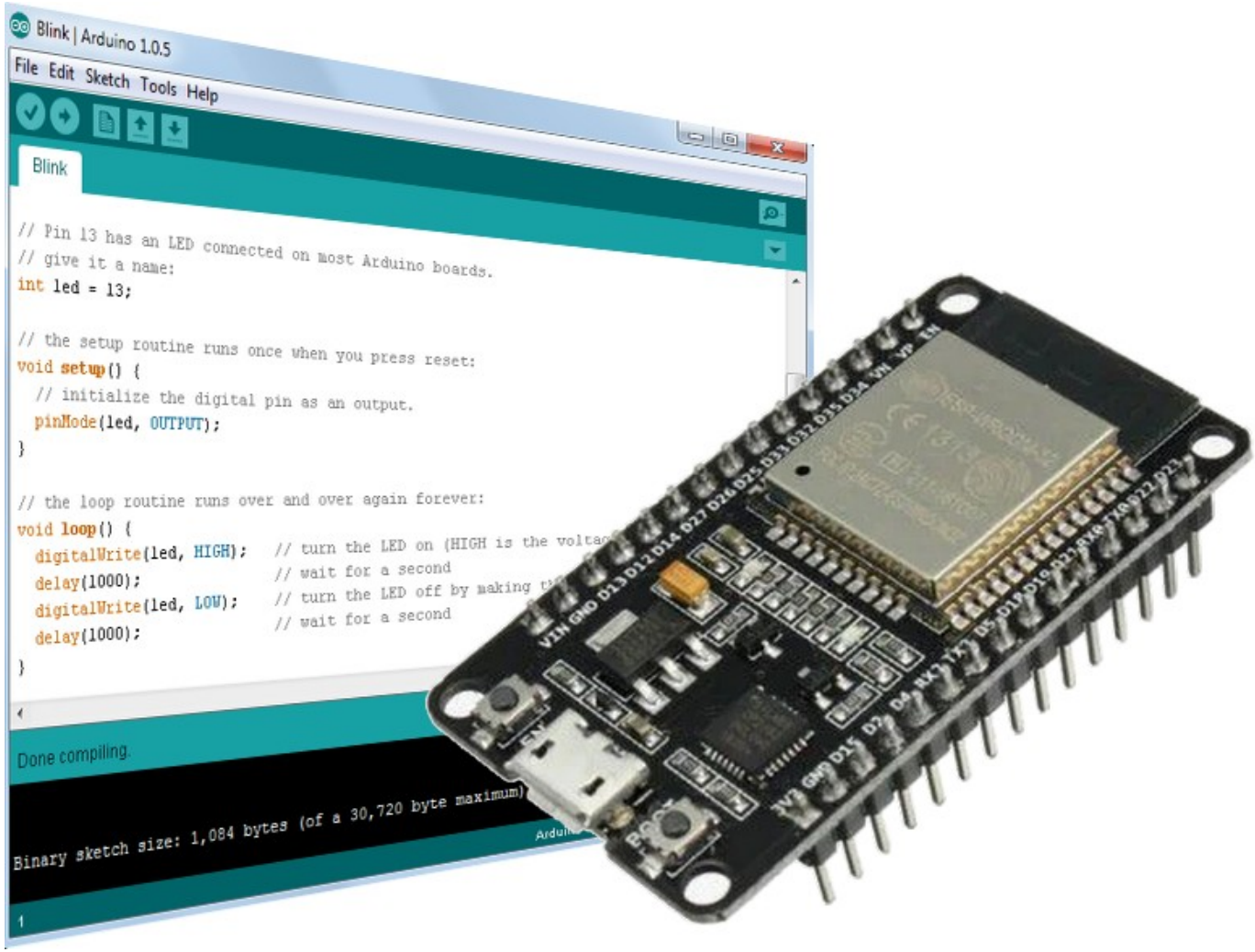
ESP32 mikrovezérlők programozása Arduino környezetben

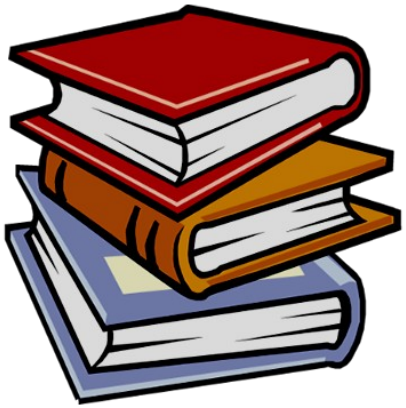


ARDUINO™

9. ESP-NOW kommunikáció

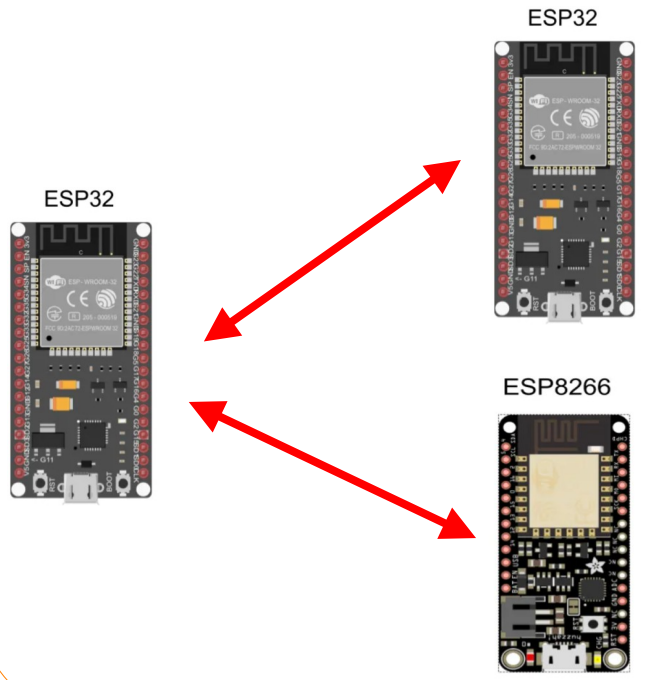
Felhasznált és ajánlott irodalom

- **Leírások, dokumentáció**
  - ESP-IDF : ESP-NOW API reference (ESP32)
  - ESPRESSIF: ESP-NOW in ESP8266 Non-OS SDK
  - ESPRESSIF: ESP32 Arduino Core Documentation



- **Mintaprojektek**
  - Microcontrollerslab: Getting Started with ESP-NOW (ESP32)
  - Random Nerd Tutorials: Getting Started with ESP-NOW (ESP32)
  - Francesco Azzola: How to connect ESP32 and ESP8266 using ESP-Now protocol
  - Random Nerd Tutorials: Getting Started with ESP-NOW (ESP8266)
- **Arduino programkönyvtárak**
  - Fabio Durigon: OLED SSD1306 - SH1106
  - Adafruit: Adafruit Unified Sensor
  - Adafruit: BME280 sensor library
  - Adafruit: DHT sensor library

ESP-NOW



- Az **ESP-NOW** az **Espressif** által kifejlesztett protokoll, amely közvetlen, vezeték nélküli kapcsolatot tesz lehetővé két, vagy több **ESP** eszköz (**ESP32,** vagy **ESP8266**) között
- Az **ESP-NOW** protokoll hasonló az alacsony fogyasztású, 2,4 GHz-es vezeték nélküli kapcsolathoz, amelyet gyakran alkalmaznak a vezeték nélküli egerekben
- Az eszközök között párosításra van szükség a kommunikáció előtt, a párosítás után a kapcsolat biztonságos és **peer-to-peer** (nincs szükség kézfogásra, sem routerhez történő kapcsolódásra)
- Üzenetküldéskor a címzéshez az ellenoldal **MAC** címét használjuk, a párosítás tehát a **MAC** címek begyűjtését vagy megszerzését jelenti
- Lehetőség van ún. **broadcast** (mindenkinek szóló) üzenetek küldésére is, amikor az `FF-FF-FF-FF-FF-FF` **MAC** címzést használjuk
- Az egy üzenetben továbbított adat legfeljebb **250 bájt** lehet
- Az üzenetek lehetnek **titkosítottak,** vagy **titkosítás nélküliek**

ESP-NOW API eltérések

ESP32

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

if (esp_now_init() != ESP_OK) {}
```

nincs

ESP8266

```
#include <espnow.h>
#include <ESP8266WiFi.h>

if (esp_now_init() != 0) {}

esp_now_set_self_role(role)
```

```
esp_now_register_recv_cb(onRecv)

esp_now_register_send_cb(onSent)
```

```
esp_now_peer_info peerInfo;
  memcpy(peerInfo.peer_addr,addr,6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

esp_err_t esp_now_add_peer(&peerInfo)

esp_err_t esp_now_send(addr,&msg,size)
```

(ha `addr = NULL`, akkor a *peer listában* regisztrált címekre megy az üzenet)

nincs ilyen

