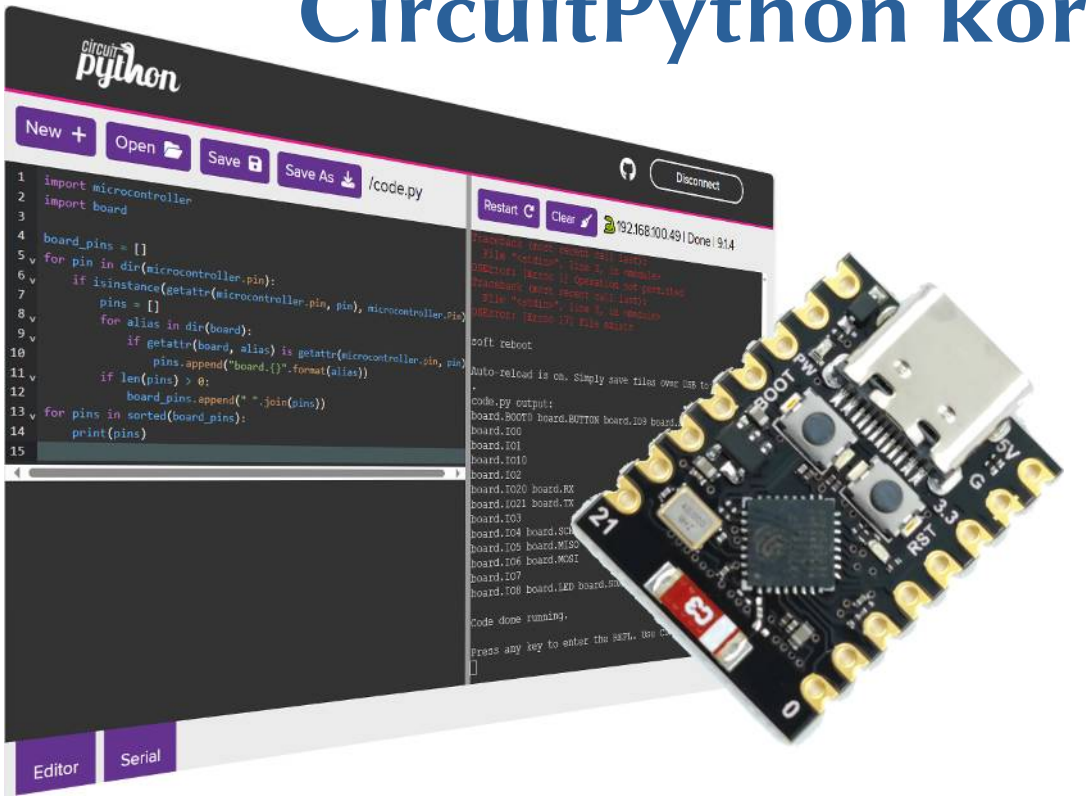
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



5. E-papír 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

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

Felhasznált és ajánlott irodalom

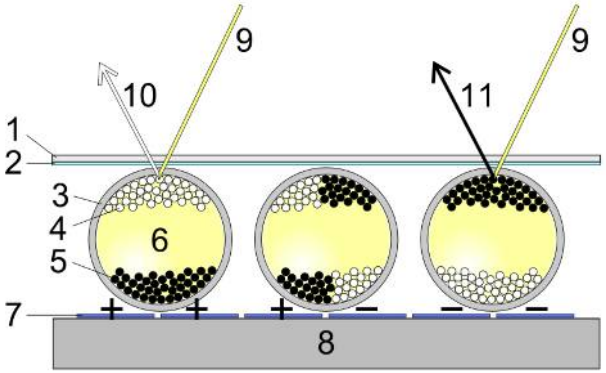
- **Tananyagok, útmutatók:**
  - Carter Nelson: CircuitPython Display Support Using displayio
  - Tim C.: CircuitPython Display_Text Library
  - Ruiz Brothers: Custom Fonts for CircuitPython Displays
  - Anna Barela: Creating Slideshows in CircuitPython

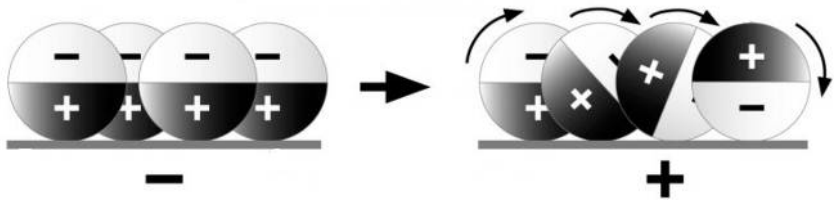
- **CircuitPython kiegészítő programkönyvtárak:**
  - Adafruit_IL0373 – displayio driver for IL0373 2.9” E-paper displays
  - Adafruit_CircuitPython_Display_Text – text labels
  - Adafruit_CircuitPython_Bitmap_Font – custom font support
  - Adafruit_CircuitPython_ImageLoad – load image files
  - Adafruit_CircuitPython_Display_Shapes – lines, circles, triangles etc.

- **Adatlapok és dokumentáció:**
  - 2.9inch e-Paper Module (B) Manual

E-papír kijelzők

- 1970-es évek: **E-papír** (elektronikus papír) fejlesztése (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)





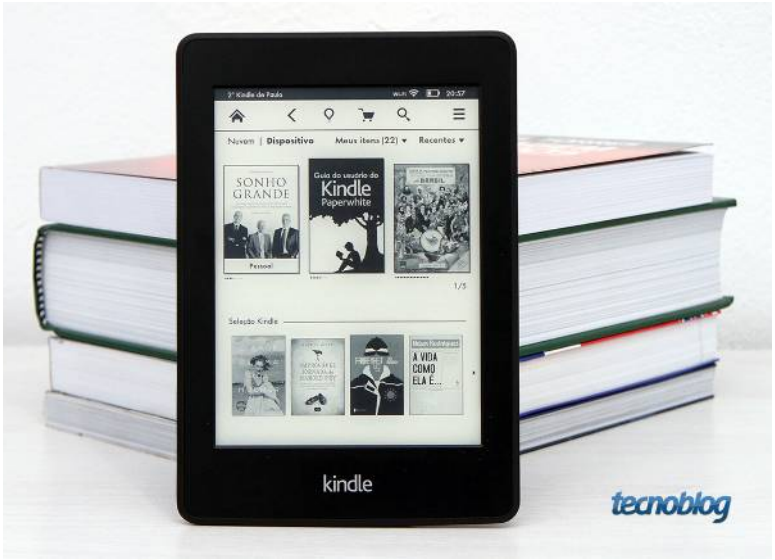
Gyricon e-ink működési elve

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 kijelzők: előnyök és hátrányok

