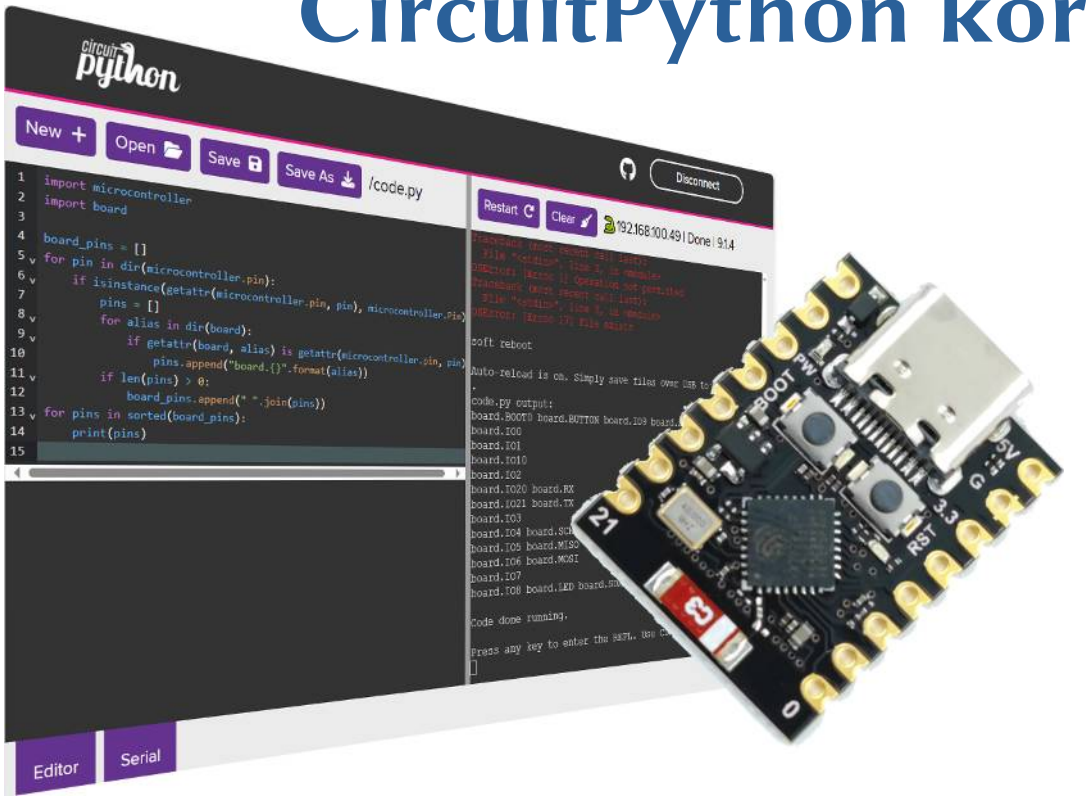
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



6. Az I2C busz (AHT10 szenzor, OLED kijelző)

Felhasznált és ajánlott irodalom

- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
  - P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- **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

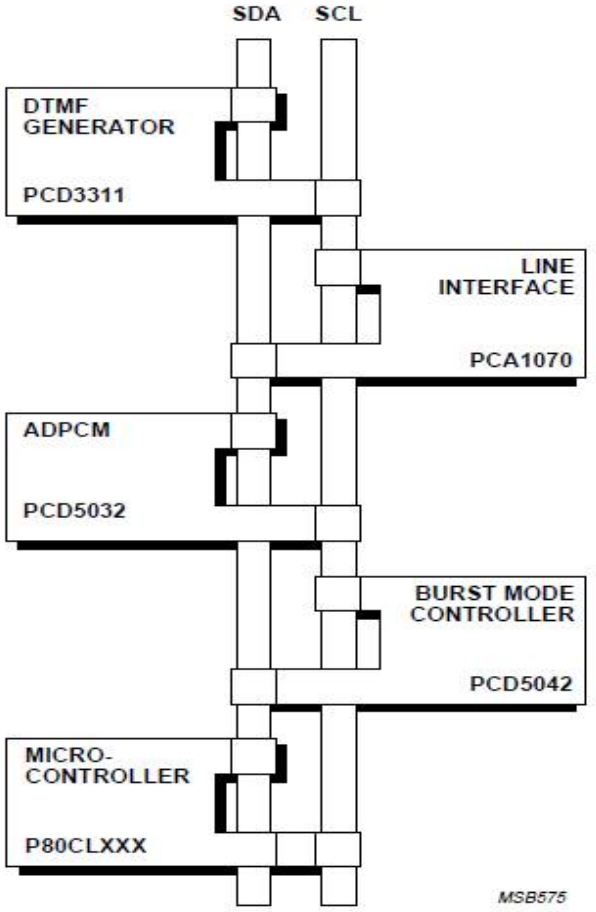
- **Online eszközök és támogatás:**
  - Learn Adafruit: CircuitPython on ESP32 Quick Start
  - Adafruit: Adafruit Web Serial ESPTool
  - Adafruit: CircuitPython Code Editor

