getWiFiChannel()` függvény megkeresi a hálózatot, és lekéri a csatornaszámot.

```
int32_t getWiFiChannel(const char *ssid) {
  if (int32_t n = WiFi.scanNetworks()) {
    for (uint8_t i=0; i<n; i++) {
      if (!strcmp(ssid, WiFi.SSID(i).c_str())) {
        return WiFi.channel(i);
      }
    }
  }
  return 0;
}
```

Ezt a kódrészletet Stephane (egyik olvasónk) javasolta. A teljes példáját itt tekintheted meg.

A hőmérséklet leolvasása

A `readDHTTemperature()` függvény leolvassa és visszaadja a DHT-érzékelő hőmérsékletét. Ha nem tudja leolvasni a hőmérsékletet, 0-t ad vissza.

```
float readDHTTemperature() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  //float t = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return 0;
  }
  else {
    Serial.println(t);
    return t;
  }
}
```

A páratartalom leolvasása

A `readDHTHumidity()` függvény leolvassa és visszaadja a DHT-érzékelő páratartalmát. Ha nem tud páratartalmat leolvasni, akkor 0-t ad vissza.

```
float readDHTHumidity() {
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow
sensor)
  float h = dht.readHumidity();
  if (isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
    return 0;
  }
  else {
    Serial.println(h);
    return h;
  }
}
```

Az OnDataSent visszahívási függvény

Az `OnDataSent()` visszahívási függvény üzenet elküldésekor kerül végrehajtásra. Ebben az esetben ez a függvény kinyomtatja, hogy az üzenetet sikeresen kézbesítették-e vagy sem.

```
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");
}
```

setup()

Inicializáljuk a soros monitort.

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

Beállítjuk az ESP32-t Wi-Fi állomásként.

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

Beállítjuk a csatornát úgy, hogy megegyezzen a vevő Wi-Fi csatornájával:

```
int32_t channel = getWiFiChannel(WIFI_SSID);

WiFi.printDiag(Serial); // Uncomment to verify channel number before
esp_wifi_set_promiscuous(true);
esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
esp_wifi_set_promiscuous(false);
WiFi.printDiag(Serial); // Uncomment to verify channel change after
```

Inicializáljuk az ESP-NOW-t.

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

Az ESP-NOW sikeres inicializálása után regisztráljuk a visszahívási függvényt, amelyet az üzenet elküldésekor hív meg. Ebben az esetben regisztrálunk a korábban létrehozott `OnDataSent()` függvényre.

```
esp_now_register_send_cb(OnDataSent);
```