```
int esp_now_add_peer(addr,role,1,NULL,0)

int esp_now_send(addr,&msg,size)
```

A MAC címek beszerzése

- A **MAC** (Media Access Control) cím egy egyedi azonosító, amelyet a hálózati hardver (pl. **WiFi** vagy Ethernet kártya) gyártója rendel az eszközhöz. A **MAC** cím 6 db kétjegyű hexadecimális számból áll
- Az **ESP-NOW** kommunikációnál az eszközök címzése **MAC** címükkel történik ezért ismernünk kell az **ESP** eszközeink **MAC** címét
- Az alábbi program RESET után kiírja az **ESP** eszköz **MAC** címét

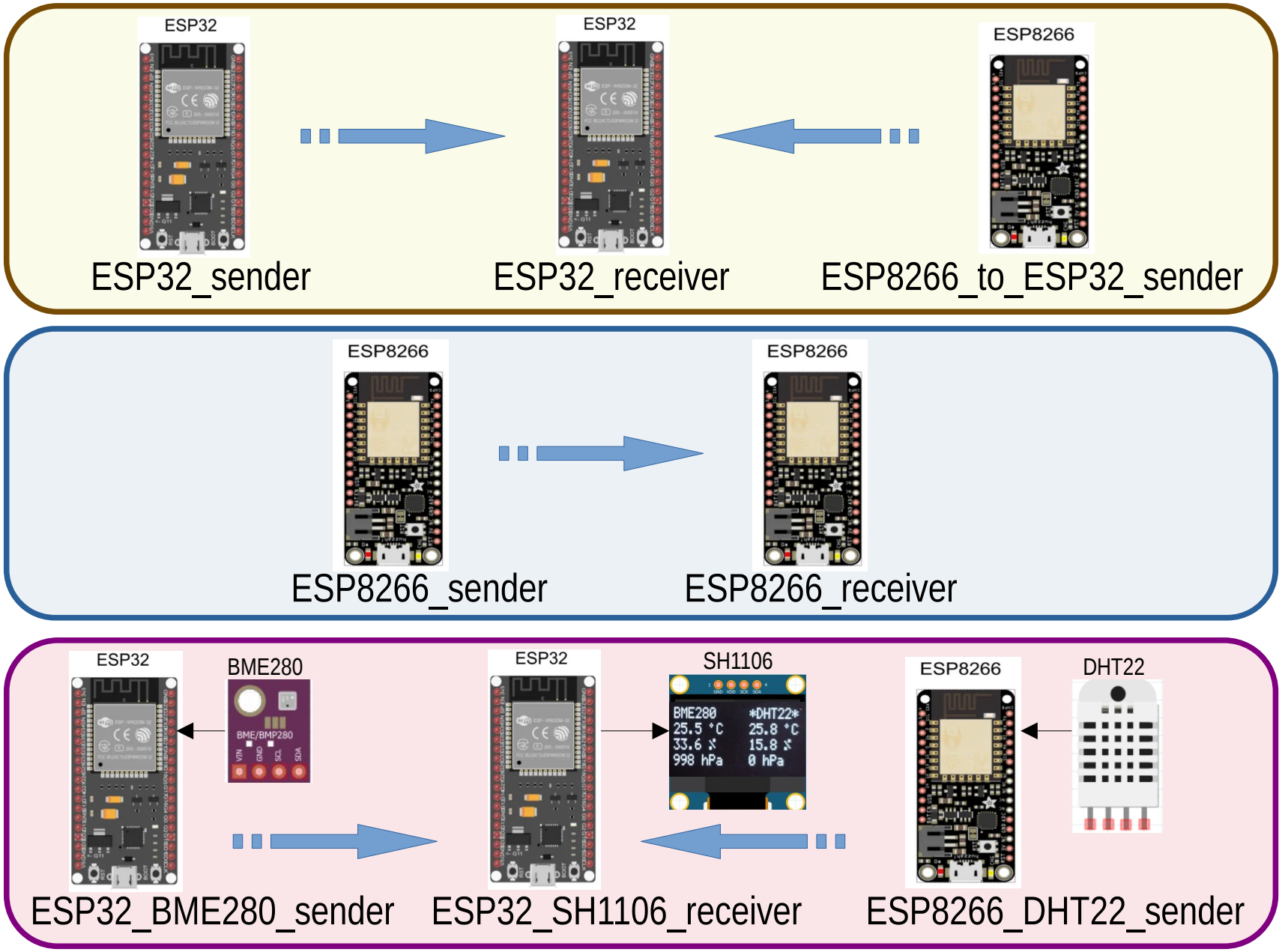
ESP32

```
#include "WiFi.h"

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

void loop(){

}
```

ESP8266

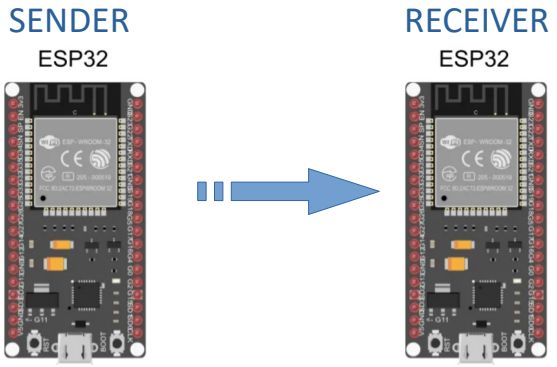
```
#include "ESP8266WiFi.h"

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

void loop(){

}
```

Példaprogramok



ESP32_sender.ino 2/1.

- Egyszerű egyirányú adatküldés: 2 másodpercenként elküldünk egy struktúrát, amelyben illusztráció gyanánt szöveg, int, float és bool típusú adat is szerepel
- Ismernünk kell a fogadó modul MAC címét

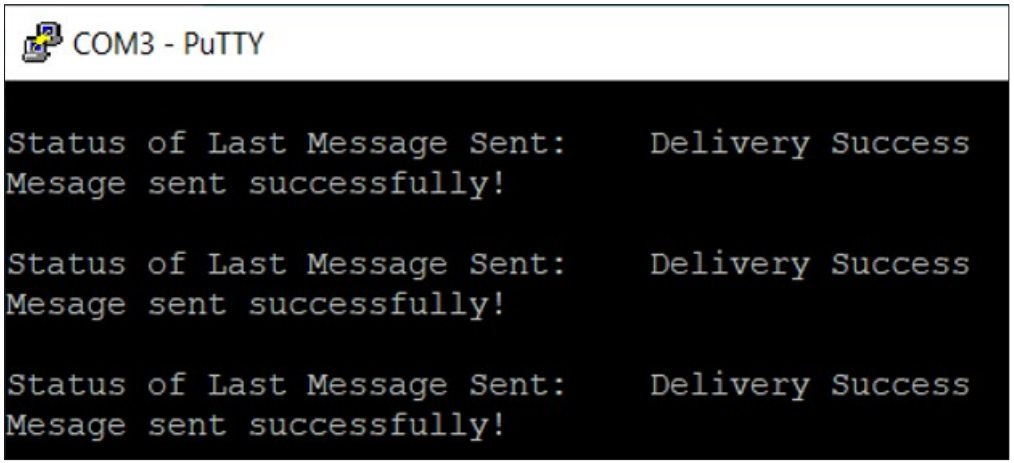



```
#include <esp_now.h>
#include <WiFi.h>
uint8_t recvAddress[] = {0x24,0x62,0xAB,0xCA,0x55,0xCC}; // Address of receiver
typedef struct {
  char character[100];
  int integer;
  float floating_value;
  bool bool_value;
} struct_message;

struct_message message;
esp_now_peer_info_t peerInfo;

// Callback function at sending data
void data_sent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nStatus of Last Message Sent:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Delivery Success" : "Delivery Fail");
}
```

- Felhasznált forrás: Microcontrollerslab: Getting Started with ESP-NOW (ESP32)
- Javítás: a **peerInfo** struktúra globális változó kell, hogy legyen (az eredeti példában **setup()** lokális változója volt)

ESP32_sender.ino 2/2.

