

Portenta X8

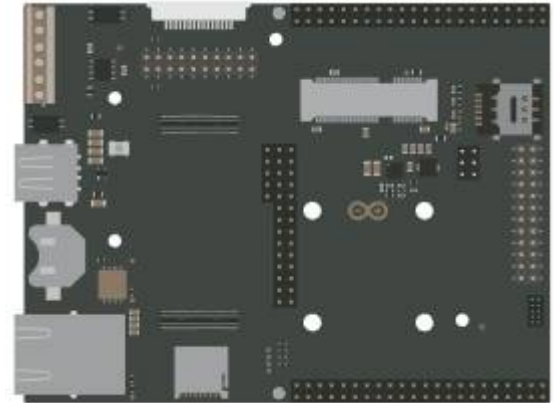
Arduino megközelítésben



Felhasznált és ajánlott irodalom

- ❖ Arduino: [Portenta X8 User Manual](#)
- ❖ Arduino: [Portenta X8 Pinout](#)
- ❖ Arduino: [Portenta X8 schematics](#)
- ❖ Arduino: [Portenta X8 tutorials](#)

- ❖ Arduino: [Arduino Portenta X8 Datasheet](#)
- ❖ Arduino: [Portenta Mid Carrier Datasheet](#)
- ❖ Arduino: [Portenta Mid Carrier Pinout](#)
- ❖ Arduino: [Portenta Mid Carrier Tutorials](#)



Az Arduino Portenta X8 kártya

- ❖ Az **Arduino Portenta X8** egy nagy teljesítményű rendszer-modul (SOM), amelyet ipari IoT-alkalmazások kiszolgálására terveztek.
- ❖ A lap az **NXP i.MX 8M Mini** processzort — amely beágyazott Linuxot futtat — ötvözi az **STM32H7** mikrokontrollerrel, hogy kihasználhassa az Arduino könyvtárak és fejlesztési lehetőségek előnyeit is
- ❖ A **Portenta X8** funkciói bővíthetők hordozópanellel és fedlapokkal, s ezek referenciaként is szolgálhatnak egyedi rendszerek fejlesztéséhez
- ❖ A **Portenta X8** Wi-Fi és Bluetooth kapcsolattal rendelkezik, emellett - hordozó lap esetén - egy Gigabit Ethernet interfész is rendelkezésre áll ami nagy sebességet és alacsony késleltetést biztosít a legigényesebb alkalmazásokhoz



MIMX8MM6CVTKZAA i.MX 8M Mini

Power switch

Displayport
konverter

Reset

Power
Management



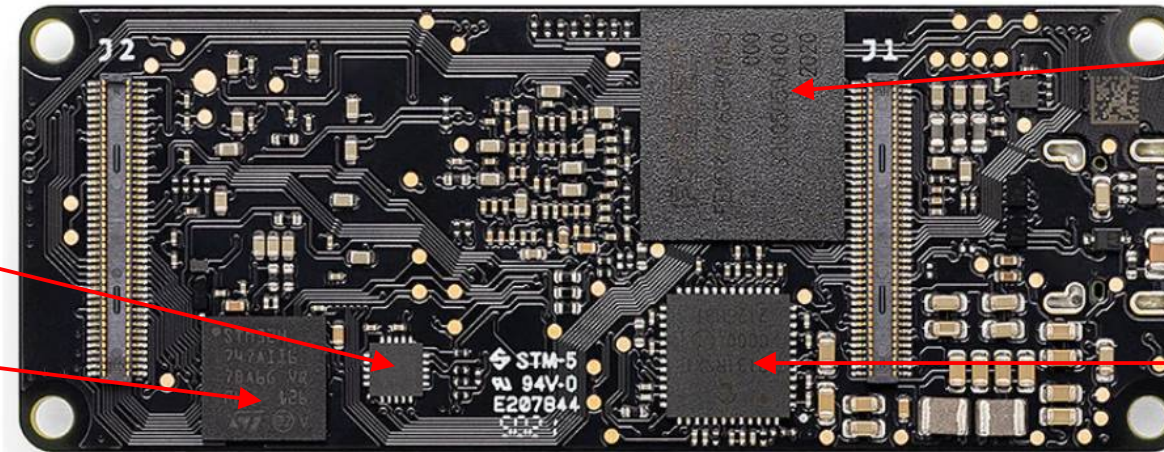
2GB LP-DDR4 DRAM

WLAN+Bluetooth

Antennacsatlakozó

IoT Secure
Element

STM32H747



16GB eMMC Flash

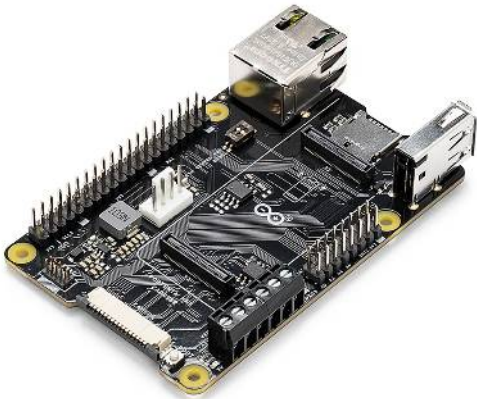
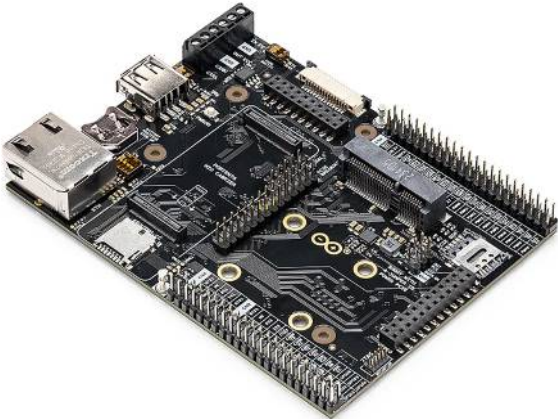
Gigabit Ethernet
Transceiver

Műszaki paraméterek

NXP i.MX 8M Mini Processor	4x Arm Cortex-A53 core up to 1.8 GHz 1x Arm Cortex-M4 core up to 400 MHz
STM32H747XI Microcontroller	1x Arm Cortex-M7 core up to 480 MHz, double precision FPU 1x Arm Cortex-M4 core up to 240 MHz with FPU
Onboard memory	2GB Low Power DDR4 DRAM 16GB Foresee eMMC Flash module
USB-C	High Speed USB DisplayPort output Host and Device operation Power Delivery support
WiFi/Bluetooth modul	Wi-Fi 802.11b/g/n 65 Mbps (2.4 GHz) Bluetooth 5.1 BR/EDR/LE
High Density connectors	PCI express, Ethernet interface, 2xUSB HS, 4xUART, 3xI2C, 2xSPI, 4xPWM, 1x SD card, MIPI DSI/CSI, 8xADC, 7xGPIO

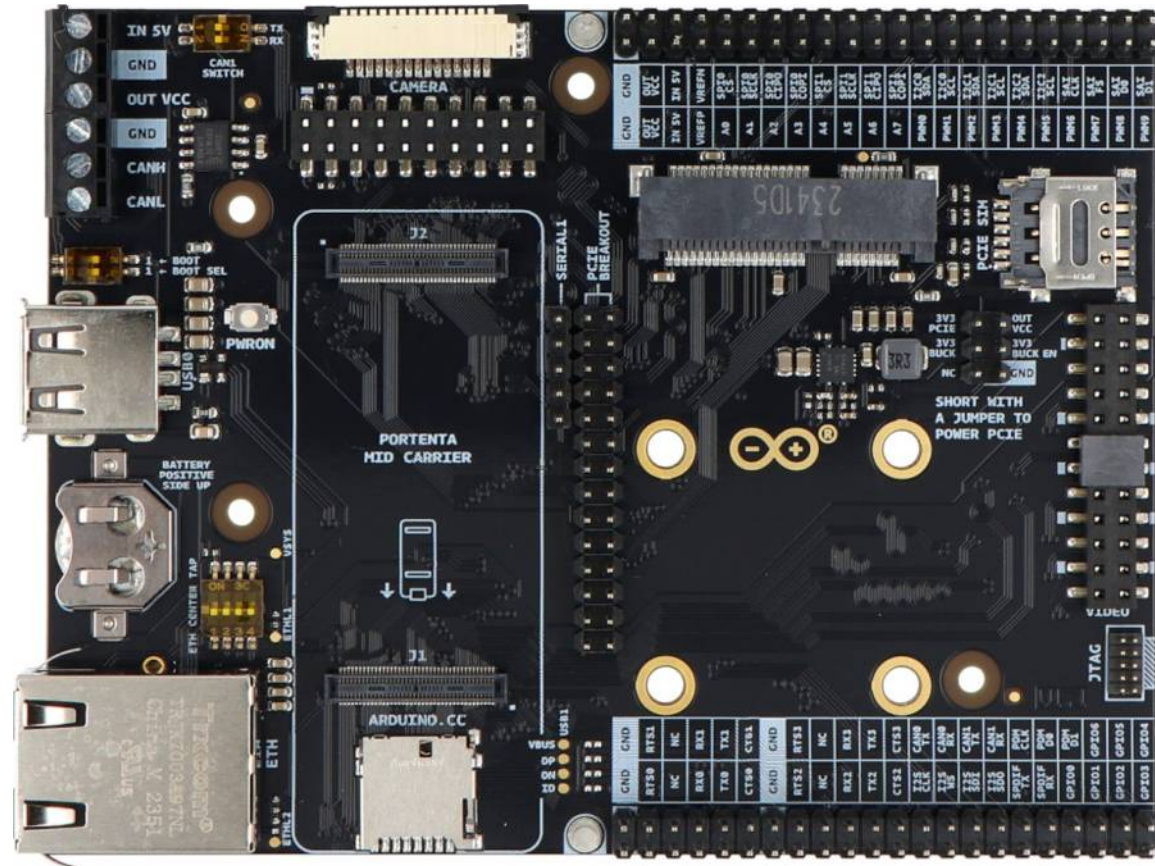
Portenta hordozókártyák

- ❖ Ahhoz, hogy a **Portenta X8** kártya jeleihez fizikailag hozzáférjünk, ki kell vezetni azokat, csatlakozókra, tűskesorokra
- ❖ A három legelterjedtebben használt gyári hordozókártya alább látható, ezek a **Portenta X8** mellett a **Portenta C33** és **Portenta H7** kártyákhoz is használhatók