Társ hozzáadása

Ahhoz, hogy adatokat küldjünk egy másik kártyára (a vevőre), párosítani kell azt társként. A következő sorok regisztrálják és hozzáadják a vevőt társként.

```
// Register peer
memcpy(peerInfo.peer_addr, broadcastAddress, 6);
peerInfo.encrypt = false;

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

loop()

A `loop()`-ban ellenőrizzük, hogy itt van-e az ideje új leolvasások beszerzésének és küldésének.

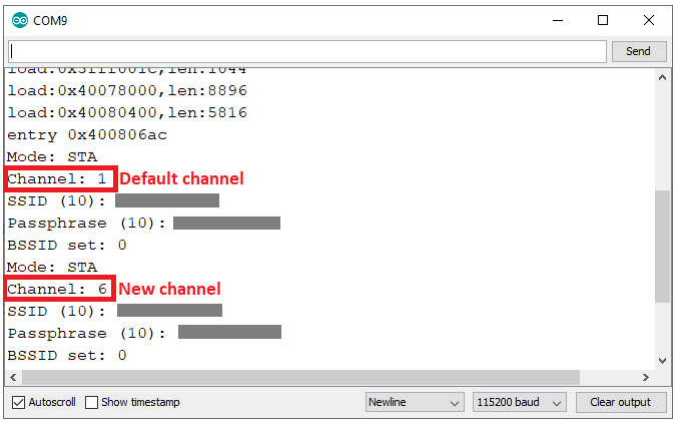
```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  // Save the last time a new reading was published
  previousMillis = currentMillis;
```

ESP-NOW üzenet küldése

Végül elküldjük az üzenet szerkezetét az ESP-NOW-on keresztül.

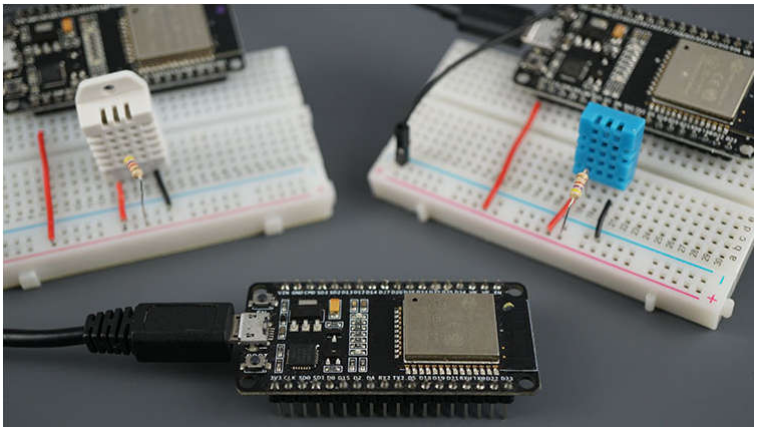
```
// Send message via ESP-NOW
esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &myData,
sizeof(myData));
if (result == ESP_OK) {
  Serial.println("Sent with success");
}
else {
  Serial.println("Error sending the data");
}
```

Töltsd fel a kódot a küldő táblákra. Figyeld meg, hogy a kártyák Wi-Fi csatornájukat a vevőkártya csatornájára váltják. Esetünkben a táblák Wi-Fi csatornaszámukat 6-ra cserélték.

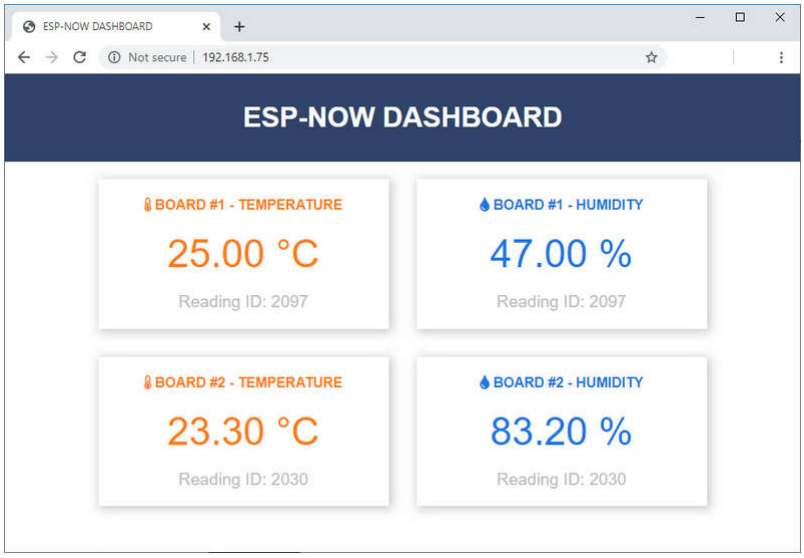


Demonstráció

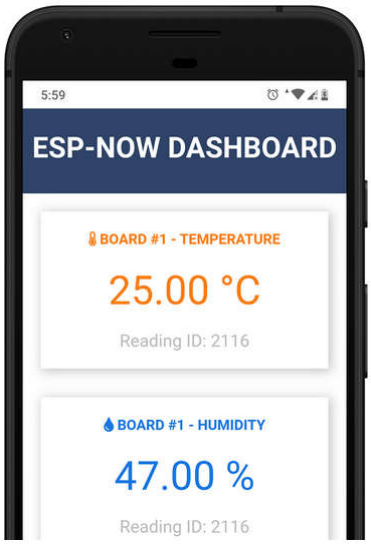
Miután feltöltötted a kódot az összes kártyára, és ha minden a várt módon megy, az ESP32 vevőkártyának el kell kezdenie fogadni a többi kártya érzékelőjeit.



Nyiss meg egy böngészőt a helyi hálózaton, és írd be az ESP32 IP-címét.



Be kell töltődnie a hőmérsékletnek, a páratartalomnak és az olvasó azonosítójának minden kártyához. Új csomag fogadásakor a weboldal automatikusan frissül a weboldal frissítése nélkül.



Lezárás

Ebben az oktatóanyagban megtanultad, hogyan használhatod az ESP-NOW-t és a Wi-Fi-t egy webszerver beállítására, amely több kártyáról fogadja az ESP-NOW csomagokat (több-az-egy konfiguráció).

Ezenkívül a szerver által küldött események funkciót is használtad a weboldal automatikus frissítésére minden alkalommal, amikor új csomag érkezik a weboldal frissítése nélkül.

Reméljük, hogy tetszett ez a projekt.

Szeretnénk köszönetet mondani olvasóinknak, akik segítettek az oktatóanyag továbbfejlesztésében, nevezetesen Stéphane Calderoninak és Lee Davidsonnak értékes hozzájárulásukért. Köszönöm.

ESP32 mélyaltatás és felébresztési források Arduino IDE-vel

Ez a cikk egy teljes útmutató az ESP32 mélyalvás módhoz Arduino IDE-vel. Megmutatjuk, hogyan helyezheted mélyalvásba az ESP32-t, és vetünk egy pillantást a különböző ébresztési módokra: időzített ébresztés, érintéssel történő ébresztés és külső ébresztés. Ez az útmutató gyakorlati példákat tartalmaz kóddal, kódmagyarázattal és kapcsolási rajzokkal.

Ez a cikk 4 különböző részre oszlik:

1. A mély alvó üzemmód bemutatása
2. Időzített ébresztés
3. Érintéses ébresztés
4. Külső ébresztés

A mély alvó üzemmód bemutatása

Az ESP32 különböző üzemmódok között válthat:

- Aktív mód
- Modem alvó mód
- Könnyű alvó üzemmód
- Mély alvó mód
- Hibernált mód

Összehasonlíthatod az öt különböző módot az ESP32 Espressif adatlap alábbi táblázatában.

Energia üzemmód	Aktív	Modem-alvó	Könnyű-alvó	Mély-alvó	Hibernált
Alvásminta		Társulási alvásminta		ULP érzékelő által felügyelt minta	-
CPU	BE	BE	SZÜNET	KI	KI

Wi-Fi/BT alapsáv és rádió	BE	KI	KI	KI	KI
RTC memória és RTC perifériák	BE	BE	BE	BE	KI
ULP koprocesszor	BE	BE	BE	BE/KI	KI

Az ESP32 Espressif adatlap egy táblázatot is tartalmaz, amely összehasonlítja a különböző energiafogyasztási módok energiafogyasztását.

Energia üzemmód	Leírás	Energiafogyasztás
Aktív (RF működik)	Wi-Fi Tx csomag 14 dBm ~ 19,5 dBm Wi-Fi / BT Tx csomag 0 dBm Wi-Fi/BT Rx és fülel	A részletekért lásd a 10. táblázatot
Modem-alvó	A CPU be van kapcsolva	Max sebesség 240 MHz: 30 mA ~ 50 mA Normál seb. 80 MHz: 20 mA ~ 25 mA Alacsony seb. 2 MHz: 2 mA ~ 4 mA
Könnyű-alvó	-	0,8 mA
Mély-alvó	Az ULP társprocesszor be van kapcsolva ULP érzékelő által felügyelt minta RTC időzítő + RTC memória	150 µA 100 µA @ 1% adó 10 µA
Hibernált	Csak RTC időzítő	5 µA
Kikapcsolt	A CPU alacsony szintre állítva, a chip ki van kapcsolva	0,1 µA

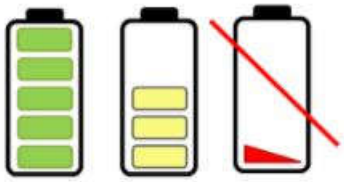
És itt van még a 10. táblázat az aktív mód energiafogyasztásának összehasonlításához:

10. táblázat: RF energiafogyasztás részletezése

Mód	Min	Tipikus	Max	Egység
Átvitel 802.11b, DSSS 1 Mbps, POUT = +19,5 dBm	-	240	-	mA
Átvitel 802.11b, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Átvitel 802.11g, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Vétel 802.11b/g/n	-	95~100	-	mA
Átvitel BT/BLE, POUT = +0 dBm	-	130	-	mA
Vétel BT/BLE	-	95~100	-	mA

Miért a mély alvó üzemmód?

Nem ideális, ha az ESP32 aktív üzemmódban működik akkumulátorokkal, mivel az akkumulátorok nagyon gyorsan lemerülnek.



Ha mélyalvás módba helyezed az ESP32-t, csökkented az energiafogyasztást, és tovább bírják az akkumulátorok.

Ha az ESP32 mélyalvás módban van azt jelenti, hogy el kell távolítani a több energiát fogyasztó tevékenységektől, de épp elég aktivitást tartunk meg ahhoz, hogy felébressze a processzort, ha valami érdekes történik.

Mélyalvó üzemmódban sem CPU nem működik, sem Wi-Fi-tevékenység nem történik, de az Ultra Low Power (ULP, ultra alacsony teljesítmény) társprocesszor továbbra is bekapcsolható.

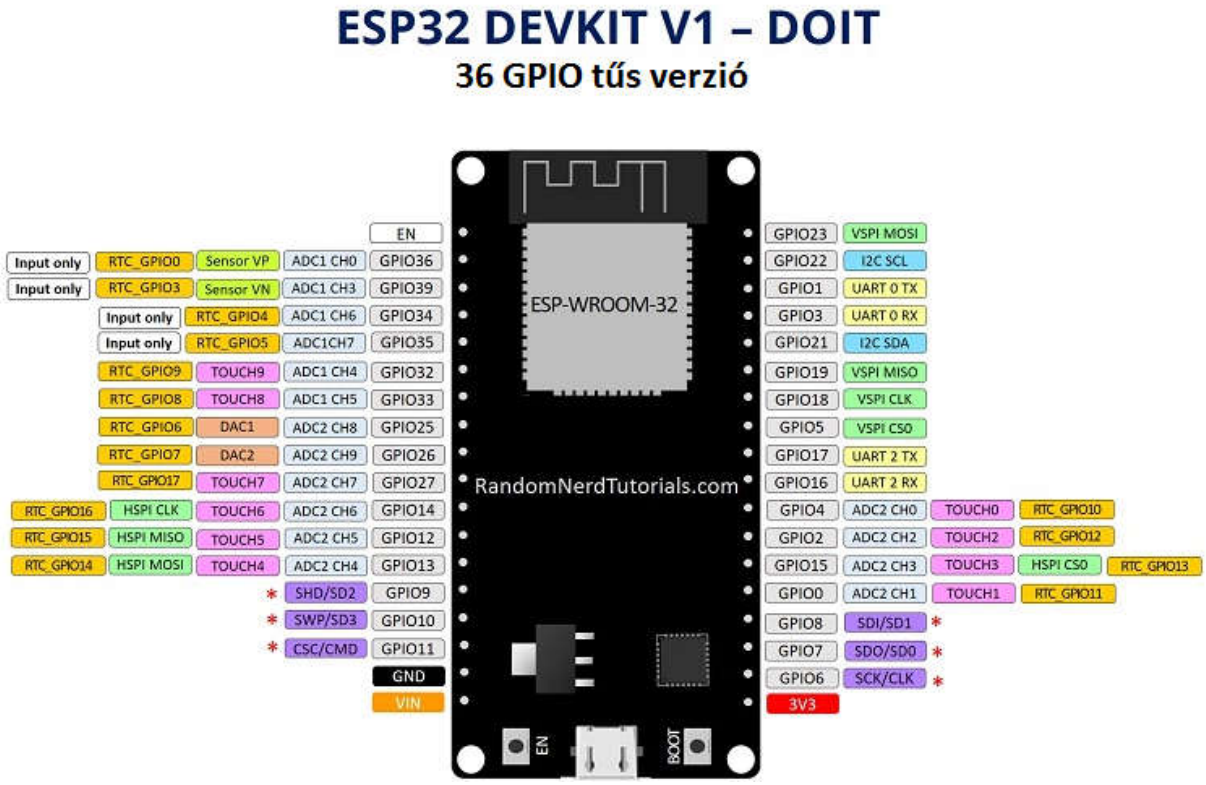
Amíg az ESP32 mélyalvó üzemmódban van, az RTC memória is bekapcsolva marad, így az ULP társprocesszorhoz programot írhatunk, amit az RTC memóriában tárolhatunk, hogy elérjük a perifériákat, a belső időzítőket és a belső érzékelőket.

Ez a működési mód akkor hasznos, ha fel kell ébresztenie a fő CPU-t egy külső esemény, időzítő vagy mindkettő hatására, miközben minimális energiafogyasztást tart fenn.

RTC_GPIO tűk

Mély alvás közben az ESP32 érintkezők egy részét az ULP társprocesszor használhatja, nevezetesen az RTC_GPIO tűket és az érintőtűket. Az ESP32 adatlap egy táblázatot tartalmaz, amely azonosítja az RTC_GPIO lábakat. A táblázatot alant találod az oldalon.

Használhatja ezt a táblázatot referenciaként, vagy tekintse meg a következő kivezetést a különböző RTC_GPIO érintkezők megkereséséhez. Az RTC_GPIO érintkezők narancssárga téglalap alakú dobozzal vannak kiemelve.



* Az SCK/CLK, SDO/SD0, SHD/SD2 és SCS/CMD érintkezők, nevezetesen, a GPIO6-GPIO11 az ESP-WROOM-32-n integrált csatlakozókhoz csatlakoznak, és más célra nem ajánlottak.

Érdemes lehet olvasni: ESP32 tűkiosztás hivatkozás: Milyen GPIO tűket érdemes használni?

Felébresztési források

Miután az ESP32-t mély alvó módba helyezted, több módon is felébresztheted:

1. Használhatod az időzítőt, és felébresztheted az ESP32-t előre meghatározott időtartamokkal;
2. Használhatod az érintőtűket;
3. A külső ébresztés két lehetőségét használhatod: használhatsz egy külső ébresztést, vagy több különböző külső ébresztési forrást;
4. Használhatod az ULP társprocesszort az ébresztéshez – ez az útmutató erre nem tér ki.

Mélyalvás vázlat írása

Ha vázlatot szeretnél írni az ESP32 mély alvó módba helyezéséhez, majd felébresztéséhez, szem előtt kell tartanod a következőket:

1. Először is be kell állítanod az ébresztési forrásokat. Ez azt jelenti, hogy be kell állítani, hogy mi fogja felébreszteni az ESP32-t. Használhatsz egy vagy több ébresztőforrást is.
2. Eldöntheted, hogy mely perifériákat kapcsolja ki vagy tartsa bekapcsolva mélyalvás közben. Alapértelmezés szerint azonban az ESP32 automatikusan kikapcsolja azokat a perifériákat, amelyekre nincs szükség az általad meghatározott ébresztőforráshoz.
3. Végül használd az `esp_deep_sleep_start()` függvényt az ESP32 mélyalvás módba helyezéséhez.

Időzített ébredés

Az ESP32 mély alvó üzemmódba léphet, majd előre meghatározott idő után felébredhet. Ez a funkció különösen akkor hasznos, ha olyan projekteket futtatsz, amelyek időbélyegzést vagy napi feladatokat igényelnek, miközben alacsony energiafogyasztást tartanak fenn.

Az ESP32 RTC vezérlő beépített időzítővel rendelkezik, amellyel előre meghatározott idő elteltével felébreszthető az ESP32-t.

Az időzített ébresztés engedélyezése

Nagyon egyszerű az ESP32 felébresztésének engedélyezése egy előre meghatározott idő után. Az Arduino IDE-ben csak meg kell adnia az alvás időtartamát mikroszekundumban a következő függvényben:

```
esp_sleep_enable_timer_wakeup(time_in_us)
```

A kód

Nézzük meg, hogyan működik ez egy példa segítségével a könyvtárból. Nyisd meg az Arduino IDE-t, és lépj a **Fájl > Példák > ESP32 DeepSleep** menüpontra, és nyisd meg a **TimerWakeUp** vázlatot.

