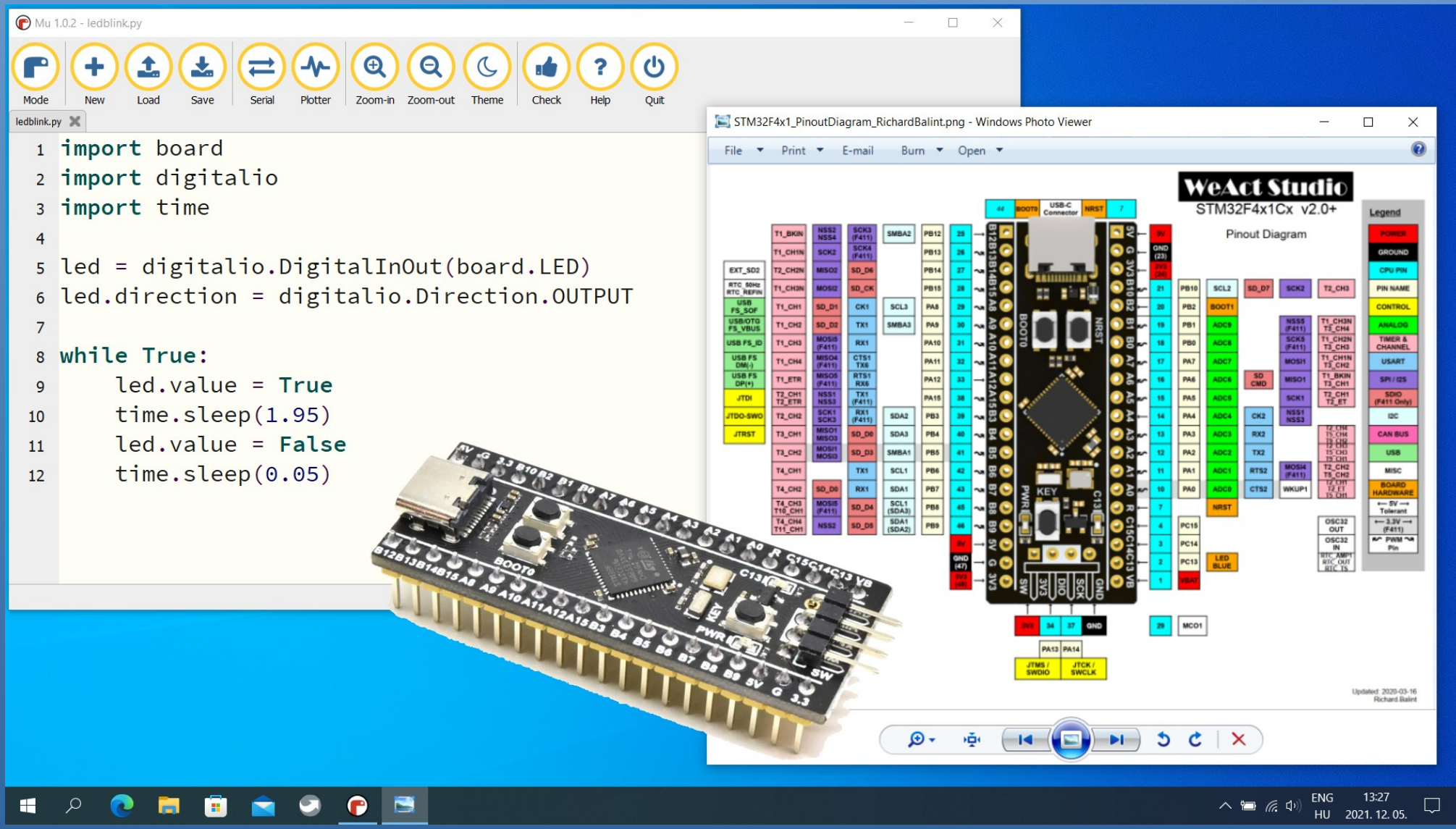
CircuitPython tanfolyam



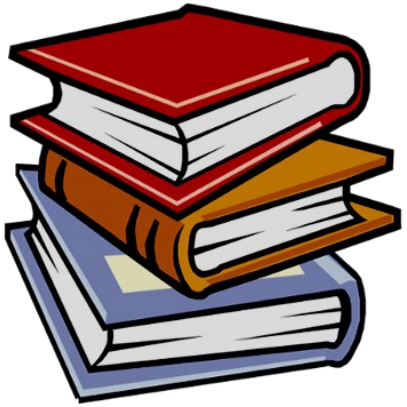
2. Analóg és digitális RGB LED-ek vezérlése

Felhasznált és ajánlott irodalom

Python:

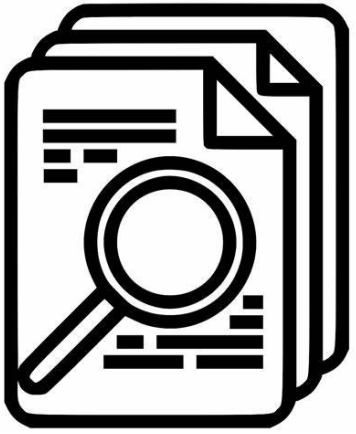
- Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével

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

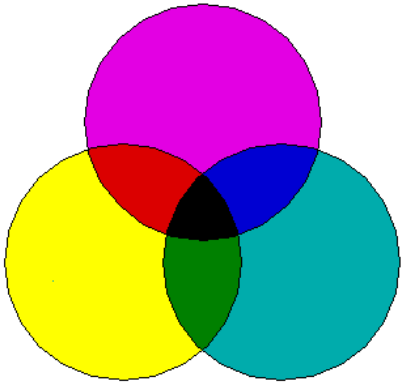
Adatlapok és dokumentáció:



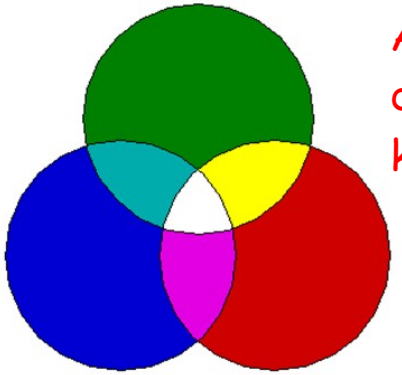
- **STM32F411CE** adatlap és termékinfo
- **STM32F411xC/E** Family Reference Manual
- **WeAct Studio:** STM32F4x1 MiniF4

Szín, színlátás, színkeverés, színtér modellek

- Az Arduino tanfolyam keretében a **2020. április 30-i** előadásban (videofelvétel, előadásvázlat, mintaprogramok) részletesen beszéltünk a színről, a színlátásról, a színkeverésről és különféle színtér modellekről, illetve az ezek között konverzióról, az alábbiakban ezeknek csak néhány részletét elevenítjük fel
- **A színkeverésnek** két módszere ismert:
  - **szubtraktív színkeverés** az emberi szemtől függetlenül, a fények természetes spektrális módosulása útján jön létre (pl. nyomdászat)
  - **additív színkeverés** az emberi látórendszerben alakul ki (pl. színes TV)



- Szubtraktív színkeverés
  (**CYMK modell,** a tinta gyengíti a fényvisszaverést bizonyos hullámhosszon)

