Az RP2040 mikrovezérlő programozása Arduino IDE környezetben

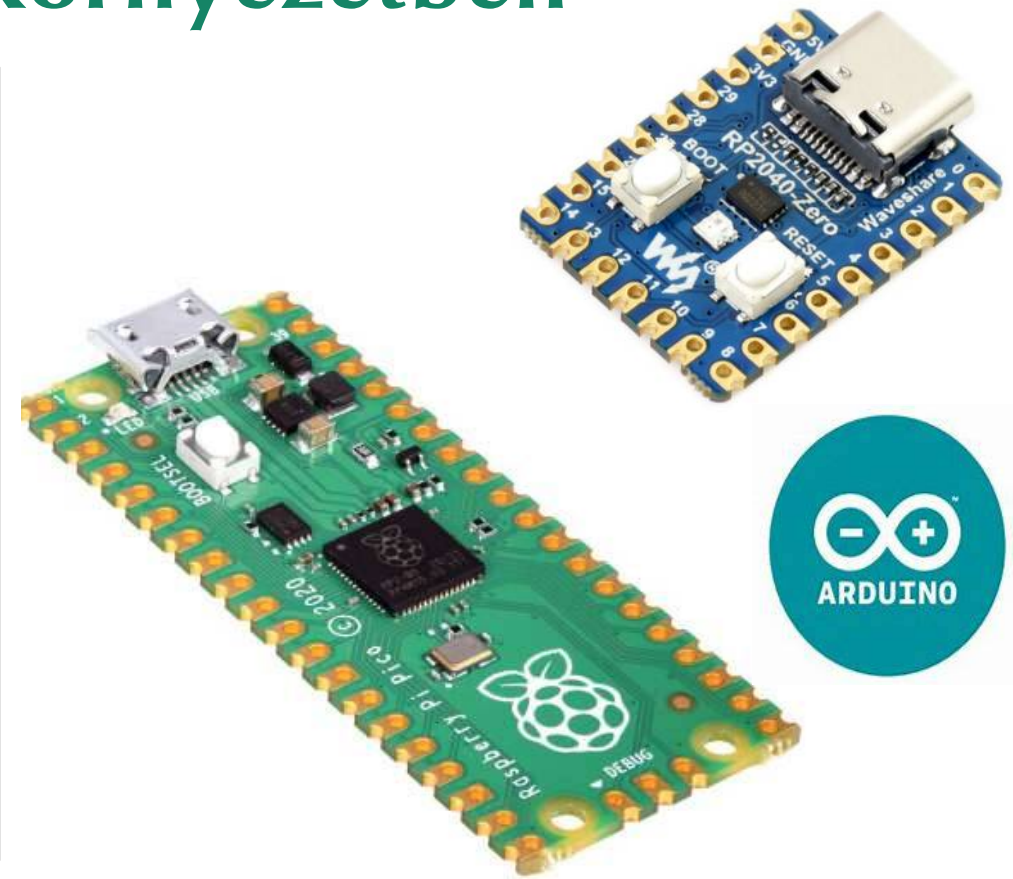
Bink2 | Arduino 1.8.19 (Windows Store 1.8.57.0)

File Edit Sketch Tools Help

Bink2 §

```
#include <mbed.h>

mbed::DigitalOut led(LED1);

void setup() {
}

void loop() {
  led.write(1);                        // Switch the LED on
  rtos::ThisThread::sleep_for(1000); // Wait for a second
  led.write(0);                        // Switch the LED off
  rtos::ThisThread::sleep_for(1000); // Wait for a second
}
```

15 Raspberry Pi Pico on COM14



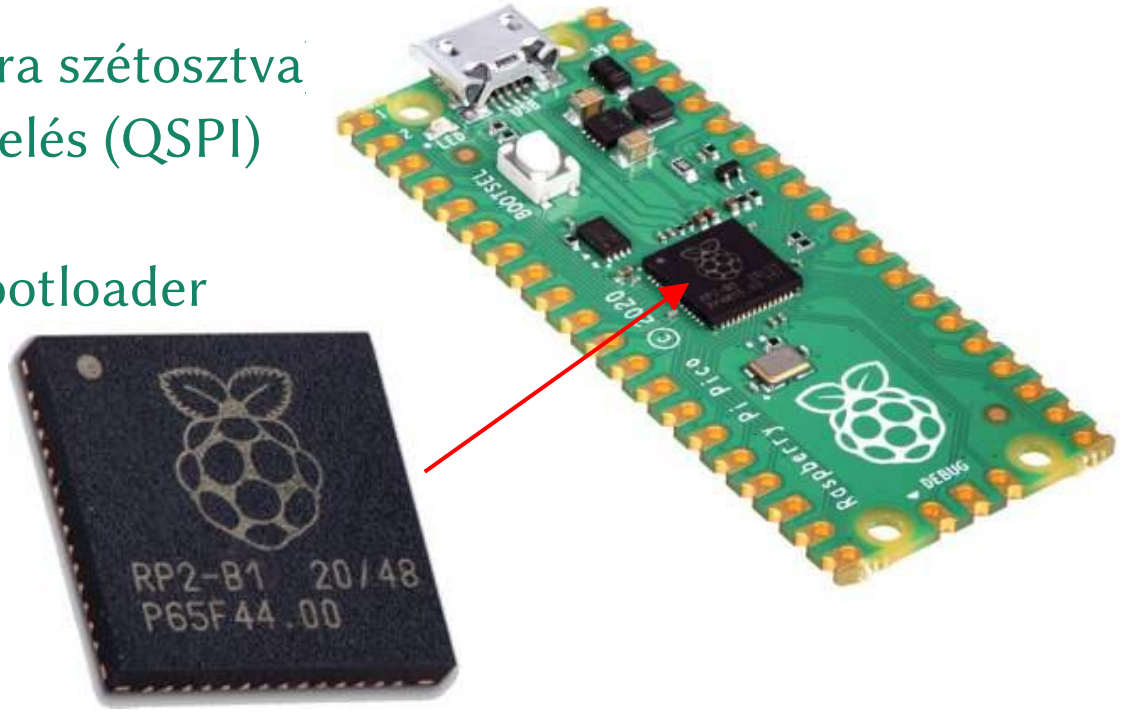
Felhasznált és ajánlott irodalom

- Raspberry Pi: Pico-series Microcontrollers
- Raspberry Pi: RP2040 adatlap (PDF)
- Raspberry Pi: Raspberry Pi Pico Datasheet
- Raspberry Pi: Getting started with Raspberry Pi Pico-series Microcontrollers
- Waveshare: RP2040-Zero, a Pico-like MCU Board