```
/*
Egyszerű mély alvás időzített ébresztéssel
==========================================
Az ESP32 mély alvó üzemmódot kínál a hatékony
energiatakarékosság érdekében, mivel az energia
fontos tényező az IoT-alkalmazásokban. Ebben az
üzemmódban a CPU-k a RAM nagy része és az
APB_CLK órajellel rendelkező összes digitális
perifériája ki van kapcsolva. A chip egyetlen részei,
amelyek még bekapcsolva maradnak: az RTC
vezérlő, az RTC perifériák és az RTC memóriák

Ez a kód megjeleníti a legalapvetőbb mély alvást
egy időzítővel, hogy felébressze, és hogyan tárolja
az adatokat az RTC memóriában az újraindítások
használatához

Ez a kód nyilvános licenc alatt áll.

Szerző:
Pranav Cherukupalli <cherukupallip@gmail.com>
*/

#define uS_TO_S_FACTOR 1000000ULL /* Átváltási tényező mikro másodpercről másodpercre */
#define TIME_TO_SLEEP  5          /* Az ESP32 alvásának ideje (másodpercben) */
```

```
RTC_DATA_ATTR int bootCount = 0;

/*
Metódus az ok kinyomtatásához, ami
miatt az ESP32 felébredt az alvásból
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Az RTC_IO használatával külső jel által
okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Az RTC_CNTL használatával külső jel
által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Az időzítő által okozott ébredés");
break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Az érintőpad okozta ébredés"); break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Az ULP program által okozott ébredés");
break;
    default:                        Serial.printf("Az ébredést nem a mély alvás okozta:
%d\n", wakeup_reason); break;
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);  // Szánunk egy kis időt a soros monitor megnyitására

  // Növelje a rendszerindítási számot, és minden újraindításkor nyomtassa ki
  ++bootCount;
  Serial.println("A Boot száma: " + String(bootCount));

  // Kinyomtatjuk az ESP32 ébredésének okát
  print_wakeup_reason();

  /*
  Először konfiguráljuk az ébresztési forrást
  Az ESP32-t úgy állítottuk be, hogy 5 másodpercenként ébredjen
  */
  esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
  Serial.println("Beállítjuk az ESP32-t " + String(TIME_TO_SLEEP) + " másodpercenkénti éb-
redésre");

  /*
 Ezután eldöntjük, hogy mely perifériákat állítsuk le/tartsuk bekapcsolva.
 Alapértelmezés szerint az ESP32 automatikusan kikapcsolja azokat a
 perifériákat, amelyekre az ébresztőforrásnak nincs szüksége, de ha nagy
 teljesítményű felhasználó akar lenni, ez az Ön számára készült. Olvassa el
 részletesen az API dokumentumokat:
 https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-
reference/system/sleep_modes.html
 A perifériák konfigurálására vonatkozó példaként hagyta meg a megjegyzéssort.
 Az alábbi sor az összes RTC-perifériát kikapcsolja mély alvás közben.
  */
  //esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
  //Serial.println("Az összes RTC perifériát úgy konfiguráltuk, hogy alvó üzemmódban kikap-
csoljon");

  /*
 Most, hogy beállítottuk az ébredés okát, és ha szükséges, beállítjuk
 a perifériák állapotát mélyalvásban, megkezdhetjük a mély alvást.
 Abban az esetben, ha nem biztosítottak ébresztési forrást, de a
 mélyalvás elindult, örökké aludni fog, hacsak nem történik hardver-reset.
```

```
  */
  Serial.println("Most megyek aludni.");
  Serial.flush();
  esp_deep_sleep_start();
  Serial.println("Ez soha nem nyomtatódik ki");
}

void loop() {
  // Ez nem kerül meghívásra
}
```

Az első megjegyzés leírja, hogy mi kapcsol ki mélyalvás közben, időzített ébresztésnél.

```
Az ESP32 mély alvó üzemmódot kínál a hatékony
energiatakarékosság érdekében, mivel az energia
fontos tényező az IoT-alkalmazásokban. Ebben az
üzemmódban a CPU-k a RAM nagy része és az
APB_CLK órajellel rendelkező összes digitális
perifériája ki van kapcsolva. A chip egyetlen részei,
amelyek még bekapcsolva maradnak: az RTC
vezérlő, az RTC perifériák és az RTC memóriák
```

Ha időzített ébresztést használsz, az RTC-vezérlő, az RTC-perifériák és az RTC-memóriák kapcsolódnak be.

Az alvási idő meghatározása

Ez az első két kódsor határozza meg azt az időtartamot, ameddig az ESP32 alvó állapotban lesz.

```
#define uS_TO_S_FACTOR 1000000ULL /* Átváltási tényező mikro másodpercről másodpercre */
#define TIME_TO_SLEEP  5          /* Az ESP32 alvásának ideje (másodpercben) */
```

Ez a példa mikroszekundumról másodpercre váltja át a váltási tényező, így a `TIME_TO_SLEEP` változóban másodpercben állíthatod be az alvásidőt. Ebben az esetben a példa az ESP32-t mély alvó módba helyezi 5 másodpercre.

Adatok mentése az RTC-memóriákba

Az ESP32 segítségével adatokat menthetsz az RTC-memóriákba. Az ESP32 RTC részen 8 kB SRAM található, az úgynevezett RTC gyors memóriát. Az itt elmentett adatok mélyalvás közben nem törlődnek. Ez azonban törlődik, amikor megnyomod a reset gombot (az EN feliratú gomb az ESP32 kártyán).

Az adatok RTC memóriába mentéséhez csak hozzá kell adnod az `RTC_DATA_ATTR`-t egy változódefiníció előtt. A példa a `bootCount` változót az RTC memóriába menti. Ez a változó azt fogja számolni, hogy az ESP32 hányszor ébredt fel mélyalvásból.

```
RTC_DATA_ATTR int bootCount = 0;
```

Ébredési ok

Ezután a kód meghatározza a `print_wakeup_reason()` függvényt, amely kiírja a forrást, amely a mélyalvásból való felébredést okozta.

```
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();
```

```
  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Az RTC_IO használatával külső
jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Az RTC_CNTL használatával kül-
ső jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Az időzítő által okozott ébre-
dés"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Az érintőpad okozta ébredés");
break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Az ULP program által okozott
ébredés"); break;
    default:                        Serial.printf("Az ébredést nem a mély alvás
okozta: %d\n", wakeup_reason); break;
  }
}
```

A setup()

A setup() az a hely, ahol meg kell adni a kódot. Az `esp_deep_sleep_start()` függvény meghívása előtt meg kell írni az összes utasítást.

Ez a példa a soros kommunikáció inicializálásával kezdődik 115 200 adatátviteli sebességgel.

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

Ezután a `bootCount` változó minden újraindításkor eggyel nő, és ez a szám megjelenik a soros monitoron.

```
++bootCount;
Serial.println("A Boot száma: " + String(bootCount));
```

Ezután a kód meghívja a `print_wakeup_reason()` függvényt, de bármely olyan függvényt meghívhat, amellyel a kívánt feladatot végrehajtani szeretnéd. Előfordulhat például, hogy naponta egyszer felébreszted az ESP32-t, hogy kiolvass egy értéket egy érzékelőből.

Ezután a kód meghatározza az ébresztési forrást a következő függvény segítségével:

```
esp_sleep_enable_timer_wakeup(time_in_us)
```

Ez a függvény argumentumként fogadja az elalvás idejét mikroszekundumban, amint azt korábban láttuk. A mi esetünkben a következőkkel rendelkezünk:

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
```

Ezután az összes feladat elvégzése után az ESP32 alvó üzemmódba lép a következő funkció meghívásával:

```
esp_deep_sleep_start()
```

Amint meghívja az `esp_deep_sleep_start()` függvényt, az ESP32 alvó állapotba kerül, és nem futtatja le a függvény után írt kódot. Amikor felébred a mély alvásból, a kódot az elejétől kezd futtatni.

loop()

A `loop()` szakasz üres, mert az ESP32 aludni fog, mielőtt elérné a kód ezen részét. Tehát az `esp_deep_sleep_start()` függvény meghívása előtt minden feladatot be kell írnia a `setup()`-ba.

Az időzített ébredés tesztelése

Töltsd fel a példavázlatot az ESP32-re. Győződj meg arról, hogy a megfelelő kártya és COM port van kiválasztva. Nyisd meg a soros monitort 115 200 adatátviteli sebességgel.

5 másodpercenként az ESP32 felébred, kinyomtat egy üzenetet a soros monitorra, és ismét mély alvásba megy.

Minden alkalommal, amikor az ESP32 felébred, a `bootCount` változó növekszik. Az alábbi ábrán látható módon kinyomtatja az ébredés okát is.

Figyeld meg azonban, hogy ha megnyomod az EN gombot az ESP32 kártyán, az újra 1-re állítja a bootolási számlálót.

Módosíthatod a megadott példát, és üzenet kinyomtatása helyett bármilyen más feladat elvégzésére késztetheted az ESP-t. Az időzítő ébresztése hasznos az ESP32-vel végzett időszakos feladatok, például a napi feladatok elvégzéséhez anélkül, hogy sok energiát fogyasztana.

Ébresztés érintéssel

Az ESP32-t mély álomból az érintőtűkkel is felébresztheted. Ez a rész bemutatja, hogyan kell ezt megoldani az Arduino IDE használatával.

