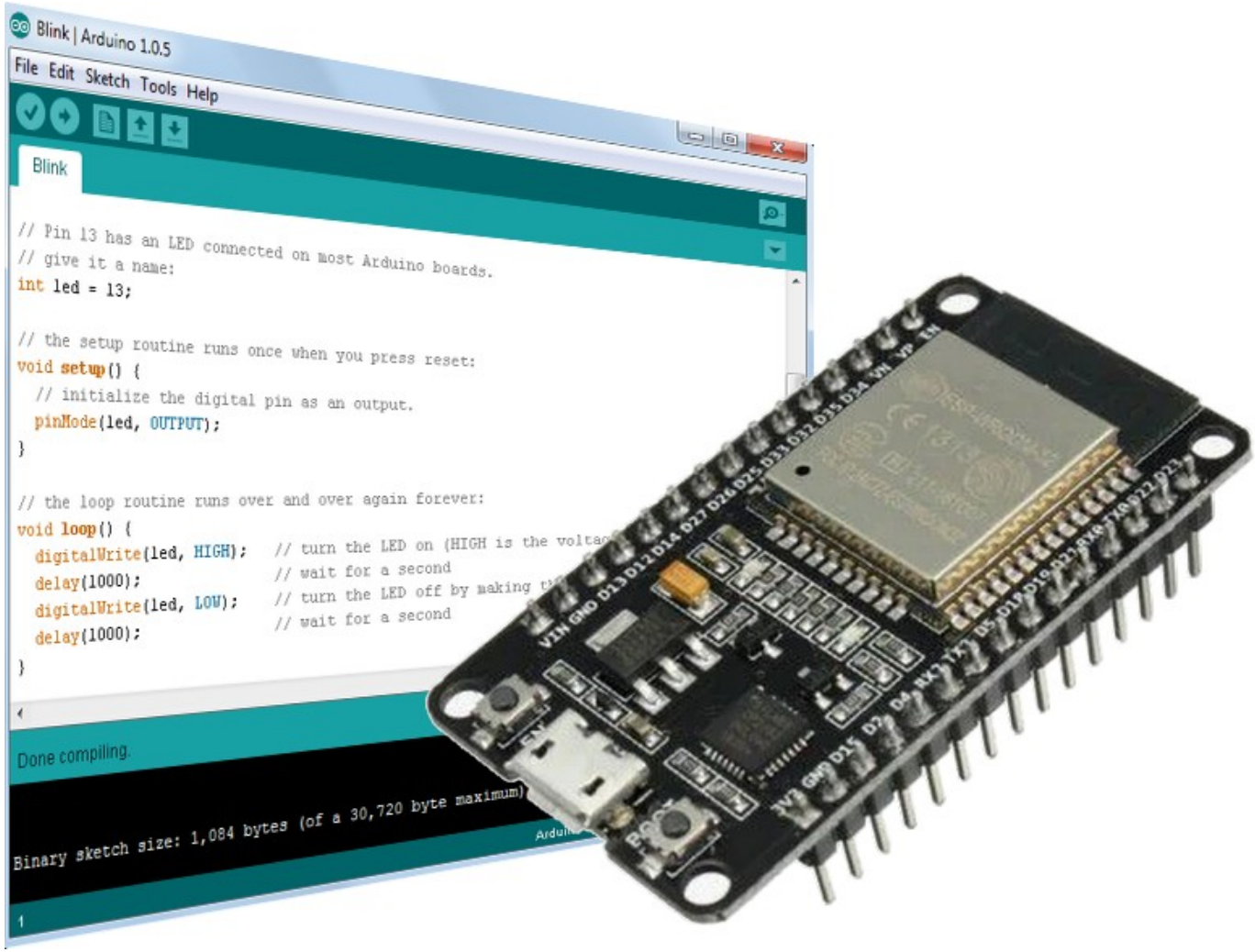
ESP32 mikrovezérlők programozása Arduino környezetben

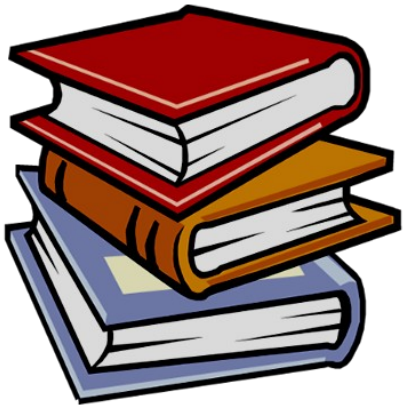


ARDUINO™

13. ESP32 Bluetooth kommunikáció – 2. rész

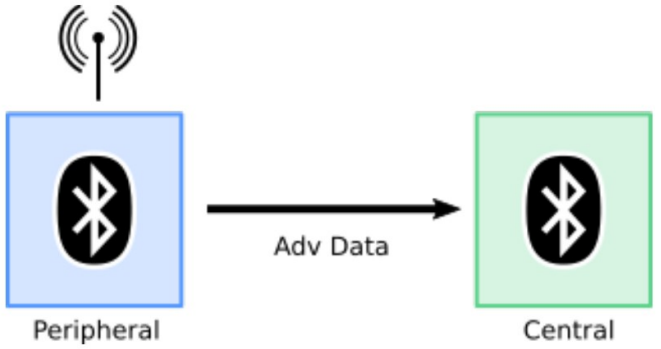
Felhasznált és ajánlott irodalom

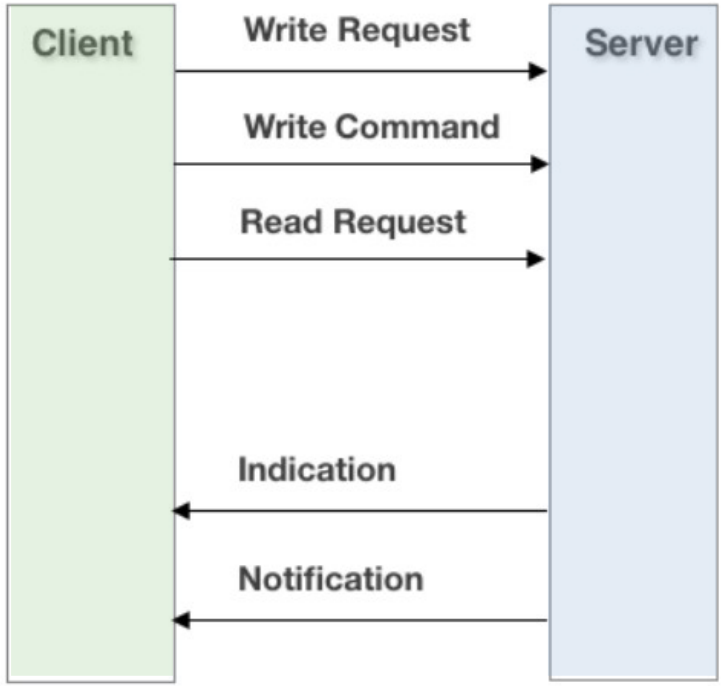
- **Leírások, dokumentáció**
  - Adafruit: Introduction to Bluetooth Low Energy
  - Neil Kolban: Kolban's book on ESP32
  - Neil Kolban: BLE C++ Guide.pdf
  - ESPRESSIF: ESP32 Arduino Core Documentation
- **Mintaprojektek**
- Rui Santos: **ESP32 BLE Server and Client**
- **Szoftver segédlet**
  - Nordic Semiconductor ASA: nRF Connect for Mobile
  - Fabio Durigon: OLED SSD1306 - SH1106 library
  - Adafruit: BME280 sensor library