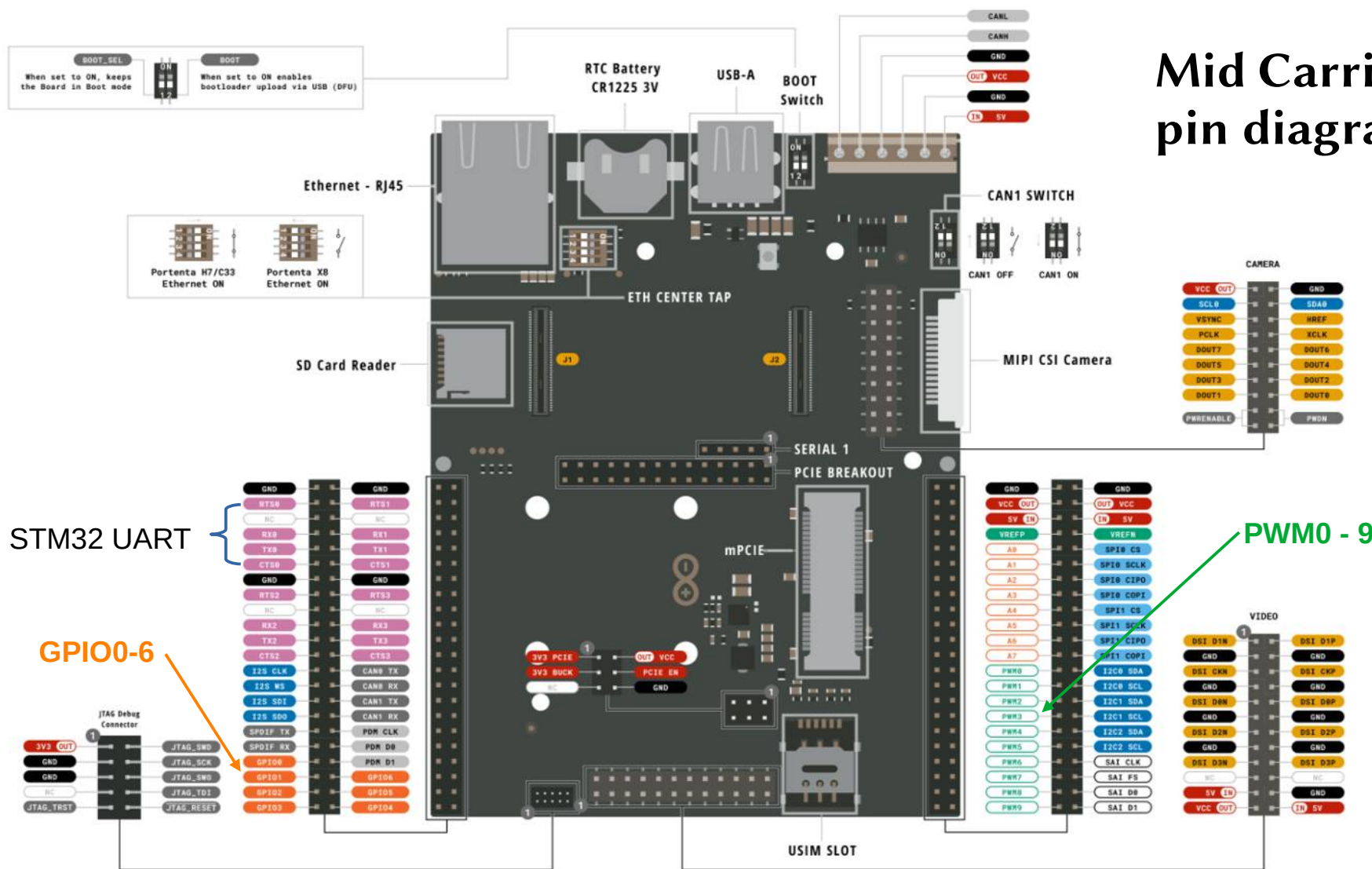
Hat carrier (€47,60)	Mid carrier (€67,10)	Max carrier (€340,00)
		

Portenta Mid Carrier

- ❖ **Periféria-elérés** – Mini PCIe, CAN Ethernet, microSD, USB, analóg I/O, SPDIF, PDM, SAI, UART, PWM, I2S, I2C, SPI, JTAG.
- ❖ **Linux-kompatibilitás** (Portenta X8) A hordozó teljes hozzáférést ad az i.MX8M interfészeihez
- ❖ **Két kamera csatlakozó** (MIPI + Arducam)
- ❖ **microSD foglalat** – Adatnaplózás és média-műveletek
- ❖ **Pro 4G modul támogatás** – mPCIe formátumú LTE/3G/2G + GNSS modul integrálható

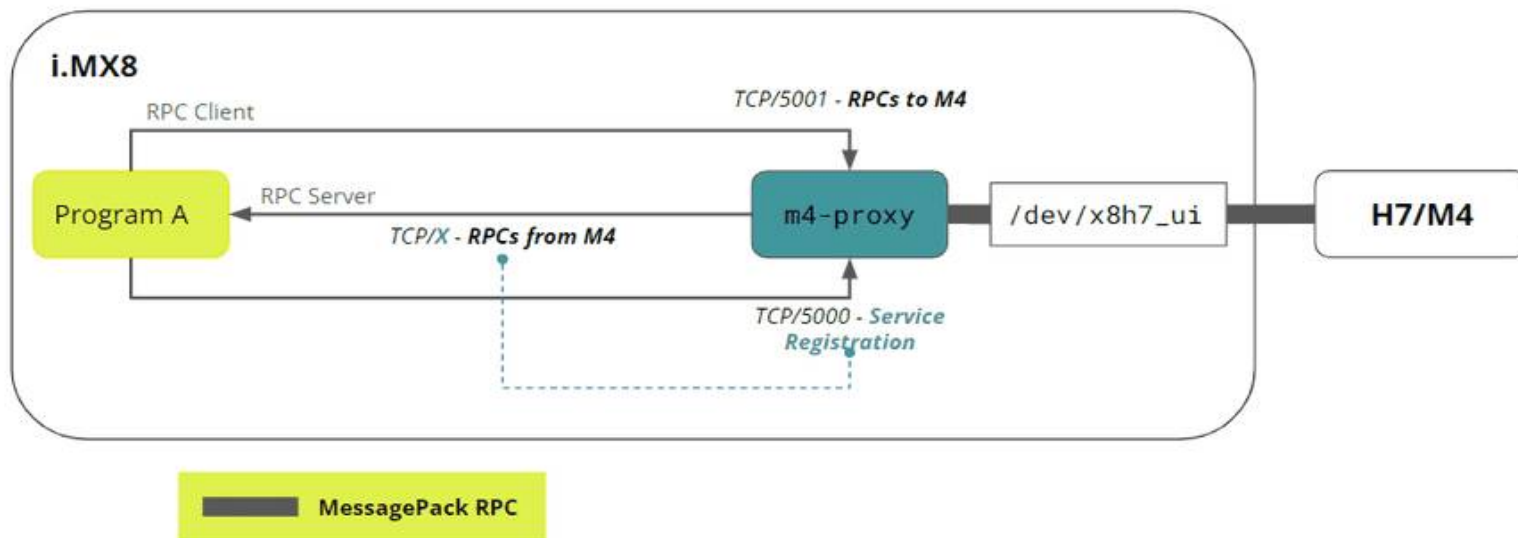


Mid Carrier pin diagram



Arduino támogatás

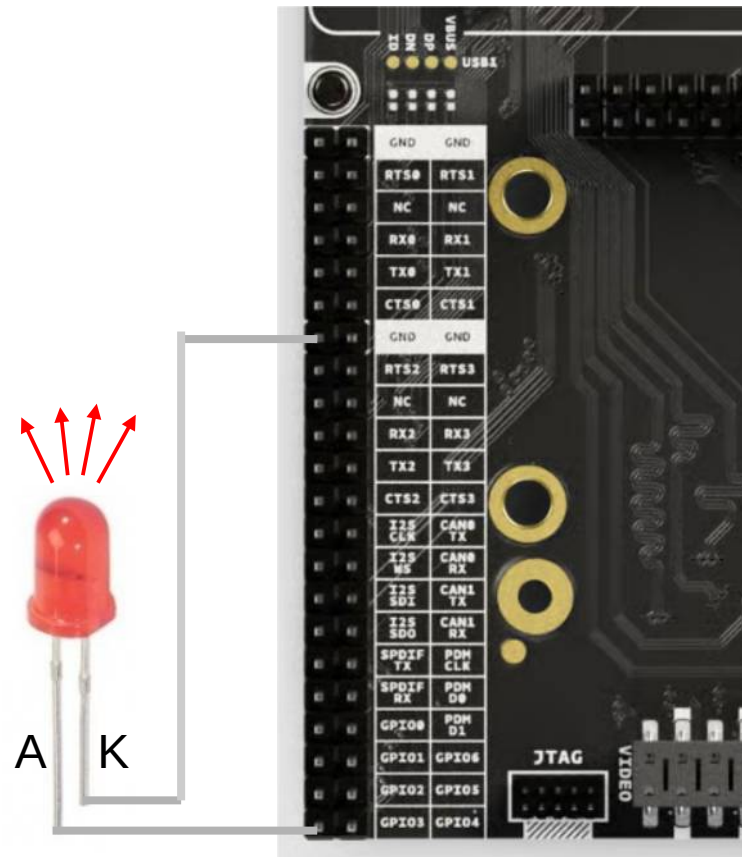
- ❖ A **Portenta X8** STM32H747 Cortex-M4 magjára Arduino sketch tölthető fel
- ❖ Mi a fejlesztésekhez az offline **Arduino IDE 1.8.19**-et használjuk, amelyet az **Arduino Mbed OS Portenta boards** csomaggal bővíteni kell (Tools → Board Manager menüben)
- ❖ Az **STM32H747-M4** és a Linuxot futtató **i.MX8M** processzor **RPC** (Remote Procedure Call) keresztül tudnak adatot cserélni



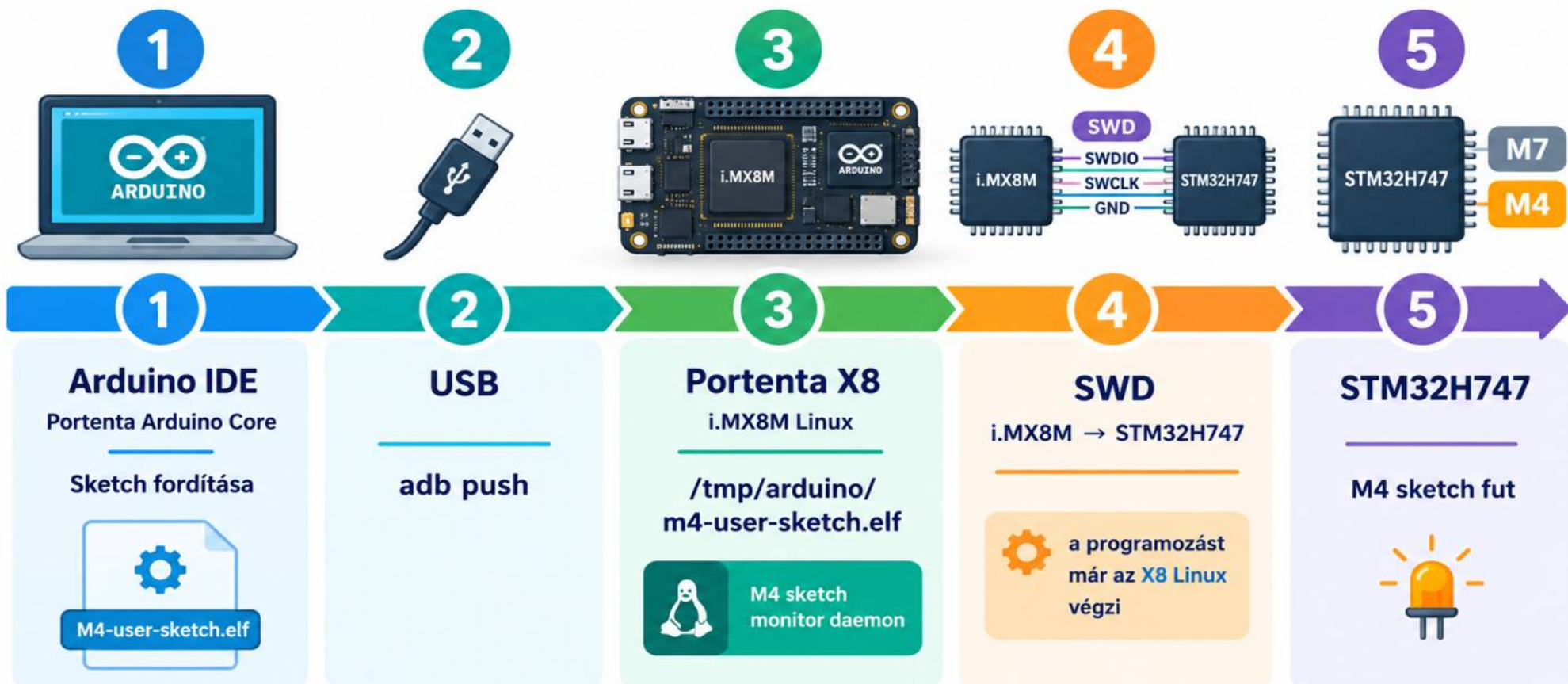
Az első lépések: blinky.ino

- ❖ A **Portenta Mid Carrier** kártya **GPIO3** kivezetésére kötött LED-et villogtatjuk
- ❖ Ezt a kivezetést a **STM32H747-M4** mikrovezérlő **PF4** portbitje vezérli
- ❖ A fordítás és kódfeltöltés látszólag a megszokott, egyszerű módon történik a háttérben azonban bonyolult mechanizmus dolgozik

```
void setup() {  
    pinMode(PF_4, OUTPUT);    // Mid Carrier GPIO3  
  
    void loop() {  
        digitalWrite(PF_4, HIGH);    // LED ON  
        delay(1000);                // 1 s várakozás  
        digitalWrite(PF_4, LOW);    // LED OFF  
        delay(1000);                // 1 s várakozás  
    }  
}
```