- Ralphjy: Program RPi Pico using Mbed library with Arduino IDE
- Alan Yorinks: NeoPixelConnect

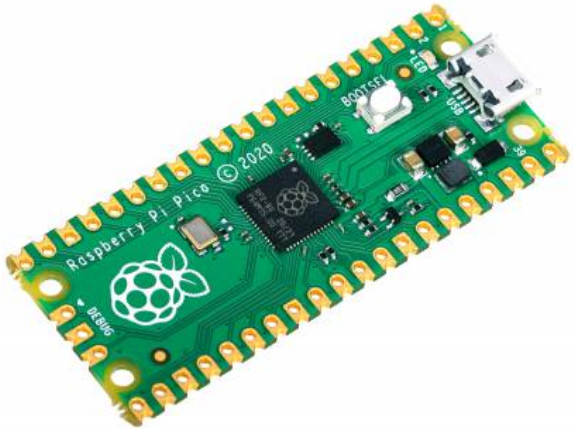
Az RP2040 bemutatása

- A **Raspberry Pi** Alapítvány fejlesztette ki az **RP2040** MCU-t, főbb jellemzői:
  - kétmagos Arm Cortex-M0+ processzor
  - 133 MHz órajel
  - 264 KB RAM (6 független bankra szétosztva
  - Akár 16 MB flash memória kezelés (QSPI)
  - DMA támogatás
  - USB tömegtároló módú UF2 bootloader
  - Perifériák:
    - 2 UART
    - 2 SPI csatorna
    - 2 I2C csatorna
    - 16 PWM csatorna
    - USB vezérlő (host & device)
    - 8 PIO állapotgép

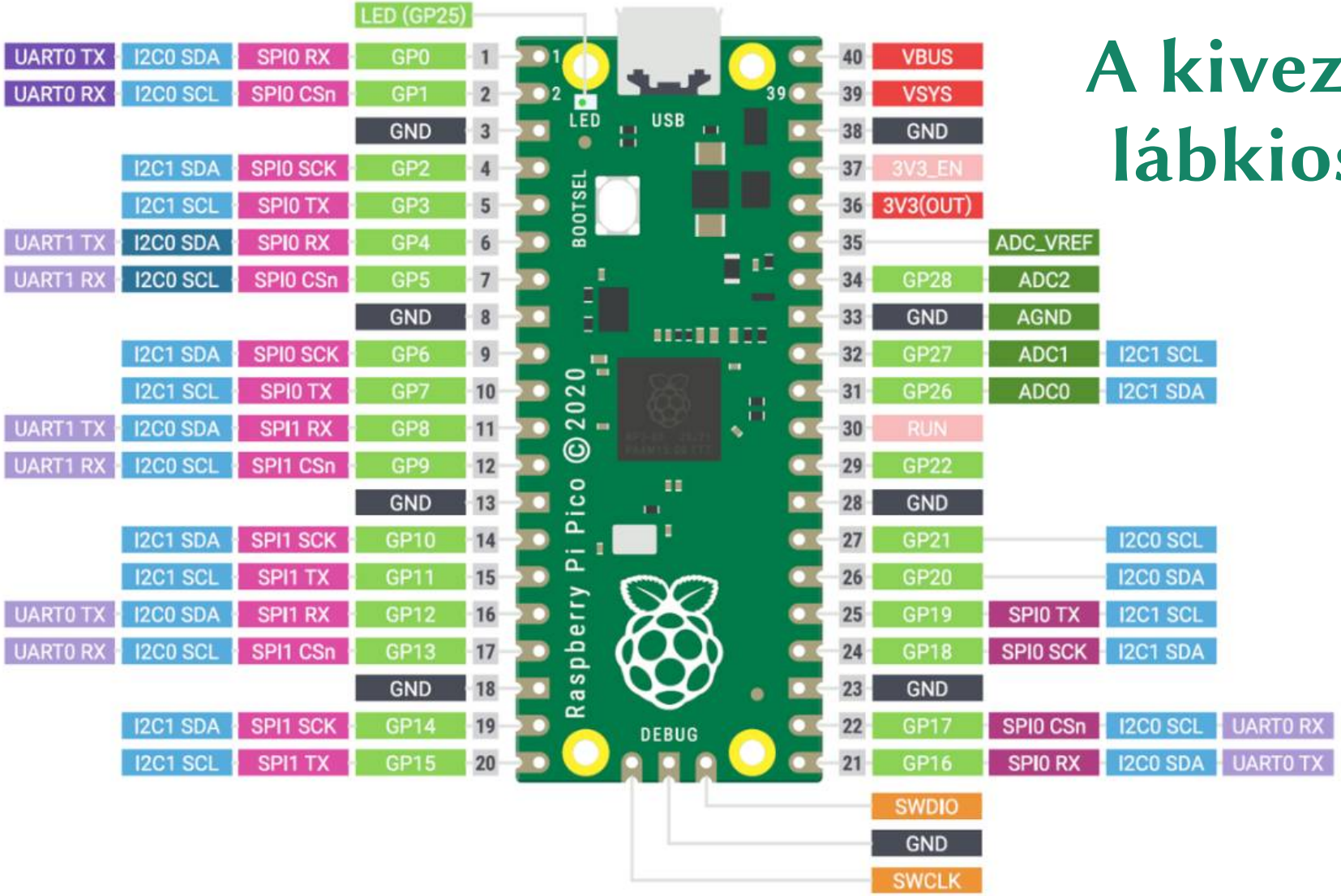


A Raspberry Pi Pico kártya

- Az Arduino nanohoz hasonló méretű kártya, **RP2040** CPU-val és 2 MB flash memóriával szerelve
- 26 db GPIO kivezetés érhető el, ezek közül három lehet analóg bemenet
- A nem kivezetett **GP25 láb** egy LED-et vezérel
- A kártyának van egy továbbfejlesztett kiadása is (Raspberry Pi Pico W) ami egy WiFi interfészt is tartalmaz

