ESP32 mikrovezérlők programozása CircuitPython környezetben

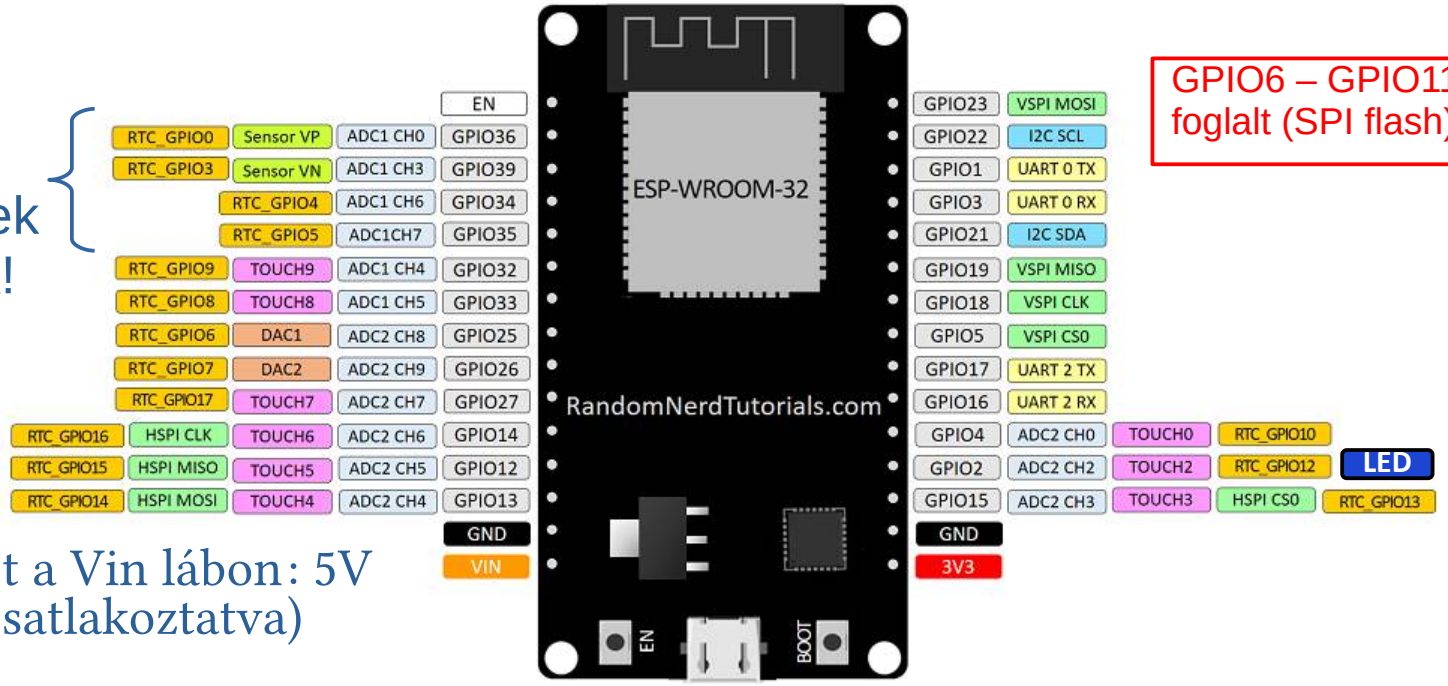
2. Hálózatkezelés a WiFi modullal

Felhasznált és ajánlott irodalom

- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- **CircuitPython:**
  - Adafruit: https://circuitpython.org/downloads
  - Learn Adafruit: Welcome to CircuitPython
  - Learn Adafruit: CircuitPython Essentials
  - Adafruit: Adafruit CircuitPython API Reference
  - Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle
  - Carter Nelson: CircuitPython on ESP32 Quick Start
  - Anne Barela: Networking in CircuitPython
- **Online eszközök és támogatás:**
  - Adafruit: Adafruit Web Serial ESPTool
  - Adafruit: CircuitPython Code Editor

Az ESP32 Devkit-1 (DOIT) kártya kivezetései

- A Doit **ESP32 Devkit-1** kártya egy **ESP WROOM-32** modult (ESP32 + 4MB flash) és az alapkártyán egy **CP2102** USB-UART átalakítót, egy 3,3 V-os stabilizátort, egy **Reset** és egy **Boot** nyomógombot tartalmaz