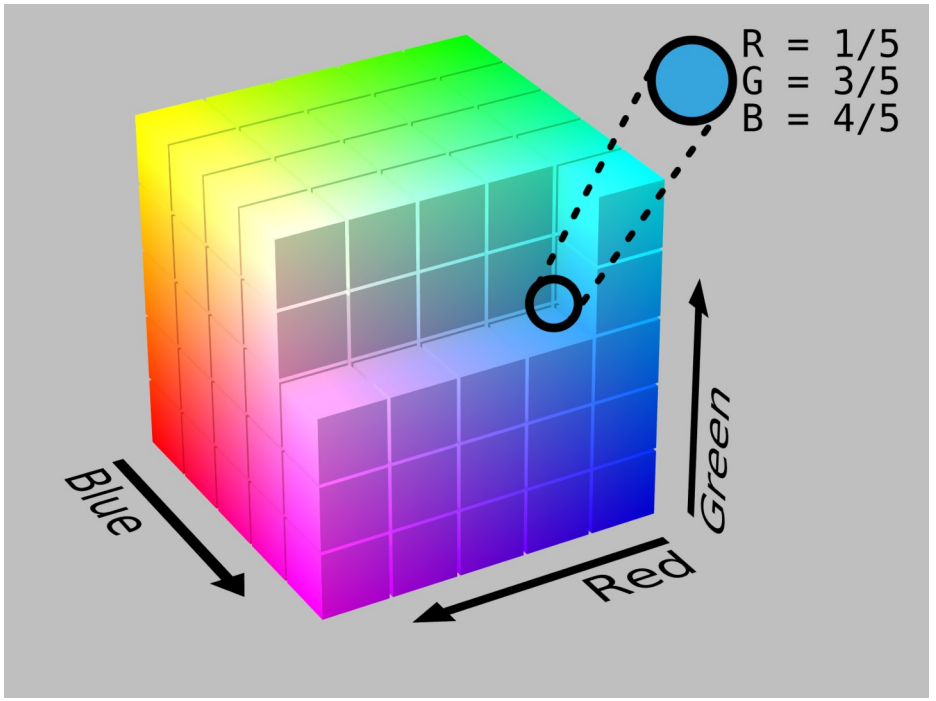


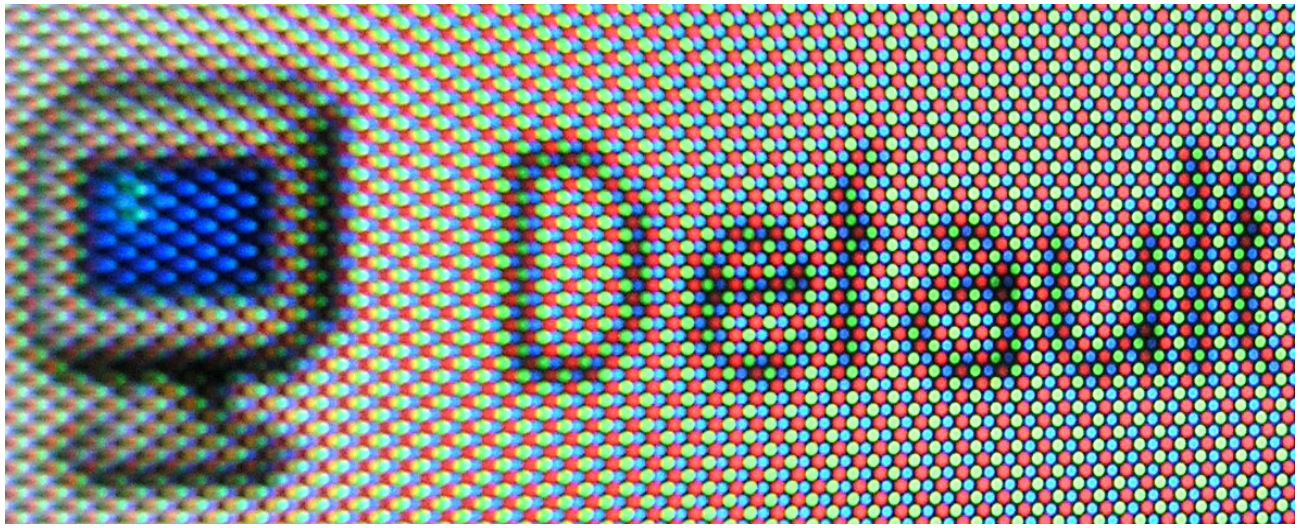
A mai előadásban ezzel kísérletezünk

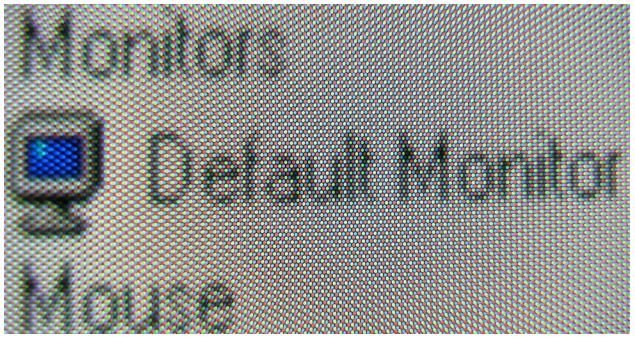
- Additív színkeverés
  (**RGB modell,** különböző hullámhosszúságú fénysugarakkal)

Az RGB színtér modell

- Az **RGB** additív színtér modellben vörös, zöld és kék összetevők keverésével próbáljuk leírni, vagy előállítani a színeket
- Minden színárnyalat az R, G és B ”koordinátákkal” jellemezhető
- Az alábbi képen egy színes monitor fényképe látható kinagyítva







Wikipedia: RGB color model,

Color spaces w RGB primaries

A HSV (vagy HSI) színtér modell

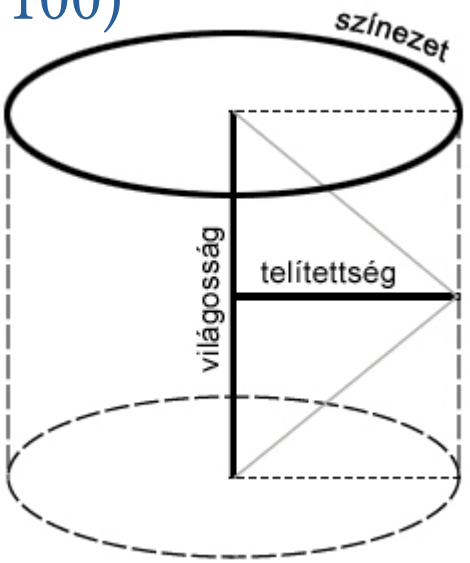
- A színérzetek jellemzőit hengerkoordináta rendszerben is ábrázolhatjuk, ahol a Φ szög a színezet (*Hue*, 0 – 360°), a tengelytől való távolság a telítettség (*Saturation*, 0 – 100), a tengelymenti távolság pedig a világosság (*Value, Intensity*, 0 – 100)
- A henger inkább kettőskúp, mert fekete, ill. fehér szín esetében nincs értelme telítettségről, vagy színezetről beszélni
- Összefüggés a HSI és az RGB értékek között:

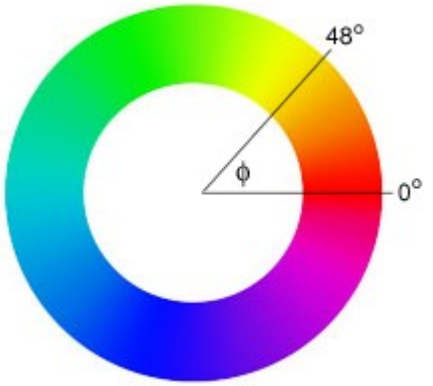
$$I = \frac{R+G+B}{3}$$

$$S = 1 - \frac{3}{(R+G+B)} \min(R,G,B)$$

$$H = \cos^{-1}\left(\frac{(R-G)+(R-B)}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right) \qquad \text{assuming } G > B$$

If $B > G$, then $H = 360 - H$.





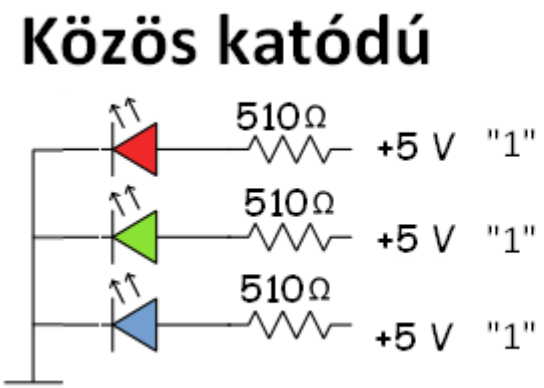
Források:

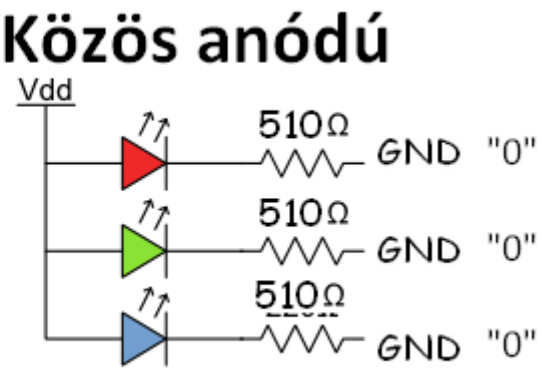
Földvári Melinda: Szín + Kommunikáció

Wikipedia: RGB color model

(analóg) RGB LED-ek

- Az RGB LED három, különböző színű (vörös, zöld, kék) LED-et tartalmaz egy közös tokban, s vagy az anódok, vagy a katódok össze vannak kötve
- Az áramkorlátozásról nekünk kell gondoskodni (pl. ellenállás)
- **Közös katód** esetén az anódokat magas szintre kell húzni
- **Közös anód** esetén a katódokat lefelé kell húzni (inverz logika)
- Kísérleteinknél elég lesz színenként 5 – 10 mA





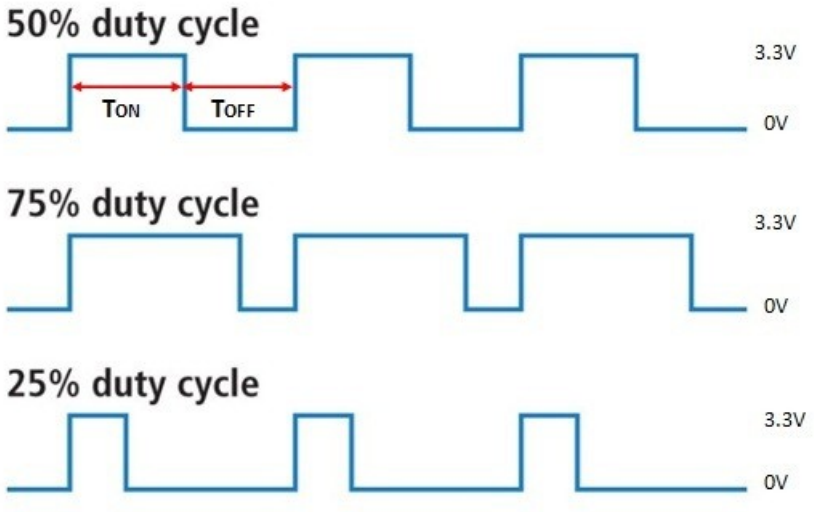
+5V helyett most
+3,3V lesz a
magas szint

Közös katód

Közös anód

Impulzus-szélesség moduláció (PWM)