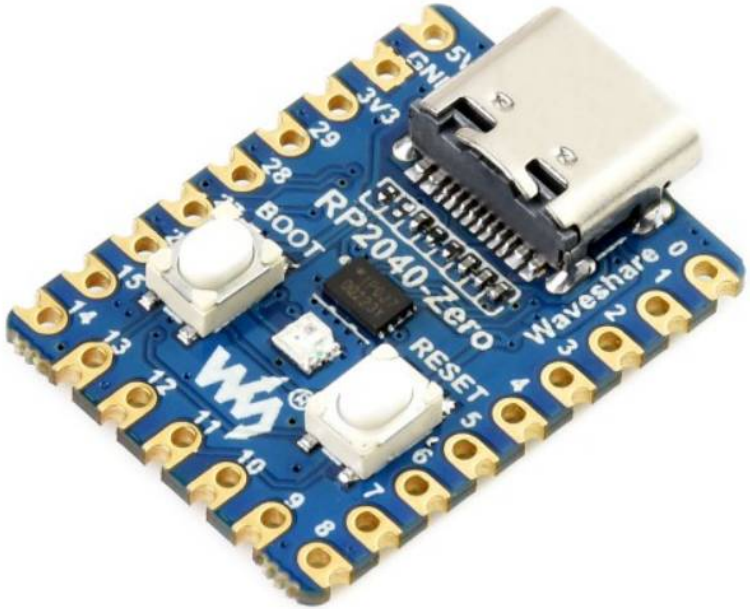


A kivezetések lábkiosztása

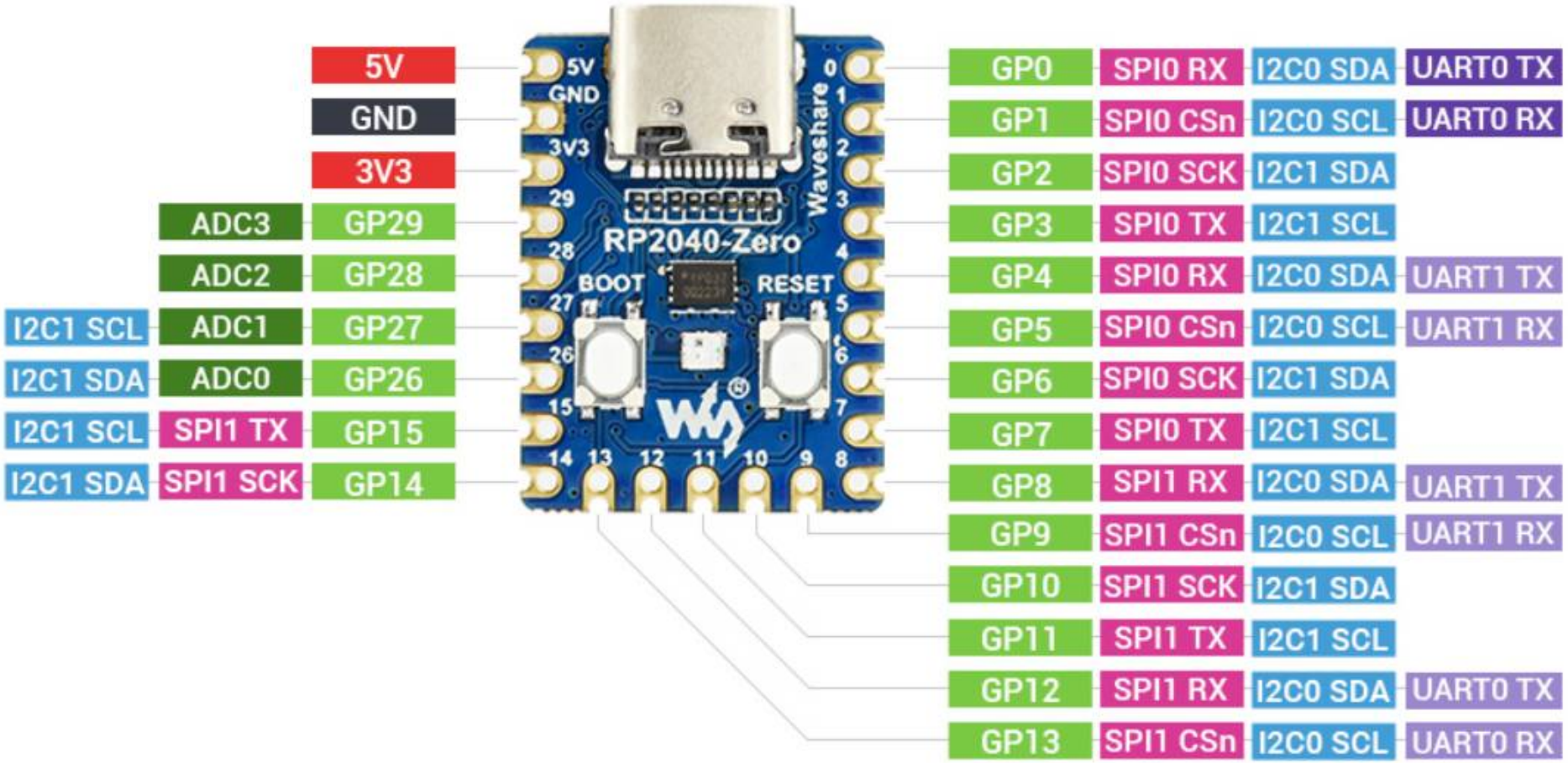


A Waveshare RP2040-Zero kártya

- A Waveshare cég **RP2040-Zero** kártyája a **Raspberry Pi Pico** kártyához hasonló, de kisebb alapterületű kártya
- Fele olyan hosszúságú, emiatt egyes kivezetések nehezebben érhetők el
- USB-C csatlakozóval szerelt
- A **BOOTSEL** gomb mellett van **RESET** gomb is
- Beépített LED helyett egy **WS2812** RGB LED-et tartalmaz, ami a **GP16** kivezetésre van kötve

