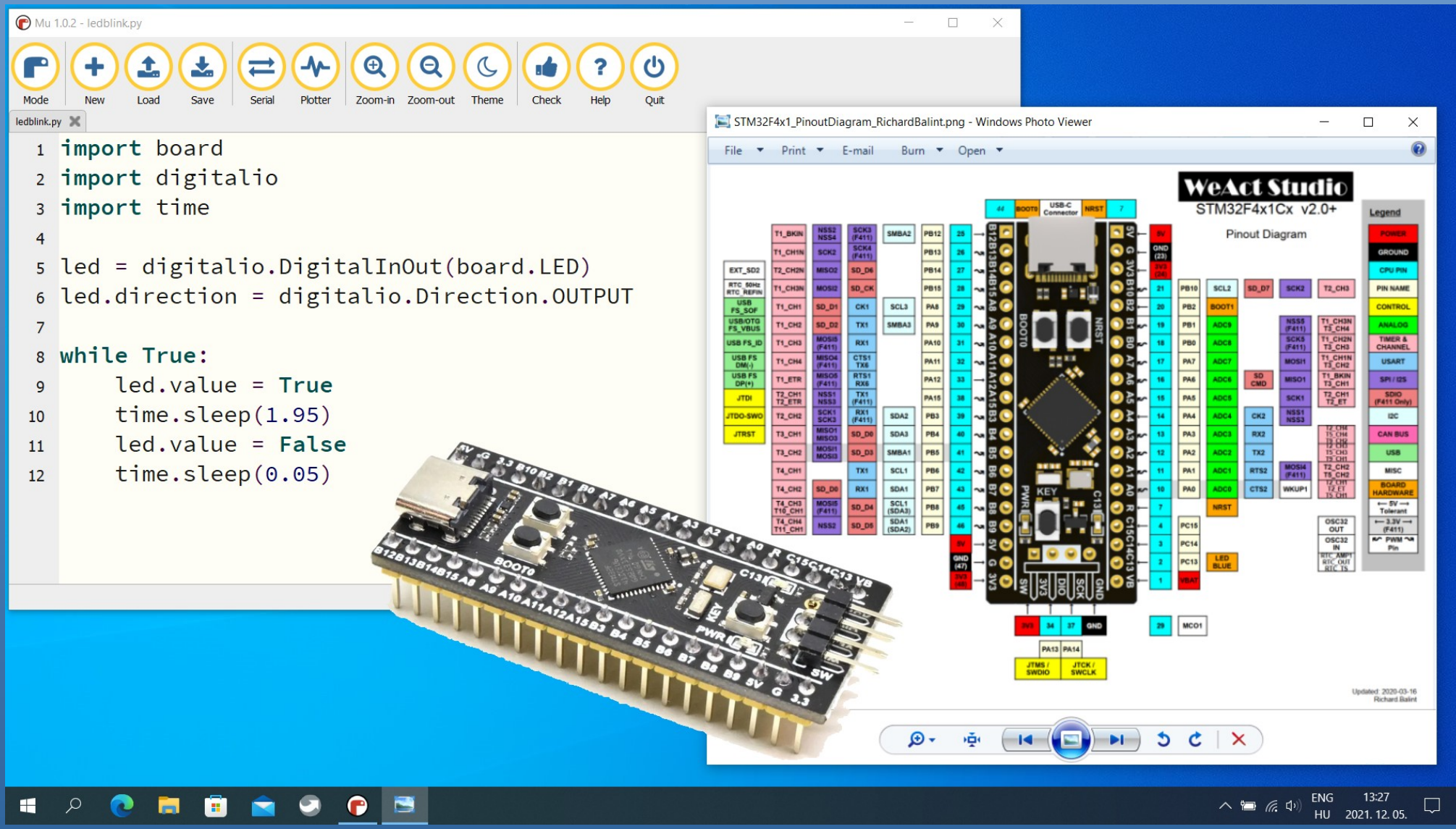
CircuitPython tanfolyam



8. Multitasking és az MCP23017 használata

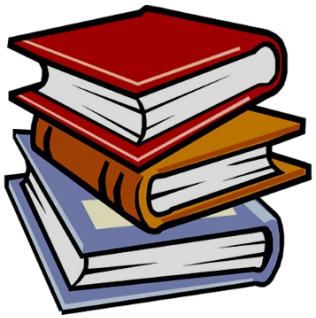
Felhasznált és ajánlott irodalom

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

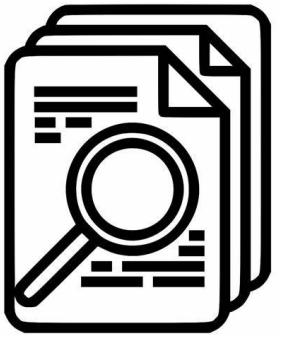
CircuitPython:

- Adafruit: https://circuitpython.org/downloads
- Adafruit: Adafruit CircuitPython API Reference
- Learn Adafruit: CircuitPython Essentials
- Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle



Adatlapok és dokumentáció:

- **MCP23 017/MCP23S17:** adatlap és termékinfo
- **STM32F411CE** adatlap és termékinfo
- **STM32F411xC/E** Family Reference Manual
- **WeAct Studio:** STM32F4x1 MiniF4



Példaprogramok

multiple_leds – több led villogtatása

asyncio_3ledblink – asyncio multitasking

ledswitch – nyomógomb pergésmentesítése

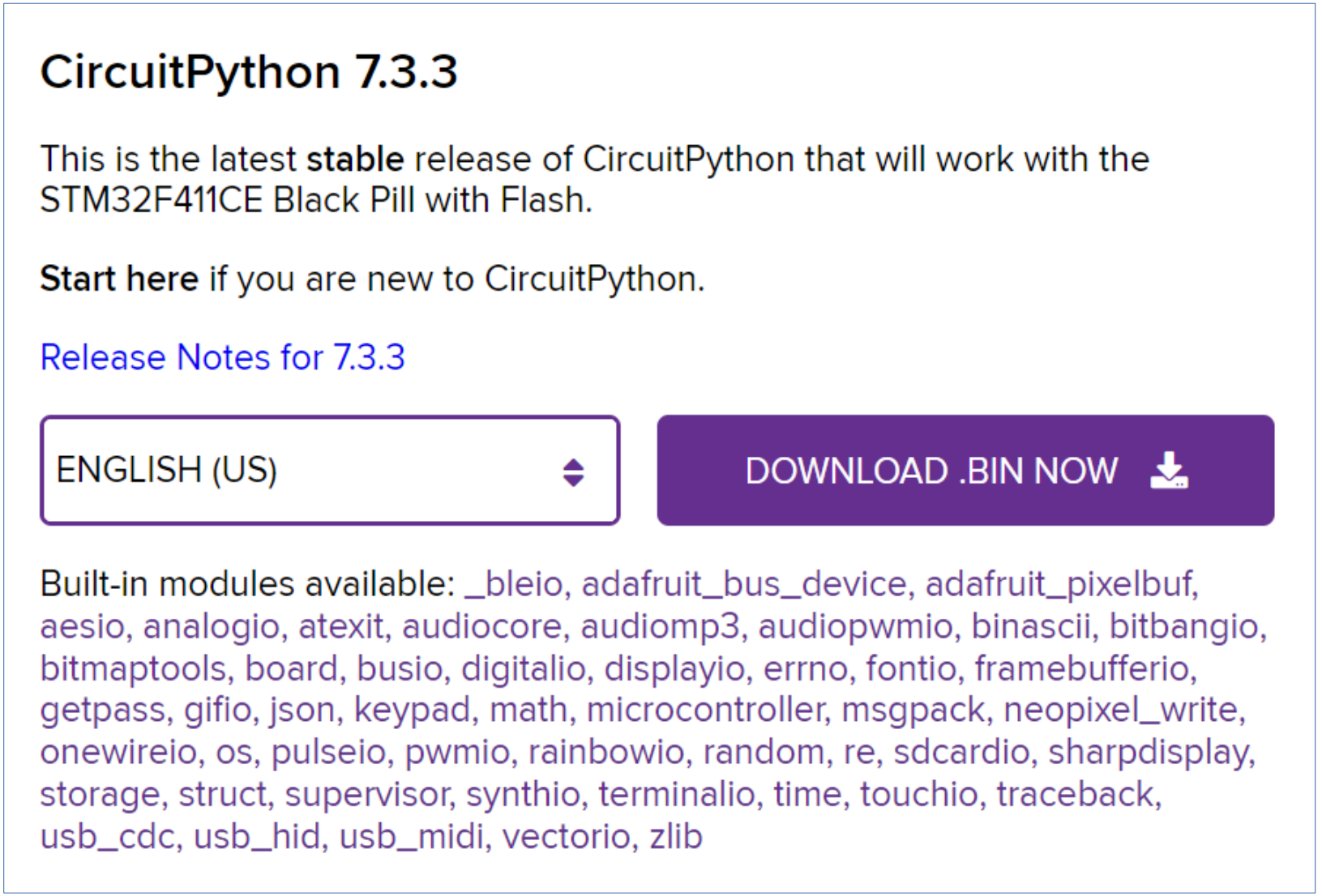
i2c_scan – az I2C busz felderítése

mcp23017_demo – egyszerű I/O kezelés

mcp23017_multitasking – két LED és három nyomógomb kezelése kooperatív többfeladatos rendszerben

