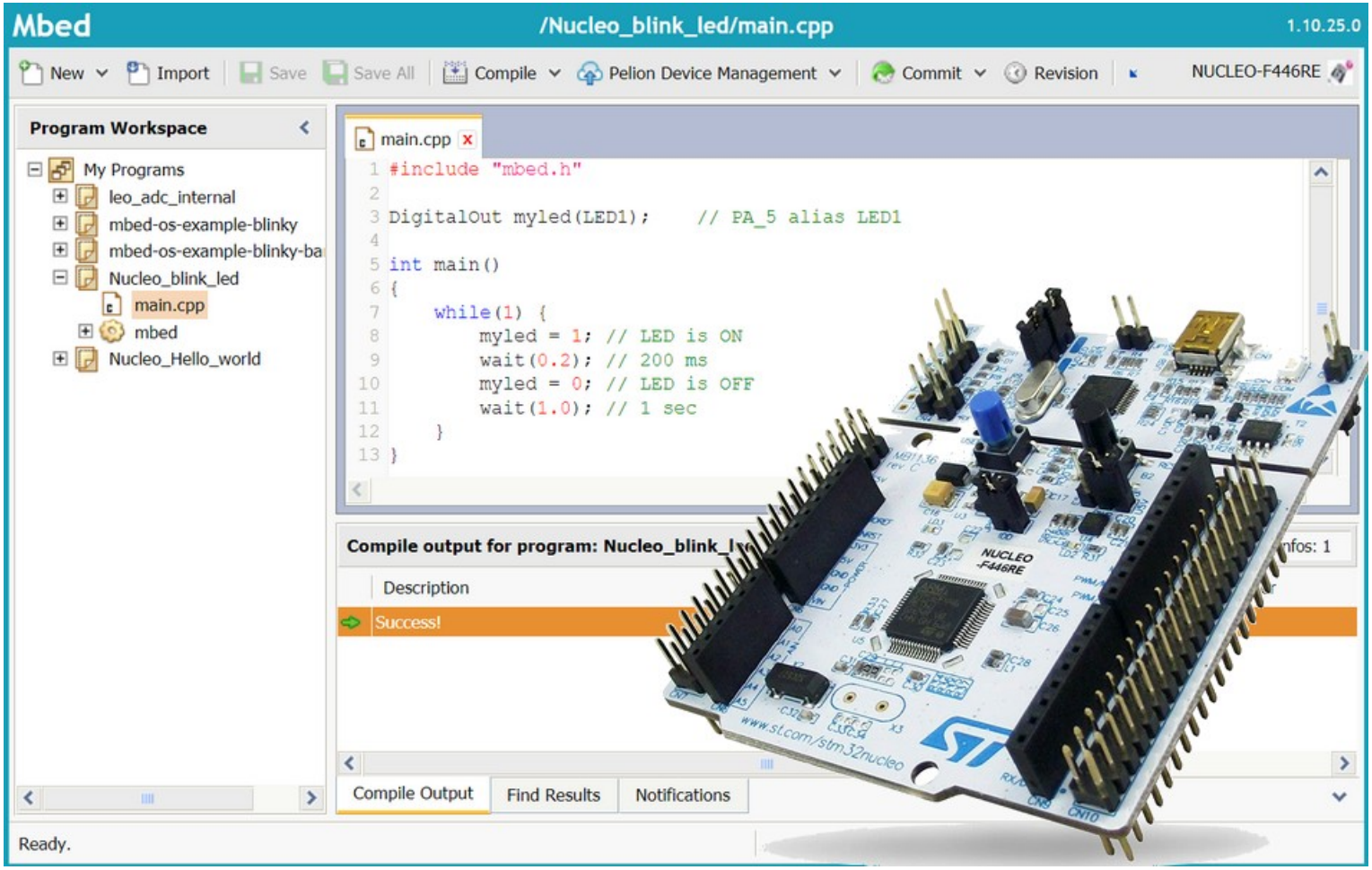
STM32 mikrovezérlők programozása ARM mbed környezetben



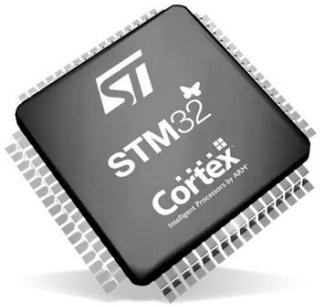
4. Programmegszakítások, időzítők

Felhasznált és ajánlott irodalom

- Cserny István: A FRDM-KL25Z kártya programozása mbed környezetben
- 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
- Dogan Ibrahim: ARM-based Microcontroller Projects Using mbed
- **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