- **PWM** kimenetek kezelése: **pwmio** modul **PWMOut** osztály
- A konstruktor:
  `PWMOut(pin:Pin,frequency:int,duty_cycle:int,var_freq:bool)`
  Ahol: ***pin*** – a választott kivezetés, ***frequency*** – a frekvencia (Hz), ***duty_cycle*** – a kitöltés (0 – 65 535), ***var_freq*** – változó frekvencia jelzése
- **PWM**-re használható kivezetések: **A0, A1, A2, A3, A8, A9, A15. B0, B1, B3, B4, B5, B6, B7, B8, B9, B10**
- **Tapasztalat:** a **var_freq** paraméter beállításától függetlenül az **F411CE blackpill** kártya esetében időzítőnként egyidejűleg csak egy PWM csatorna használatát engedélyezi a CircuitPython
- Legkönnyebben az előadásvázlat végén található **Pinout diagram** segítségével kereshetjük ki, hogy melyik kivezetés melyik időzítő(k)höz tartozik

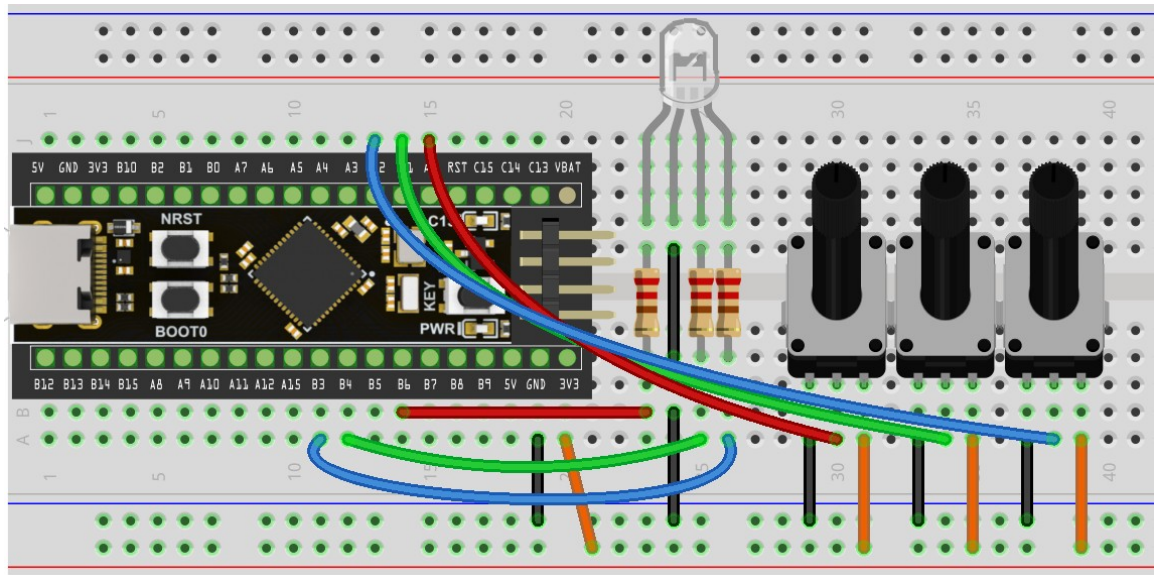


TON: Time requires for pulse to remain ON i.e. HIGH State
TOFF: Time requires for pulse to remain OFF i.e. LOW State

Kézivezérelt színkeverés

- **Kapcsolási elrendezés:**
  Itt a tápfeszültséget lustaságból a **B5** lábról adjuk, nem követendő példa!
- **B6** – Red (piros)
- **B5** – közös elektróda
- **B4** – Green (zöld)
- **B3** – Blue (kék)

A0 – 1. potméter csúszka (piros)
A1 – 2. potméter csúszka (zöld)
A2 – 3. potméter csúszka (kék)
Az áramkorlátozás itt 3 x 510 Ω, de legalább 3 x 220 Ω legyen!



Az RGB LED lábaira 510 Ω-os ellenállásokat forrasztottunk és egy tüskesorra építettük.

Így akár az **F411CE** kártya mellé is dughatjuk.

A megoldás hátránya, hogy akkor mindhárom LED árama a közös **B5** ágon folyik

manual_mixing.py

- Az **A0, A1, A2** bemenetekre kötött potméterekkel vezéreljük a **B3, B4, B5, B6** kivezetésekre kötött RGB LED-et *(**B5** a közös láb)*

```
import board
import time
from pwmio import PWMOut
from analogio import AnalogIn
from digitalio import *

potm0=AnalogIn(board.A0)
potm1=AnalogIn(board.A1)
potm2=AnalogIn(board.A2)
rled=PWMOut(board.B6, duty_cycle=0)
gled=PWMOut(board.B4, duty_cycle=0)
bled=PWMOut(board.B3, duty_cycle=0)
common=DigitalInOut(board.B5)
common.direction=Direction.OUTPUT
common.value=False # True: common anode; False: common cathode

while True:
    rled.duty_cycle = potm0.value
    gled.duty_cycle = potm1.value
    bled.duty_cycle = potm2.value
    time.sleep(0.025)
#end
```

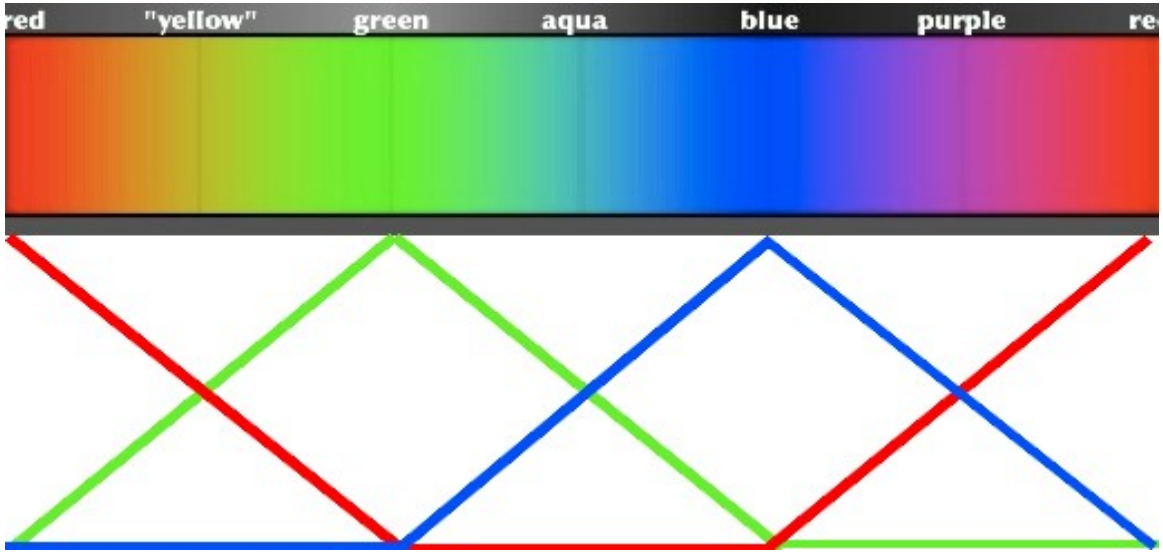
Módosítás nélkül a program közös katódú kapcsolással használható
Közös anódú RGB LED esetén **`common.value = True`** írandó és a közös láb 3,3V-ra kötendő. A PWM kimeneteket is invertálni kellene, de elég a potmétereket fordított irányba tekerni.

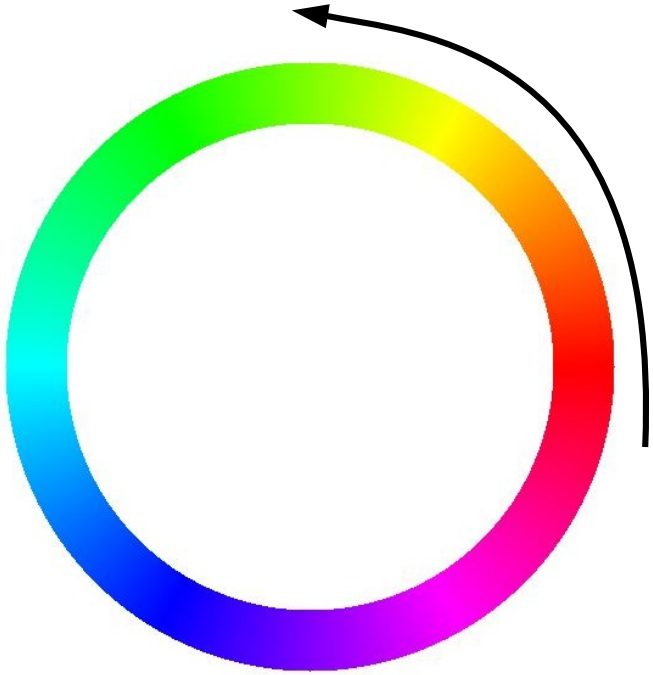
Ezek a sorok csak a lusta kapcsolás miatt kellenek

Itt érdemes volna bevetni egy gördülő simítást az ingadozások csökkentésére

Automatikus színátmenet

- Állítsuk elő a színkerék folytonos színátmeneteit!
- Három színátmenettel oldhatjuk meg, például:
  - Piros ⟶ zöld átmenet
  - Zöld ⟶ kék átmenet
  - Kék ⟶ piros átmenet
- A színkomponensek intenzitásának változásait az alábbi ábrán láthatjuk