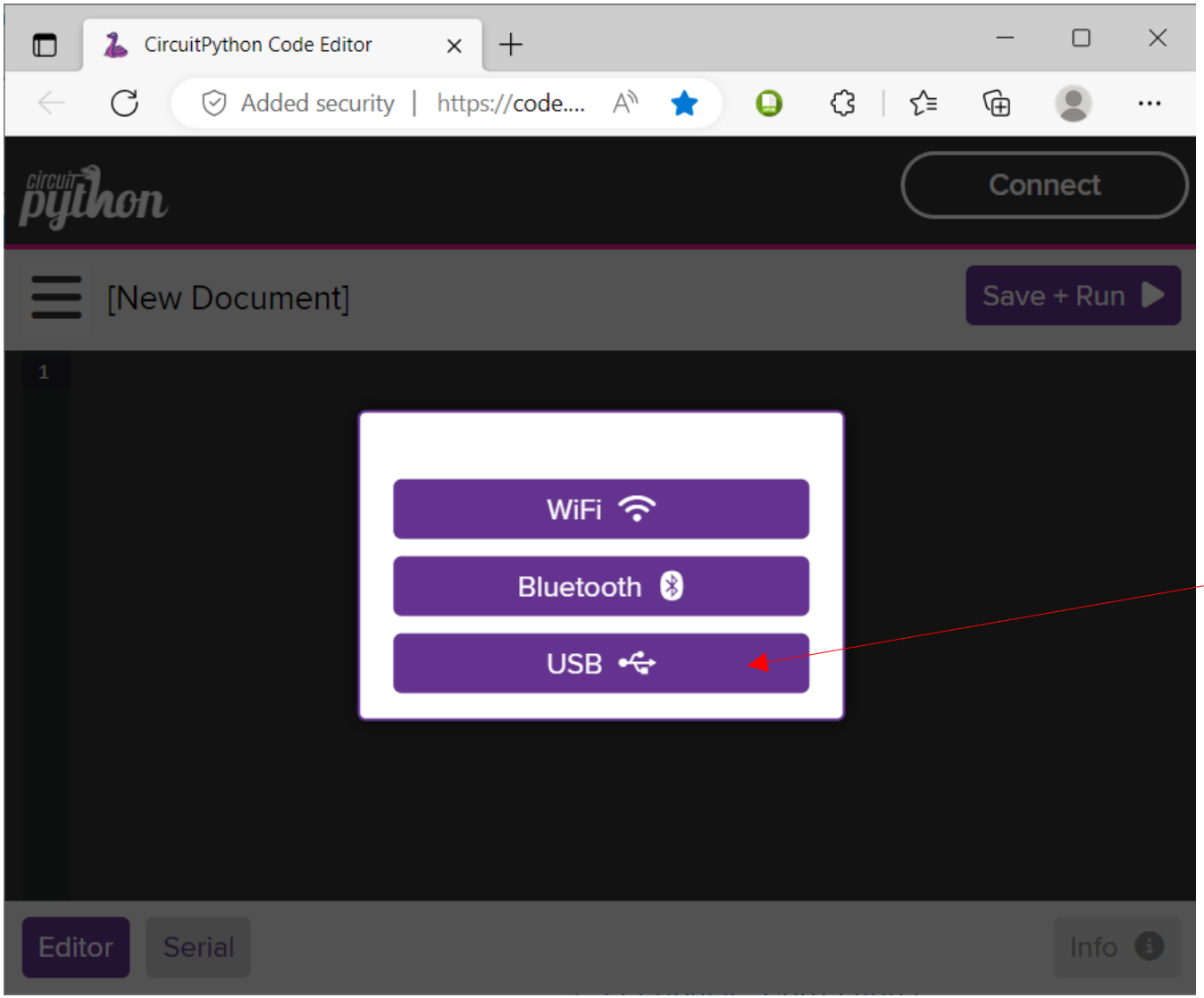
CircuitPython for STM32F411CE Black Pill with Flash

- A jelenlegi legújabb stabil firmware: 7.3.3
- Link: https://circuitpython.org/board/stm32f411ce_blackpill_with_flash/



CircuitPython online kódszerkesztő

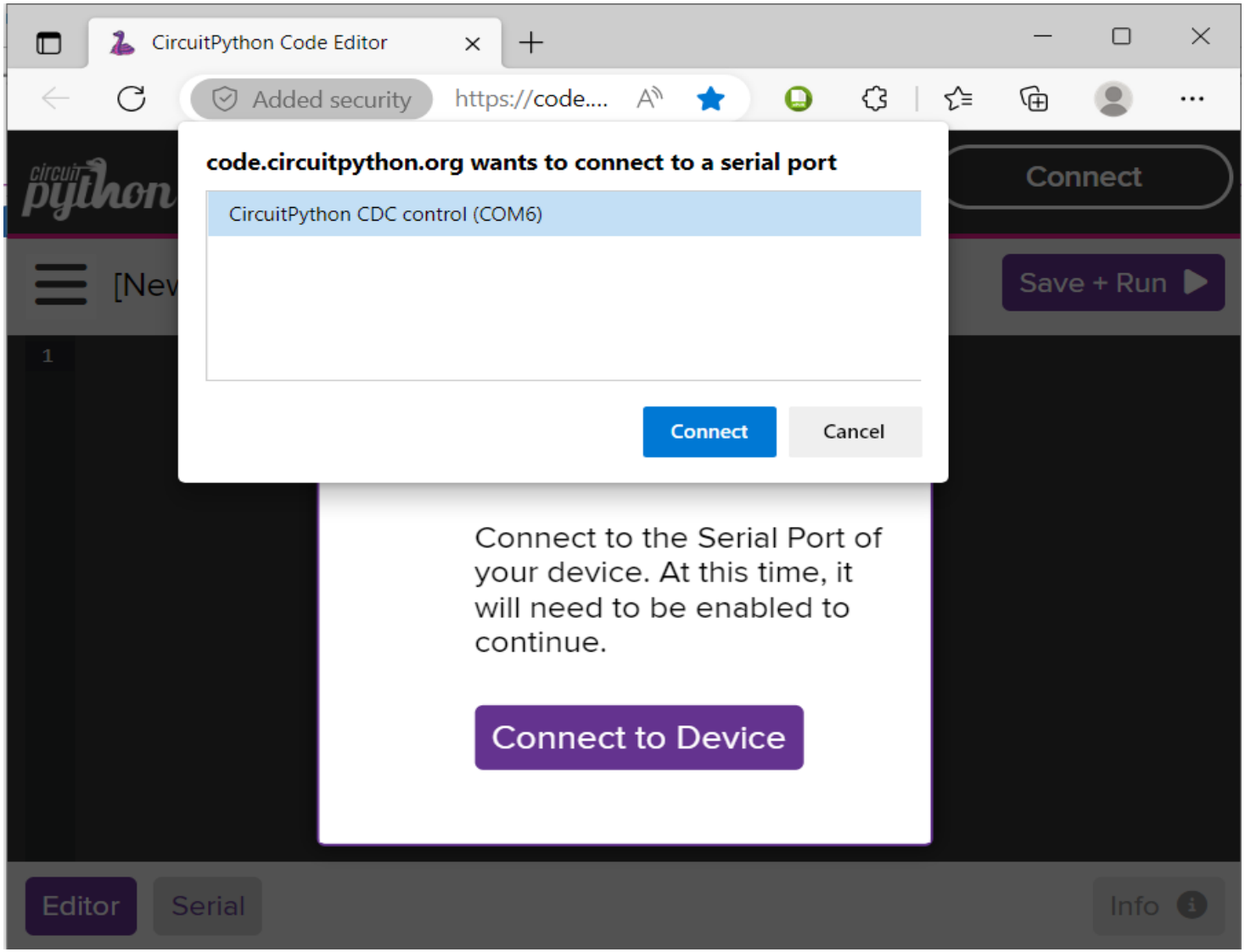
- Leírás: CircuitPython Web Workflow Code Editor Quick Start
- Belépés: https://code.circuitpython.org/



Ezt választjuk

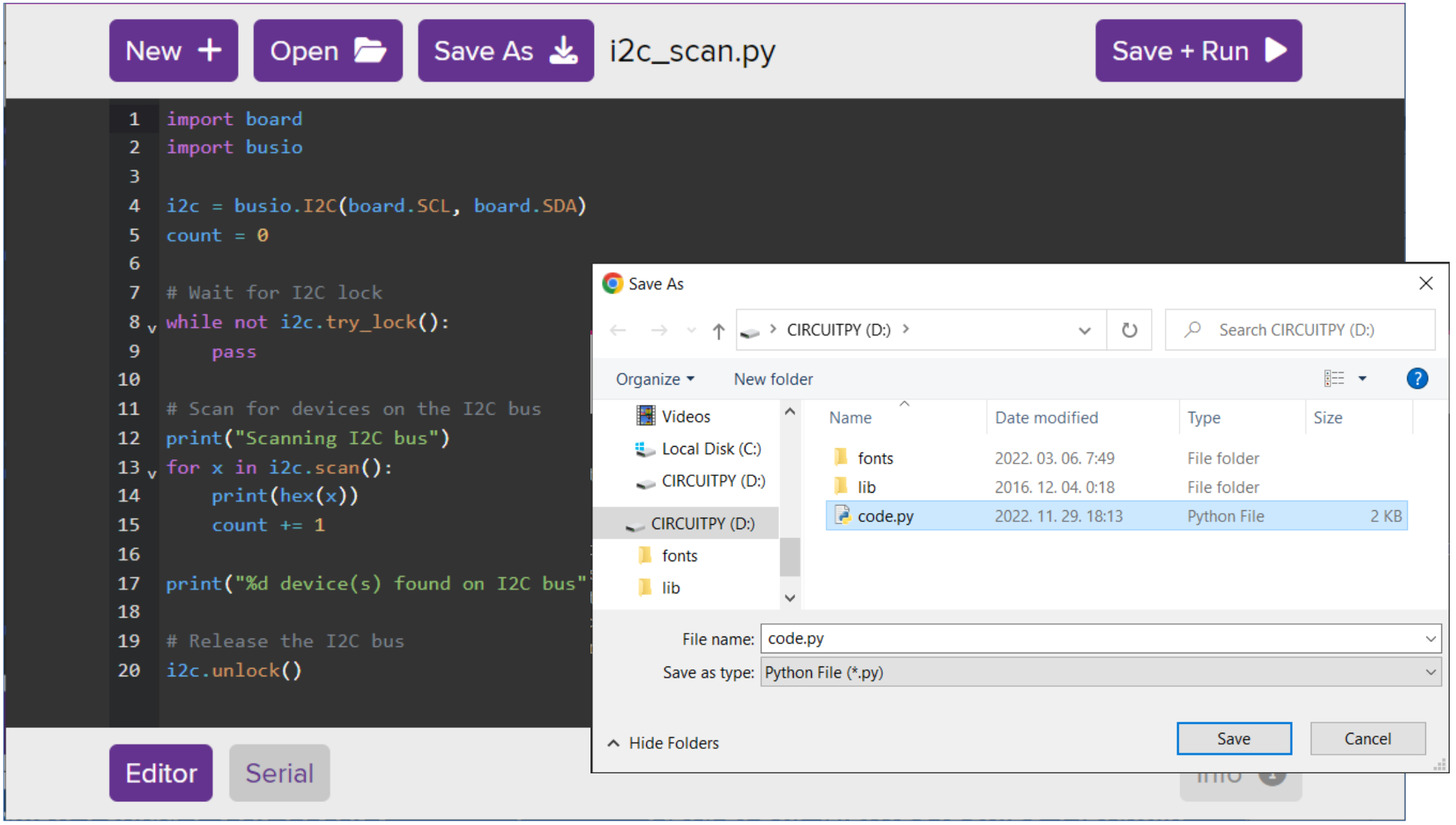
CircuitPython online kódszerkesztő

- A csatlakoztatott eszközt válasszuk ki a listán, majd **Connect** (csak Chromium alapú böngészőn működik



CircuitPython online kódszerkesztő

- Az „Ask where to save each file before downloading” opció legyen bekapcsolva a böngészőn és a CIRCUITPY:\\code.py fájlba mentsünk!