- 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

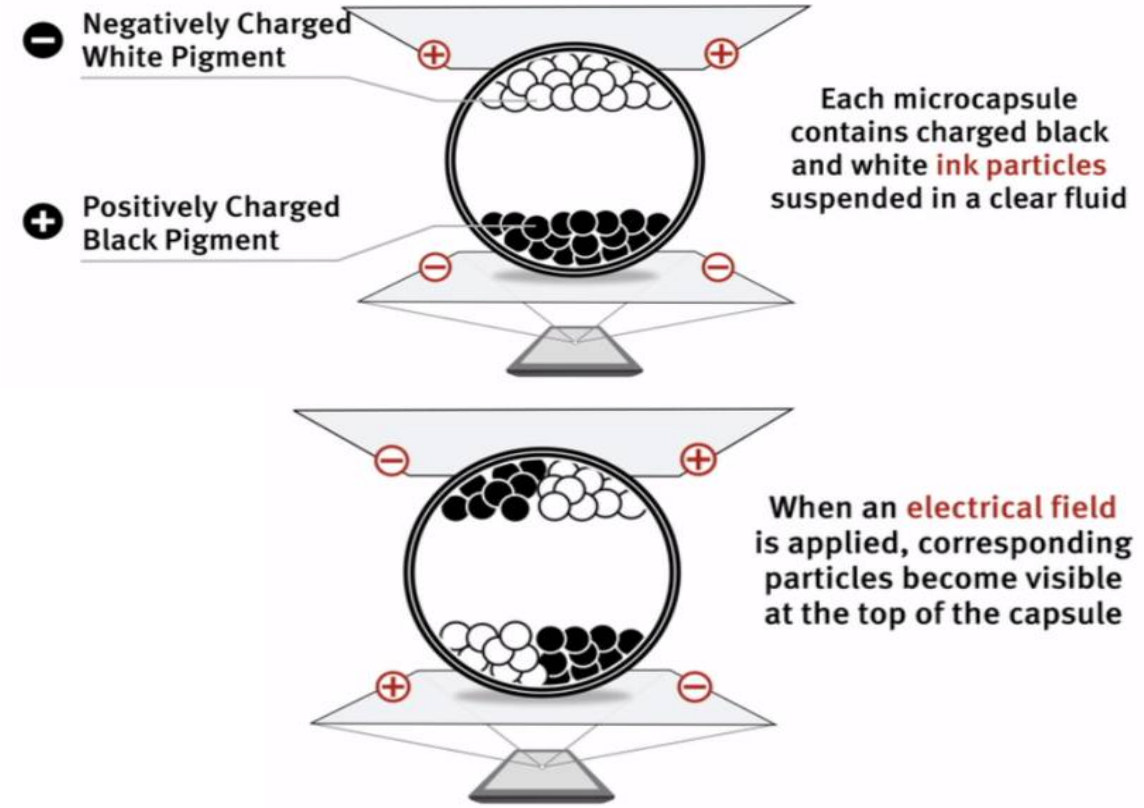
E-papír kijelzők alkalmazásai

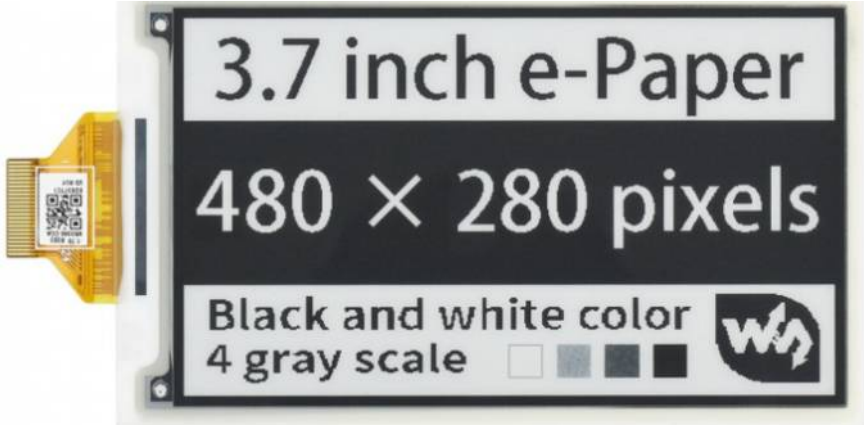


Yotaphone 2

Modern E-ink technológiák

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

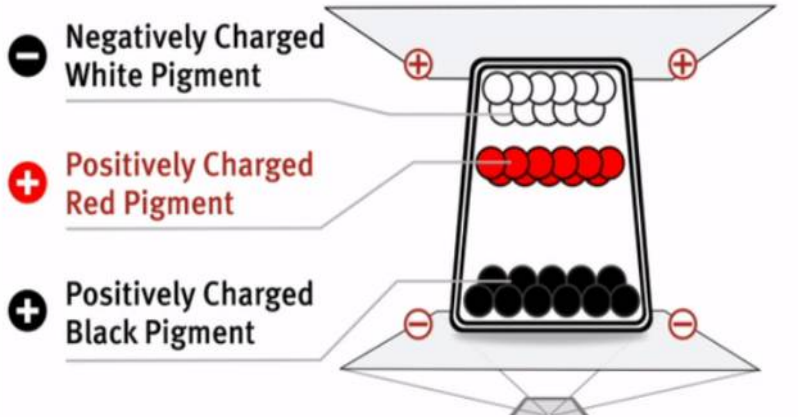




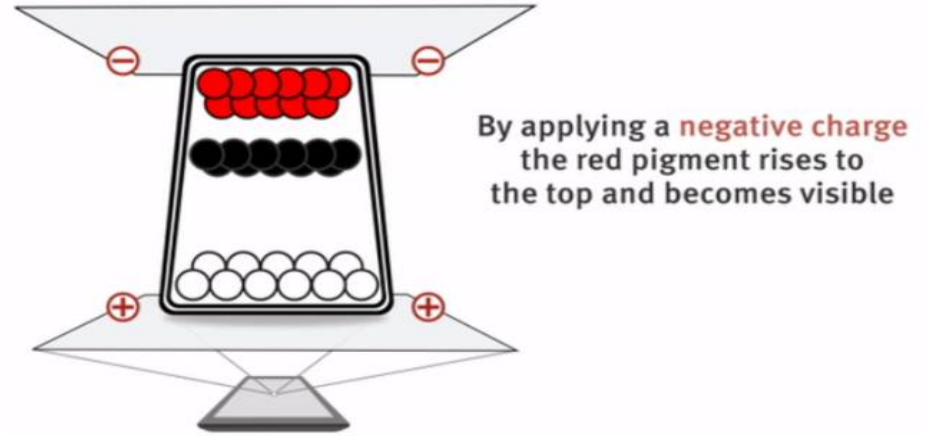
Egy konkrét megvalósítás

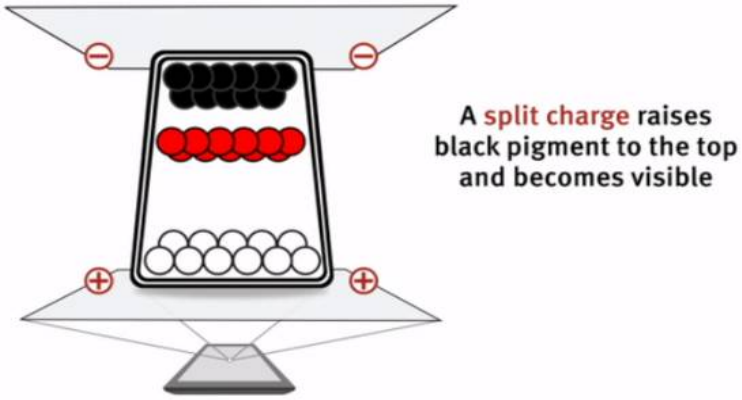
Háromszínű kijelző, három pigmenttel

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

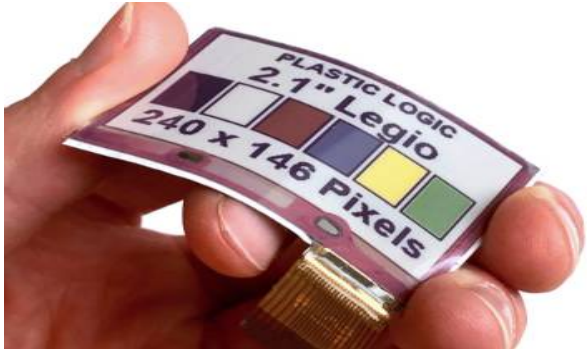