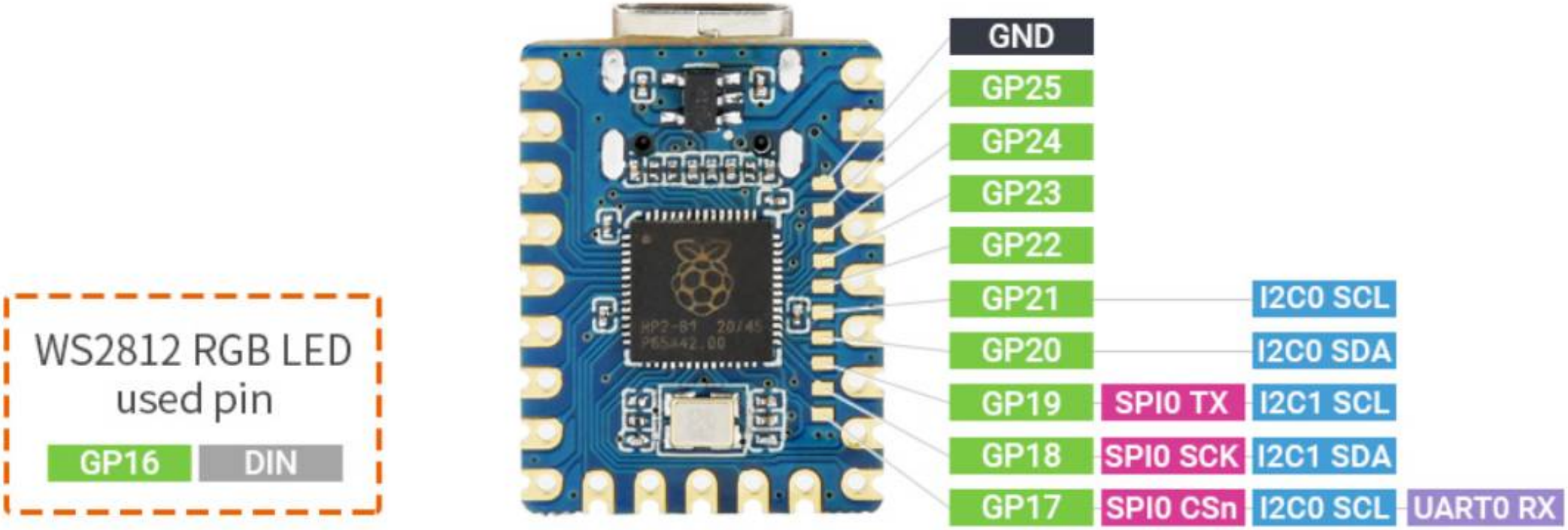


A kivezetések lábkiosztása



További kivezetések

- Néhány kivezetés a kártya hátoldalán van kivezetve



Fejlesztői támogatás

Pico C/C++ SDK

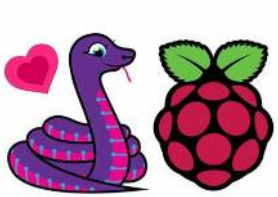
The Raspberry Pi official C SDK can be used from the command line, or from popular integrated development environments like Visual Studio Code and Eclipse.

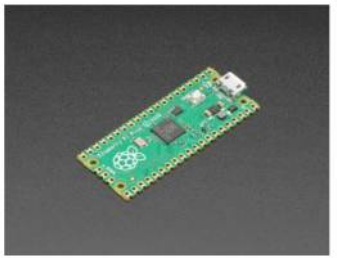
MicroPython

MicroPython is a full implementation of the Python 3 programming language that runs directly on embedded hardware like Raspberry Pi Pico.

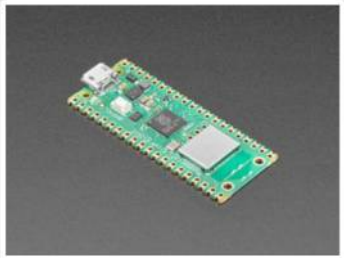
További lehetőségek

- CircuitPython



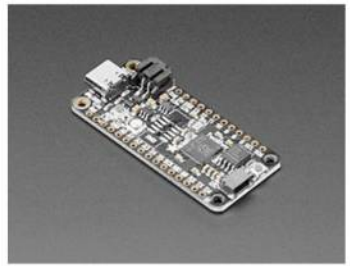


Pico
By Raspberry Pi



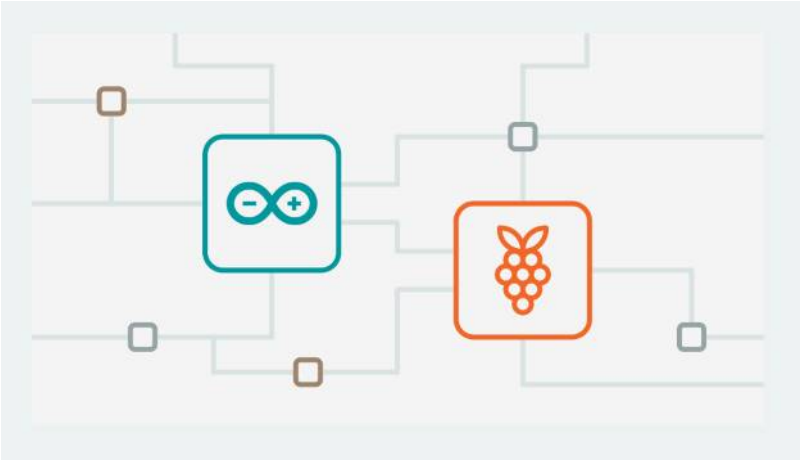
Pico W
By Raspberry Pi

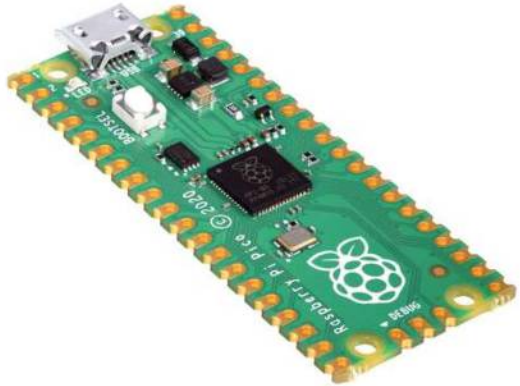
RP2040-Zero
By Waveshare



Feather RP2040
By Adafruit

- Arduino Mbed Core for RP2040 boards





Raspberry Pi Pico

(részleges támogatás)