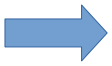


Többfeladatos programok CircuitPython-ban

- Tutorial: **Tim C.: Multi-tasking with CircuitPython**
- A cél több eszköz, vagy több feladat egyidejű kezelése
- Az első lépés a blokkoló várakozások kiiktatása. Például a LED villogtatását így írhatjuk át:

```
import time
import digitalio
import board

led = digitalio.DigitalInOut(board.LED)
led.direction = digitalio.Direction.OUTPUT

while True:
    led.value = True
    time.sleep(0.5)
    led.value = False
    time.sleep(0.5)
```

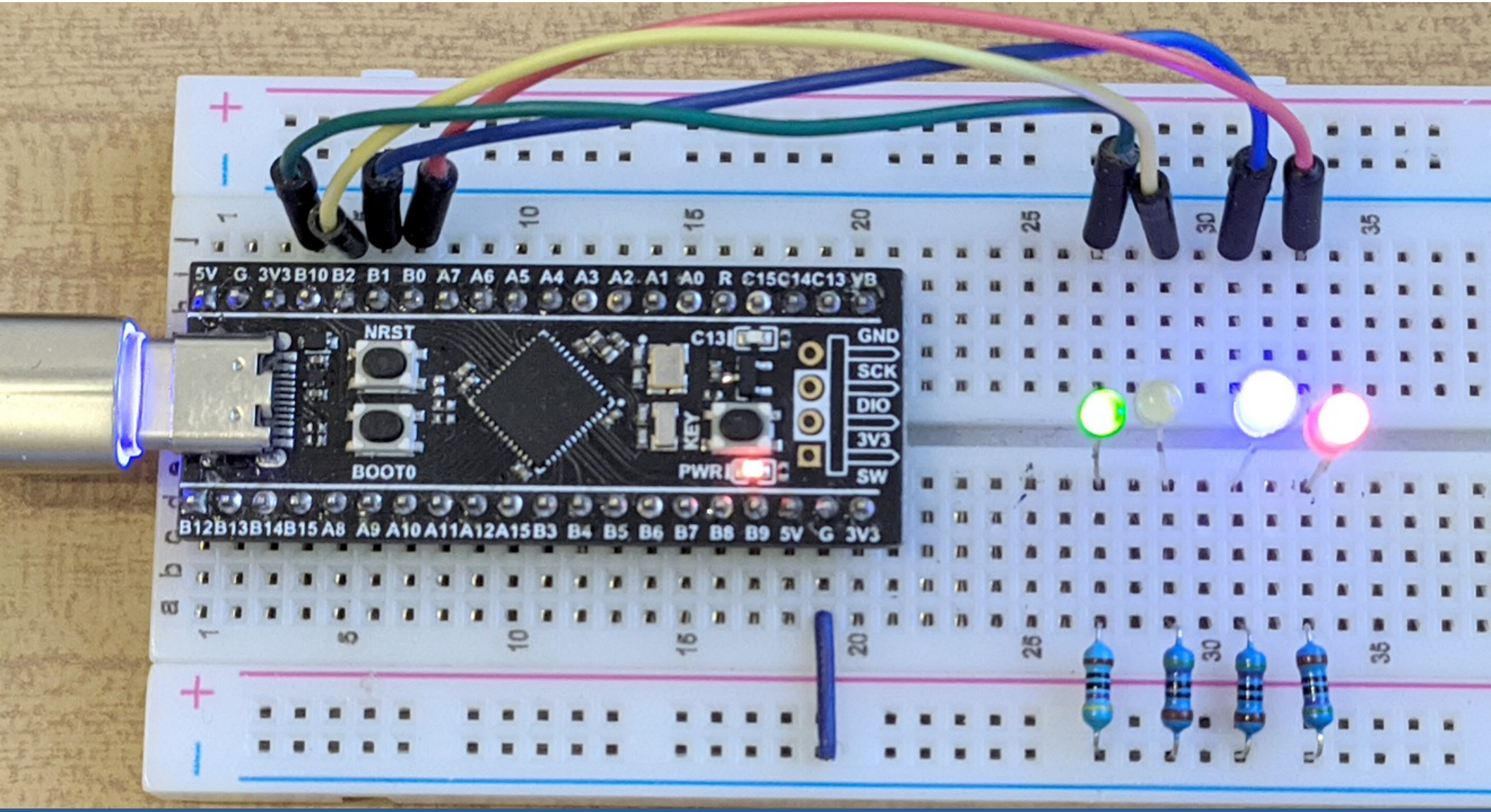
Blokoló várakozások



```
import time
import digitalio
import board
LAST_BLINK_TIME = -1
led = digitalio.DigitalInOut(board.D13)
led.direction = digitalio.Direction.OUTPUT
while True:
    now = time.monotonic()
    if not led.value:
        if now >= LAST_BLINK_TIME + 0.5:
            led.value = True
            LAST_BLINK_TIME = now
    if led.value:
        if now >= LAST_BLINK_TIME + 0.5:
            led.value = False
            LAST_BLINK_TIME = now
```

multiple_leds.py

- A LED-ek a **B0, B1, B2** és **B10** kivezetésekre vannak kötve, egy-egy 470 Ω-os ellenálláson keresztül



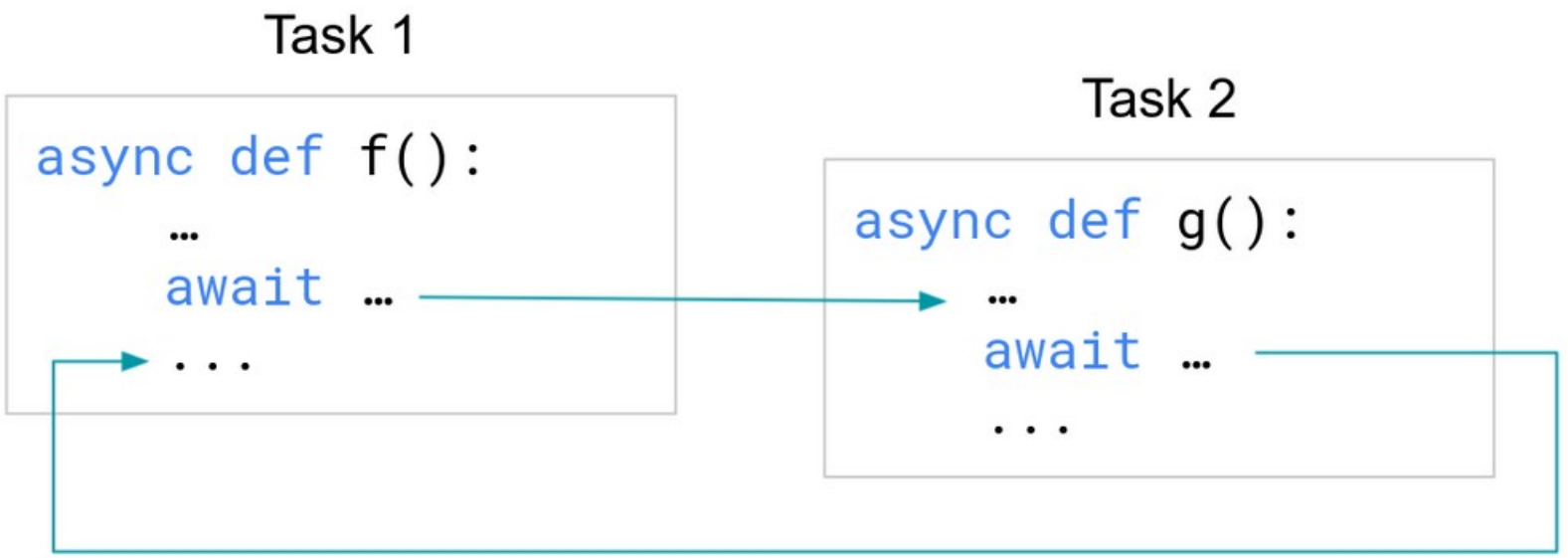
multiple_leds.py

- Több LED egyidejű villogtatását pl. így oldhatjuk meg

```
import time
import board
import digitalio
BLINK_LIST = [
    { "ON": 0.40, "OFF": 0.25, "PREV_TIME": -1, "PIN": board.B0, },
    { "ON": 0.25, "OFF": 0.25, "PREV_TIME": -1, "PIN": board.B1, },
    { "ON": 0.35, "OFF": 0.25, "PREV_TIME": -1, "PIN": board.B2, },
    { "ON": 0.50, "OFF": 0.25, "PREV_TIME": -1, "PIN": board.B10,}  ]
for led in BLINK_LIST:
    led["PIN"] = digitalio.DigitalInOut(led["PIN"])
    led["PIN"].direction = digitalio.Direction.OUTPUT
while True:
    now = time.monotonic()
    for led in BLINK_LIST:
        if led["PIN"].value is False:
            if now >= led["PREV_TIME"] + led["OFF"]:
                led["PREV_TIME"] = now
                led["PIN"].value = True
        if led["PIN"].value is True:
            if now >= led["PREV_TIME"] + led["ON"]:
                led["PREV_TIME"] = now
                led["PIN"].value = False
```