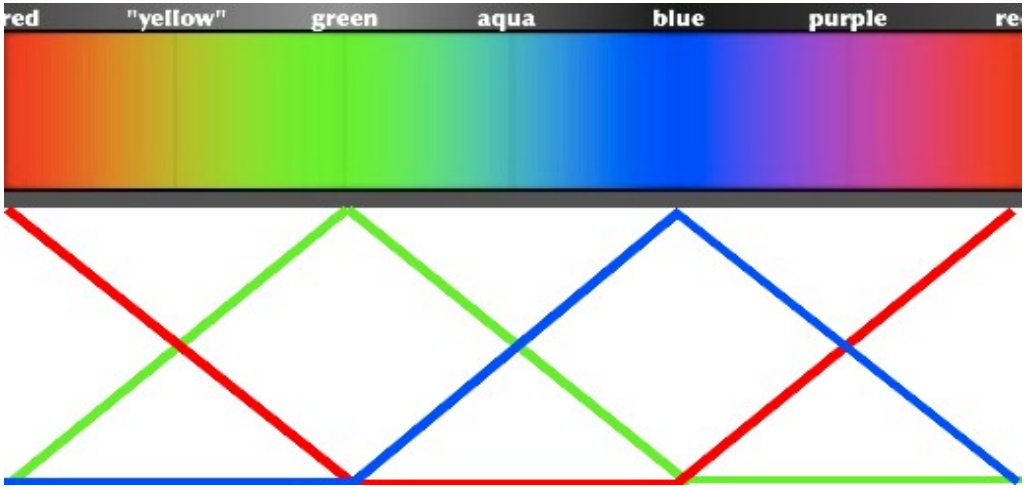
A kapcsolás ugyanaz, mint a 10. oldalon, csak a potmétereket most nem használjuk

RGB_colorwheel.py

```
import board
import time
from pwmio import PWMOut
from digitalio import *
rled=PWMOut(board.B6, duty_cycle=0)
gled=PWMOut(board.B4, duty_cycle=0)
bled=PWMOut(board.B3, duty_cycle=0)
common=DigitalInOut(board.B5)
common.direction=Direction.OUTPUT
common.value=True        # True: common anode; False: common cathode
r=b=g=0

def setRGB(red,green,blue):
    rled.duty_cycle = 65535 - 257*red     # Inverting for common anode
    gled.duty_cycle = 65535 - 257*green   # Inverting for common anode
    bled.duty_cycle = 65535 - 257*blue    # Inverting for common anode
    time.sleep(0.025)

while True:
    for g in range(256):
        r = 255 - g
        setRGB(r,g,b)
    for b in range(256):
        g = 255 - b
        setRGB(r,g,b)
    for r in range(256):
        b = 255 - r
        setRGB(r,g,b)
```

Itt a közös anódú kapcsoláshoz igazított programot mutatjuk be

Közös katódú kapcsoláshoz a megjelölt részeket kell átírni…



RGBLED osztályt készítünk

- Az RGB LED-ek kényelmesebb kezeléséhez most definiálunk egy objektumosztályt, amellyel egységbe foglalhatjuk egy RGB LED jellemzőit (hová vannak kötve a lábai, katód-, vagy anódvezérelt-e, s hogy éppen milyen színben világít)
- A kényelmesebb kezelés érdekében azt is lehetővé tesszük, hogy egy RGB LED színét egyetlen utasítással állíthassuk be (vagy RGB számhármassal, vagy egyetlen 24 bites színkóddal), például:

```
rgbled.color = (255, 100, 30)
```

vagy

```
rgbled.color = 0xFF641E
```

- Az RGBLED osztályt az Adafruit_CircuitPython_RGBLED könyvtár mintájára készítjük el, némileg leegyszerűsítve azt, s közben tanulunk egy-két programozói fogást és Pythonból is kupálódunk...
- A kapcsolás ugyanaz, mint az előző programoknál, de a közös elektródát most **B5** helyett „tisztességesen” a tápfeszültségre (közös katód esetén pedig GND-re) kötjük!

rgbled_class.py 3/1.

- Definiáljunk egy **RGBLED** objektumosztályt a LED kezelésére!
  A LED bekötése: B3=kék, B4=zöld, B6=piros szín vezérlése

```
import board
import time
from pwmio import PWMOut
from rainbowio import colorwheel

class RGBLED:
    def __init__(self, red_pin, green_pin, blue_pin, invert_pwm=False):
        self._rgb_led_pins = [red_pin, green_pin, blue_pin]
        for i in range(3):
            self._rgb_led_pins[i] = PWMOut(self._rgb_led_pins[i])
            self._rgb_led_pins[i].duty_cycle = 0
        self._invert_pwm = invert_pwm
        self._current_color = (0, 0, 0)
        self.color = self._current_color

    def deinit(self):
        for pin in self._rgb_led_pins:
            pin.deinit()
        self._current_color = (0, 0, 0)

    @property
    def color(self):
        """Returns the RGB LED's current color."""
        return self._current_color
```

A Hue érték 0..255 között indexelhető, s a kiválasztott szaturált szín 24 bites kódját adja meg

Az osztálydefiníció kezdete

Az __init__() függvény példányosításkor automatikusan meghívásra kerül

Felhasznált forrás:
Adafruit_CircuitPython_RGBLED

Dekorátor

rgbled_class.py 3/2.

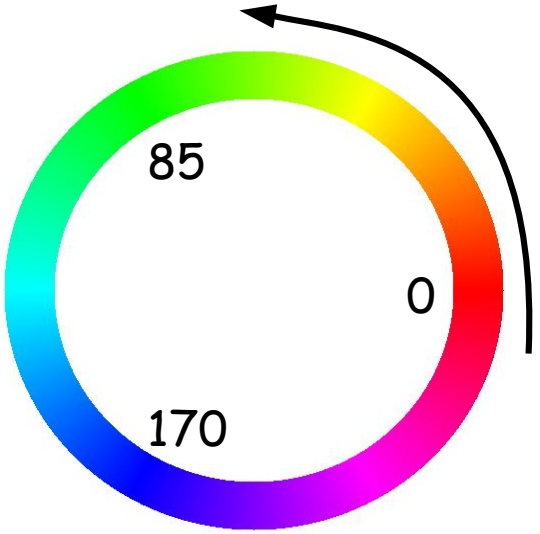
- A **@color.setter** dekoráció segítségével beállító függvényt is készítünk – ettől lesz a **color** tulajdonság írható úgy, hogy egyúttal az RGB LED állapotát is beállítsa

```
    @color.setter
    def color(self, value):
        """Sets the RGB LED to a desired color.
        :param type value: RGB LED desired value - can be RGB tuple or 24-bit integer
        """
        self._current_color = value
        if isinstance(value, tuple):
            rgb = list(value)                       # Convert tuple to list
        elif isinstance(value, int):
            if value >> 24:                         # Range check
                raise ValueError("Only bits 0->23 valid for integer input")
            r = value >> 16                         # Red component
            g = (value >> 8) & 0xFF                 # Green component
            b = value & 0xFF                        # Blue component
            rgb = [r, g, b]
        else:
            raise ValueError("Color must be a tuple or 24-bit integer value.")
RGB     for i in range(0, 3):
LED         rgb[i] = int(rgb[i])*257        # Map from 0..255 to 0..65535
beállí-     if self._invert_pwm:
tása            rgb[i] -= 65535             # Invert for common anode LED
        self._rgb_led_pins[i].duty_cycle = abs(rgb[i])
```

rgbled_class.py 3/3.

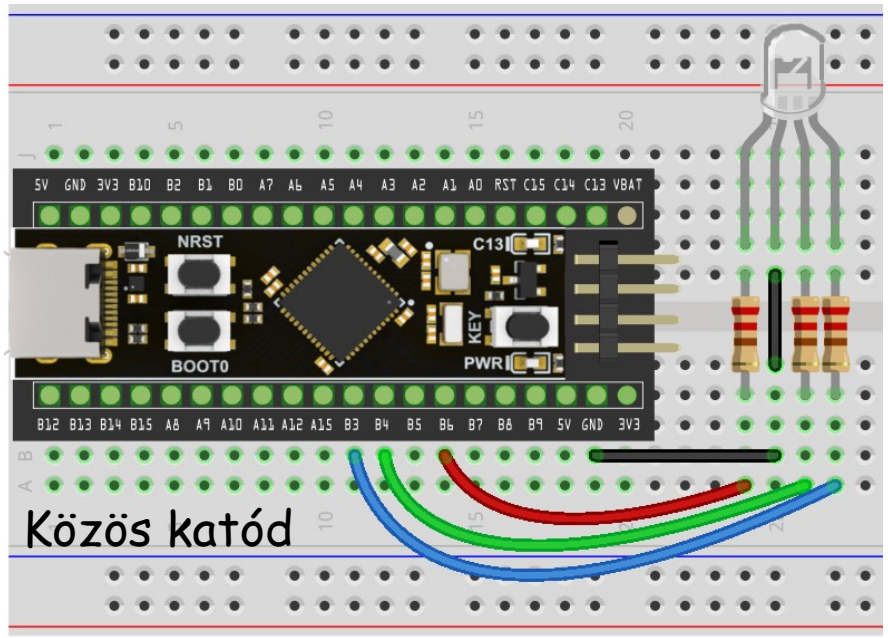
- Az előzőekben definiált **RGBLED** osztályt itt ***led*** néven példányosítjuk – a **B3, B4** és **B6** kivezetéseket rendeljük hozzá, és közös anódú LED esetén invertáljuk a kimeneteket (*invert_pwm = True*)
- Bemutatjuk a színbeállítást RGB számhármassal is és 24 bites egésszel is, majd a Hue értéket végtelen ciklusban folyamatosan változtatjuk