Példaprogramok

Lab04_button_interrupt – megszakítás
Lab04_b2int – LED kapcsolgatás két gombbal
Lab04_button_bounce – nyomógomb pergése
Lab04_button_debouncer – pergésmentesítés
Lab04_reaction_time – reakcióidő mérése
Lab04_rtc_time – Real time óra beírás/kiolvasás
Lab04_wakeup_STM32 – energiatakarékos mód
Lab04_check_Standby_os2 – az alvó MCU áramfelvételének ellenőrzése és mintaalkalmazás

A mintaprogramok az os.mbed.com/users/cspista/code/ oldalon is megtalálhatók

Lekérdezés vagy programmegszakítás?

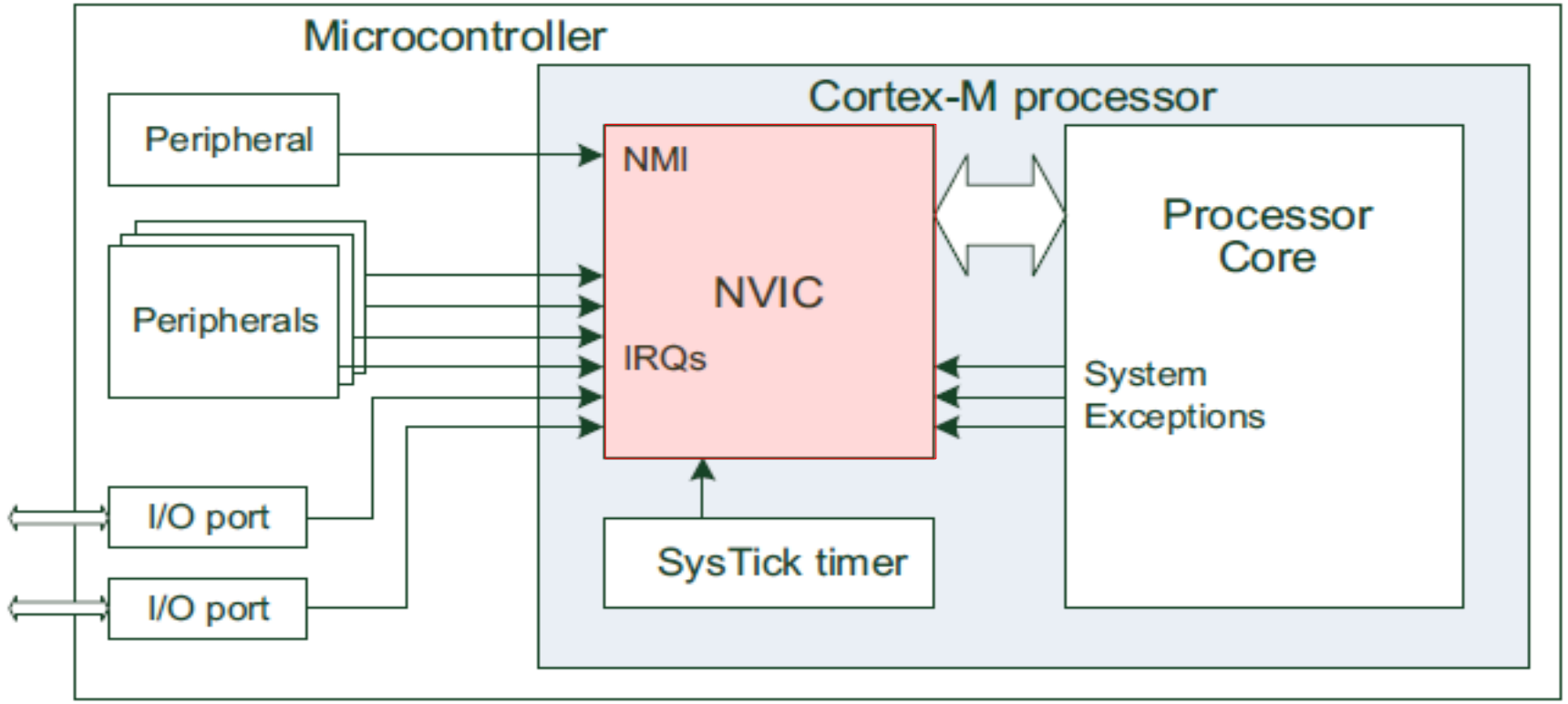
- **A lekérdezéses mód *(polling)*** olyan, mintha a telefonba időnként belehallóznánk, hogy van-e ott valaki. Ez egyszerű módszer, de nem hatékony