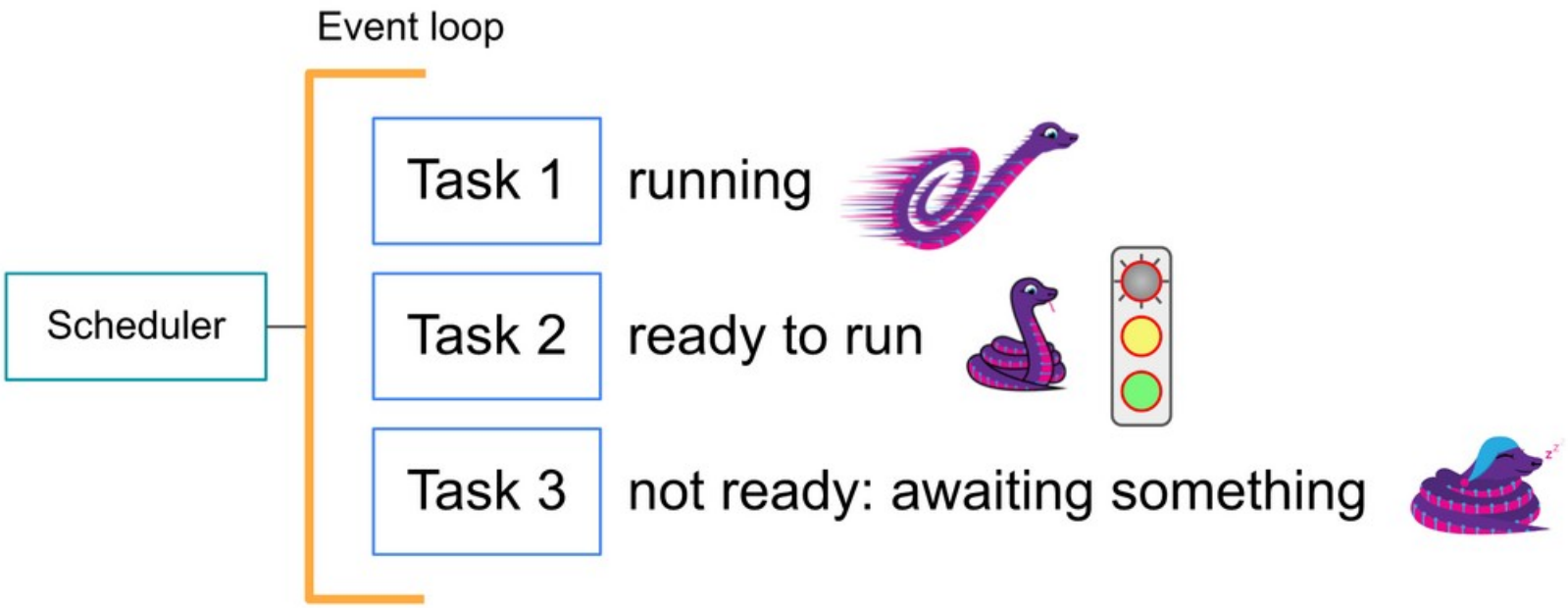
Az asycio könyvtár használata

- Tutorial: Dan Halbert: Cooperative Multitasking in CircuitPython with asyncio
- Az **asyncio** könyvtár által kínált új megközelítésben feladatokra (task) osztjuk a programot, s ezek várakozáskor átadják egymásnak a vezérlést (kooperatív többfeladatúság)
- A feladatok **async** módosító kulcsszóval definiált korutinok, amelyekben a feltételektől függő várakozásokat az **await** kulcszóval együtt kell megadni
- A használathoz telepíteni kell az adafruit_ayncio és az adafruit_ticks könyvtárakat



Az asycio könyvtár használata

- A feladatok lehetnek éppen futó, futásra kész, vagy várakozó állapotban

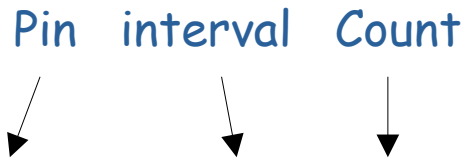


- A nem blokkoló várakozás **await asyncio.sleep(time)** hívással történik
- A feladatokat a **main()** task indítja és fogja össze
  - **task1 = asyncio.create_task(name(parameters))** – feladat indítása
  - **await asyncio.gather(task1, task2, ...)** – feladatok végének összevárása

asyncio_ledblink.py

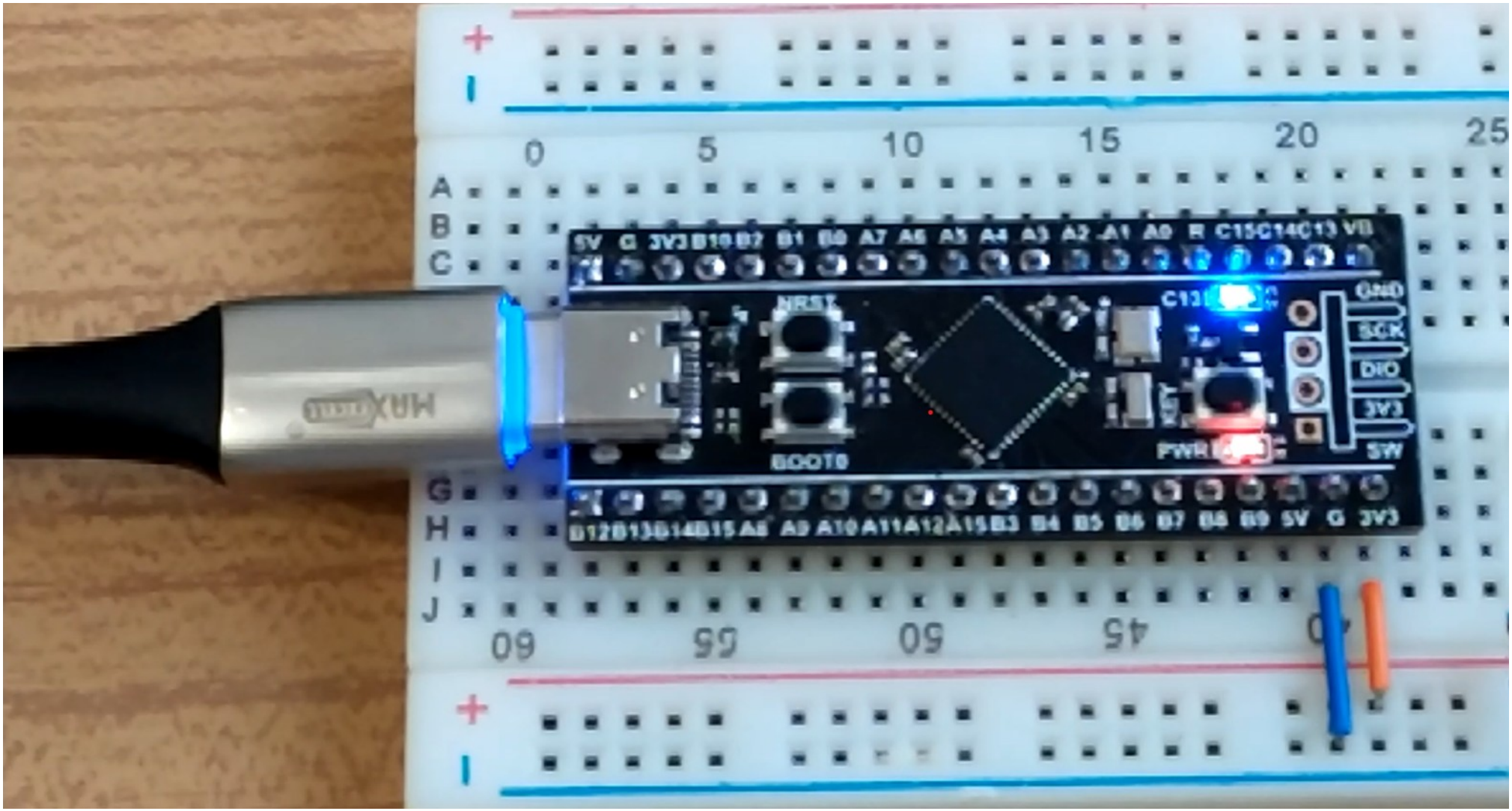
- A beépített LED-et villogtatjuk, megadott számszor felvillantva
- Figyeljük meg a kulcsszavak és az asyncio tagfüggvények használatát!

```
import asyncio
import board
import digitalio

async def blink(pin, interval, count):     # Don't forget the async!
    with digitalio.DigitalInOut(pin) as led:
        led.switch_to_output(value=False)
        for _ in range(count):
            led.value = True
            await asyncio.sleep(interval)  # Don't forget the await!
            led.value = False
            await asyncio.sleep(interval)

async def main():  # Don't forget the async!
    led_task = asyncio.create_task(blink(board.C13, 0.25, 10))
    await asyncio.gather(led_task)         # Don't forget the await!
    print("done")

asyncio.run(main())
```



asyncio_ledblink.py futási eredmény

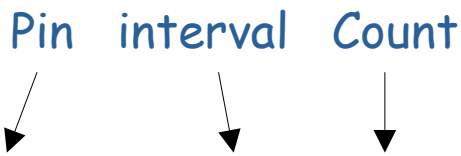
- A beépített LED-et villogtatjuk (board.C13)