A Portenta X8 M4 sketch útja



Bővebben lásd: [Uploading Sketches to the M4 Core on Arduino Portenta X8](#)

GPIO elérés Linuxból

- ❖ **Chip azonosítás:** A **gpiodetect** parancs segít felderíteni a rendszerben lévő GPIO chipeket. A **GPIO3** kivezetés a **gpiochip5** 3-as vonalának felel meg
- ❖ **Állapot ellenőrzése:** A vonalak foglaltságának és státuszának lekérdezésére a **gpioinfo gpiochip5** parancs szolgál
- ❖ **Kimenet beállítása:** A modern karakteres illesztőn keresztül a kimeneti érték beállítása a **gpioset gpiochip5 3=1** paranccsal történhet
- ❖ **Olvasás:** A **GPIO3** bemenetként történő használatakor a **gpioget gpiochip5 3** paranccsal valósítható meg az olvasás

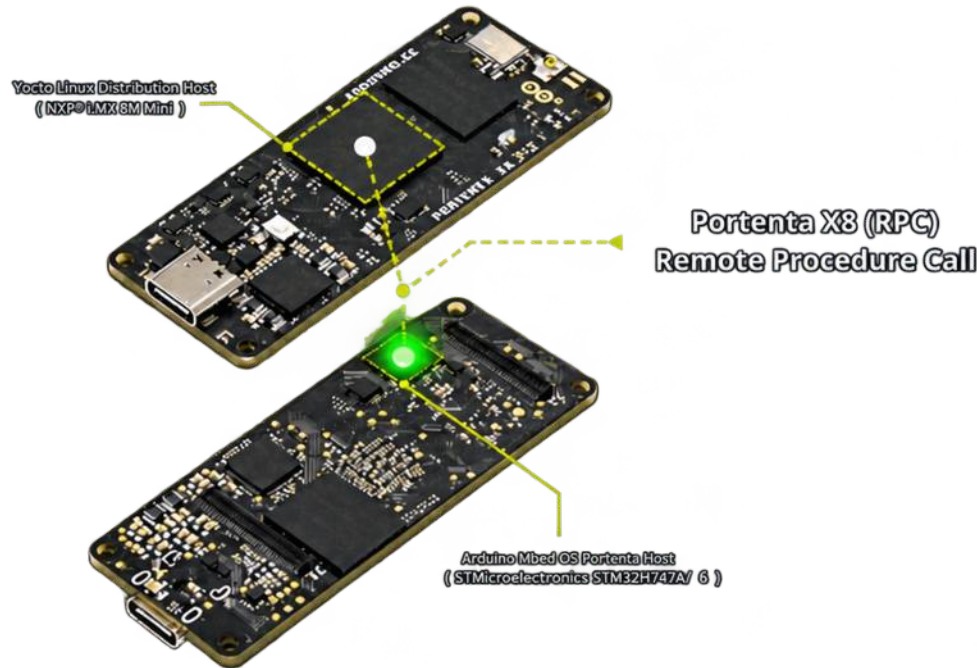
blinky.sh: A **gpioset** parancsokat egy shell scriptből is kiadhatjuk

A **GPIO3**-ra kötött LED villogtatása például így néz ki:

```
#!/bin/bash
#
while true; do
    gpioset gpiochip5 3=1
    sleep 0.5
    gpioset gpiochip5 3=0
    sleep 0.5
done
```

A hagyományos (/sys/class/gpio) elérésről itt olvashatunk: [Hello World Carrier](#)

Adatcsere az Arduino program és a Linux között



```
Windows PowerShell
Nov 28 23:43:44 portenta-x8-1822aa09dab6fad9 systemd[1]: Stopped M4 builtin led forwarder.
Nov 28 23:43:44 portenta-x8-1822aa09dab6fad9 systemd[1]: Started M4 builtin led forwarder.
^C
<ocker compose up1822aa09dab6fad9:/home/fio/python-blink-r
[+] Running 1/0
# Container python-blink-rpc-dummy-python-blink-rpc-1 Created 0.0s
Attaching to python-blink-rpc-dummy-python-blink-rpc-1
python-blink-rpc-dummy-python-blink-rpc-1 | =====
python-blink-rpc-dummy-python-blink-rpc-1 | == Portenta X8 M4 output ==
python-blink-rpc-dummy-python-blink-rpc-1 | =====
python-blink-rpc-dummy-python-blink-rpc-1 | count: 102
python-blink-rpc-dummy-python-blink-rpc-1 | led: 1
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 24.3099994659112383
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 41.790000915527344
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 849.989999234375
python-blink-rpc-dummy-python-blink-rpc-1 | count: 103
python-blink-rpc-dummy-python-blink-rpc-1 | led: 0
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 24.270000457763672
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 41.36600061035156
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 850.4500122070312
python-blink-rpc-dummy-python-blink-rpc-1 | count: 104
python-blink-rpc-dummy-python-blink-rpc-1 | led: 1
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 24.5
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 40.92000198364258
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 849.9299926757812
python-blink-rpc-dummy-python-blink-rpc-1 | count: 105
python-blink-rpc-dummy-python-blink-rpc-1 | led: 0
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 24.170000076293945
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 41.470001220703125
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 850.8800044828125
python-blink-rpc-dummy-python-blink-rpc-1 | count: 106
python-blink-rpc-dummy-python-blink-rpc-1 | led: 1
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 23.40999984741211
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 41.40999998474211
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 850.4500122070312
python-blink-rpc-dummy-python-blink-rpc-1 | count: 108
python-blink-rpc-dummy-python-blink-rpc-1 | led: 1
python-blink-rpc-dummy-python-blink-rpc-1 | temperature: 24.90999984741211
python-blink-rpc-dummy-python-blink-rpc-1 | humidity: 41.65999998474211
python-blink-rpc-dummy-python-blink-rpc-1 | pressure: 851.0299682617188
```

m4-sensors-read

- ❖ A CPU-k közti kommunikációt a Linuxon futó **M4-proxy** biztosítja **RPC** (Remote Procedure Call: távoli függvényhívás) segítségével néha nem árt újraindítani: `sudo systemctl restart m4-proxy`
- ❖ Az **M4**-en futó Arduino program **RPC.bind()**-dal publikál függvényeket
- ❖ A **Python** oldalon a **rpc.call("fn", data)** ezeket a publikált függvényeket hívja meg és válaszul megkapja a függvények visszatérési értékét
- ❖ A függvények „kiajánlása” és meghívása fordított irányban is működik
- ❖ Az első mintapéldában ([m4-sensors-read](#)) az Arduino még csak tesztadatokat küld a kapcsolat kipróbálására (a gyári forráskód itt található: [GitHub](#))

rpc-sensors.ino

- ❖ A fiktív szenzort képviselő Arduino kód lényege csak ennyi: konstans adatokat küld vissza, amelyetel távolról meghívható névtelen (lambda) függvények szolgáltatják. A sorszámszerű adatok segítenek detektálni az adatok elcsúszását

```
#include <RPC.h>

void setup() {
  RPC.begin();
  RPC.bind("temperature", []{ return 100; });
  RPC.bind("humidity", []{ return 200; });
  RPC.bind("pressure", []{ return 300; });
  RPC.bind("gas", []{ return 400; });
  RPC.bind("altitude", []{ return 500; });
}

void loop() {
  delay(100);
}
```

main.py - 2/1.

- ❖ A Python script valamivel bonyolultabb, mert „tudnia kell” az **M4-proxy**-hoz történő kapcsolódásról, és a **msgpackrpc** egy hibáját is ki kell kerülnie

```
import time
from msgpackrpc import Address as RpcAddress, Client as RpcClient, error as RpcError

publish_interval = 5

# The M4 Proxy address needs to be mapped via Docker's extra hosts
m4_proxy_host = 'm4-proxy'
m4_proxy_call_port = 5001

def get_data_from_m4(rpc_address):
    data = ()
    sensors = ('temperature', 'humidity', 'pressure', 'gas', 'altitude')
    try:
        get_value = lambda value: RpcClient(rpc_address).call(value)
        data = tuple(get_value(measure) for measure in sensors)
    except RpcError.TimeoutError:
        print("Unable to retrieve data from the M4.")
    return data
```

Egy „ismert programhiba”
miatt kell minden
paraméternek külön
RpcClient-et nyitni...

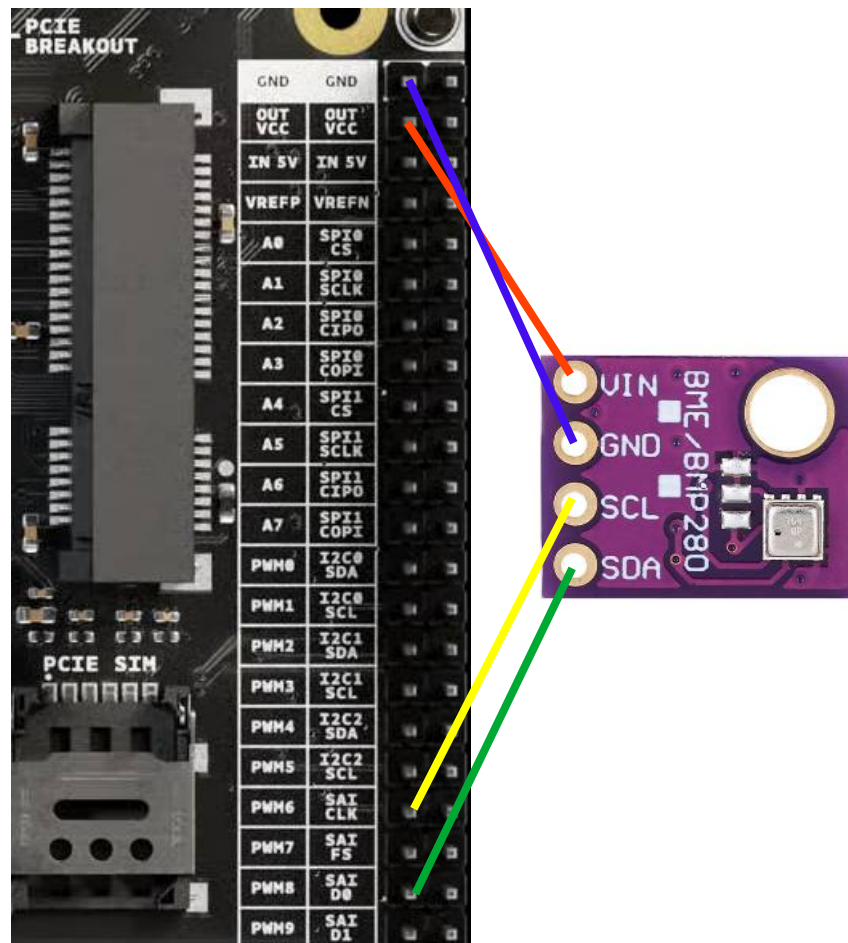
main.py - 2/1.

```
if __name__ == '__main__':
    print()
    print("=====")
    print("==          Portenta X8 Sensor reading          ==")
    print("=====")
    print()
    try:
        rpc_address = RpcAddress(m4_proxy_host, m4_proxy_call_port)
        while True:
            data = get_data_from_m4(rpc_address)
            if len(data) > 0:
                print("Temperature: ", data[0])
                print("Humidity: ", data[1])
                print("Pressure: ", data[2])
                print("Gas: ", data[3])
                print("Altitude: ", data[4])
                time.sleep(publish_interval)
    except Exception as e:
        print(f"Failed to create RpcClient: {e}")
```

Itt éppen el
vannak csúszva
az adatok,
ilyenkor kell
újraindítani az
M4-proxy-t

Csináljuk valódi szenzorral!