Az érintéses ébresztés engedélyezése

Az ESP32 felébresztésének engedélyezése egy érintőtűvel egyszerű. Az Arduino IDE-ben a következő függvényt kell használnod – add át argumentumként az érintőtűt és az érintési küszöböt:

```
touchSleepWakeUpEnable(TOUCH_PIN, THRESHOLD);
```

A kód

Nézzük meg, hogyan működik ez egy példa segítségével a könyvtárból. Nyisd meg az Arduino IDE-t, és lépj a **Fájl > Példák > ESP32 DeepSleep** menüpontra, és nyisd meg a **TouchWakeUp** vázlatot.

```
/*
Mély alvás érintéssel ébresztéssel
==================================
Ez a kód megmutatja, hogyan használhatod a mély alvást
érintéssel történő ébresztéshez, és hogyan tárolhatsz
adatokat az RTC-memóriában az bootolások elől mentve.

Az ESP32 több érintőpadot is engedélyezhet ébresztési
forrásként
Az ESP32-S2 és ESP32-S3 csak 1 érintőpadot támogat
ébresztési forrásként

Ez a kód nyilvános licenc alatt áll.
```

```
Szerző:
Pranav Cherukupalli <cherukupallip@gmail.com>
*/

#if CONFIG_IDF_TARGET_ESP32
#define THRESHOLD 40   /* Minél nagyobb az érték, annál nagyobb az érzékenység */
#else                  /* ESP32-S2 és ESP32-S3 + alapértelmezés más chipekhez (be-
állítandó) */
#define THRESHOLD 5000 /* Minél alacsonyabb az érték, annál nagyobb az érzékenység
*/
#endif

RTC_DATA_ATTR int bootCount = 0;
touch_pad_t touchPin;
/*
Metódus az ok kinyomtatásához, ami
miatt az ESP32 felébredt az alvásból
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Az RTC_IO használatával külső
jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Az RTC_CNTL használatával kül-
ső jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Az időzítő által okozott ébre-
dés"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Az érintőpad okozta ébredés");
break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Az ULP program által okozott
ébredés"); break;
    default:                        Serial.printf("Az ébredést nem a mély alvás
okozta: %d\n", wakeup_reason); break;
  }
}

/*
Metódus annak az érintőpadnak a kinyomtatására, amellyel
az ESP32-t felébresztették alvó állapotából
*/
void print_wakeup_touchpad() {
  touchPin = esp_sleep_get_touchpad_wakeup_status();

#if CONFIG_IDF_TARGET_ESP32
  switch (touchPin) {
    case 0:  Serial.println("Érintés észlelve a GPIO 4-en"); break;
    case 1:  Serial.println("Érintés észlelve a GPIO 0-n"); break;
    case 2:  Serial.println("Érintés észlelve a GPIO 2-n"); break;
    case 3:  Serial.println("Érintés észlelve a GPIO 15-ön"); break;
    case 4:  Serial.println("Érintés észlelve a GPIO 13-on"); break;
    case 5:  Serial.println("Érintés észlelve a GPIO 12-n"); break;
    case 6:  Serial.println("Érintés észlelve a GPIO 14-en"); break;
    case 7:  Serial.println("Érintés észlelve a GPIO 27-en"); break;
    case 8:  Serial.println("Érintés észlelve a GPIO 33-on"); break;
    case 9:  Serial.println("Érintés észlelve a GPIO 32-n"); break;
```

```
    default: Serial.println("Ébredés nem érinőpadtól"); break;
  }
#else
  if (touchPin < TOUCH_PAD_MAX) {
    Serial.printf("Érintés észlelve a GPIO %d\n", touchPin);
  } else {
    Serial.println("Ébredés nem érinőpadtól");
  }
#endif
}

void setup() {
  Serial.begin(115200);
  delay(1000);  // Szánunk egy kis időt a soros monitor megnyitására

  // Növeljük a bootolási számot, és minden újraindításkor kinyomtjuk
  ++bootCount;
  Serial.println("Boot száma: " + String(bootCount));

  // Kinyomtatjuk az ESP32 ébresztés okát és az érintőpad számát is
  print_wakeup_reason();
  print_wakeup_touchpad();

#if CONFIG_IDF_TARGET_ESP32
  // Az ébresztés beállítása az érintőpad 3 + 7-en (GPIO15 + GPIO 27)
  touchSleepWakeUpEnable(T3, THRESHOLD);
  touchSleepWakeUpEnable(T7, THRESHOLD);

#else  //ESP32-S2 + ESP32-S3
  // Az ébresztés beállítása az érintőpad 3-on (GPIO3)
  touchSleepWakeUpEnable(T3, THRESHOLD);

#endif

  // Most menj aludni
  Serial.println("Most megyek aludni");
  esp_deep_sleep_start();
  Serial.println("Ez soha nem nyomtatódik ki");
}

void loop() {
  // Ez nem kerül meghívásra
}
```

A küszöb beállítása

Az első dolog, amit meg kell tennünk, hogy az érintési tűk küszöbértékét be kell állítani.

```
#define THRESHOLD 40   /* Minél nagyobb az érték, annál nagyobb az érzékenység */
```

Az érintőtű által leolvasott értékek csökkennek, amikor megérinted. A küszöbérték azt jelenti, hogy az ESP32 felébred, ha az érintőtűn leolvasott érték 40 alá esik. Ezt az értéket a kívánt érzékenységtől függően módosíthatod.

Fontos: ha azonban ESP32-S2 vagy ESP32-S3 modellt használsz, a dolgok egy kicsit másképp működnek. Ezért van egy másik szakasz a kódban, amely más küszöbértéket határoz meg ezekhez a táblákhoz. Ebben az esetben minél kisebb az érték, annál nagyobb az érzékenység.

```
#if CONFIG_IDF_TARGET_ESP32
#define THRESHOLD 40   /* Minél nagyobb az érték, annál nagyobb az érzékenység */
#else                  /* ESP32-S2 és ESP32-S3 + alapértelmezés más chipekhez (be-
állítandó) */
#define THRESHOLD 5000 /* Minél alacsonyabb az érték, annál nagyobb az érzékenység
*/
#endif
```

Érintőtűk beállítása ébresztőforrásként

Az érintőtű ébresztési forrásként való beállításához használhatod a `touchSleepWakeUpEnable()` függvényt, amely argumentumként fogadja az érintőtűt és a táblát felébresztő küszöbértéket.

Ebben a példában a GPIO 15-öt (T3) és a GPIO 27-et (T7) állítjuk be ébresztési forrásként azonos küszöbértékkel.

```
#if CONFIG_IDF_TARGET_ESP32
  // Az ébresztés beállítása az érintőpad 3 + 7-en (GPIO15 + GPIO 27)
  touchSleepWakeUpEnable(T3, THRESHOLD);
  touchSleepWakeUpEnable(T7, THRESHOLD);
```

Ha ESP32-S2 vagy S3 modellt használsz, a példa csak a GPIO 3 (T3) ébresztését határozza meg. Vedd figyelembe, hogy az érintőtűk számozása a használt kártyamodelltől függően eltérő lehet.

```
#else // ESP32-S2 + ESP32-S3
  // Az ébresztés beállítása az érintőpad 3-on (GPIO3)
  touchSleepWakeUpEnable(T3, THRESHOLD);
```

Végül meghívjuk az `esp_deep_sleep_start()` függvényt, hogy az ESP32 mélyalvó módba kerüljön.

```
esp_deep_sleep_start();
```

Ezek voltak az érintőtűk ébresztési forrásként való beállítására vonatkozó általános utasítások. Most pedig vessünk egy pillantást a kód többi szakaszára.

setup()

Amikor az ESP32 felébred alvó állapotból, a kódot elejétől futtatja, amíg meg nem találja az `esp_deep_sleep_start()` függvényt.

Ebben a konkrét példában van egy `bootCount` nevű vezérlőváltozónk, amelyet az egyes alvási ciklusok között növelni fogunk, így van elképzelésünk arról, hogy az ESP32 hányszor ébredt fel.

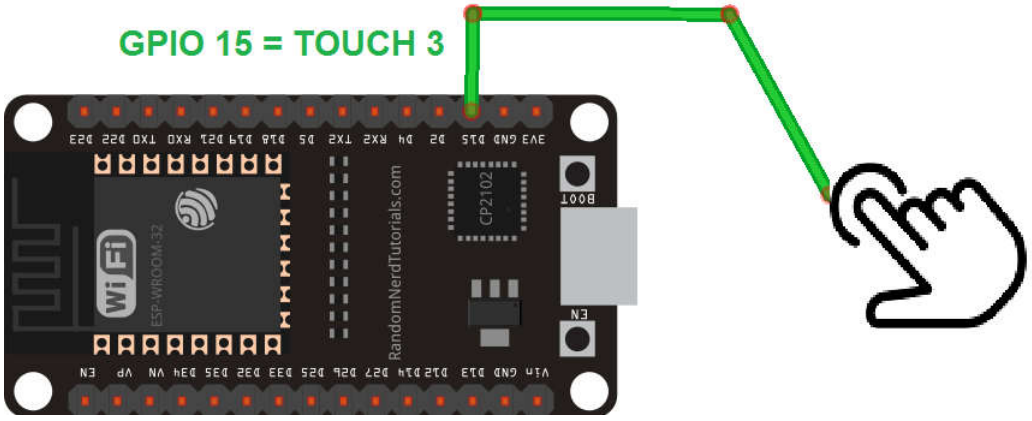
```
// Növeljük a bootolási számot, és minden újraindításkor kinyomtjuk
++bootCount;
Serial.println("Boot száma: " + String(bootCount));
```

Meghívjuk a kódban korábban definiált `print_wakeup_reason()` és `print_wakeup_touchpad()` függvényeket is, hogy kinyomtassuk az ébresztés okát és az ébresztést okozó PIN számát.

```
// Kinyomtatjuk az ESP32 ébresztés okát és az érintőpad számát is
  print_wakeup_reason();
  print_wakeup_touchpad();
```

Az áramkör bekötése

A példa teszteléséhez csatlakoztass egy kábelt a GPIO 15-höz, az alábbi kapcsolási rajz szerint.