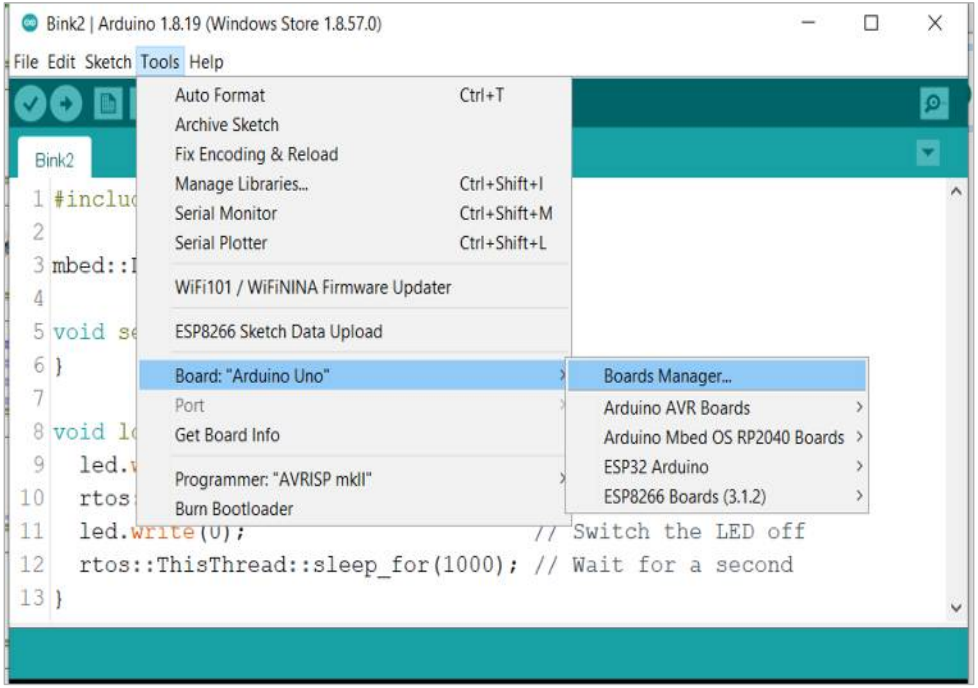
RP2040-Zero

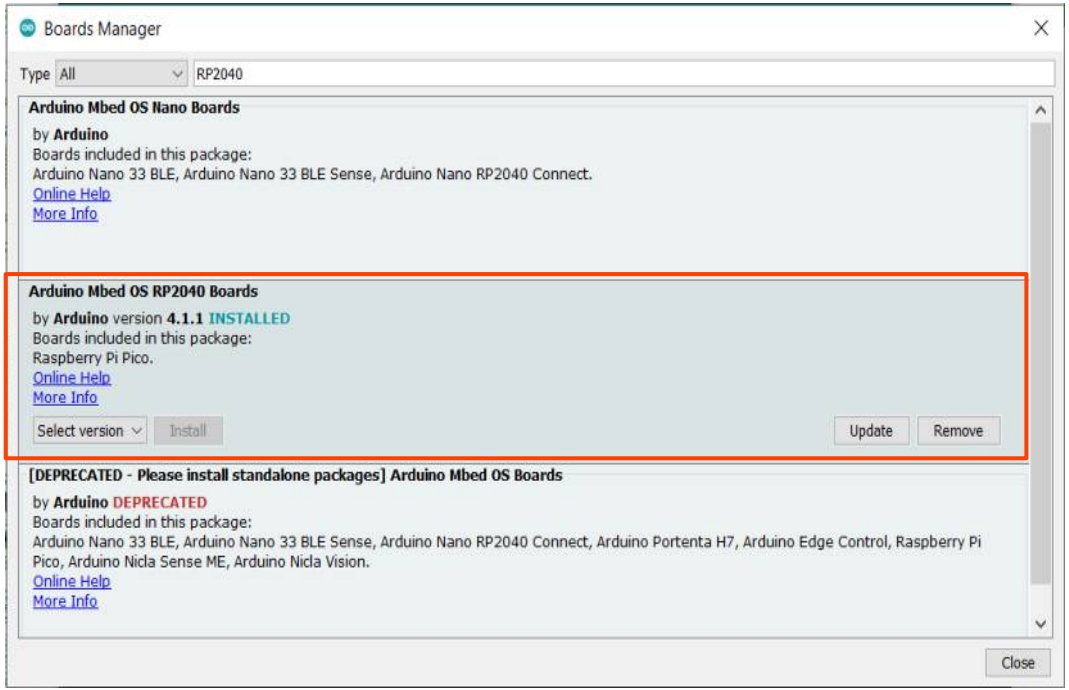
Az Arduino RP2040 támogatás története

- 2021. január: A **Raspberry Pi Foundation** bemutatta a **Raspberry Pi Pico** kártyát, amely a saját fejlesztésű **RP2040** mikrokontrollert használja
- 2021. március: **Earle F. Philhover** közzétett **arduino-pico bővítőcsomagja** lehetővé tette az **RP2040** MCU-val ellátott kártyák Arduino IDE környezetben történő programozását
- 2021. április: Az **Arduino Team** bejelentette a **Raspberry Pi Pico** kártya támogatását az Arduino IDE-ben az **Arduino Mbed Core for RP2040 boards** bővítménnyel, amely az **Mbed OS** alapjaira épül. Így az Arduino programokban az Mbed OS API-ja is elérhető, sőt a két API bizonyos mértékben keverhető is
- A későbbiekben a **Raspberry Pi Pico** és az **RP2040** alapú Arduino nano kártyák támogató csomagját különválasztották, ezért telepítésnél ügyeljünk arra, hogy az **Arduino Mbed OS RP2040 boards** nevű bővítményt használjuk
- Mi a ver 4.1.1 kiadással próbáltuk ki az előadásban bemutatott programokat