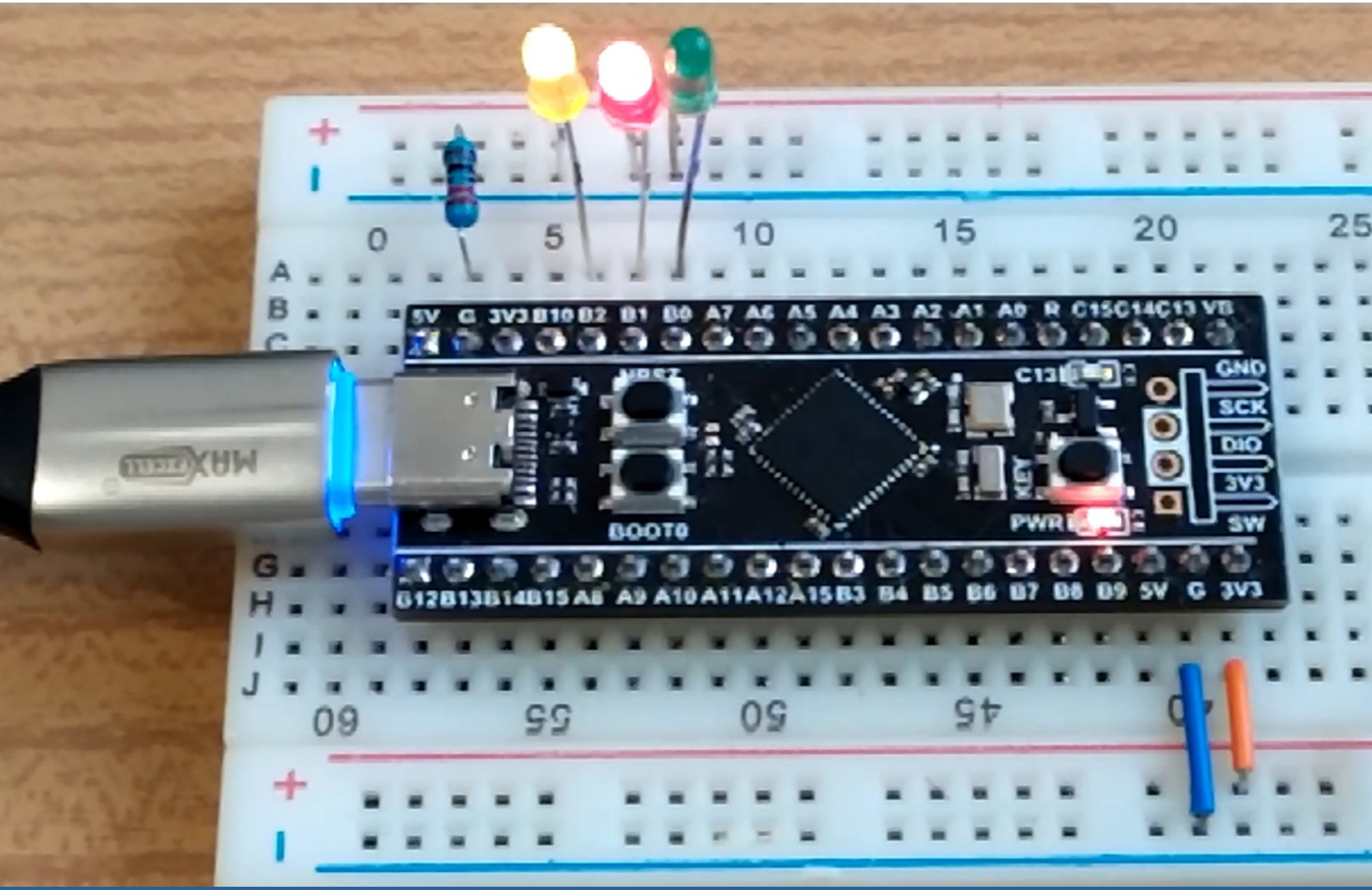
asyncio_3ledblink.py

- Egyidejűleg három LED-et villogtatunk, megadott számszor villantva

```
import asyncio
import board
import digitalio

async def blink(pin, interval, count):
    with digitalio.DigitalInOut(pin) as led:
        led.switch_to_output(value=False)
        for _ in range(count):
            led.value = True
            await asyncio.sleep(interval)
            led.value = False
            await asyncio.sleep(interval)

async def main():
    led1_task = asyncio.create_task(blink(board.B0, 0.25, 10))
    led2_task = asyncio.create_task(blink(board.B1, 0.1, 20))
    led3_task = asyncio.create_task(blink(board.B2, 0.5, 10))

    await asyncio.gather(led1_task, led2_task, led3_task)
    print("done")

asyncio.run(main())
```



asyncio_3ledblink.py futási eredménye



Nyomógomb pergésmentesítése

- A digitális bemenetek pergésmentesítését kényelmesen megoldhatjuk az adafruit_debouncer könyvtár segítségével (a könyvtárat telepíteni kell)

Lab01/ledswitch.py

```
from digitalio import *
import time, board
led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.drive_mode = DriveMode.OPEN_DRAIN
led.value = True

button = DigitalInOut(board.A0)
button.direction = Direction.INPUT
button.pull = Pull.UP

while True:
    while button.value:
        pass
    led.value = not led.value
    time.sleep(0.02)
    while not button.value:
        pass
    time.sleep(0.02)
```

Lab08/ledswitch.py

```
from digitalio import *
import time, board
from adafruit_debouncer import Debouncer

led = DigitalInOut(board.LED)
led.direction = Direction.OUTPUT
led.drive_mode = DriveMode.OPEN_DRAIN
led.value = True

button = DigitalInOut(board.A0)
button.direction = Direction.INPUT
button.pull = Pull.UP
key = Debouncer(button)

while True:
    key.update()
    if key.fell:
        led.value = not led.value
    time.sleep(0.02)
```

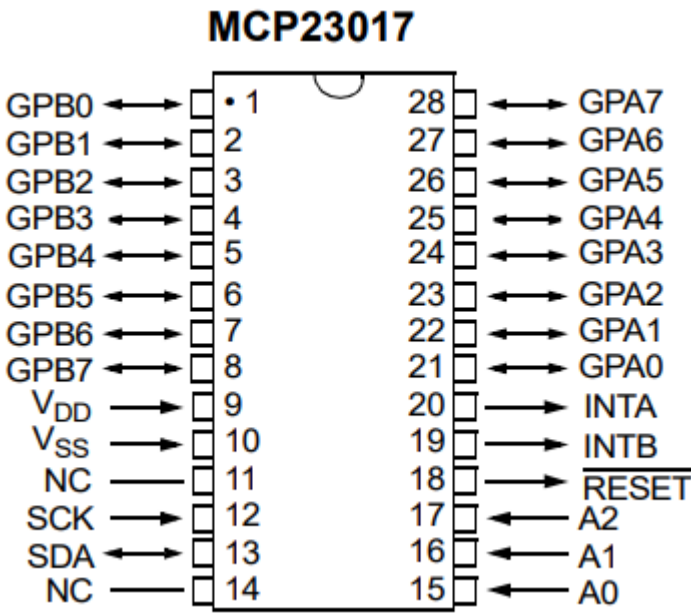
adafruit_debouncer osztályok, tulajdonságok és metódusok

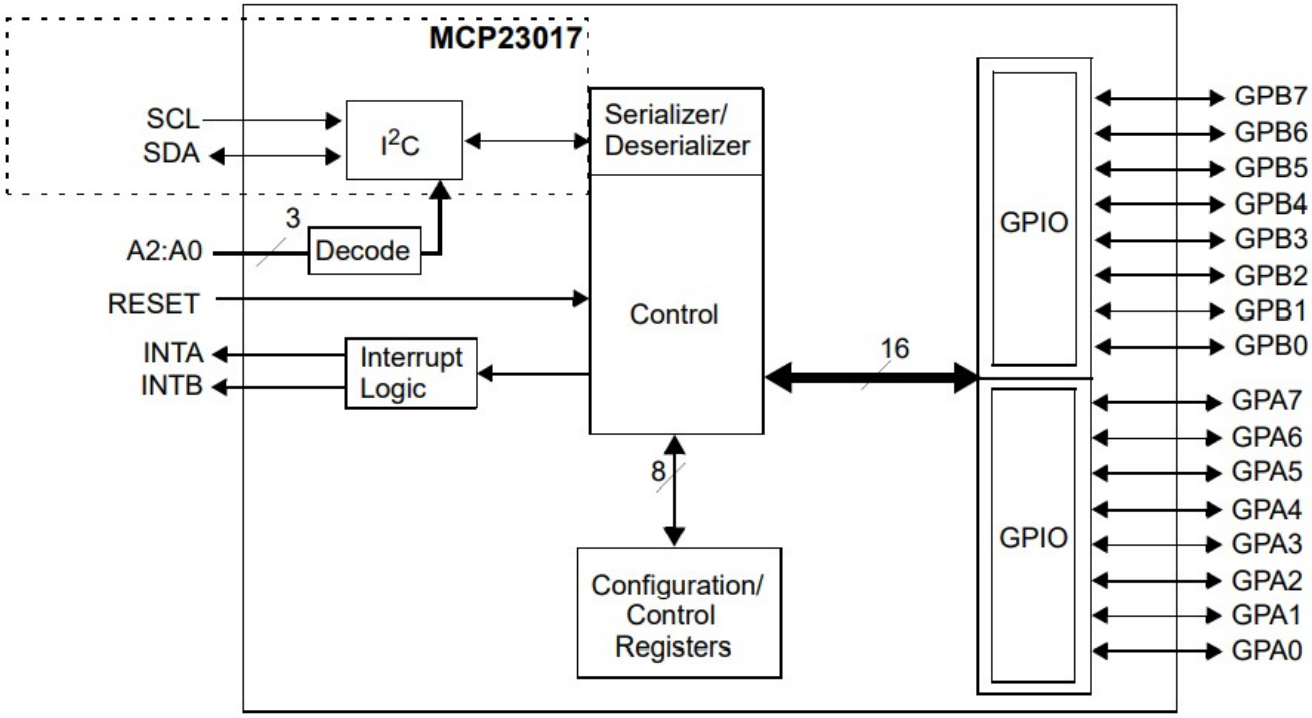
- class **adafruit_debouncer.Button**(pin, short_duration_ms, long_duration_ms, value_when_pressed)
  - **update** – az állapot frissítése
  - **pressed** – lenyomás esetén True
  - **released** – felengedett álalpotban True
  - **long_pressed** – hosszú lenyomás esetén True
  - **short_count** – a rövid lenyomások száma
- class **adafruit_debouncer.Debouncer**(io, interval)
  - **interval** – pergésmentesítési idő *(default: 0.01 s)*
  - **update** – az állapot frissítése
  - **fell** – magas/alacsony átmenet történt
  - **rose** – alacsony/magas átmenet történt
  - **value** – a jelenlegi pergésmentesített állapot
  - **last_duration** – a legutóbbi átmenet óta eltelt idő (s)