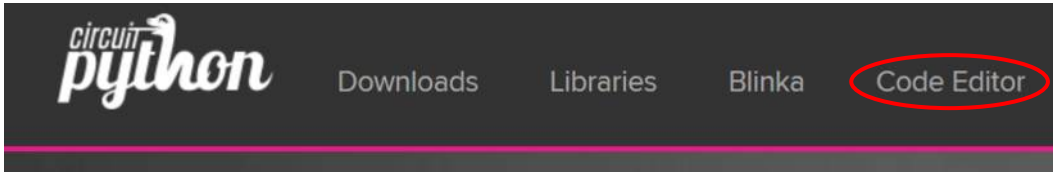
- Tápellátás bemenet a Vin lábon: 5V (ha nincs USB-re csatlakoztatva)

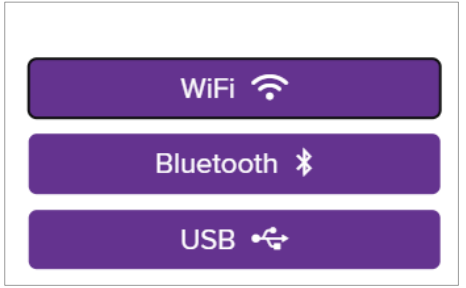
Forrás:
randomnerdtutorials.com/getting-started-with-esp32/

Pin 30 version

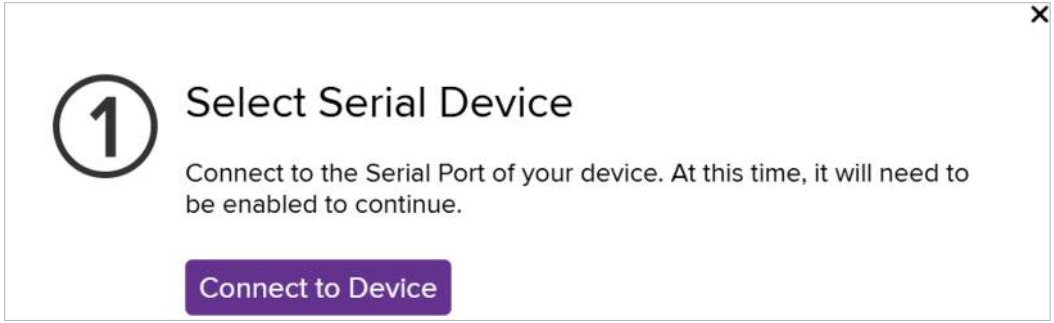
Webes munkakörnyezet

- A **circuitpython.org** honlapon a **Code Editor** gombra kattintva elindíthatjuk a webes munkakörnyezetet





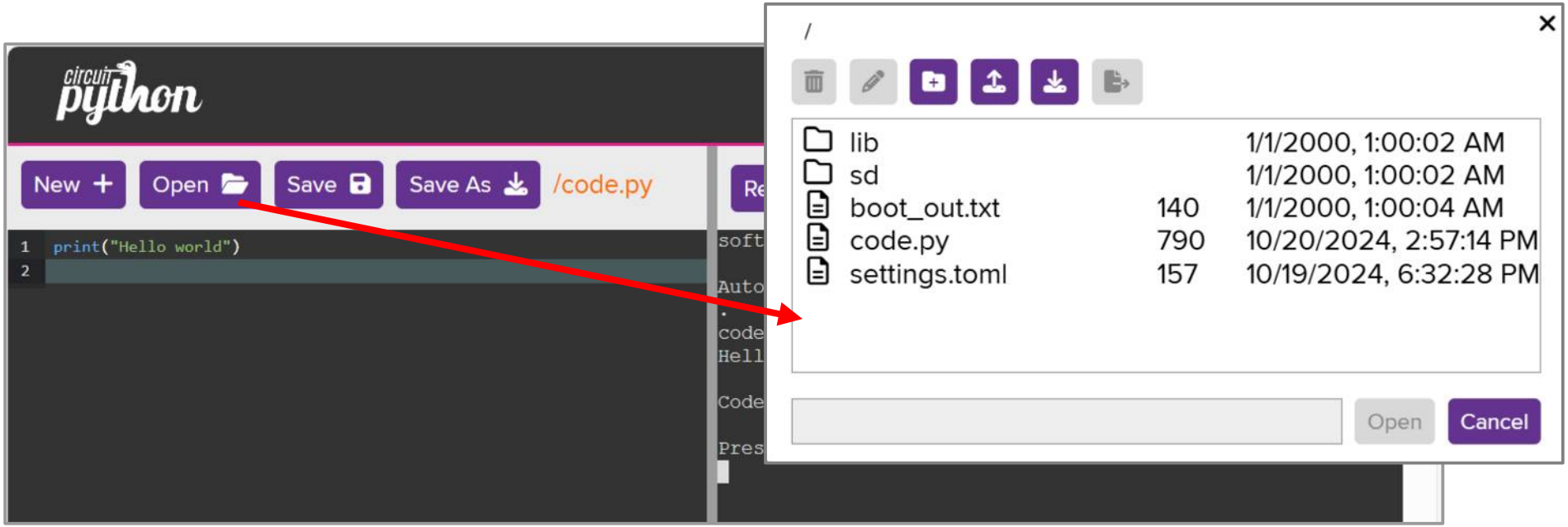
- A kapcsolódáshoz az USB opciót kell választanunk
- A **Connect to Device** gombra kattintva a szokásos módon ki kell választani a **Web Serial** kapcsolat számára a kártyához tartozó soros portot