Az Arduino Mbed OS RP2040 boards bővítmény telepítése

- Az Arduino IDE **Tools** menüjében válasszuk a **Boad/Boards Manager** menüpontot, majd a felbukkanó ablakban keressük meg az **Arduino Mbed OS RP2040 boards** nevű bővítmény, s az **INSTALL** gomjára kattintva telepítsük azt!





Két dudás egy csárdában

- Az **Raspberry Pi Pico** kártya beépített LED-jének villogtatása Arduino API hívásokkal
- A **LED_BUILTIN** esetünkben a **GP25**-re kötött beépített LED-et jelenti

Blink.ino

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);// turn the LED on
  delay(1000);                    // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off
  delay(1000);                    // wait for a second
}
```

- Az **Rpi Pico** kártya beépített LED-jének (p25) villogtatása Mbed OS API hívásokkal
- Az **Mbed OS** függvények, objektumok más névtérben vannak

Blink2.ino

```
#include <mbed.h>

mbed::DigitalOut led(LED1);

void setup() { }

void loop() {
  led.write(1);          // Switch the LED on
  // Wait for a second
  rtos::ThisThread::sleep_for(1000);
  led.write(0);          // Switch the LED off
  // Wait for a second
  rtos::ThisThread::sleep_for(1000);
}
```

Mire valók a névterek (namespace) a c++ nyelvben?

- **A névterek** (namespace) a C++ nyelvben arra szolgálnak, hogy elkerüljék a névütközéseket a program különböző részei között
- **Névütközések elkerülése:** Ha két különböző könyvtárban vagy modulszerkezetben ugyanazokat a nevet használják (pl. open() függvény), akkor ezek névütközéshez vezethetnek
- **Kód szervezése:** A névterek segítségével a kódot logikai egységekre oszthatjuk
- **Egyszerűsített kódírás:** A **using namespace** kulcsszóval jelezzük, hogy az adott névtér összes elemét használni akarjuk, így nem kell minden elem előtt a névteret megadnunk. Például `using namespace std;` megengedi, hogy a `std::cout` helyett `cout`-ot írjunk.
- **Hierarchikus szerveződés:** a C++ névterek arra is lehetőséget biztosítanak, hogy több szintet használjunk, például:

```
namespace outer {
    namespace inner { ... } }.
```

A using namespace használata a gyakorlatban

- A **using namespace <név>;** direktíva használatával egyszerűbben és áttekinthetőbben írhatjuk a programonkat, de a névütközések elkerülése ekkor a programozó felelőssége
- A második programban a **DigitalOut** objektum értékadásánál az **egyszerűsített írásmódot** is bemutatjuk a **write()** és **read()** tagfüggvények használata helyett: **`led = !led;`** valójában ezt jelenti: **`led.write(!led.read());`**

Blink2.ino

```
#include <mbed.h>

mbed::DigitalOut led(LED1);

void setup() { }

void loop() {
  led.write(1);             // Switch the LED on
  // Wait for a second
  rtos::ThisThread::sleep_for(1000);
  led.write(0);             // Switch the LED off
  // Wait for a second
  rtos::ThisThread::sleep_for(1000);
}
```

Blink3.ino

```
#include <mbed.h>

using namespace mbed;                    // for Digitalout
using namespace rtos;                    // for ThisThread
using namespace std::chrono_literals;// for time units

DigitalOut led(LED1);

void setup() { }

void loop() {
  led = !led;                            // Negate LED state
  ThisThread::sleep_for(250ms);          // Wait for 250ms
}
```

DigitalOut

- A digitális be- és kimenetek kezelésére az mbed API a **DigitalIn, DigitalOut** és **DigitalInOut** komponenseket nyújtja. A **DigitalOut** C++ objektumosztály általános célú *digitális kimenetek* konfigurálására és kezelésére szolgál
- A konstruktor hívásakor opcionálisan a kimenet kezdeti állapotát (0 vagy 1) is megadhatjuk: `DigitalOut név(pin,állapot);`
- A `write()` és `read()` tagfüggvények segítségével írhatunk a kimenetre vagy visszaolvashatjuk az állapotát, de értékadásnál használhatjuk a rövidített alakot is

Függvény	Használat
DigitalOut név(pin)	Létrehoz egy "név" nevű **DigitalOut** objektumot és a pin paraméterrel megadott kimenethez rendeli
write(data)	A kimenet beállítása 0 vagy 1 állapotba
read()	Az int típusú visszatérési érték a kimenet állapotát adja meg (0 vagy 1)
operator =	Rövidített alak **write(data)** helyett `pl. led = 1;`
operator int()	Rövidített alak **read()** helyett `pl. ledstate = led;`

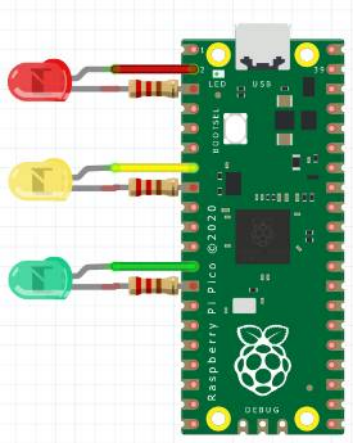
A Scheduler könyvtár

- Az Arduino **Scheduler könyvtár** célja több feladat egyidejű futtatása megszakítás nélkül. Kooperatív többfeladatúságot valósít meg
- Támogatott platformok: Elsősorban a SAM és SAMD architektúrákhoz készült (Arduino Due, Zero), de néhány más platformmal (pl. **RP2040)** is működik
- Fő funkciók:
  - **Scheduler.startLoop()**: Új feladat elindítása
  - **yield()**: Vezérlés átadása más feladatoknak
  - **delay()**: Vezérlés átadása adott ideig
- Az alábbi példában **loop()** egy adott időre szakítja meg a futását és adja át a vezérlést, a **loop1()** taszk viszont csak akkor adja át a vezérlést, ha van másik, futásra képes feladat