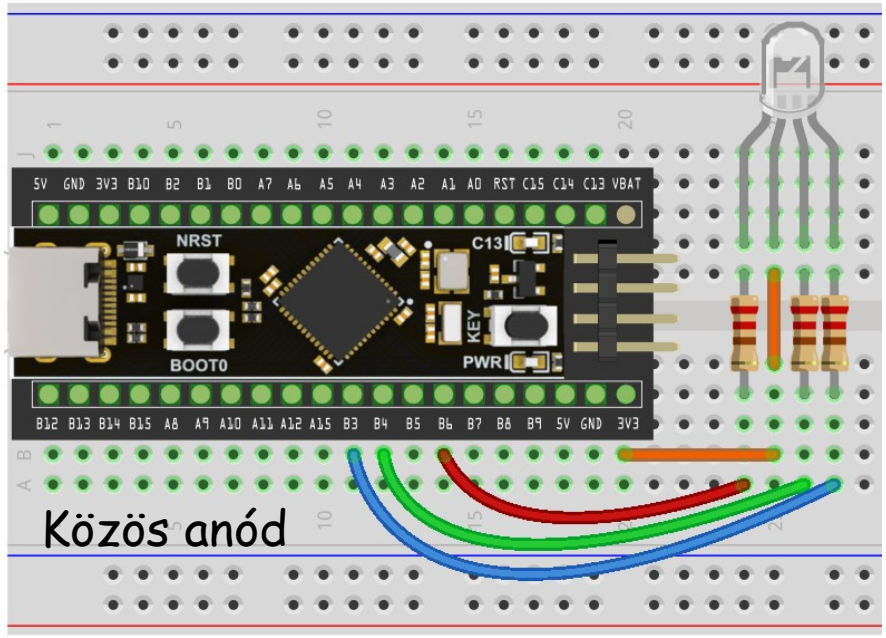
```
# Pin the Red LED is connected to
RED_LED = board.B3            # Timer2 CH2
# Pin the Green LED is connected to
GREEN_LED = board.B4          # Timer3 CH1
# Pin the Blue LED is connected to
BLUE_LED = board.B6           # Timer4 CH1

# Common anode RGB LED definition
led = RGBLED(RED_LED, GREEN_LED, BLUE_LED, invert_pwm=True)
# setting RGB LED color as RGB Tuples (R, G, B)
led.color = (0, 255, 0)       # Green color
time.sleep(1)
# setting RGB LED color as 24-bit integer values
led.color = 0x0000FF          # Blue color
time.sleep(1)

while True:
    for i in range(256):
        led.color = colorwheel(i)  # change Hue value
        time.sleep(0.05)
```



Adafruit_CircuitPython_RGBLED

- Az előzőeket csak a tanulságok miatt csináltuk végig
- A kényelmesebb út, az Adafruit_CircuitPython_RGBLED programkönyvtár használata, ehhez azonban telepíteni kell az előző előadásban már használt Adafruit_CircuitPython_SimpleIO programkönyvtárat is (most csak a **map_range()** függvényt használjuk belőle, ami a 0..255 közötti színkódokat a PWM csatornák 0..65535 közötti tartományára skálázza át)
- A kapcsolásban lehet közös katódú, vagy közös anódú RGB LED





rgbled_demo.py 2/1.

```
import time
import board
from rainbowio import colorwheel
import adafruit_rgbled

# Pin the Red LED is connected to
RED_LED = board.B6

# Pin the Green LED is connected to
GREEN_LED = board.B4

# Pin the Blue LED is connected to
BLUE_LED = board.B3

# Create the RGB LED object
led = adafruit_rgbled.RGBLED(RED_LED, GREEN_LED, BLUE_LED)

# Optionally, you can also create the RGB LED object with inverted PWM
# led = adafruit_rgbled.RGBLED(RED_LED, GREEN_LED, BLUE_LED, invert_pwm=True)


def rainbow_cycle(wait):
    for i in range(255):
        led.color = colorwheel(i)
        time.sleep(wait)
```

Közös katódú RGB LED esetén így írjuk

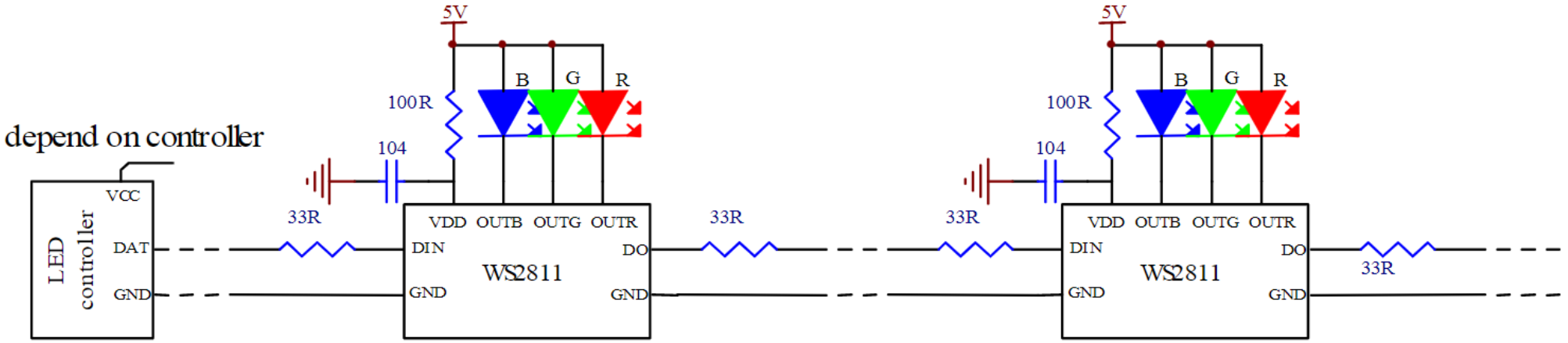
Közös anódú RGB LED esetén így írjuk

rgbled_demo.py 2/2.

```
# setting RGB LED color to RGB Tuples (R, G, B)
led.color = (255, 0, 0)
time.sleep(1)

led.color = (0, 255, 0)
time.sleep(1)

led.color = (0, 0, 255)
time.sleep(1)

# setting RGB LED color to 24-bit integer values
led.color = 0xFF0000
time.sleep(1)

led.color = 0x00FF00
time.sleep(1)

led.color = 0x0000FF
time.sleep(1)

while True:
    # rainbow cycle the RGB LED
    rainbow_cycle(0.05)
#end
```

WS2812 alapok

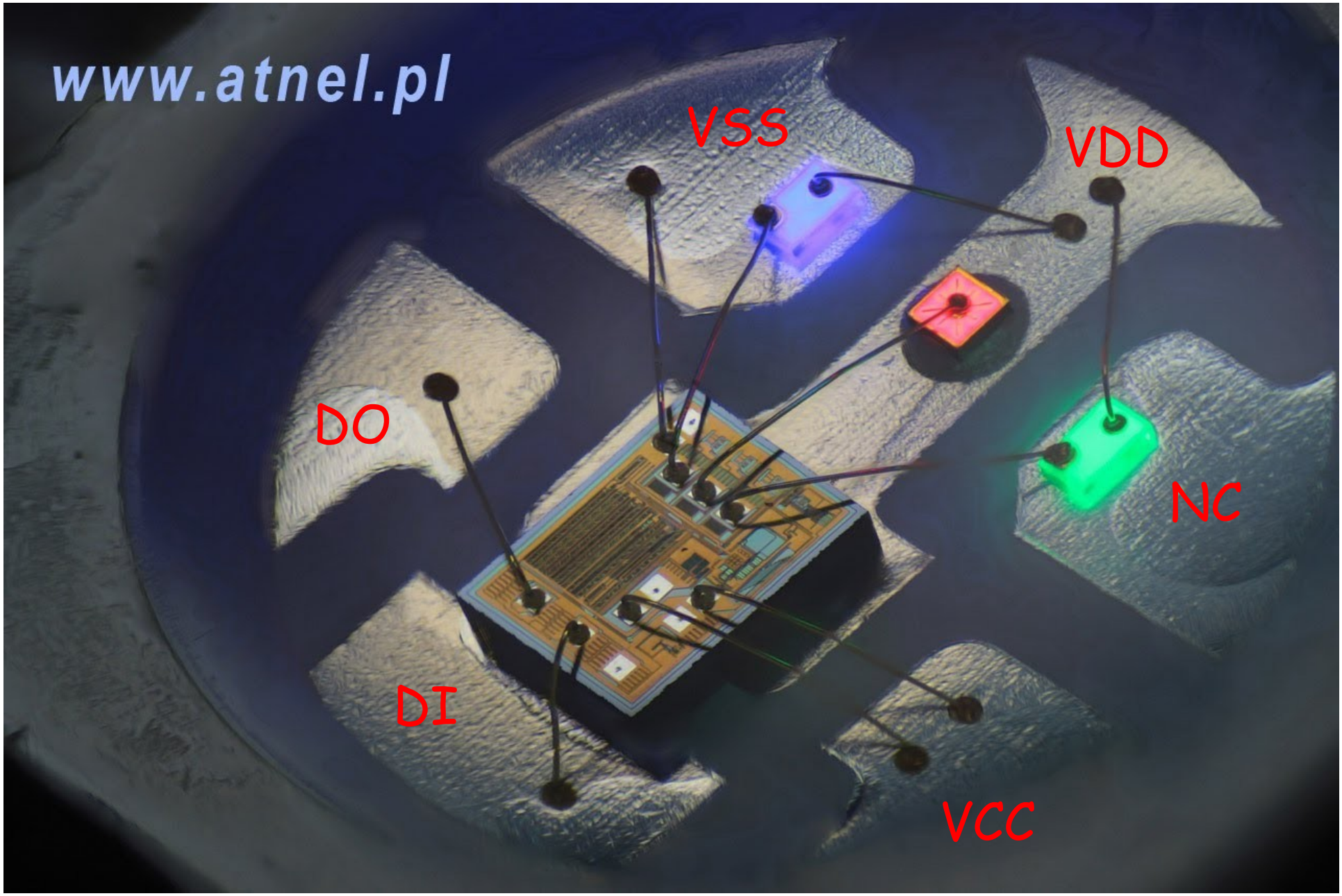
Kezdetben volt a **WS2801,** majd a **WS2811** LED vezérlő IC (WS = World Semi, a gyártó).