- ❖ Az STM32-nek csak egy I2C csatornája van kivezetve, méghozzá inkognitóban, a **PWM6** (PA_8 = I2C_SCL) és a **PWM8** (PA_12, PC_9 = I2C_SDA) feliratú tűskéken
- ❖ Vigyázat, a PA_12 és a PC_9 kivezetések össze vannak kötve, ezért fontos, hogy az egyik lábat (esetünkben: PA_12) bemenetre állítsuk!
- ❖ A **BME280** szenzor bekötése:
VIN → OUT VCC (3.3 V)
GND → GND
SCL → PWM6
SDA → PWM8



rpc-bme280.ino – 3/1.

```
#include <RPC.h>
#include <SerialRPC.h>
#include <Wire.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme;

float cached_temp = 1.0f;
float cached_hum = 2.0f;
float cached_press = 3.0f;
float cached_alt = 5.0f;

// Státusz változó: 0 = indulás, 1 = sikeres olvasás, -1 = hiba/nincs szenzor
int sensor_status = 4;

unsigned long last_read_time = 0;
const unsigned long read_interval = 2000; // 2 másodperces ütemezés

void setup() {
    pinMode(PA_12, INPUT);
    Serial.begin(115200);
    Wire.begin();
    RPC.begin();
}
```

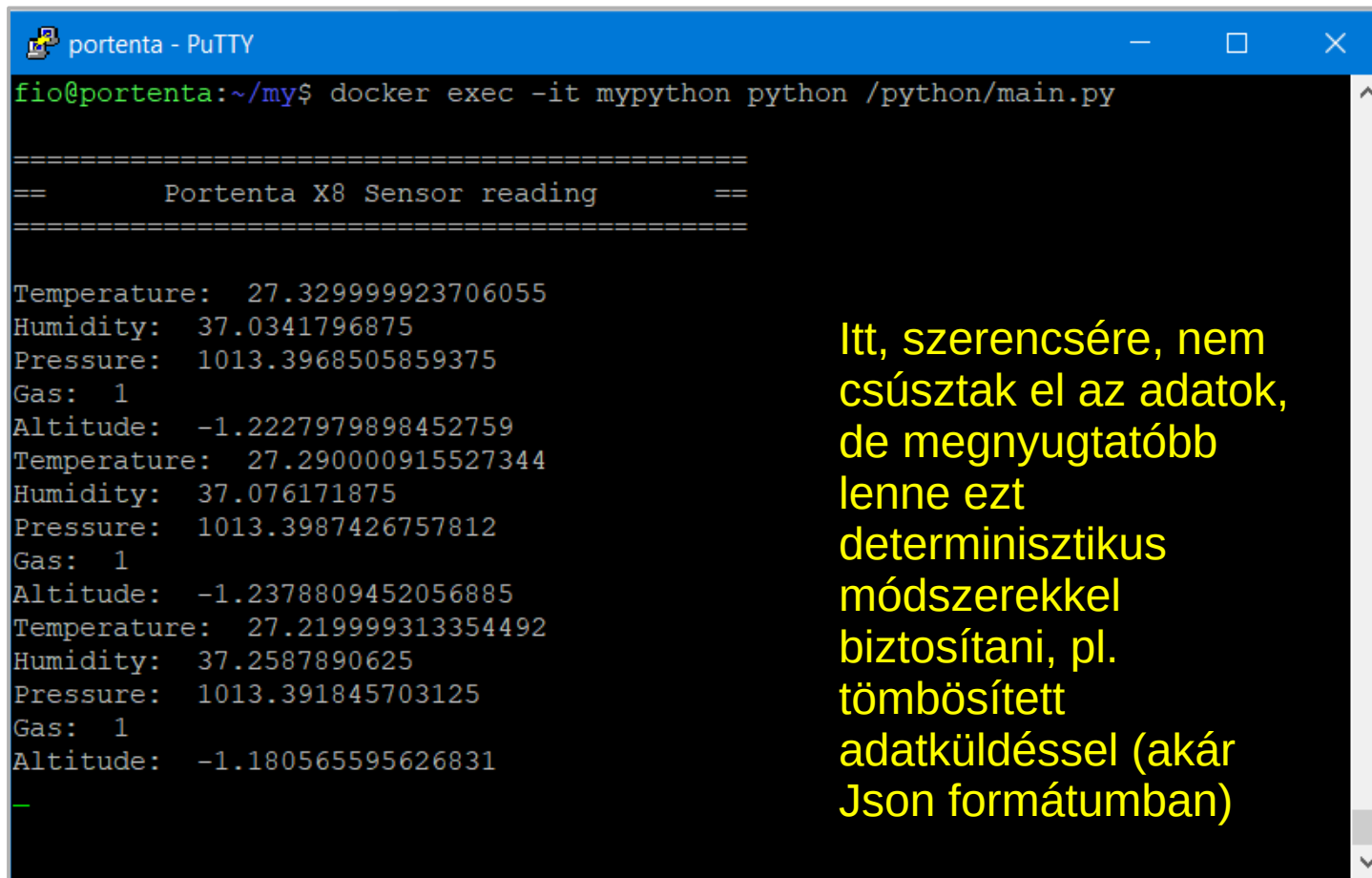
rpc-bme280.ino – 3/2.

```
Serial.println("Initializing BME280...");
// Kulturált, nem végtelenbe blokkoló inicializálás
int retries = 0;
while (!bme.begin(0x76) && retries < 5) {
    Serial.println("BME280 not found, retrying...");
    delay(500);
    retries++;
}
if (retries >= 5) {
    Serial.println("CRITICAL: BME280 initialization failed!");
    sensor_status = -1;
} else {
    Serial.println("BME280 found and initialized.");
    sensor_status = 1;
}
// RPC kötések (a "gas" mostantól a valós státuszt adja vissza)
RPC.bind("temperature", []{ return cached_temp; });
RPC.bind("humidity", []{ return cached_hum; });
RPC.bind("pressure", []{ return cached_press; });
RPC.bind("gas", []{ return sensor_status; });
RPC.bind("altitude", []{ return cached_alt; });
Serial.println("System initialized.");
}
```

rpc-bme280.ino – 3/3.

```
void loop() {  
    // Ha a státusz hiba (-1), meg sem próbáljuk olvasni a BME-t, ne terheljük a buszt  
    if (sensor_status == -1) {  
        delay(2000);  
        return;  
    }  
  
    unsigned long current_time = millis();  
    if (current_time - last_read_time >= read_interval) {  
        last_read_time = current_time;  
  
        cached_temp = bme.readTemperature();  
        cached_hum = bme.readHumidity();  
        cached_press = bme.readPressure() / 100.0F;  
        cached_alt = bme.readAltitude(1013.25);  
  
        // Sikeres olvasás után biztosítjuk, hogy 1 legyen a státusz  
        sensor_status = 1;  
        Serial.print("Temp: ");  
        Serial.print(cached_temp);  
        Serial.println(" *C [Status: OK]");  
    }  
}
```

A main.py ugyanaz, mint korábban



```
portenta - PuTTY
fio@portenta:~/my$ docker exec -it mypython python /python/main.py

=====
==      Portenta X8 Sensor reading      ==
=====

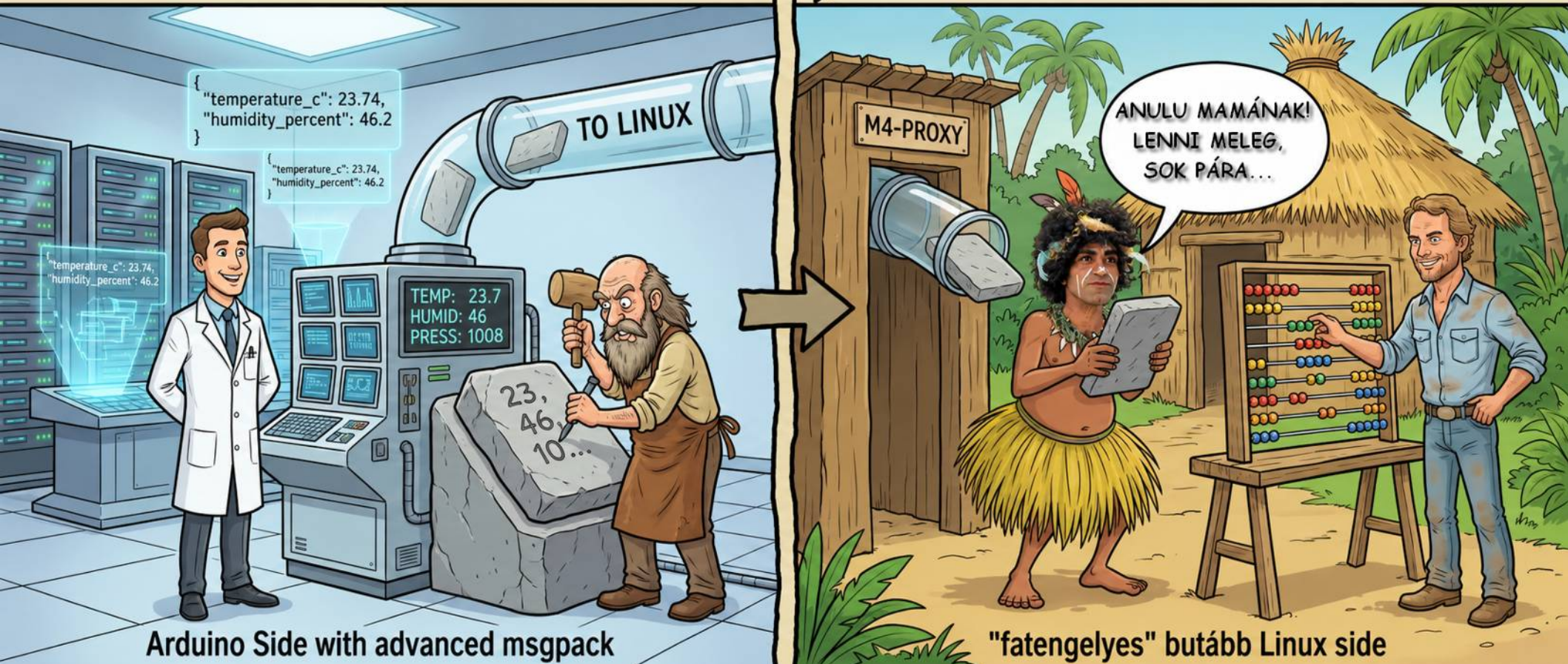
Temperature:  27.329999923706055
Humidity:    37.0341796875
Pressure:   1013.3968505859375
Gas:         1
Altitude:   -1.2227979898452759
Temperature: 27.290000915527344
Humidity:    37.076171875
Pressure:   1013.3987426757812
Gas:         1
Altitude:   -1.2378809452056885
Temperature: 27.219999313354492
Humidity:    37.2587890625
Pressure:   1013.391845703125
Gas:         1
Altitude:   -1.180565595626831
-
```

Itt, szerencsére, nem csúsztak el az adatok, de megnyugtatóbb lenne ezt determinisztikus módszerekkel biztosítani, pl. tömbösített adatküldéssel (akár Json formátumban)

Hogy odaát is megértsék ...