```
#include <Scheduler.h>

void setup() {
  Scheduler.startLoop(loop1);
  // loop() automatikusan indul!
}

void loop() {
  // Fő feladat
  delay(100);     // Vezérlés átadása
}

void loop1() {
  // Másik feladat
  yield();        // Vezérlés átadása
}
```

scheduler_ledblink.ino

- Három LED-et kapcsolgatunk, három konkurens **loop()** függvényben



scheduler_ledblink.ino

```
#include <Scheduler.h>

int led1 = 1;              // Red LED
int led2 = 5;              // Yellow LED
int led3 = 9;              // Green LED

void setup() {
  Serial.begin(115200);

  // Setup the 3 pins as OUTPUT
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);

  // Add "loop2" and "loop3" to scheduling.
  // "loop" is always started by default.
  Scheduler.startLoop(loop2);
  Scheduler.startLoop(loop3);
}
```

```
// accept commands from Serial
// '0' turns off LED3
// '1' turns on LED3

void loop3() {
  if (Serial.available()) {
    char c = Serial.read();
    if (c == '0') {
      digitalWrite(led3, LOW);
      Serial.println("OFF");
    }
    if (c == '1') {
      digitalWrite(led3, HIGH);
      Serial.println("ON");
    }
  }

  // IMPORTANT:
  // We must call 'yield' at a
  // regular basis to pass
  // control to other tasks.
  yield();
}
```

```
// blink LED1 with 1s delay
void loop() {
  digitalWrite(led1, HIGH);
  delay(1000);
  digitalWrite(led1, LOW);
  delay(1000);
}

// blink LED2 with 0.1s delay
void loop2() {
  digitalWrite(led2, HIGH);
  delay(100);
  digitalWrite(led2, LOW);
  delay(100);
}
```

threads.ino

- A programszálak kezelése az **Mbed OS API**-ban már ismerős lehet a korábbi évek **Mbed** témájú előadásaiból
- A programban két program-szálat indítunk (egy-egy végtelen ciklus, amelyben egy rövid üzenetet kiírunk). A **sleep_for()** várakozás engedi a többi programszál futását
- A **loop()** függvény egy harmadik programszál lehetne, de itt most nem írtunk elő semmilyen tevékenységet a számára

threads.ino

```
#include <mbed.h>
using namespace mbed;
using namespace rtos;
using namespace std::chrono_literals;
Thread t1;
Thread t2;

void func1() {
  for (;;) {                    // Repeat forever
    Serial.println("A");        // Print A to the Serial port
    ThisThread::sleep_for(2s); // Wait for 2 seconds
  }
}

void func2() {
  for (;;) {                    // Repeat forever
    Serial.println("B");        // Print B to the Serial port
    ThisThread::sleep_for(3s); // Wait for 3 seconds
  }
}

void setup() {
  Serial.begin(115200);         // Start Serial communication
  t1.start(func1);              // Pass func1 to thread1
  t2.start(func2);              // Pass func2 to thread2
}

void loop() {                   // loop() is executed in its own thread
   // now left empty
}
```

rtos_ledfade.ino 2/1.

- Az **Element14** közösségi oldalán bemutatott program minimális módosítással a 18 oldalon a **scheduled_ledblink.ino** programnál mutatott kapcsolással is futtatható
- Két programszálat használunk: **led_red_thread** a GP1-re kötött piros LED-et, **led_green_thread** pedig a GP9-re kötött zöld LED-et kezeli az Arduino API által biztosított **analogWrite()** függvény segítségével (PWM)

rtos_ledfade.ino

```
#include "rtos.h"
#define RED 1
#define GREEN 9
#define MS(x) chrono::milliseconds(x)

using namespace std; // we will be using std::chrono
using namespace rtos; // we will be using rtos::ThisThread
Thread led_red_thread;
Thread led_green_thread;

void led_red_function() {
  for (;;) {
    // delay while red fades out and green fades in
    ThisThread::sleep_for(MS(512));
    // Fade red out
    for(int i = 255; i >= 0; i--) {
      analogWrite(RED, 255 - i);
      ThisThread::sleep_for(MS(1));
    }
    // delay while green fades out and red fades in
    ThisThread::sleep_for(MS(512));
    // Fade red back in
    for(int i = 05; i < 255; i++) {
      analogWrite(RED, 255 - i);
      ThisThread::sleep_for(MS(1));
    }
  }
}
```

Folytatás a következő oldalon...

rtos_ledfade.ino 2/2.

- Ha két programszálat indítunk, valójában három programszál fog futni, mert a **loop()** függvény is automatikusan elindul
- Mi most a **loop()** függvényt üresen hagytuk, nincs benne előírt tevékenység
- Vegyük észre, hogy a programban nem kellett az **mbed.h,** csak az **rtos.h** fejléc állomány!

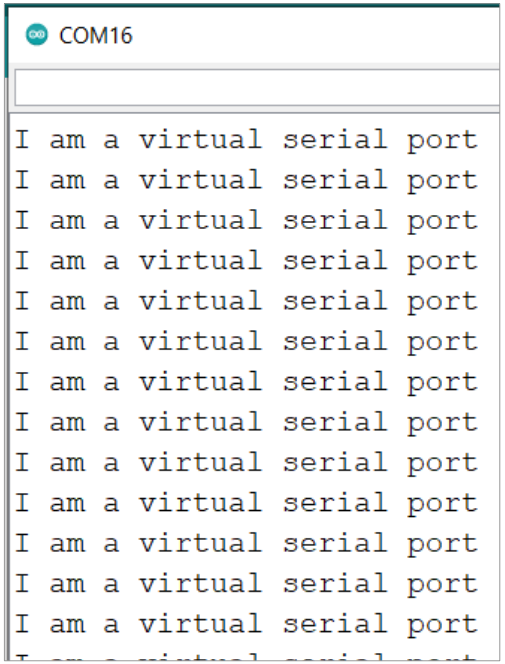
```
// function to be attached to led_green_thread
void led_green_function() {
  for (;;) {
    // delay while red fades out
    ThisThread::sleep_for(MS(256));
    // fade green in
    for(int i = 0; i < 255; i++) {
      analogWrite(GREEN, 255 - i);
      ThisThread::sleep_for(MS(1));
    }
    // delay while green fades out and red fades in
    ThisThread::sleep_for(MS(512));
    // fade green out
    for(int i = 255; i>= 0; i--) {
      analogWrite(GREEN, 255 - i);
      ThisThread::sleep_for(MS(1));
    }
    // delay while red fades in
    ThisThread::sleep_for(MS(256));
  }
}