```
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) { Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_send_cb(data_sent);
  memcpy(peerInfo.peer_addr, broadcastAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo)!=ESP_OK){
    Serial.println("Failed to add peer");
    return;
  }
}
```



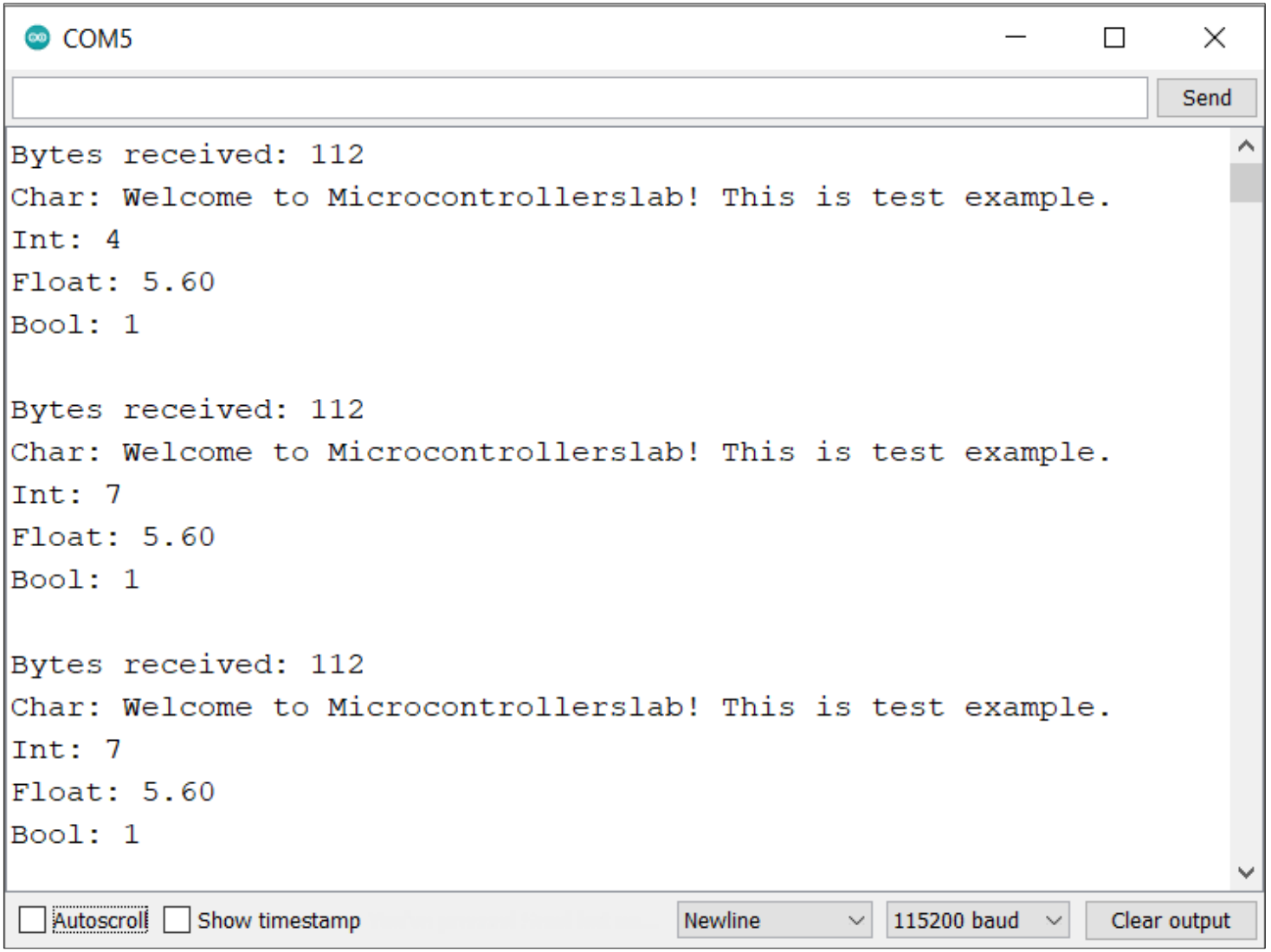

```
void loop() {
  strcpy(message.character, "Welcome to Microcontrollerslab! This is test example.");
  message.integer = random(1,10);
  message.floating_value = 5.6;
  message.bool_value = true;
  esp_err_t outcome = esp_now_send(recvAddress, (uint8_t *) &message, sizeof(message));
  if (outcome == ESP_OK) { Serial.println("Mesage sent successfully!"); }
  else { Serial.println("Error sending the message"); }
  delay(2000);
}
```

ESP32_receiver.ino

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

typedef struct { char character[100]; int integer;
                 float floating_value; bool bool_value;
               } struct_message;

struct_message message;

void data_receive(const uint8_t * mac, const uint8_t *incomingData, int len) {
  memcpy(&message, incomingData, sizeof(message));
  Serial.print("Bytes received: "); Serial.println(len);
  Serial.print("Char: ");  Serial.println(message.character);
  Serial.print("Int: ");   Serial.println(message.integer);
  Serial.print("Float: "); Serial.println(message.floating_value);
  Serial.print("Bool: ");  Serial.println(message.bool_value);
  Serial.println();
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init()!=ESP_OK) {Serial.println("Error initializing ESP-NOW"); return;}
  esp_now_register_recv_cb(data_receive);
}

void loop() {}
```

ESP32_receiver.ino

- Az adatküldés eredménye (a vevőoldali kiíratások)



ESP8266_to_ESP32_sender.ino 2/1.

- Ebben a példában az **ESP32_sender** programot írjuk át **ESP8266**-ra
- A fogadó eszköz változatlanul az **ESP32_receiver** programot futtató **ESP32** modul lesz, amelynek immár két eszközről is küldünk üzeneteket

```
#include <ESP8266WiFi.h>
#include <espnow.h>
// Mac address of the slave
uint8_t peer1[] = {0x24,0x62,0xAB,0xCA,0x55,0xCC};

typedef struct message {
  char character[100];
  int integer;
  float floating_value;
  bool bool_value;
} message;

struct message myMessage;

void onSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.println("Status:");
  Serial.println(sendStatus);
}
```

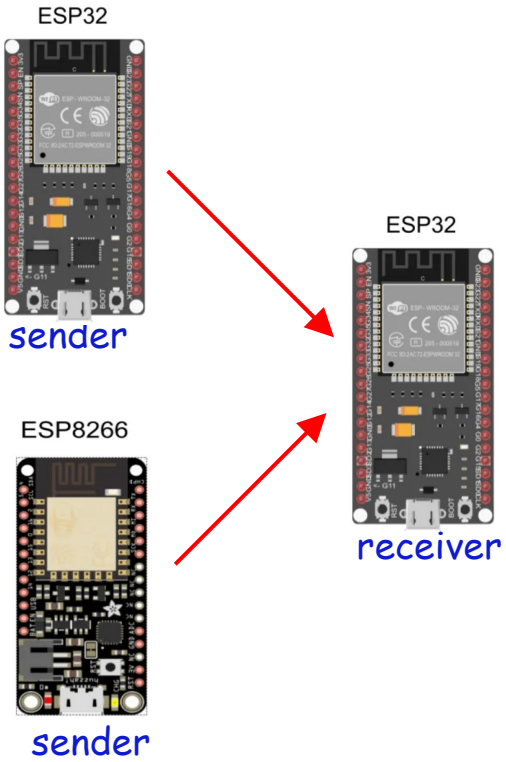
ESP8266_to_ESP32_sender.ino 2/2.

- Meg kell adni az eszközök szerepét (role)
- A hibajelzők értéke ***int*** típusú
- A ***peer*** jellemzői itt paraméterek

```
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != 0) {
    Serial.println("Problem during ESP-NOW init");
    return;
  }
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  Serial.println("Registering a peer");
  esp_now_add_peer(peer1, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
  Serial.println("Registering send callback function");
  esp_now_register_send_cb(onSent);
}

void loop() {
  strcpy(myMessage.character, "Message from the ESP8266 #12");
  myMessage.integer = random(1,10);
  myMessage.floating_value = 2.3;
  myMessage.bool_value = false;
  Serial.println("Send a new message");
  esp_now_send(NULL, (uint8_t *) &myMessage, sizeof(myMessage));
  delay(2000);
}
```

Ketten egy ellen ...

- Az **ESP32_receiver** mind a két küldő üzeneteit megkapja...



COM5

Send

```
Bytes received: 112
Char: Message from the ESP8266 #12
Int: 8
Float: 2.30
Bool: 0

Bytes received: 112
Char: Welcome to Microcontrollerslab! This is test example.
Int: 8
Float: 5.60
Bool: 1

Bytes received: 112
Char: Message from the ESP8266 #12
Int: 5
Float: 2.30
Bool: 0
```

Autoscroll Show timestamp Newline 115200 baud Clear output

ESP8266_sender.ino 2/1.

- Ebben a példában broadcast címet használunk és egy képzeletbeli RGB fényforrás vezérléséhez küldünk ki egy számhármast, két másodpercenként
- Itt most a példa kedvéért a küldő (sender) és a vevő (receiver) is **ESP8266,** a vett adatokat most csak kiíratjuk, de a program továbbfejleszthető…

Forrás: Francesco Azzola, How to connect ESP32 and ESP8266 using ESP-Now protocol

```
#include <ESP8266WiFi.h>
#include <espnow.h>
// Mac address of the slave
uint8_t peer1[] = {0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};  // Broadcast address
typedef struct message {
   int red;
   int green;
   int blue;
};

struct message myMessage;

void onSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Status: ");
  Serial.println(sendStatus);
}
```

ESP8266_sender.ino 2/2.