Emlékeztető: BLE = Bluetooth Low energy

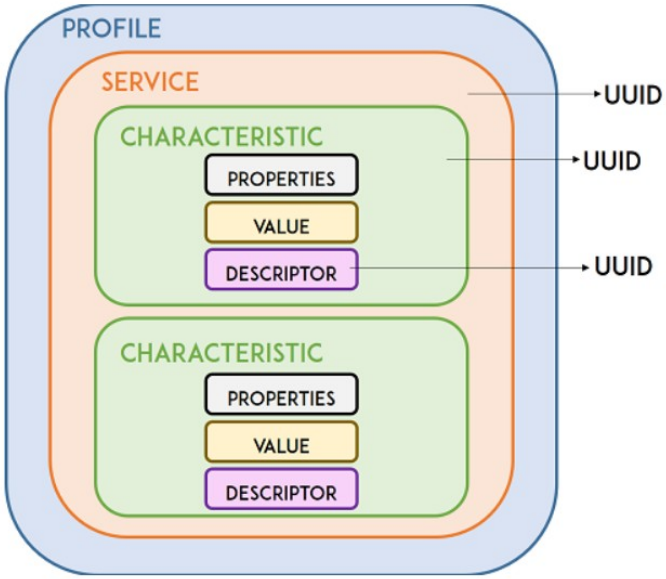
- A **Bluetooth** rövid hatótávolságú, adatcseréhez használt, vezetéknélküli kommunikációs szabvány, melynek energiatakarékos változata a **BLE**
- **GAP** – az *általános hozzáférési protokoll* a felderítési folyamatot, valamint az eszközök közötti kapcsolat kialakítását határozza meg
  - A perifériák hirdethetik magukat, illetve pásztázáskor válaszüzenetet küldhetnek
  - A központi eszköz felderíti a hálózatot, majd kliensként kapcsolódik a kiválasztott szerverhez
- **GATT** – az általános attribútum protokoll mechanizmust biztosít az adatok szabványos átadására, miután már létrejött a kapcsolat
- A *szerver* birtokolja az adatot, a *kliens* csak engedély alapján írhatja/olvashatja





Emlékeztető: Szolgáltatások és jellemzők

- A **GATT** tranzakciók az egymásba ágyazott profil, szolgáltatás és jellemző elemeken alapulnak
- **A profil** a szolgáltatások előre meghatározott gyűjteménye. (ld: GATT profilok listája)
- **A szolgáltatások** egy vagy több adattömböt, úgynevezett jellemzőket tartalmaznak
- **A jellemző** egyetlen adatelemet (attribútum) tartalmaz, melyhez tulajdonságok, és opcionálisan leírók tartoznak



- **A tulajdonság** szabja meg, hogy a *jellemzővel* milyen művelet végezhető
- **A leírók** olyan metaadatokat tartalmaznak, amelyek vagy kiegészítik a a *jellemzővel* kapcsolatos részleteket, vagy az adott *jellemző* viselkedését befolyásolják. Például az értesítési üzenetek be- és kikapcsolása egy speciális leíró, az **Ügyféljellemzők konfigurációs leírója** (0x2902) használatával történik

UUID (Universally Unique Identifier)

- Minden *szolgáltatásnak*, *jellemzőnek* és *leírónak* van egy egyedi 128 bites (16 bájt) UUID azonosítója (Universally Unique Identifier)
- Általában hexadecimálisan írják ezeket (1 bájt 2 hexadecimális számjegynek felel meg), 4-2-2-2-6 (byte) formátumban. Például:

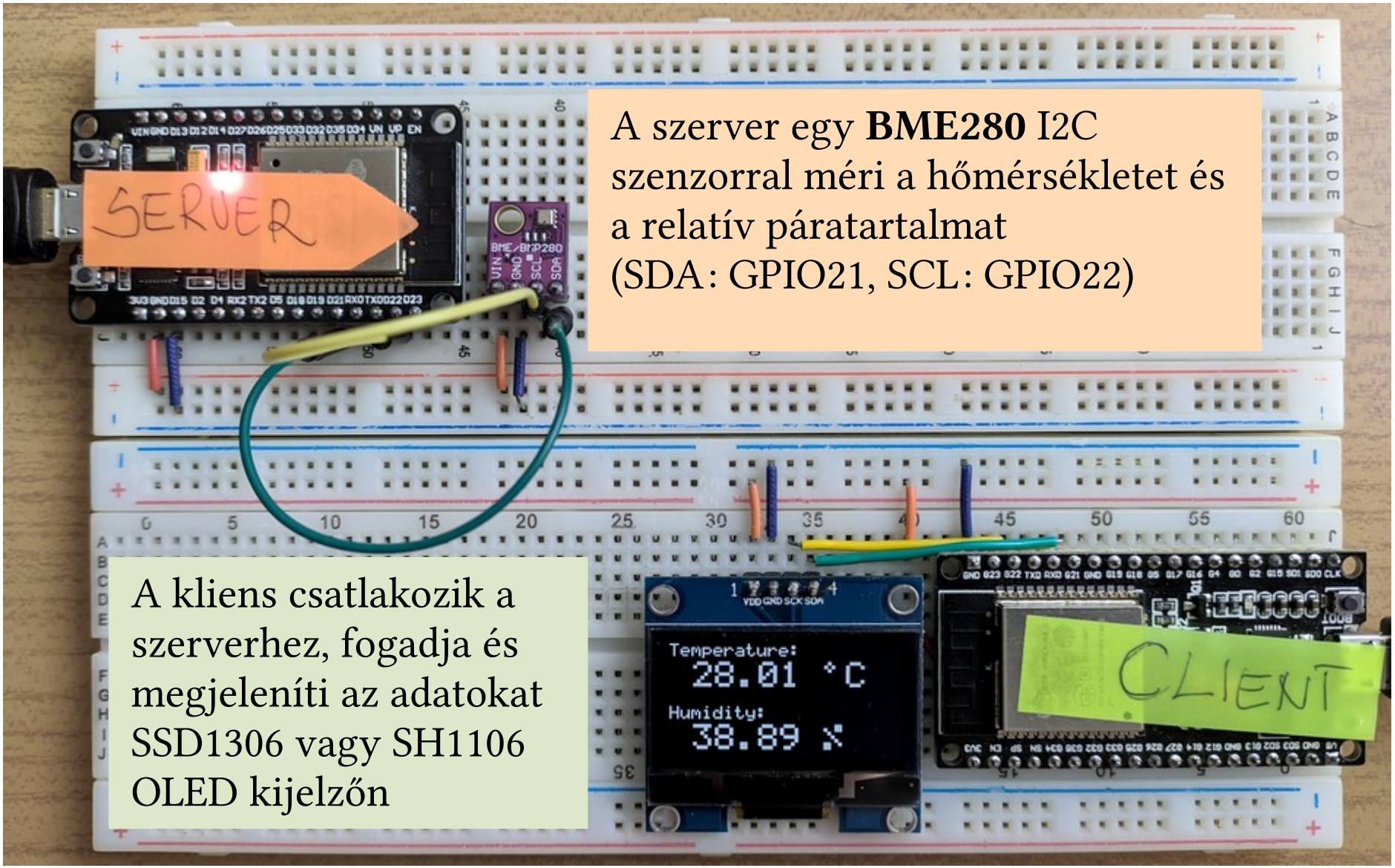
```
4fafc201-1fb5-459e-8fcc-c5c9c331914b
```

- Mivel a 16 bájt sok adat egy szolgáltatás azonosságának leírásához, megadhatjuk az UUID-t 16 bites vagy 32 bites rövidített formában is, de ezek a Bluetooth specifikáció tulajdonosai által tervezett szabványos szolgáltatások számára vannak fenntartva
- Amennyiben az alkalmazásunknak saját UUID-re van szüksége, előállíthatjuk ezt az UUID-generátor webhely használatával, mint például:

```
#define SERVICE_UUID  "ecc34a9c-e4f4-11ec-8fea-0242ac120002"
#define TEMPCC_UUID   "ecc34ca4-e4f4-11ec-8fea-0242ac120002"
#define PRESSURE_UUID "ecc34dda-e4f4-11ec-8fea-0242ac120002"
#define HUMIDITY_UUID "ecc34ef2-e4f4-11ec-8fea-0242ac120002"
```

ESP32 BLE szerver és kliens

A Random Nerd Tutorials **ESP32 BLE Server and Client** mintapéldáját adaptáljuk



BLE_BME280_server.ino – 5/1.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Alapértelmezetten Celsius fokokban dolgozunk
// Tedd kommentbe az alábbi sort, ha Fahrenheit skála kell helyette
#define temperatureCelsius

// A BLE szerver neve
#define bleServerName "ESP32_BME280"

Adafruit_BME280 bme; // BME280 I2C szenzor

float temp;
float tempF;
float hum;

// Timer variables
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;

bool deviceConnected = false;
```

BLE_BME280_server.ino – 5/2.

