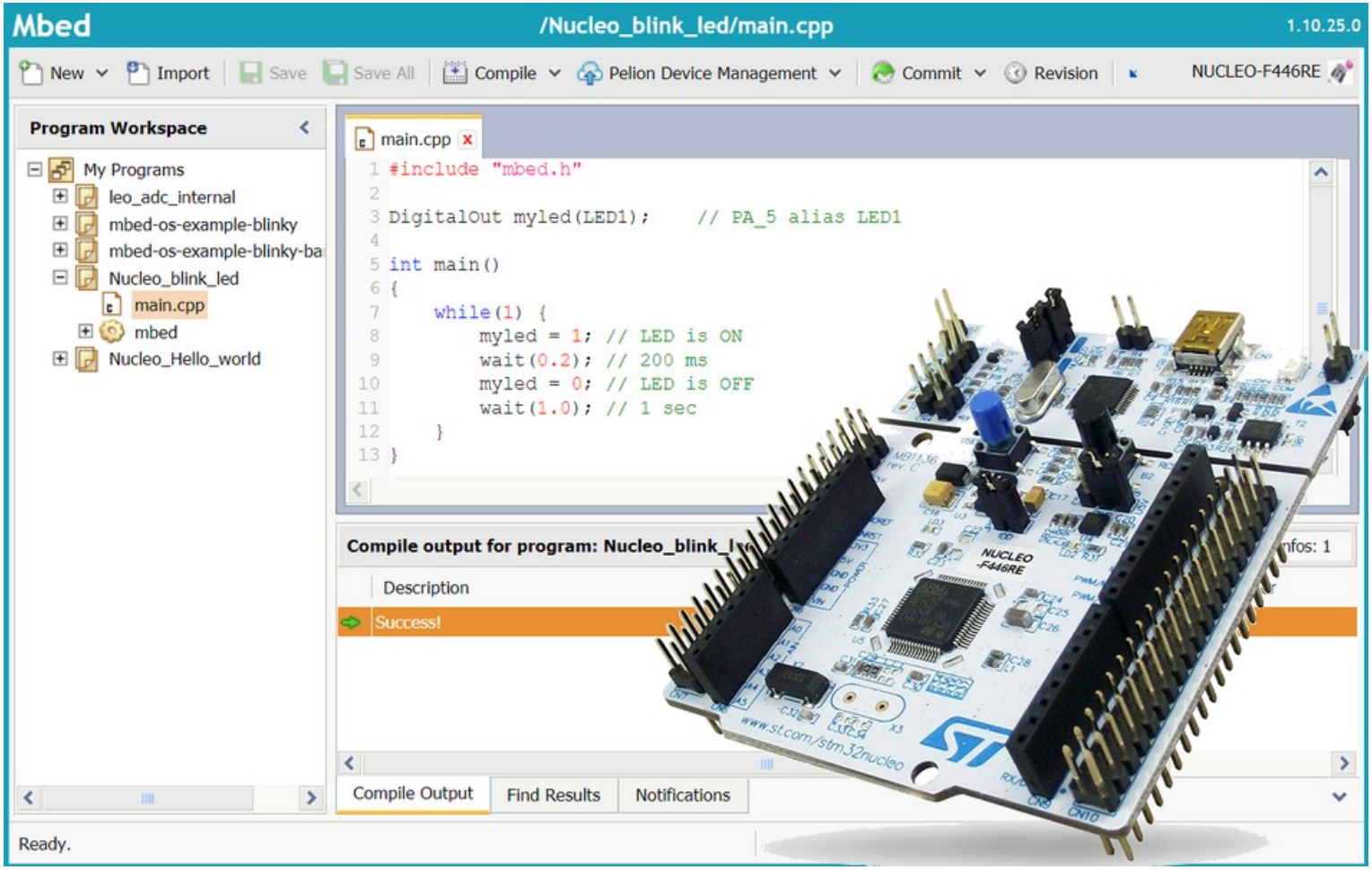
STM32 mikrovezérlők programozása ARM mbed környezetben



1. ARM mbed alapok, I/O műveletek

A tanfolyam célja

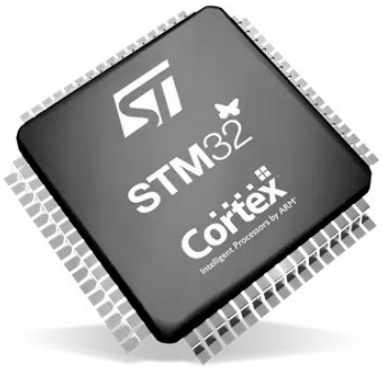
- Megismertetjük a hallgatókkal az C/C++ alapú **ARM mbed** környezetet, amellyel a beágyazott rendszerek fejlesztése hatékonyan végezhető
- Az **STM32F446RE** mikrovezérlő és a **NUCLEO-F446RE** kártya segítségével a gyakorlatban is kipróbálhatjuk a bemutatott mintaprogramokat, illetve saját alkalmazásokat is fejleszthetünk rá
- A **NUCLEO-F446RE** kártyára épített ST-Link V2-1 eszközt az STM32 mikrovezérlő programozására, megfelelő fejlesztő környezetben (pl. **Mbed Studio,** vagy **STM32CubeIDE**) hardveres nyomkövetésre, illetve USB-soros átalakítóként a PC-vel történő kommunikációra használjuk
- A **NUCLEO-F446RE** kártyához csatlakoztatott kiegészítők (szenzorok, kijelzők, aktuátorok) segítségével *fizikai számítástechnikai* feladatokat végezhetünk, azaz olyan interaktív rendszereket hozhatunk létre hardver és szoftver segítségével, melyek képesek érzékelni a világban létrejövő jeleket és reagálni is tudnak rá

Felhasznált és ajánlott irodalom

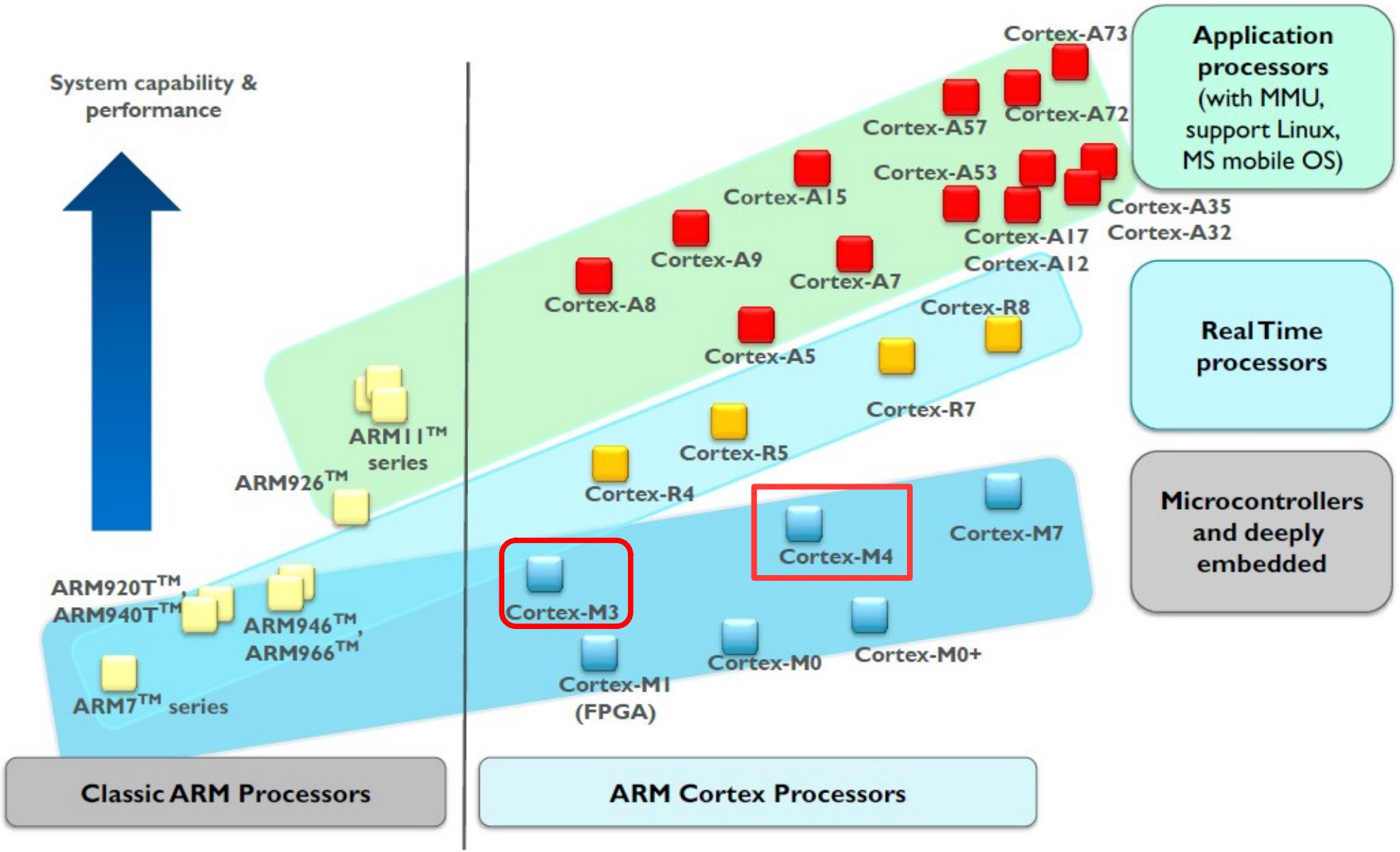
- Joseph Yiu: The Definitive Guide To The ARM Cortex-M3 and Cortex-M4
- Rob Toulson and Tim Wilmhurst:
  Fast and Effective Embedded Systems Design: Applying the ARM mbed
- Perry Xiao: Designing Embedded Systems and the Internet of Things (IoT)
  with the ARM mbed
- Cserny István: A FRDM-KL25Z kártya programozása mbed környezetben
- **ARM mbed honlap: https://os.mbed.com/**
- ARM mbed Compiler: https://ide.mbed.com/compiler/
- ARM mbed 2 Handbook (elavult): https://os.mbed.com/handbook/Homepage
- ARM mbed 2 Cookbook (elavult): https://os.mbed.com/cookbook/Homepage
- ARM mbed forráskód: https://github.com/ARMmbed/mbed-os

Adatlapok:

- **STM32F446RE adatlap és termékinfo**
- **STM32F446** Family Reference Manual



ARM processzorok

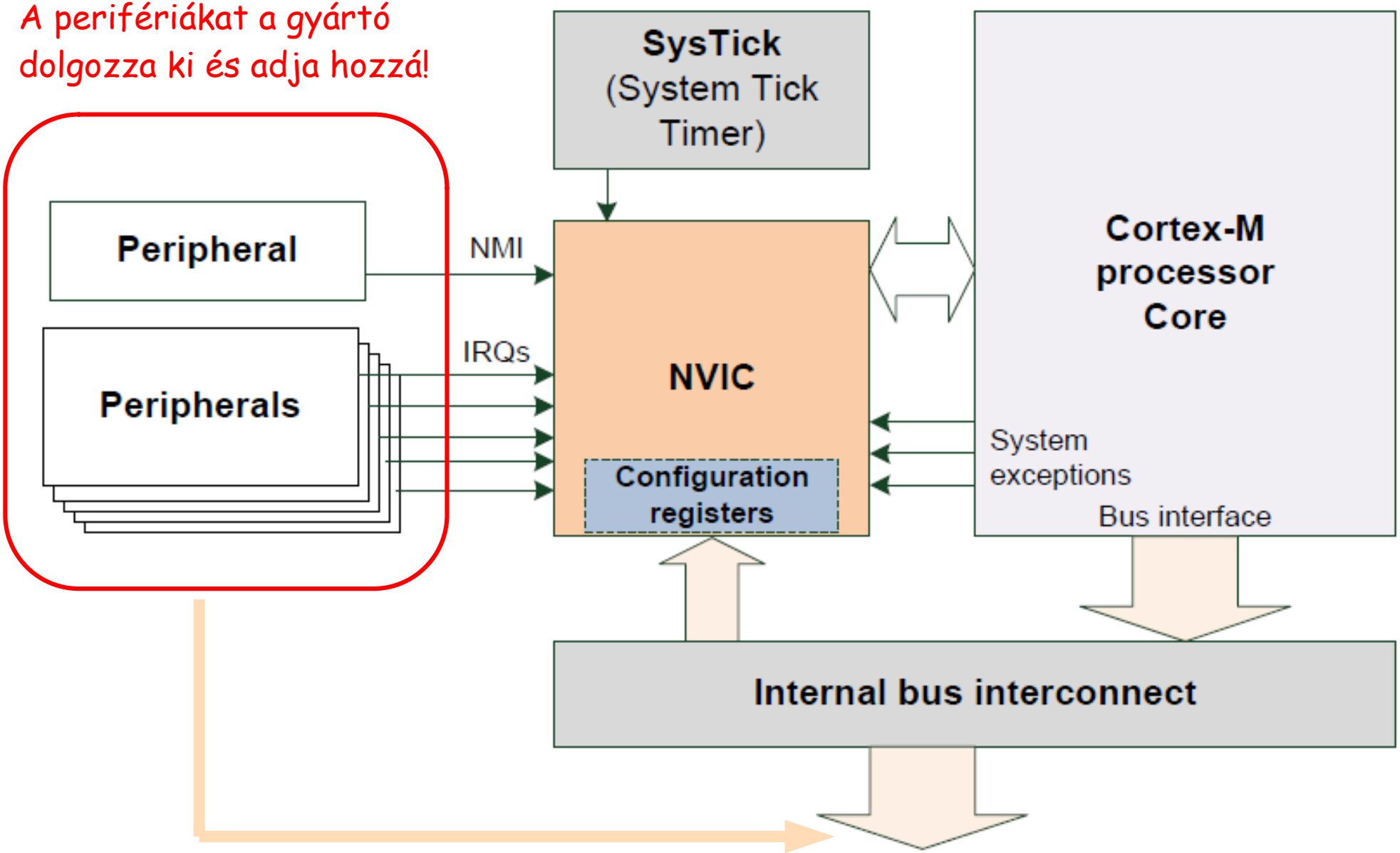


ARM Cortex-M mikrovezérlők

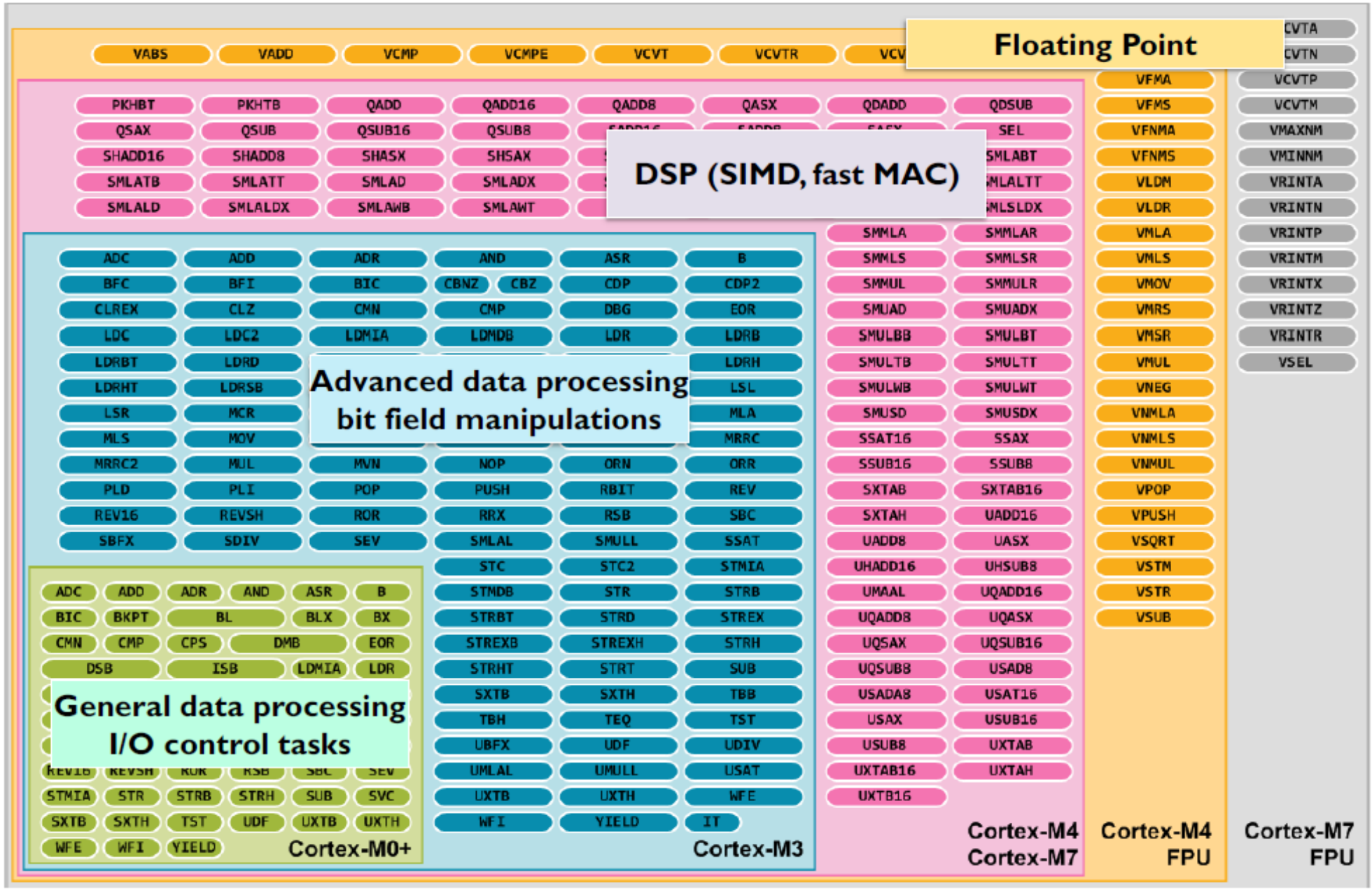
Cortex-M0	Egyszerű felépítésű (~12K kapu), kisfogyasztású processzor beágyazott alkalmazásokhoz
Cortex-M0+	Továbbfejlesztett Cortex-M0 processzor, ultra-kisfogyasztású, egyciklusú I/O, áthelyezhető vektortábla, stb.
Cortex-M1	FPGA-hoz kifejlesztett „szoftveres processzor” (Verilog). Utasításkészlete megegyezik a Cortex-M0-val
Cortex-M3	Hatékony mikrovezérlő, gazdag utasításkészlettel a komplex feladatokhoz. pl. Multiply-Accumulate (MAC) utasítások, hardveres osztás
Cortex-M4	Ugyanaz, mint a Cortex-M3, kibővítve további és hatékonyabb DSP utasításokkal. Opcionálisan egyszeres pontosságú lebegőpontos műveletekkel (Cortex-M4F)
Cortex-M7	Nagyteljesítményű mikrovezérlő CPU intenzív alkalmazásokhoz, duplapontosságú lebegőpontos utasítás, hatékonyabb memóriakezelés

ARM Cortex-M közös jellemzők

A perifériákat a gyártó dolgozza ki és adja hozzá!



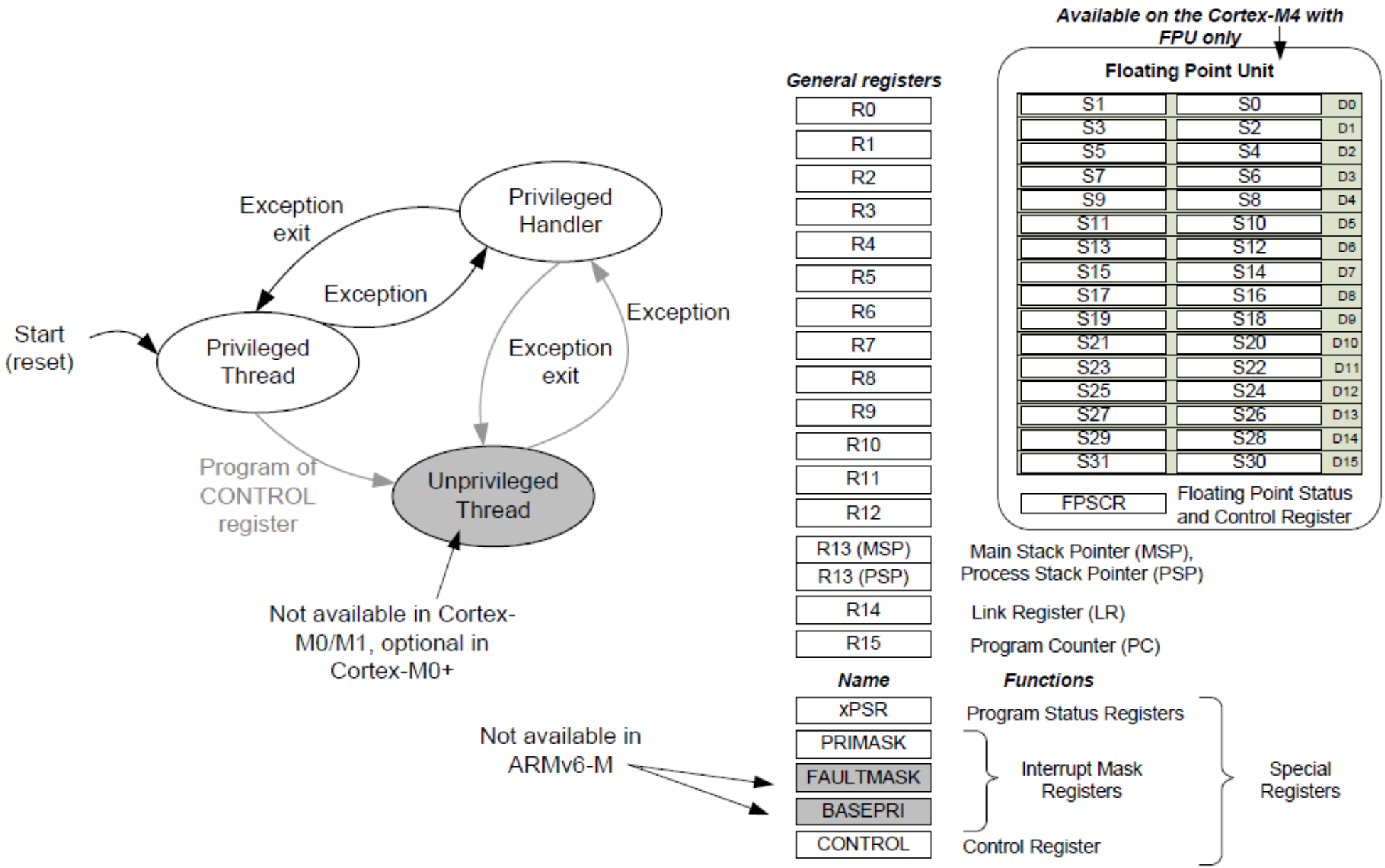
ARM Cortex-M utasításkészlet



ARM Cortex-M utasításkészlet

- A **Cortex-M0/M0+/M1** processzorok csak az alapvető I/O vezérlő műveletek és az általános adatfeldolgozási feladatokhoz alkalmas utasításkészlettel rendelkeznek (56 utasítás, melyek többnyire 16 bitesek – két utasítás egy szóban). Ez a készlet kevés logikai kapuval megvalósítható (~12k gate), de pl. a magasabb sorszámú regiszterek (R8-R12) elérése korlátozott.
- A **Cortex-M3 és M4** processzorok utasításkészlete jóval gazdagabb, számos 32 bites utasítást is nyújt, amelyekkel a felső regiszterek is használhatók. Emellett olyan utasításokkal is rendelkezik, mint:
  - Táblázatos elágaztató utasítások és feltételes végrehajtás
  - hardveres osztó utasítás
  - Szorzás és összegzés (MAC) utasítások
  - Különféle bitmanipulációs műveletek