- A kék **Connect** gombra kattintva megnyílik a kódszerkesztő és fájlkezelő ablak (USB port híján nincs CIRCUITPY: meghajtó!!!)

Fájlkezelés a webes környezetben

- Az Adafruit CircuitPython Webes környezetben a kártyához kapcsolódunk, majd az **Open** gombra kattintva előhívjuk a fájlkezelő ablakot
- A kezelő gombok: Delete, Rename, Create, Upload, Download, ill. Open
- A műveletek fájlokra, illetve mappákra is vonatkozhatnak



Hálózatkezelés CircuitPythonnal

- Bár a CircuitPythonnal kompatibilis kártyák száma folyamatosan nő, a hálózatkezelésre képes hardverek támogatása csak néhány gyártóra szorítkozik:
  - Expressif ESP32
  - Raspberry Pi Pico W

  } **wifi** modul
  - Adafruit Airlift (ESP32/SPI) — **adafruit_esp32spi** modul
  - Wiznet W5000, W5500 — **adafruit_wiznet5k** modul
- A mai előadásban csak az ESP32 beépített WiFi moduljának kezelésével foglalkozunk, amelynek az elsődleges kezelője a CircuitPython firmware-be beépített **wifi** könyvtár
- Az alkalmazásainkban használt különféle protokollok kezelésére azonban kiegészítő könyvtárakat is kell majd telepíteni

A settings.toml állomány

- A „környezeti változók” – különösen a hálózati kapcsolódásokkal kapcsolatos adatokat a **settings.toml** állományban célszerű tárolni
- Egy tipikus **settings.toml** beállítás látható az alábbi ábrán:

```
CIRCUITPY_WIFI_SSID = "HUAWEI-2.4G-9bQR"
CIRCUITPY_WIFI_PASSWORD = "mypassword"
CIRCUITPY_WEB_API_PASSWORD = "mypass"
THINGSPEAK_API_KEY = "**************"
```

WiFi paraméterek

Web Browser jelszó

Thingspeak write key

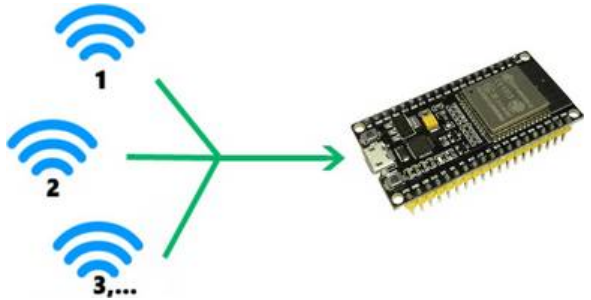
- A környezeti változók az **os.getenv("név")** metódussal vehetők elő:

```
import os, wifi

ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(ssid, password)
```

A wifi modul használata

- A **wifi** modul az alapvető hálózati kapcsolódásra összpontosít, mint például a hálózathoz való csatlakozás és IP-cím megszerzése

Mit csinálhatunk a wifi modullal, külső könyvtárak nélkül?



- **Wi-Fi hálózatok keresése**
  Lehetőséget nyújt a környező Wi-Fi hálózatok felderítésére és információik lekérdezésére
- **Csatlakozás Wi-Fi hálózathoz**
  Az alapvető Wi-Fi műveletek közé tartozik a vezeték nélküli hálózathoz való csatlakozás. Ehhez is a **wifi** modult használjuk
- **IP-címek, MAC-cím és egyéb információk lekérdezése**
  A **wifi** modulial különféle hálózati információkat is lekérdezhetünk, mint például az IP-cím, a MAC-cím vagy a csatlakozás állapota...