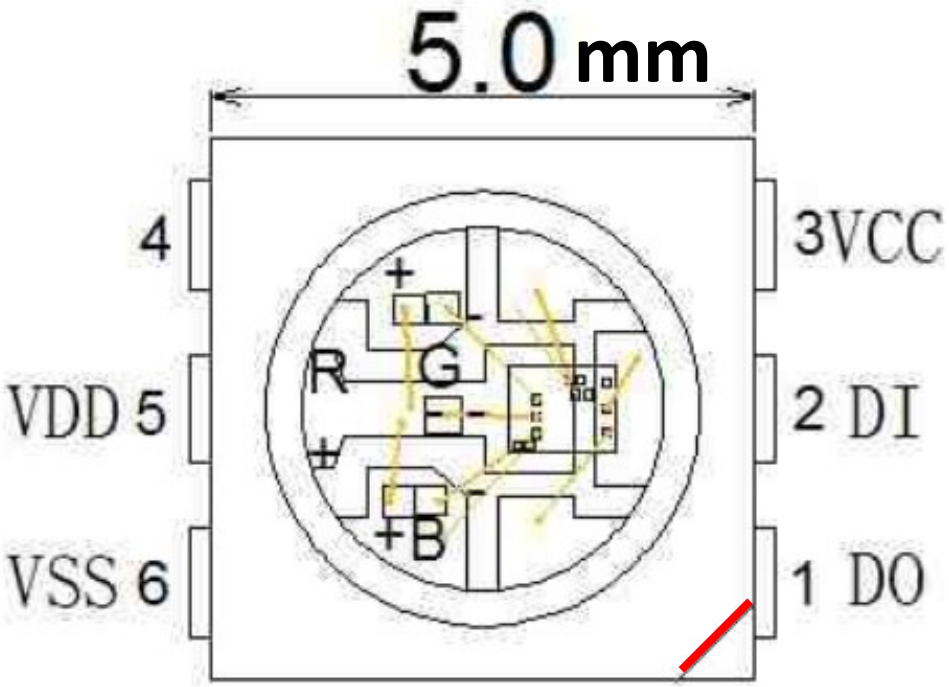
A **WS2812** (NeoPixel) viszont egy olyan 5x5 mm-es RGB LED, amelybe már be van építve egy WS2811-hez hasonló vezérlő!

5050 méretű RGB LED **5050 méretű WS2812**

A WS2812 belső felépítése



WS2812 lábkiosztás, paraméterek



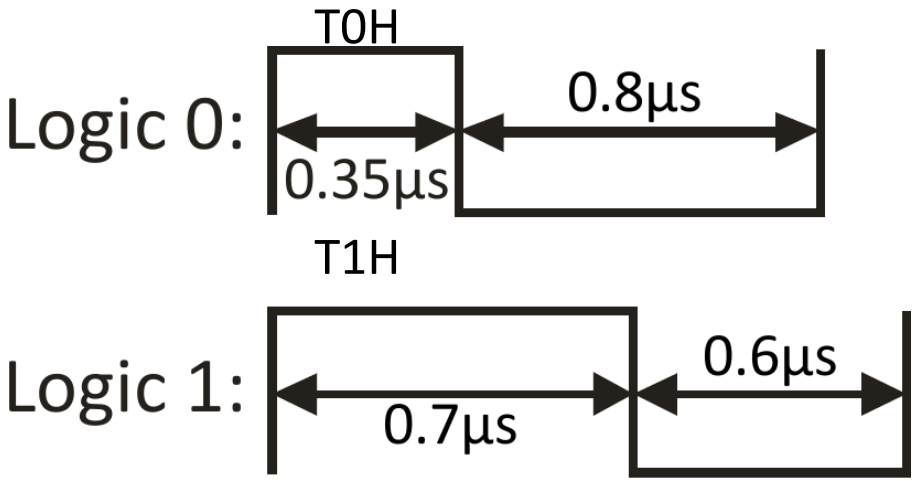
Láb	Név	Funkció
1	DO	Adat kimenet
2	DI	Adat bemenet
3	VCC	Vezérlő tápfeszültsége
4	NC	Nincs bekötve
5	VDD	LED-ek tápfeszültsége
6	VSS	Közös pont (GND)

Paraméter	Jel	Határérték	Egység
Tápfeszültség (vezérlő)	VCC	+6,0 – 7,0	V
Tápfeszültség (LED)	VDD	+6,0 – 7,0	V
Bemeneti jelszint	DI	-0,5 – VDD+0,5	V
Áramfelvétel LED-enként	I	0 – 20	mA

Megjegyzések: Erős a gyanú, hogy az adatlapban van némi keveredés…

1. A tápfeszültség inkább +3,5 – 7,0 V, nem pedig +6,0 – 7,0 V!
2. A bemeneti jelszint inkább a vezérlő, mintsem a LED-ek tápfeszültségéhez igazodik!

WS2812 kommunikáció

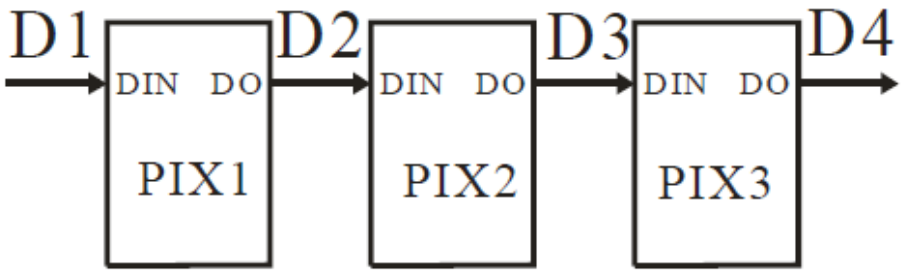


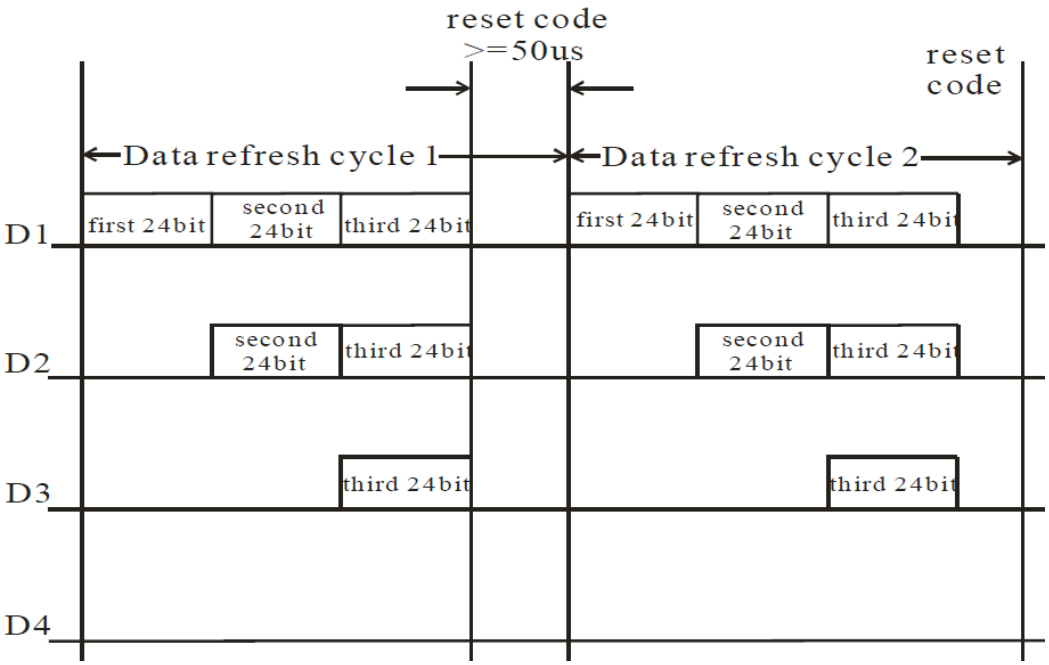
Az egyes bitek értéke a magas szinten eltöltött időtől függ.
T0H: (350 ± 150) ns
T1H: (700 ± 150) ns

RESET: alacsony szint T ≥ 50 000 ns ideig.

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

Adatformátum: 3 x 8 bit, GRB sor-rendben, magas helyiértékű bit elöl.