```
#define SERVICE_UUID "91bad492-b950-4226-aa2b-4ede9fa42f59"

// Temperature Characteristic and Descriptor
#ifdef temperatureCelsius
  BLECharacteristic bmeTemperatureCelsiusCharacteristics(
     "cba1d466-344c-4be3-ab3f-189f80dd7518", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmeTemperatureCelsiusDescriptor(BLEUUID((uint16_t)0x2902));
#else
  BLECharacteristic bmeTemperatureFahrenheitCharacteristics(
     "f78ebbff-c8b7-4107-93de-889a6a06d408", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmeTemperatureFahrenheitDescriptor(BLEUUID((uint16_t)0x2902));
#endif

// Humidity Characteristic and Descriptor
BLECharacteristic bmeHumidityCharacteristics(
     "ca73b3ba-39f6-4ab3-91ae-186dc9577d99", BLECharacteristic::PROPERTY_NOTIFY);
BLEDescriptor bmeHumidityDescriptor(BLEUUID((uint16_t)0x2903));

//Setup callbacks onConnect and onDisconnect
class MyServerCallbacks: public BLEServerCallbacks {
  void onConnect(BLEServer* pServer) {
    deviceConnected = true;
  };
  void onDisconnect(BLEServer* pServer) {
    deviceConnected = false;
  }
};
```

Szerver visszahívási függvények

BLE_BME280_server.ino – 5/3.

```
void setup() {
  Serial.begin(115200);     // Start serial communication
  // Init BME Sensor
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }

  BLEDevice::init(bleServerName);   // Create the BLE Device

  BLEServer *pServer = BLEDevice::createServer();   // Create the BLE Server
  pServer->setCallbacks(new MyServerCallbacks());

  // Create the BLE Service
  BLEService *bmeService = pServer->createService(SERVICE_UUID);

  // Create BLE Characteristics and Create a BLE Descriptor
  #ifdef temperatureCelsius
    bmeService->addCharacteristic(&bmeTemperatureCelsiusCharacteristics);
    bmeTemperatureCelsiusDescriptor.setValue("BME temperature Celsius");
bmeTemperatureCelsiusCharacteristics.addDescriptor(&bmeTemperatureCelsiusDescriptor);
  #else
    bmeService->addCharacteristic(&bmeTemperatureFahrenheitCharacteristics);
    bmeTemperatureFahrenheitDescriptor.setValue("BME temperature Fahrenheit");
bmeTemperatureFahrenheitCharacteristics.addDescriptor(&bmeTemperatureFahrenheitDescriptor);
  #endif
```

BLE_BME280_server.ino – 5/4.

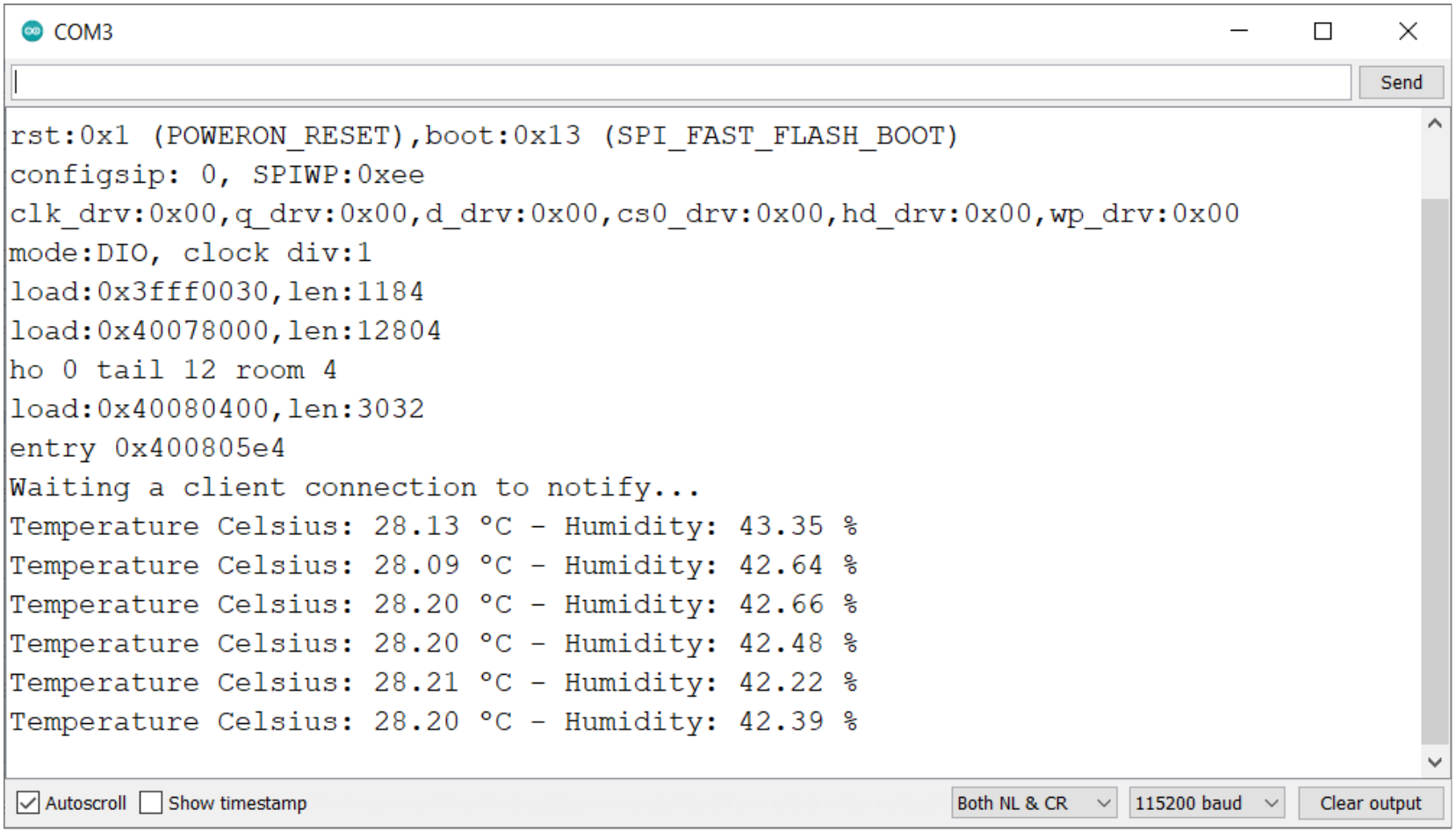
```
  // Humidity
  bmeService->addCharacteristic(&bmeHumidityCharacteristics);
  bmeHumidityDescriptor.setValue("BME humidity");
  bmeHumidityCharacteristics.addDescriptor(new BLE2902());

  BmeService->start();   // Start the service

  // Start advertising
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pServer->getAdvertising()->start();
  Serial.println("Waiting a client connection to notify...");
}