ARDUINO SIDE (Advanced MsgPack)

LINUX SIDE (m4-proxy, msgpack-rpc-python)



Az RPC hívás / küldés szoftverrétegei (Arduino M4 → Linux)

- ❖ Alkalmazás (M4): rpclib + MsgPack szerializáció
- ❖ Kétmagos belső kommunikáció (M4 ↔ M7): OpenAMP / RPMsg csatorna
- ❖ M7 Firmware: STM32H747AIH6_CM7.bin (rendszervezérlés)
- ❖ Kernel réteg: /dev/x8h7_ui eszközillesztő

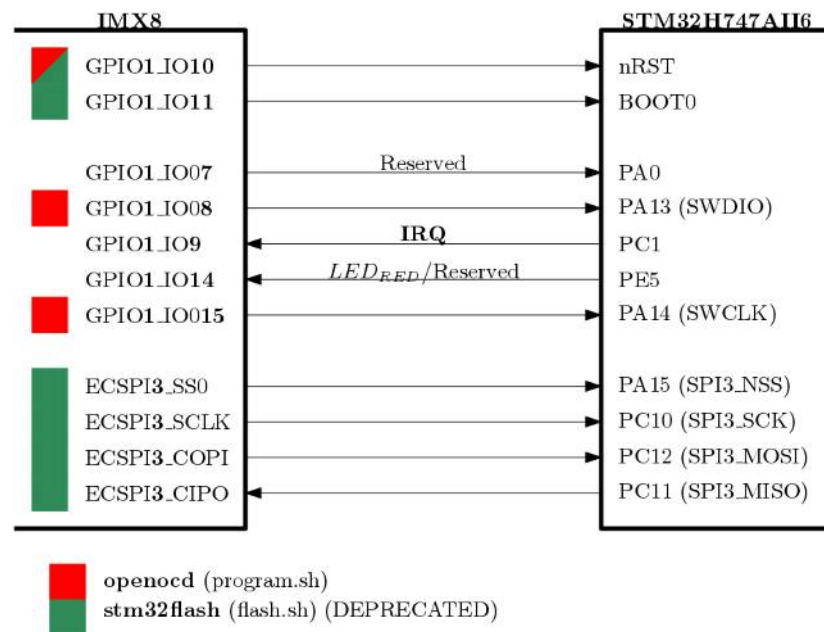


- ❖ **M4 Proxy:** Go-alapú háttérdeмон (socket/IPC konverzió)
- ❖ Python Kliens: msgpack-rpc-python
- ❖ Célalkalmazás: main.py (dashboard)

- ❖ **A probléma:** a két oldalon az **Msgpack** különböző verziójú, különböző csomagformátumot támogat, ezért nem értenek szót

$$GPIO_{NUM} = ((GPIO_BANK - 1) * 32) + IO_{NUM}$$

i.e. $GPIO1_IO07 = ((1 - 1) * 32) + 7 = 7$



A probléma megoldása: okos enged...

- ❖ Az Arduino oldali **rpclib** modernebb beépített **Msgpack** szerializálója csak rövid stringek esetén (max 31 karakterig) kompatibilis a Linux oldali „butább” **proxy** és **msgpack-rpc-python** rétegekkel
- ❖ Az alábbi módosítással kikényszerítjük, hogy az Arduino oldal a hosszabb STR típusú üzeneteket is a régebbi típusú formátumban küldje (FIXRAW, RAW16, RAW32)
- ❖ File: `packages\arduino\hardware\mbed_portenta\4.5.0\libraries\rpclib\src\rpc\msgpack\v1\object.hpp`
- ❖ Ezt a részletet

```
case c_lmddep_msgpack::type::STR:
    o.pack_str(v.via.str.size);
    o.pack_str_body(v.via.str.ptr, v.via.str.size);
    return o;
```

cseréljük ki erre:

```
case c_lmddep_msgpack::type::STR:
    o.pack_v4raw(v.via.str.size);
    o.pack_v4raw_body(v.via.str.ptr, v.via.str.size);
    return o;
```

rpc-json-bme280.ino – 3/1.

```
#include <RPC.h>
#include <SerialRPC.h>
#include <Wire.h>
#include <Adafruit_BME280.h>
#include <ArduinoJson.h>

Adafruit_BME280 bme;
JsonDocument doc;

char jsonBuffer[80];
float cached_temp = 25.0f;
int cached_hum = 45;
int cached_press = 1010;
int sensor_status = 0; // Státusz változó: 0 = indulás, 1 = sikeres olvasás, -1 = hiba/nincs szenzor
unsigned long last_read_time = 0;
const unsigned long read_interval = 2000; // 2 másodperces ütemezés

void updateJsonCache() {
    doc.clear();
    doc["t"] = cached_temp;
    doc["h"] = cached_hum;
    doc["p"] = cached_press;
    doc["s"] = sensor_status;
    serializeJson(doc, jsonBuffer, sizeof(jsonBuffer));
}
```

A javítás után most már bátran küldhetünk Json stringeket is, így, a névvel azonosított változókkal dolgozva a korábbi „elcsúszásokat” is elkerüljük. További előny, hogy az adatok így egyetlen adatcsomagként utaznak így lekérésenként csak egyetlen RpcClientet kell nyitni

Megjegyzés: A programban már az ArduinoJson 7.4.3 kiadását használjuk

rpc-json-bme280.ino – 3/2.

```
void setup() {
  pinMode(PA_12, INPUT); // Enélkül nem megy az I2C busz
  Serial.begin(115200);
  Wire.begin();
  RPC.begin();
  Serial.println("Initializing BME280...");
  int retries = 0;
  while (!bme.begin(0x76) && retries < 5) {
    Serial.println("BME280 not found, retrying...");
    delay(500);
    retries++;
  }
  if (retries >= 5) {
    Serial.println("CRITICAL: BME280 initialization failed!");
    sensor_status = -1;
  } else {
    Serial.println("BME280 found and initialized.");
    sensor_status = 1;
  }
  updateJsonCache();
  RPC.bind("getjson", []() -> const char* { return jsonBuffer; });
  Serial.println("System initialized.");
}
```

rpc-json-bme280.ino – 3/3.

```
void loop() {  
    // státusz hiba (-1) esetén nem olvassuk a szenzort  
    if (sensor_status == -1) {  
        updateJsonCache();  
        return;  
    }  
    unsigned long current_time = millis();  
    if (current_time - last_read_time >= read_interval) {  
        last_read_time = current_time;  
  
        // Értékek olvasása és kerekítése  
        cached_temp = (int)(bme.readTemperature() * 10.0f + 0.5f) / 10.0f; // 1 tizedesjegy  
        cached_hum = (int)(bme.readHumidity() + 0.5f); // egész  
        cached_press = (int)((bme.readPressure() / 100.0F) + 0.5f); // egész  
  
        sensor_status = 1;  
        updateJsonCache();  
  
        // Debug kiírás a tty-ra  
        Serial.println(jsonBuffer);  
        Serial.println("-----");  
    }  
}
```

Hogy fogadjuk a „becsomagolt” adatokat?

- ❖ A megváltozott adatküldés miatt (egy blokkban, Json formátumban) módosítanunk kellett a korábbi **main.py** programot
- ❖ Az új program neve **getjson.py** lett, s az adatlekéréshez a **'getjson'** névvel elérhető függvény távoli hívásával szerzi meg az adatokat
- ❖ A **main_test.py** programmal (ha átállítjuk a lekérdezést a **'getjson'** névre) megvizsgálhatjuk a kapott adatok formátumát és típusát:

```
=====
==          Portenta X8 Single Data Test          ==
=====

Received data: b'{"t":26.2,"h":39,"p":1009,"s":1}'
Type: <class 'bytes'>
Received data: b'{"t":26.2,"h":39,"p":1009,"s":1}'
Type: <class 'bytes'>
Received data: b'{"t":26.1,"h":39,"p":1009,"s":1}'
Type: <class 'bytes'>
```

Ez a bájtstring a Python beépített **json** könyvtárral könnyen értelmezhető, s a „felcímkézett” adatok miatt a keveredés vagy elcsúszás sem fenyeget

getjson.py - 2/1.

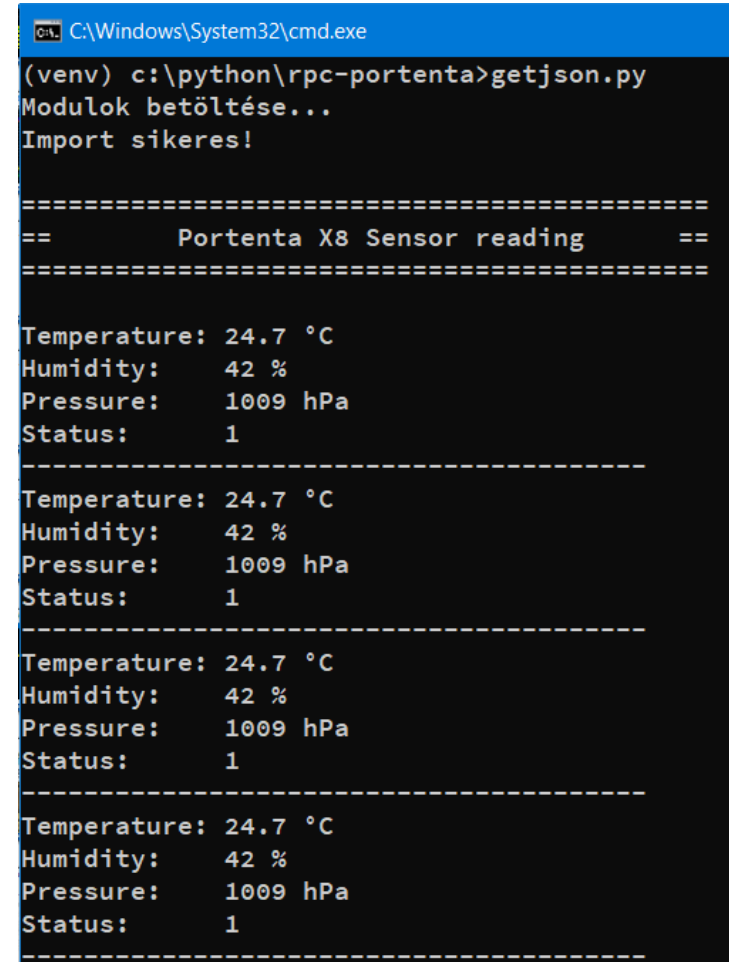
```
import sys, time, json
import time
import json
from msgpackrpc import Address as RpcAddress, Client as RpcClient, error as RpcError

publish_interval = 5
m4_proxy_host = "127.0.0.1"
m4_proxy_call_port = 5001

def get_data_from_m4(rpc_address):
    client = None
    try:
        client = RpcClient(rpc_address)
        raw_value = client.call('getjson')    # Itt vesszük át az adatokat
        return raw_value
    except RpcError.TimeoutError:
        print("Timeout az M4 adatok lekérdezése közben.")
        return None
    finally:
        if client:
            try: client.close()
            except: pass
```