Az I2C kommunikációs csatorna

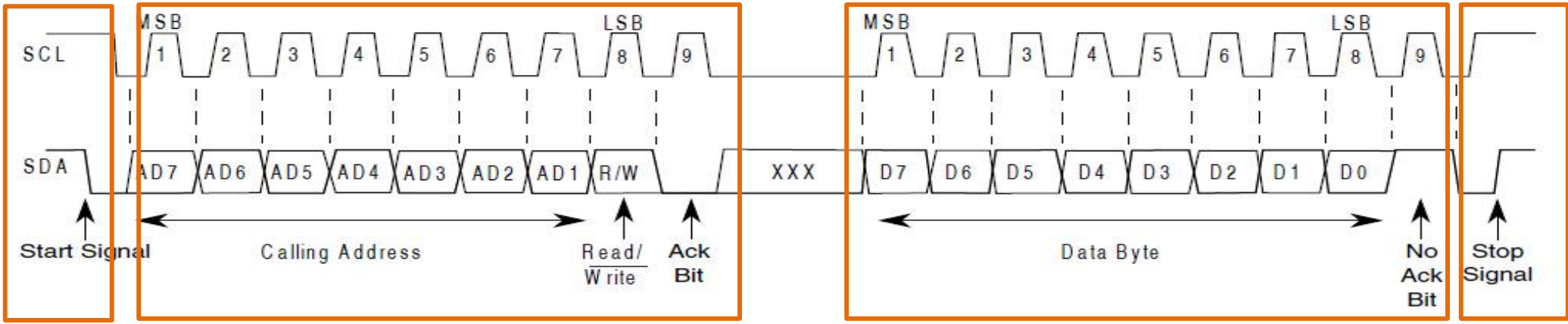
- Az **I2C** kommunikációs csatorna is bit-soros, órajellel szinkronizált adatátvitel, de a címzés itt az első bájtban kiküldött címmel történik
- Részletes leírás: **”Az I2C-busz specifikációja és használata”**
- **A korábbi előadásainkban található ismertetők:**
  - Az I2C kommunikációs csatorna (2020. 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 busz

- „Inter-Integrated Circuit” busz – vagy „kétvezetékes busz”, amelyet eredetileg a Philips cég dolgozott ki 1982-ben
- Több eszköz (master és slave) fűzhető fel a buszra
- A buszt a mester (master) eszközök vezérlik és kezdeményezik az adatforgalmat. A szolga (slave) eszköz akkor válaszol, ha címzéssel megszólítják
- Az I2C busz két jelvezetéket használ
  - SCL: szinkronizáló órajel
  - SDA: soros adat
- A buszvonalakat az eszközök csak lefelé húzhatják, fel csak egy-egy passzív elem (tipikusan 4.7 kΩ-os ellenállás) húzhatja
- Az ”Az I2C-busz specifikációja és használata” című dokumentumban olvasható a részletes leírása