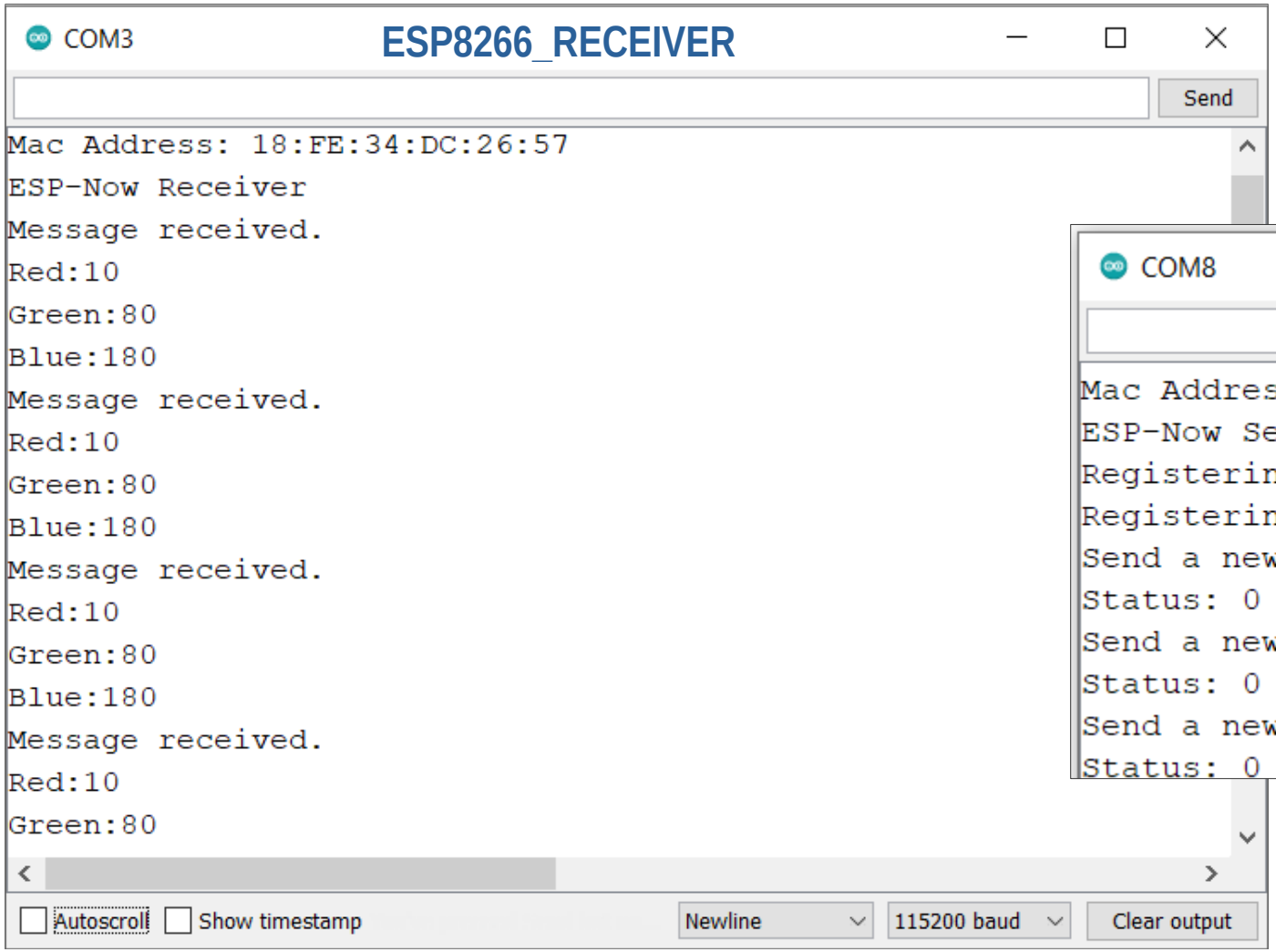
```
void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != 0) {
    Serial.println("Problem during ESP-NOW init");
    return;
  }
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  // Register the peer
  Serial.println("Registering a peer");
  esp_now_add_peer(peer1, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
  Serial.println("Registering send callback function");
  esp_now_register_send_cb(onSent);
}

void loop() {
  myMessage.red = 10;
  myMessage.green = 80;
  myMessage.blue = 180;
  Serial.println("Send a new message");
  esp_now_send(NULL, (uint8_t *) &myMessage, sizeof(myMessage));
  delay(2000);
}
```