Az MCP23017 periféria bővítő

Az MCP23017 periféria bővítő

- A Microchip gyártmányú **MC23017** IC egy 16 bites kétirányú portbővítő, ami az I2C buszra csatlakozik (100, 400 és 1700 kHz)
- Három címvonallal nyolcféle cím állítható be (0x20-0x27)
- A 2x8 bites portok megszakításkérő jelet is adnak
- Konfigurálható a megszakításkérő jel forrása, a 8, vagy 16 bites mód, a bemenő jel polaritása és a belső felhúzás
- Tápfeszültség: 1.8–5.5 V
- Hardver RESET





Az MCP23017 periféria bővítő

- **Bájt módú** címzésnél nincs automatikus cím inkrementálás, `IOCON.BANK=0` esetén azonban 16 bites kiolvasás van: kiolvasáskor a regiszterek **A** és **B** fele váltakozik
- **Szekvenciális módnál** a cím minden kiolvasás után automatikusan inkrementálódik

FIGURE 3-4: I²C CONTROL BYTE FORMAT

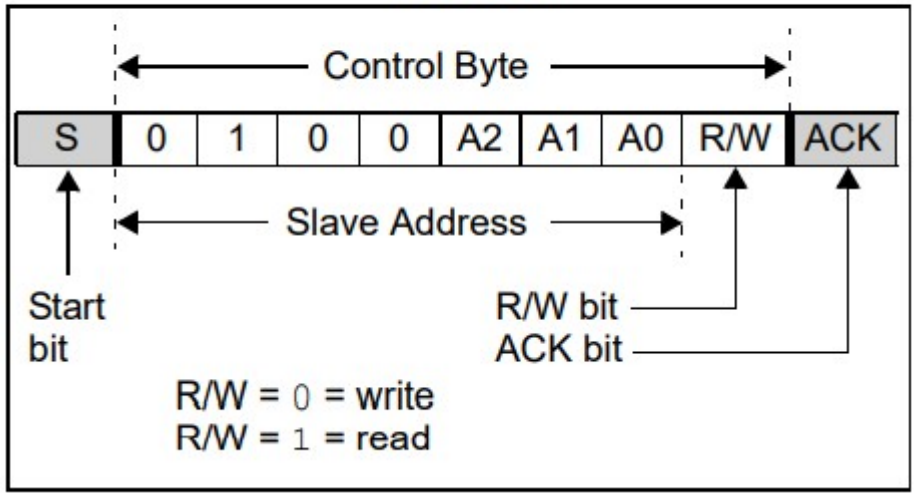


TABLE 3-1: REGISTER ADDRESSES

Address IOCON.BANK = 1	Address IOCON.BANK = 0	Access to:
00h	00h	IODIRA
10h	01h	IODIRB
01h	02h	IPOLA
11h	03h	IPOLB
02h	04h	GPINTENA
12h	05h	GPINTENB
03h	06h	DEFVALA
13h	07h	DEFVALB
04h	08h	INTCONA
14h	09h	INTCONB
05h	0Ah	IOCON
15h	0Bh	IOCON
06h	0Ch	GPPUA
16h	0Dh	GPPUB
07h	0Eh	INTFA
17h	0Fh	INTFB
08h	10h	INTCAPA
18h	11h	INTCAPB
09h	12h	GPIOA
19h	13h	GPIOB
0Ah	14h	OLATA
1Ah	15h	OLATB

Az MCP23017 periféria bővítő

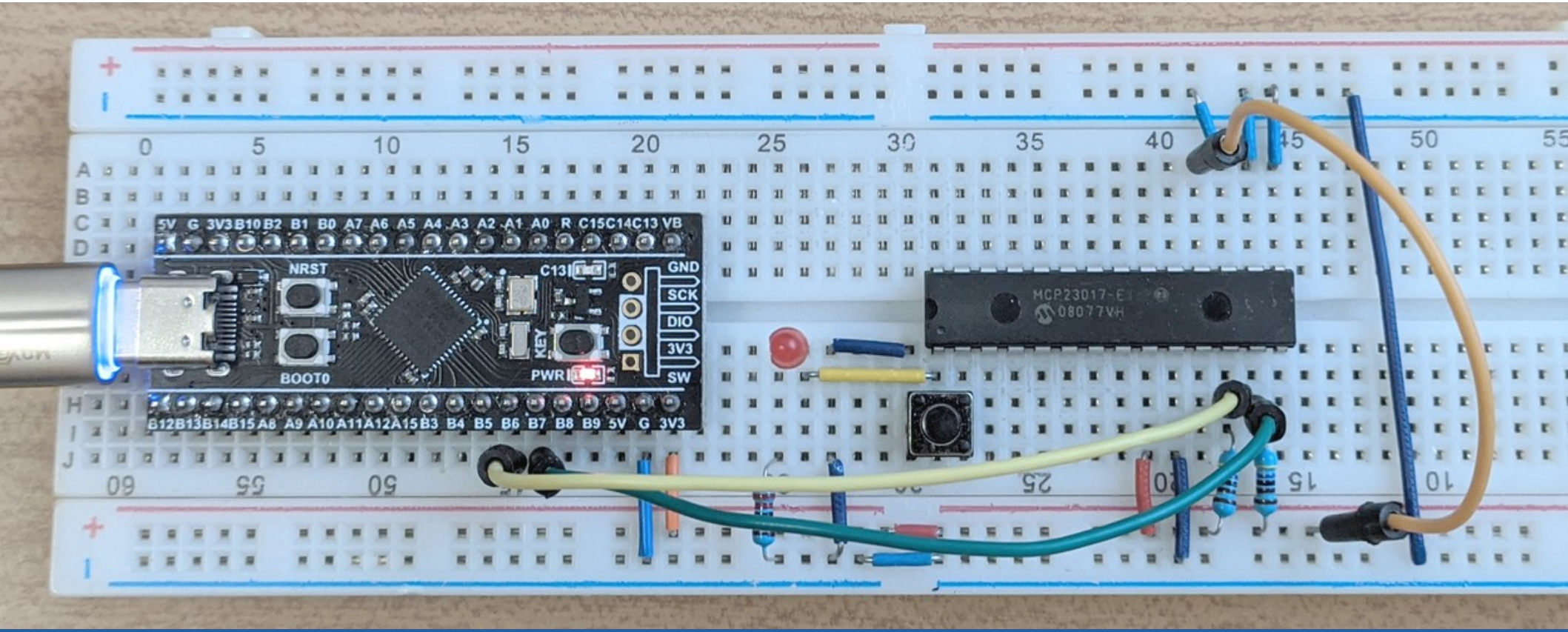
A regiszterkészlet:

- **IODIRA/B** – adatáramlási irány (0: kimenet, 1: bemenet)
- **IPOLA/B** – bemeneti polaritás (0: normál, 1: inverz)
- **GPINTENA/B** – interrupt-on-change engedélyezése
- **DEFVALA/B** – összehasonlító regiszter (default value)
- **INTCONA/B** – megszakítási mód (0: az előző értékhez hasonlít, 1: az összehasonlító regiszterhez (DEFVAL) hasonlít)
- **IOCON** – konfigurációs regiszter (`BANK,MIRROR,SEQOP,INTPOL...`)
- **GPPUA/B** – belső felhúzás engedélyezése
- **INTFA/B** – megszakításkérő jelzőbitek
- **INTCAPA/B** – megszakításkor rögzített állapot
- **GPIOA/B** – portkivezetések aktuális állapota
- **OLATA/B** – kimeneti adatregiszter

Az Adafruit_MCP230xx könyvtár használata

- Telepítsük az adafruit_mcp230xx programkönyvtárat és nézzünk egy roppant egyszerű mintapéldát az alábbi tananyagból:
  Tony DiCola: Using MCP23008 & MCP23017 with CircuitPython