void loop() {
  if (deviceConnected) {
    if ((millis() - lastTime) > timerDelay) {
      // Read temperature as Celsius (the default)
      temp = bme.readTemperature();
      // Fahrenheit
      tempF = 1.8*temp +32;
      // Read humidity
      hum = bme.readHumidity();
```

Kiolvassuk a szenzort

BLE_BME280_server.ino – 5/5.

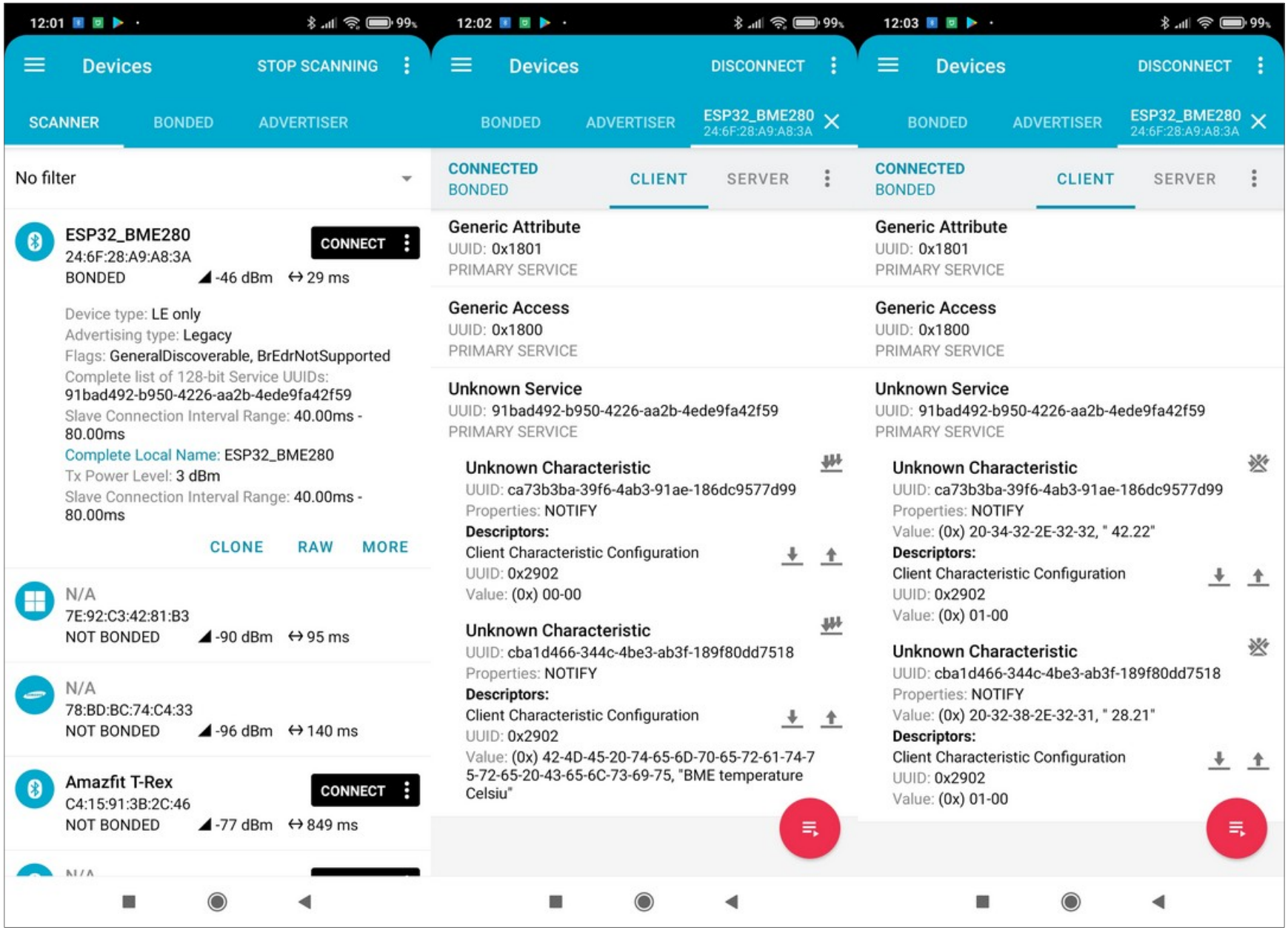
```
 //Notify temperature reading from BME sensor
      #ifdef temperatureCelsius
        static char temperatureCTemp[6];
        dtostrf(temp, 6, 2, temperatureCTemp);
        //Set temperature Characteristic value and notify connected client
        bmeTemperatureCelsiusCharacteristics.setValue(temperatureCTemp);
        bmeTemperatureCelsiusCharacteristics.notify();
        Serial.print("Temperature Celsius: ");
        Serial.print(temp);
        Serial.print(" ºC");
      #else
       ...
      #endif
      //Notify humidity reading from BME
      static char humidityTemp[6];
      dtostrf(hum, 6, 2, humidityTemp);
      //Set humidity Characteristic value and notify connected client
      bmeHumidityCharacteristics.setValue(humidityTemp);
      bmeHumidityCharacteristics.notify();
      Serial.print(" - Humidity: ");
      Serial.print(hum);
      Serial.println(" %");
      lastTime = millis();
} } }
```

BLE_BME280_server futási eredménye



- A program további működésének vizsgálatához telepítsük a mobiltelefonunkra a Nordic Semiconductor: nRF Connect for Mobile nevű alkalmazását !

BLE_BME280_server futási eredménye



BLE_SH1106_client.ino – 5/1.

```
#include "BLEDevice.h"
#include "oled.h"
#define temperatureCelsius
//BLE Server name (the other ESP32 name running the server sketch)
#define bleServerName "ESP32_BME280"
/* UUID's of the service, characteristic that we want to read*/
static BLEUUID bmeServiceUUID("91bad492-b950-4226-aa2b-4ede9fa42f59");
// BLE Characteristics
#ifdef temperatureCelsius
  //Temperature Celsius Characteristic
  static BLEUUID temperatureCharacteristicUUID("cba1d466-344c-4be3-ab3f-189f80dd7518");
#else
  //Temperature Fahrenheit Characteristic
  static BLEUUID temperatureCharacteristicUUID("f78ebbff-c8b7-4107-93de-889a6a06d408");
#endif
// Humidity Characteristic
static BLEUUID humidityCharacteristicUUID("ca73b3ba-39f6-4ab3-91ae-186dc9577d99");
//Flags stating if should begin connecting and if the connection is up
static boolean doConnect = false;
static boolean connected = false;

//Address of the peripheral device. Address will be found during scanning...
static BLEAddress *pServerAddress;
//Characteristicd that we want to read
static BLERemoteCharacteristic* temperatureCharacteristic;
static BLERemoteCharacteristic* humidityCharacteristic;
```

BLE_SH1106_client.ino – 5/2.

```
//Activate notify
const uint8_t notificationOn[] = {0x1, 0x0};
const uint8_t notificationOff[] = {0x0, 0x0};
//Declaration for an OLED display connected to I2C (SDA, SCL pins)
//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
//Variables to store temperature and humidity
char* temperatureChar;
char* humidityChar;

//Flags to check whether new temperature and humidity readings are available
boolean newTemperature = false;
boolean newHumidity = false;

//Callback function gets called, when another device's advertisement has been received
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
  void onResult(BLEAdvertisedDevice advertisedDevice) {
    if (advertisedDevice.getName() == bleServerName) { //Check if the name matches
      advertisedDevice.getScan()->stop(); //Scan stopped, found what we are looking for

      //Address of advertiser is the one we need
      pServerAddress = new BLEAddress(advertisedDevice.getAddress());
      doConnect = true; //Set indicator, stating that we are ready to connect
      Serial.println("Device found. Connecting!");
    }
  }
};
```

BLE_SH1106_client.ino – 5/3.

```
//Connect to the BLE Server that has the name, Service, and Characteristics
bool connectToServer(BLEAddress pAddress) {
   BLEClient* pClient = BLEDevice::createClient();
  pClient->connect(pAddress);    // Connect to the remove BLE Server.
  Serial.println(" - Connected to server");
  // Obtain a reference to the service we are after in the remote BLE server.
  BLERemoteService* pRemoteService = pClient->getService(bmeServiceUUID);
  if (pRemoteService == nullptr) {
    Serial.print("Failed to find our service UUID: ");
    Serial.println(bmeServiceUUID.toString().c_str());
    return (false);
  }
  // Obtain a reference to the characteristics in the service of the remote BLE server.
  temperatureCharacteristic = pRemoteService->getCharacteristic(
                                                    temperatureCharacteristicUUID);
  humidityCharacteristic = pRemoteService->getCharacteristic(humidityCharacteristicUUID);
  if (temperatureCharacteristic == nullptr || humidityCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID");
    return false;
  }
  Serial.println(" - Found our characteristics");

