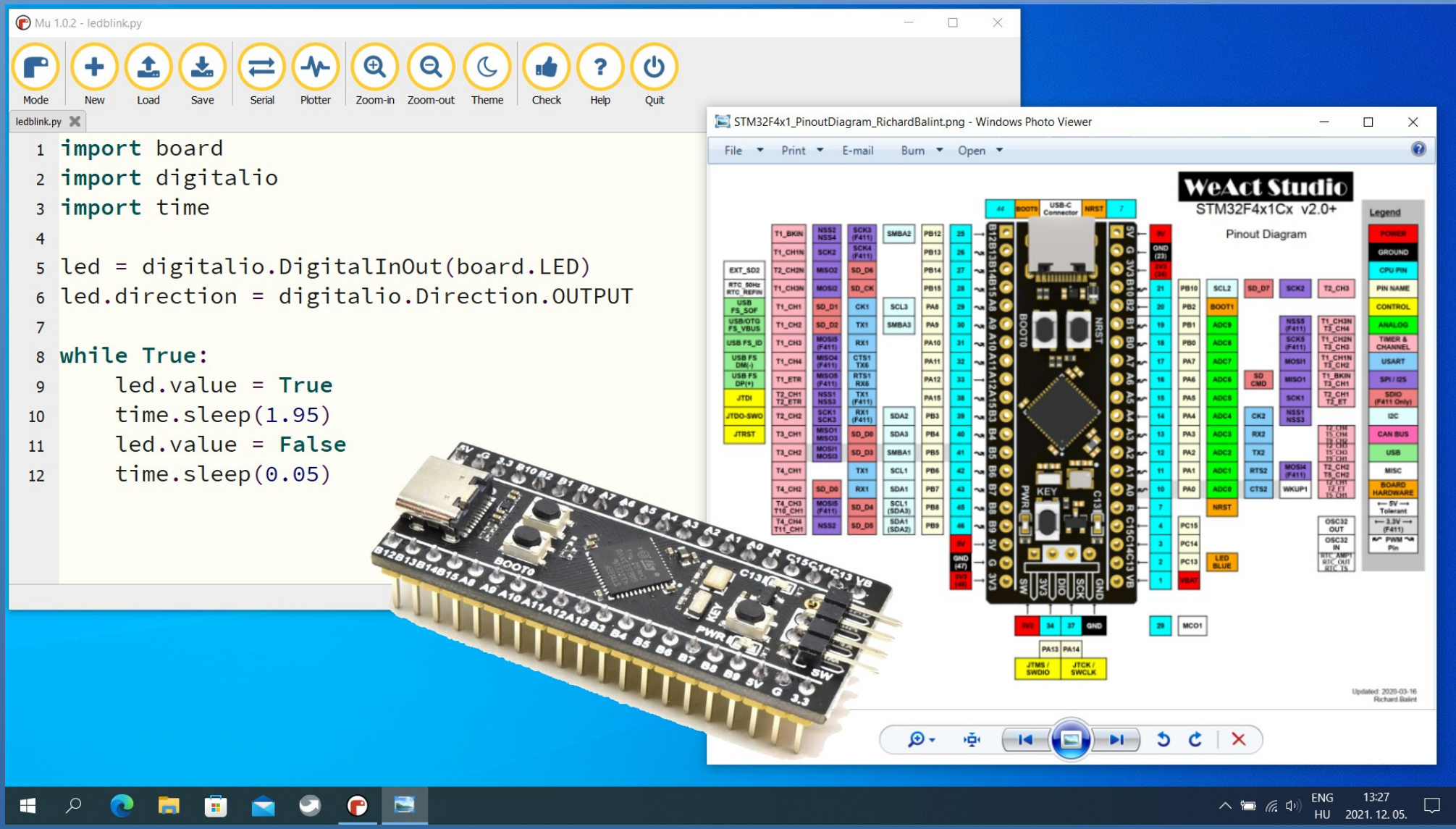
CircuitPython tanfolyam



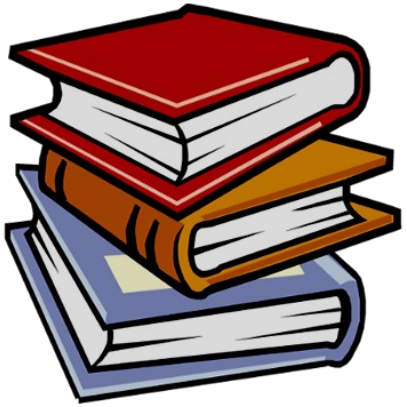
3. E-papír és OLED kijelzők használata

Felhasznált és ajánlott irodalom

Python:

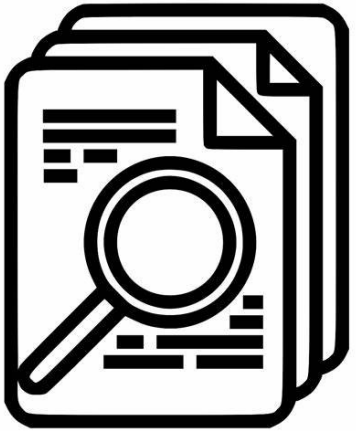
- Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
- P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével

CircuitPython:



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

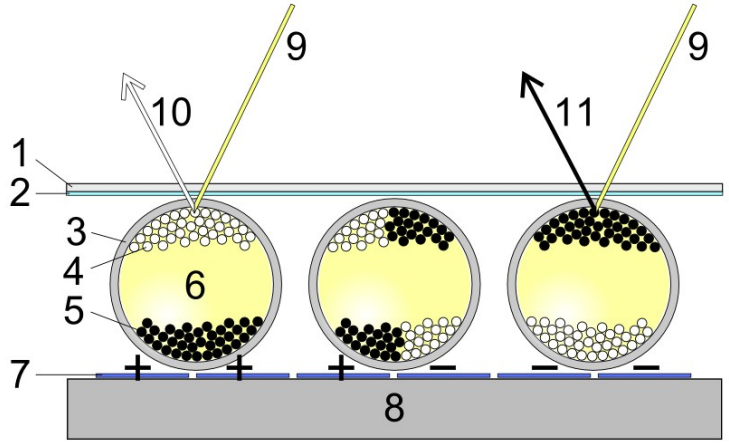
Adatlapok és dokumentáció:

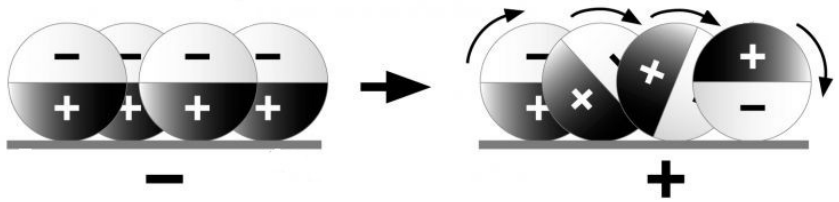


- **STM32F411CE** adatlap és termékinfo
- **STM32F411xC/E** Family Reference Manual
- **WeAct Studio:** STM32F4x1 MiniF4

E-papír kijelzők

- Az **E-papírt** vagy elektronikus papírt az 1970-es években kezdték fejleszteni (Xerox, Gyricon technológia)
- 2004-ben jelent meg az első E-papír könyvolvasó (Sony LIBRIé)
- 2007 Amazon Kindle könyvolvasó; Fujitsu színes **E-papír**
- 2012 Fuji Xerox: ***elektroforézis*** elvén működő színes **E-papír** kijelző (***elektroforézis:*** töltött részecskék vándorlása elektromos tér hatására)
- Az **E-papír** fő részei: a tartalom megjelenítő (elektronikus tinta, E-ink) és a hátlapi elektronika, ami a tartalom létrehozást generálja





Gyricon e-ink működési elve

Az **E-Ink** elektroforézis technológia vázlata: 1 fedő réteg. 2 átlátszó elektródaréteg. 3 átlátszó mikrokapszulák. 4 pozitív töltésű fehér pigmentek. 5 negatív töltésű fekete pigmentek. 6 átlátszó olaj. 7 elektródapixel réteg. 8 alsó támaszréteg. 9 beeső fény 10 fehér képpont. 11 fekete képpont.