Programozói modell



Kivételek és megszakítások

Exception Type	ARMv6-M	ARMv7-M	Vector Table	Vector address (initial)
255		Device Specific Interrupts 240/ 8 pri	Interrupt#239 vector [1]	0x000003FC
47	Device Specific Interrupts 32/ 4 pri		Interrupt#31 vector [1]	0x000000BC
17			Interrupt#1 vector [1]	0x00000044
16			Interrupt#0 vector [1]	0x00000040
15	SysTick	SysTick	SysTick vector [1]	0x0000003C
14	PendSV	PendSV	PendSV vector [1]	0x00000038
13	Not used	Not used	Not used	0x00000034
12		Debug Monitor	Debug Monitor vector [1]	0x00000030
11	SVC	SVC	SVC vector [1]	0x0000002C
10	Not used	Not used	Not used	0x00000028
9			Not used	0x00000024
8			Not used	0x00000020
7			SecureFault (ARMv8-M Mainline) [1]	0x0000001C
6		Usage Fault	Usage Fault vector [1]	0x00000018
5		Bus Fault	Bus Fault vector [1]	0x00000014
4		MemManage (fault)	MemManage vector [1]	0x00000010
3	HardFault	HardFault	HardFault vector [1]	0x0000000C
2	NMI	NMI	NMI vector [1]	0x00000008
1			Reset vector [1]	0x00000004
0			MSP initial value	0x00000000
	Cortex-M0	Cortex-M3		

Interrupt késedelem

- **Interrupt latency:** A programmegszakítás érvényesülésétől a megszakítást kiszolgáló eljárás első utasításáig eltelt órajelciklusok száma

	Interrupt latency (number of clock cycles)
Cortex-M0	16
Cortex-M0+	15
Cortex-M3	12
Cortex-M4	12
Cortex-M7	Typically 12, worst case 14

Mit nyújtanak az STM32 mikrovezérlők?

Fejlett perifériakészlet

Kedvező árfekvés

Skálázható termék portfolió

Program-könyvtárak

Fejlesztőeszközök gazdag választéka

Ultra-kis-fogyasztás

Fejlett központi egység

life.augmented

Mi az ST Cortex-M portfólió előnye?

- Elfelejthetjük a hagyományos 8/16/32 bites kategóriákat
- Minden célra találunk alkalmas architektúrát
- Kis fogyasztásra és könnyű felhasználásra optimalizált MCU-k

STM32 F4 termékcsalád portfolio

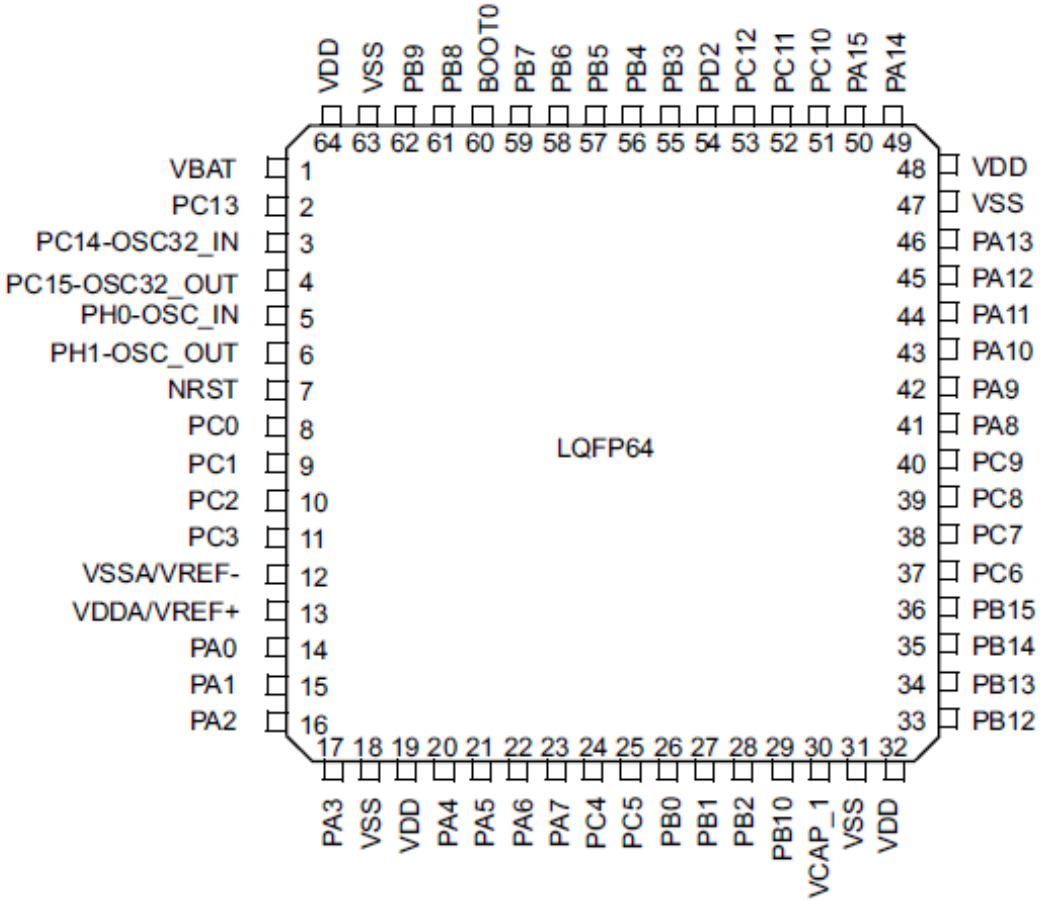
Product lines	F_{CPU} (MHz)	Flash (Kbytes)	RAM (KB)	Ethernet I/F IEEE 1588	2x CAN	Camera I/F	SDRAM I/F	Dual Quad-SPI	SAI	SPDIF RX	Chrom-ART Graphic Accelerator™	TFT LCD Controller	MIPI DSI
Advanced lines													
STM32F469[2]	180	512 K to 2056 K	384	•	•	•	•	•	•		•	•	•
STM32F429[2]	180	512 K to 2056 K	256	•	•	•	•		•		•	•	
STM32F427[2]	180	1024 K to 2056 K	256	•	•	•	•		•		•		
Foundation lines													
STM32F446	180	256 K to 512 K	128		•	•	•	•	•	•			
STM32F407[2]	168	512 K to 1024 K	192	•	•	•							
STM32F405[2]	168	512 K to 1024 K	192		•								

STM32 F4 termékcsalád portfolio

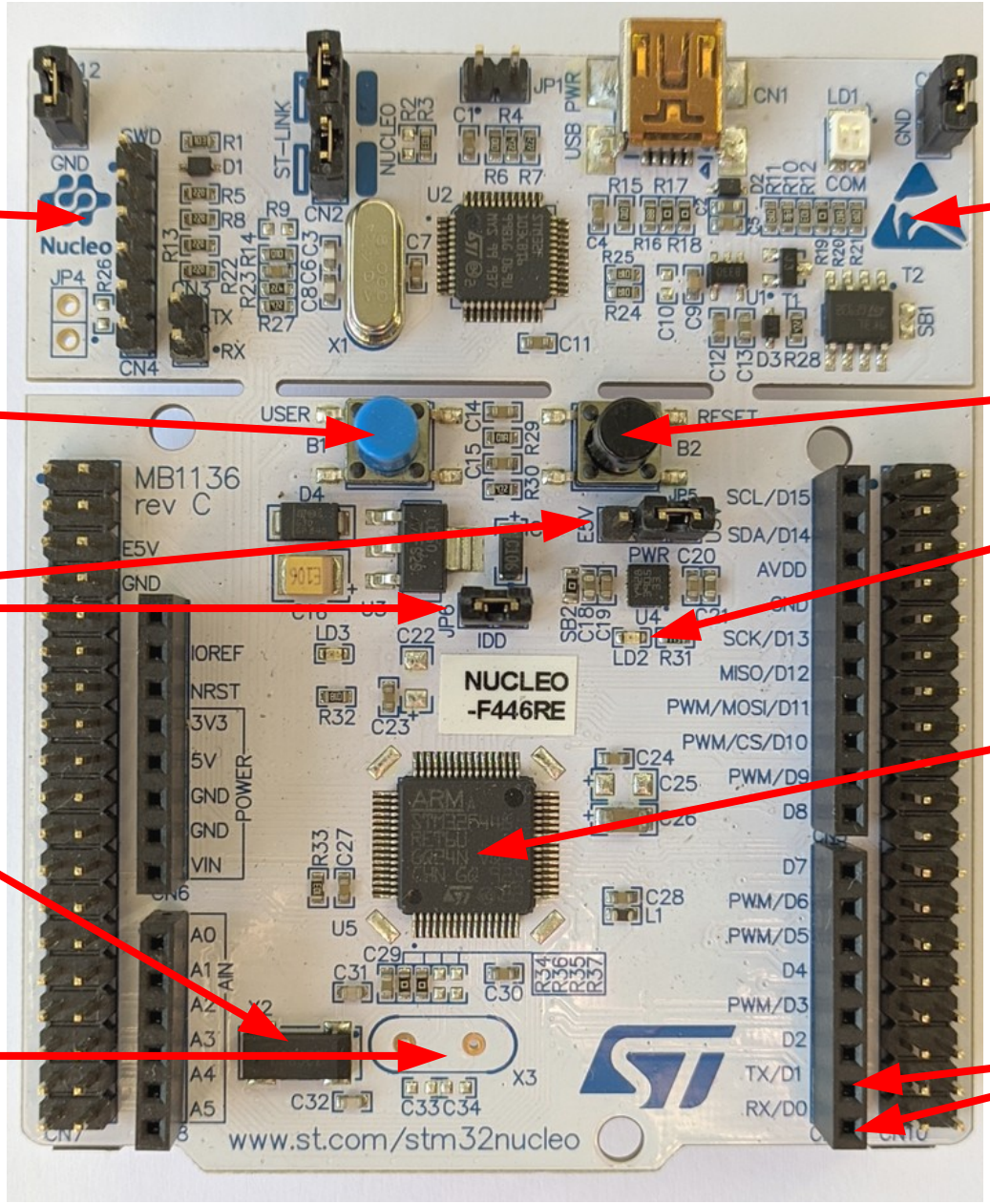
Product lines	F_{CPU} (MHz)	Flash (Kbytes)	RAM (KB)	RUN current (µA/MHz)	STOP current (µA)	Small package (mm)	FSMC (NOR/ PSRAM/LCD support	QSPI	DFSDM	DAC	TRNG	DMA Batch Aquisition Mode	USB 2.0 OTG FS
Access lines													
STM32F401	84	128 K to 512 K	up to 96	Down to 128	Down to 10	Down to 3x3							•
STM32F410	100	64 K to 128 K	32	Down to 89	Down to 6	Down to 2.553x 2.579				•	•	BAM	-
STM32F411	100	256 K to 512 K	128	Down to 100	Down to 12	Down to 3.034x 3.22						BAM	•
STM32F412	100	512 K to 1024 K	256	Down to 112	Down to 18	Down to 3.653x 3.651	•	•	•		•	BAM	• +LPM[1]
STM32F413[2]	100	1024 K to 1536 K	320	Down to 115	Down to 18	Down to 3.951x 4.039	•	•	•	•	•	BAM	• +LPM[1]