Egy konkrét megvalósítás



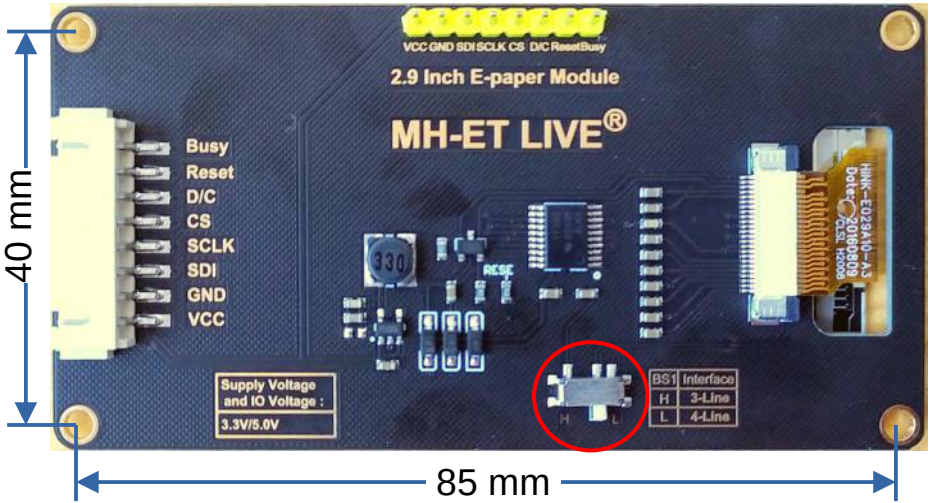


Hobbi célra alkalmas olcsó kijelzők



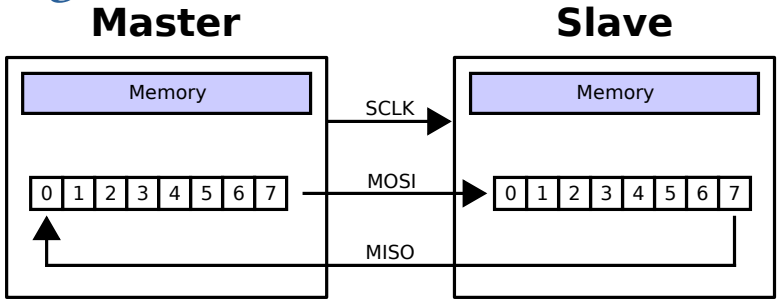
Az MH-ET LIVE kijelző kártya

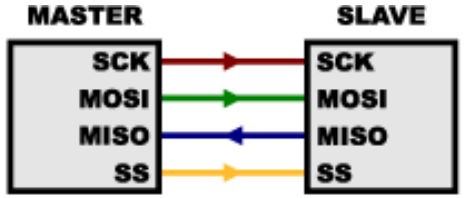
- Kijelző méret: **2,9”** (66,89x29,05 mm)
- Felbontás: **296x128**
- Betekintési szög: 170°
- Szín: fekete, fehér, piros v. sárga
- A kártyán szintátalakító is van
- Interfész: **SPI** 4 vagy 3 vezetékes
  (utóbbi esetben 9 bitet kell küldeni
  a **D/C** vonal vezérlése helyett)
- **Reset** és **Busy** 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