E-papír előnyei és hátrányai

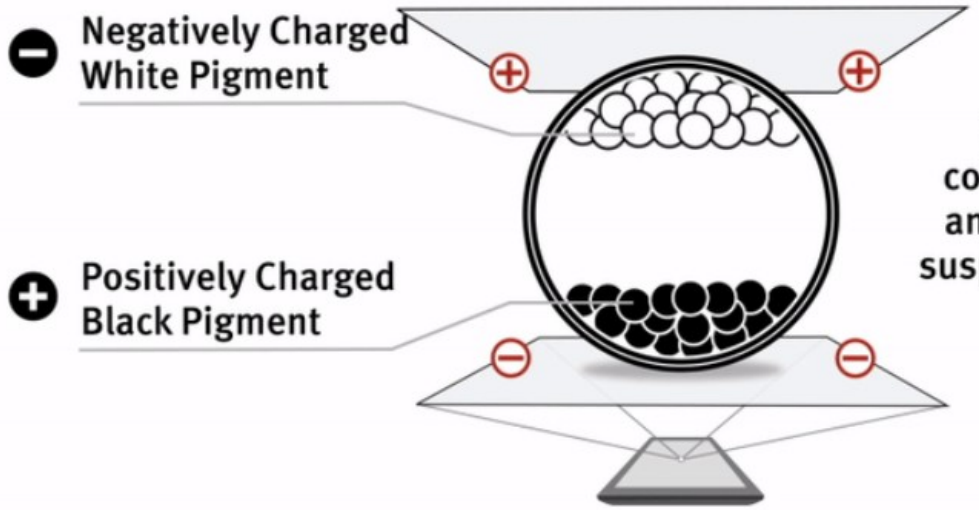
- **Előnyök:**
  - Napfényben is könnyen olvasható.
  - A szöveg megtartása nem igényel befektetett energiát
  - Kontrasztosabb képet ad, mint a háttérvilágítással rendelkező kijelzők
  - Nagy a betekintési szöge, ezért oldalról is jobban olvasható, mint a háttérvilágítással rendelkező kijelzők
- **Hátrányok:**
  - Sötétben kiegészítő (külső vagy háttér) világításra van szükség
  - Drága a színes kijelző
  - A színes kijelzők színválasztéka igen szerény
  - Lassú a képfrissítés ideje, pl. videó lejátszásra alkalmatlan

E-papír kijelzők alkalmazásai

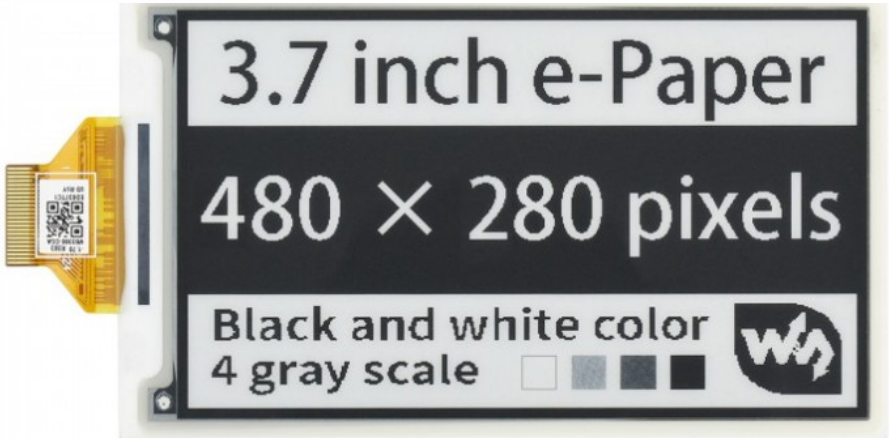
Yotaphone 2

Modern E-ink technológiák

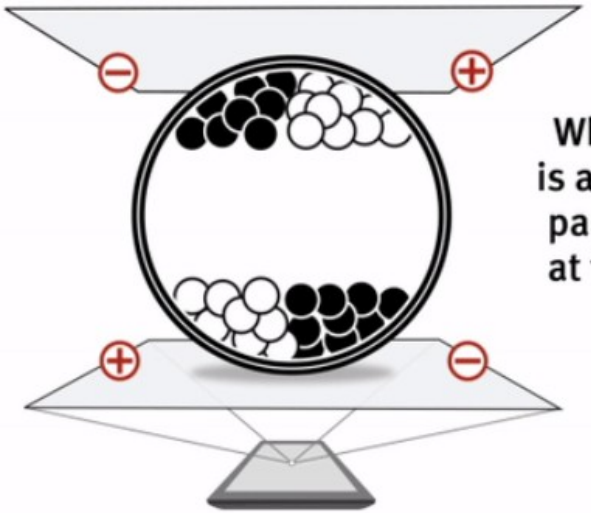
- Forrás: https://www.eink.com/electronic-ink.html



Each microcapsule contains charged black and white ink particles suspended in a clear fluid



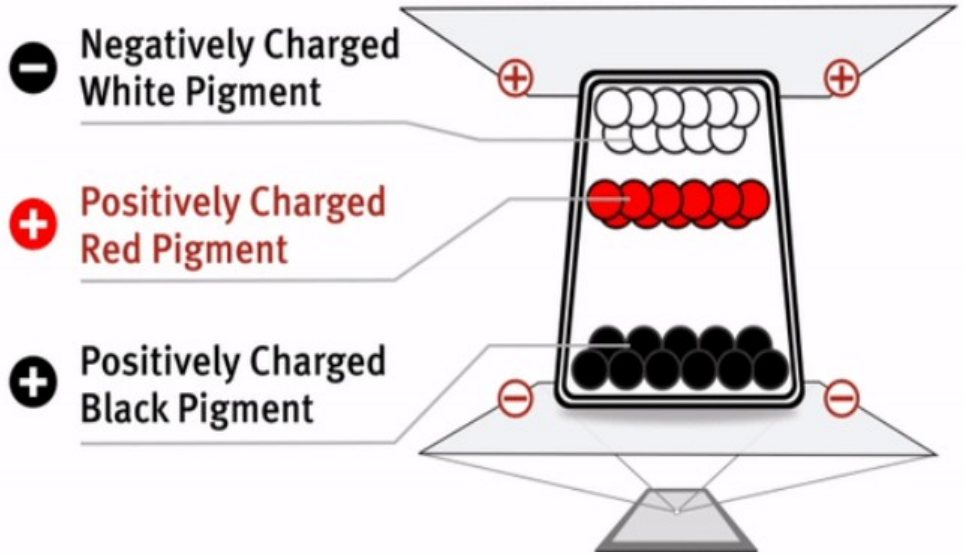
Egy konkrét megvalósítás



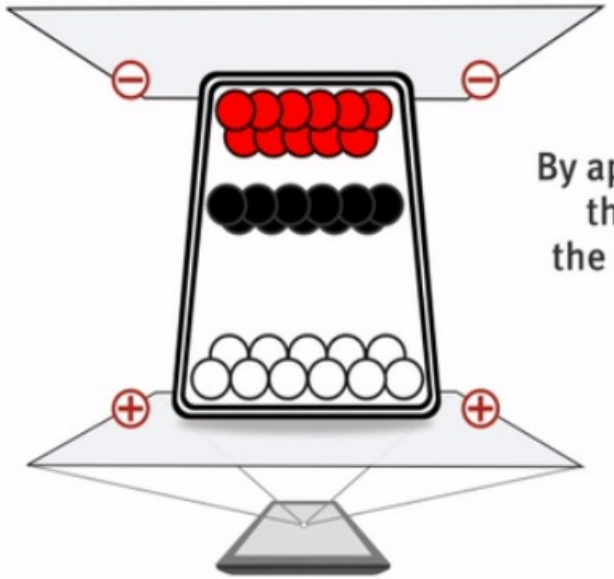
When an electrical field is applied, corresponding particles become visible at the top of the capsule

Háromszínű kijelző, három pigmenttel

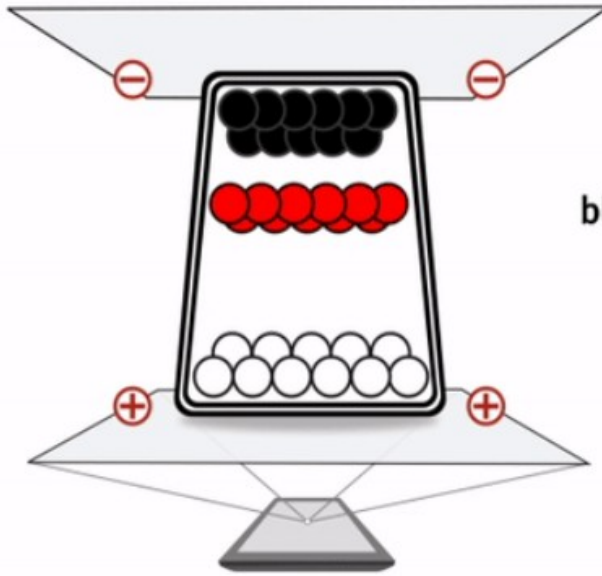
- Forrás: https://www.eink.com/electronic-ink.html



Egy konkrét megvalósítás

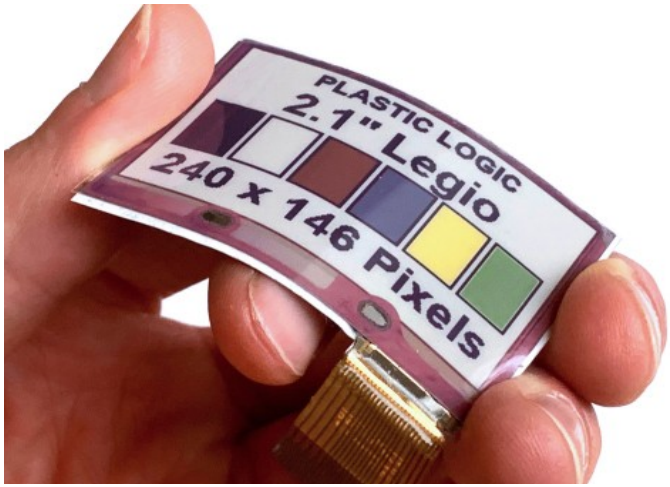


By applying a negative charge
the red pigment rises to
the top and becomes visible