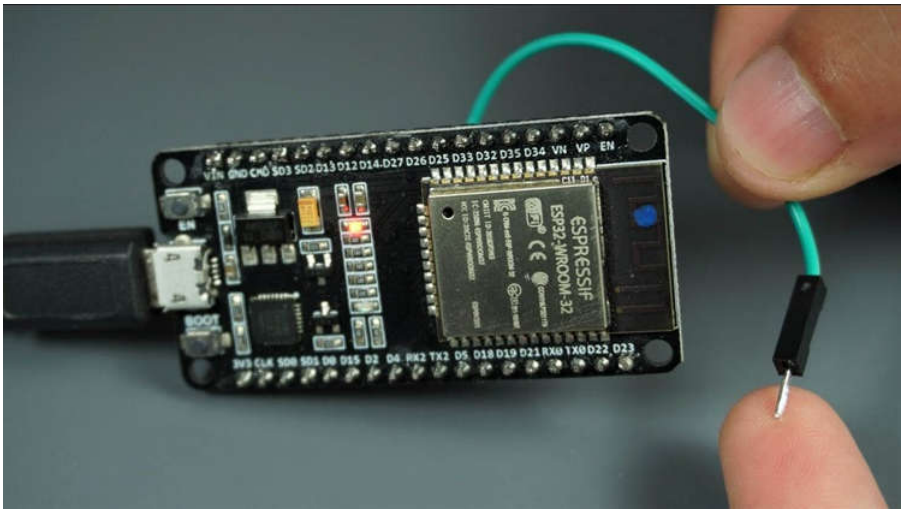
Ha ESP32-S2 vagy ESP32-S3 modellt használsz, ellenőrizd az érintőtűk helyét.

A példa tesztelése

Töltsd fel a kódot az ESP32-re, és nyisd meg a soros monitort 115200 adatátviteli sebességgel.

Az ESP32 mély alvó módba lép.

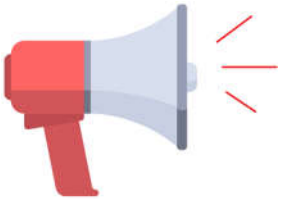
A 3. érintőtűhöz csatlakoztatott vezeték megérintésével felébresztheted.



Amikor megérinted a tűt, az ESP32 megjeleníti a soros monitoron: a Bootolási számot, az ébresztés okát, és azt, hogy melyik érintésérzékeny GPIO okozta az ébresztést.

Külső ébresztés

Az ESP32-t az időzítőn és az érintőtűkön kívül úgy is felébreszthetjük a mélyalvásból, hogy egy gomb-nyomáshoz hasonlóan a tűn lévő jel értékét váltogatjuk. Ezt külső ébresztésnek hívják. Két lehetősége van a külső felébresztésre: ext0 és ext1.



Külső ébresztés (ext0)

Ez az ébresztőforrás lehetővé teszi, hogy tű segítségével felébreszd az ESP32-t.

Az ext0 ébresztési forrás opció RTC GPIO-kat használ az ébresztéshez. Tehát az RTC perifériák bekapcsolva maradnak mély alvás közben is, ha ezt az ébresztési forrást kérik.

Ennek az ébresztőforrásnak a használatához használd a következő függvényt:

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_X, level)
```

Ez a függvény első argumentumként a használni kívánt PIN-kódot fogadja, ebben a GPIO_NUM_X formátumban, amelyben **X** az adott láb GPIO-számát jelöli.

A második argumentum, a level, 1 vagy 0 lehet. Ez a GPIO állapotát jelöli, amely kiváltja az ébredést.

Megjegyzés: ezzel az ébresztőforrással csak olyan tűket használhatsz, amelyek RTC GPIO-k.

Külső ébresztés (ext1)

Ezt az ébresztési forrást az RTC vezérlő vezérli. Az ext0-val ellentétben ez az ébresztési forrás akkor is támogatja az ébresztést, ha az RTC periféria ki van kapcsolva. Tehát az RTC perifériák és az RTC memóriák kikapcsolhatók ebben az üzemmódban.

Ennek az ébresztőforrásnak a használatához használd a következő függvényt:

```
esp_sleep_enable_ext1_wakeup_io(bitmask, mode);
```

Ez a függvény két argumentumot fogad:

- Az ébresztést okozó GPIO-számok bitmaszkja;
- Mód: az ESP32 felébresztésének logikája. Ez lehet:
    - ESP_EXT1_WAKEUP_ALL_LOW: felébred, amikor az összes GPIO lemerül;
    - ESP_EXT1_WAKEUP_ANY_HIGH: ébredjen fel, ha valamelyik GPIO magasra emelkedik.

Ha ESP32-S2-t, ESP32-S3-at, ESP32-C6-ot vagy ESP32-H2-t használsz, a következő módok állnak rendelkezésre:

- ESP_EXT1_WAKEUP_ANY_LOW: ébredjen fel, ha a kiválasztott GPIO-k bármelyike alacsony
- ESP_EXT1_WAKEUP_ANY_HIGH: ébredjen fel, ha a kiválasztott GPIO-k bármelyike magas

Megjegyzés: csak olyan tűket használhatsz, amelyek RTC GPIO-k.

Az ext1 mélyalvás ébresztési forrásával kapcsolatos összes részletért tekintsd meg az Espressif mélyalvás dokumentációját.

A kód

Vizsgáljuk meg az ESP32 könyvtárhoz tartozó példát. Menj a **Fájl > Példák > ESP32 DeepSleep > ExternalWakeUp** fájlhoz:

```
/*
  Mély alvás külső ébresztéssel
  =============================
  Ez a kód megmutatja, hogyan kell a mély alvást
  külső triggerrel használni ébresztési forrásként,
  és hogyan kell adatokat tárolni az RTC memóriában
  az újraindítások feletti használatához.

  Ez a kód nyilvános licenc alatt áll.

  Hardver csatlakozások
  =====================
  Nyomógomb a GPIO 33-hoz 10 kOhm-os ellenállással lehúzva

  MEGJEGYZÉS:
  ===========
  Csak az RTC IO használható külső ébresztőforrás
  forrásaként. Ezek a tűk: 0,2,4,12-15,25-27,32-39.

  Szerző:
  Pranav Cherukupalli <cherukupallip@gmail.com>
*/
#include "driver/rtc_io.h"

#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER hex-ben
#define USE_EXT0_WAKEUP          1               // 1 = EXT0 ébresztés, 0 = EXT1
ébresztés
#define WAKEUP_GPIO              GPIO_NUM_33     // Csak az RTC IO engedélyezett –
ESP32 tűs példa
RTC_DATA_ATTR int bootCount = 0;

/*
Metódus az ok kinyomtatásához, ami
miatt az ESP32 felébredt az alvásból
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Az RTC_IO használatával külső
jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Az RTC_CNTL használatával kül-
ső jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Az időzítő által okozott ébre-
dés"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Az érintőpad okozta ébredés");
break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Az ULP program által okozott
```

```
ébredés"); break;
    default:                        Serial.printf("Az ébredést nem a mély alvás
okozta: %d\n", wakeup_reason); break;
  }
}

void setup() {
  Serial.begin(115200);
  delay(1000);  // Szánunk egy kis időt a soros monitor megnyitására

  // Növeljük a bootolási számot, és minden újraindításkor kinyomtatjuk
  ++bootCount;
  Serial.println("Boot szám: " + String(bootCount));

  // Kinyomtatjuk az ESP32 ébredésének okát
  print_wakeup_reason();

  /*
    Először konfiguráljuk az ébresztési forrást.
    Beállítjuk az ESP32-nket, hogy külső triggerre ébredjen.
    Az ESP32-höz két típusa létezik, az ext0 és az ext1.
    Az ext0 az RTC_IO-t használja az ébresztéshez, így az
    RTC-perifériáknak bekapcsolva kell lenniük, míg az ext1-nek
    az RTC-vezérlőt használja, így nincs szükség perifériák bekapcsolására.
    Vedd figyelembe, hogy a belső felhúzások/lehúzások használatához
    az RTC perifériákat is be kell kapcsolni.
  */
#if USE_EXT0_WAKEUP
  esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1);  //1 = High, 0 = Low
  // Konfigurálja a felhúzást/lehúzást az RTCIO-n keresztül, hogy az ébresztőtűket
inaktív szintre kapcsolja mélyalvás közben.
  // Az EXT0 ugyanabban a teljesítménytartományban (RTC_PERIPH) található, mint az
RTC IO fel/lehúzása.
  // Az EXT1-gyel ellentétben nem szükséges kifejezetten megtartani ezt a telje-
sítménytartományt.
  rtc_gpio_pullup_dis(WAKEUP_GPIO);
  rtc_gpio_pulldown_en(WAKEUP_GPIO);

#else  // EXT1 WAKEUP
  // Ha az ext1-et használnád, úgy használnád
  esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO),
ESP_EXT1_WAKEUP_ANY_HIGH);
  /*
    Ha nincs külső fel/lehúzás, az ébresztőtűket kösse inaktív
    szintre belső felhúzással/lehúzással az RTC IO-n keresztül
    mélyalvás közben. Az RTC IO azonban az RTC_PERIPH
    táptartományra támaszkodik.
    Ennek az energiatartománynak a bekapcsolva tartása növeli
    az energiafogyasztást. Ha azonban kikapcsoljuk az RTC_PERIPH
    tartományt, vagy ha bizonyos lapkákból hiányzik az RTC_PERIPH
    tartomány, akkor a HOLD funkciót használjuk a tűk fel- és
    lehúzásának fenntartásához alvás közben.
  */
  rtc_gpio_pulldown_en(WAKEUP_GPIO);  // A GPIO33 a GND-hez kapcsolódik, hogy
HIGH-ra ébredjen
  rtc_gpio_pullup_dis(WAKEUP_GPIO);   // Tiltsa le a PULL_UP funkciót, hogy HIGH-
ra ébredjen fel
#endif
  // Most menj aludni
```