Wi-Fi hálózatok keresése

wifi_demo.py (részletek)

```
import wifi

print("Elérhető Wi-Fi hálózatok keresése...")
for network in wifi.radio.start_scanning_networks():
    print(f"SSID: {network.ssid}, Jelerősség: {network.rssi} dBm")

wifi.radio.stop_scanning_networks()  # Ne felejtsd el leállítani a keresést
```

- **start_scanning_networks():** Ez a funkció elindítja a hálózatkeresést, és visszaad egy listát az elérhető Wi-Fi hálózatokról
- **stop_scanning_networks():** Leállítja a hálózatkeresést, miután már megszerezted az információkat
- A **wifi.Network** osztály paraméterei:
  - **ssid** – a hálózat neve
  - **bssid** – az AP MAC címe
  - **country** – országkód
  - **authmode** – autentikációs mód
  - **rssi** – jelerősség

```
code.py output:
Elerheto Wi-Fi halozatok keresese...
SSID: HUAWEI-2.4G-9bQR, Jelerosseg: -73 dBm
SSID: DIRECT-UE-BRAVIA, Jelerosseg: -75 dBm
SSID: Telekom-1SrW9Z, Jelerosseg: -88 dBm
SSID: DIGI-8Usb, Jelerosseg: -90 dBm
SSID: T-E7CD88, Jelerosseg: -94 dBm
SSID: DIGI-vKHx, Jelerosseg: -92 dBm
SSID: DIGI_e5ac18, Jelerosseg: -83 dBm
SSID: DIGI-77fS, Jelerosseg: -95 dBm

Code done running.
```

Csatlakozás Wi-Fi hálózathoz

- A kapcsolódáshoz szükséges a hálózat neve (SSID) és a jelszó, mintapéldánkban ezeket a **settings.toml** fájlból olvassuk ki
- A sikeres kapcsolódás után lekérdezhetjük az ESP32 által beszerzett IP-címet

wifi_demo.py (részletek)

```
import wifi
import os
# Wi-Fi hálózat elérése környezeti változókból
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")

# Csatlakozás a Wi-Fi hálózathoz
print(f"Csatlakozás a {ssid} hálózathoz...")
wifi.radio.connect(ssid, password)
# Ha sikeres a csatlakozás, kiírja az IP címet
print("Csatlakozás sikeres!")
print("IP cím: ", wifi.radio.ipv4_address)
```

```
Csatlakozás a HUAWEI-2.4G-9bQR hálózathoz...
Csatlakozás sikeres!
IP cím:  192.168.100.33
```

Hálózati információk lekérése

wifi_radio.py

A legnagyobb jelerősségű hálózat lesz elől

```
import os, wifi
networks = []
for network in wifi.radio.start_scanning_networks():
    networks.append(network)
wifi.radio.stop_scanning_networks()
networks = sorted(networks, key=lambda net: net.rssi, reverse=True)
for network in networks:
    print("ssid:",network.ssid, "ch:", network.channel, "rssi:",network.rssi)

print("connecting...")
wifi.radio.connect(ssid=os.getenv('CIRCUITPY_WIFI_SSID'),
                   password=os.getenv('CIRCUITPY_WIFI_PASSWORD'))
print("my IP addr:", wifi.radio.ipv4_address)
print("MAC addr:", wifi.radio.mac_address)
print("gateway:", wifi.radio.ipv4_gateway)
print("mgateway ap:", wifi.radio.ipv4_gateway_ap)
print("subnet:", wifi.radio.ipv4_subnet)
print("DNS", wifi.radio.ipv4_dns)
print("hostname:", wifi.radio.hostname)
print("tx_power:", wifi.radio.tx_power)
```

Hálózati információk lekérése

```
code.py output:
ssid: HUAWEI-2.4G-9bQR ch: 6 rssi: -73
ssid: DIRECT-UE-BRAVIA ch: 1 rssi: -73
ssid: DIGI_e5ac18 ch: 5 rssi: -80
ssid: DIGI-8Usb ch: 11 rssi: -88
ssid: Telekom-1SrW9Z ch: 1 rssi: -90
ssid: Digi46 ch: 1 rssi: -91
ssid: Telekom-735285 ch: 9 rssi: -92
ssid: Vodafone-231C ch: 6 rssi: -93
ssid: T-E7CD88 ch: 11 rssi: -94
ssid: DIGI-vKHx ch: 2 rssi: -95
connecting...
my IP addr: 192.168.100.33
MAC addr: b'\x9c\x9en\xc3\x1ah'
gateway: 192.168.100.1
mgeteway ap: None
subnet: 255.255.255.0
DNS 192.168.100.1
hostname: doit_esp32_devkit_v1
tx_power: 11.0
```

Mit NEM tudunk csinálni külső könyvtárak nélkül?