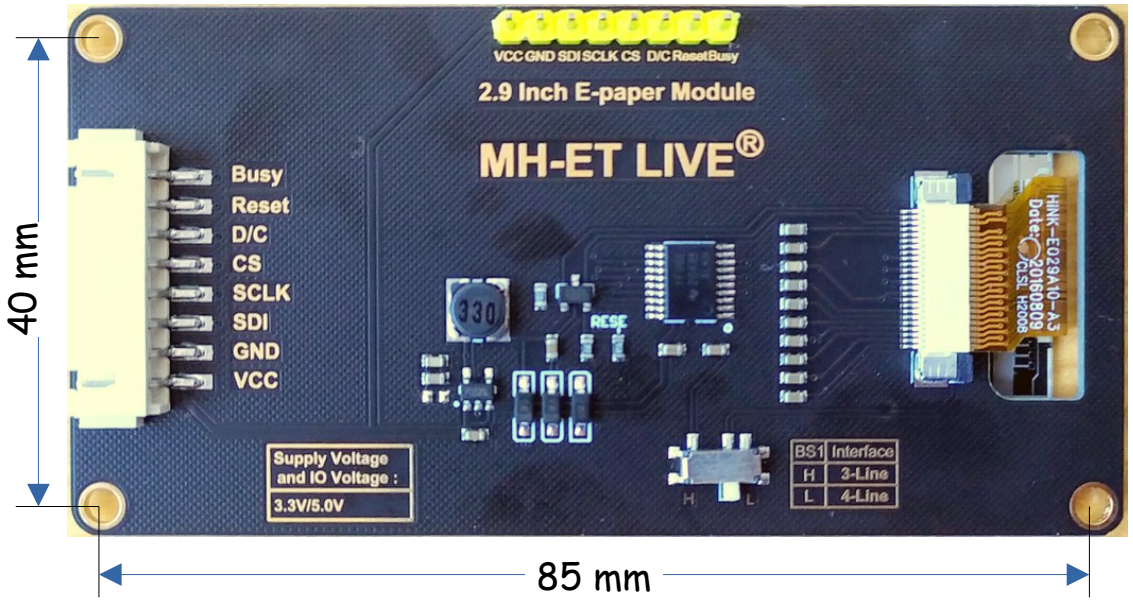
A split charge raises
black pigment to the top
and becomes visible

Hobbi célú E-papír kijelzők



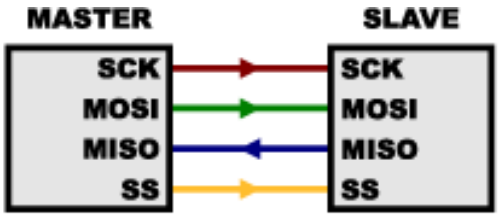
Az MH-ET LIVE kijelző kártya

- Kijelző méret: **2,9”** (66,89x29,05 mm)
- Felbontás: **296x128**
- Betekintési szög: 170°
- Szín: fekete, fehér, piros v. sárga
- A kártyán szintátalakító is van
- Interfész: **SPI** 4 vagy 3 vezetékes (utóbbi esetben 9 bitet kell küldeni a **D/C** vonal vezérlése helyett)
- **Reset** és **Busy** vonal (opcionális)
- Kijelző: **HINK-E029A10-A3** (legalábbis nálam ez van)
- Frissítési idő: 15 s
- Tápfeszültség: 2,8 – 5,3 V
- Felvett teljesítmény: 26.4 mW (frissítéskor)
- Pixelméret: 0,226x0,227 mm
- Kártya méret: 90x45 mm

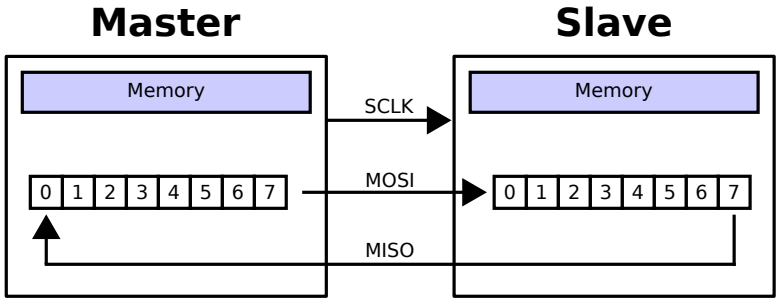


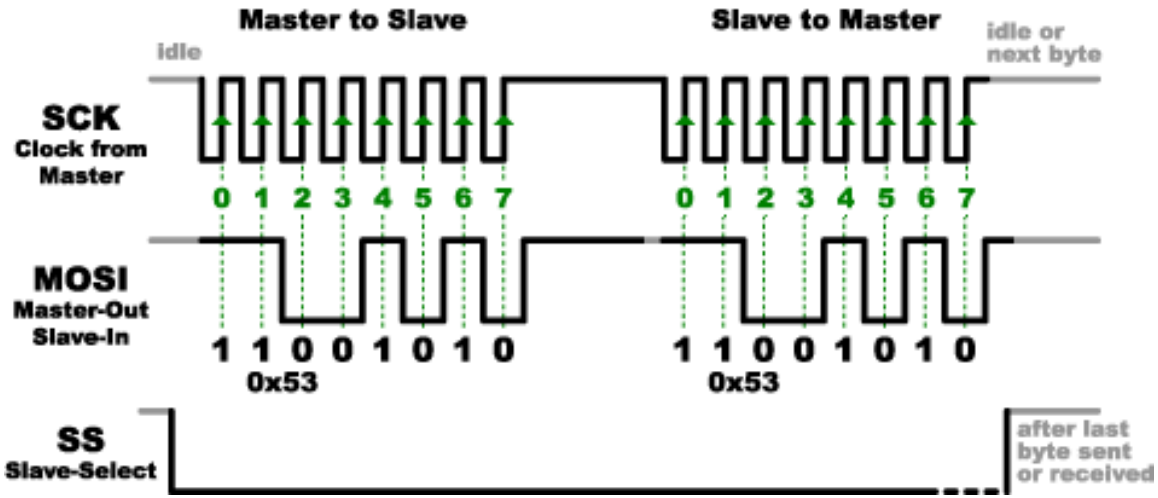
Az SPI kommunikációs csatorna

- A Soros Periféria Illesztő (**SPI**) bitsoros, kétirányú kommunikáció (mi most csak az egyik irányt használjuk)
- A Master (esetünkben a mikrovezérlő) állítja elő az órajelet, ez szinkronizál és ütemezi az adatküldés sebességét
- Az **SPI** csatorna jelei:
  SCK – az órajel
  MOSI – a master kimenő adatvonala
  MISO – a master adatbemenete
  CS, vagy **SS** – a slave kiválasztó jele



- A kommunikáció során két léptetőregiszter adatot cserél





Az SPI osztály tagfüggvényei

- A **busio** modul tartalmazza a különféle soros perifériákat kezelő osztályokat (I2C, SPI, OneWire, UART)
- Leírásuk az Adafruit CircuitPython dokumentációban található

```
>>> import busio
>>> dir(busio)
['__class__', '__name__', 'I2C', 'OneWire', 'SPI', 'UART']

>>> help(busio.SPI)
object <class 'SPI'> is of type type
  deinit -- <function>              # Periféria elengedése
  __enter__ -- <function>           # Context Manager (pl. with) segítő
  __exit__ -- <function>            # Context Manager (pl. with) segítő
  configure -- <function>           # Csatorna konfigurálása
  try_lock -- <function>            # Csatorna lefoglalása
  unlock -- <function>              # Csatorna felszabadítása
  readinto -- <function>            # Adatbeolvasás
  write -- <function>               # Adatkiírás
  write_readinto -- <function>      # Írás/olvasás (duplex mód)
  frequency -- <property>           # Adatsebesség beállítása
>>>
```