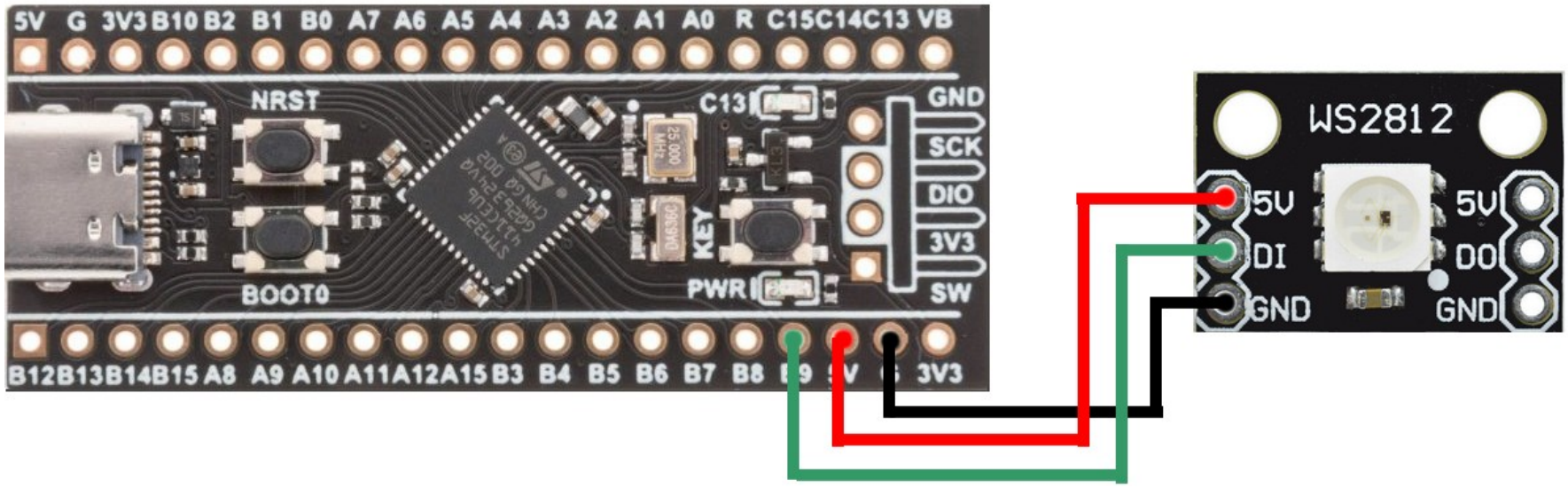




neopixel_write.py

- A **CircuitPython** beépített könyvtári moduljai között már megtalálható a **WS2812** kompatibilis RGB LED-ek alacsonyszintű kezelését végző **neopixel_write** könyvtár, amely a hasonló nevű függvényt tartalmazza
- A **B9** kimenetre kötött neopixel LED zöldre állítása pl. így végezhető:

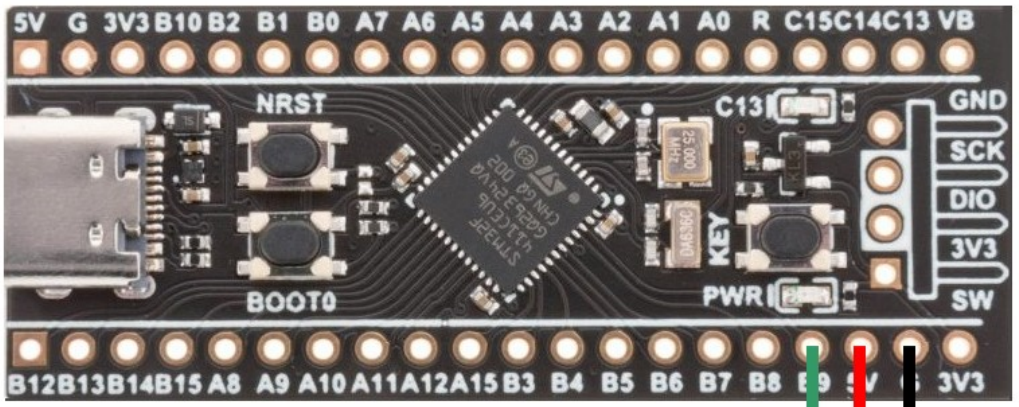
```
import board
import neopixel_write
import digitalio
pin = digitalio.DigitalInOut(board.B9)
pin.direction = digitalio.Direction.OUTPUT
pixel_color = bytearray([255, 0, 0])
neopixel_write.neopixel_write(pin, pixel_color)
```



neopixel_write_8.py

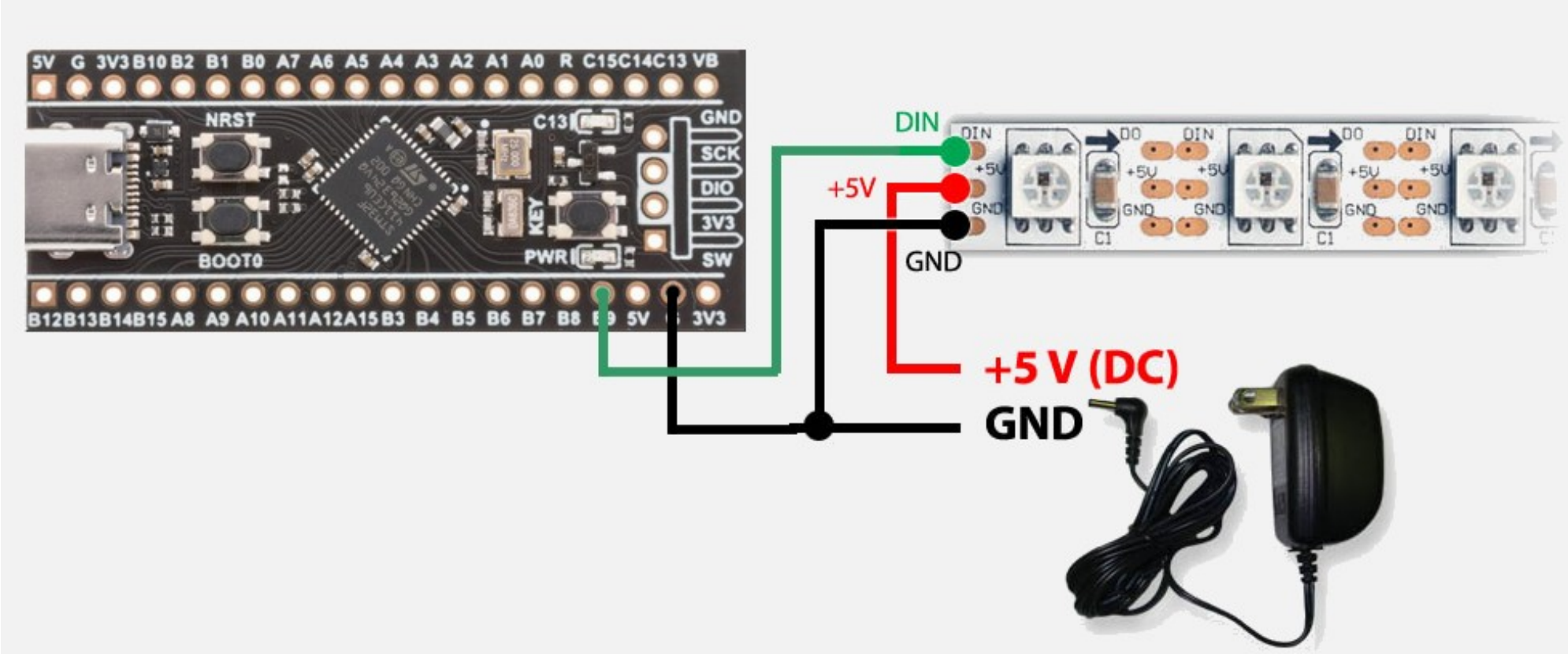
- A **neopixel_write** függvény segítségével több LED-et is írhatunk, ha a pixel_color bufferbe megfelelő számú számhármast írunk
- A legelső számhármas a legközelebbi LED színét adja meg (az alábbi példában ez a zöld)

```
import board
from neopixel_write import *
from digitalio import *
pin = DigitalInOut(board.B9)
pin.direction = Direction.OUTPUT
pixel_color = bytearray([0,200,0, 200,0,0, 0,0,200, 200,100,0,
100,0,100, 0,200,200, 200,200,200, 100,200,0])
neopixel_write(pin, pixel_color)
```



WS2812 szalag bekötése

- Az áramfelvétel számításához LED-enként 3 x 20 mA maximális áramot vegyünk! Méterenként 30 db LED-del számolva ez 1.8A/m áramfelvételt jelent.



Adafruit Neopixel programkönyvtár

- A **WS2812** LED szalagok vezérlésére szolgál az Adafruit CircuitPython Bundle csomagban is megtalálható **Adafruit_CircuitPython_NeoPixel** programkönyvtár
- A használatához a ***neopixel.py***, vagy a ***neopixel.mpy*** állományt be kell másolni a CircuitPythont futtató kártyánk ***lib*** mappájába
- A **neopixel.NeoPixel** objektumosztály ***n*** darab felfűzött LED vezérlését végzi, a darabszámot a példányosításkor kell megadni
- Példányosításkor azt is megválaszthatjuk, hogy az adatbeírás azonnal módosítsa az érintett LED állapotát, vagy pedig majd a ***show()*** metódussal egyszerre frissítjük minden LED állapotát
- Példa: egy 8-ledes Neopixel stick, a **B9** kimenetre kötve

```
import board
import neopixel
pixels = neopixel.NeoPixel(board.B9, 8) # Az adatbeírás azonnal érvényesül
pixels.fill(0xff0000)                   # Legyen minden LED piros!
```

Adafruit Neopixel programkönyvtár

- A konstruktor paraméterei:
  NeoPixel(*pin,n,bpp=3,brightness=1.0,auto_write=True,pixel_order=None*)
  ahol:
  pin – a LED vezérlő digitális kimenet
  n – a LED-ek száma
  bpp – az egy LED-ez tartozó bájtok száma (3, vagy 4)
  brightness – a fényerőt szabályozó paraméter (0 – 1.0 közötti float)
  auto_write – *True:* a beírás azonnal érvényesül, *False*: *show()* érvényesít
  pixel_order – az adatkiküldés előírt sorrendje (alapértelmezetten GRB)
- **Megjegyzések:**
  - A színmegadás sorrendje mindig RGB(W), függetlenül a kiküldés sorrendjétől
  - Az adatbuffer kezelése az **adafruit_pixelbuf** modulban van megírva, a **NeoPixel** osztály ugyanis a **PixelBuf** osztály leszármazottja

Adafruit Neopixel programkönyvtár

- **Tulajdonságok:**
  - **bpp**
  - **brightness**
  - **auto_write**
  - **byteorder**
- **Metódusok:**
  - **fill**(color) – minden LED-et beállít a megadott színre
  - **show**() – a beírt színkódokat kiírja a LED-ekre
- **Értékadás (színbeállítás):**
  - pixels[i] = (r, g, b)
  - pixels[i] = 0xrrggbb
  - pixels[::2] = [(r,g,b)] * len(pixels) // 2 # Szeletelés

adafruit_neopixel_demo.py 2/1.