ESP8266_receiver.ino

```
#include <ESP8266WiFi.h>
#include <espnow.h>
typedef struct message {int red; int green; int blue; } message;
message myMessage;
void onDataReceiver(uint8_t * mac, uint8_t *incomingData, uint8_t len) {
  Serial.println("Message received.");
  memcpy(&myMessage, incomingData, sizeof(myMessage));
  Serial.print("Red:");   Serial.println(myMessage.red);
  Serial.print("Green:"); Serial.println(myMessage.green);
  Serial.print("Blue:");  Serial.println(myMessage.blue);
}

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != 0) {
    Serial.println("Problem during ESP-NOW init");
    return;
  }
  //esp_now_set_self_role(ESP_NOW_ROLE_SLAVE);
  esp_now_register_recv_cb(onDataReceiver);
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

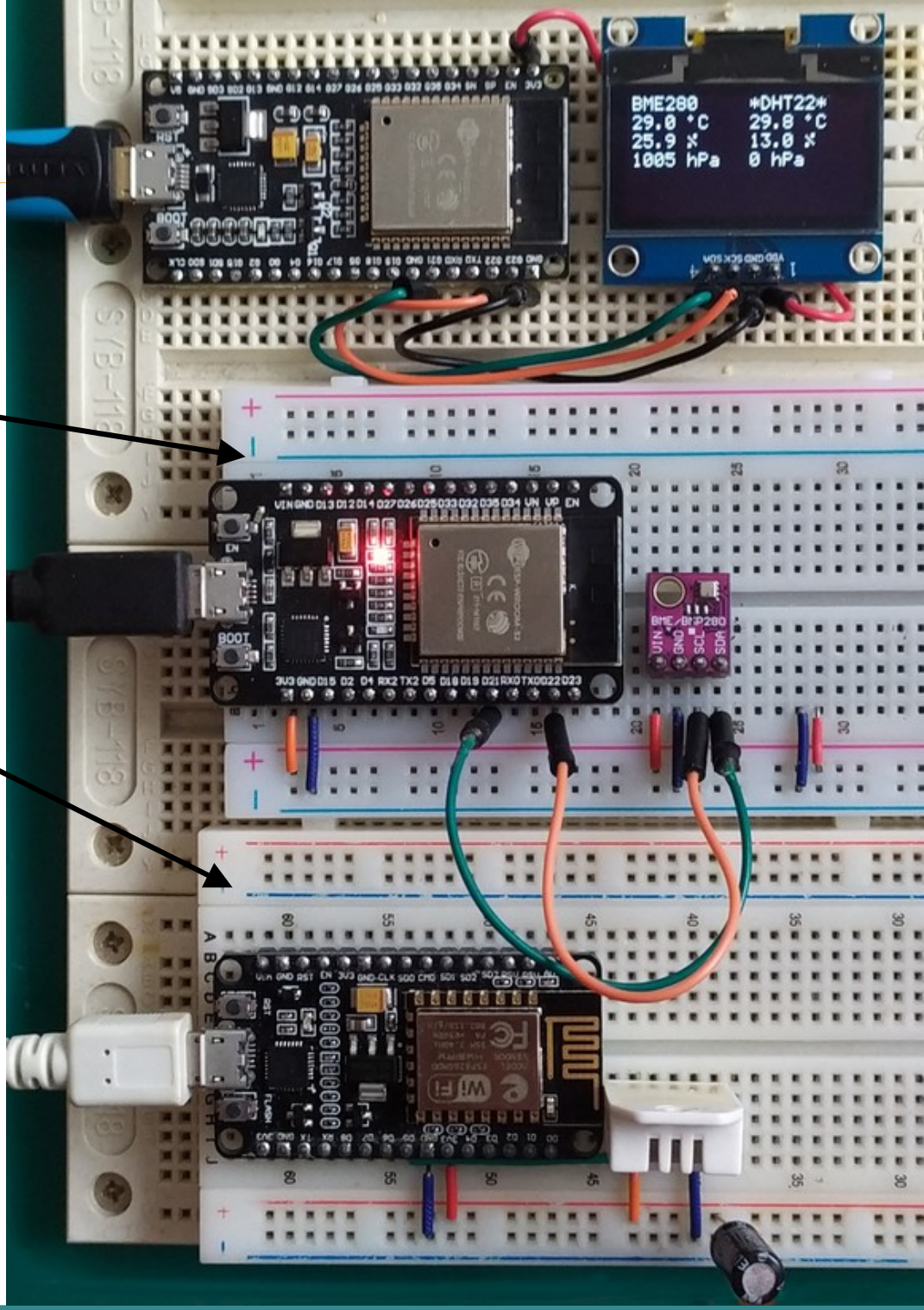
ESP8266_sender/receiver futási eredmény

- Az alábbi ábrákon a küldő és a vevő modul kiírásai láthatók



Szenzor adatokat gyűjtünk és mutogatunk

- Ebben a példában az első három mintaprogramot dolgoztuk át:
- Az **ESP32_BME280_sender** program egy **BME280** szenzor mérési adatait küldi el (hőmérséklet, páratartalom és légnyomás)
- Az **ESP8266_DHT22_sender** program egy **DHT22** szenzor mérési adatait küldi el (hőmérséklet és páratartalom)
- Az **ESP32_SH1106_receiver** program mindkét szenzor adatait fogadja és megjeleníti egy **I2C** OLED kijelzőn (itt **SH1106**, de lehetne **SSD1306** típusú is)



A BME280 szenzor

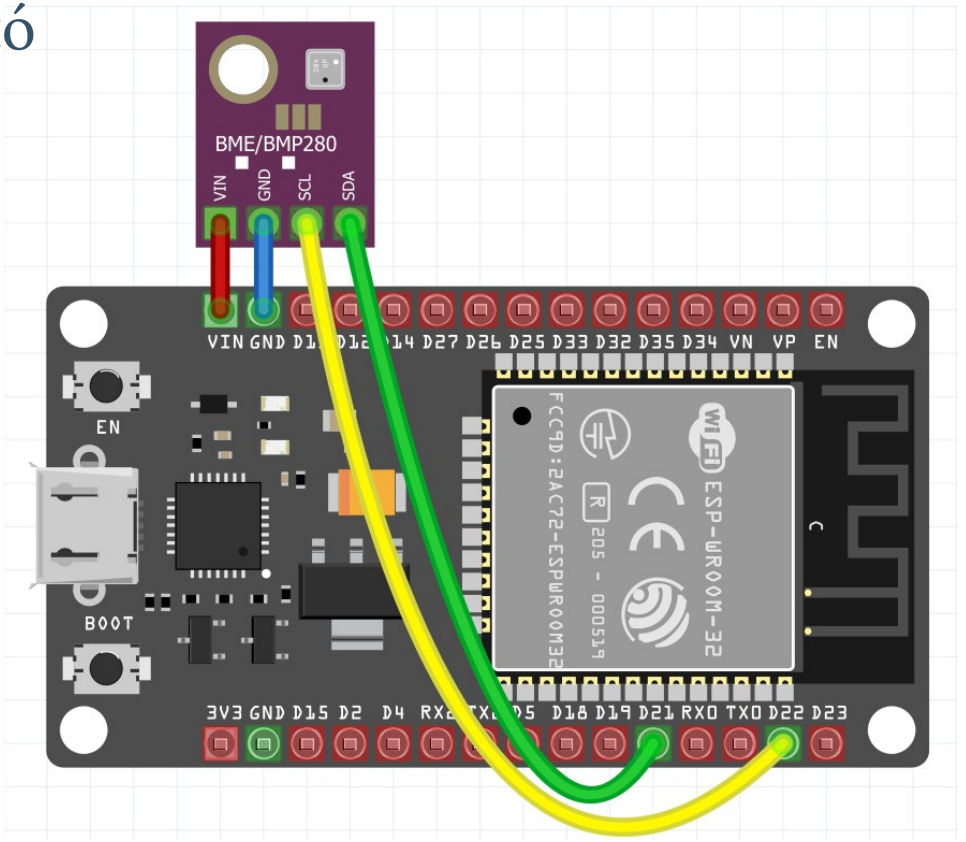
- A **BME** típusjelzésben az E betű „environmental”, azaz környezeti szenzort jelez, ami a hőmérsékleten és légnyomáson kívül a levegő relatív páratartalmának mérésére is alkalmas
- Az általam használt szenzor modulon az **I2C** cím **0x76,** tehát eltér az **Adafruit_BME280** programkönyvtár alapértelmezett címétől, a cím forrasztással megváltoztatható
- A bekötési vázlat az ábrán látható:

  SDA ⟶ GPIO21

  SCL ⟶ GPIO22

  VIN ⟶ VIN (5 V)

  GND ⟶ GND



ESP32_BME280_sender.ino 2/1.

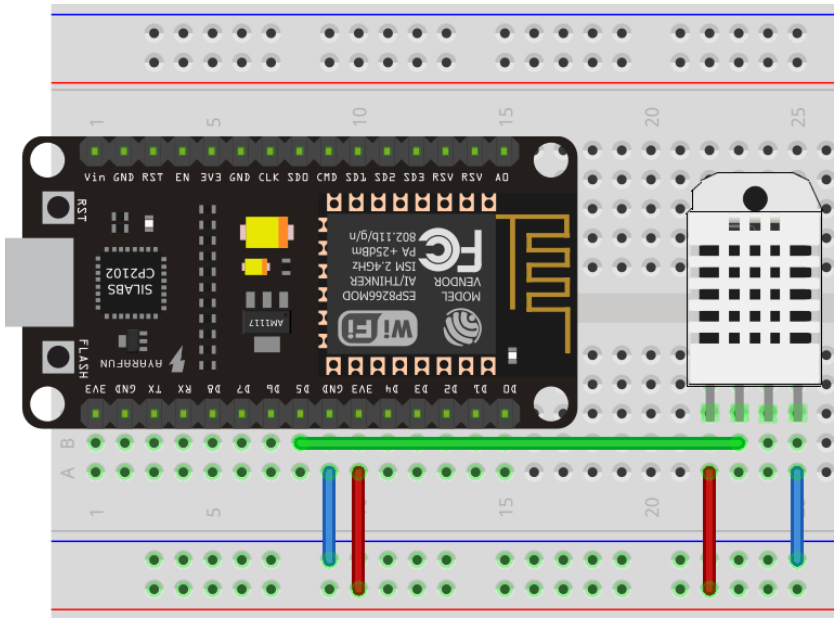
```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme;           // I2C
uint8_t recvAddress[] = {0x24, 0x62, 0xAB, 0xCA, 0x55, 0xCC};

typedef struct {
  char  myname[8] = "BME280 ";
  float temperature;
  float humidity;
  float pressure;
} struct_message;

struct_message message;
esp_now_peer_info_t peerInfo;
unsigned long lastMsg;

void data_sent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  Serial.print("\r\nStatus of Last Message Sent:\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ?"Delivery Success":"Delivery Fail");
}
```

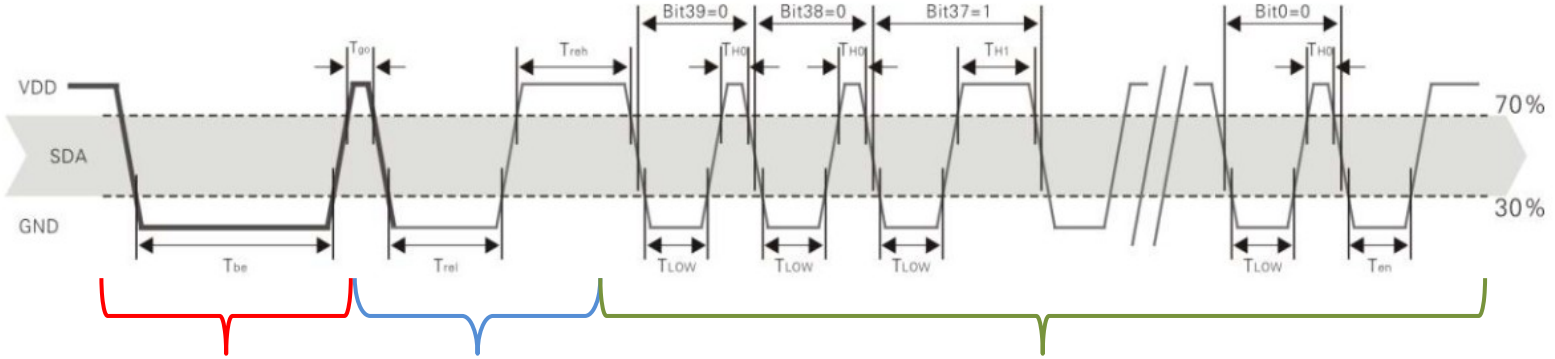
ESP32_BME280_sender.ino 2/2.