- **HTTP kérések:** Az alapvető Wi-Fi modul nem tud HTTP kéréseket kezelni, így ha egy weboldallal vagy API-val szeretnénk kommunikálni (pl. adatokat küldeni vagy lekérni), szükségünk lesz az **adafruit_requests** könyvtárra, amely kezeli a **GET, POST, PUT, DELETE HTTP** kéréseket
- **SSL/TLS titkosított kapcsolatok:** Ha titkosított (HTTPS) kapcsolatot akarunk használni, szükségünk lesz egy SSL/TLS kezelésre képes könyvtárra, mint például az **ssl** vagy az **adafruit_requests.** Az alap **wifi** modul önmagában nem biztosít titkosított kapcsolatokat.
- **DNS kezelés:** Az alap **wifi.radio** modul ugyan kezeli az IP-címek hozzárendelését és a hálózathoz való csatlakozást, de a domain név feloldást (DNS-t) és az internetes kommunikáció más aspektusait csak külső könyvtárak (pl. **socket, requests**) segítségével oldhatjuk meg

Az adafruit_requests könyvtár

- Az **adafruit_requests** könyvtár a **CircuitPython**-ban a HTTP/HTTPS kérések kezelésére szolgál
- Főbb jellemzők:
  - **HTTP kérések:** A könyvtár támogatja a leggyakoribb HTTP kéréseket (GET, POST, PUT, DELETE)
  - **HTTPS támogatás:** Lehetővé teszi titkosított HTTPS kapcsolatok használatát SSL/TLS segítségével
  - **Egyszerű API:** Könnyen használható API-t kínál, amely egyszerűsíti a webes kommunikációt a **CircuitPython** projektekben
- Az **adafruit_requests.mpy** állományt az **Adafruit CircuitPython Library Bundle** csomaggal tölthetjük le, és kibontás után az eszközön a **/lib** mappába kell bemásolni
- Dokumentáció: docs.circuitpython.org/projects/requests/en/latest/index.html

Az adafruit_requests könyvtár használata

- Az alapvető használathoz kapcsolódás után létre kell hozni az ún. **socketpool**-t, és az alapértelmezett **ssl kontextust** az alábbiak szerint

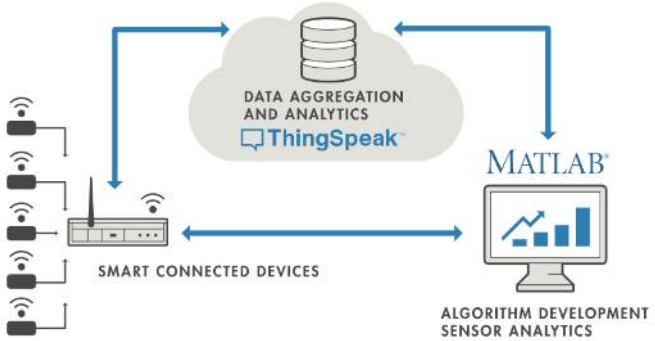
adafruit_requests_demo.py (részletek)

```
import os, wifi, ssl, socketpool
import adafruit_requests
ssid = os.getenv("CIRCUITPY_WIFI_SSID")
password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(ssid, password)
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())

response = requests.get("http://wifitest.adafruit.com/testwifi/index.html")
print(response.text)
```

code.py output:
This is a test of Adafruit WiFi!
If you can read this, its working :)