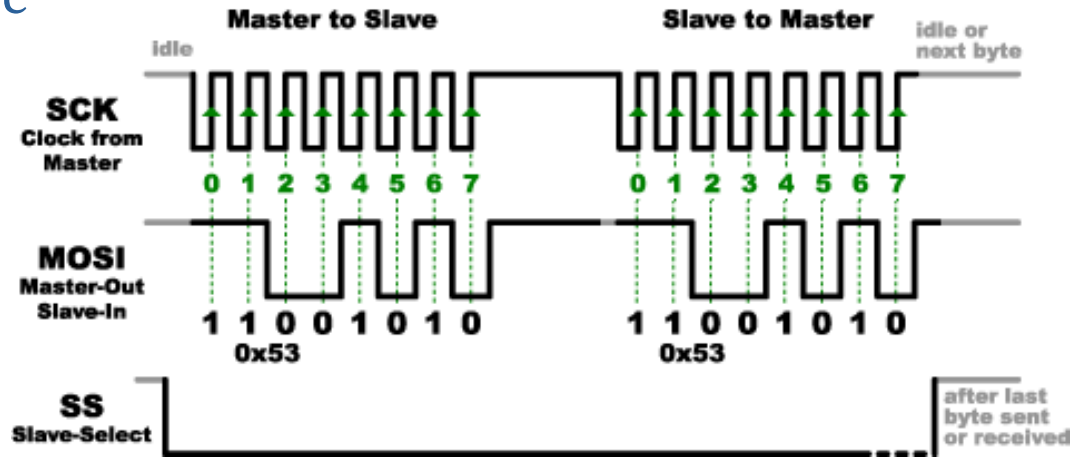


Az SPI kommunikációs csatorna

- 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 a két léptető-regiszter adatot cserél



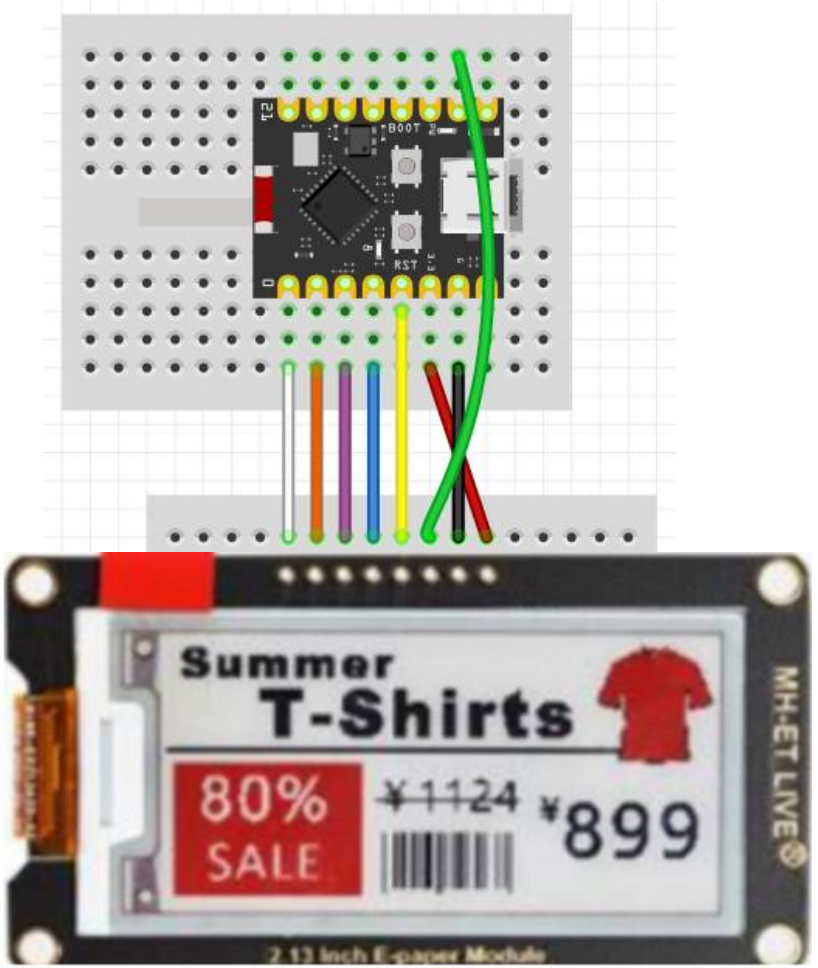




A javasolt bekötési vázlat

- A kijelző kivezetéseit így kötöttük be:

VCC	3V3	Tápfeszültség
GND	GND	Közös pont
SDI	IO6	SPI MOSI
SCK	IO4	SPI órajel
CS	IO3	SPI kiválasztó jel
D/C	IO2	Adat/Parancs választó
RST	IO0	Hardver reset
Busy	IO0	Foglaltság jelzése



E-Paper használata CircuitPythonban – két megközelítés

- **1. Adafruit_EPD + Adafruit_IL0373** (framebuffer-alapú, régebbi megközelítés)
- A teljes képet egy RAM-ban lévő ***framebuffer*** tárolja
- Előny: egyszerű API, teljes kontroll a pixeladatok felett
- Hátrány: nagy RAM-igény, nincs ***displayio*** integráció
- **Megjegyzés:** ez a módszer már kevésbé támogatott, új fejlesztéshez nem ajánlott

- **2. Displayio + Adafruit_IL0373** (réteges, objektumalapú grafika)
- Előny: szöveg, font, ikonok, képek könnyen kombinálhatók
- Hátrány: bonyolult API, nincs közvetlen parancsküldés, nincs power-off API

- **Megjegyzés:** kétféle **Adafruit_IL0373** könyvtár van, ezeket ne keverjük össze! A *framebuffer* alapú meghajtó az **Adafruit_EPD** könyvtár része. A különálló **Adafruit_IL0373** könyvtár pedig a *displayio* könyvtárral használható

il0373_2.9_color.py – 2/1.