I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

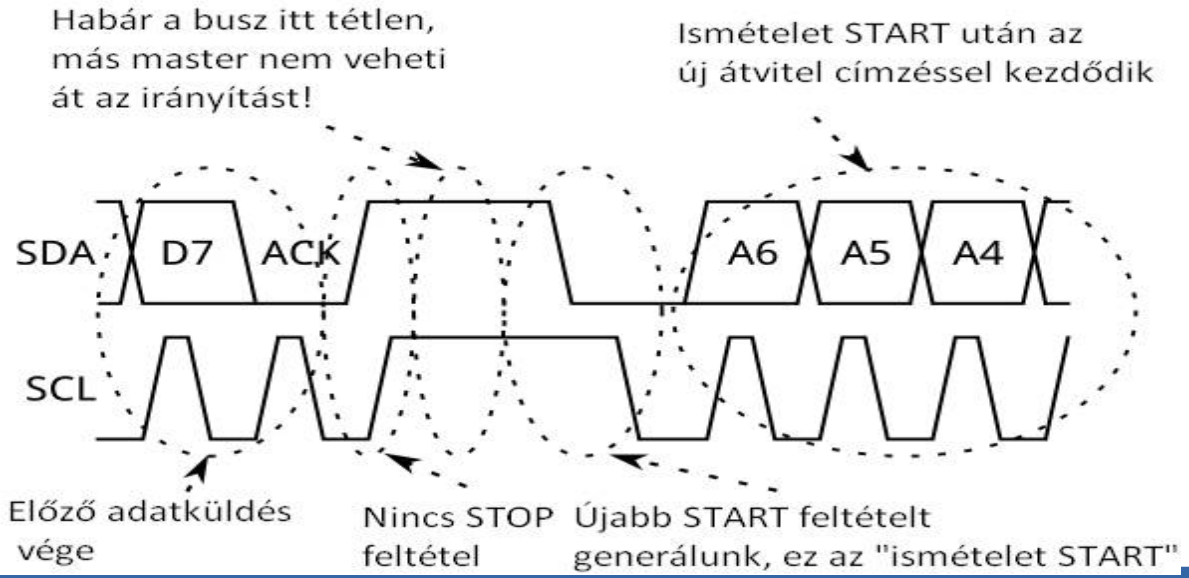
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

Nyugtázás (ACK): a 9. órajel impulzus tartamára a vevő alacsony szinten tartja az **SDA** jelvezetéket.

Negatív nyugtázás (NAK): a 9. órajel impulzus idején senki sem húzza le az **SDA** jelvezetéket – az magas szinten marad.

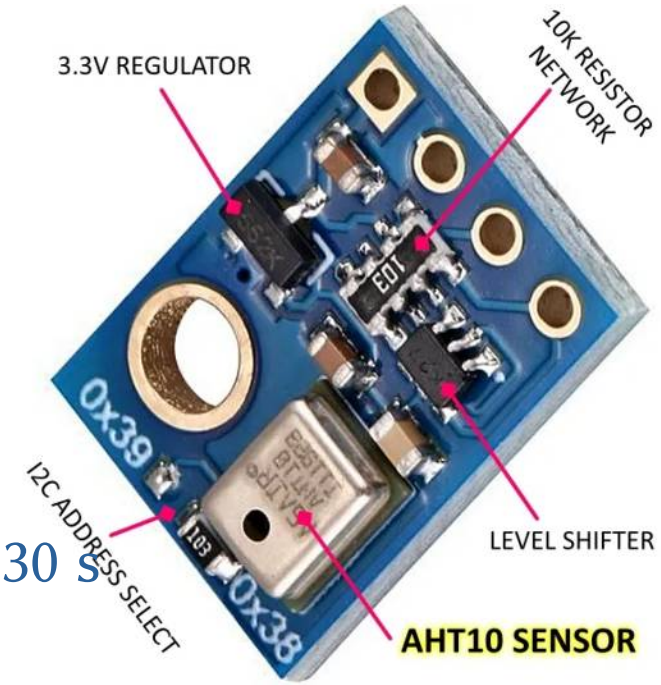
Ismételt START feltétel (repeated Start)

Gyakran szükség van arra, hogy egy összetett tranzakciót egy master egy menetben végigvihessen. Ilyen eset lehet például egy eszköz valamely regiszterének vagy memóriaterületének megcímzése, majd onnan adatok beolvasása. Ha az adatküldés és adatfogadás között nem generálunk **STOP** feltételt, akkor a master magánál tudja tartani a busz feletti vezérlést. A **STOP** nélkül generált újabb **START** feltételt **ismételt START**-nak (repeated START) nevezzük. Az **ismételt START** feltételt újabb cím/parancs bájt küldése kell, hogy kövesse.