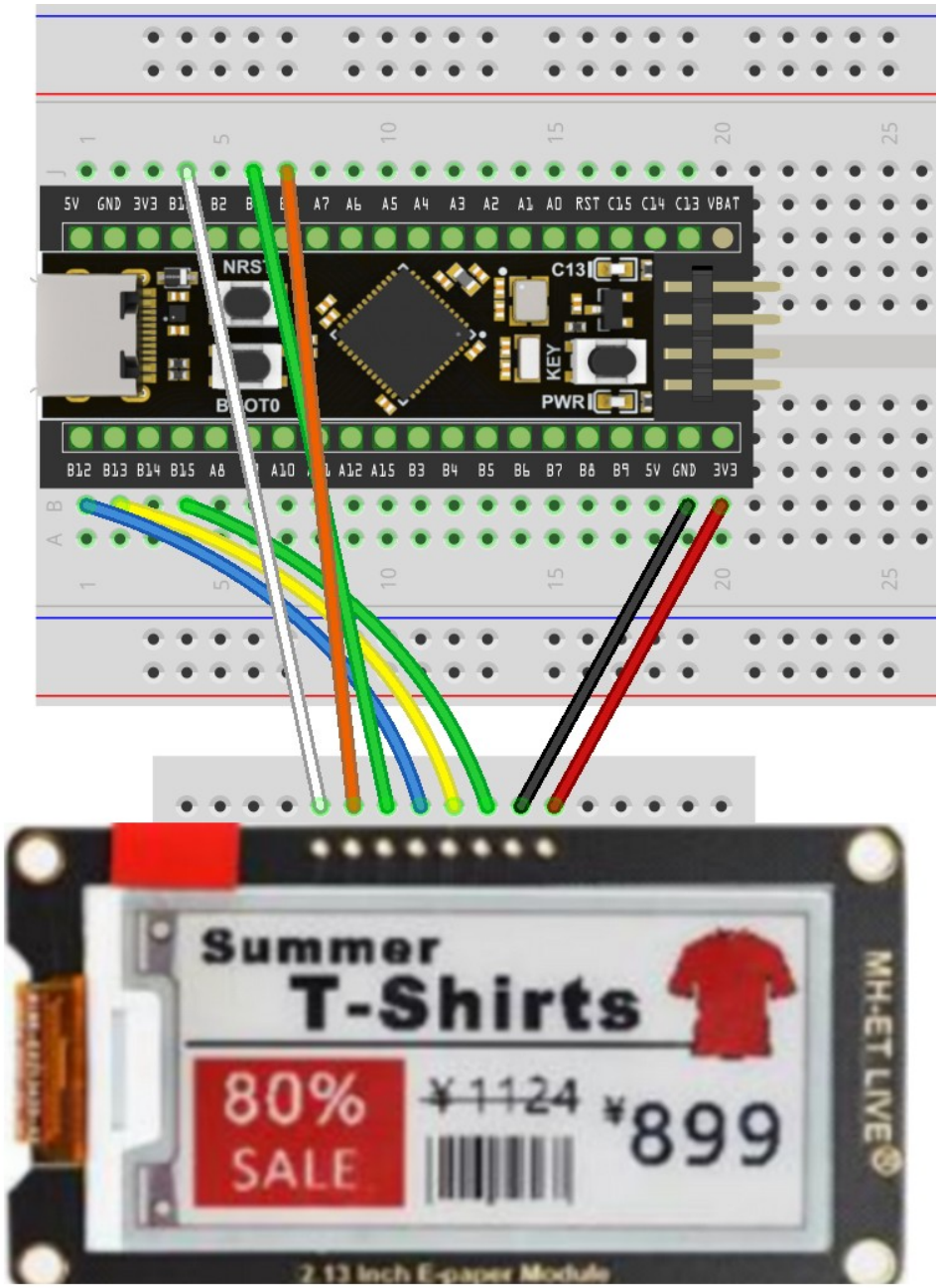
Adafruit_CircuitPython_EPD library

- Az **Adafruit_CircuitPython_EPD** programkönyvtár többféle E-papír kijelzőt támogat, a **HINK-E029A10-A3** kijelzőt az **Adafruit_IL0373** (al)osztály segítségével sikerült működésbe hozni [kell hozzá az **adafruit_framebuf.mpy** és a **font5x8.bin** is!]
- **Adafruit_IL0373**(*width, height, spi, ecs, dc, srcs, rst, busy*) – a konstruktor
- **display**() – a display buffer megjelenítése a képernyőn
- **fill**(*color*) – a display buffer feltöltése a megadott színnel
- **fill_rect**(*x, y, width, height, color*) – kitöltött téglalap rajzolása
- **hline**(x, y, width, color) – vízszintes vonal rajzolása
- **line**(x_0, y_0, x_1, y_1, color) – szakasz rajzolása
- **pixel**(x, y, color) – egy képpont kirajzolása a display bufferbe
- **rect**(*x, y, width, height, color*) – téglalap rajzolása
- **rotation**(*n*) – a kép elforgatása 90 fokkal (*0, 1, 2, 3*)
- **text**(*string, x, y, color, *, font_name='font5x8.bin', size=1*) – adott szöveg kiírása az (x, y) pozíciótól kezdve, megadott színnel a fontfájl neve és a fontméret megadása opcionális

A javasolt bekötési vázlat

- A kijelző kivezetéseit így kötöttük be (jobbról balra haladva):

VCC	3V3	Tápfeszültség
GND	GND	Közös pont
SDI	B15	SPI MOSI
SCK	B13	SPI órajel
CS	B12	SPI kiválasztó jel
D/C	B1	Adat/Parancs választó
RST	B0	Hardver reset
Busy	B10	Foglaltság jelzése



- **B12, B13, B15:** az **SPI2** kivezetései (**B14** a MISO2, de nem használjuk)

epd_simpletest.py 2/1.

- Az Adafruit_Circuit_EPD programkönyvtár mintapéldája
  Link: https://github.com/adafruit/Adafruit_CircuitPython_EPD

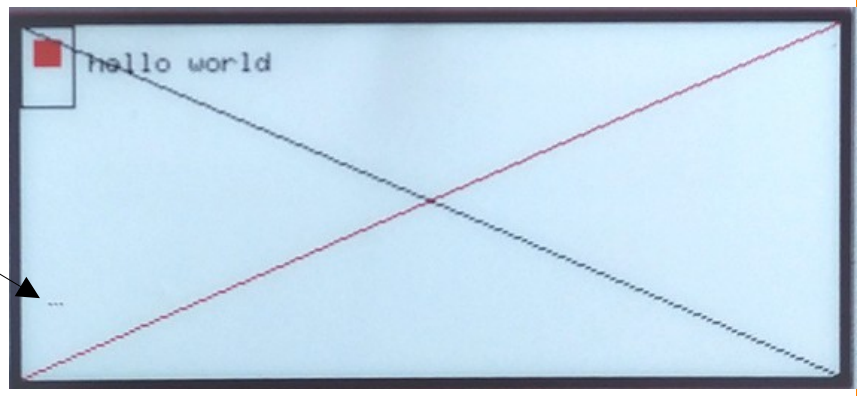
```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373

# create the spi device and pins we will need
spi = busio.SPI(board.B13, MOSI=board.B15, MISO=board.B14)
ecs = digitalio.DigitalInOut(board.B12)    # Chip Select pin
dc = digitalio.DigitalInOut(board.B1)      # Data/Command selector
srcs = None                                # can be None to use internal memory
rst = digitalio.DigitalInOut(board.B0)     # can be None to not use this pin
busy = digitalio.DigitalInOut(board.B10)   # can be None to not use this pin

# give them all to our drivers
print("Creating display")
display = Adafruit_IL0373(128, 296,          # 2.9" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)
```

Az SPI kommunikációs csatorna megnyitása

epd_simpletest.py 2/2.

```
# IF YOU HAVE A 2.9" FLEXIBLE DISPLAY uncomment these lines!
display.set_black_buffer(0, True) # changed from 1 to zero
display.set_color_buffer(1, True)

# Set rotation
display.rotation = 1    # 0 narrow side (MHT-ET Live sign) at top
                        # 1 pins at 6 o'clock, 3 pins at 12 o'clock

# clear the buffer
print("Clear buffer")
display.fill(Adafruit_EPD.WHITE)
display.pixel(10, 100, Adafruit_EPD.BLACK)
display.pixel(12, 100, Adafruit_EPD.RED)
display.pixel(14, 100, Adafruit_EPD.BLACK)

print("Draw Rectangles")
display.fill_rect(5, 5, 10, 10, Adafruit_EPD.RED)
display.rect(0, 0, 20, 30, Adafruit_EPD.BLACK)

print("Draw lines")
display.line(0, 0, display.width - 1, display.height - 1, Adafruit_EPD.BLACK)
display.line(0, display.height - 1, display.width - 1, 0, Adafruit_EPD.RED)

print("Draw text")
display.text("hello world", 25, 10, Adafruit_EPD.BLACK)
display.display()
# end
```

Ez a rész fölösleges, de ha mégis itt van, ne legyen hibás!



Bitkép kirajzolása

- Sajnos, a kép kirajzolására szolgáló **image**(*image*) metódus egy **PIL** (Python Image Library) **Image** objektumpéldányt vár paraméterként, de ez a programkönyvtár mikrovezérlőkön nem áll rendelkezésre, ezért a bitmap ábrákat pontonként kell kirajzolni
- Az Adafruit_EPD programkönyvtár **epd_bitmap.py** mintapéldája a megadott fáljból egy BMP képet (24 bites színmélységű legyen!) olvas be és rajzol ki. A kép mérete egyezzen meg a kijelzővel
- Első lépésként készítsünk tehát egy 296x128 méretű képet, és másoljuk fel a mikrovezérlő kártya fájlrendszerébe **logo.bmp** néven!

epd_bitmap.py 4/1.

```
import digitalio
import busio
import board
from adafruit_epd.epd import Adafruit_EPD
from adafruit_epd.il0373 import Adafruit_IL0373

spi = busio.SPI(board.B13, MOSI=board.B15, MISO=board.B14)
ecs = digitalio.DigitalInOut(board.B12)    # Chip Select pin
dc = digitalio.DigitalInOut(board.B1)      # Data/Command selector
srcs = None                                # can be None to use internal memory
rst = digitalio.DigitalInOut(board.B0)     # can be None to not use this pin
busy = digitalio.DigitalInOut(board.B10)   # can be None to not use this pin

# give them all to our drivers
print("Creating display")
display = Adafruit_IL0373(128, 296,        # 2.9" Tri-color display
    spi,
    cs_pin=ecs,
    dc_pin=dc,
    sramcs_pin=srcs,
    rst_pin=rst,
    busy_pin=busy,
)

# Set rotation
display.rotation = 3  # 0 narrow side (MHT-ET Live sign) at top
                      # 1 pins at 6 o'clock, 3 pins at 12 o'clock
```

epd_bitmap.py 4/2.

```
FILENAME = "logo.bmp"                    # A kirajzolandó képfájl

def read_le(s):
    return int.from_bytes(s,'little') # Bájtok egésszé konvertálása (little endian)

class BMPError(Exception):
    pass

def display_bitmap(epd, filename):
    try:
        f = open(filename, "rb")
    except OSError:
        print("Couldn't open file")
        return

    print("File opened")
    try:
        if f.read(2) != b'BM':                  # check signature
            raise BMPError("Not BitMap file")

        bmpFileSize = read_le(f.read(4))
        f.read(4)                               # Read & ignore creator bytes
        bmpImageoffset = read_le(f.read(4))  # Start of image data
        headerSize = read_le(f.read(4))
        bmpWidth = read_le(f.read(4))
        bmpHeight = read_le(f.read(4))
        flip = True
```

epd_bitmap.py 4/3.