Az STM32F446RET6 mikrovezérlő

- 32-bit ARM Cortex-M4F CPU
- Flash: 512KB / RAM: 128KB
- Órajel: max. 180 MHz
- Tápfesz.: 3,3 V (1.8 V – 3.6 V)
- Digitális I/O: 50
- Analóg bemenet: 16
- 3db 12 bites ADC
- 2 db 12 bites DAC
- USB FS/HS, 4 SPI, 4 I2C,
  4 USART, 2 UART, 2 CAN
- 10+2+2 Timer
- RTC, SDIO



A NUCLEO-F446RE fejlesztői kártya



ST Link kimenet

Felhasználói nyomógomb [PC_13]

Átkötések

32 kHz kvarc

8 MHz kvarc nincs beépítve (az órajel az STLink-ből érkezik)

STLink V2-1 programozó/ hibavadász

RESET gomb

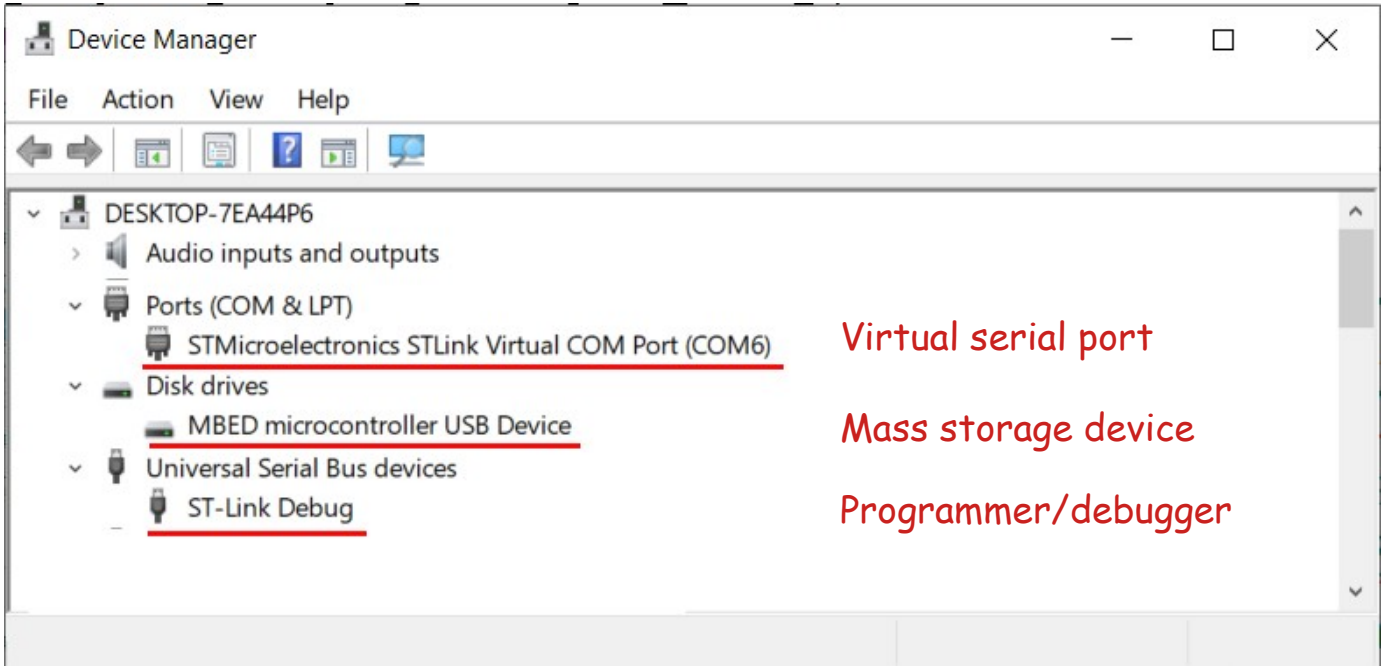
LED

STM32F446RE MCU

Soros port a PC-hez (STLinken át)

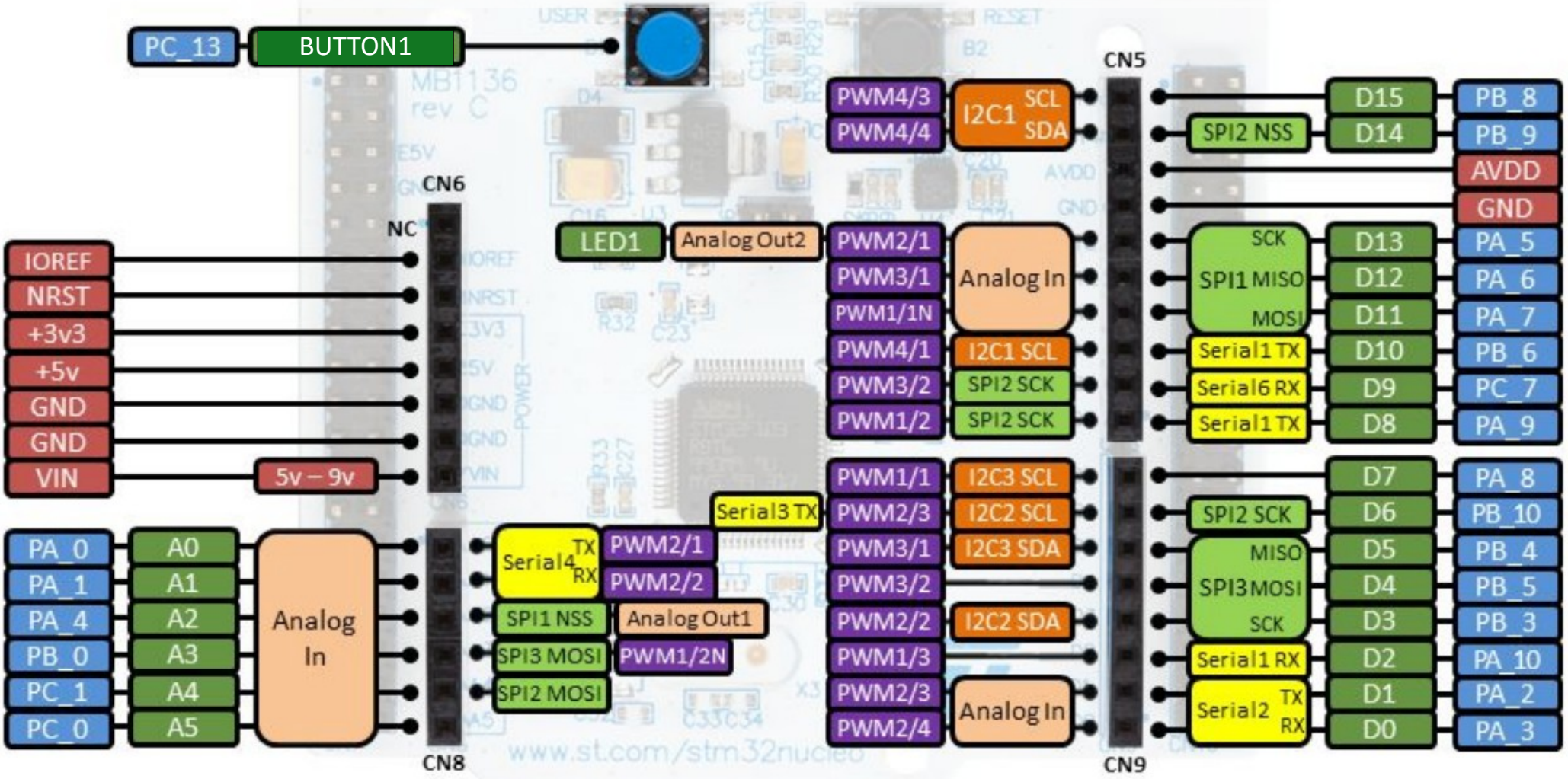
Meghajtó a NUCLEO-F446RE kártyához

- A **NUCLEO** kártyákon található **ST-Link v2-1** programozó USB-re csatlakozik (USB mini-B aljzat) és az SWD programozó illetve hibavadász funkció mellett USB MSD alapú programletöltést is támogat, illetve sorsos kommunikációt is biztosít a PC és a target MCU között (kompozit USB eszközként látszik)
- A kártya USB meghajtó programját az STM32Cube Programmer szoftverrel célszerű telepíteni