  //Assign callback functions for the Characteristics
  temperatureCharacteristic->registerForNotify(temperatureNotifyCallback);
  humidityCharacteristic->registerForNotify(humidityNotifyCallback);
  return true;
}
```

BLE_SH1106_client.ino – 5/4.

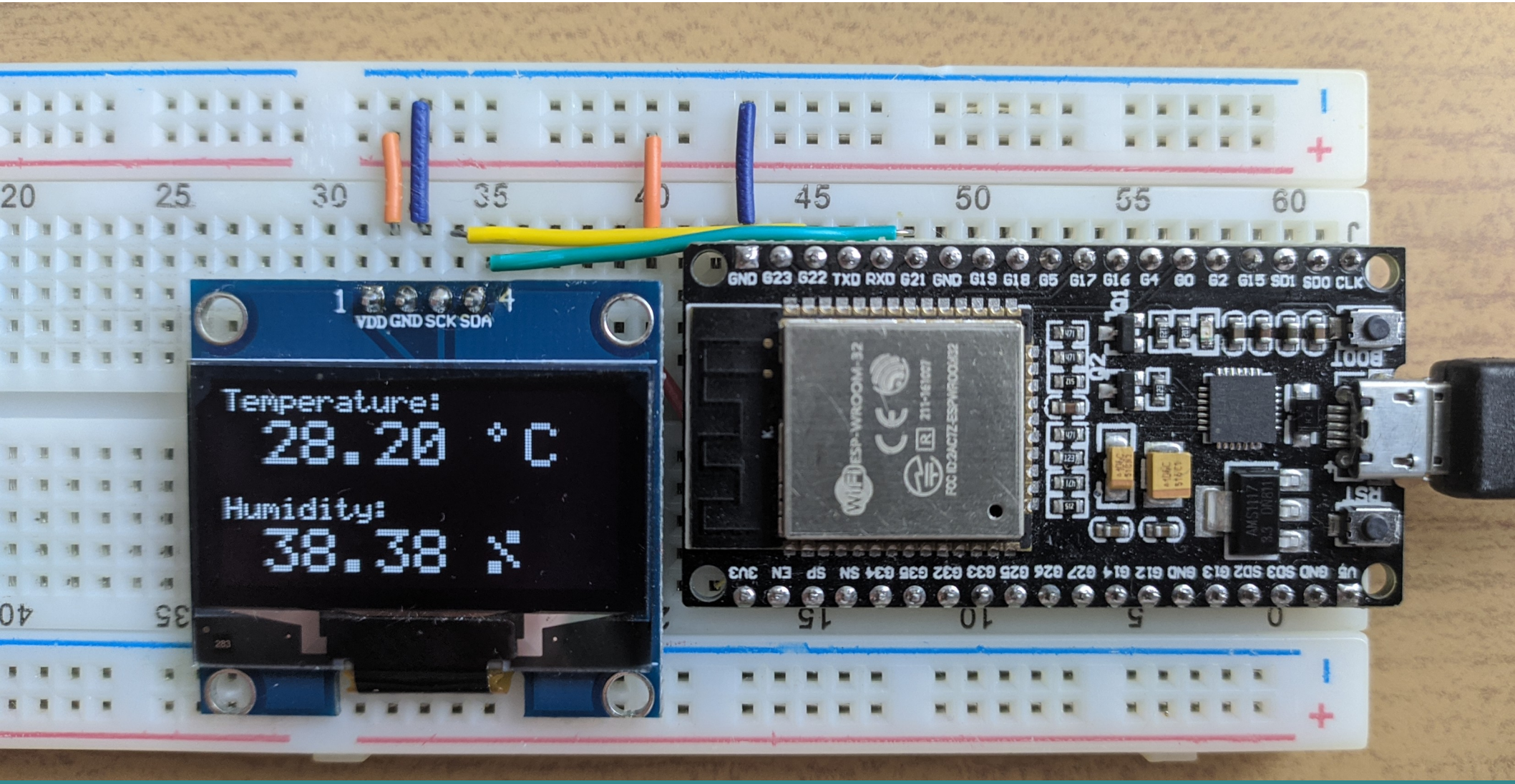
```
static void temperatureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                      uint8_t* pData, size_t length, bool isNotify) {
  //store temperature value
  temperatureChar = (char*)pData;
  newTemperature = true;
}

static void humidityNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                   uint8_t* pData, size_t length, bool isNotify) {
  //store humidity value
  humidityChar = (char*)pData;
  newHumidity = true;
}

void printReadings(){
  char msg[20];
  display.clear();
  display.draw_string(0, 0, "Temperature:");
  sprintf(msg,"%s %cC",temperatureChar,133);
  Serial.printf("Temperature: %s    ",msg);
  display.draw_string(0,10,msg, OLED::DOUBLE_SIZE);
  display.draw_string(0,35, "Humidity:");
  sprintf(msg,"%s %%",humidityChar,133);
  display.draw_string(0,45,msg, OLED::DOUBLE_SIZE);
  display.display();
  Serial.printf("Humidity: %s\r\n",msg);
}
```

BLE_SH1106_client.ino – 5/5.

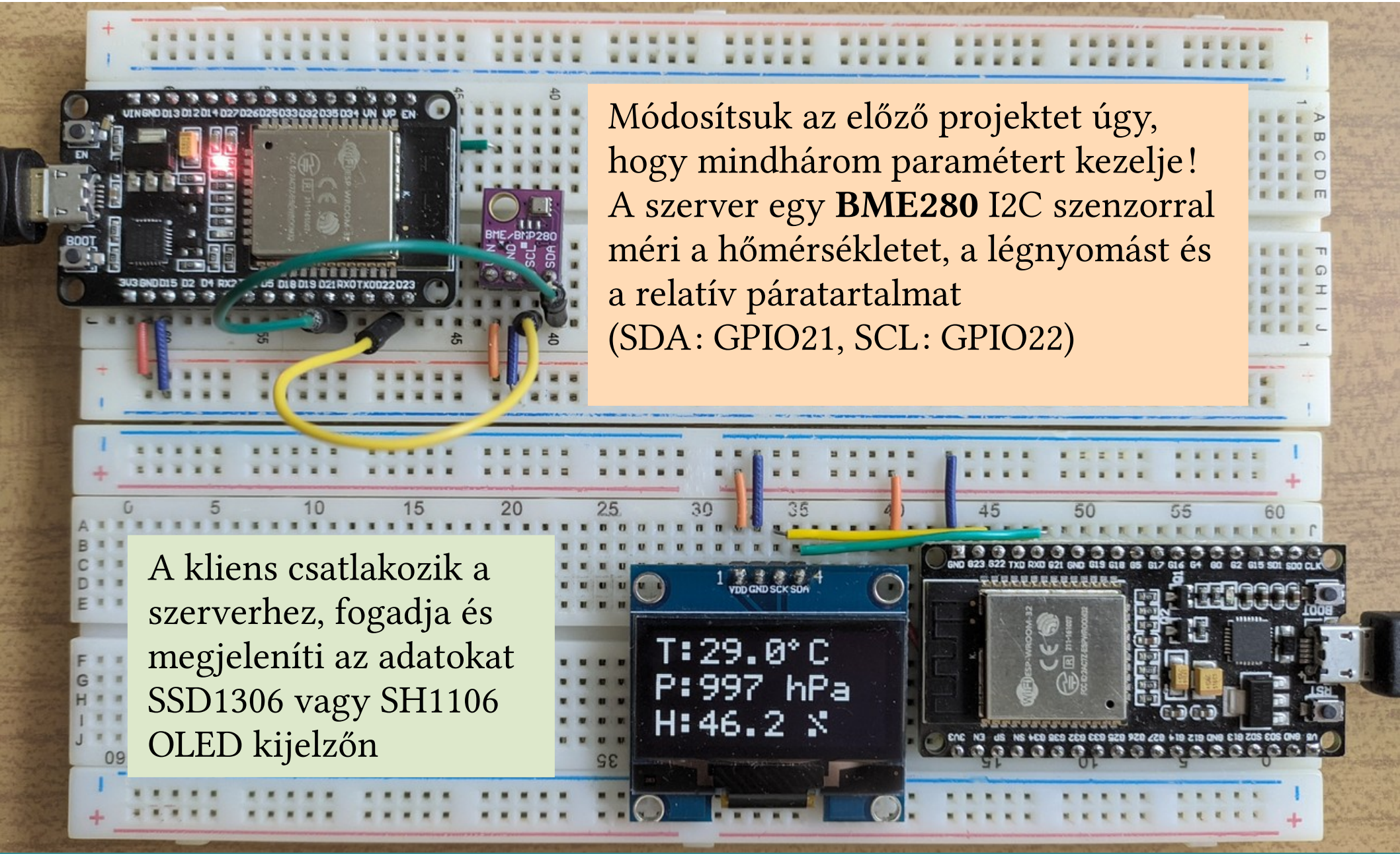
```
void setup() {
  display.begin();  //OLED display setup
  display.clear();
  display.draw_string(4, 12, "BLE Client", OLED::DOUBLE_SIZE);
  display.display();
  Serial.begin(115200);
  BLEDevice::init("");   //Init BLE device
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan→start(30);   // start the scan to run for 30 seconds.
}
void loop() {
  if (doConnect == true) {
    if (connectToServer(*pServerAddress)) {
      TemperatureCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->
                                    writeValue((uint8_t*)notificationOn, 2, true);
      HumidityCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->
                                   writeValue((uint8_t*)notificationOn, 2, true);
      connected = true;
    } else {Serial.println("Failed to connect to server; Restart your device."); }
    doConnect = false;
  }
  if (newTemperature && newHumidity){ PrintReadings();
     newTemperature = false; newHumidity = false; }
  delay(1000); // Delay a second between loops.
}
```

BLE_SH1106_client futási eredménye

- Az általunk használt OLED SSD1306-SH1106 programkönyvtár szoftveresen kezeli az I2C kapcsolatot, így mindegy, hogy a kijelzőt hová kötjük



ESP32 BLE szerver és kliens 3 paraméterrel



Módosítsuk az előző projektet úgy, hogy mindhárom paramétert kezelje!
A szerver egy **BME280** I2C szenzorral méri a hőmérsékletet, a légnyomást és a relatív páratartalmat
(SDA: GPIO21, SCL: GPIO22)

A kliens csatlakozik a szerverhez, fogadja és megjeleníti az adatokat SSD1306 vagy SH1106 OLED kijelzőn

BLE_BME280_server3.ino – 4/1.

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme;      // BME280 I2C szenzor
float tempC;
float pressure;
float humidity;
bool deviceConnected = false;
bool oldDeviceConnected = false;
char msg[10];
unsigned long lastTime = 0;
unsigned long nextTime = 5000;

// Generated by https://www.uuidgenerator.net/
#define SERVICE_UUID  "ecc34a9c-e4f4-11ec-8fea-0242ac120002"
#define TEMPCC_UUID   "ecc34ca4-e4f4-11ec-8fea-0242ac120002"
#define PRESSURE_UUID "ecc34dda-e4f4-11ec-8fea-0242ac120002"
#define HUMIDITY_UUID "ecc34ef2-e4f4-11ec-8fea-0242ac120002"

BLECharacteristic* pTempCharacteristic;
BLECharacteristic* pPressCharacteristic;
BLECharacteristic* pHumidityCharacteristic;
```