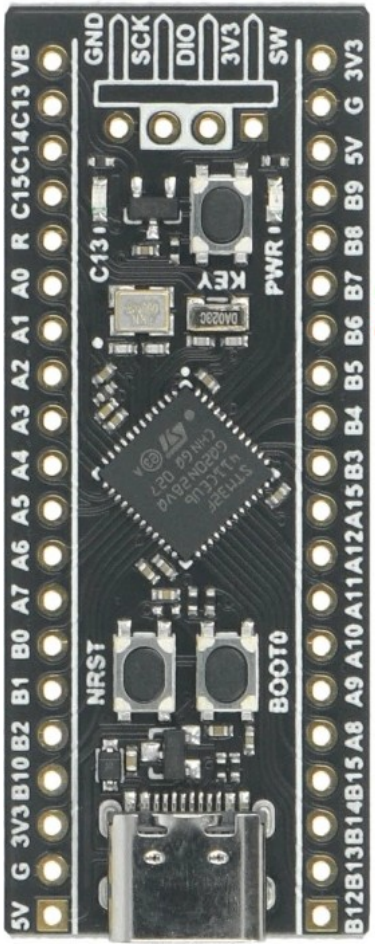
```
        print("Size: %d\nImage offset: %d\nHeader size: %d" %
              (bmpFileSize, bmpImageoffset, headerSize))
        print("Width: %d\nHeight: %d" % (bmpWidth, bmpHeight))
        if read_le(f.read(2)) != 1:   raise BMPError("Not singleplane")
        bmpDepth = read_le(f.read(2))  # bits per pixel
        print("Bit depth: %d" % (bmpDepth))
        if bmpDepth != 24:
            raise BMPError("Not 24-bit")
        if read_le(f.read(2)) != 0:
            raise BMPError("Compressed file")
        print("Image OK! Drawing...")
        rowSize = (bmpWidth * 3 + 3) & ~3  # 32-bit line boundary
        for row in range(bmpHeight):  # For each scanline...
            if flip:  # Bitmap is stored bottom-to-top order (normal BMP)
                pos = bmpImageoffset + (bmpHeight - 1 - row) * rowSize
            else:  # Bitmap is stored top-to-bottom
                pos = bmpImageoffset + row * rowSize
            # print ("seek to %d" % pos)
            f.seek(pos)
            rowdata = f.read(3*bmpWidth)
            for col in range(bmpWidth):
                b, g, r = rowdata[3*col:3*col+3]  # BMP files store RGB in BGR
                if r < 0x80 and g < 0x80 and b < 0x80:
                    epd.pixel(col, row, Adafruit_EPD.BLACK)
                elif r >= 0x80 and g >= 0x80 and b >= 0x80:
                    pass  # epd.pixel(row, col, Adafruit_EPD.WHITE)
                elif r >= 0x80:
                    epd.pixel(col, row, Adafruit_EPD.RED)
```

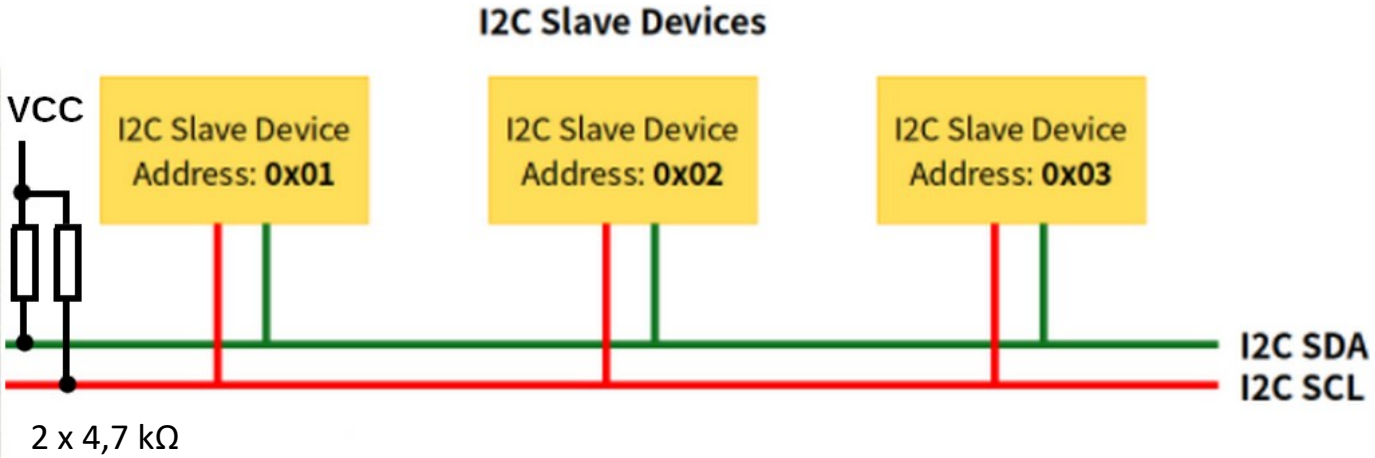
epd_bitmap.py 4/4.

```
    except OSError:
        print("Couldn't read file")
    except BMPError as e:
        print("Failed to parse BMP: " + e.args[0])
    finally:
        f.close()
    print("Finished drawing")

display.fill(Adafruit_EPD.WHITE)
display_bitmap(display, FILENAME)      # Figyelem! Ez nem a display objektum metódusa!
display.display()
```

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 osztály tagfüggvényei

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

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

i2c_scan.py

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

```
import time
import board
i2c = board.I2C()           # To use default I2C bus (most boards)
# To create I2C bus on specific pins
# import busio
# i2c = busio.I2C(board.B10, board.B9)
                            (SCL, SDA)
while not i2c.try_lock():
    pass

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

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

```
Adafruit CircuitPython REPL          (DS3231)

Auto-reload is on. Simply save files over
code.py output:
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
I2C addresses found: ['0x57', '0x68']
```

i2c_tsl2561.py

- Az alábbi kis programmal kiírjuk egy **TSL2561** fénymérővel mért értékeket (a program forrása: CircuitPython Essentials, de abban más típusú szenzor és más I2C cím szerepelt)

```
import time
import board
import adafruit_tsl2561  # Ezt a könyvtárat telepíteni kell!

i2c = board.I2C()
# Lock the I2C device before we try to scan
while not i2c.try_lock():
    pass
# Print the addresses found once
print("I2C addresses found:", [hex(device_address) for device_address in i2c.scan()])