Az AHT10 szenzor

- Az **AHT10** egy integrált hőmérséklet- és páratartalom-érzékelő, amelyet a Guangzhou Aosong Electronic Co., Ltd. gyárt
- **Főbb jellemzők:**
  - Felbontás: hőmérséklet: 0,01℃
    Páratartalom: 0,024% RH
  - Pontosság: ±3% relatív páratartalom (RH)
    és ±0,5℃ hőmérséklet
  - Működési tartomány: 0-100% RH
    és –40℃-tól 85℃-ig.
  - Válaszidő: Páratartalom 8 s, hőmérséklet 5-30 s
  - Interfész: I²C (alapértelmezett cím: 0x38)
  - I2C sebesség: 100 kHz, vagy 400 kHz
  - Tápfeszültség: 1,8V-tól 3,6V-ig



AHT10 parancsok

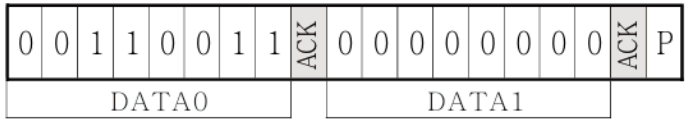
Command	Definition	Code
Initialization	Keep main engine	1110'0001
Trigger Measurement	Keep main engine	1010'1100
Soft reset		1011'1010

Table 9 Basic Commands

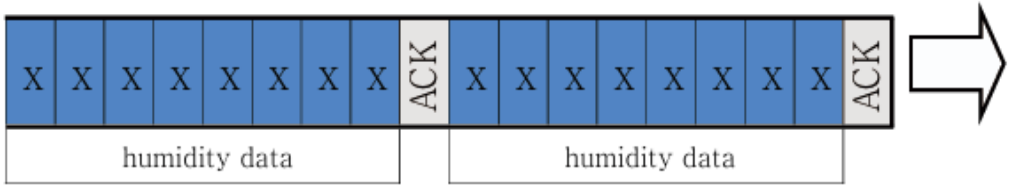
Bit	Definition	Description
Bit[7]	(Busyindication)	1 — Busy in measurement 0 — Free in dormant state
Bit [6:5]	(ModeStatus)	00 in NOR mode 01 in CYC mode 1x in CMD mode
Bit [4]	Remained	Remained
Bit [3]	CAL Enable	1--calibrated 0--uncalibrated
Bit [2 : 0]	Remained	Remained

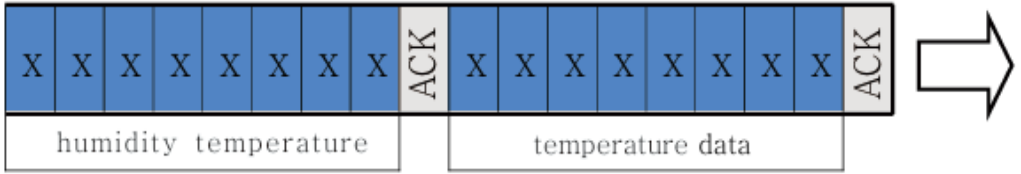
Table 10. State bit description.