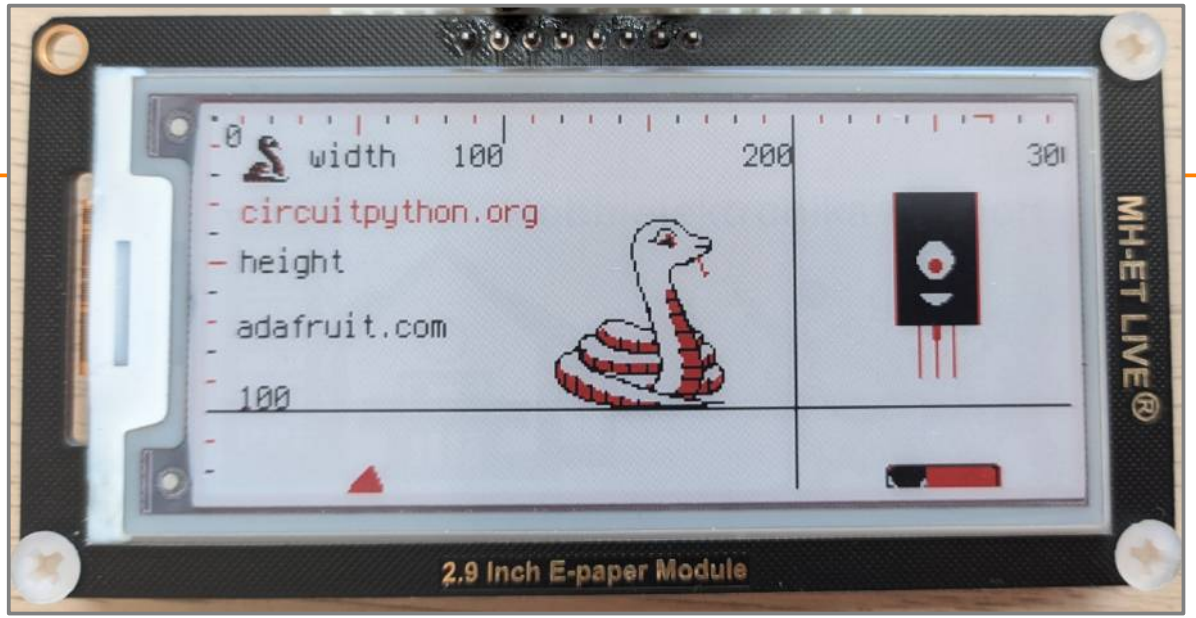
- Az **Adafruit_IL0373** mintapéldája **displayio** módszerrel inicializálja az E-papír kijelzőt és rátölti a **display_ruler.bmp** nevű bitmap képfájlt (már amennyi ráfér belőle a kijelzőre)
- Az itt látható kijelző inicializálást fogjuk használni a többi példaprogramban is

```
import time, board, displayio, fourwire
import adafruit_il0373
displayio.release_displays()

spi = board.SPI()       # Uses SCK=board.IO4 and MOSI=board.IO6
epd_cs = board.IO3      # SPI chip select
epd_dc = board.IO2      # Data/command selector
epd_reset = board.IO1   # Hardware reset
epd_busy = board.IO0    # Busy signal

display_bus=fourwire.FourWire(spi,command=epd_dc,chip_select=epd_cs,reset=epd_reset,baudrate=1000000)
time.sleep(1)       # Wait a bit

# Create the display object - landscape mode, the third color is red (0xff0000)
display = adafruit_il0373.IL0373(display_bus, width=296, height=128, rotation=270,
    busy_pin=epd_busy, highlight_color=0xFF0000)
```

il0373_2.9_color.py – 2/2.

```
# Create a display group for our screen objects
g = displayio.Group()

# Display a ruler graphic from the root directory of the CIRCUITPY drive
pic = displayio.OnDiskBitmap("/display_ruler.bmp")
t = displayio.TileGrid(pic, pixel_shader=pic.pixel_shader)
g.append(t)

display.root_group = g
print("root_group set")
display.refresh()
```

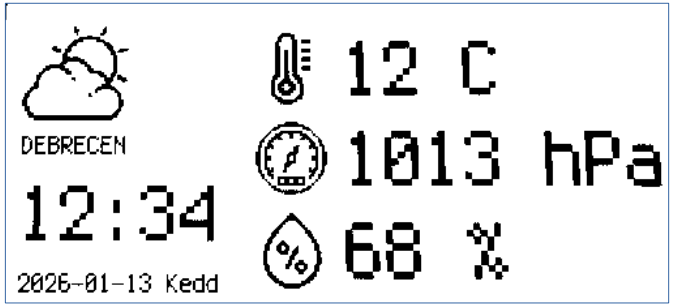
A kijezőről közelítőleg leolvashatjuk, hogy valóban 296x128 px a felbontás és láthatóan működik a három szín megjelenítése



dashboard_demo.py – 4/1.

- Az alábbi program egy képzeletbeli időjárás-áttekintő kijelző elrendezését és megvalósítását mutatja be, a ***displayio*** rétegzett grafikus modelljére építve, ikonok, szövegek és háttérelemek kombinálásával

```
import time, board, displayio, terminalio, fourwire
import adafruit_il0373
from adafruit_display_text import label
displayio.release_displays()

spi = board.SPI()       # Uses SCK=board.IO4 and MOSI=board.IO6
epd_cs = board.IO3      # SPI chip select
epd_dc = board.IO2      # Data/command selector
epd_reset = board.IO1   # Hardware reset
epd_busy = board.IO0    # Busy signal

display_bus=fourwire.FourWire(spi,command=epd_dc,chip_select=epd_cs,reset=epd_reset,baudrate=1000000)
time.sleep(1)       # Wait a bit

# Create the display object - landscape mode, the third color is red (0xff0000)
display = adafruit_il0373.IL0373(display_bus, width=296, height=128, rotation=270,
    busy_pin=epd_busy, highlight_color=0xFF0000)
```



dashboard_demo.py – 4/2.

- Ha fehér hátteret szeretnénk, akkor egy fehér csempe legyen a legalsó réteg

```
# ----------------------------------------
# 2. ROOT GROUP + FEHÉR HÁTTÉR
# ----------------------------------------

root = displayio.Group()

# Háttér bitmap (fehér)
palette = displayio.Palette(3)
palette[0] = 0xFFFFFF   # fehér
palette[1] = 0x000000   # fekete
palette[2] = 0xFF0000   # piros

bg_bitmap = displayio.Bitmap(296, 128, 1)
bg_bitmap.fill(0)  # fehér
bg = displayio.TileGrid(bg_bitmap, pixel_shader=palette)
root.append(bg)
```

```
KÉPELEMEK HIERARCHIÁJA (displayio struktúra):

  display.root_group
  └── root (Group)
      ├── bg (TileGrid: fehér háttér)
      ├── icon (TileGrid: bal oldali időjárás ikon)
      ├── city (Label: városnév)
      ├── time_label (Label: idő, piros)
      ├── date_label (Label: dátum)
      ├── [add_row(…)] hőmérséklet ikon + Label
      ├── [add_row(…)] légnyomás ikon + Label
      └── [add_row(…)] páratartalom ikon + Label
```

dashboard_demo.py – 4/3.