# Unlock I2C now that we're done scanning.
i2c.unlock()

# Create library object on our I2C port
tsl2561 = adafruit_tsl2561.TSL2561(i2c, 0x29)

# Use the object to print the sensor readings
while True:
    print("Lux:", tsl2561.lux)
    time.sleep(2)
```

```
Adafruit CircuitPython REPL
code.py output:
I2C addresses found: ['0x29']
Lux: 104.29
Lux: 108.668
Lux: 108.668
Lux: 105.106
Lux: 43.5937
Lux: 34.0753
Lux: 16.3405
Lux: 107.142
```

OLED kijelzők az I2C buszon

- Az **I2C OLED** grafikus kijelzőkben többnyire **SSD1306** vagy **SH1106** vezérlő található (létezik SPI változat 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”
- I2C cím **0x3C** (0x3D)
- 3,3V-5.0V tápfeszültség
- Normál/inverz mód
- Grafikus megjelenítés
- Szoftveresen állítható kontraszt
- Tükrözés/elforgatás

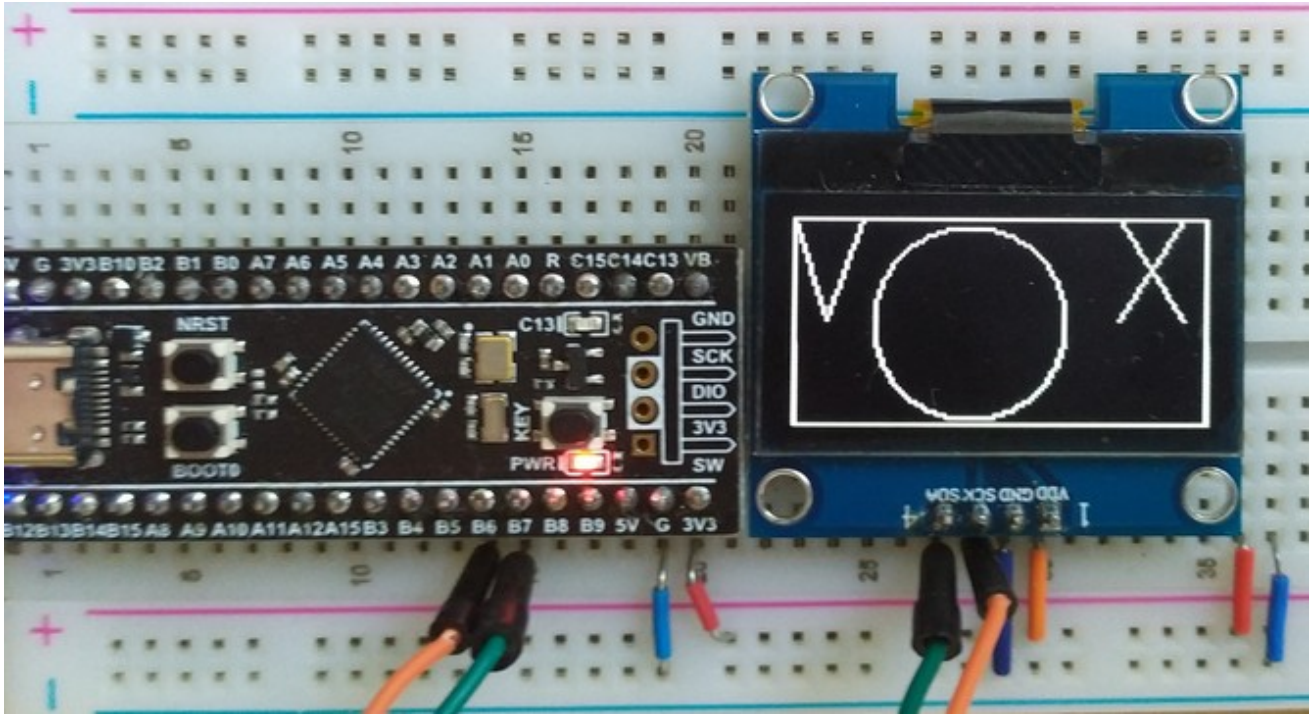
OLED kijelzők: „Két út van előttem…”

- Az egyszerűbb lehetőség az **adafruit_ssd1306** meghajtó használata, amely a **busio** modul és az **adafruit_framebuf** (utóbbit telepíteni kell!) segítségével vezérli a kijelzőt (lehet akár **I2C**, akár **SPI** illesztőjű)
- Mivel **adafruit_sh1106** meghajtó nem állt rendelkezésre, ezért az **adafruit_ssd1306** mintájára elkészítettük (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 132x64 felbontásúnak definiálni és csak a középső 128x64 képpontot használni)
- A **displayio** megjelenítés használatakor a soros porti üzenetek a kijelzőn is megjelennek!

oled_simpletest.py

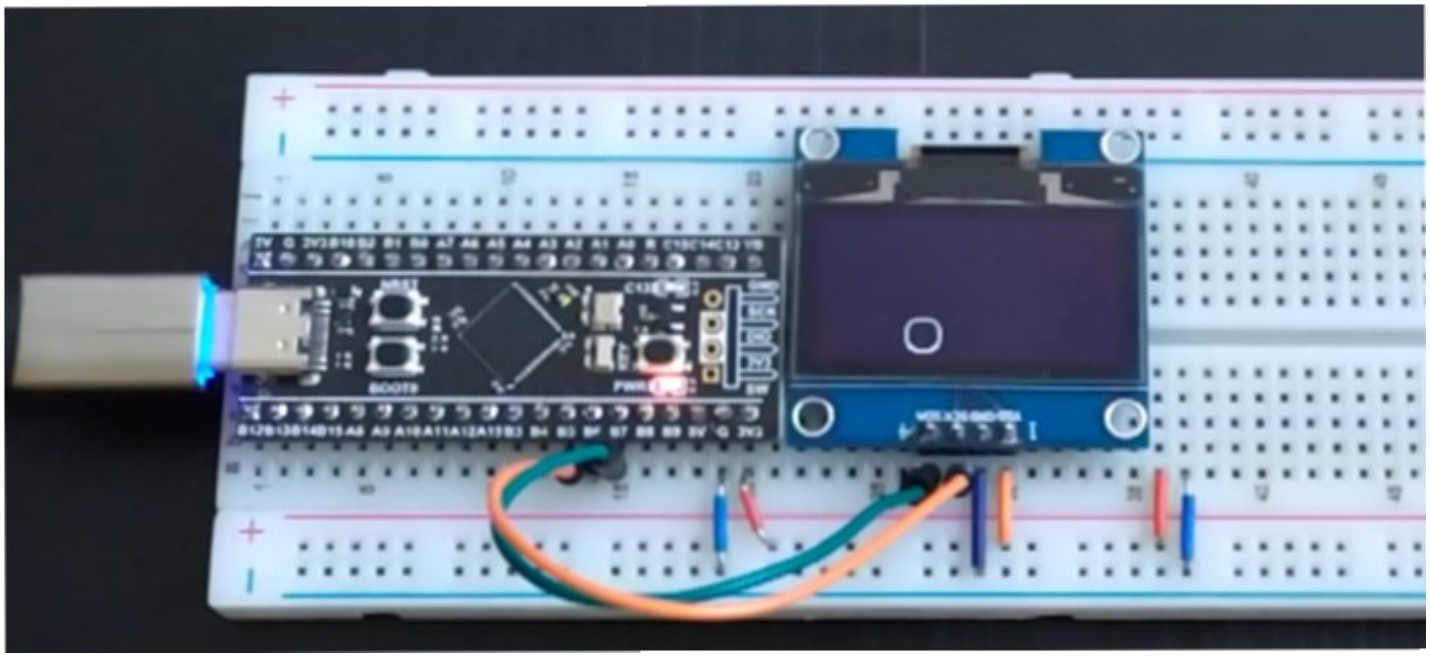
```
import board
import busio
import adafruit_sh1106
import time
# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA, frequency=400000)
# Create the SH1106 OLED class.
oled = adafruit_sh1106.SH1106_I2C(128, 64, i2c, addr=0x3c, external_vcc=False)