Pin	1 B0	2 B1	9 VDD	10 GND	12 SCL	13 SDA	15 A0	16 A1	17 A2	18 RST
Signal	LED	BTN	3.3V	GND	B6	B7	GND	GND	GND	3.3V



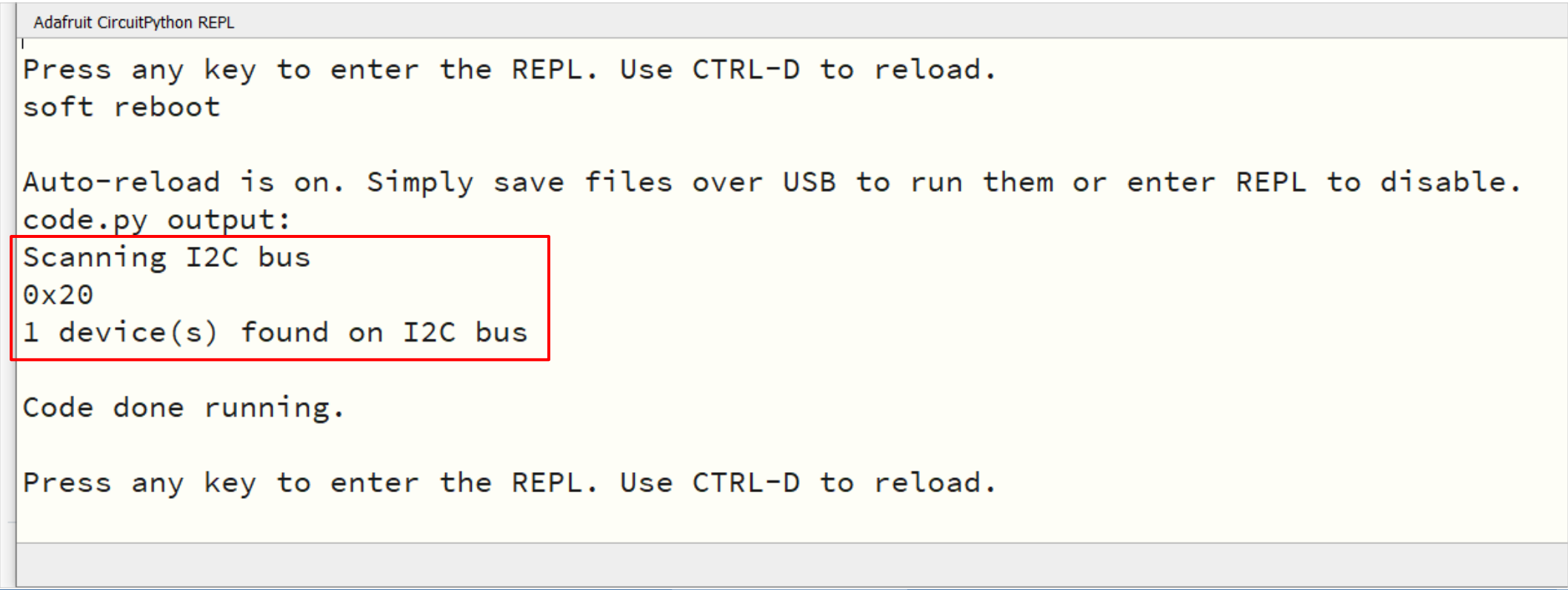
i2c_scan.py

- Az I2C busz felderítését az alábbi programmal végezhetjük
- Alapértelmezetten **SCL:** board.B6, **SDA:** board.B7 a Blackpill kártyán

```
import board
import busio

i2c = busio.I2C(board.SCL, board.SDA)
count = 0

# Wait for I2C lock
while not i2c.try_lock():
    pass

# Scan for devices on the I2C bus
print("Scanning I2C bus")
for x in i2c.scan():
    print(hex(x))
    count += 1

print("%d device(s) found on I2C bus" % count)

# Release the I2C bus
i2c.unlock()
```

i2c_scan.py futási eredmény

- Ha az **A0, A1, A2** címvonalakat alacsony szintre (GND) kötjük, akkor az **MCP23 017** az alapértelmezett **0x20** I2C címre hallgat
- Az **i2c_scan.py** futtatásakor is ezt az értéket kaptuk, ahogy azt az alábbi ábra mutatja
- Ne feledkezzünk meg az I2C busz felhúzó ellenállásairól!
  (az előző oldalon bemutatott kapcsolásban 2 x 4,7 kΩ szerepel)



mcp23017_demo.py – 2/1.

- Az **MCP23017** 1. kivezetésére (**GPIOB0**) egy LED-et kötünk és kb. 1 Hz frekvenciával villogtatjuk
- A 2. kivezetés (**GPIOB1**) és a GND közé egy nyomógombot kötünk, melynek állapotát 1 másodpercenként kiíratjuk

```
import time
import board
import busio
import digitalio
# from adafruit_mcp230xx.mcp23008 import MCP23008
from adafruit_mcp230xx.mcp23017 import MCP23017

i2c = busio.I2C(board.SCL, board.SDA)               # Initialize the I2C bus

# Create an instance of either the MCP23008 or MCP23017 class depending on which chip you're using:
# mcp = MCP23008(i2c)  # MCP23008
mcp = MCP23017(i2c)  # MCP23017
# mcp = MCP23017(i2c, address=0x21)  # MCP23017 w/ A0 set

# Now call the get_pin function to get an instance of a pin on the chip
# For the MCP23017 you specify a pin number from 0 to 15 for the GPIOA0...GPIOA7, GPIOB0...GPIOB7 pins
pin0 = mcp.get_pin(8)   # B0
pin1 = mcp.get_pin(9)   # B1
```

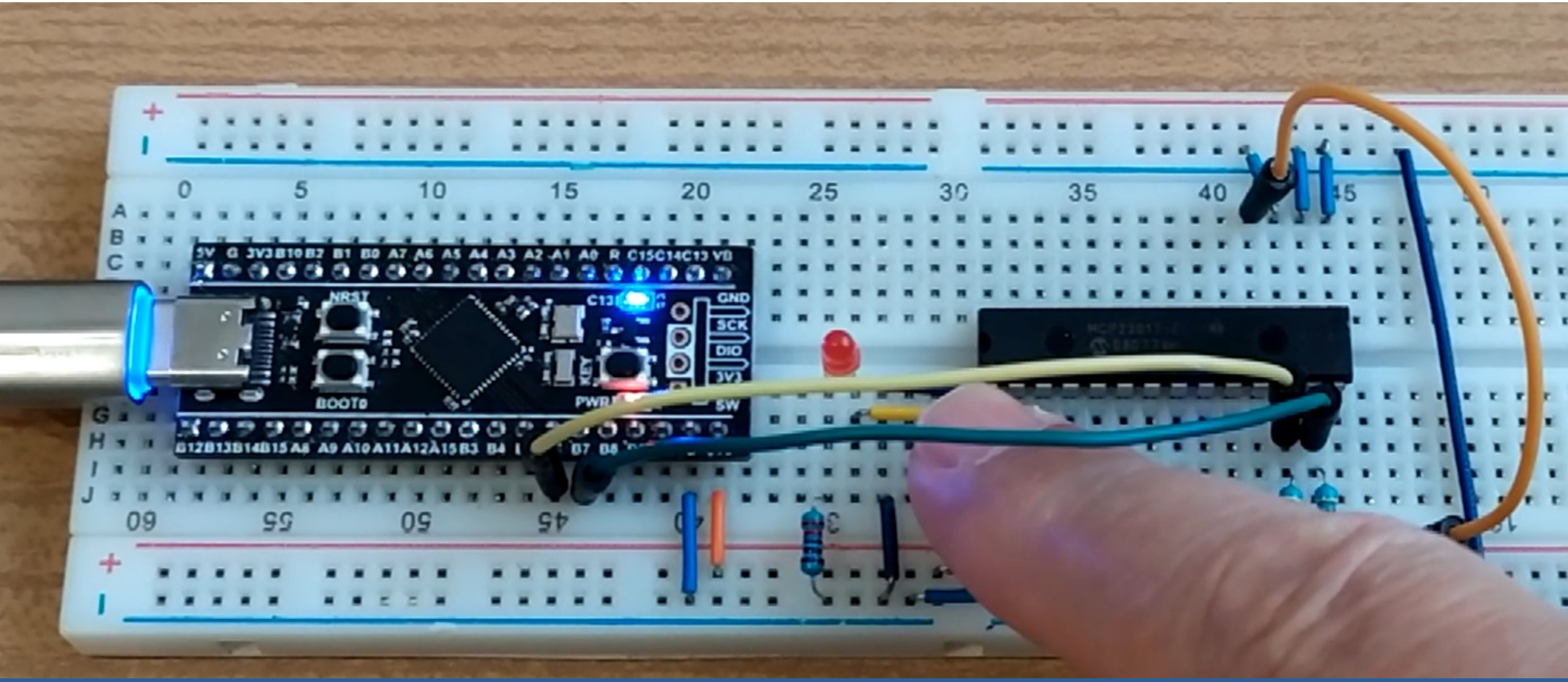
mcp23017_demo.py – 2/2.

```
pin0.switch_to_output(value=True)          # Setup pin0 as an output that's at a high logic level
pin1.direction = digitalio.Direction.INPUT  # Setup pin1 as an input with a pull-up resistor enabled
pin1.pull = digitalio.Pull.UP

# Now loop blinking the pin 0 output and reading the state of pin 1 input.
while True:
    # Blink pin 0 on and then off.
    pin0.value = True
    time.sleep(0.5)
    pin0.value = False
    time.sleep(0.5)
    # Read pin 1 and print its state.
    print("Pin 1 is at a high level: {0}".format(pin1.value))
```

```
Adafruit CircuitPython REPL
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: False
Pin 1 is at a high level: False
Pin 1 is at a high level: False
Pin 1 is at a high level: False
Pin 1 is at a high level: True
Pin 1 is at a high level: True
Pin 1 is at a high level: False
Pin 1 is at a high level: False
Pin 1 is at a high level: False
Pin 1 is at a high level: True
Pin 1 is at a high level: True
```