- **Programmegszakítás *(interrupt):*** egy hardver esemény bekövetkezte megszakítja a program futását, kiszolgálja az eseményt, majd visszatér
- **A programmegszakítás** olyan, mint amikor a telefon csöng. Abbahagyjuk, amit éppen csináltunk, s felvesszük a kagylót. A telefonálás után ott folytatjuk a korábbi tevékenységet, ahol félbeszakítottuk, amikor a telefont felvettük

- A megszakított tevékenység lehet ”alvás” (energiatakarékos mód) is, ha ébresztési **eseményt *(event)*** keltünk

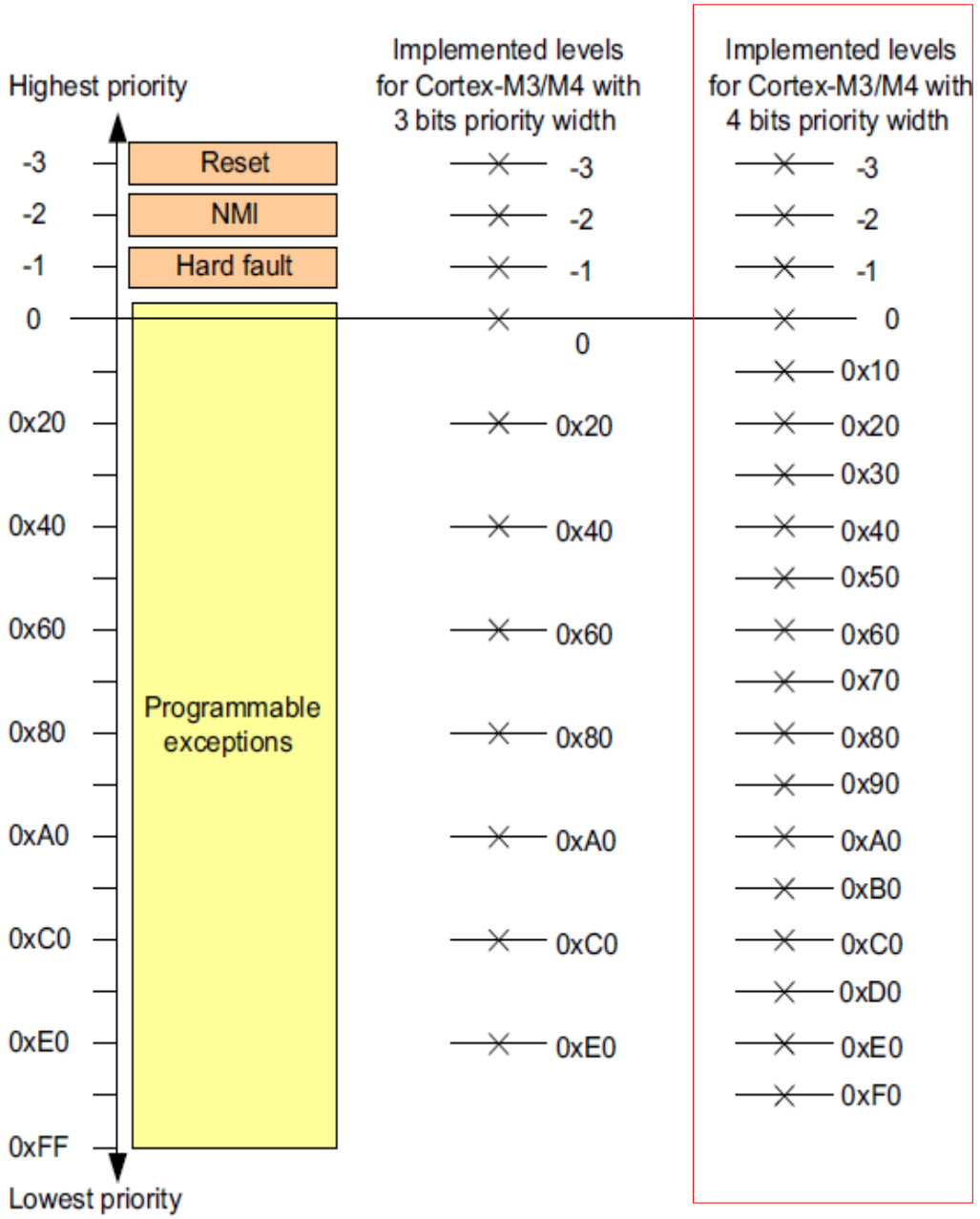
ARM Cortex-M megszakítási rendszer

- Az MCU megszakítás vezérlője (**NVIC** = Nested and Vectored Interrupt Controller) számos forrásból fogad megszakítási kérelmeket
- A megszakítási rendszer külön belépési pontot (vektor) és beállítható prioritást rendel az egyes megszakításokhoz
- A magasabb prioritású megszakítás az alacsonyabb prioritású megszakítás kiszolgálását is megszakíthatja