Trigger measurement data



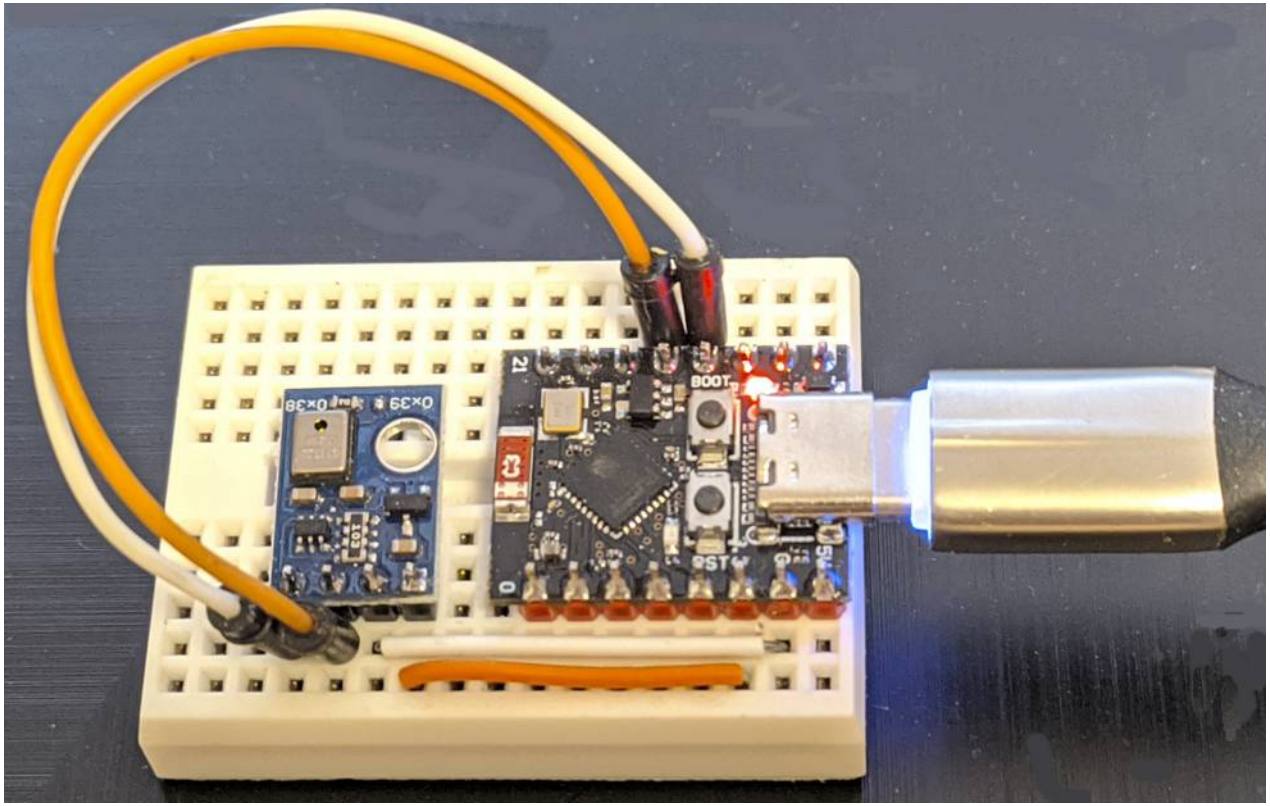
Read temperature and humidity data





Bekötési vázlat

- A Vin bemenetre 3,3 V-ot kötöttünk, SCL → GPIO9, SDA → GPIO8



aht10_demo.py – a nehezebbik út – 2/1.

- A **CircuitPython** beépített függvényeivel is kezelhetjük at **I2C** buszt és kiolvashatjuk az **AHT10** szenzort
- Az **i2c** objektum az összefogott **GPIO8** és **GPIO9** lábakon keresztül kezeli az **I2C** buszt
- A tranzakciók előtt az **I2C** buszt le kell foglalni (**trylock()**). Ennek itt nincs sok értelme, de enélkül nem adhatunk ki írás/olvasás parancsot

```
import board, busio, time
i2c = busio.I2C(board.SCL, board.SDA)          # I2C inicializálása
while not i2c.try_lock():
    pass
try:
    i2c.writeto(0x38, bytes([0xE1,0x08,0x00]))   # 0x38 az AHT10 alapértelmezett címe
    time.sleep(0.02)                           # Várakozás a kalibrálás befejezésére
finally:
    i2c.unlock()

while True:
    data = bytearray(6)                        # 6 bájt adatot akarunk kiolvasni
    while not i2c.try_lock():
        pass
    try:
        i2c.writeto(0x38,bytes([0xAC,0x33,0x00])) # Mérés indítása
        time.sleep(0.05)                       # Várakozás a mérés befejezésére
        i2c.readfrom_into(0x38, data)
    finally:
        i2c.unlock()
```

aht10_demo.py – a nehezebbik út – 2/2.