getjson.py - 2/2.

```
if __name__ == '__main__':
    print("=====")
    print("==          Portenta X8 Sensor reading          ==")
    print("=====")
    rpc_address = RpcAddress(m4_proxy_host, m4_proxy_call_port)
    while True:
        raw_data = get_data_from_m4(rpc_address)
        if raw_data is not None:
            json_str = raw_data.decode('utf-8')
            payload = json.loads(json_str)
            temp = payload.get("t")
            hum = payload.get("h")
            press = payload.get("p")
            status = payload.get("s")
            print(f"Temperature: {temp} °C")
            print(f"Humidity:      {hum} %")
            print(f"Pressure:      {press} hPa")
            print(f>Status:        {status}")
            print("-" * 40)
            time.sleep(publish_interval)
```



```
C:\Windows\System32\cmd.exe
(venv) c:\python\rpc-portenta>getjson.py
Modulok betöltése...
Import sikeres!

=====
==          Portenta X8 Sensor reading          ==
=====

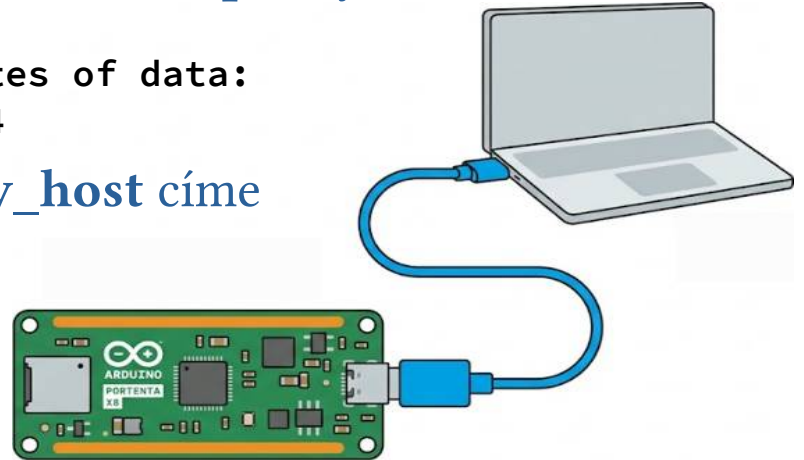
Temperature: 24.7 °C
Humidity:      42 %
Pressure:      1009 hPa
Status:        1
-----
Temperature: 24.7 °C
Humidity:      42 %
Pressure:      1009 hPa
Status:        1
-----
Temperature: 24.7 °C
Humidity:      42 %
Pressure:      1009 hPa
Status:        1
-----
Temperature: 24.7 °C
Humidity:      42 %
Pressure:      1009 hPa
Status:        1
-----
```

A `getjson.py` program eredménye

Adatlekérés távhívással

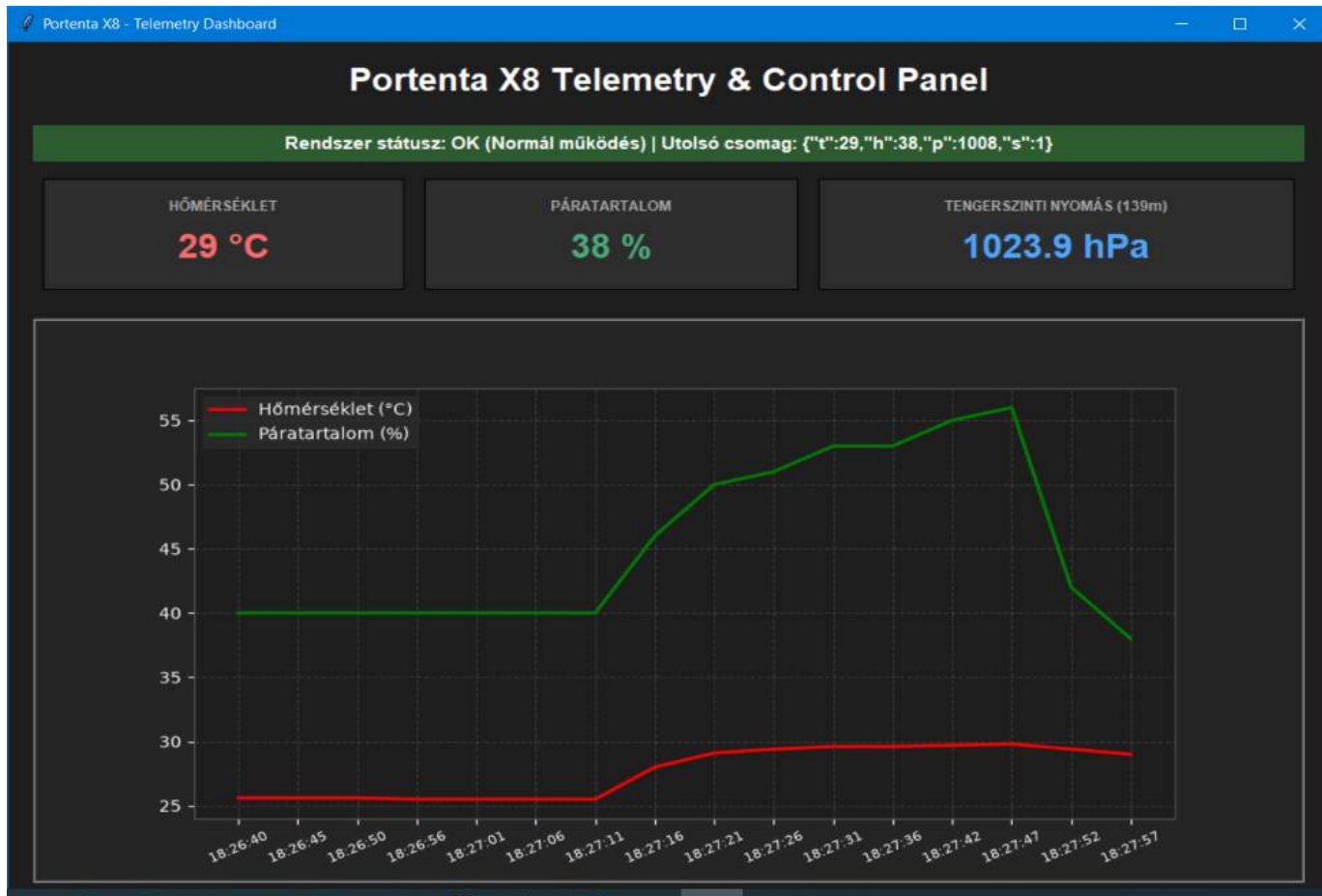
- ❖ **Egy kis meglepetés:** A Portenta X8 Arduinóban programozott M4 MCU-jának szenzorkezelő függvényei az **inter-core RPC** architektúrának köszönhetően a Linuxos környezet számára helyileg, a **TCP/IP** hálózaton keresztül pedig (legyen szó Wi-Fi-ről vagy vezetékes Ethernet kapcsolatról) az **m4-proxy** rétegen keresztül távolról is transzparensen elérhetők és meghívhatók
- ❖ **Hogyan csináljuk:** egy parancsablakban lekérdezzük a **Portenta X8** hálózati IP számát, és ezt adjuk meg a **getjson.py** programban az **m4_proxy_host** címének
`ping -4 portenta.local`
Pinging portenta.local [192.168.100.25] with 32 bytes of data:
Reply from 192.168.100.25: bytes=32 time=9ms TTL=64
- ❖ Esetünkben tehát **192.168.100.25** lesz az **m4_proxy_host** címe
- ❖ A Pythont az alábbi csomagokkal kell bővíteni:

<code>msgpack-python</code>	<code>0.5.6</code>
<code>msgpack-rpc-python</code>	<code>0.4.1</code>
<code>backports.ssl_match_hostname</code>	<code>3.7.0.1</code>
- ❖ A verzióütközések elkerülésére használjunk virtuális Python környezetet!



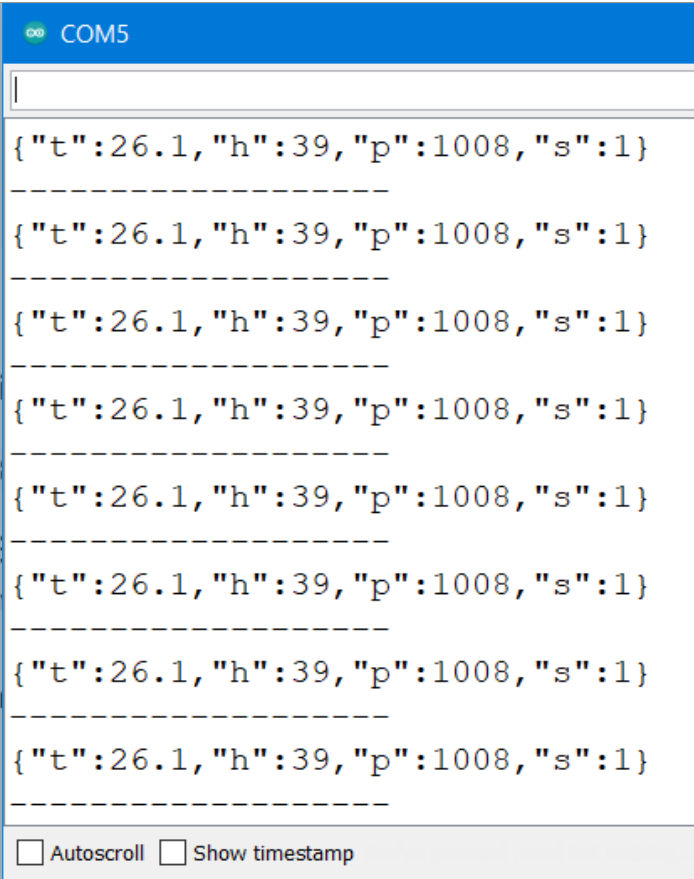
Ha már lúd, legyen kövér!

- ❖ Készítsünk egy Python grafikus felületű alkalmazás a beépített **Tkinter** könyvtár segítségével!
- ❖ Az előadás mellékletében a **portenta_dashboard.py** egy vibe-kódolt alkalmazás, ami az **rpc-json-bme280** ino Arduino program párjaként a **BME280** szenzor adatait kéri le és jeleníti meg
- ❖ **Vibe coding**: természetes nyelvű kérésekkel irányított, AI-val végzett kódolás



A soros portok elérése

- ❖ Az Arduino programoknál leggyakrabban a soros porti kiíratásokat használjuk nyomkövetésre, hibakeresésre. Az ArduiniCore-mbed környezetben ehhez alapértelmezetten a **Serial** és **Serial1** csatornák állnak rendelkezésre, melyek közül a **Serial** általában USB virtuális soros port, a **Serial1** pedig hardveres UART kimenet
- ❖ A **Portenta X8** esetében azonban az **M4** mag nem rendelkezik USB porttal, így a **Serial** kapcsolatot a **SerialRPC** könyvtár és az **RPC** kapcsolat felhasználásával, valamint egy linuxos segédprogram (**serialpc.py**) közvetítésével jut el a PC-hez. Ekkor az Arduino IDE terminál ablakában ugyanúgy kezelhetjük, mintha közvetlenül kapcsolódnánk az **M4** mikrovezérlőhöz