Globális változók, hogy értékadáskor (setValue) elérhessük!

BLE_BME280_server3.ino – 4/2.

```
void setup() {
  Serial.begin(115200);    // Start serial communication
  InitBME();               // Init BME Sensor
  BLEDevice::init("ESP32_BME280");                  // Create the BLE Device
  BLEServer *pServer = BLEDevice::createServer();  // Create the BLE Server
  pServer->setCallbacks(new MyServerCallbacks());
  BLEService *bmeService = pServer->createService(SERVICE_UUID); // Create BLE Service
  pTempCharacteristic = bmeService->createCharacteristic(
                          TEMPCC_UUID,
                          BLECharacteristic::PROPERTY_READ   |
                          BLECharacteristic::PROPERTY_NOTIFY
                        );
  pTempCharacteristic->addDescriptor(new BLE2902());
  pPressCharacteristic = bmeService->createCharacteristic(
                           PRESSURE_UUID,
                           BLECharacteristic::PROPERTY_READ   |
                           BLECharacteristic::PROPERTY_NOTIFY
                         );
  pPressCharacteristic->addDescriptor(new BLE2902());
  pHumidityCharacteristic = bmeService->createCharacteristic(
                              HUMIDITY_UUID,
                              BLECharacteristic::PROPERTY_READ   |
                              BLECharacteristic::PROPERTY_NOTIFY
                            );
  pHumidityCharacteristic->addDescriptor(new BLE2902());

  bmeService->start();   // Start the service
```

BLE_BME280_server3.ino – 4/3.

```
  //--- Start advertising ---
  BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
  pAdvertising->addServiceUUID(SERVICE_UUID);
  pAdvertising->setScanResponse(true);
  pAdvertising->setMinPreferred(0x06);  // that help with iPhone connections issue
  pAdvertising->setMaxPreferred(0x12);
  BLEDevice::startAdvertising();
  Serial.println("Waiting a client connection to notify...");
}

//Setup callbacks onConnect and onDisconnect
class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

void initBME() {
  if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
  }
}
```

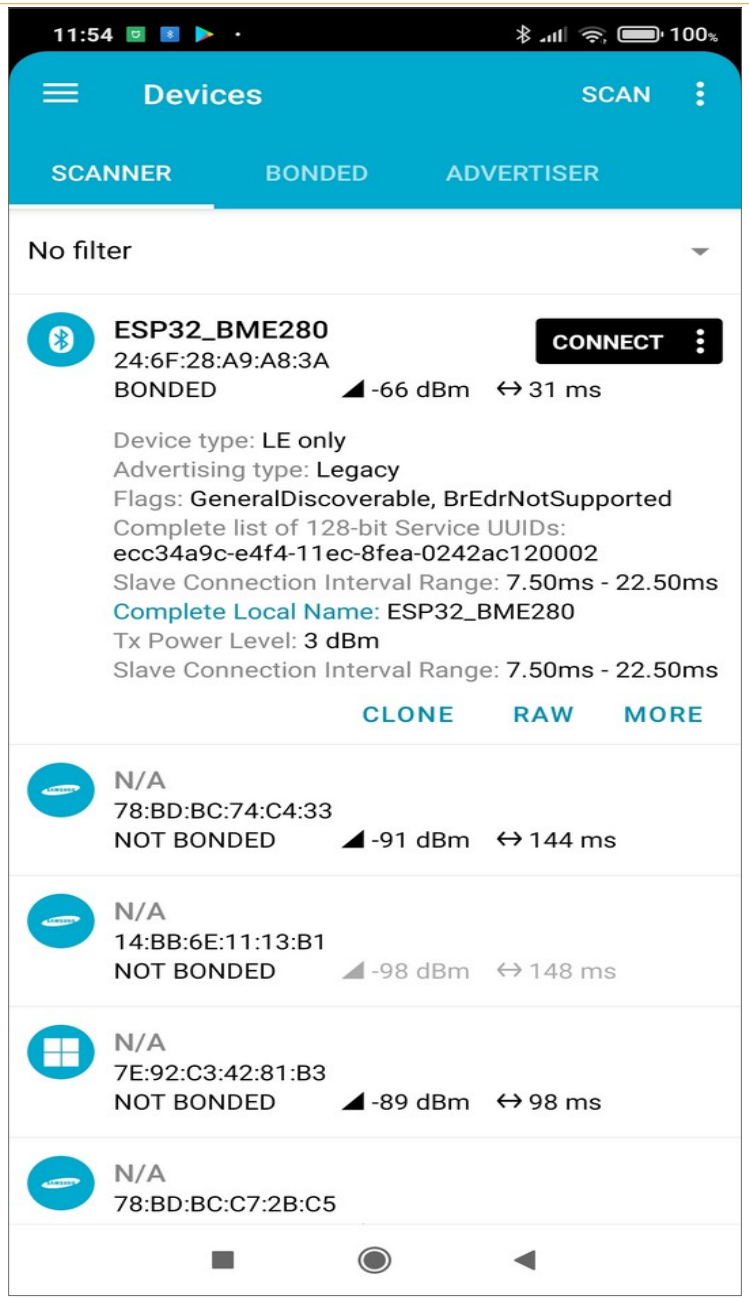
BLE_BME280_server3.ino – 4/4.