- A baloldali elemek: egy ikon és három szövegcímke (a középső 3x nagyítással)
  Az egyszerűség kedvéért a beépített **terminalio.FONT**-ot használjuk

```
# Időjárás ikon (48 x 48 px)
weather_bmp = displayio.OnDiskBitmap("/icon1.bmp")
icon = displayio.TileGrid(weather_bmp,
pixel_shader=weather_bmp.pixel_shader, x=4, y=4)
root.append(icon)

# Helynév
city = label.Label(terminalio.FONT,text="DEBRECEN",
    color=0x000000, scale=1, x=4, y=60)
root.append(city)

# Idő (nagy)
time_label = label.Label(terminalio.FONT,text="12:34",
    color=0xFF0000, scale=3, x=4, y=90)
root.append(time_label)

# Dátum
date_label = label.Label(terminalio.FONT, text="2026-01-13 Kedd",
    color=0x000000, scale=1, x=4, y=120)
root.append(date_label)
```

```
KÉPELEMEK HIERARCHIÁJA (displayio struktúra):

  display.root_group
  └── root (Group)
      ├── bg (TileGrid: fehér háttér)
      ├── icon (TileGrid: bal oldali időjárás ikon)
      ├── city (Label: városnév)
      ├── time_label (Label: idő, piros)
      ├── date_label (Label: dátum)
      ├── [add_row(…)] hőmérséklet ikon + Label
      ├── [add_row(…)] légnyomás ikon + Label
      └── [add_row(…)] páratartalom ikon + Label
```

dashboard_demo.py – 4/4.

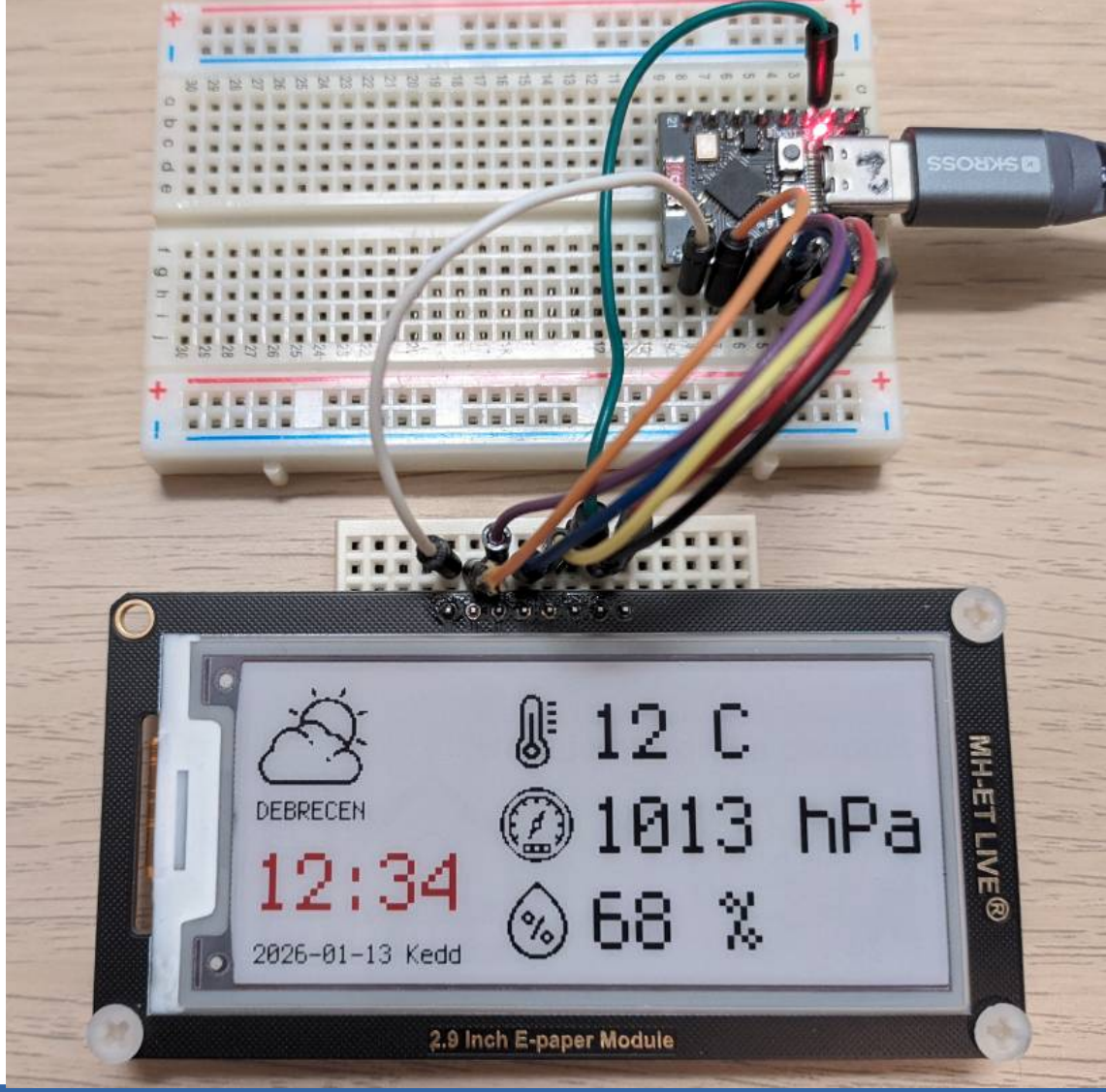
- A jobboldali elemek (egy-egy ikon és szövegcímke) egy paraméterezhető függvénnyel hatékonyan létrehozhatók

```
def add_row(y, bmp_filename, text):
    """Egy sor: ikon + nagy szöveg."""
    bmp = displayio.OnDiskBitmap(bmp_filename)
    ic = displayio.TileGrid(bmp, pixel_shader=bmp.pixel_shader,x=110,y=y)
    root.append(ic)

    lbl = label.Label(terminalio.FONT, text=text,
        color=0x000000, scale=3, x=150, y=y+16)
    root.append(lbl)

add_row(10, "/temp.bmp", "12 °C")
add_row(50, "/press.bmp", "1013 hPa")
add_row(90, "/humidity.bmp", "68 %")

# --- MEGJELENÍTÉS ---
display.root_group = root
while display.busy:
    pass
display.refresh()
```

```
KÉPELEMEK HIERARCHIÁJA (displayio struktúra):

  display.root_group
  └── root (Group)
      ├── bg (TileGrid: fehér háttér)
      ├── icon (TileGrid: bal oldali időjárás ikon)
      ├── city (Label: városnév)
      ├── time_label (Label: idő, piros)
      ├── date_label (Label: dátum)
      ├── [add_row(…)] hőmérséklet ikon + Label
      ├── [add_row(…)] légnyomás ikon + Label
      └── [add_row(…)] páratartalom ikon + Label
```