void setup() {
  pinMode(RED, OUTPUT);
  pinMode(GREEN, OUTPUT);
  led_red_thread.start(led_red_function); // start red thread
  led_green_thread.start(led_green_function); // start green thread
}

void loop() {       // loop() is executed in its own thread
                    // now left empty
}
```

Az USBSerial használata

- Az **MbedOS USBSerial** könyvtára segítségével *„klasszikus mbed”* programot is írhatunk, ami az USB-n keresztül virtuális soros portként kommunikál, de ne feledjük, hogy ez nem az a soros port, amit az **Arduino** biztosít, emiatt az automatikus programletöltés sem fog működni e program rátöltése után

USBSerial.ino

```
#include "mbed.h"
#include "USBSerial.h"
using namespace mbed;
using namespace rtos;

USBSerial serial; // Virtual serial port over USB

int main(void) {
    while (1) {
        serial.printf("I am a virtual serial port\r\n");
        ThisThread::sleep_for(1000);
    }
}
```



A NeoPixelConnect könyvtár használata

- A **Waveshare RP2040-Zero** kártya beépített LED helyett egy **WS2812** RGB LED-et tartalmaz, ennek kezeléséhez most a : NeoPixelConnect könyvtárat fogjuk használni, ami egyébként felfűzött LED-sort is tudna vezérelni
- **NeoPixelConnect***(byte pinNumber, uint16_t numberOfPixels)* – konstruktor
- **neoPixelInit***(byte pinNumber, uint16_t numberOfPixels)* – inicializálás
- **neoPixelSetValue***(uint16_t pixel_number, uint8_t r=0, uint8_t g=0, uint8_t b=0, bool autoShow=false)* – egy pixel átírása
- **neoPixelClear***(bool autoShow=true)* – az összes pixel kioltása
- **neoPixelFill***(uint8_t r=0, uint8_t g=0, uint8_t b=0, bool autoShow=true)* – az összes pixel átírása a megadott színre
- **putPixel***(uint32_t pixel_grb)* – a 32 bites paraméter megadja a módosítandó pixel sorszámát és GRB színkódját
- **neoPixelShow***(void)* – megjeleníti a korábbi beírások eredményét

fill_demo.ino

- A WS2812 LED a GP16-os kimenetre van kötve, ezért esetünkben a pinNumber 16 lesz
- Mivel csak egy LED-ünk van, a `MAXIMUM_NUM_NEOPIXELS` értéke = 1
- Az *autoShow* paraméter true értéke azt jelzi, hogy a beírást követően automatikusan jelenjen is meg a beállított szín és fényerő, *false* értéke esetén viszont használnunk kellene a **neoPixelShow()** függvényt

```
#define MAXIMUM_NUM_NEOPIXELS 1
#include <NeoPixelConnect.h>
NeoPixelConnect p(16, MAXIMUM_NUM_NEOPIXELS, pio0, 0);

void setup() {
  Serial.begin(115200);
  delay(2000);
  Serial.println("In setup");
  delay(1000);
}

void loop() {
  p.neoPixelFill(255, 0, 0, true); //set all LEDs to red
  Serial.print("R");
  delay(1000);
  p.neoPixelClear(true);
  delay(1000);
  p.neoPixelFill(0, 255, 0, true); //set all LEDs to green
  Serial.print("G");
  delay(1000);
  p.neoPixelClear(true);
  delay(1000);
  p.neoPixelFill(0, 0, 255, true); //set all LEDs to blue
  Serial.println("B");
  delay(1000);
  p.neoPixelClear(true);
  delay(1000);
}
```

multicore blinky.ino

- Az RP2040 C/C++ SDK lehetőséget biztosít a kétmagos futtatásra. Az alábbi program bemutatja, hogyan használhatjuk ezt fel Arduino programokban
- Az alapértelmezett **core0** mellett elindítjuk a második magot (**core1**), hogy párhuzamosan vezéreljünk két LED-et
- Ügyelnünk kell azonban arra, hogy a második magon nem futhat akármi, így például nem használható a **delay()** függvény, vagy a kiíratás a **Serial.print()** segítségével

```
#include <Arduino.h>
#include "pico/multicore.h"
#define CORE0_LED 1u
#define CORE1_LED 5u
int brightness = 0;
int fadevalue = 10;
void core1_loop() {
  While (1) {
    brightness = brightness + fadevalue;
    if (brightness <= 0 || brightness >= 2048) {
        fadevalue = -fadevalue;     }
    digitalWrite(CORE1_LED, HIGH);
    sleep_us(brightness);
    digitalWrite(CORE1_LED, LOW);
    sleep_us(2000);
  }
}
void setup() {
  pinMode(CORE1_LED, OUTPUT);
  pinMode(CORE0_LED, OUTPUT);
  delay(1000);  // Enélkül Core1 leállhat!
  multicore_launch_core1(core1_loop);
}
void loop() {
  digitalWrite(CORE0_LED, HIGH);   delay(50);
  digitalWrite(CORE0_LED, LOW);   delay(2500);
}
```

Fórumokon többen is jelezték hogy core1 indítása előtt kell egy kis késleltetés, különben core1 „lefagy”