oled.rotate(0)
oled.fill(0)
oled.line(0,0,10,31,1)
oled.line(10,31,21,0,1)
oled.line(100,0,120,31,1)
oled.line(100,31,120,0,1)
oled.circle(54,32,30,1)
oled.rect(0,0,128,64,1)
oled.show()
while True:
    oled.contrast(0x60)
    time.sleep(2)
    oled.invert(1)
    time.sleep(2)
    oled.invert(0)
    time.sleep(2)
    oled.contrast(0x90)
    time.sleep(2))
```

A line, circle, rect és hasonló objektumok rajzolását az adafruit_framebuf támogatja



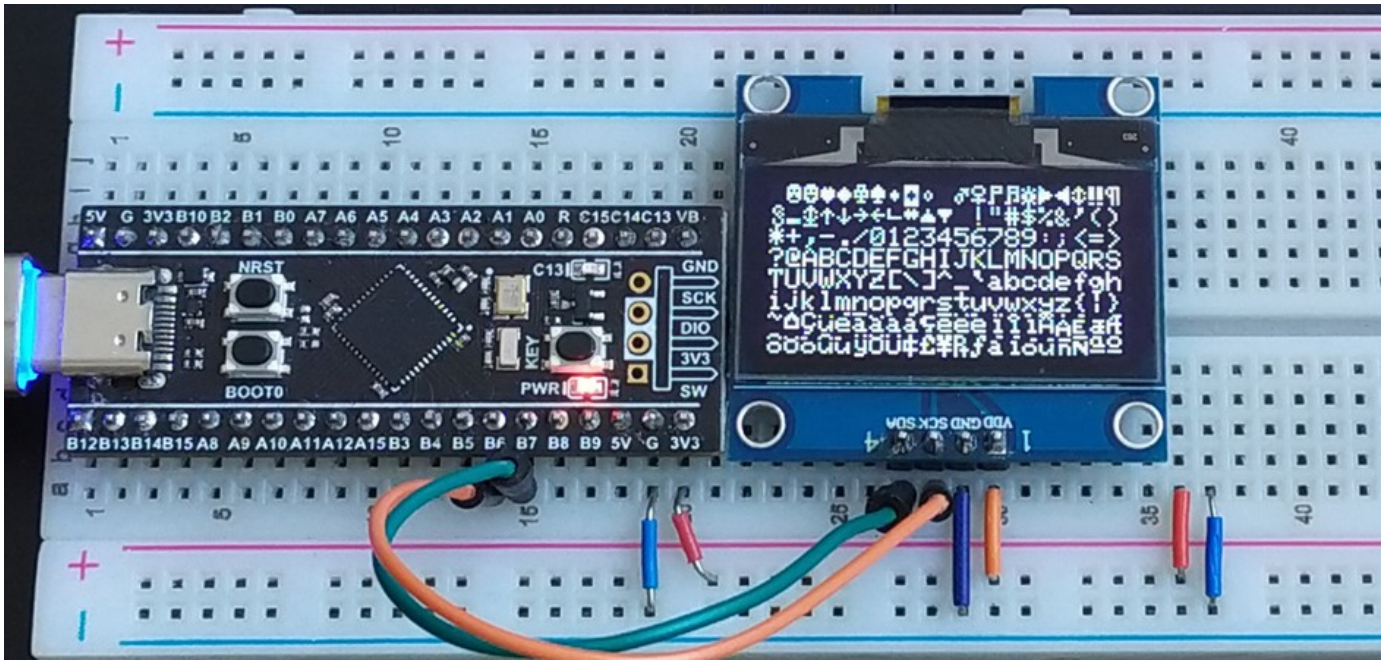
SH1106_bouncing_ball.py

- Az **Adafruit_CircutPython_SSD1306** programkönyvtár **ssd1306_bouncing_ball.py** mintapéldájának átdolgozott változatát nem akarom részletesen ismertetni, csupán azért teszem közzé, mert néhány hibát ki kellett javítani benne
- A javítások mellett a saját körrajzoló függvény helyett az **adafruit framebuf**-tól megörökölt körrajzoló függvényét használjuk
- Az **I2C** busz frekvenciájának megnövelésével és a lépésköz megduplázásával az animációt felgyorsítottak



framebuffer_test.py

- Ehhez a programhoz az Adafruit_CircutPython_SSD1306 programkönyvtár **ssd1306_framebuftest.py** mintapéldáját használtuk fel, amely a rajzolási és szövegmegjelenítési funkciókat mutatja be
- **Követelmények:**
  adafruit_framebuf.mpy (lib mappában)
  adafruit_ssd1306.mpy vagy **adafruit_sh1106.py** (lib mappában)
  font5x8.bin (a code.py állomány mellett)

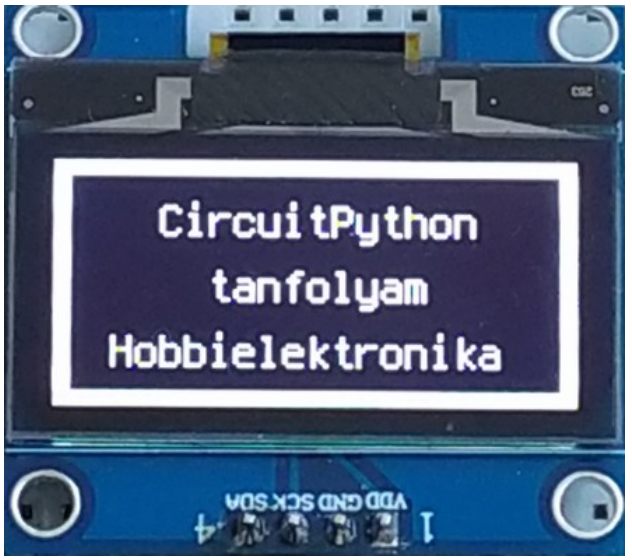


OLED kijelzők displayio támogatása

- A következő mintapéldában a **displayio** könyvtárat fogjuk használni, amely eleve része a CircuitPython firmware csomagnak
- A kijelző kezeléséhez azonban telepítenünk kell az **adafruit_displayio_ssd1306.mpy** vagy **adafruit_displayio_sh1106.mpy** könyvtárat (az Adafruit CircuitPython Bundle tartalmazza)
- **Megjegyzés:** ügyeljünk az elnevezésre, ne keverjük össze ezek nevét az előző programoknál használt **displayio** nélküliekkel!
- Telepítenünk kell az **adafruit_display_text** programkönyvtárat is
- A **SH1106_displayio_demo.py** példaprogram a kijelző inicializálása után kirajzol egy nagy fehér téglalapot, annak közepében egy fekete kitöltött téglalapot, s abban kiír egy szöveget
- A program forrása: az adafruit_displayio_ssd1306 könyvtár mintaprogramja, amelyen csak apróbb módosításokat eszközöltünk

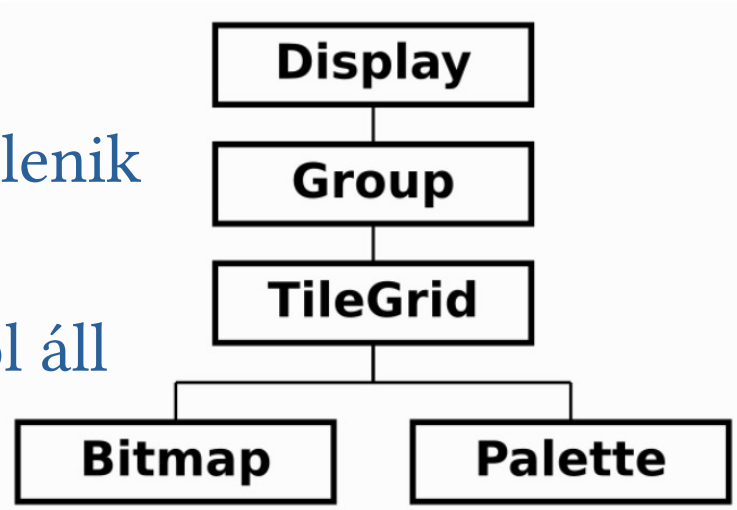
SH1106_displayio_test.py

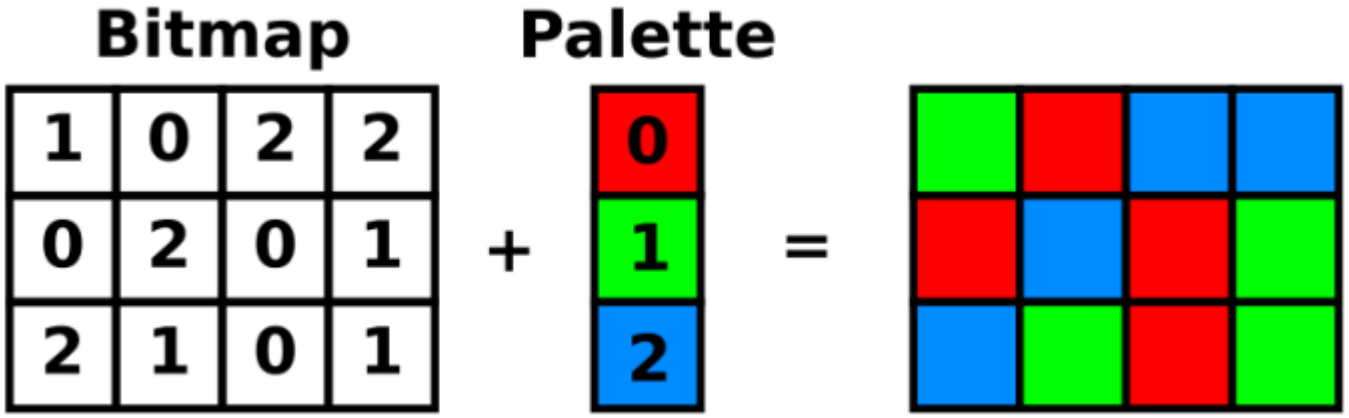
```
import board
import displayio
import terminalio
from adafruit_display_text import label
from adafruit_displayio_sh1106 import SH1106
displayio.release_displays() # Elengedjük a kijelzőt...
i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3C)
display=SH1106(display_bus,width=132,height=64,rotation=180)

splash = displayio.Group()
display.show(splash)
color_bitmap = displayio.Bitmap(128, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF  # White
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=2, y=0)
splash.append(bg_sprite)
inner_bitmap = displayio.Bitmap(120, 56, 1) # Draw a smaller inner rectangle
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000  # Black
inner_sprite = displayio.TileGrid(inner_bitmap, pixel_shader=inner_palette, x=6, y=4)
splash.append(inner_sprite)
display.show(splash)
text = "   CircuitPython\r\n     tanfolyam\r\n Hobbielektronika"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=10, y=15)
splash.append(text_area)

while True: pass
```



displayio modul osztályai

- A **displayio** modul API dokumentációja
- A **Group** (vagy a legfelső Group) végeredményben az, ami a kijelzőn megjelenik a **show()** metódus meghívásakor
- A **TileGrid Bitmap** és **Palette** elemekből áll
- **Bitmap** és **Palette** kapcsolata:





SH1106_displayio_test.py – hogy működik?

- Ez az általános gyűjtő objektum:

```
splash = displayio.Group()
display.show(splash)
```

- A befoglaló fehér téglalap:

```
color_bitmap = displayio.Bitmap(128, 64, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0xFFFFFF  # White
bg_sprite = displayio.TileGrid(color_bitmap,
pixel_shader=color_palette, x=2, y=0)
splash.append(bg_sprite)
```

SH1106 esetén eltolás kell

- A belső fekete téglalap:

```
inner_bitmap=displayio.Bitmap(120,56,1) # Draw smaller rectangle
inner_palette = displayio.Palette(1)
inner_palette[0] = 0x000000  # Black
inner_sprite = displayio.TileGrid(inner_bitmap,
pixel_shader=inner_palette, x=6, y=4)
splash.append(inner_sprite)
```

SH1106 esetén eltolás kell

SH1106_displayio_test.py – hogy működik?

- Szövegcímke (Text Label) rajzolása:

```
text = "   CircuitPython\r\n     tanfolyam\r\n Hobbielektronika"
text_area = label.Label(terminalio.FONT, text=text, color=0xFFFF00, x=10, y=15)
splash.append(text_area)
```

- A CircuitPython Display_Text Library tananyag szerint az **adafruit_display_text** modul kétféle szövegcímke típust kezel, de nem világos, hogy hogyan lehet választani. Lehetséges, hogy a

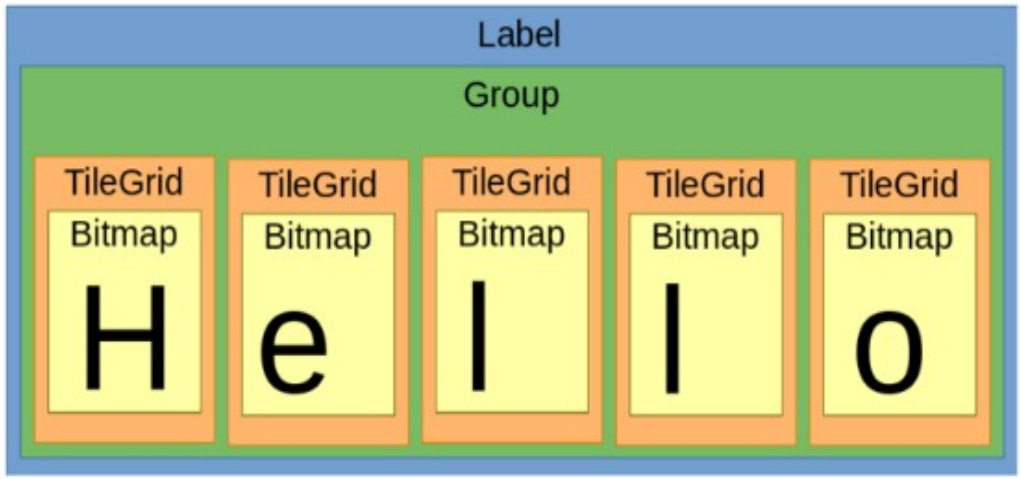
```
from adafruit_display_text import label
```

helyett csupán

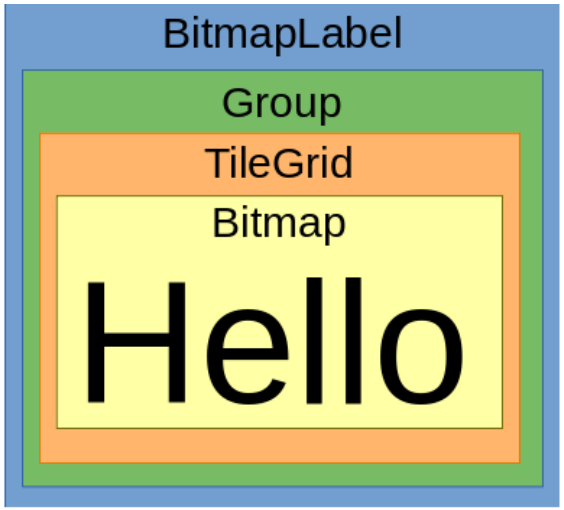
```
from adafruit_display_text import bitmap_label
```

kell?

Label (hagyományos)



BitmapLabel

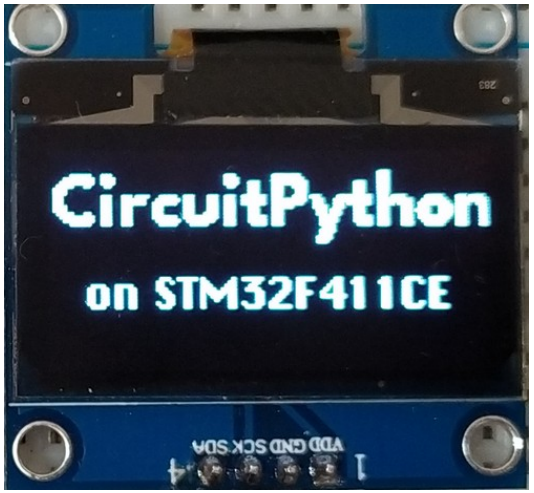


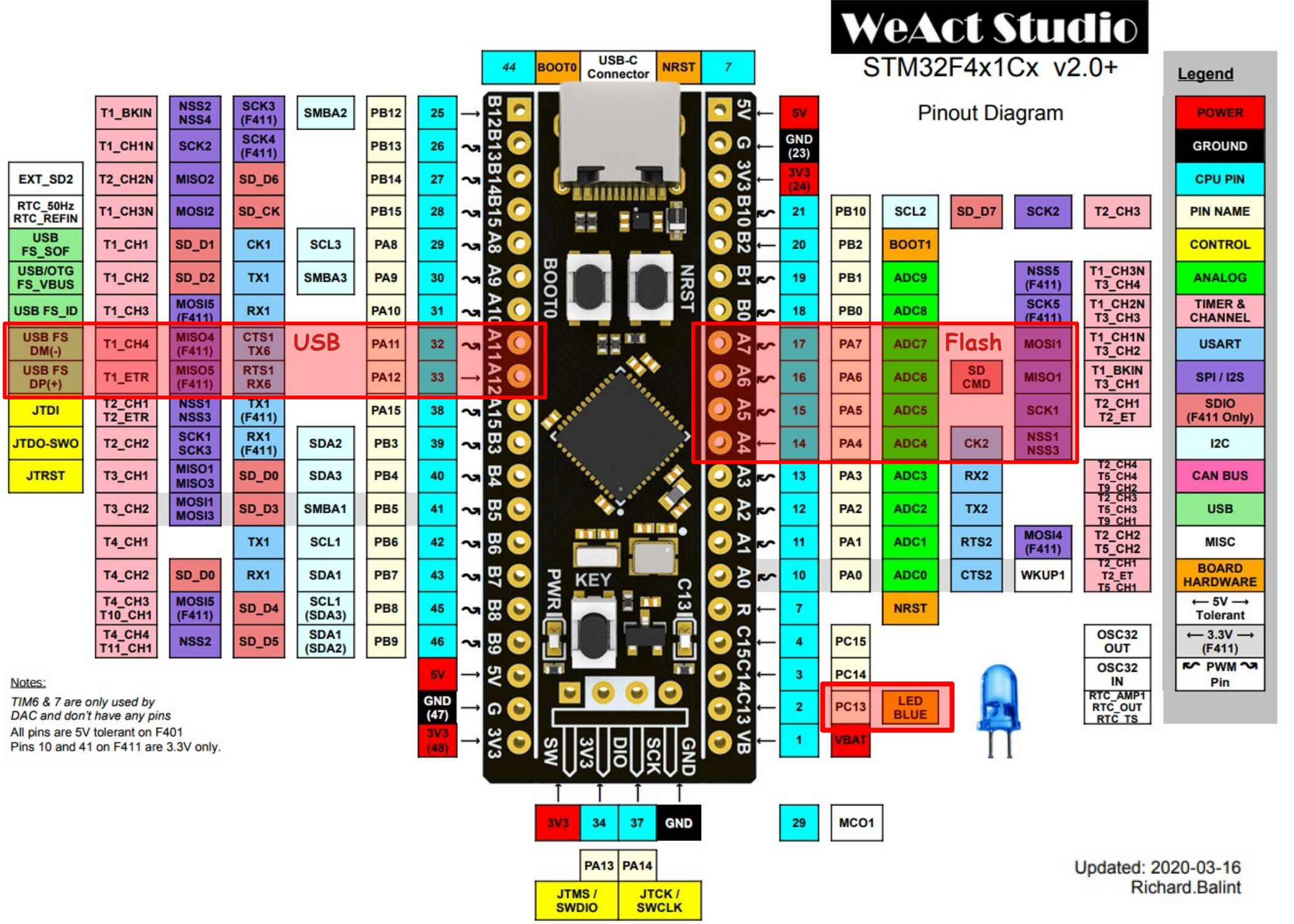
Bitmap fontok használata

- A **displayio** kompatibilis kijelzőkön tetszés szerinti fontokat is használhatunk, egy lehetőséget az alábbi példában mutatunk be, ahol a kirajzolandó szöveg egy Text Label elemként kezelhető
- A mintapéldát az **adafruit_display_text** modul display_text_bitmap_label_simpletest.py mintapéldájából kiindulva készítettük el
- **Követelmények:** telepíteni kell az **adafruit_display_text** és az **adafruit_bitmap_font** könyvtárakat, továbbá a **fonts** mappába a **Chicago-12.bdf**, valamint a **LeagueSpartan-Bold-16.bdf** fontokat
- Természetesen telepíteni kell a használni kívánt kijelző **displayio** kompatibilis meghajtóját is (**adafruit_displayio_sh1106.mpy**)

bitmap_font_label_simpletest.py

```
import board, displayio, terminalio
from adafruit_display_text import label
from adafruit_displayio_sh1106 import SH1106
from adafruit_bitmap_font import bitmap_font

i2c = board.I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3C)
display = SH1106(display_bus, width=132, height=64, rotation=180)

big_font = bitmap_font.load_font('fonts/LeagueSpartan-Bold-16.bdf')
small_font = bitmap_font.load_font('fonts/Chicago-12.bdf')
color = 0xFFFFFF

text1 = label.Label(big_font,text="CircuitPython",color=color)
text1.x = 2
text1.y = 5

text2 = label.Label(small_font,text="on STM32F411CE",color=color)
text2.x = 2
text2.y = 32

splash = displayio.Group()
display.show(splash)
splash.append(text1)
splash.append(text2)

while True:
    pass
```




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
LED BLUE
SW
3V3
DIO
SCK
GND
JTMS / SWDIO
JTCK / SWCLK
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