dashboard_demo.py – a végeredmény

- A felhasznált és a program mellé telepítendő ikonok:
  - icon1.bmp (48x48)
  - temp.bmp (32x32)
  - press.bmp (32x32)
  - humidity.bmp (32x32)



Adafruit_display_shapes

- Alakzatok rajzolásához az Adafruit_Display_Shapes könyvtárat használhatjuk (lásd: displayio UI quickstart, dokumentáció: Adafruit Display_Shapes Library)

Rajzelemek:

- **Line**(*x0,y0,x1,y1,color*) – végpontokkal adott szakasz rajzolása
- **Triangle**(*x0,y0,x1,y1,x2,y2,fill,outline)* – kitöltött vagy üres háromszög rajzolása (a *None* értékkel definált szín átlátszó)
- **Rect**(*x0,y0,width,height,fill,outline,stroke*) – (kitöltött) téglalap rajzolása
- **RoundRect**(*x0,y0,width,height,r,fill,outline,stroke*) – lekerekített sarkú (kitöltött) téglalap rajzolása (*r* – a sugár)
- **Circle**(*x0,y0,r,fill,outline,stroke*) – (kitöltött) kör rajzolása
- **Polygon**(*points,outline*) – poligon rajzolása (*points: (x,y)* tupletek listája)
- **Sparkline**(*width,height,max_items, y_min, y_max,x0,y0,color*) – egyszerű vonaldiagram rajzolása (**add_value**(*adat*) – adat hozzáfűzése)

bp_demo.py – 4/1.

Ismét egy fiktív alkalmazás (vérnyomás diagram) képernyőképét tervezzük és rajzoljuk meg. Mivel a tengelyek és a tengelyfeliratok nem változnak, ezeket háttérképként töltjük be

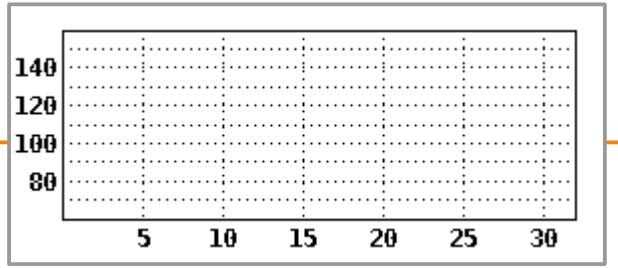
```
import time, board, displayio, terminalio, fourwire
import adafruit_il0373
from adafruit_display_text import label
from adafruit_display_shapes.circle import Circle
from adafruit_display_shapes.line import Line
displayio.release_displays()

spi = board.SPI()       # Uses SCK=board.IO4 and MOSI=board.IO6
epd_cs = board.IO3      # SPI chip select
epd_dc = board.IO2      # Data/command selector
epd_reset = board.IO1   # Hardware reset
epd_busy = board.IO0    # Busy signal

display_bus=fourwire.FourWire(spi,command=epd_dc,chip_select=epd_cs,reset=epd_reset,baudrate=1000000)
time.sleep(1)       # Wait a bit
# Create the display object - landscape mode, the third color is red (0xff0000)
display = adafruit_il0373.IL0373(display_bus, width=296, height=128, rotation=270,
    busy_pin=epd_busy, highlight_color=0xFF0000)
```

bp_demo.py – 4/2.

```
# --- MÉRET ÉS MARGÓK ---
W, H = 296, 128                    # A kijelző mérete pixelben (szélesség, magasság)
ML, MR, MT, MB = 26, 14, 12, 22    # Margók (bal, jobb, felső, alsó)

# --- SKÁLÁK ---
X_MIN, X_MAX = 0, 32               # X tengely skálája (napok)
Y_MIN, Y_MAX = 60, 160             # Y tengely skálája (vérnyomásértékek Hgmm-ben)

# --- HASZNOS RAJZTERÜLET ---
PW, PH = W - ML - MR, H - MT - MB  # Plot szélesség, magasság

# --- TRANSZFORMÁCIÓK ---
def x_to_px(day):                  # A nap (1..31) sorszámát vízszintes pixelpozícióra alakítja
    return int(ML + (day - X_MIN) * PW / (X_MAX - X_MIN))

def y_to_px(val):                  # A vérnyomásértéket függőleges pixelpozícióra alakítja
    return int(H - MB - (val - Y_MIN) * PH / (Y_MAX - Y_MIN))

# --- SYS ÉS DIA ÉRTÉKEK (31 nap) ---
sys_values = [ 132, 118, 140, 127, 136, 112, 144, 121, 129, 138, 115, 134, 142, 119, 131, 125,
    137, 113, 128, 133, 141, 117, 130, 139, 122, 135, 114, 143, 126, 120, 124]

dia_values = [ 82, 74, 88, 79, 85, 71, 90, 76, 81, 87, 72, 84, 89, 75, 83, 78,
    86, 70, 80, 82, 88, 73, 84, 89, 77, 85, 71, 90, 79, 74, 81]
```

bp_demo.py – 4/3.