- A kiolvasott nyers adatok átszámításához az alábbi képleteket kell használni:

$$RH[\%]=\left(\frac{S_{RH}}{2^{20}}\right)*100\% \qquad T(℃)=\left(\frac{S_{T}}{2^{20}}\right)*200-50$$

```
# Páratartalom számítása
humidity = (data[1] << 12) | (data[2] << 4) | (data[3] >> 4)
humidity = humidity * 100 / 0x100000

# Hőmérséklet számítása
temperature = ((data[3] & 0x0F)<<16) | (data[4]<< 8)| data[5]
temperature = temperature * 200 / 1048576 - 50

# Eredmények kiíratása
print(f"Humidity: {humidity:.1f}%")
print(f"Temperature: {temperature:.1f} C")
time.sleep(2)  # 2 másodpercenként frissít
```

```
----------------------
Humidity: 22.4%
Temperature: 25.0 C
----------------------
Humidity: 22.6%
Temperature: 25.0 C
----------------------
Humidity: 22.6%
Temperature: 25.0 C
----------------------
Humidity: 22.6%
Temperature: 25.0 C
----------------------
Humidity: 22.6%
Temperature: 25.0 C
----------------------
```

AHT10 támogatói könyvtár

- Az **AHT10** szenzor kezeléséhez használhatunk külső programkönyvtárat is, pl. az Adafruit CircuitPython Library Bundle gyűjteményből az **adafruit_ahtx0.mpy** nevűt
  A kiválasztott könyvtárat az **ESP32-C3** saját fájlrendszerébe, a **lib** mappába kell bemásolni
- **CircuitPython**
  - **Adafruit_CircuitPython_AHTx0**
- **Inicializálás:** `sensor = adafruit_ahtx0.AHTx0(i2c,0x38)`
- **Tagfüggvények/tulajdonságok:**
  - **reset()** - soft reset
  - **calibrate()** - önkalibrálást indít
  - **status** - státuszbájt kiolvasása
  - **temperature** - hőmérséklet
  - **relative_humidity** - páratartalom

Opcionális paraméter (address)

- **Dokumentáció:** docs.circuitpython.org/projects/ahtx0

ahtx0_simpletest.py

- **CircuitPython**-ban így olvashatjuk ki a szenzort:

```
import time
import board
import adafruit_ahtx0

# Create sensor object, communicating over the board's
# default I2C bus
i2c = board.I2C()  # uses board.SCL and board.SDA
sensor = adafruit_ahtx0.AHTx0(i2c)
sensor.calibrate()
time.sloop(0.02)

while True:
    print("Temperature: %0.1f C" % sensor.temperature)
    print("Humidity: %0.1f %%" % sensor.relative_humidity)
    time.sleep(2)
```

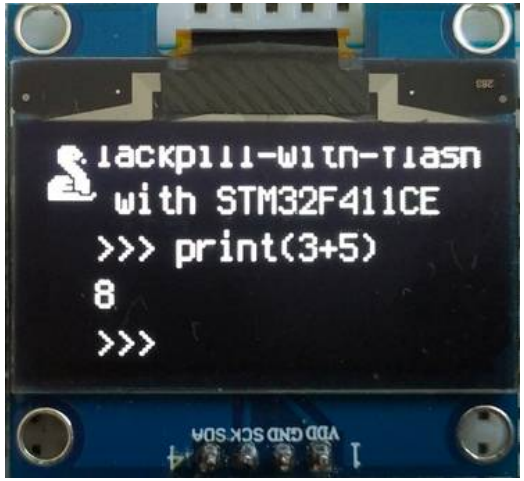
```
code.py output:
Temperature: 24.0 C
Humidity: 46.6 %
Temperature: 24.0 C
Humidity: 46.6 %
Temperature: 24.0 C
Humidity: 46.6 %
Temperature: 24.0 C
Humidity: 46.6 %
Temperature: 24.0 C
Humidity: 46.5 %
Temperature: 24.0 C
Humidity: 46.5 %
Temperature: 24.0 C
Humidity: 46.6 %
```