Arduino kompatibilis kivezetések

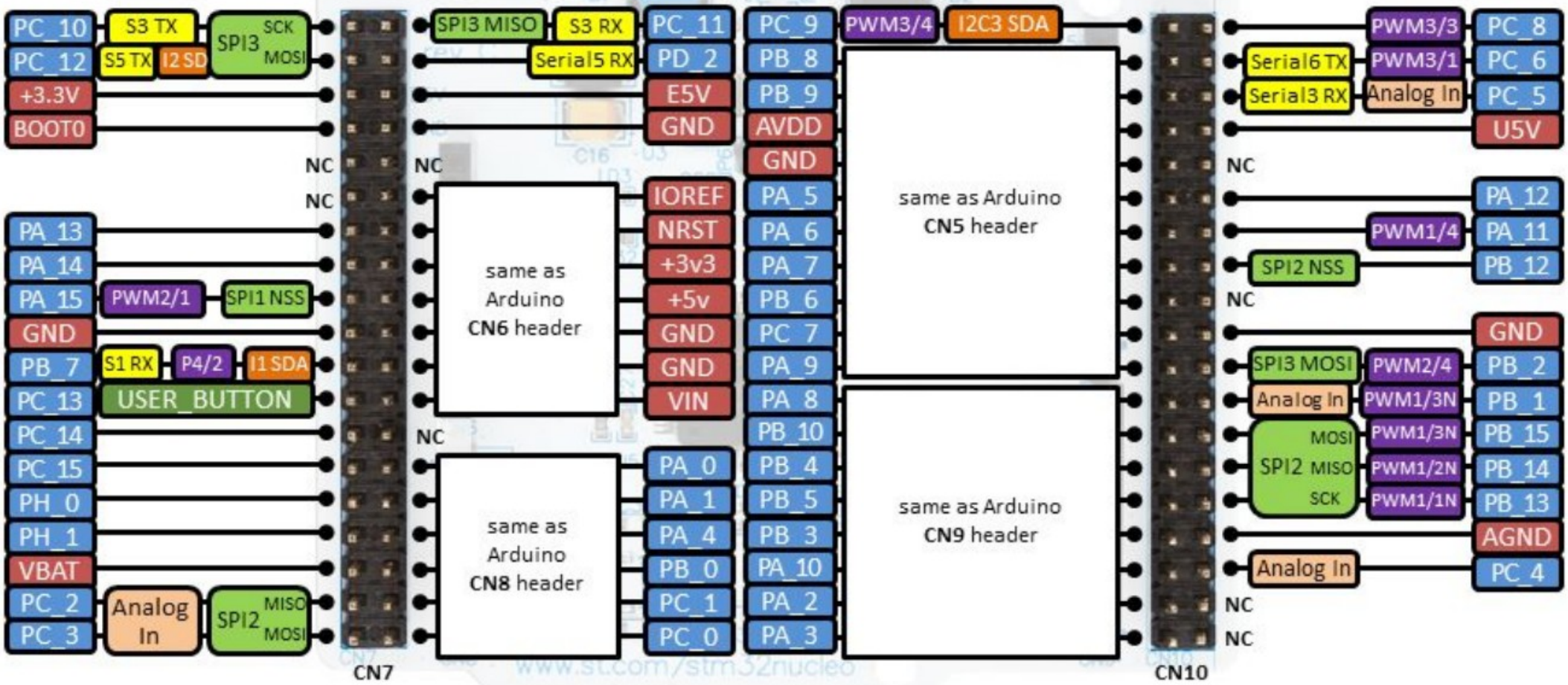
- Kivezetés azonosításra a kék, illetve a sötétzöld címkék használhatók (pl. PA_5, D13, vagy LED1)



Morpho kivezetések

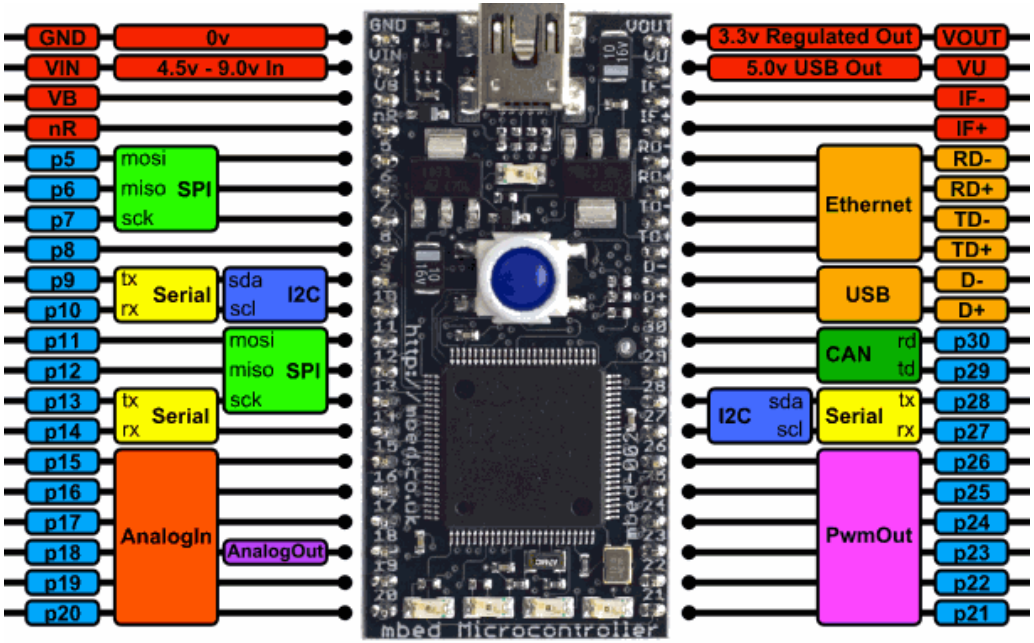
Nucleo F446RE

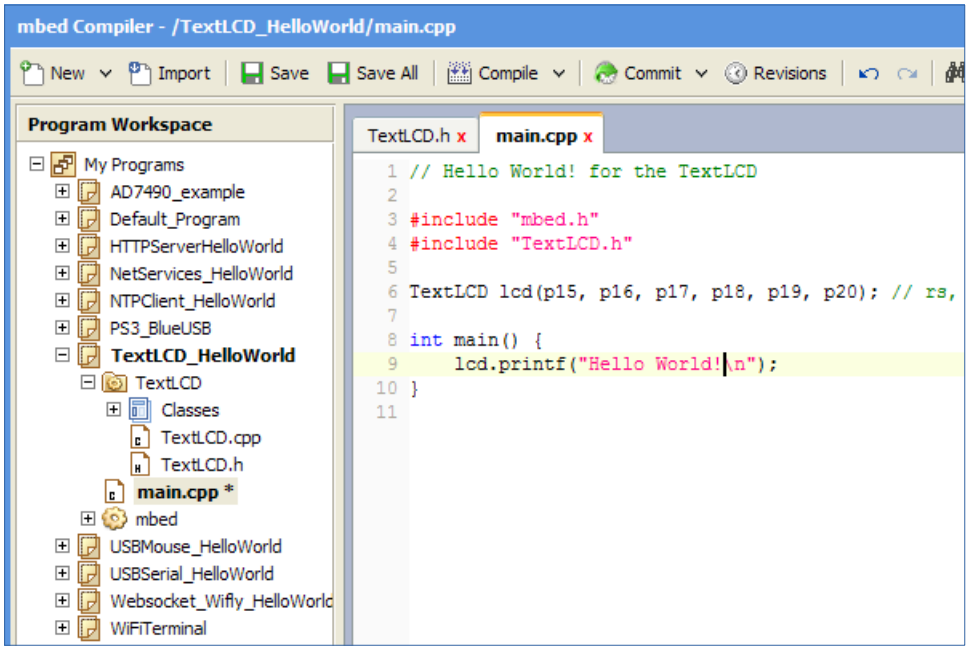
Morpho Headers



MBED – ahogy elkezdődött...

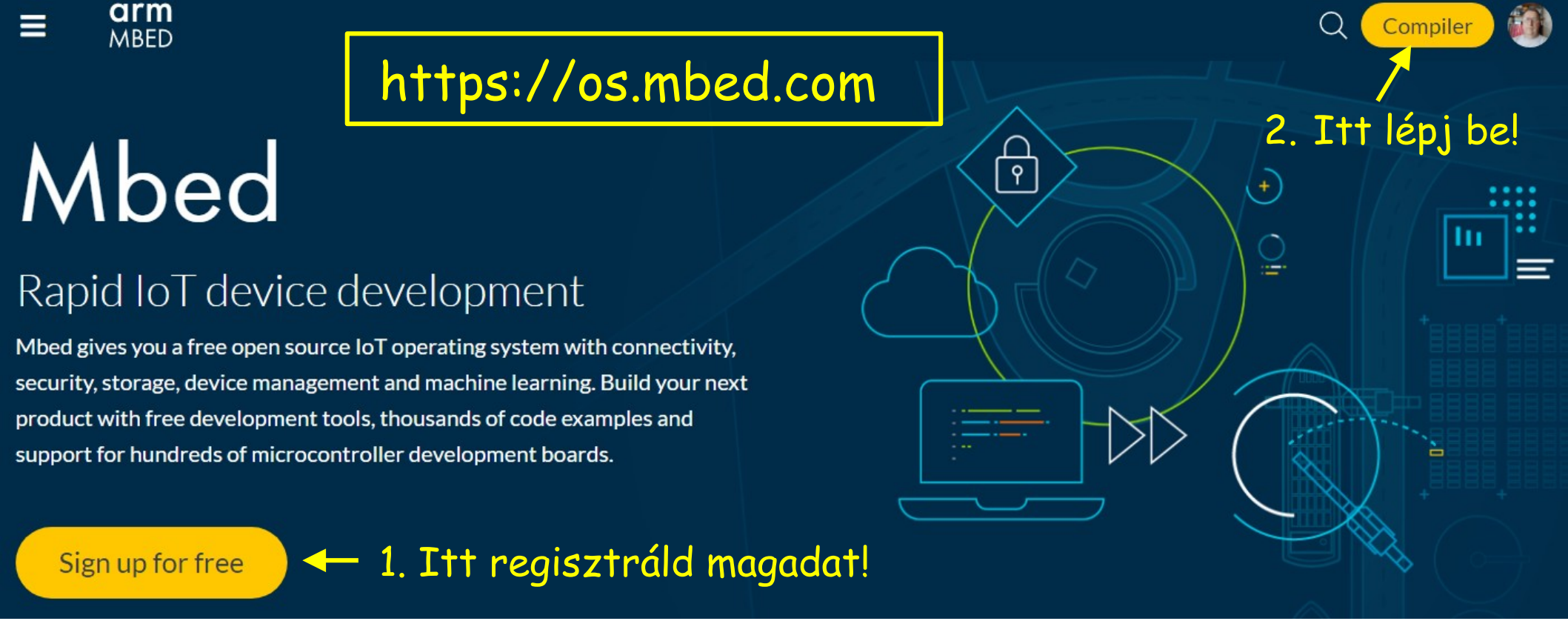
- Az első kiadás: **2009. szeptember 21.**
- A támogatott hardver: NXP **LPC1768** alapú **mbed** kártya (Cortex M3 96 MHz, 512k Flash/32k RAM, Ethernet, USB)
- USB Mass storage alapú programletöltés
- Online Compiler (cloud computing), magas szintű API (C++), hatékony HW+SW prototípus készítésére





https://os.mbed.com

2. Itt lépj be!



1. Itt regisztráld magadat!

- Mi az MBED? Az **MBED** gyorsabbá teszi az eszközfejlesztést, sokféle hardvert támogat (jelenleg 174 kártya támogatott)

MbedOS – C++ API, perifériakönyvtár, RTOS

Operating system

Fejlesztői környezet

Compiler & IDE

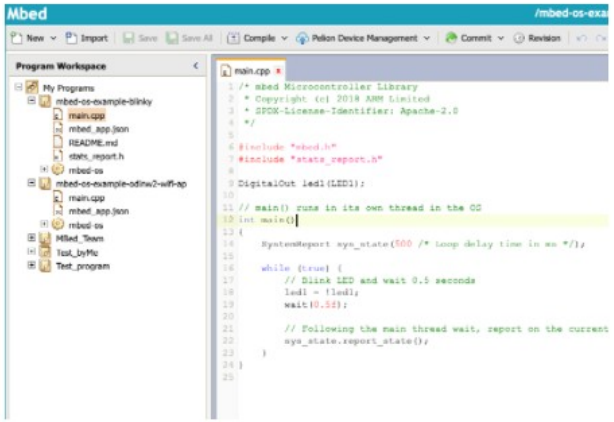
Mintapéldák

MBED fejlesztői környezet(ek)

- A felhőalapú Mbed Online Compiler használatát javaslom, ami nem igényel telepítést és az **mbed 2** projekteket is támogatja
- Az **Mbed Studio** az offline használat mellett a hardveres nyomkövetést is támogatja, de rendkívül erőforrás-igényes, s a ma már elavultnak tekintett **mbed 2** projekteket nem támogatja

Mbed Online Compiler

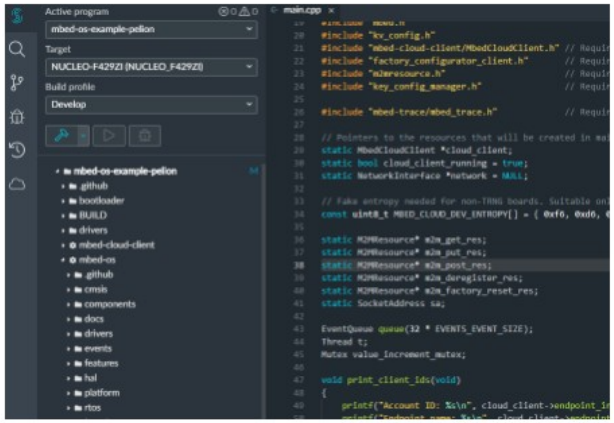
The easiest way to get started.