- A CircuitPython Essentials-ból átvett mintapélda bemutatja a **neopixel** programkönyvtár használatát és lehetőségeit

```
import time
import board
from rainbowio import colorwheel
import neopixel

pixel_pin = board.B9                    # A LED szalagot vezérlő kimenet
num_pixels = 8                          # A LED-ek száma
pixels = neopixel.NeoPixel(pixel_pin, num_pixels, brightness=0.3, auto_write=False)

def color_chase(color, wait):           # Futófény effektus
    for i in range(num_pixels):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
    time.sleep(0.5)

def rainbow_cycle(wait):                # Szivárvány futófény effektus
    for j in range(255):
        for i in range(num_pixels):
            rc_index = (i * 256 // num_pixels) + j
            pixels[i] = colorwheel(rc_index & 255)
        pixels.show()
        time.sleep(wait)
```

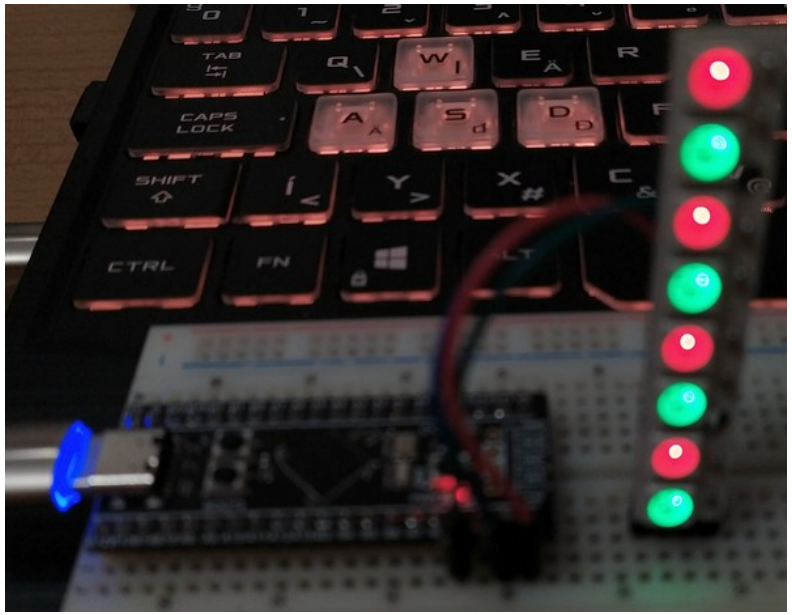
adafruit_neopixel_demo.py 2/2.

```
RED = (255, 0, 0)
YELLOW = (255, 150, 0)
GREEN = (0, 255, 0)
CYAN = (0, 255, 255)
BLUE = (0, 0, 255)
PURPLE = (180, 0, 255)

while True:
    pixels.fill(RED)
    pixels.show()
    # Increase or decrease to change the speed of the solid color change.
    time.sleep(1)
    pixels.fill(GREEN)
    pixels.show()
    time.sleep(1)
    pixels.fill(BLUE)
    pixels.show()
    time.sleep(1)
    color_chase(RED, 0.1)  # Increase the number to slow down the color chase
    color_chase(YELLOW, 0.1)
    color_chase(GREEN, 0.1)
    color_chase(CYAN, 0.1)
    color_chase(BLUE, 0.1)
    color_chase(PURPLE, 0.1)

    rainbow_cycle(0)       # Increase the number to slow down the rainbow
```

neopixel_with.py

- Ez a program két dolgot mutat be:
  - **NeoPixel** objektum használatát a ***with*** paranccsal
  - Pixel buffer értékadást, szeleteléssel
- A programban egy 8 db LED-ből álló füzért vezérlünk: először mindegyiket piros színre állítjuk, majd a szeletelő opcióval minden páros sorszámú LED-et zöld színre állítunk
- A ***with*** parancs miatt a késleltetés leteltekor a ***pixels*** objektum "megsemmisül", a LED-ek kialszanak

```
import neopixel
import board
import time

RED = 0x200000 # (0x20, 0, 0) also works
GREEN = 0x002000

with neopixel.NeoPixel(board.B9, 8) as pixels:
    pixels.fill(RED)
    pixels[::2] = [GREEN] * (len(pixels) // 2)
    time.sleep(5)
```



A with parancs a Python programokban

- A **with** parancs segítségével a Python programokban egyszerűbben és áttekinthetőbben írhatjuk meg a kivételkezeléssel és erőforrás felszabadítással kapcsolatos kódot
- Az első példában ha íráskor hiba keletkezik, akkor elmarad a fájl lezárása
- A második példa kezeli ezt az esetet, de a **with** segítségével kompaktabb módon írhatjuk a kódot, ahogy a harmadik eset mutatja
- Saját fejlesztésű objektumok esetén egy **__enter__**() és egy **__exit__**() metódus definiálásával támogathatjuk a **with** használatát (az erőforrás lefoglalására és inicializálására, illetve az erőforrás korrekt felszabadítására kellenek)

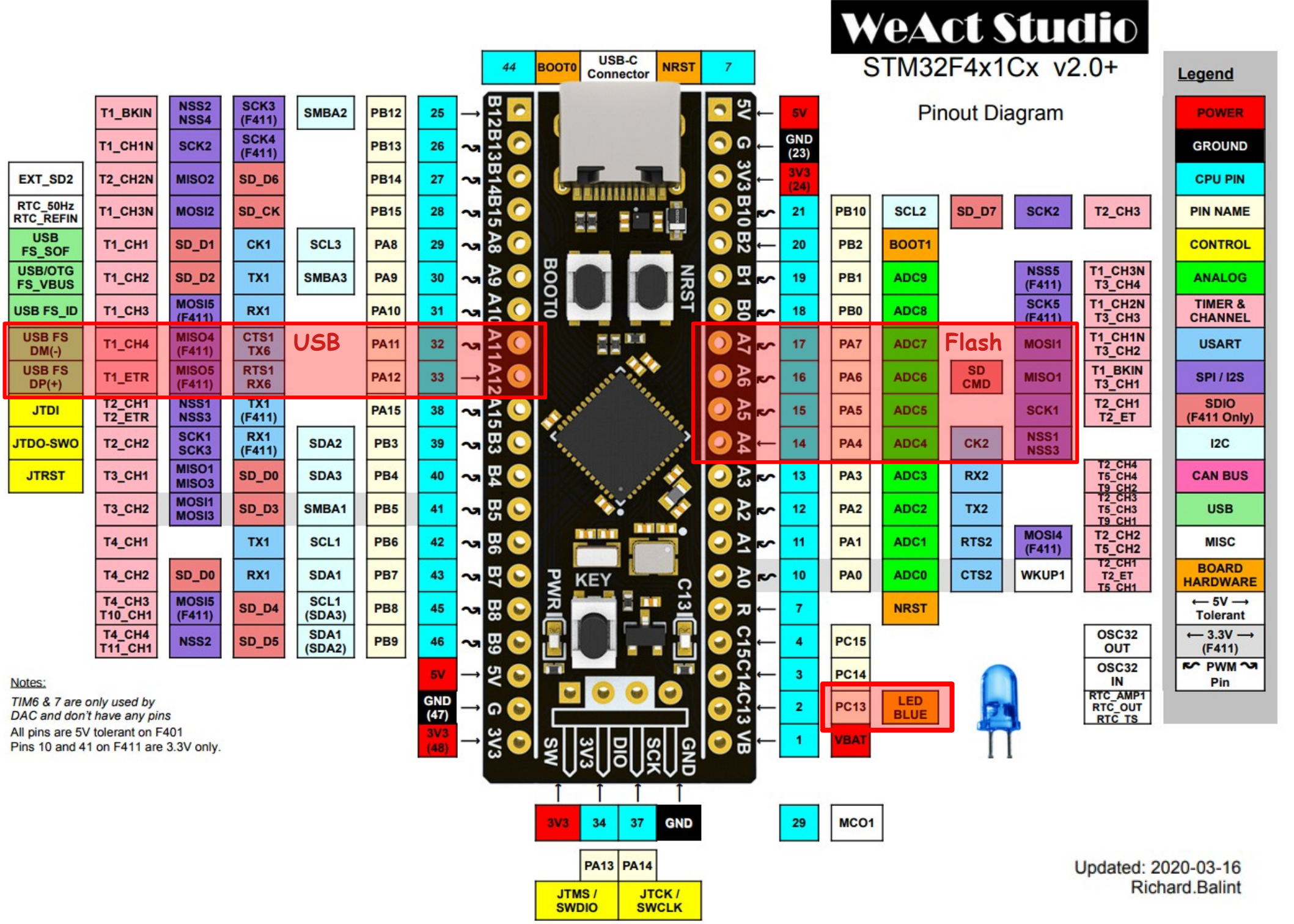
Fájlkezelés with nélkül:

```
# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()
```

Fájlkezelés with használatával:

```
# 3) using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Felhasznált forrás: Geeks for Geeks - with statement in Python


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
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