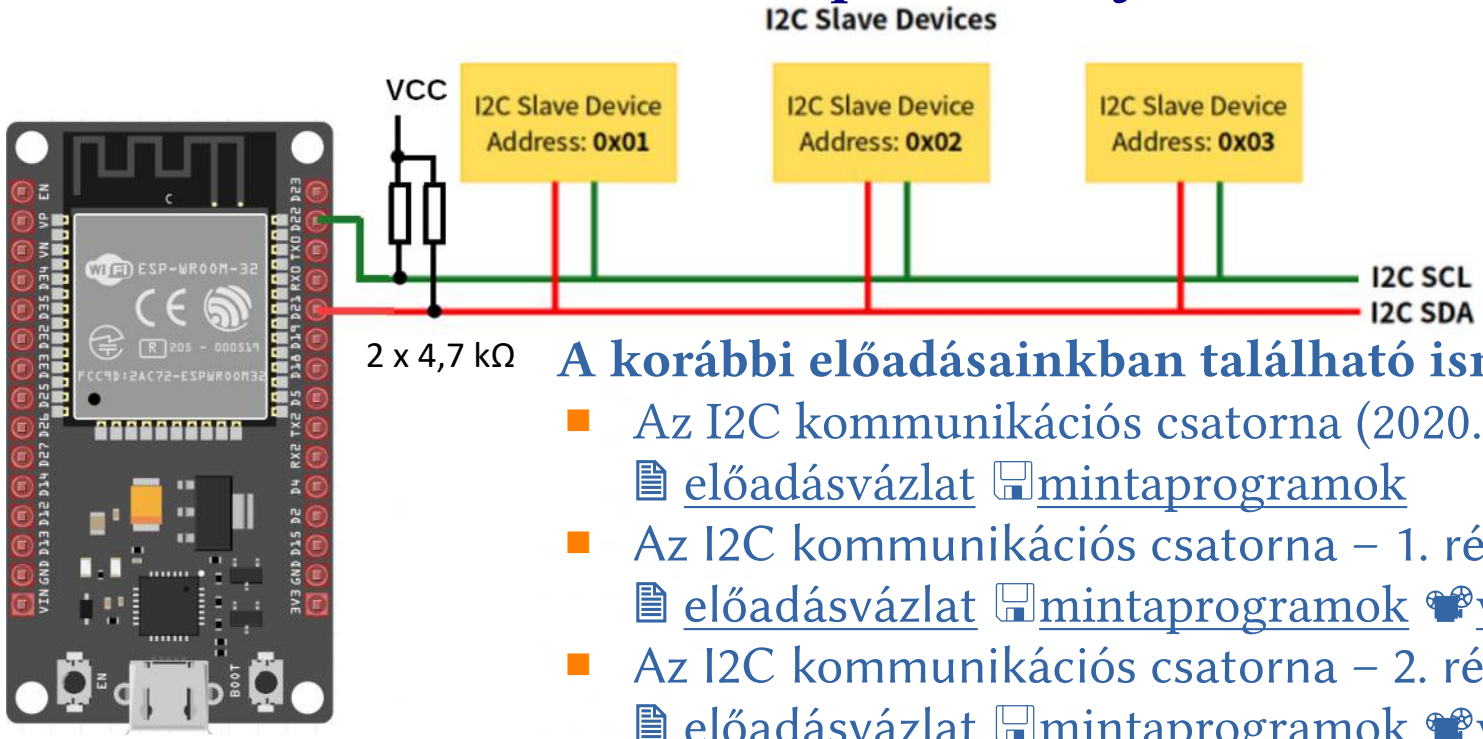
Adatküldés a Thingspeak szerverre

- A **POST request** használatát egy komplex példán keresztül mutatjuk be: egy **BME280** szenzor adatait küldjük el a **Thingspeak** szerverre (ehhez előzőleg regisztrálnunk kell hozzá egy adatcsatornát és be kell szerezni a **Write API Key** kulcsot)
- A **BME280** Bosch szenzor esetünkben az **I2C** buszon **0x76** címen érhető el és méri a hőmérséklet, a légnyomás és a relatív páratartalom értékét
- A **Thingspeak szerver** többféle adatküldési lehetőséget biztosít, most a **POST request** használatával, **JSON** formátumban küldjük az adatainkat a https://api.thingspeak.com/update.json URL-re



Az I2C kommunikációs csatorna

- Az **I2C** kommunikációs csatorna bit-soros, órajellel szinkronizált adatátvitel, ahol a címzés az első bájtban kiküldött címmel történik
- Részletes leírás: **”Az I2C-busz specifikációja és használata”**



A korábbi előadásainkban található ismertetők:

- Az I2C kommunikációs csatorna (2020. február 6.)
  előadásvázlat mintaprogramok
- Az I2C kommunikációs csatorna – 1. rész (2021. február 4.)
  előadásvázlat mintaprogramok video
- Az I2C kommunikációs csatorna – 2. rész (2021. február 18.)
  előadásvázlat mintaprogramok video

Az I2C osztály tagfüggvényei

- Az **I2C** osztályt a **busio** modul tartalmazza
- Leírása az Adafruit CircuitPython dokumentációban található

```
>>> import busio
>>> dir(busio)
['__class__', '__name__', 'I2C', 'OneWire', 'SPI', 'UART']
>>> help(busio.I2C)
object <class 'I2C'> is of type type
  deinit -- <function>            # Periféria elengedése
  __enter__ -- <function>         # Context Manager (pl. with) segítő
  __exit__ -- <function>          # Context Manager (pl. with) segítő
  scan -- <function>              # Eszközcímek felderítése a buszon
  try_lock -- <function>          # Csatorna lefoglalása
  unlock -- <function>            # Csatorna felszabadítása
  readfrom_into -- <function>     # Adatbeolvasás
  writeto -- <function>           # Adatkiírás
  writeto_then_readfrom -- <function> # Adatkiírás, majd beolvasás
>>>
```

i2c_scan.py

- Az alábbi kis program felderíti és kilistázza az I2C buszon található eszközök címeit (a program forrása: CircuitPython Essentials)