Jump into application development with Mbed OS directly without installing anything. Create an Mbed account to get started.

Mbed Studio

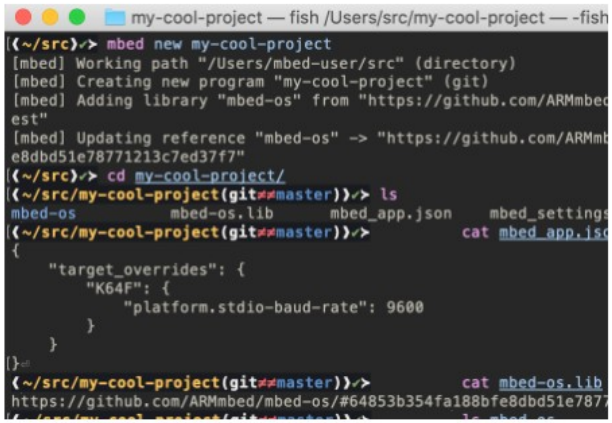
The desktop IDE for Mbed OS.



Mbed Studio is an IDE for application and library development. A single environment with everything you need to create, compile and debug your Mbed programs.

Mbed CLI

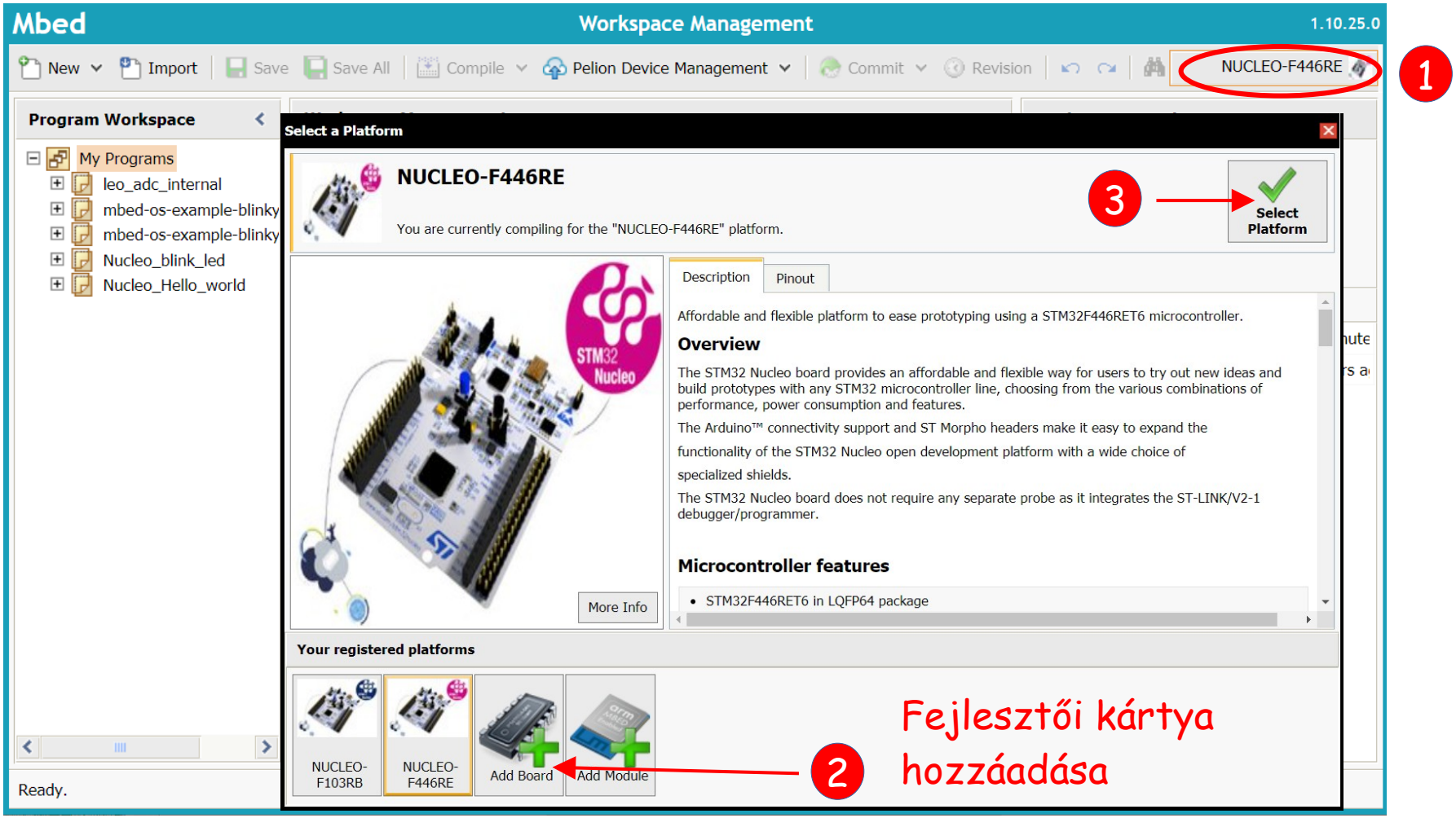
The command line tool for Mbed OS.



Integrate Mbed functionality into your preferred editor, or enhance your automation setup by using Mbed CLI, our command line interface.

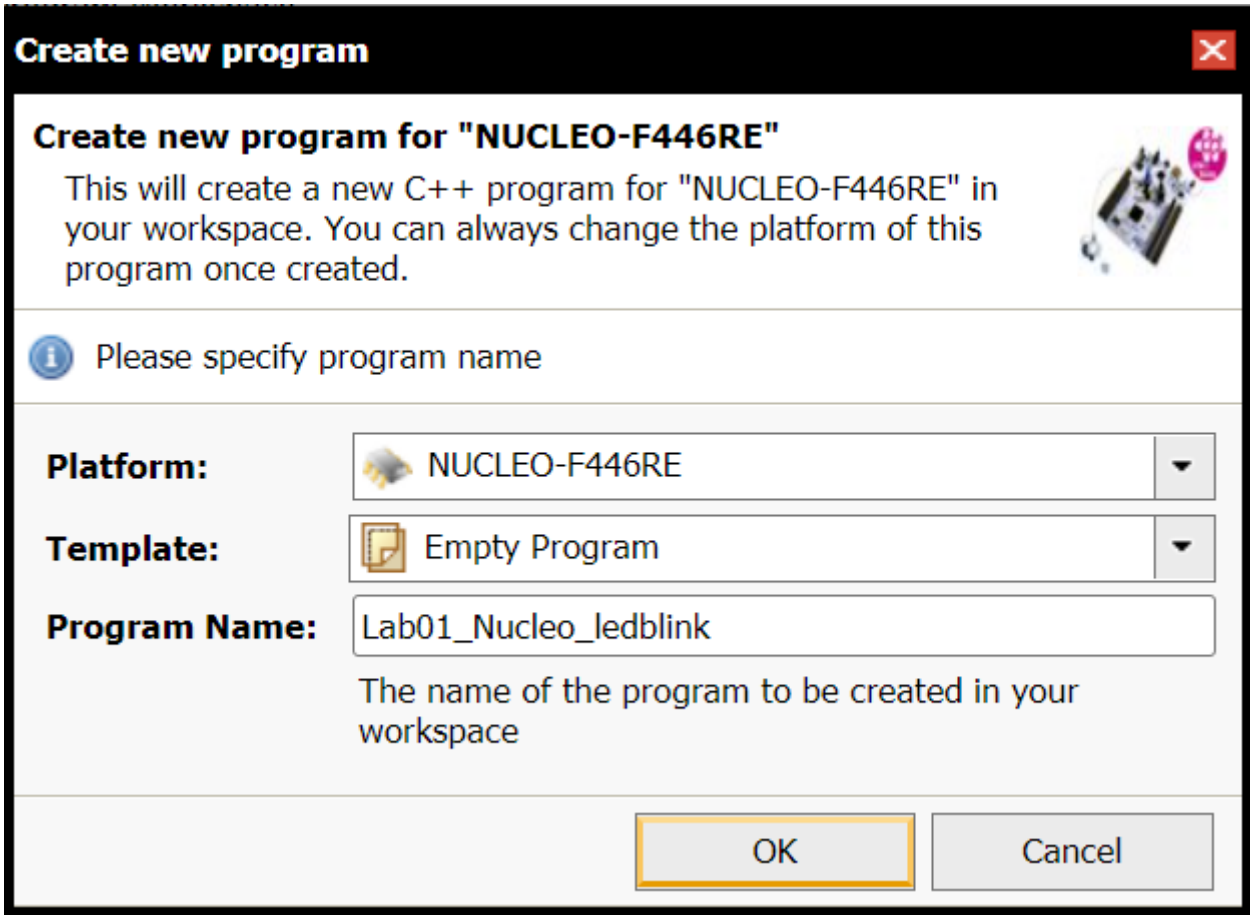
Fejlesztői platform választása

- Ha már regisztráltuk magunkat és bejelentkeztünk az mbed weblapon, akkor a **NUCLEO-F446RE** fejlesztői platformot (kártyát) hozzá kell adni (1, 2), illetve ki kell választani (3)