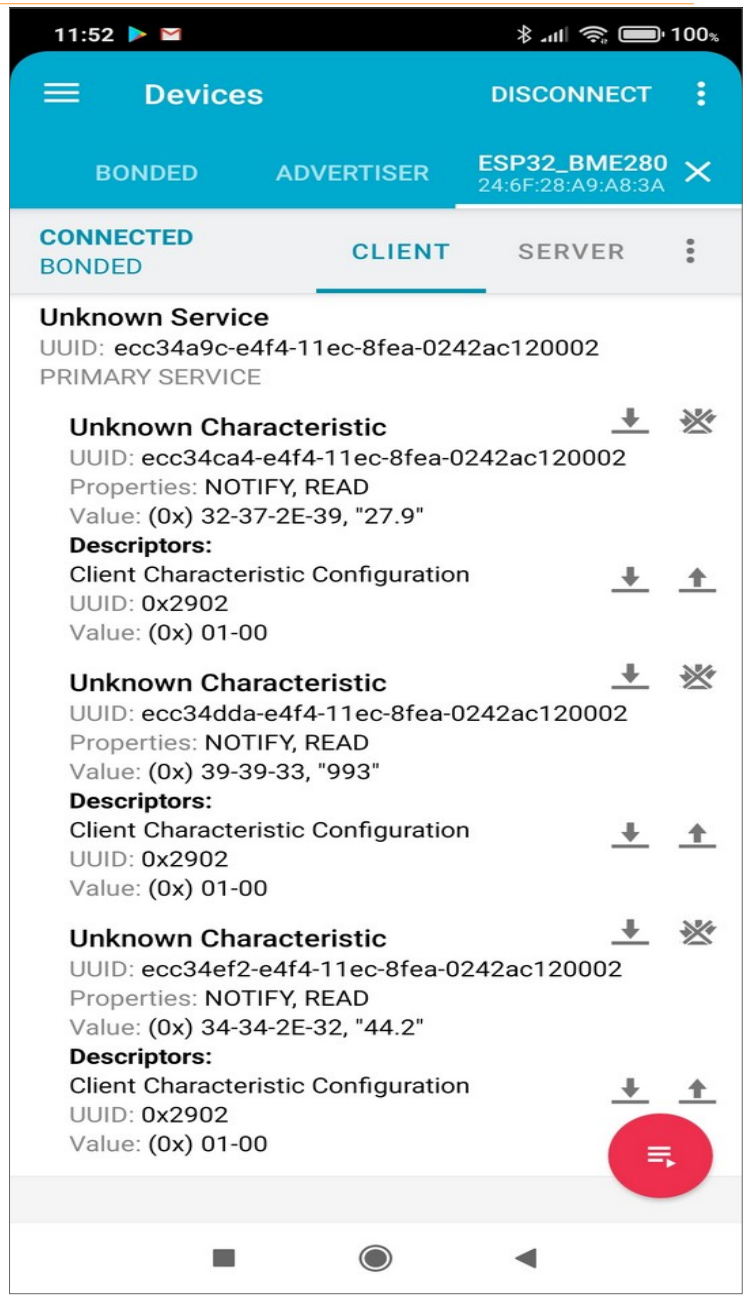
```
void loop() {
  if (deviceConnected) {
    if ((millis() - lastTime) > nextTime) {
      tempC = bme.readTemperature();
      pressure = bme.readPressure() / 100.0;
      humidity = bme.readHumidity();
      dtostrf(tempC, -6, 1, msg);
      //Set temperature Characteristic value and notify connected client
      pTempCharacteristic->setValue(msg);
      pTempCharacteristic->notify();
      Serial.print("Temperature: ");
      Serial.print(tempC); Serial.print(" °C");
      dtostrf(pressure, -4, 0, msg);
      //Set pressure Characteristic value and notify connected client
      pPressCharacteristic->setValue(msg);
      pPressCharacteristic->notify();
      Serial.print(" - Pressure: ");
      Serial.print(pressure); Serial.print(" hPa");
      dtostrf(humidity,-4, 1, msg);
      pHumidityCharacteristic->setValue(msg);
      pHumidityCharacteristic->notify();
      Serial.print(" - Humidity: ");
      Serial.print(humidity); Serial.println(" %");
      lastTime += nextTime;
    }
  }
}
```

BLE_BME280_server3 kipróbálása

nRF Connect for Mobile

alkalmazással





BLE_SH1106_client3.ino – 6/1.

```
#include "BLEDevice.h"
#include "oled.h"

static BLEUUID bmeServiceUUID("ecc34a9c-e4f4-11ec-8fea-0242ac120002");
static BLEUUID temperature_UUID("ecc34ca4-e4f4-11ec-8fea-0242ac120002");
static BLEUUID pressure_UUID("ecc34dda-e4f4-11ec-8fea-0242ac120002");
static BLEUUID humidity_UUID("ecc34ef2-e4f4-11ec-8fea-0242ac120002");

//Flags stating if should begin connecting and if the connection is up
static boolean doConnect = false;
static boolean connected = false;

//Address of the peripheral device. Address will be found during scanning...
static BLEAddress* pServerAddress = NULL;

//Characteristicd that we want to read
BLERemoteCharacteristic* pTempCharacteristic = NULL;
BLERemoteCharacteristic* pPressCharacteristic = NULL;
BLERemoteCharacteristic* pHumidityCharacteristic = NULL;

const uint8_t notificationOn[] = {0x1, 0x0};  // Activate notify
const uint8_t notificationOff[] = {0x0, 0x0}; // Deactivate notify

//Declaration for an OLED display connected to I2C (SDA, SCL pins)
//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
```

BLE_SH1106_client3.ino – 6/2.

```
//Variables to store temperature and humidity
char temperatureChar[6];
char pressureChar[6];
char humidityChar[6];

//Flags to check whether new temperature and humidity readings are available
boolean newTemperature = false;
boolean newPressure = false;
boolean newHumidity = false;

void setup() {
  //OLED display setup
  display.begin();
  display.clear();
  display.draw_string(4, 12, "BLE Client", OLED::DOUBLE_SIZE);
  display.display();
  Serial.begin(115200);   //Start serial communication
  Serial.println("Starting Arduino BLE Client application...");
  BLEDevice::init("");    //Init BLE device

  // Retrieve a Scanner and set the callback for new device detection
  BLEScan* pBLEScan = BLEDevice::getScan();
  pBLEScan->setAdvertisedDeviceCallbacks(new MyAdvertisedDeviceCallbacks());
  pBLEScan->setActiveScan(true);
  pBLEScan->start(30);
}
```

BLE_SH1106_client3.ino – 6/3.

```
void loop() {
  // If the flag "doConnect" is true then we have scanned for and found the desired
  // BLE Server with which we wish to connect.  Now we connect to it
  if (doConnect == true) {
    if (connectToServer(*pServerAddress)) {
      Serial.println("We are now connected to the BLE Server.");
      // Activate the Notify property of each Characteristic
      pTempCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->
                           writeValue((uint8_t*)notificationOn, 2, true);
      pPressCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->
                           writeValue((uint8_t*)notificationOn, 2, true);
      pHumidityCharacteristic->getDescriptor(BLEUUID((uint16_t)0x2902))->
                           writeValue((uint8_t*)notificationOn, 2, true);
      connected = true;
    } else {
      Serial.println("We have failed to connect to the server; Restart your device");
    }
    doConnect = false;
  }
  if (newTemperature && newPressure && newHumidity) {
    newTemperature = false;
    newPressure = false;
    newHumidity = false;
    printReadings();
  }
  delay(1000); // Delay a second between loops.
}
```

BLE_SH1106_client3.ino – 6/4.

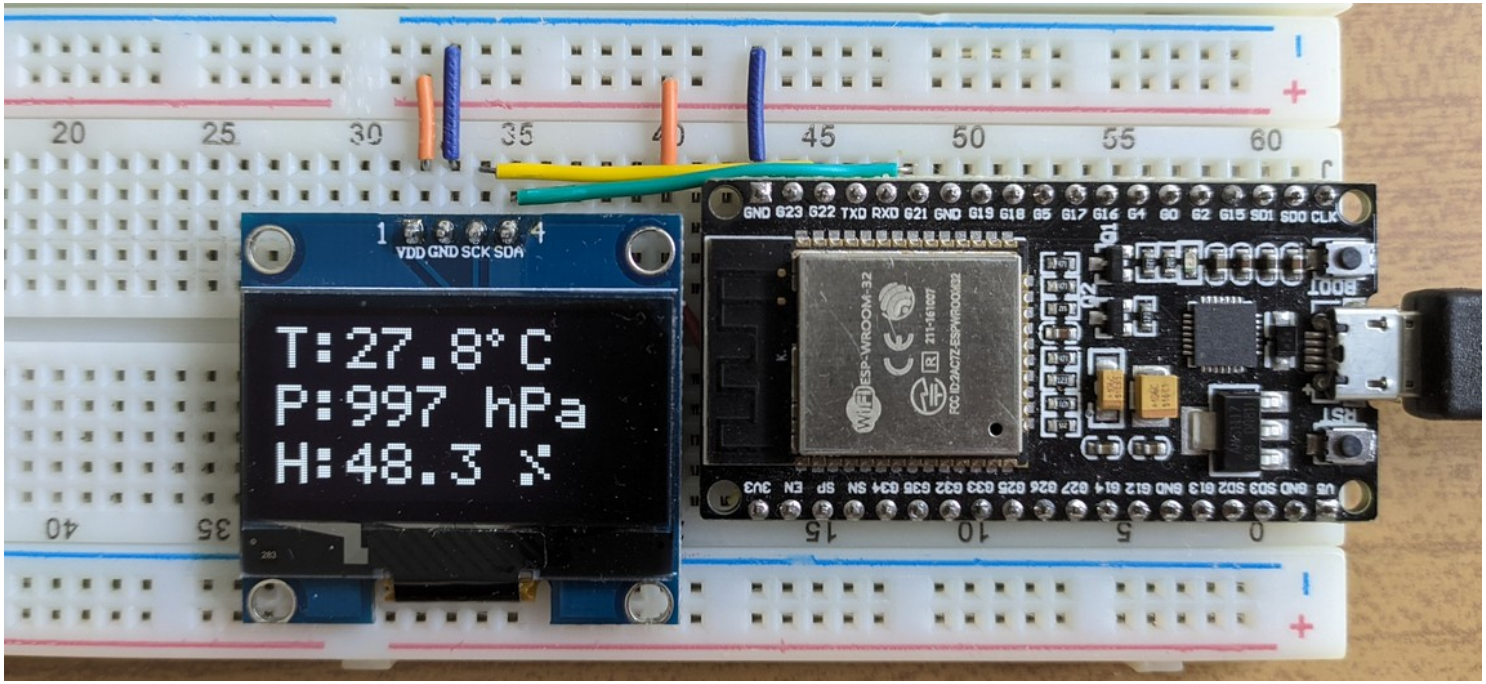
```
static void temperatureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                      uint8_t* pData, size_t length, bool isNotify) {
  memcpy(temperatureChar,(char*)pData,length);   //store temperature value
  newTemperature = true;
}
static void pressureNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                   uint8_t* pData, size_t length, bool isNotify) {
  memcpy(pressureChar,(char*)pData,length);     //store pressure value
  newPressure = true;
}
static void humidityNotifyCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic,
                                   uint8_t* pData, size_t length, bool isNotify) {
  memcpy(humidityChar,(char*)pData,length);    //store humidity value
  newHumidity = true;
}

void printReadings() {
  char msg[20];
  display.clear();
  sprintf(msg, "T:%s%cC", temperatureChar, 133);
  display.draw_string(0, 4, msg, OLED::DOUBLE_SIZE);
  sprintf(msg, "P:%s hPa", pressureChar);
  display.draw_string(0, 24, msg, OLED::DOUBLE_SIZE);
  sprintf(msg, "H:%s %%", humidityChar);
  display.draw_string(0, 44, msg, OLED::DOUBLE_SIZE);
  display.display();
}
```