```
  Serial.println("Most megyek aludni");
  esp_deep_sleep_start();
  Serial.println("Ez soha nem nyomtatódik ki");
}

void loop() {
  // Ez nem kerül meghívásra
}
```

Ez a példa felébreszti az ESP32-t, amikor a GPIO 33-at magasra kapcsolja. A kódpélda bemutatja, hogyan kell használni mindkét metódust: az ext0-t és az ext1-et. Ha úgy töltöd fel a kódot, ahogy van, az az ext0-t fogja használni. Az ext1 használatához tartozó függvény megjegyzést kap. Megmutatjuk, hogyan működik mindkét módszer, és hogyan kell használni őket.

Vessünk egy pillantást a kódra.

Adatok mentése az RTC-memóriákban

Az ESP32 segítségével adatokat menthetsz az RTC-memóriákba. Az ESP32 RTC részen 8 kB SRAM található, az úgynevezett RTC gyors memóriát. Az itt elmentett adatok mélyalvás közben nem törlődnek. Ez azonban törlődik, amikor megnyomod a reset gombot (az EN feliratú gomb az ESP32 kártyán).

Az adatok RTC memóriába mentéséhez csak hozzá kell adnod az `RTC_DATA_ATTR`-t egy változódefiníció előtt. A példa a `bootCount` változót az RTC memóriába menti. Ez a változó azt számolja, hogy az ESP32 hányszor ébredt fel mélyalvásból.

```
RTC_DATA_ATTR int bootCount = 0;
```

Az ébredés oka

Ezután a kód definiálja a `print_wakeup_reason()` függvényt, amely kiírja az okot, ami miatt az ESP32 felébredt az alvó állapotból.

```
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;
  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:     Serial.println("Az RTC_IO használatával külső
jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_EXT1:     Serial.println("Az RTC_CNTL használatával kül-
ső jel által okozott ébredés"); break;
    case ESP_SLEEP_WAKEUP_TIMER:    Serial.println("Az időzítő által okozott ébre-
dés"); break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD: Serial.println("Az érintőpad okozta ébredés");
break;
    case ESP_SLEEP_WAKEUP_ULP:      Serial.println("Az ULP program által okozott
ébredés"); break;
    default:                        Serial.printf("Az ébredést nem a mélyalvás
okozta: %d\n", wakeup_reason); break;
  }
}
```

setup()

A `setup()`-ban először inicializáljuk a soros kommunikációt:

```
Serial.begin(115200);
delay(1000);  // Szánunk egy kis időt a soros monitor megnyitására
```

Ezután növeljük eggyel a `bootCount` változót, és kinyomtatjuk a változót a soros monitoron.

```
++bootCount;
Serial.println("Boot szám: " + String(bootCount));
```

Ezután kinyomtatjuk az ébredés okát a korábban definiált `print_wakeup_reason()` függvénnyel.

```
// Kinyomtatjuk az ESP32 ébredésének okát
print_wakeup_reason();
```

Ezt követően engedélyezni kell az ébresztési forrásokat. Minden egyes ébresztési forrást, az ext0-t és az ext1-et, külön teszteljük.

ext0

Ebben a példában az ESP32 felébred, amikor a GPIO 33 magasra vált:

```
esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1);  // 1 = High, 0 = Low
```

A GPIO 33 helyett bármilyen más RTC GPIO tűt használhatsz. Csak add meg a WAKEUP_GPIO változóban a kód elején.

```
#define WAKEUP_GPIO GPIO_NUM_33 // Csak az RTC IO engedélyezett
```

Belső fel- és lehúzó ellenállások

A következő két sort is hívjuk:

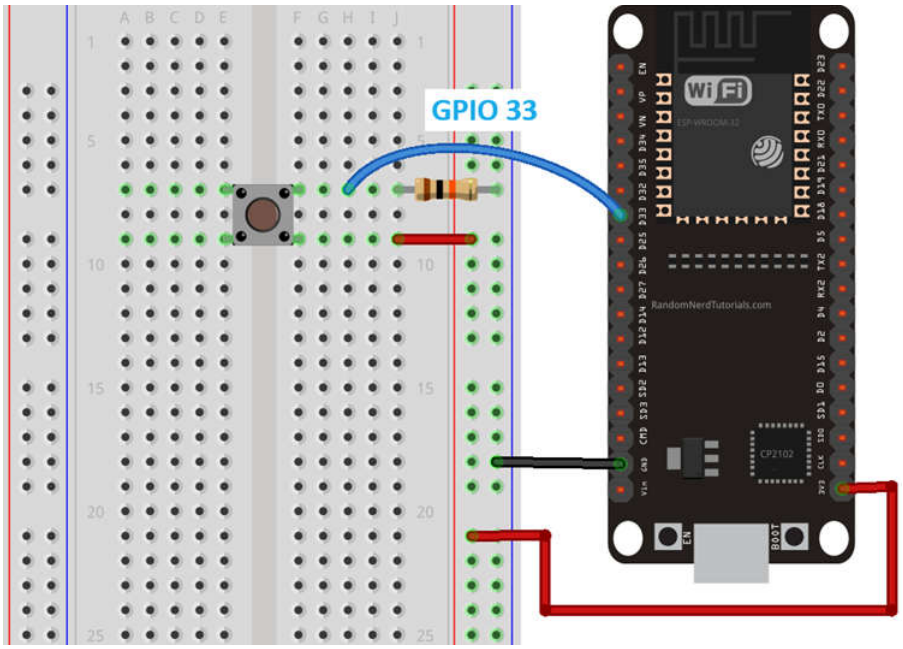
```
rtc_gpio_pullup_dis(WAKEUP_GPIO);
rtc_gpio_pulldown_en(WAKEUP_GPIO);
```

Az `rtc_gpio_pullup_dis()` függvény letilt minden belső felhúzó ellenállást az ébresztő GPIO-n – ez biztosítja, hogy a tű ne kerüljön véletlenül magasra. Ez azért fontos, mert egy felhúzó ellenállás magasan tartaná a tűt, ami nem kívánt ébresztést okozhat.

Az `rtc_gpio_pulldown_en()` függvény engedélyezi a belső lehúzó ellenállást az ébresztő GPIO-n – ez biztosítja, hogy a tű alacsonyan maradjon, amíg érvényes ébresztőjelet (HIGH) nem kap. A tű lehúzó ellenállással történő konfigurálásával garantáljuk, hogy mélyalvás közben is stabilan alacsony állapotban maradjon. Ez a stabilitás biztosítja, hogy az ESP32 csak akkor ébredjen fel, ha a megadott GPIO érintkező külső magas jelet kap, amely megfelel az `esp_sleep_enable_ext0_wakeup(WAKEUP_GPIO, 1)` által beállított ébresztési feltételeknek.

Az áramkör bekötése

A példa teszteléséhez csatlakoztasson egy nyomógombot az ESP32-hez a következő vázlatos diagramot követve. A gomb a GPIO 33-hoz csatlakozik egy 10kOhm-os lehúzható ellenállás segítségével.

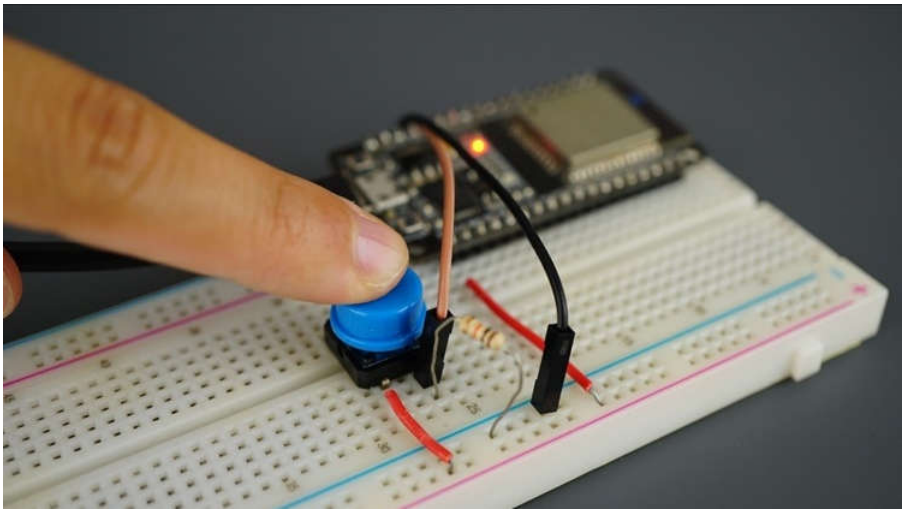


Megjegyzés: csak RTC GPIO-k használhatók ébresztőforrásként. A GPIO 33 helyett bármilyen RTC GPIO érintkezőt használhat a gomb csatlakoztatásához.

A példa tesztelése

Teszteljük ezt a példát. Töltsd fel a példakódot az ESP32-re. Győződj meg arról, hogy a megfelelő kártya és COM port van kiválasztva. Nyisd meg a soros monitort 115200 adatátviteli sebességgel.

Nyomd meg a nyomógombot az ESP32 felébresztéséhez.



Próbáld ki ezt többször, és nézd meg, hogy a bootolások száma minden gombnyomással növekszik.

Ez a módszer hasznos az ESP32-nek egy nyomógombbal történő felébresztéséhez, például egy adott feladat végrehajtásához. Ezzel a módszerrel azonban csak egy GPIO-t használhatsz ébresztőforrásként.

Mi a teendő, ha különböző gombokat szeretnél, amelyek mindegyike felébreszti az ESP-t, de különböző feladatokat végez el?

ext1

Az ext1 ébresztési forrás lehetővé teszi az ESP32 felébresztését különböző GPIO-k használatával, és különböző feladatok végrehajtását az ébresztést okozó GPIO-tól függően.

Az `esp_sleep_enable_ext0_wakeup()` függvény használata helyett az `esp_sleep_enable_ext1_wakeup_io()` függvényt használd.