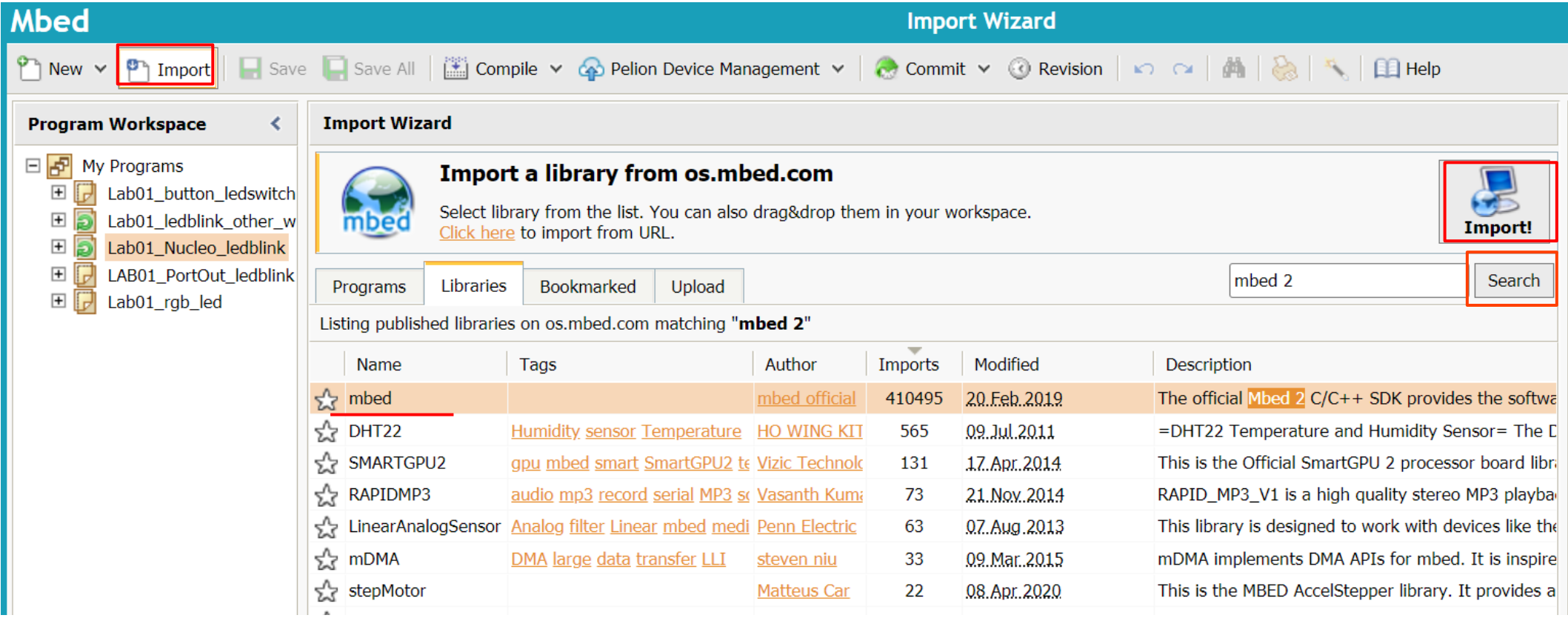
Új projekt létrehozása

- Az Online Compiler ablakában a **File→New→New program** menüpontra kattintsunk
- A felugró ablak rovatait az alábbiak szerint töltsük ki, majd **OK**
- **Mbed 2** projekthez az **Empty Program** sablont válasszuk!



Új projekt létrehozása

- Az üres projekthez minimálisan két dolgot kell hozzáadni:
  - Importáljuk az **mbed** programkönyvtárat: **Import** gomb, majd ***mbed 2*** keresőszó után ezt válasszuk: **mbed** *(mbed official)*
  - Létrehozzuk a **main.cpp** állományt: kattintsunk a **File→New→New file** menüpontra



Lab01_Nucleo_ledblink

- Egészítsük ki az előzőleg létrehozott üres projektet – a **main.cpp** állományba írjuk be az alábbi sorokat:

```
#include "mbed.h"

DigitalOut myled(LED1); // PA_5 alias LED1

int main() {
    while(1) {
        myled = 1;      // LED is ON
        wait(0.2);      // 200 ms
        myled = 0;      // LED is OFF
        wait(1.0);      // 1 sec
    }
}
```

Egy kattintással importálhatjuk a projektet a saját munka-területünkre, ha előtte bejelentkeztünk az **os.mbed.com** lapon az **Online Compiler** környezetbe

- Fordításhoz kattintsunk a **Compile** gombra!
- Futtatáshoz a `Lab01_Nucleo_ledblink.NUCLEO_F446RE.bin` állományt másoljuk be a kártya fájlrendszerébe (**NODE_446RE** meghajtó a PC-n)

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);`

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. state = int(led);`

- A `write()` és `read()` tagfüggvények segítségével a kimenetre írhatunk vagy annak állapotát olvashatjuk vissza, pl. `led.write(1)` vagy `led.read()`, de használhatjuk a rövidebb formát is: `led = 1;`

DigitalIn

- A **DigitalIn** C++ objektumosztály általános célú *digitális bemenetek* konfigurálására és kezelésére szolgál
- A **mode()** taggfüggvény segítségével a bemenet üzemmódja állítható be (**PullUp, PullDown, PullNone, OpenDrain**)

Függvény	Használat
DigitalIn név(pin)	Létrehoz egy "név" nevű objektumot, amelyet a "pin" bemenethez rendel
read()	Az objektumhoz rendelt bemenet állapotának beolvasása (0 vagy 1)
mode()	Az objektumhoz rendelt bemenet üzemmódjának beállítása (PullUp, PullDown, OpenDrain, PullNone)
operator int()	Rövidített alak **read()** helyett

- **Megjegyzés:** A konstruktor függvény hívásakor opcionálisan a kezdeti üzemmód beállítását is megadhatjuk második paramétereként. Szintaxis: `DigitalIn név(pin,mód);`

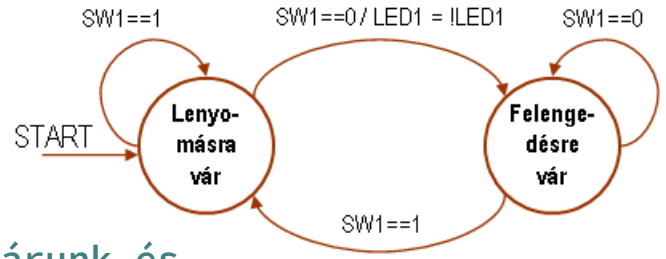
Lab01_button_ledswitch

- A kártya beépített nyomógombjával (BUTTON1) kapcsolgatjuk a beépített LED-ed (LED1)

Import into Compiler

```
#include "mbed.h"

DigitalOut myled(LED1);                     // PA_5 alias LED1
DigitalIn mybutton(BUTTON1,PullUp);         // PC13 alias BUTTON1
int bState, waitforpress=true, led_state=0;
int main()
{
    while(1) {
        bState = mybutton;
        if(waitforpress) {                  //Ha lenyomásra várunk és
            if(!bState) {                   //Ha lenyomás történt...
                led_state = !led_state;     //LED állapotának átbillentése
                myled = led_state;
                waitforpress = false;       //Következő stáció: felengedésre várunk
            }
        } else {                            //Ha felengedésre vártunk és
            if(bState) {                    //Ha felengedést észlelünk...
                waitforpress = true;        //Következő stáció: lenyomásra várunk
            }
        }
        wait(0.02);                         // 20 ms pergésmentesítő késleltetés
    }
}
```



DigitalInOut

- A **DigitalOut** C++ objektumosztály kétirányú digitális be/kimenetek konfigurálására és kezelésére szolgál
- Az adatáramlás irányát az `input()`, illetve az `output()` hívásával kapcsolhatjuk át. A többi tagfüggvény használata megegyezik a **DigitalOut** és a **DigitalIn** objektumosztálynál leírtakkal.

Függvény	Használat
DigitalInOut név(pin)	Létrehoz egy "név" nevű objektumot, amelyet a "pin" kivezetéshez rendel
input()	Bemenetre állítja át a kivezetést
output()	Kimenetre állítja át a kivezetést
write(data)	A kimenet beállítása 0 vagy 1 állapotba
read()	Az objektumhoz rendelt bemenet állapotának beolvasása (0 vagy 1)
mode()	Az objektumhoz rendelt bemenet üzemmódjának beállítása (**PullUp, PullNone**)
operator =	Rövidített alak write(data) helyett `pl. led = 1;`
operator int()	Rövidített alak **read()** helyett `pl. state = int(led);`

Lab01_ledblink_other_way

- Villogtassuk a beépített LED-et más módon: a **LED1 (PA_5)** kivezetés üzemmódjának váltogatásával:
  - ha kimenetnek állítjuk be, akkor meg tudja hajtani a LED1-et
  - ha bemenetnek állítjuk be, akkor megszünteti a LED áramát

Import into Compiler

```
#include "mbed.h"

DigitalInOut myled(LED1);  // LED1 (PA_5) kétirányú kivezetés legyen

int main() {
    while(1) {
        myled.output();    // Kimenetre állítjuk
        myled = 1;         // Magas szintre húzzuk
        wait(0.5);         // 500 ms várakozás
        myled.input();     // Bemenetre állítjuk (LED1 kialszik)
        wait(0.5);         // 500 ms várakozás
    }
}
```

BusOut

- A **BusOut** objektumosztály segítségével több (legfeljebb 16) kimenetet összefoghatunk egyetlen egységgé, a kényelmesebb használat érdekében (természetesen van **BusIn** és **BusInOut** objektumosztály is…)

Függvény	Használat
BusOut név(pin1,pin2,…)	Létrehoz egy "név" nevű **BusOut** objektumot melynek bitjeit rendre a pin1, pin2,… paraméterekkel megadott kimenetekhez rendeli.
write(data)	A kimenetek beállítása a 'data' bitjeivel megadott állapotba
read()	Az int típusú visszatérési érték bitjei a kimenetek állapotát adják meg
operator =	Rövidített alak **write(data)** helyett
operator int()	Rövidített alak **read()** helyett

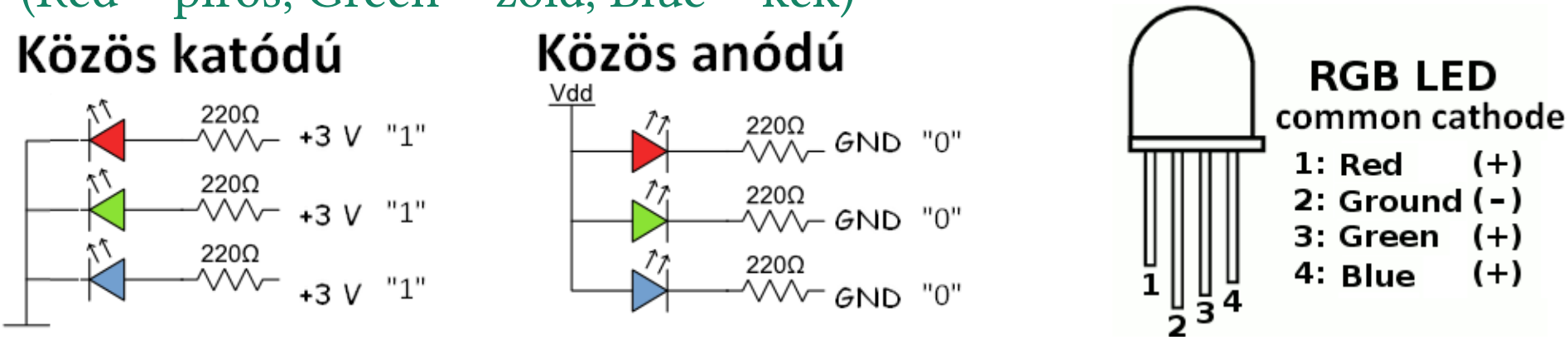
- Ha belenézünk **BusOut** forráskódjába, láthatjuk, hogyan is működik: egy 16 elemű tömbbe tárolja el a konstruktor meghívásakor felsorolt kivezetéseket, s kiíráskor, illetve beolvasáskor ezeket egy ***for*** ciklusban egyenként kezeli. Nem gyorsabb tehát, mintha mi kezelnénk egyenként a kimeneteket, csak kényelmesebbé teszi számunkra a programírást és áttekinthetőbbé, karbantarthatóbbá teszi a forráskódot