BLE_SH1106_client3.ino – 6/5.

```
//Connect to the BLE Server that has the name, Service, and Characteristics
bool connectToServer(BLEAddress pAddress) {
  BLEClient* pClient = BLEDevice::createClient();
  pClient->connect(pAddress);   // Connect to the remove BLE Server.
  Serial.println(" - Connected to server");
  // Obtain a reference to the service we are after in the remote BLE server.
  BLERemoteService* pRemoteService = pClient->getService(bmeServiceUUID);
  if (pRemoteService == nullptr) {
    Serial.print("Failed to find our service UUID: ");
    Serial.println(bmeServiceUUID.toString().c_str());
    return (false);
  }
  // Obtain a reference to the characteristics in the service of the remote BLE server.
  pTempCharacteristic = pRemoteService->getCharacteristic(temperature_UUID);
  pPressCharacteristic = pRemoteService->getCharacteristic(pressure_UUID);
  pHumidityCharacteristic = pRemoteService->getCharacteristic(humidity_UUID);
  if (pTempCharacteristic == nullptr || pHumidityCharacteristic == nullptr ||
      pPressCharacteristic == nullptr) {
    Serial.print("Failed to find our characteristic UUID");
    return false;
  }
  Serial.println(" - Found our characteristics");
  //Assign callback functions for the Characteristics
  pTempCharacteristic->registerForNotify(temperatureNotifyCallback);
  pPressCharacteristic->registerForNotify(pressureNotifyCallback);
  pHumidityCharacteristic->registerForNotify(humidityNotifyCallback);
  return true;
}
```

BLE_SH1106_client3.ino – 6/6.

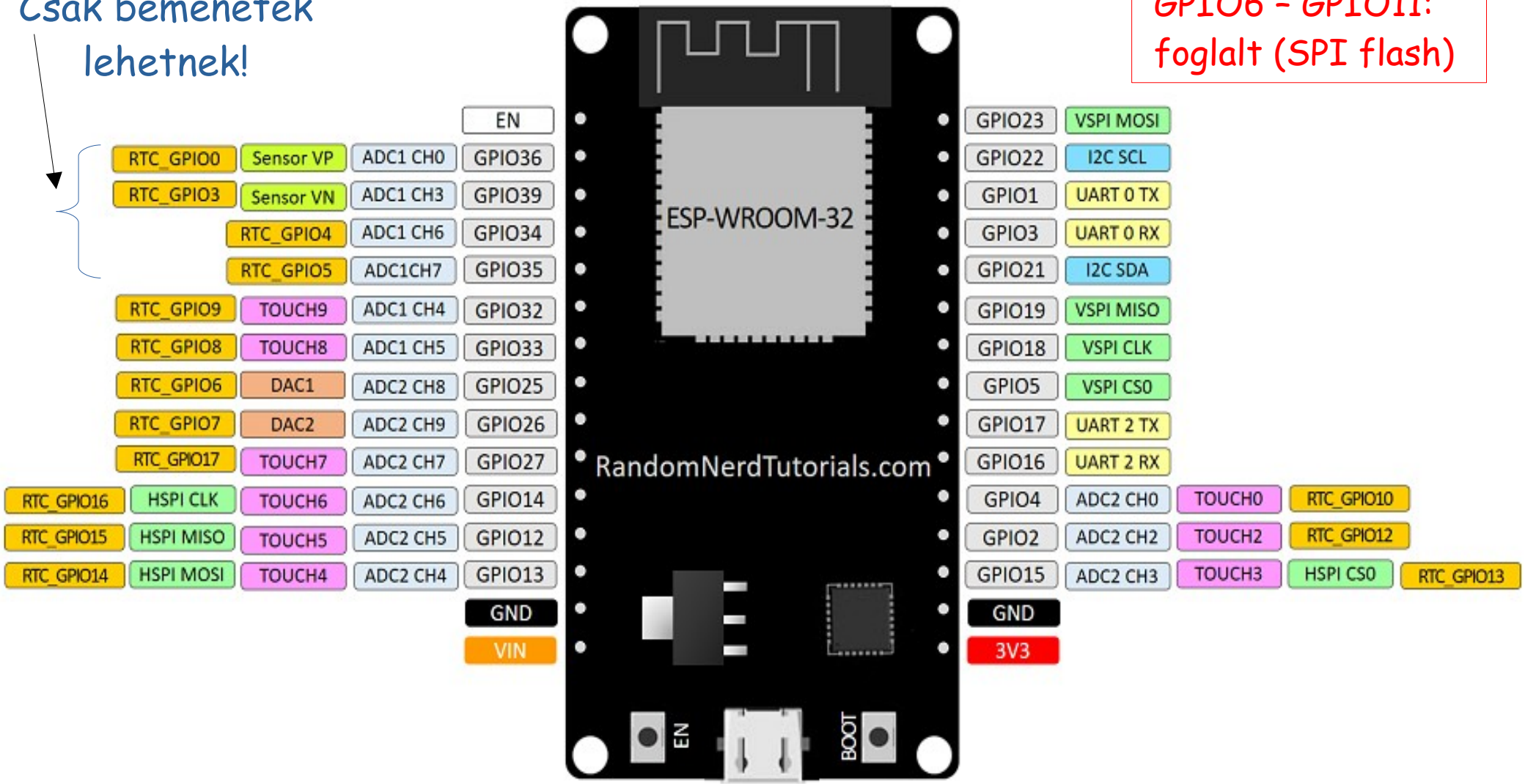
```
// Callback function that gets called, when another device's advertisement
// has been received
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
     void onResult(BLEAdvertisedDevice advertisedDevice) {
      if (advertisedDevice.getName() == "ESP32_BME280") { //Check if the name matches
        advertisedDevice.getScan()->stop(); //Scan stopped, found what we are looking for
        pServerAddress = new BLEAddress(advertisedDevice.getAddress()); //Address we need
        doConnect = true; //Set indicator, stating that we are ready to connect
        Serial.println("Device found. Connecting!");
      }
    }
}
```



A DOIT ESP32 Devkit-1 kártya kivezetései

Csak bemenetek lehetnek!

GPIO6 - GPIO11: foglalt (SPI flash)



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

Ellenállás színkódok