```
esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO),
                                ESP_EXT1_WAKEUP_ANY_HIGH);
```

Ebben a konkrét példában a kód adott részének futtatásához állítsd be a `USE_EXT0_WAKEUP` változót 0-ra a kód elején, így:

```
#define USE_EXT0_WAKEUP  0  // 1 = EXT0 ébresztés, 0 = EXT1 ébresztés
```

Az esp_sleep_enable_ext1_wakeup_io() függvény

Az `esp_sleep_enable_ext1_wakeup_io()` függvény első argumentuma az ébresztési forrásként használt GPIO-k bitmaszkja, a második argumentum pedig az ESP32 felébresztésének logikáját határozza meg.

GPIO-k bitmaszkja

A GPIO bitmaszk meghatározásához használhatjuk a kód elején definiált `BUTTON_PIN_BITMASK()` makrót.

```
#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER hex-ben
```

A `BUTTON_PIN_BITMASK(GPIO)` egy makró, amely bitmaszkot hoz létre egy adott GPIO számára. Ez nem egy függvény, de némileg hasonlóan viselkedik, ha elfogad egy argumentumot és visszaad egy értéket.

Tehát, ha egy adott GPIO-hoz szeretnéd visszaadni a bitmaszkot, akkor csak így kell hívnod:

```
BUTTON_PIN_BITMASK(WAKEUP_GPIO)
```

Tehát az `esp_sleep_enable_ext1_wakeup_io()` így fog kinézni:

```
esp_sleep_enable_ext1_wakeup_io(BUTTON_PIN_BITMASK(WAKEUP_GPIO),
ESP_EXT1_WAKEUP_ANY_HIGH);
```

Bitmaszk több GPIO-hoz

Ha több GPIO-hoz szeretnél bitmaszkot létrehozni a `BUTTON_PIN_BITMASK(GPIO)` makró használatával, a bitenkénti VAGY (|) használatával kombinálhatod az egyes GPIO-tűk egyedi bitmaszkjait. A következőképpen teheted meg:

Tegyük fel, hogy szeretnél létrehozni egy bitmaszkot a GPIO 2. és 15. lábhoz. Ezt úgy teheted meg, hogy az egyes érintkezőkhöz tartozó egyedi bitmaszkokat a következőképpen kombinálod:

```
uint64_t bitmask = BUTTON_PIN_BITMASK(GPIO_NUM_2) | BUTTON_PIN_BITMASK(GPIO_NUM_15);
```

A következő példában több GPIO-t használunk ébresztőforrásként.

Ébresztési mód

Az `esp_sleep_enable_ext1_wakeup_io()` függvény második argumentuma az ébresztési mód. Amint azt korábban láttuk, ezek a lehetőségek:

- `ESP_EXT1_WAKEUP_ALL_LOW`: felébred, amikor az összes GPIO low-ba megy;
- `ESP_EXT1_WAKEUP_ANY_HIGH`: felébred, ha valamelyik GPIO high-ba megy.

Ha ESP32-S2-t, ESP32-S3-at, ESP32-C6-ot vagy ESP32-H2-t használsz, a következő módok állnak rendelkezésre:

- `ESP_EXT1_WAKEUP_ANY_LOW`: felébred, ha a kiválasztott GPIO-k bármelyike low-ba megy;
- `ESP_EXT1_WAKEUP_ANY_HIGH`: felébred, ha a kiválasztott GPIO-k bármelyike high-ba megy.

Konkrét példánkban az `ESP_EXT1_WAKEUP_ANY_HIGH` értéket használjuk. Tehát az ESP32 felébred, ha a bitmaszkból származó GPIO-k bármelyike high-ba megy.

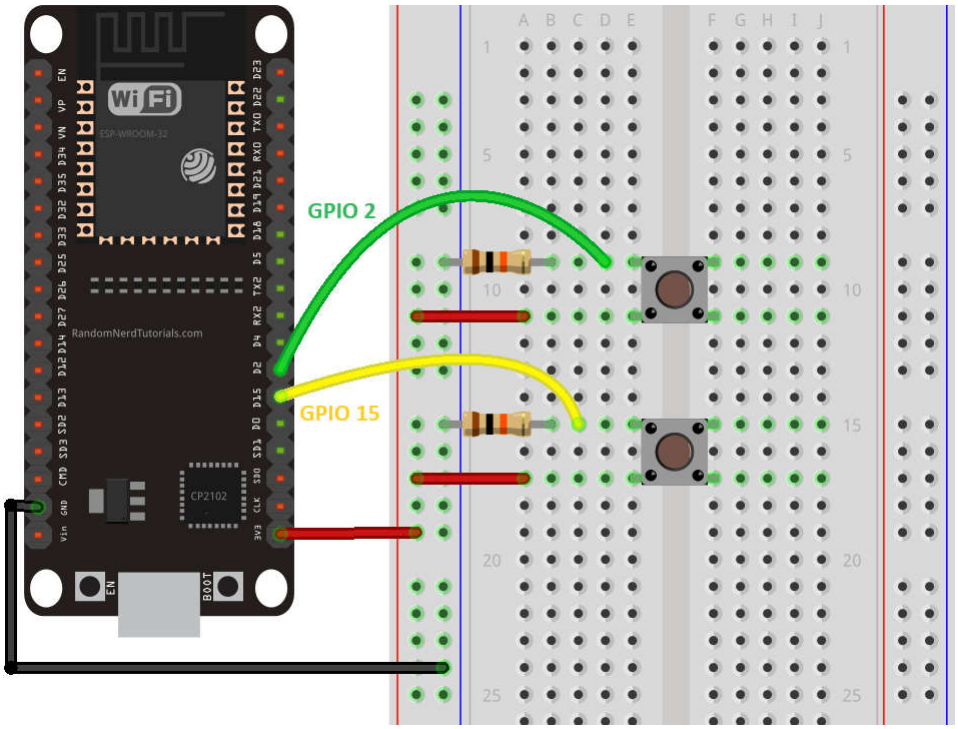
Az `ext0`-hoz hasonlóan letiltjuk a felhúzó belső ellenállásokat, és engedélyezzük a lehúzó ellenállásokat, hogy biztosítsuk az ébresztő GPIO olvasásának stabilitását.

Külső ébresztés – Több GPIO

Most már képesnek kell lenned arra, hogy különböző gombokkal felébreszd az ESP32-t, és azonosítsd, melyik gomb okozta az ébresztést. Ebben a példában a GPIO 2-t és a GPIO 15-öt használjuk ébresztési forrásként.

A vázlat

Két gombot ssatlakoztaunk az ESP32-hez. Ebben a példában GPIO 2-t és GPIO 15-öt használunk, de a gombokat bármely RTC GPIO-hoz csatlakoztathatod.



Több GPIO kódja – Külső ébresztés

Néhány módosítást kell végrehajtanod a korábban használt példakódon:

- létrehozunk egy bitmaszkot a GPIO 15 és GPIO 2 használatához. Korábban már bemutattuk, hogyan kell ezt megtenni;
- ext1 engedélyezése ébresztési forrásként;
- létrehozunk egy függvényt, amely azonosítja az ébresztést okozó GPIO-t.

A következő vázlat az összes változtatást tejesítette.

```
/*
  Rui Santos & Sara Santos - Random Nerd Tutorials
  https://RandomNerdTutorials.com/learn-esp32-with-arduino-ide/
  Ezúton ingyenes engedélyt adunk minden olyan személynek, aki másolatot
szerez a szoftverről és a kapcsolódó dokumentációs fájlokról.
  A fenti szerzői jogi megjegyzést és ezt az engedélyt tartalmazó megjegy-
zést a Szoftver minden másolatának vagy jelentős részének tartalmaznia
kell.
*/

#include "driver/rtc_io.h"

#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER hex-ben
#define WAKEUP_GPIO_2              GPIO_NUM_2     // Csak RTC IO engedélyezett –
```

```
ESP32 tű példa
#define WAKEUP_GPIO_15             GPIO_NUM_15     // Csak RTC IO engedélyezett –
ESP32 tű példa

// Definiálunk egy bitmaszkot a több GPIO-hoz
uint64_t bitmask = BUTTON_PIN_BITMASK(WAKEUP_GPIO_2) |
BUTTON_PIN_BITMASK(WAKEUP_GPIO_15);

RTC_DATA_ATTR int bootCount = 0;

/*
Metódus az ébresztést kiváltó GPIO kinyomtatásához
*/
void print_GPIO_wake_up(){
  int GPIO_reason = esp_sleep_get_ext1_wakeup_status();
  Serial.print("GPIO, amely kiváltotta az ébresztést: GPIO ");
  Serial.println((log(GPIO_reason))/log(2), 0);
}

/*
  Metódus az ok kinyomtatásához, ami miatt az ESP32 felébredt az alvásból
*/
void print_wakeup_reason() {
  esp_sleep_wakeup_cause_t wakeup_reason;

  wakeup_reason = esp_sleep_get_wakeup_cause();

  switch (wakeup_reason) {
    case ESP_SLEEP_WAKEUP_EXT0:
        Serial.println("Az RTC_IO használatával külső jel által okozott ébredés");
        break;
    case ESP_SLEEP_WAKEUP_EXT1:
        Serial.println("Az RTC_CNTL használatával külső jel által okozott ébre-
dés");
        print_GPIO_wake_up();
        break;
    case ESP_SLEEP_WAKEUP_TIMER:
        Serial.println("Az időzítő által okozott ébredés");
        break;
    case ESP_SLEEP_WAKEUP_TOUCHPAD:
        Serial.println("Az érintőpad okozta ébredés");
        break;
    case ESP_SLEEP_WAKEUP_ULP:
        Serial.println("Az ULP program által okozott ébredés");
        break;
    default:
        Serial.printf("Az ébredést nem a mélyalvás okozta: %d\n", wakeup_reason);
        break;
  }
}
```