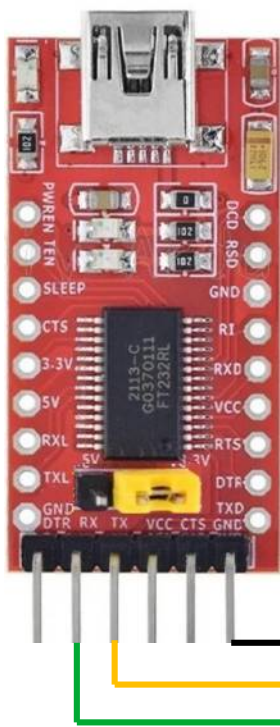
```
COM5

{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----
{"t":26.1,"h":39,"p":1008,"s":1}
-----

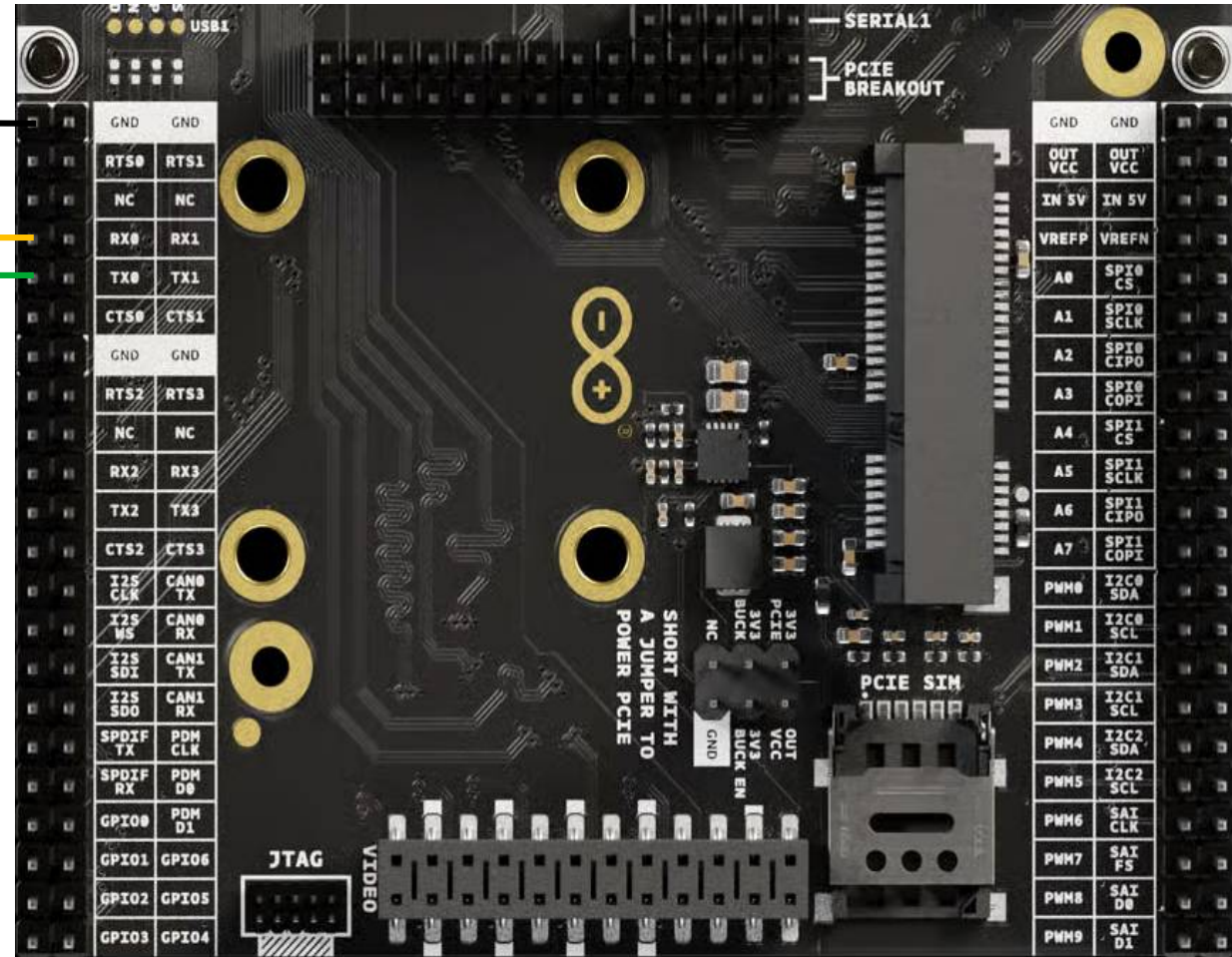
☐ Autoscroll ☐ Show timestamp
```

Az rpc-json-bme280.ino program kimenete

A Serial1 port elérése

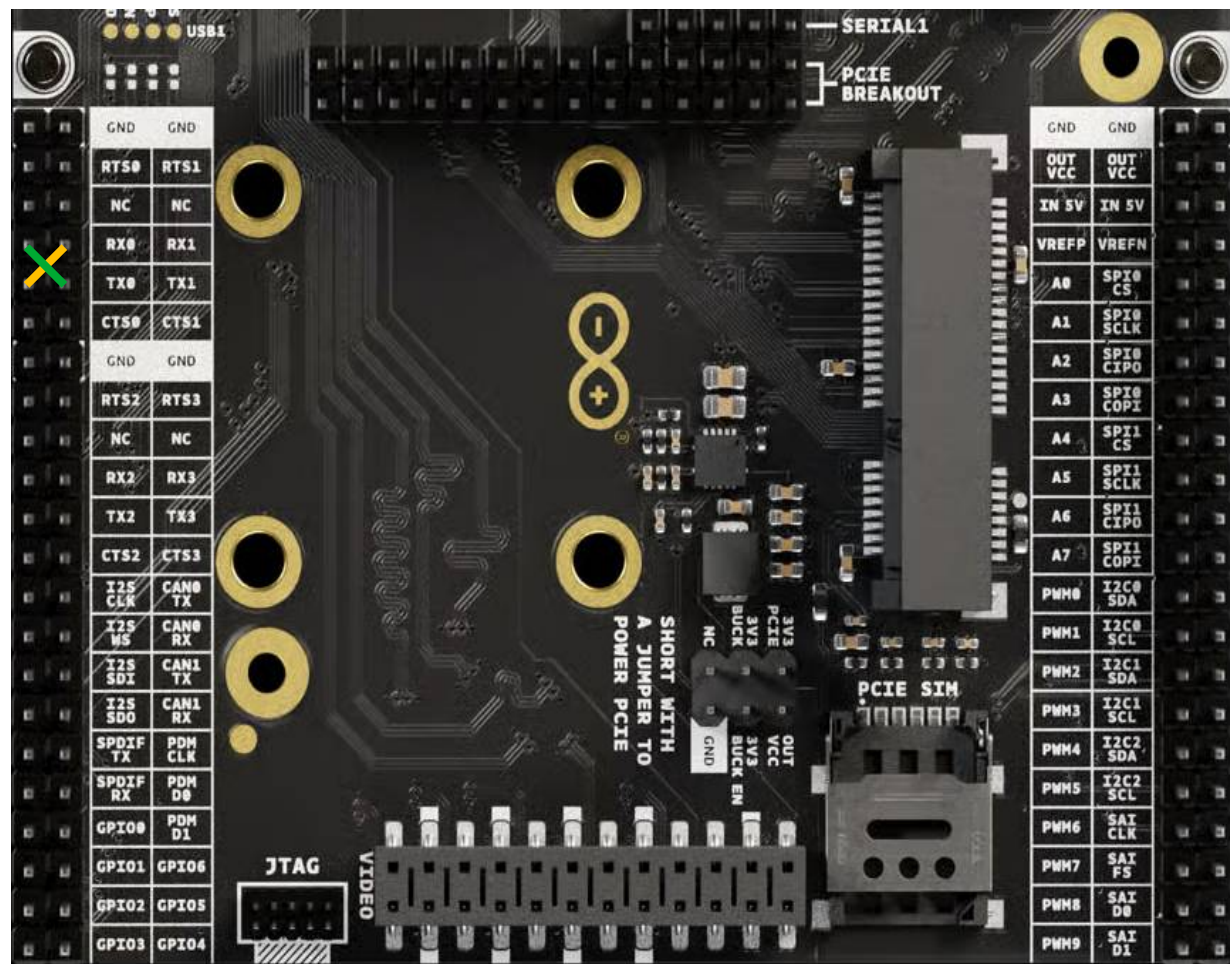


- ❖ A **Serial1** port közvetlenül (pl. egy FTDI USB-UART átalakítóval), a Mid Carrier kártya **RX0/TX0** tűskéin érhető el



A Serial1 port elérése másképp

- ❖ Egy másik lehetőség a Serial1 port elérésére az, hogy átkötjük a **Mid Carrier** kártya **RX1/TX1** tuskéire
- ❖ Ekkor a **Portenta X8 Linux** oldalán a **/dev/ttymxc1** eszközön férünk hozzá
- ❖ Konfigurálás:
`sudo stty -F /dev/ttymxc1 115200 cs8 -cstopb -parenb -ixon -ixoff -crtcts raw`
- ❖ Kiolvasás pl. így:
`sudo tail -f /dev/ttymxc1`



M4-serial1.ino

```
void setup()
{
    Serial1.begin(115200);
}

void loop()
{
    Serial1.println("HELLO FROM M4");
    delay(1000);
}
```

Portenta X8 – ADC

Analóg feszültség mérése Arduino programmal

Analóg jel
(0 – 3,3 V)

Portenta X8
(i.MX 8M Mini)

ADC
(analóg → digitális)



```
fio@portenta:~$ ./adc_exam
ADC = 2048
Feszültség = 1.65 V
ADC = 1987
Feszültség = 1.60 V
ADC = 2105
Feszültség = 1.70 V
fio@portenta:~$
```



Analóg
jel



Átalakítás



Arduino
Program



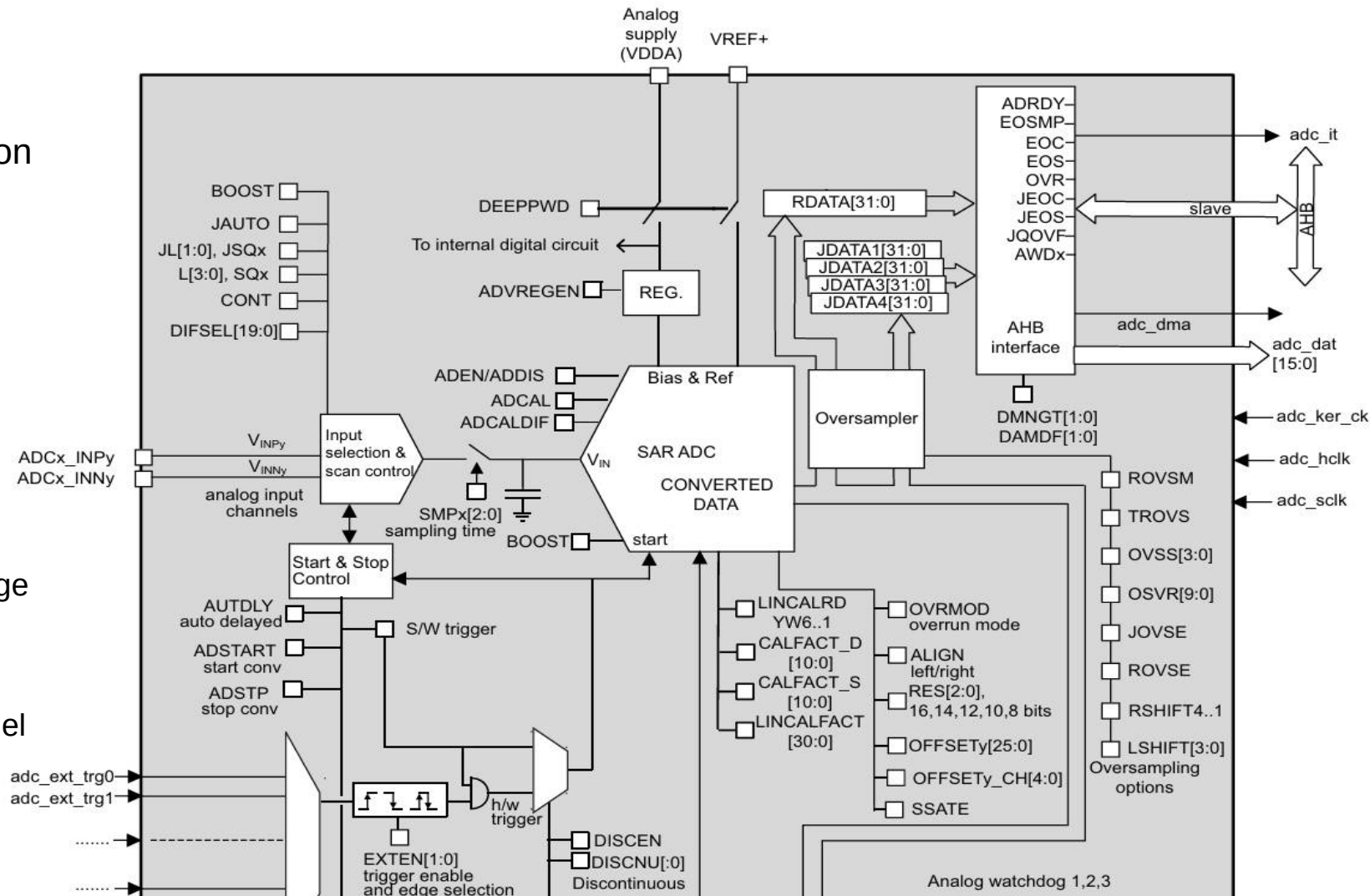
Linux
Feldolgozás

STM32 747AI

- **3 x ADC** with 16, 14, 12, 10 or 8-bit configurable resolution
- Oversampling ratio adjustable from 2 to 1024x
- 4 dedicated data registers for the injected channels
- Self-calibration (both offset and linearity)