```
void setup() {
  Serial.begin(115200);
  bme.begin(0x76, &Wire);
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != ESP_OK) {Serial.println("Error initializing ESP-NOW");  return;}
  esp_now_register_send_cb(data_sent);
  memcpy(peerInfo.peer_addr, recvAddress, 6);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;
  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }
}

void loop() {
  if ((millis() - lastMsg) > 5000) {
    message.temperature = bme.readTemperature();
    message.humidity = bme.readHumidity();
    message.pressure = bme.readPressure()/98.58566F; // Sea level correction for Debrecen
    esp_err_t outcome = esp_now_send(recvAddress,(uint8_t *)&message,sizeof(message));
    if (outcome == ESP_OK) { Serial.println("Mesage sent successfully!"); }
    else { Serial.println("Error sending the message"); }
    lastMsg += 5000;
  }
}
```

ESP8266_DHT22_sender.ino

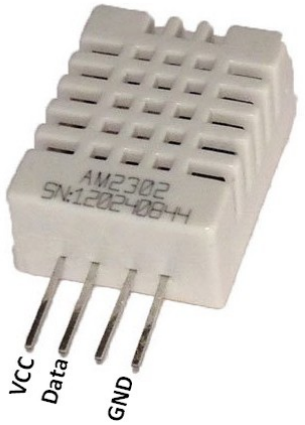
- Az **ESP8266_to_ESP32_sender.ino** programot módosítsuk úgy, hogy egy **DHT22** szenzor adatait küldje el
- **Emlékeztető: DHT22** szenzorral hőmérsékletet és rel. páratartalmat mérünk
- Könyvtárak: Adafruit_DHT, Adafruit Unified Sensor
- Adatlap: sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf





Host indítójel | Szenzor nyugtázó jel | 40 bitnyi adat: 32 bit információ + 8 bit ellenőrző összeg

Összesen tehát 85 időzítést tartalmaz egy-egy tranzakció …



ESP8266_DHT22_sender.ino 2/1.

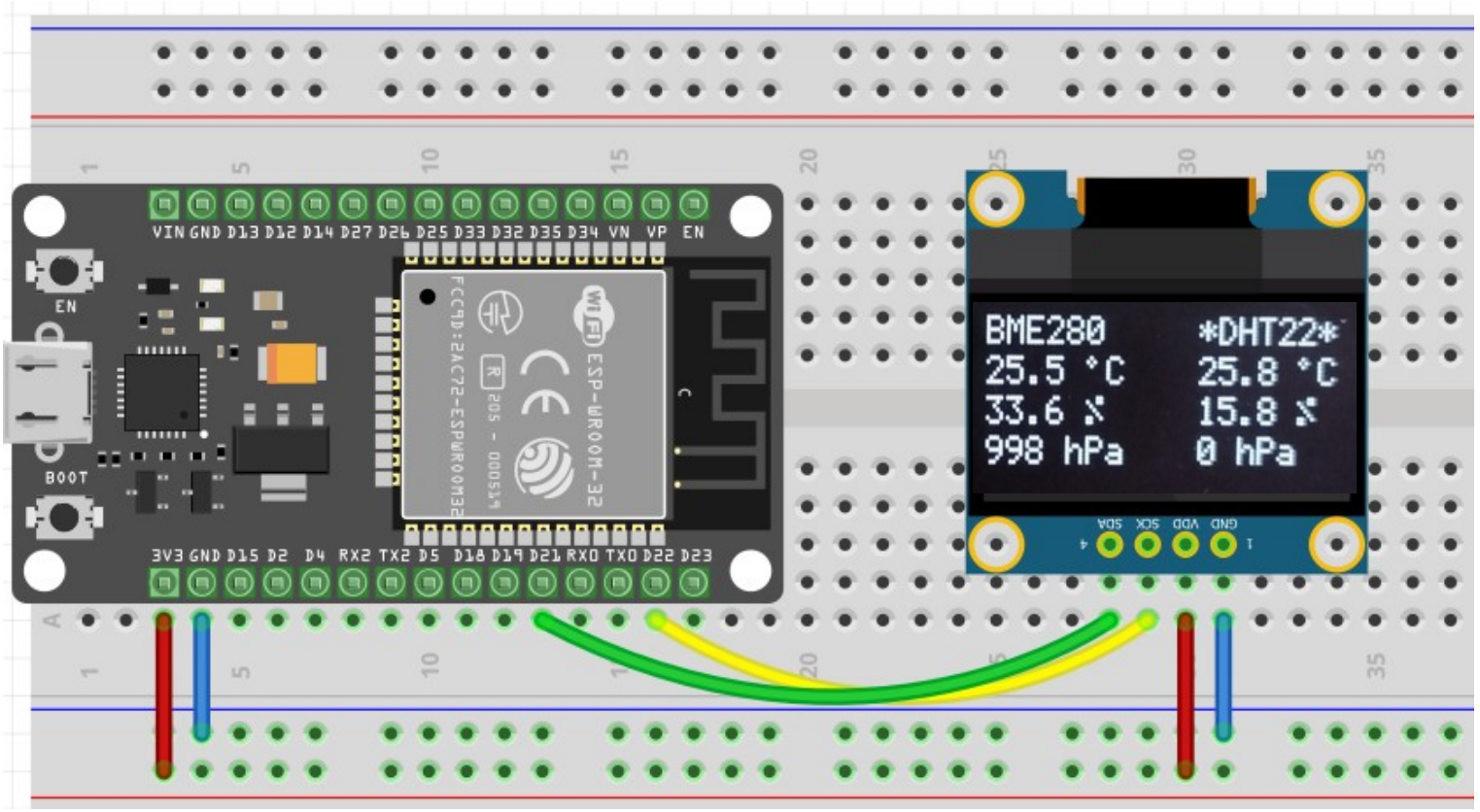
```
#include <ESP8266WiFi.h>
#include <espnow.h>
#include "DHT.h"

#define DHTPIN 14                    // GPIO14 (D5) pin for DHT sensor
DHT dht(DHTPIN, DHT22);

uint8_t recvAddress[] = {0x24, 0x62, 0xAB, 0xCA, 0x55, 0xCC};

typedef struct {
  char  myname[8] = "*DHT22*";
  float temperature;
  float humidity;
  float pressure=0.0f;               // We don't have pressure data...
} struct_message;

struct_message message;
unsigned long lastMsg;

void onSent(uint8_t *mac_addr, uint8_t sendStatus) {
  Serial.print("Status:");
  Serial.println(sendStatus);
}
```

ESP8266_DHT22_sender 2/2.

```
void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.mode(WIFI_STA);
  if (esp_now_init() != 0) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }
  esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
  esp_now_add_peer(recvAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
  esp_now_register_send_cb(onSent);
}