```
void setup() {
  Serial.begin(115200);
  delay(1000);  //Szánunk egy kis időt a soros monitor megnyitására

  //Növeljük a bootolási számot, és minden újraindításkor kinyomtatjuk
  ++bootCount;
  Serial.println("Boot szám: " + String(bootCount));

  //Kinyomtatjuk az ESP32 ébresztésének okát
  print_wakeup_reason();

  //Az ext1-et használjuk ébresztési forrásként
  esp_sleep_enable_ext1_wakeup_io(bitmask, ESP_EXT1_WAKEUP_ANY_HIGH);
  // engedélyezzük a lehúzó ellenállásokat és tiltjuk felhúzó ellenállásokat
  rtc_gpio_pulldown_en(WAKEUP_GPIO_2);
  rtc_gpio_pullup_dis(WAKEUP_GPIO_2);
  rtc_gpio_pulldown_en(WAKEUP_GPIO_15);
  rtc_gpio_pullup_dis(WAKEUP_GPIO_15);

  //Most menj aludni
  Serial.println("Most megyek aludni");
  esp_deep_sleep_start();
  Serial.println("Ez soha nem nyomtatódik ki ");
}

void loop() {
  //Ez nem fog meghívódni
}
```

Hogyan működik a kód?

Vessünk egy pillantást a kód működésére.

GPIO Bitmaszk

A GPIO-k bitmaszkját a kód elején határozzuk meg:

```
#define BUTTON_PIN_BITMASK(GPIO) (1ULL << GPIO)  // 2 ^ GPIO_NUMBER hex-ben
#define WAKEUP_GPIO_2              GPIO_NUM_2     // Csak RTC IO engedélyezett
#define WAKEUP_GPIO_15             GPIO_NUM_15     // Csak RTC IO engedélyezett

// Definiálunk egy bitmaszkot a több GPIO-hoz
uint64_t bitmask = BUTTON_PIN_BITMASK(WAKEUP_GPIO_2) |
BUTTON_PIN_BITMASK(WAKEUP_GPIO_15);
```

Azonosítjuk azt a GPIO-t, amely az ébredést okozta

Létrehozunk egy `print_GPIO_wake_up()` nevű függvényt, amely kiírja az ébresztést okozó GPIO-t:

```
void print_GPIO_wake_up(){
  int GPIO_reason = esp_sleep_get_ext1_wakeup_status();
  Serial.print("GPIO, amely kiváltotta az ébresztést: GPIO ");
```

```
  Serial.println((log(GPIO_reason))/log(2), 0);
}
```

Az `esp_sleep_get_ext1_wakeup_status()` függvény egy bitmaszkot ad vissza az ébresztést okozó GPIO-val. A GPIO-számot a következőképpen kaphatjuk meg:

```
log(GPIO_reason))/log(2)
```

Az ébredés okának kinyomtatása

Módosítottuk a `print_wakeup_reason()` függvényt, hogy kinyomtassa az ébresztést okozó GPIO-t, amikor az ébresztési forrás ext1:

```
case ESP_SLEEP_WAKEUP_EXT1:
    Serial.println("Az RTC_CNTL használatával külső jel által okozott ébredés");
    print_GPIO_wake_up();
    break;
```

Az ext1 engedélyezése ébresztőforrásként

Engedélyezzük az ext1-et ébresztési forrásként a GPIO bitmaszk és ébresztési mód átadásával.

```
esp_sleep_enable_ext1_wakeup_io(bitmask, ESP_EXT1_WAKEUP_ANY_HIGH);
```

Ne felejtsd el letiltani a belső felhúzó ellenállásokat és engedélyezni a lehúzó ellenállásokat az ébresztőforrásként használt GPIO-kon.

```
rtc_gpio_pulldown_en(WAKEUP_GPIO_2);
rtc_gpio_pullup_dis(WAKEUP_GPIO_2);
rtc_gpio_pulldown_en(WAKEUP_GPIO_15);
rtc_gpio_pullup_dis(WAKEUP_GPIO_15);
```

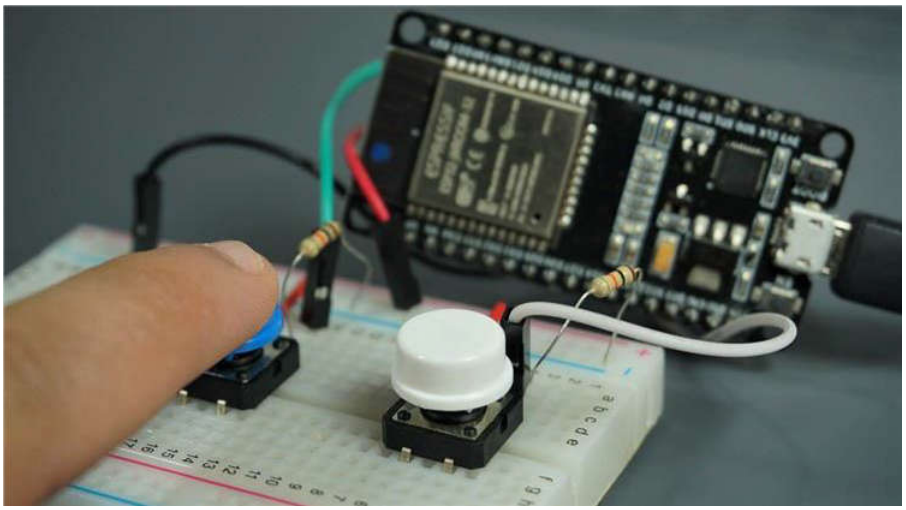
Mélyalvás engedélyezése

Végül meghívjuk az `esp_deep_sleep_start()` függvényt, hogy az ESP32 mély alvó módba kerüljön.

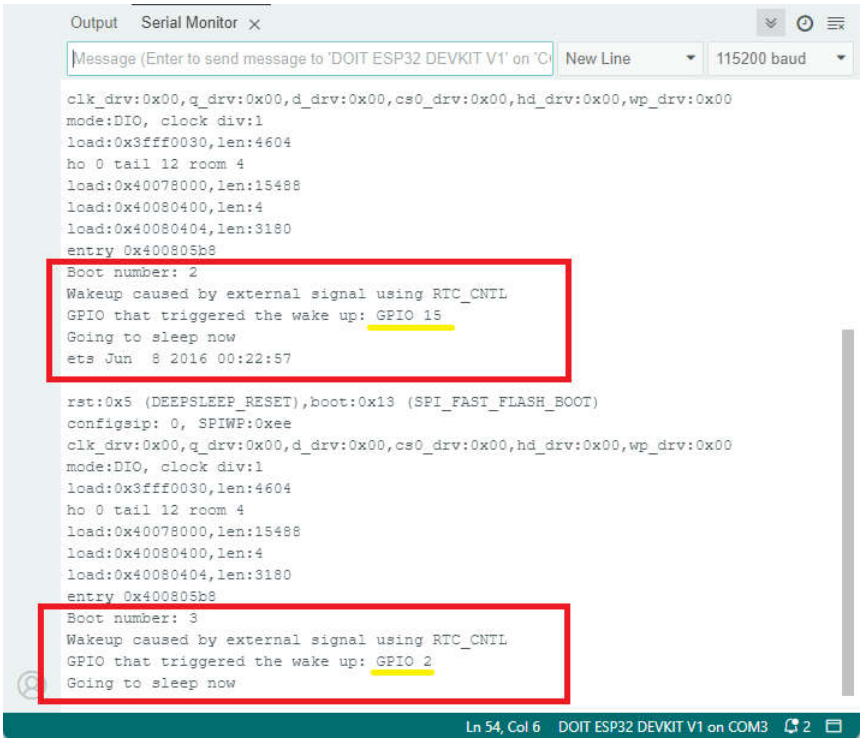
```
esp_deep_sleep_start();
```

A vázlat tesztelése

A kód kártyára való feltöltése után az ESP32 mély alvó módba kerül. A nyomógombok megnyomásával felébresztheted.



Nyisd meg a soros monitort 115200 adatátviteli sebességgel. Nyomd meg a nyomógombokat az ESP32 felébresztéséhez. A soros monitoron meg kell kapnod az ébresztés okát és az ébresztést okozó GPIO-t.



Összegzés

Ebben a cikkben megmutattuk, hogyan használhatod a mélyalvást az ESP32-vel, és hogyan ébresztheted fel. Az ESP32 felébreszthető egy időzítő, az érintőtűk vagy GPIO-állapot megváltoztatásával.

Foglaljuk össze, mit tanultál az egyes ébresztőforrásokról:

Időzített ébresztés

- Az időzített ébresztés engedélyezéséhez használd az `esp_sleep_enable_timer_wakeup(time_in_us)` függvényt;

- Használd az `esp_deep_sleep_start()` függvényt a mélyalvás elindításához.

Érintéses ébresztés

- Az érintőtűk ébresztési forrásként való engedélyezéséhez használd a `touchSleepWakeUpEnable()` függvényt;
- Végül az `esp_deep_sleep_start()` függvény segítségével az ESP32-t mélyalvás módba helyezed.

Külső ébresztés

- Csak az RTC GPIO-kat használhatod külső ébresztésként;
- Két különböző módszert használhatsz: ext0 és ext1;
- Az ext0 lehetővé teszi az ESP32 felébresztését egyetlen GPIO tű használatával;
- Az ext1 lehetővé teszi az ESP32 felébresztését több GPIO érintkező használatával.

Reméljük, hogy hasznosnak találtad ezt az útmutatót. Ha többet szeretnél megtudni az ESP32-ről, ellenőrizd az összes ESP32 projektünket és az ESP32 e-könyvünket.