Mivel a tengelyek és a tengelyfeliratok nem változnak, ezeket háttérképként töltjük be



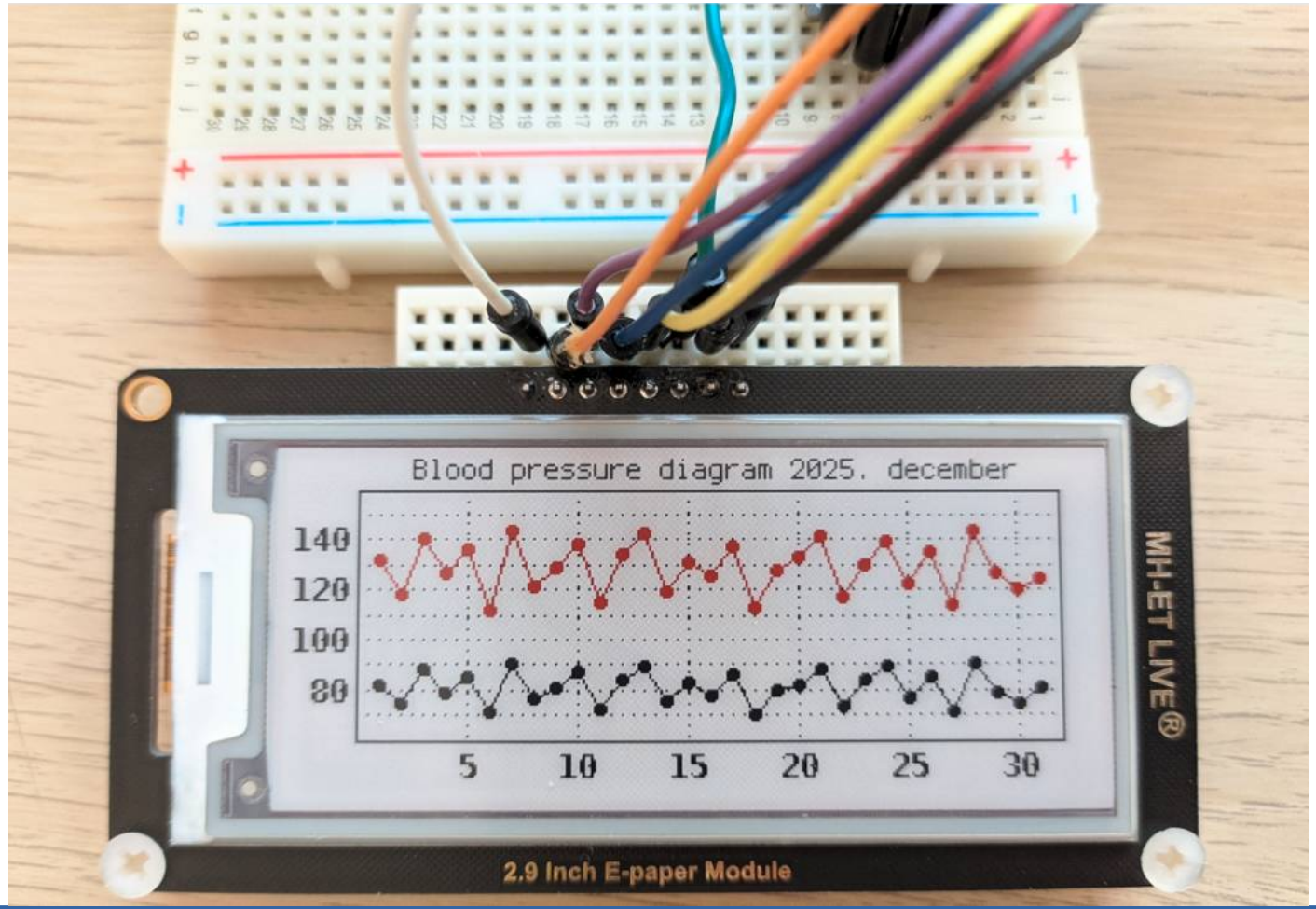

```
# --- FŐ GROUP ---
main = displayio.Group()

# --- HÁTTÉRKÉP BETÖLTÉSE ---
bg_bitmap = displayio.OnDiskBitmap("/bp_background.bmp")
bg = displayio.TileGrid(bg_bitmap, pixel_shader=bg_bitmap.pixel_shader)
main.append(bg)

# --- FŐCÍM KIÍRÁSA ---
title = label.Label(
    terminalio.FONT,
    text="Blood pressure diagram 2025. december",
    color=0x000000,
    x=ML + 20,          # bal margóhoz igazítva
    y=MT - 8            # a keret fölé húzva
)
main.append(title)
```

```
------------------------------------------------
A displayio hierarchiája a programban:
------------------------------------------------
display.root_group = main
    ├── TileGrid (háttér bitmap)
    ├── Label (címsor)
    └── Group (graph)
        ├── Line objektumok (SYS és DIA görbék)
        └── Circle objektumok (adatpontok)
```

bp_demo.py – 4/4.

```
# --- GÖRBÉK RAJZOLÁSA ---
graph = displayio.Group()

# --- SYS (piros) ---
for i in range(30):
    graph.append(Line(x_to_px(i+1), y_to_px(sys_values[i]),x_to_px(i+2), y_to_px(sys_values[i+1]),
        color=0xFF0000))
    graph.append(Circle(x_to_px(i+1), y_to_px(sys_values[i]), 2, fill=0xFF0000))
graph.append(Circle(x_to_px(31), y_to_px(sys_values[30]), 2, fill=0xFF0000))

# --- DIA (fekete) ---
for i in range(30):
    graph.append(Line(x_to_px(i+1), y_to_px(dia_values[i]),x_to_px(i+2), y_to_px(dia_values[i+1]),
        color=0x000000))
    graph.append(Circle(x_to_px(i+1), y_to_px(dia_values[i]), 2, fill=0x000000))
graph.append(Circle(x_to_px(31), y_to_px(dia_values[30]), 2, fill=0x000000))
main.append(graph)

# --- KIJELZÉS ---
display.root_group = main
display.refresh()
```

bp_demo.py – az eredmény



Az ESP32 C3 Super Mini kártya kivezetései