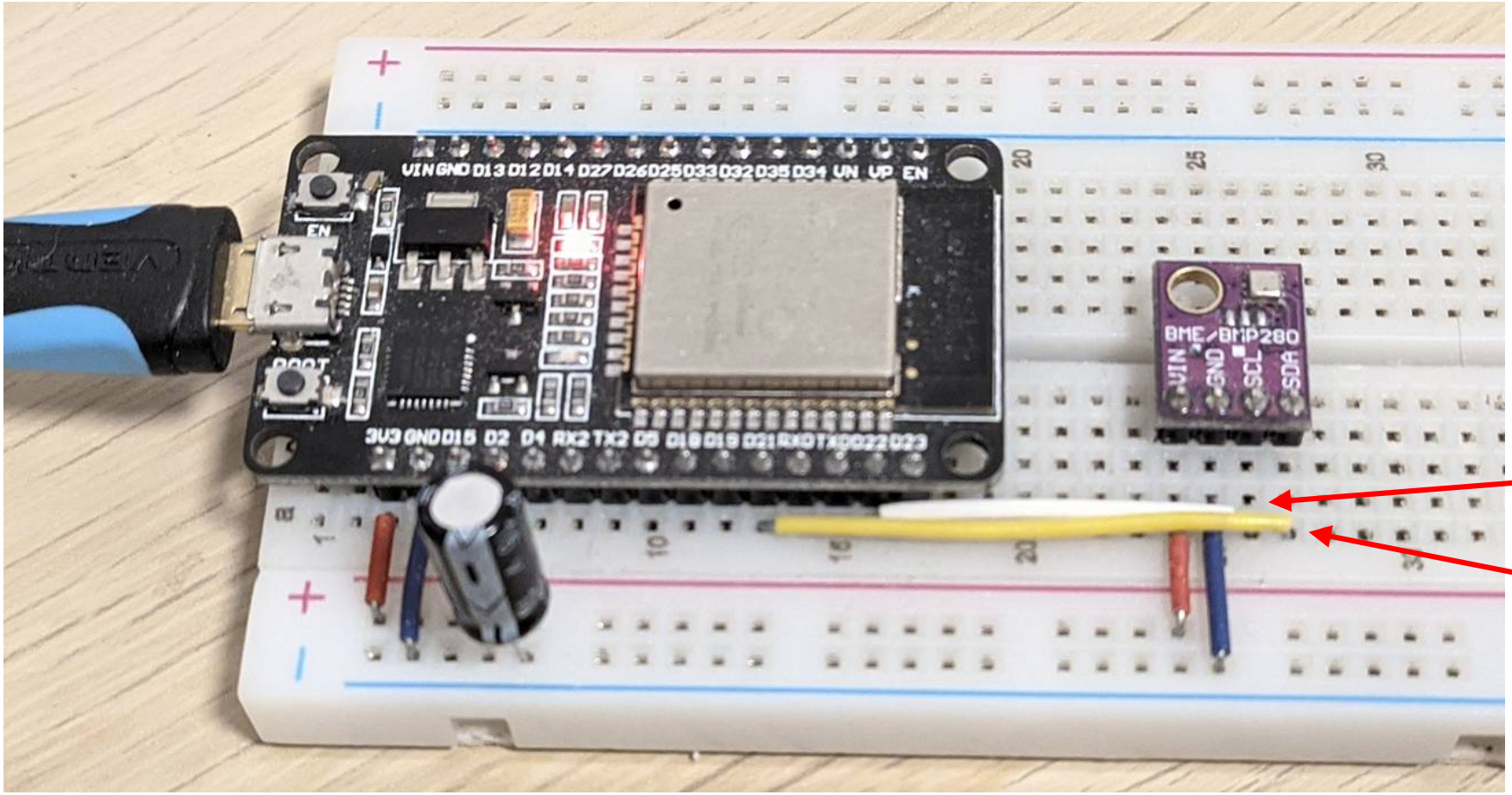
i2c_scan.py

```
import time, board

i2c = board.I2C()           # To use default I2C bus (most boards)
# To create I2C bus on specific pins
# import busio
# i2c = busio.I2C(board.D22, board.D21)

while not i2c.try_lock():
    pass

try:
    while True:
        print("I2C addresses found:",
            [hex(device_address) for device_address in i2c.scan()],
        )
        time.sleep(2)

finally:  # unlock the i2c bus when ctrl-c'ing out of the loop
    i2c.unlock()
```

BME280 címének felderítése

```
code.py output:
I2C addresses found: ['0x76']
I2C addresses found: ['0x76']
I2C addresses found: ['0x76']
I2C addresses found: ['0x76']
I2C addresses found: ['0x76']
```

A kapcsolási elrendezés



bme280_thingspeak.py – 3/1.