void loop() {
  if ((millis() - lastMsg) > 5000) {
    message.temperature = dht.readTemperature();
    message.humidity = dht.readHumidity();
    int outcome = esp_now_send(NULL,(uint8_t *)&message,sizeof(message));
    if (outcome == 0) { Serial.println("Mesage sent successfully!"); }
    else { Serial.println("Error sending the message"); }
    lastMsg += 5000;
  }
}
```

Bekötési vázlat

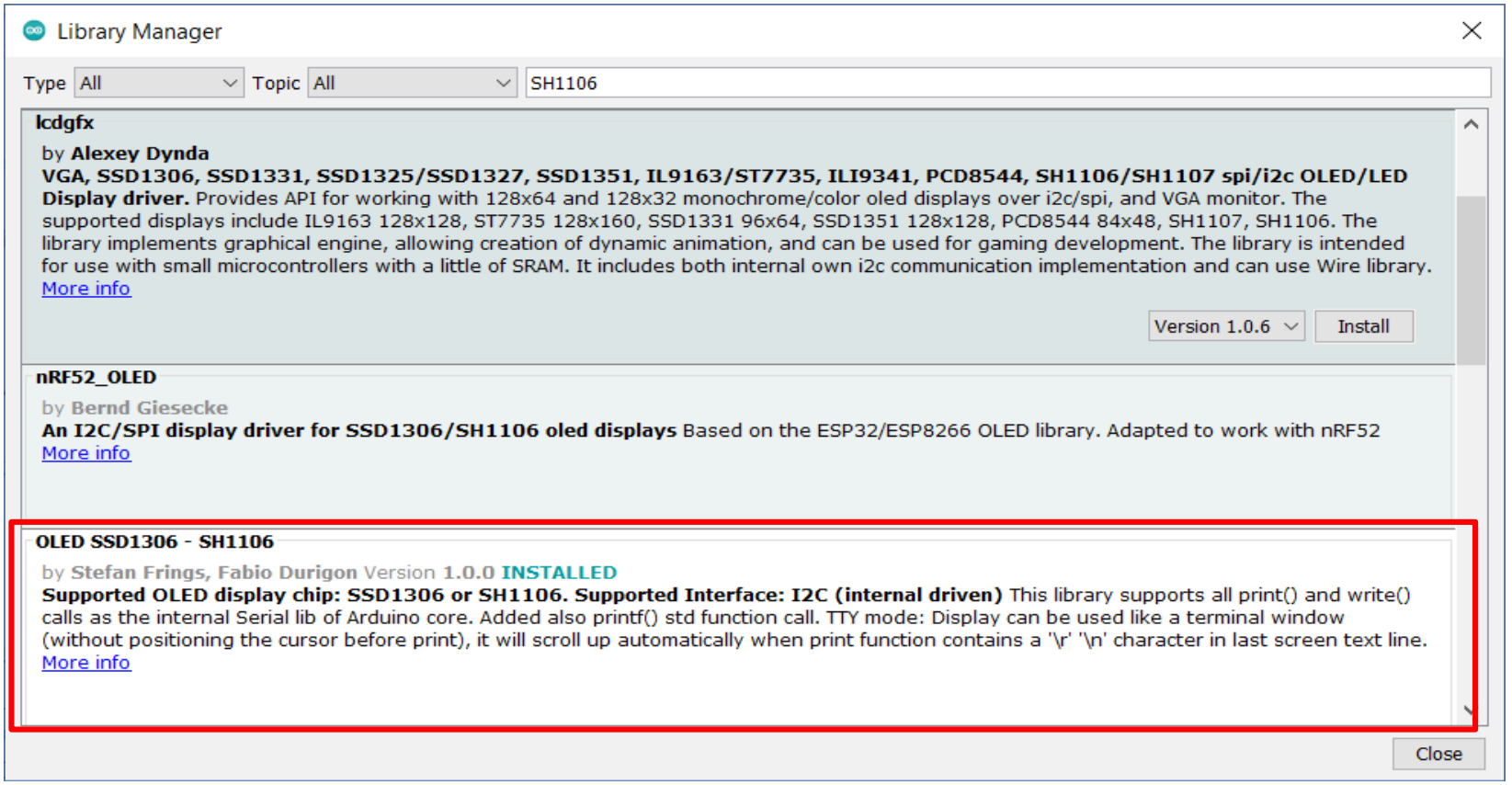
- Az **OLED** kijelző bekötésénél ügyeljünk a GND és a VDD kivezetések sorrendjére, mert az SSD1306 és az SH1106 kijelzők esetében különböző
- Itt most az alapértelmezett **I2C** kivezetéseket használjuk, de bármelyik digitális kimenetre köthettük volna, mert nem az **I2C** perifériával kezeljük a kijelzőt. Más **I2C** eszközzel nem közösíthetjük a kijelzőt!



Az OLED programkönyvtár

- Az **OLED SSD1306 – SH1106** könyvtárat használjuk, de az **oled.cpp** állományban a **display()** metódust **SH1106**-hoz javítani kellett:

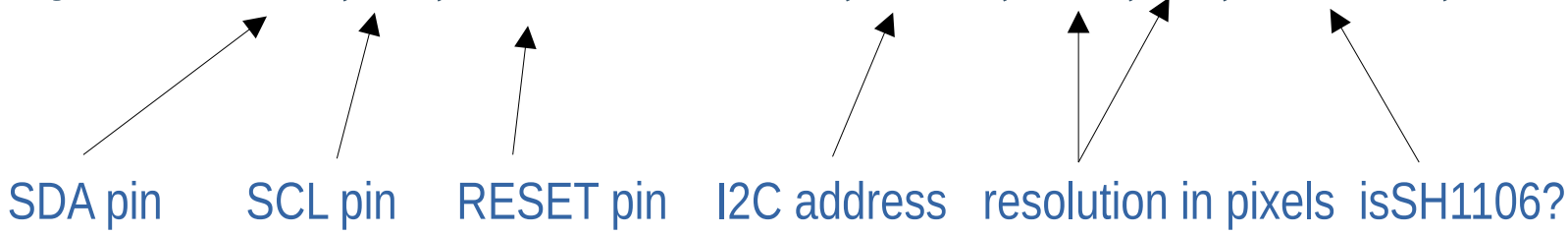
```
if (isSH1106) {
    i2c_send(0xB0 + page); // set page
    i2c_send(0x02);        // lower columns address =2
    i2c_send(0x10);        // upper columns address =0
}
```



A kijelző inicializálása

- Az OLED kijelző **I2C** címe **0x3C** (esetleg (0x3D)
- **Megjegyzés:** ez a programkönyvtár **szoftveres I2C** kezelést használ, ezért a kijelző nem közösíthető más **I2C** eszközzel!
- A kivezetéseket, az **I2C** címet és az egyéb paramétereket (grafikai felbontás, van-e RESET kivezetés, **SH1106** vagy **SSD1306** vezérlő) a konstruktornak kell megadni, például:

```
OLED display=OLED(21,22,NO_RESET_PIN,0x3C,128,64,false);
```



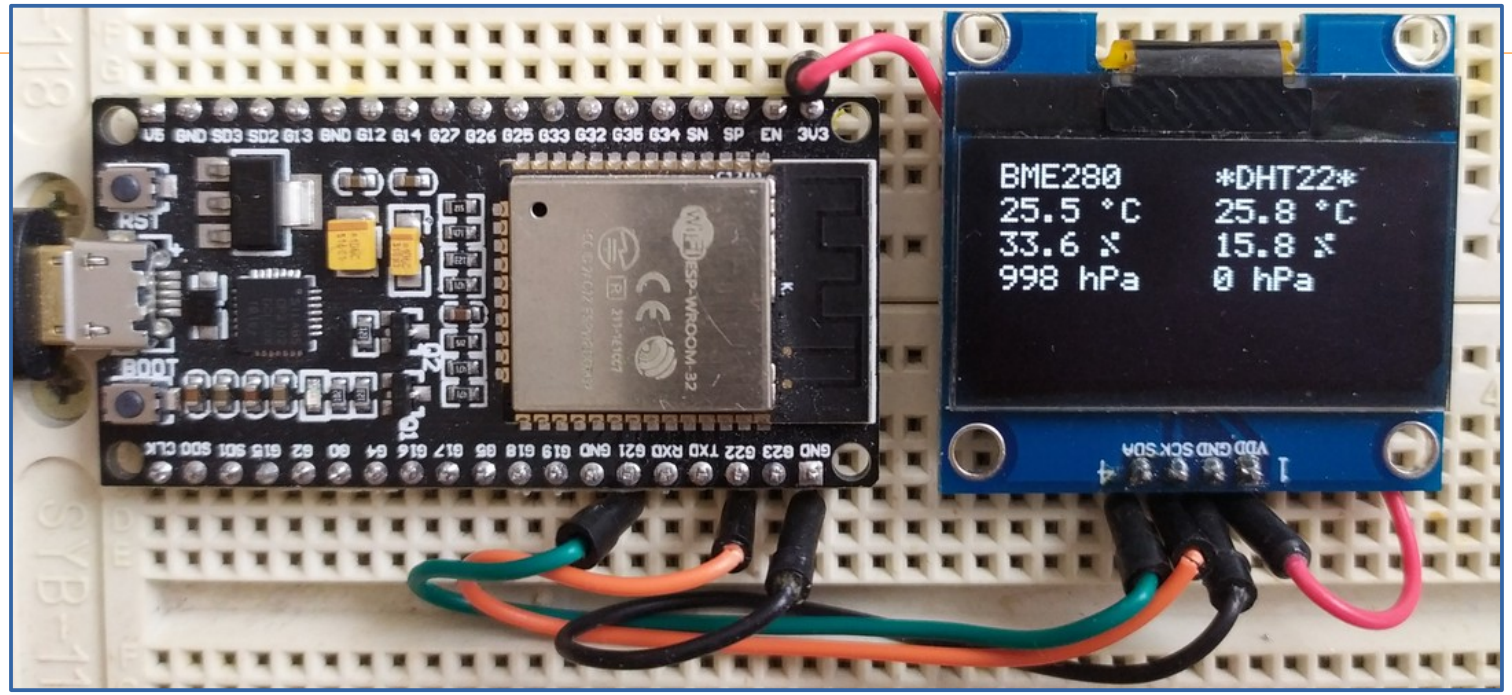