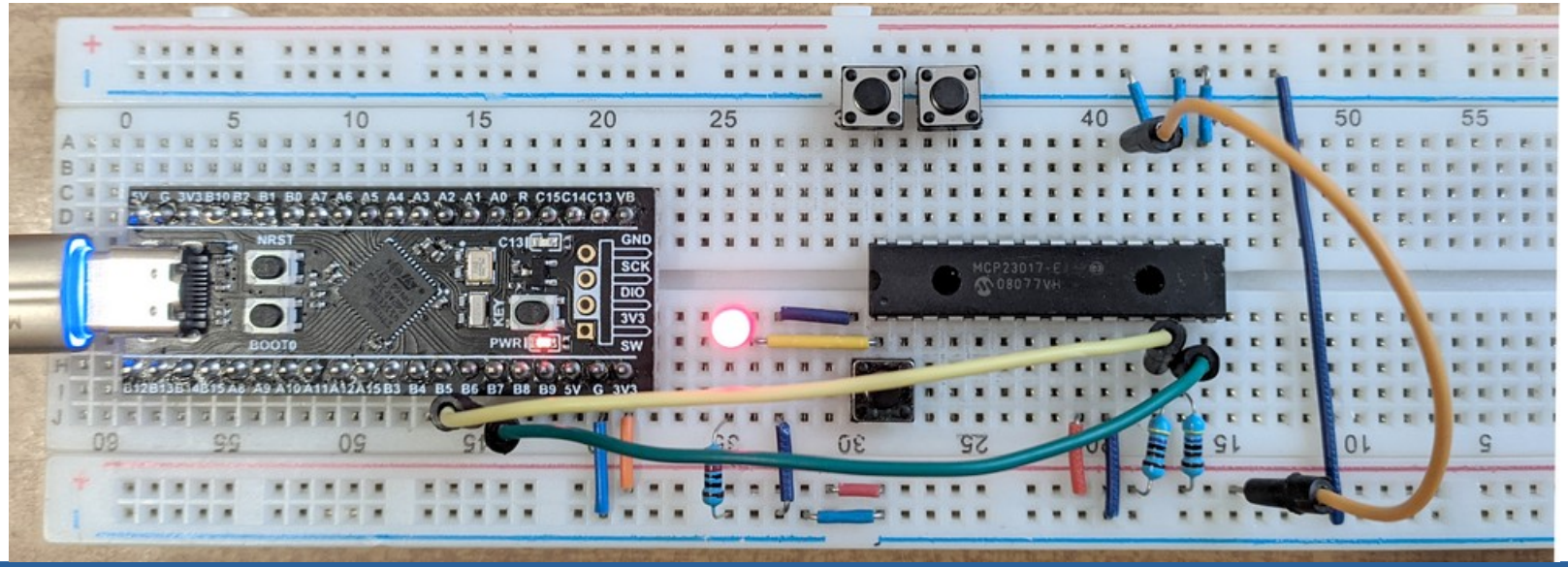
mcp23017_demo2.py

- Az előző program módosított változatában kiíratás helyett a beépített LED segítségével jelezzük ki a nyomógomb állapotát és a villogtatás ütemét is gyorsabbra állítottuk (kb. 2 Hz)
- A nyomógomb „reakcióideje” így is elég lassú…



mcp23017_multitasking.py

- Kombináljuk az **MCP23017** kezelését az **asyncio**-val megvalósítható kooperatív többfeladatos módszerrel!
- Előnyök: gyorsabban lehet reagálni a nyomógombok lenyomására és akár több LED-et, vagy más kimenetet is kezelhetünk egyidejűleg
- Az alábbiakban az előző kapcsolást két további nyomógombbal bővítjük (**GPIOA7** és **GPIOA4** kivezetések és a GND közé kötve)
- A **GPIOB0**-ra kötött LED villogtatásának időállandóját a fenti nyomógombokkal léptetjük, 0.1 s lépésközökkel fel, illetve le.
- A **GPIOB1**-re kötött nyomógombbal pedig a beépített LED-et (board.C13) kapcsolgatjuk be- és ki



mcp23017_multitasking.py – 3/1.

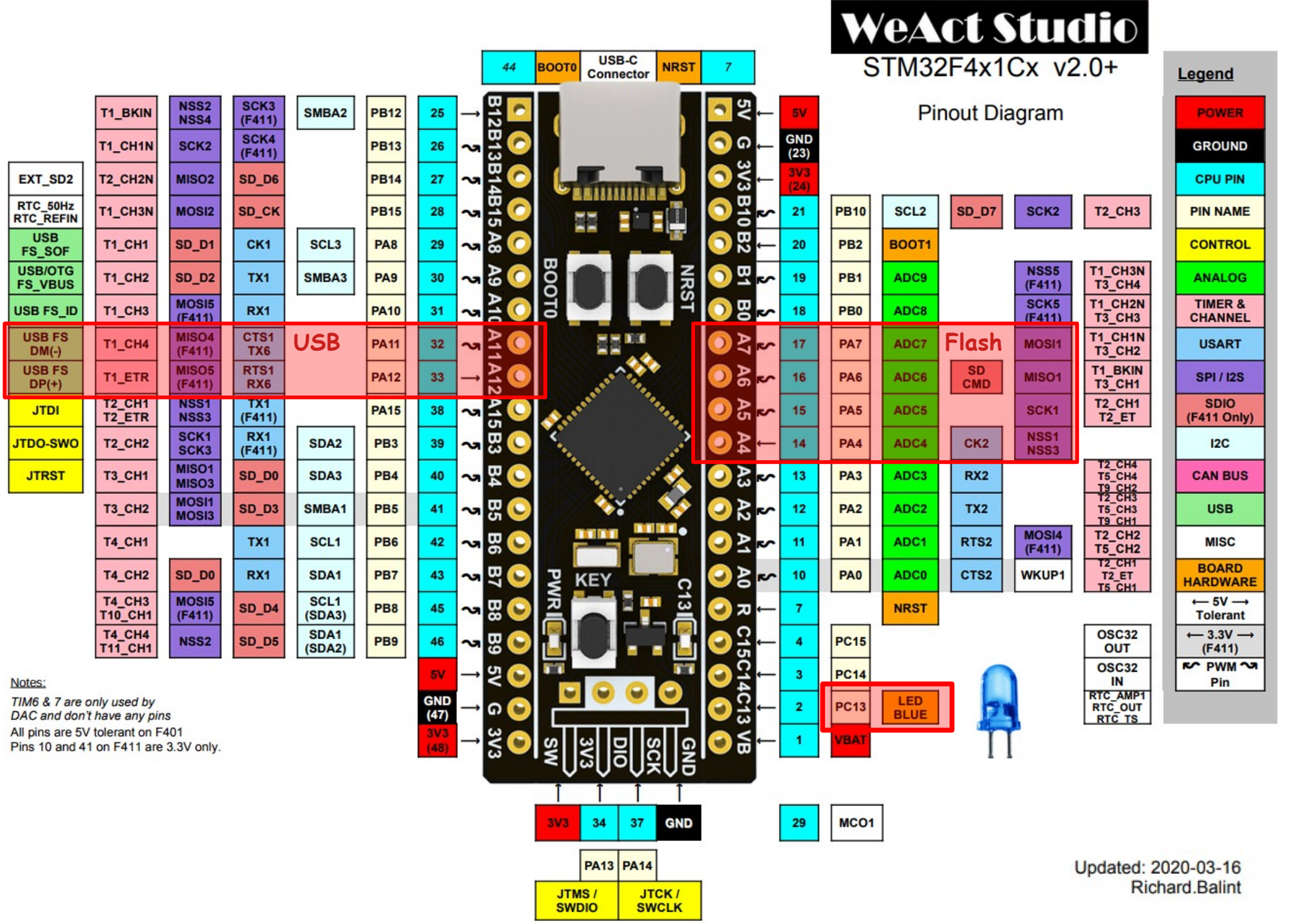
```
import asyncio
import board
import digitalio
import busio
from adafruit_debouncer import Debouncer
from adafruit_mcp230xx.mcp23017 import MCP23017

class Interval:                                  #Simple class to hold an interval value

    def __init__(self, initial_interval):
        self.value = initial_interval

interval = Interval(0.25)
led = digitalio.DigitalInOut(board.LED)          # Builtin LED @ board.C13
led.direction = digitalio.Direction.OUTPUT
led.value = True

i2c = busio.I2C(board.SCL, board.SDA)            # Initialize the I2C bus:
mcp = MCP23017(i2c, address=0x20)                # MCP23017 with A0,A1,A2 grounded

pin8 = mcp.get_pin(8)  # GPB0
pin9 = mcp.get_pin(9)  # GPB1
pin7 = mcp.get_pin(7)  # GPA7
pin4 = mcp.get_pin(4)  # GPA4
```

mcp23017_multitasking.py – 3/2.

```
pin8.switch_to_output(value=True)                    # Setup pin8 as an output that's at a high logic level

# Setup pushbutton pins as input with internal pull-up enabled
pin9.direction = digitalio.Direction.INPUT           # GPIOB1
pin9.pull = digitalio.Pull.UP
pin7.direction = digitalio.Direction.INPUT           # GPIOA7
pin7.pull = digitalio.Pull.UP
pin4.direction = digitalio.Direction.INPUT           # GPIOA4
pin4.pull = digitalio.Pull.UP

key = Debouncer(pin9)                                # KEY switching builtin LED on/off
btn_up = Debouncer(pin7)                             # Button to increase interval by 0.1 s
btn_dn = Debouncer(pin4)                             # Button to decrease interval by 0.1 s

#---  TASK1: Monitor two buttons, one lengthens the interval, the other shortens it ---
async def interval_buttons():
  while True:
      btn_up.update()
      if btn_up.fell:
          interval.value += 0.1
      btn_dn.update()
      if btn_dn.fell:
          interval.value = max(0.1, interval.value - 0.1)
      await asyncio.sleep(0.02)                      # debouncing delay
```

mcp23017_multitasking.py – 3/3.

```
#---  TASK2:   „key” button switches the builtin LED on  and off ---
async def led_switch():
   while True:
      key.update()
      if key.fell:
         led.value = not led.value  # Toggle builtin LED's state
      await asyncio.sleep(0.02)

#---  TASK3:   blinking the LED connected to GPIOB0 ---
async def led_blink():
   while True:
      pin8.value = True
      await asyncio.sleep(interval.value)        # Don't forget the "await"!
      pin8.value = False
      await asyncio.sleep(interval.value)        # Don't forget the "await"!

#--- This is the MAIN task ----------------------------------------
async def main():
   task1 = asyncio.create_task(interval_buttons())
   task2 = asyncio.create_task(led_switch())
   task3 = asyncio.create_task(led_blink())
   await asyncio.gather(task1, task2, task3)   # Don't forget "await"!

asyncio.run(main())
```


WeAct Studio
STM32F4x1Cx v2.0+
Pinout Diagram
Legend
POWER
GROUND
CPU PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI / I2S
SDIO (F411 Only)
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V Tolerant
3.3V (F411)
PWM Pin
USB-C Connector
BOOT0
NRST
USB
Flash
KEY
PWR
C13
SW
3V3
DIO
SCK
GND
JTMS / SWDIO
JTCK / SWCLK
LED BLUE
Notes:
TIM6 & 7 are only used by DAC and don't have any pins
All pins are 5V tolerant on F401
Pins 10 and 41 on F411 are 3.3V only.
Updated: 2020-03-16
Richard.Balint