bme280_thingspeak.py

```
import os, time, board, busio, digitalio
import wifi, ssl, socketpool, adafruit_requests
from adafruit_bme280 import basic as adafruit_bme280

i2c = busio.I2C(scl=board.SCL, sda=board.SDA)  # Adjust pins as needed
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c, address=0x76)

THINGSPEAK_API_KEY = os.getenv("THINGSPEAK_API_KEY")  # Fetch the API key
THINGSPEAK_URL = "https://api.thingspeak.com/update.json"
altitude_m = 140    # Altitude for pressure correction (in meters)
L = 0.0065          # Temperature lapse rate in K/m
T0 = 288.15         # Standard temperature at sea level in K
g = 9.80665         # Gravitational acceleration in m/s^2
M = 0.0289644       # Molar mass of Earth's air in kg/mol
R = 8.31447         # Universal gas constant in J/(mol·K)

print("Connecting to WiFi...")
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"), os.getenv("CIRCUITPY_WIFI_PASSWORD"))
print("Connected to WiFi!")
pool = socketpool.SocketPool(wifi.radio)
ssl_context = ssl.create_default_context()  # Creating SSL context for secure connection
requests = adafruit_requests.Session(pool, ssl_context)
```

bme280_thingspeak.py – 3/2.

```python
def correct_pressure_to_sea_level(pressure, temperature, altitude):
    temperature_k = temperature + 273.15       # Convert temperature to Kelvin
    # Calculate sea level pressure
    sea_level_pressure = pressure * (1-(L*altitude)/temperature_k) ** (-g * M/(R * L))
    return sea_level_pressure

def send_data_to_thingspeak(temperature, pressure, humidity):
    json_payload = {
        "api_key": THINGSPEAK_API_KEY,
        "field1": temperature,
        "field2": pressure,
        "field3": humidity
    }

    try:
        print("Sending data to ThingSpeak...")
        response = requests.post(THINGSPEAK_URL, json=json_payload)
        print("Response: ", response.text)
    except Exception as e:
        print("Failed to send data: ", e)
```

A **Json** formátumú adatcsomag összeállítása

Adatküldés **POST request**-tel

bme280_thingspeak.py – 3/3.

```
while True:
  try:
    # Read BME280 sensor data
    temperature = bme280.temperature
    pressure = bme280.pressure
    humidity = bme280.humidity

    # Correct pressure to sea level
    corrected_pressure = correct_pressure_to_sea_level(pressure, temperature, altitude_m)

    print(f"Temperature: {temperature} C")
    print(f"Pressure (corrected to sea level): {corrected_pressure} hPa")
    print(f"Humidity: {humidity} %")

    # Send data to ThingSpeak
    send_data_to_thingspeak(temperature, corrected_pressure, humidity)

  except Exception as e:
    print("Error reading sensor data: ", e)
    # Wait for 15 seconds before sending the next batch of data
  time.sleep(15)
```

Channel Stats

Created: 3 years ago
Last entry: less than a minute ago
Entries: 137938

Field 1 Chart

Field 2 Chart

Field 3 Chart

Network Time Protocol (NTP) lekérések

- Az **adafruit_ntp** programkönyvtár segítségével NTP lekéréseket is végezhetünk
- `class` **`adafruit_ntp.NTP`**`(socketpool, *, server: str = '0.adafruit.pool.ntp.org', port: int = 123, tz_offset: float = 0, socket_timeout: int = 10, cache_seconds: int = 0)`
- Properties: **`datetime:`** `struct_time` - current time from NTP server

  `utc_ns:` `int` – current time from NTP server

ntp_simpletest.py

```
import os, time, socketpool, wifi
import adafruit_ntp

wifi_ssid = os.getenv("CIRCUITPY_WIFI_SSID")
wifi_password = os.getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(wifi_ssid, wifi_password)

pool = socketpool.SocketPool(wifi.radio)
ntp = adafruit_ntp.NTP(pool, tz_offset=0, cache_seconds=3600)

while True:
    print(ntp.datetime)
    time.sleep(1)
```

adafruit_ntp_demo.py futási eredmény

```
code.py output:
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=24, tm_wday=3, tm_yday=298, tm_isdst=-1)
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=25, tm_wday=3, tm_yday=298, tm_isdst=-1)
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=26, tm_wday=3, tm_yday=298, tm_isdst=-1)
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=27, tm_wday=3, tm_yday=298, tm_isdst=-1)
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=28, tm_wday=3, tm_yday=298, tm_isdst=-1)
struct_time(tm_year=2024, tm_mon=10, tm_mday=24, tm_hour=10, tm_min=27,
tm_sec=29, tm_wday=3, tm_yday=298, tm_isdst=-1)
```