```
A fenti példa egy SSD1306 típusú 128 x 64 pixel
felbontású I2C kijelzőt definiál
```

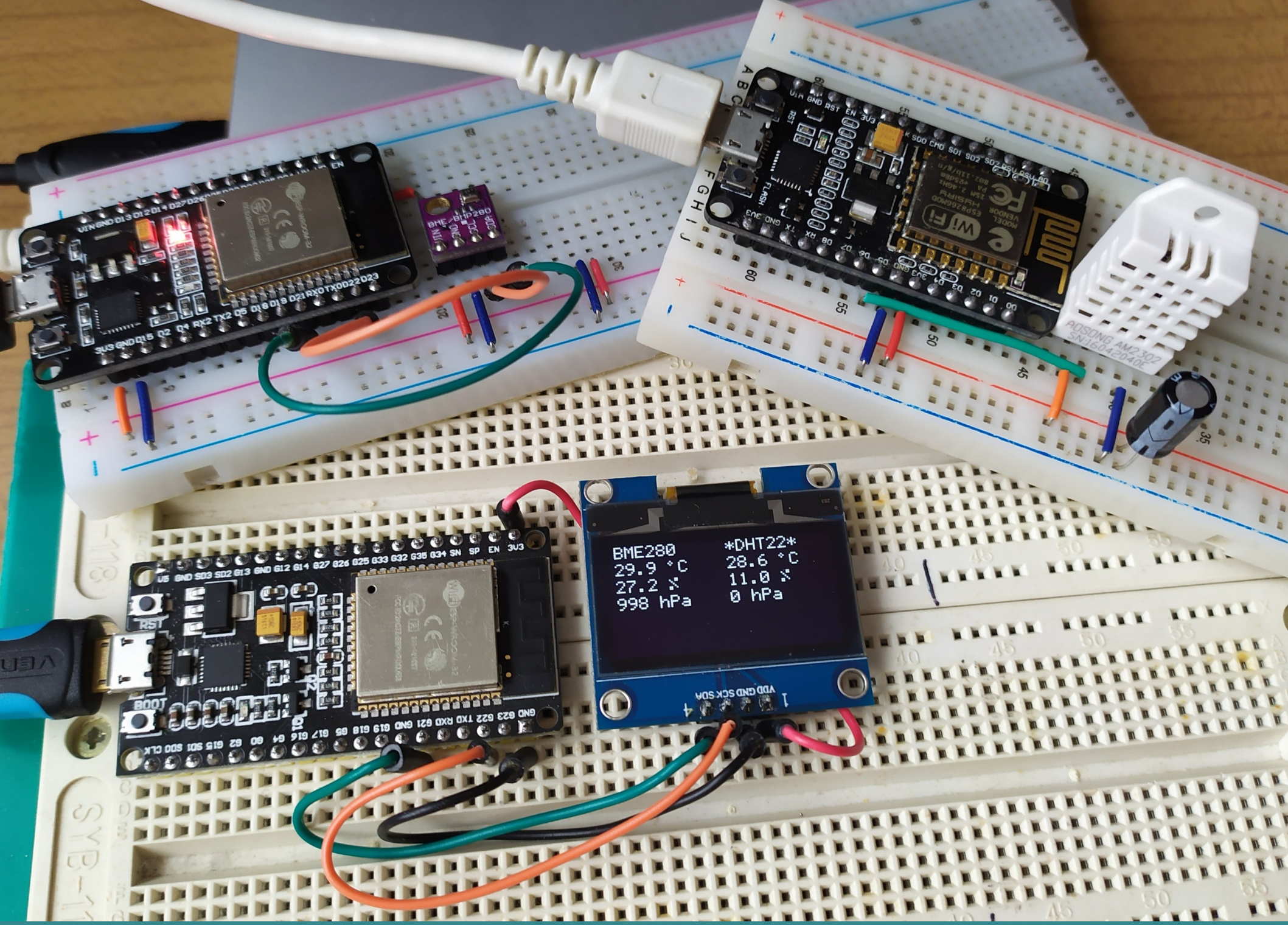
ESP32_SH1106_receiver.ino 2/1.

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

// OLED display = OLED(21, 22, NO_RESET_PIN, 0x3C, 128, 64, false); // SSD1306
OLED display = OLED(21, 22, NO_RESET_PIN, 0x3C, 128, 64, true); // SH1106

typedef struct {
  char  myname[8];
  float temperature;
  float humidity;
  float pressure;
} struct_message;
struct_message message;
static char msg[30];      // character buffer

void setup() {
  WiFi.mode(WIFI_STA);
  if (esp_now_init()!=ESP_OK) {Serial.println("Error initializing ESP-NOW");return;}
  esp_now_register_recv_cb(data_receive);
  display.begin();
  display.clear();
  display.display();
}

void loop() {}
```

ESP32_SH1106_receiver.ino 2/2.

```
void data_receive(uint8_t * mac, uint8_t *incomingData, int len) {
  int x = 0;
  memcpy(&message, incomingData, sizeof(message));
  if (message.myname[0] == 'B') { x = 0; } else { x = 64; }
  display.draw_rectangle(x, 0, x+63, 63, OLED::SOLID, OLED::BLACK);
  display.printf(x + 2, 0, "%s", message.myname);
  display.printf(x + 2, 10, "%.1fÂ°C", message.temperature);
  display.printf(x + 2, 20, "%.1f %%", message.humidity);
  display.printf(x + 2, 30, "%.0f hPa", message.pressure);
  display.display();
}
```

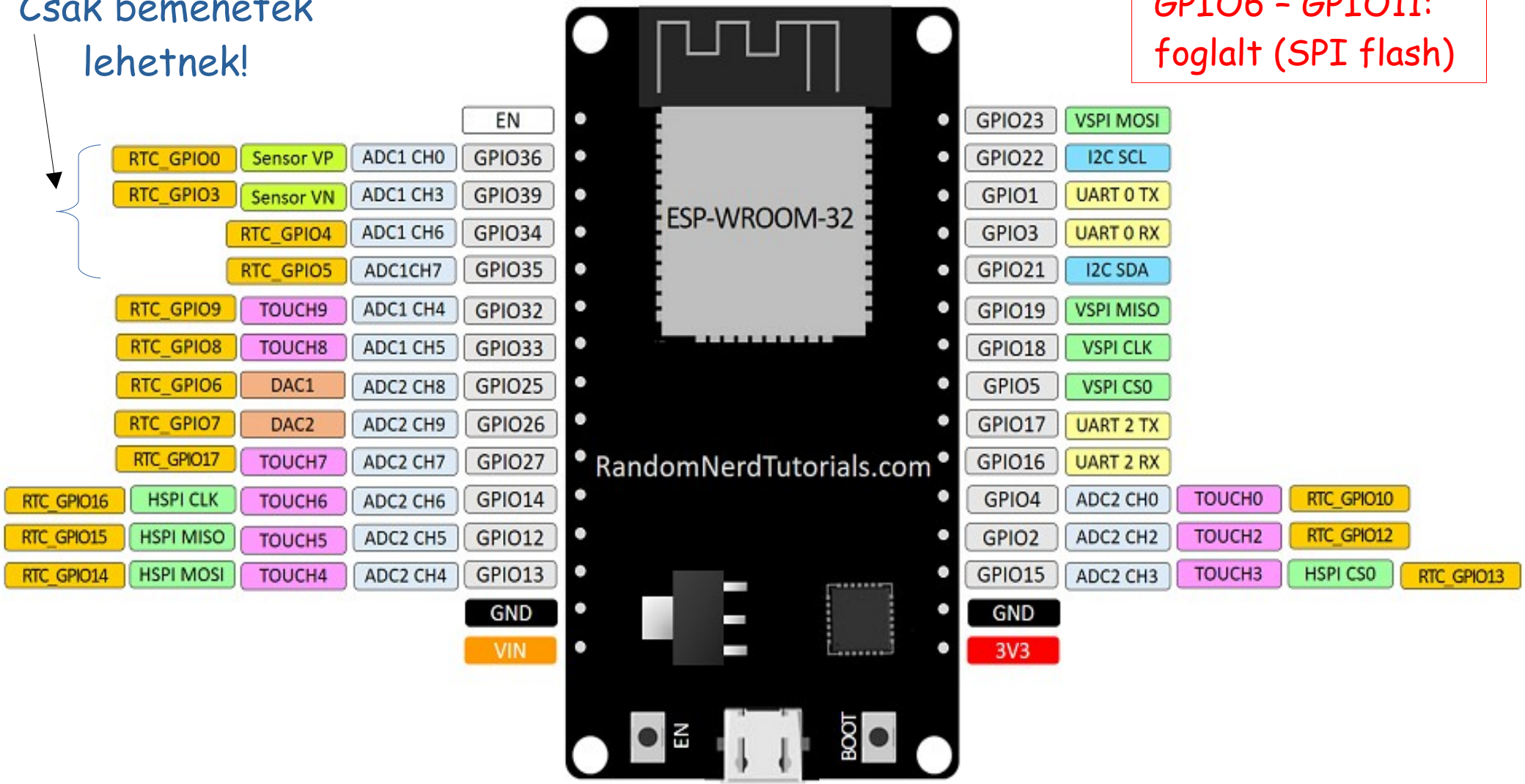



BME280
29.9 °C
27.2 %
998 hPa
DHT22
28.6 °C
11.0 %
0 hPa
AOSONG AM2302

A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



- **Forrás:** randomnerdtutorials.com/getting-started-with-esp32/

Ellenállás színkódok