OLED kijelzők az I2C buszon

- Az **I2C OLED** grafikus kijelzőkben többnyire **SSD1306** vagy **SH1106** vezérlő található (ezek a kijelzők kaphatók SPI változatban is…)
- **OLED**-nek (*organic light-emitting diode*-nak) nevezzük a szerves fénykibocsátó diódát, amelyben a fénykibocsátásért felelős **elektrolumineszcens** réteg szerves félvezető vegyület, mely elektromos áram hatására világít
- 128x64 (128x32) felbontás
- 0.96” vagy 1.3” méret
- I2C cím **0x3C** (0x3D)
- 3,3V-5.0V tápfeszültség
- Normál/inverz mód
- Grafikus megjelenítés
- Szoftveresen állítható kontraszt
- Tükrözés/elforgatás

OLED kijelzők: „Két út van előttem…”

- Az egyszerűbb használati mód az **adafruit_ssd1306** meghajtó használata, amely a **busio** modul és az **adafruit_framebuf** (utóbbit telepíteni kell!) segítségével vezérli a kijelzőt (ami lehet akár **I2C,** akár **SPI** illesztőjű) A mai előadásban ezt használjuk!
- Mivel **adafruit_sh1106** meghajtó nem állt rendelkezésre, ezért a 2021/22-es tanfolyam keretében az **adafruit_ssd1306** mintájára készítettünk egyet (csak **I2C** illesztőhöz!)
- A másik lehetőség a **displayio** modul használata, ehhez **adafruit_displayio_ssd1306** és **adafruit_displayio_sh1106** meghajtó is található (az utóbbi, sajnos, nem megfelelően kezeli a 2 pixeles eltolódást, ezért nekünk kell megoldanunk azt az alkalmazásokban (a legegyszerűbb megoldás 132x64 felbontásúnak definiálni a kijelzőt és csak a középső 128x64 képpontot használni)
- A **displayio** megjelenítés használatakor a soros porti üzenetek a kijelzőn is megjelennek, ami néha zavaró lehet!



A kijelző vezérlésére használható függvények

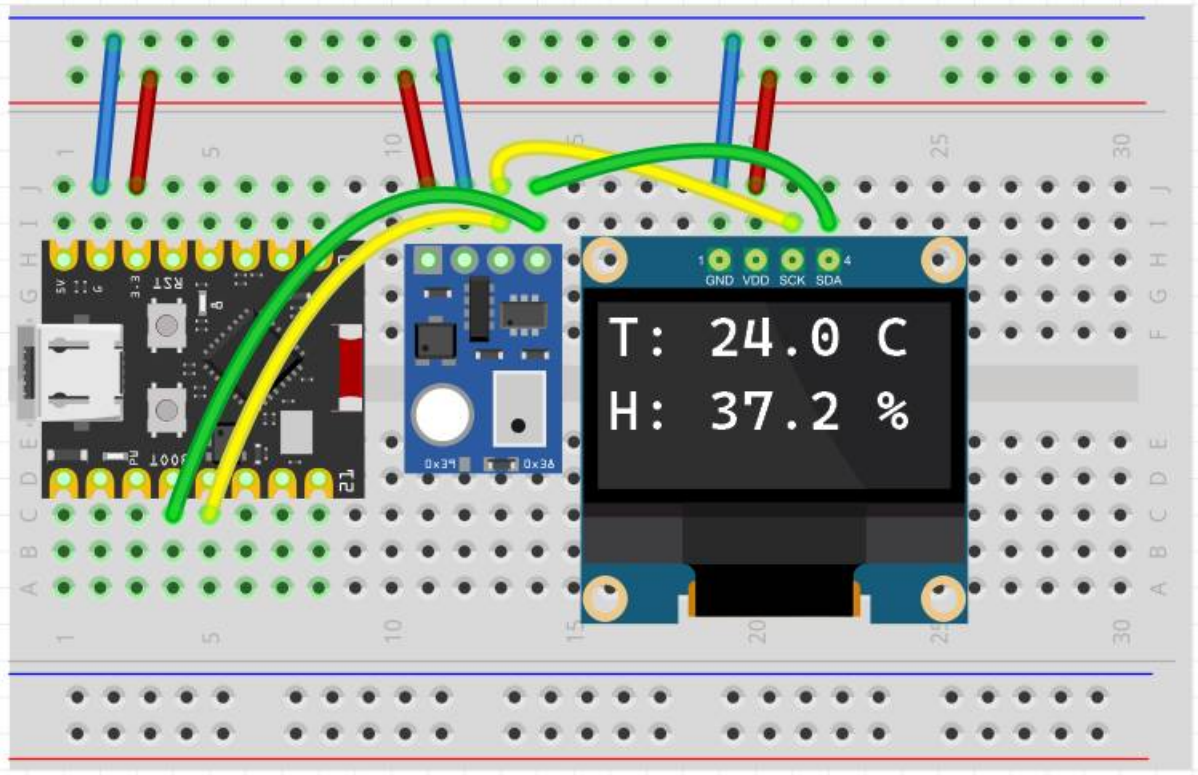
Az **adafruit_ssd1306** osztály az alábbi tagfüggvényeket örökli a **FrameBuffer** osztálytól:

- **circle**(center_x, center_y, radius, color) – kör rajzolása
- **fill**(color) – a FrameBuffer kitöltése a megadott színnel
- **fill_rect**(x, y, width, height, color) – kitöltött téglalap rajzolása
- **hline**(x, y, width, color) – vízszintes vonal rajzolása adott hosszúságig
- **line**(x0, y0, x1, y1, color) – szakasz rajzolás Bresenham algoritmussal
- **pixel**(x, y, color=None) – pont rajzolása, vagy színének leolvasása
- **rect**(x, y, width, height, color, , fill=False) – téglalap rajzolása
- **rotation** – a kép forgatása 90 fokonként: 0, 1, 2 vagy 3
- **scroll**(delta_x, delta_y) – FrameBuffer eltolása X és Y irányba.
- **text**(string, x, y, color, *, font_name='font5x8.bin', size=1) – szöveg megjelenítése
- **vline**(x, y, height, color) – függőleges vonal rajzolása

Hő- és páratartalom mérés kijelzéssel

- Állítsunk össze egy komplett alkalmazást, amelyben a hőmérsékletet és a relatív páratartalmat mérjük egy **AHT10** szenzor segítségével, s az eredményt egy **SSD1306 I2C OLED** kijelzőn íratjuk ki!
- **Bekötési lista**

ESP32-C3	AHT10	OLED
GND	GND	GND
VCC	VCC	VCC
GPIO8	SDA	SDA
GPIO9	SCL	SCL



- **I2C címek:**
  - AHT10: 0x38
  - OLED: 0x3C

ahtx0_ssd1306.py

```
import time
import board
import busio
import adafruit_ahtx0
import adafruit_ssd1306

i2c = busio.I2C(board.SCL, board.SDA, frequency=400000)
sensor = adafruit_ahtx0.AHTx0(i2c)
oled = adafruit_ssd1306.SSD1306_I2C(128,64,i2c,addr=0x3c,external_vcc=False)
oled.rotate(1)      # 180 fokos elforgatás

while True:
    temperature = sensor.temperature
    humidity = sensor.relative_humidity
    oled.fill(0)    # Képernyő törlés
    oled.text(f"T: {temperature:.1f} C", 0, 0, 1, size=2)
    oled.text(f"H: {humidity:.1f} %", 0, 32, 1, size=2)
    oled.show()
    time.sleep(2)  # Adatok frissítése 2 másodpercenként
```

Feltöltendő fájlok :
lib/adafruit_ahtx0.mpy
lib/adafruit_framebuf.mpy
lib/adafruit_ssd1306.mpy
font5x8.bin

Az ESP32 C3 Super Mini kártya kivezetései