5 internal dedicated channels

- Internal reference voltage (VREFINT)
- Internal temperature sensor (VSENSE)
- VBAT monitoring channel (VBAT/4)
- Connection to DAC internal channel



Feszültségmérés Linux alatt

- ❖ A Portenta X8 Linux oldaláról is elérhetők az **A0** – **A7** analóg bemenetek

```
/sys/bus/iio/devices/iio:device0/in_voltage0_raw  
/sys/bus/iio/devices/iio:device0/in_voltage1_raw  
/sys/bus/iio/devices/iio:device0/in_voltage2_raw  
/sys/bus/iio/devices/iio:device0/in_voltage3_raw  
/sys/bus/iio/devices/iio:device0/in_voltage4_raw  
/sys/bus/iio/devices/iio:device0/in_voltage5_raw  
/sys/bus/iio/devices/iio:device0/in_voltage6_raw  
/sys/bus/iio/devices/iio:device0/in_voltage7_raw
```

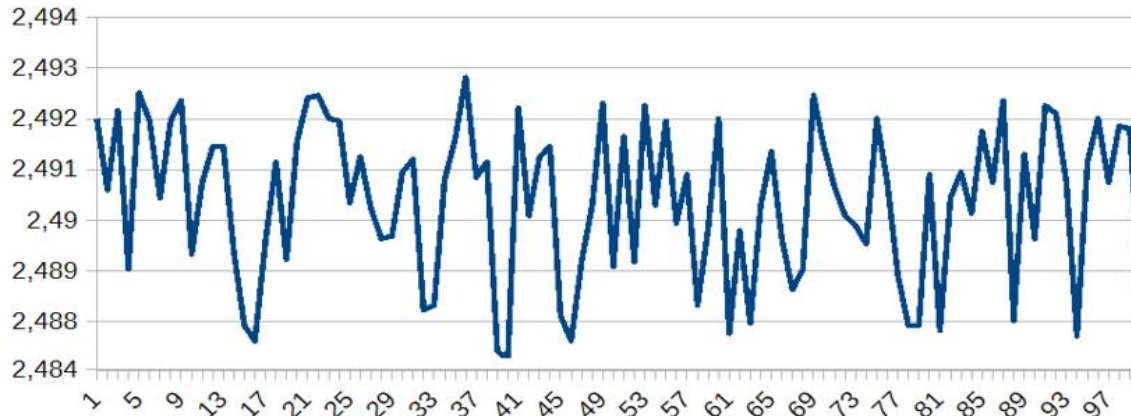
- ❖ Pl. **A0** kiolvasása: `cat /sys/bus/iio/devices/iio:device0/in_voltage0_raw` → 6652
- ❖ Alapértelmezetten a 0 – 3.3 V közötti bemenő feszültség 0 – 65 535 közötti értéket ad a mért feszültség tehát voltokban az alábbi képlettel számítható:

$$V_{\text{mért}} = \frac{ADC_{\text{raw}} * 3.3 \text{ V}}{65536}$$

adc.py

```
import time
def read_iio_adc():
    path = "/sys/bus/iio/devices/iio:device0/in_voltage0_raw"
    with open(path, "r") as f:
        raw_value = int(f.read().strip())
        voltage = raw_value * (3.3 / 65535.0)
        return raw_value, voltage

for i in range(0,100):
    val, volt = read_iio_adc()
    print("ADC0: ", val, volt, "V")
    time.sleep(0.1)
```



ADC0	49489	2.49200732433051	V
ADC0	49461	2.49059739070726	V
ADC0	49492	2.49215838864729	V
ADC0	49430	2.48903639276722	V
ADC0	49499	2.4925108720531	V
ADC0	49488	2.49195696955825	V
ADC0	49458	2.49044632639048	V
ADC0	49488	2.49195696955825	V
ADC0	49496	2.49235980773632	V
ADC0	49436	2.48933852140078	V
ADC0	49464	2.49074845502403	V
ADC0	49478	2.49145342183566	V
ADC0	49478	2.49145342183566	V
ADC0	49437	2.48938887617304	V
ADC0	49408	2.48792858777752	V
ADC0	49402	2.48762645914397	V
ADC0	49441	2.48959029526207	V
ADC0	49472	2.49115129320211	V
ADC0	49434	2.48923781185626	V
ADC0	49480	2.49155413138018	V
ADC0	49497	2.49241016250858	V
ADC0	49498	2.49246051728084	V
ADC0	49489	2.49200732433051	V
ADC0	49488	2.49195696955825	V
ADC0	49456	2.49034561684596	V
ADC0	49474	2.49125200274662	V
ADC0	49454	2.49024490730144	V
ADC0	49442	2.48964065003433	V
ADC0	49443	2.48969100480659	V
ADC0	49468	2.49094987411307	V

```
#include <RPC.h>
#include <SerialRPC.h>
uint16_t cached_adcraw = 1.0f;
float cached_adcvolt = 2.0f;
unsigned long last_read_time = 0;
const unsigned long read_interval = 2000;
```

```
void setup() {
  Serial.begin(115200);   delay(1500);
  RPC.begin();
  analogReadResolution(16);
  RPC.bind("adcraw", [] { return cached_adcraw; });
  RPC.bind("adcvolt", [] { return cached_adcvolt; });
  Serial.println("System initialized.");
}

void loop() {
  unsigned long current_time = millis();
  if (current_time - last_read_time >= read_interval) {
    last_read_time = current_time;
    cached_adcraw = analogRead(A0);
    cached_adcvolt = cached_adcraw * (3.3 / 65535.0);
    Serial.print("ADC0: "); Serial.print(cached_adcraw);
    Serial.print(" "); Serial.print(cached_adcvolt, 3);
    Serial.println(" V");
  }
}
```

rpc_adc.ino

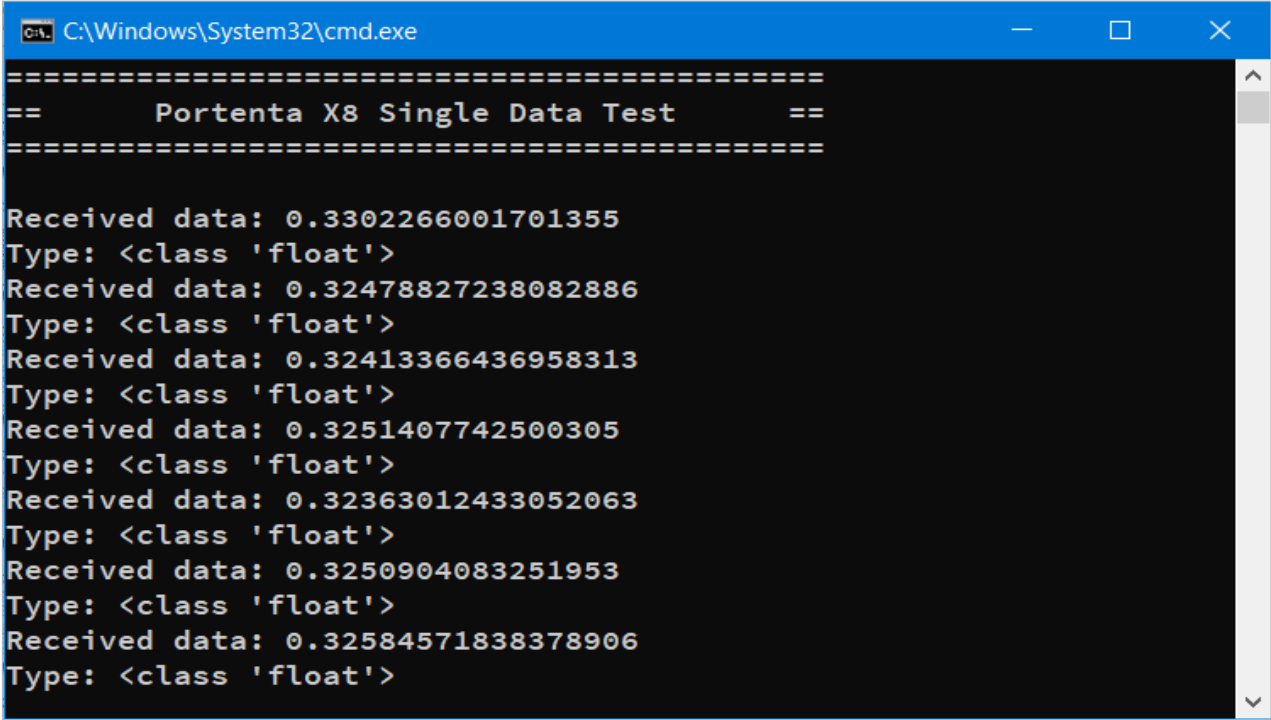
Arduino oldalról a szokásos módon, az **analogRead()** függvénnyel is kiolvashatjuk az értékeket – de a fejlett lehetőségekhez mélyebbre kell ásni (STM32 HAL, LL függvények)

COM5			
ADC0:	24149	1.216	V
ADC0:	20235	1.019	V
ADC0:	20734	1.044	V
ADC0:	20632	1.039	V
ADC0:	20332	1.024	V
ADC0:	21062	1.061	V
ADC0:	19255	0.970	V
ADC0:	20894	1.052	V
ADC0:	20366	1.026	V
ADC0:	21393	1.077	V
ADC0:	19750	0.995	V
ADC0:	19580	0.986	V
ADC0:	21381	1.077	V
ADC0:	6612	0.333	V
ADC0:	6629	0.334	V
ADC0:	6643	0.335	V

☐ Autoscroll ☐ Show timestamp Both NL & CR

rpc_adc.ino

- ❖ Az **rpc_adc.ino** program kimenő adatai a **Serial** porton kívül **RPC**-vel is elérhetők, itt pl. a **main_test.py** program segítségével kérjül le az '**adcvolt**' függvény távhívásával a mért és voltokra átszámított adatokat



```
C:\Windows\System32\cmd.exe

=====
==      Portenta X8 Single Data Test      ==
=====

Received data: 0.3302266001701355
Type: <class 'float'>
Received data: 0.32478827238082886
Type: <class 'float'>
Received data: 0.32413366436958313
Type: <class 'float'>
Received data: 0.3251407742500305
Type: <class 'float'>
Received data: 0.32363012433052063
Type: <class 'float'>
Received data: 0.3250904083251953
Type: <class 'float'>
Received data: 0.32584571838378906
Type: <class 'float'>
```