BusOut.cpp forráskód (részletek)

```
#include "BusOut.h"
BusOut::BusOut(PinName p0, PinName p1, PinName p2, ... , PinName p15) {
    PinName pins[16] = {p0, p1, p2, ..., p15};
    _nc_mask = 0;
    for (int i=0; i<16; i++) {
        _pin[i] = (pins[i] != NC) ? new DigitalOut(pins[i]) : 0;
        if (pins[i] != NC) {
            _nc_mask |= (1 << i);
        }
    }
}
void BusOut::write(int value) {
    for (int i=0; i<16; i++) {
        if (_pin[i] != 0) {
            _pin[i]->write((value >> i) & 1);
        }
    }
}
int BusOut::read() {
    int v = 0;
    for (int i=0; i<16; i++) {
        if (_pin[i] != 0) {
            v |= _pin[i]->read() << i;
        }
    }
    return v;
}
```

Lab01_rgb_led

- Az RGB LED három, különböző színű LED egy közös tokban (Red = piros, Green = zöld, Blue = kék)



- A LED-et vezérlő kimeneteket itt egy **BusOut** objektumba fogtuk össze

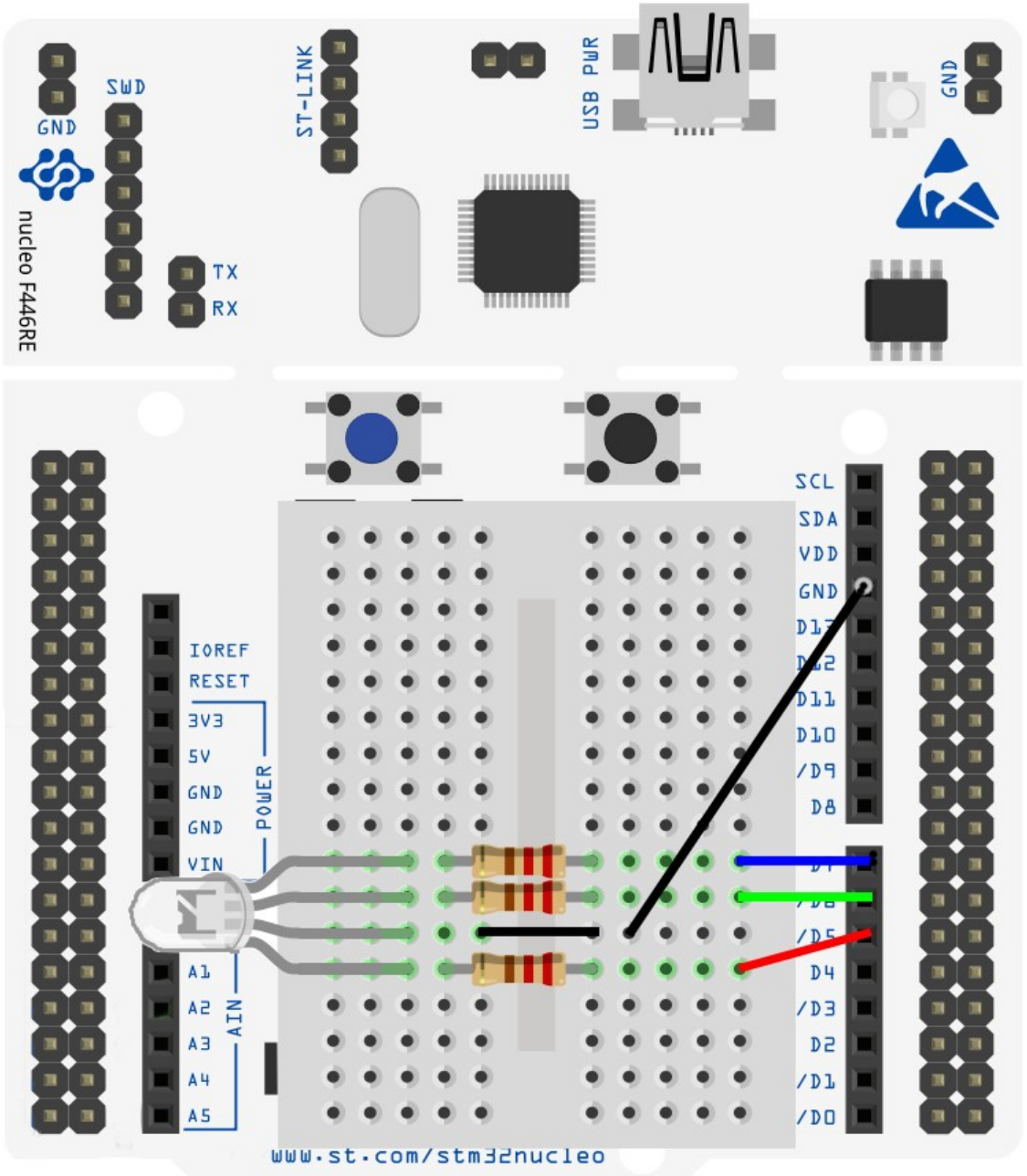
```
#include "mbed.h"

BusOut rgbled(D5, D6, D7);          // RGB outputs

int main()  {                       // A program közös katódú LED-hez való
    while (true) {
        for(int i=0; i<8; i++) {
            rgbled = i;             // Közös anódú LED esetén 7-i kellene...
            wait(2.0);
        }
    }
}
```

Lab01_rgb_led

- A közös katódú RGB LED bekötési vázlat az ábrán látható:
  R ← D5 (PB_4)
  G ← D6 (PB_10)
  B ← D7 (PA_8)
  közös láb ← GND
- Közös anódú LED esetén a közös kivezetést tápfeszültségre (3,3 V) kell kötni!
- Megjegyzés: egyes típusoknál a G és B kivezetés sorrendje fordított



BusIn

- A **BusIn** objektumosztály segítségével több (legfeljebb 16) digitális bemenetet foghatunk össze egyetlen egységgé

Függvény	Használat
BusIn név(*pin1,pin2,...*)	Létrehoz egy "név" nevű **BusIn** objektumot melynek bitjeit rendre a *pin1, pin2,...* paraméterekkel megadott (legfeljebb 16 db) bemenetekhez rendeli
read()	Az *int* típusú visszatérési érték bitjei a hozzájuk rendelt kimenetek állapotát adják meg
mode(pull**)**	A **BusIn** objektumhoz rendelt mindegyik bemenet üzemmódját egységesen beállítja a *pull* paraméterrel megadott üzemmódba (esetünkben **PullUp** vagy **PullNone** mód használható)
mask()	A visszatérési érték bitjei az adott sorszámú bemenet definiáltságát jelzik (0: nem definiált, 1: definiált). Például `BusIn buttons(D3,D4);` megadása után `buttons.mask()` visszatérési értéke 3 lesz.
operator int()	Rövidített alak **read()** helyett
operator[index**]**	A **BusIn** objektumhoz rendelt bemenetek közül az *index* sorszámú bemenet állapotának beolvasása (0 vagy 1)

- **Megjegyzés:** Egy **BusInOut** objektumhoz rendelt bemenetek üzemmódja csak együttesen változtatható. Vagy mindegyiket felhúzott bemenetnek (**PullUp** mód) állítjuk, vagy egyiket sem

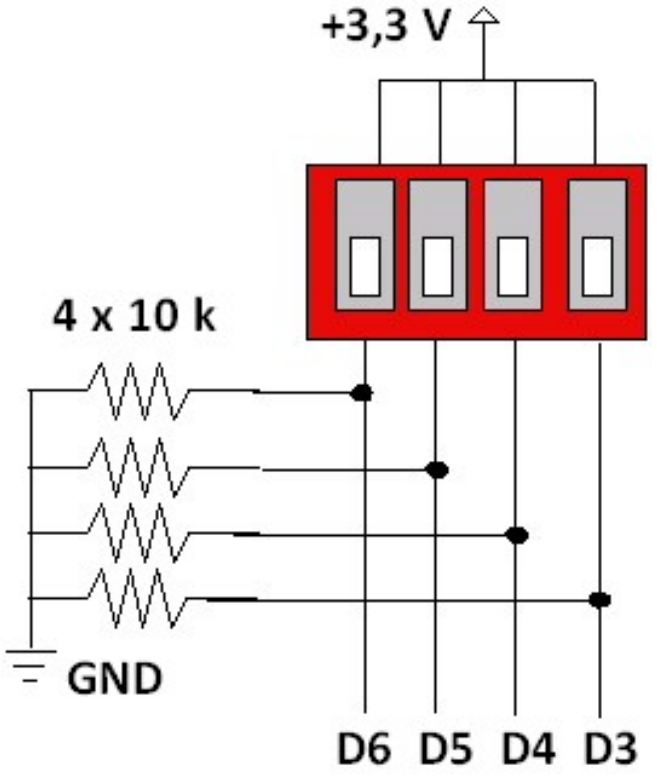
BusIn – egy felhasználási ötlet

- A **BusIn** objektumosztály felhasználható például egy kapcsolósoron beállított bináris szám beolvasására, amivel a programban valamit vezérelhetünk vagy kiválaszthatunk (pl. dallamcsengőn a dallamot).
- A rajz szerint bekötött kapcsolósor esetén például így hozhatjuk létre a **myswitch** nevű **BusIn** típusú objektumot:

```
BusIn myswitch(D3, D4, D5, D6);
```

- A kapcsolósoron beállított értéket pedig így olvashatjuk be:

```
int state = myswitch.read();
```



PortOut

- Az egy porthoz tartozó több (legfeljebb 16) digitális ki- vagy bemenetet csoportos kezelésére az mbed API a **PortOut, PortIn** és **PortInOut** objektumosztályokat definiálja. Itt csak **PortOut** tagfüggvényeit mutatjuk

Függvény	Használat
PortOut név(port,mask)	Létrehoz egy "név" nevű **PortOut** objektumot melyhez a megadott *port*-ot rendeli (**PortA, PortB,...**) és amelynek csak a *mask*-kal megadott bitjeit akarjuk írni.
write(data)	A kimenetek beállítása a 'data' bitjeivel megadott állapotba
read()	Az int típusú visszatérési érték bitjei a kimenetek állapotát adják meg
operator =	Rövidített alak **write(data)** helyett
operator int()	Rövidített alak **read()** helyett

- **Megjegyzés:** A **PortOut** metódusainak **mbed 2** implementációja a bitcsoportok írásánál nem használja sem a mikrovezérlő hardveres támogatását (bit banding), sem bitenkénti kizáró vagy (XOR) nyújtotta lehetőséget az olvasás - módosítás - visszaírás műveletsor biztonságos elvégzéséhez, ezért konkurens hozzáférésre ne használjuk!
- Az **mbed-os** már kritikus szekcióban végzi a bitcsoport módosításokat

Lab01_portOut_ledblink

- Az RGB LED-et vezérlő kimeneteket itt egy **PortOut** objektum fogja össze (ügyeljünk rá, hogy minden kivezetés azonos porthoz tartozzon!)

Import into Compiler

```
#include "mbed.h"
#define LED_MASK     0x00000038    //bits 3, 4, 5
#define RED_MASK     0x00000008    //bit 3 (D3): LED_RED        1<<3
#define GREEN_MASK   0x00000020    //bit 5 (D4): LED_GREEN      1<<5
#define BLUE_MASK    0x00000010    //bit 4 (D5): LED_BLUE       1<<4

PortOut ledport(PortB, LED_MASK);

int main() {
    while(1) {
        ledport = RED_MASK;        // Piros fény
        wait(0.5);
        ledport = GREEN_MASK;      // Zöld fény
        wait(0.5);
        ledport = BLUE_MASK;       // Kék fény
        wait(0.5);
        ledport = LED_MASK;        // Fehér fény
        wait(0.5);
    }
}
```