Megszakítások prioritása

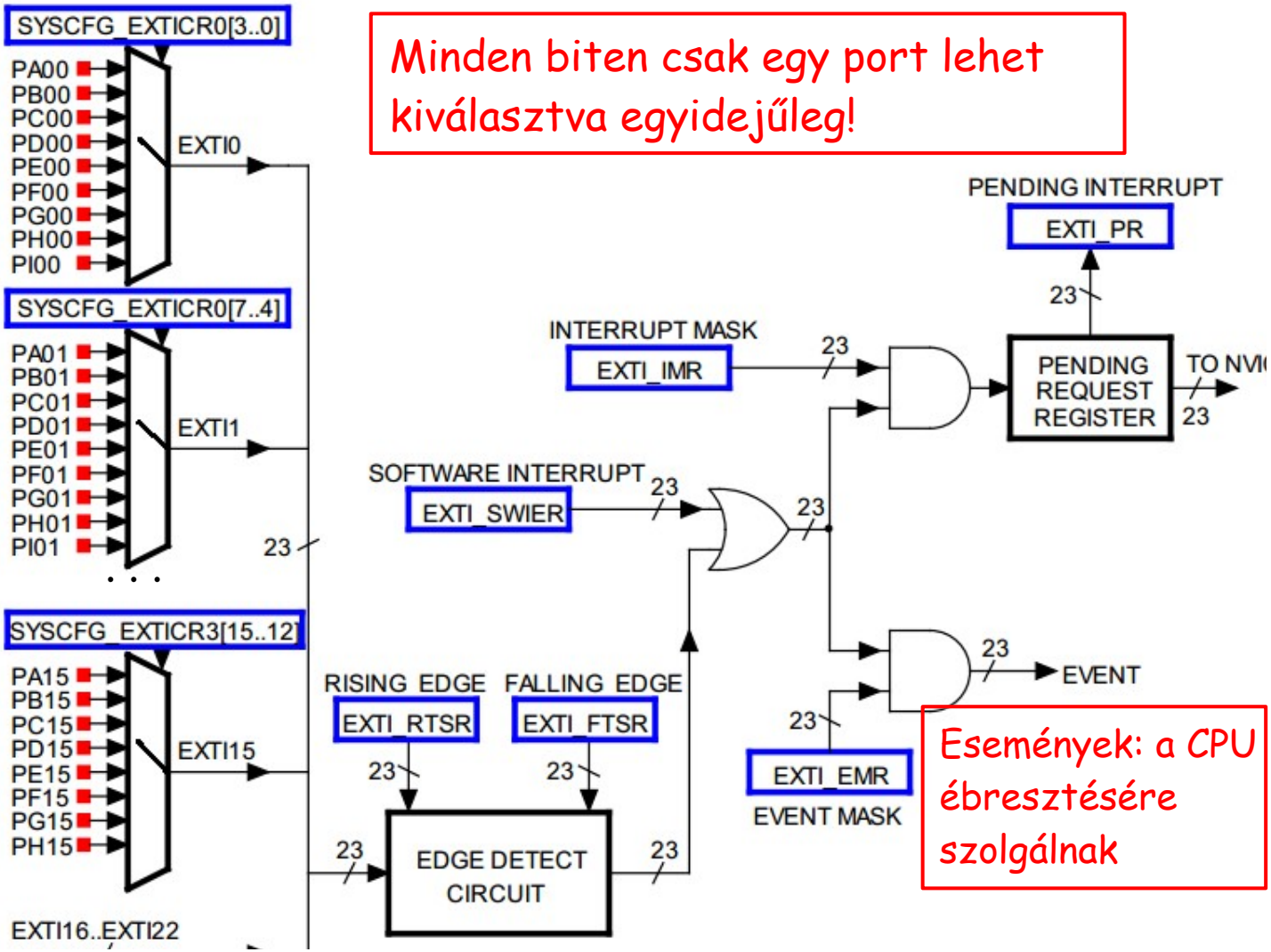
- A Cortex-M3 és Cortex-M4 processzorok felépítése három fix, magas prioritású és legfeljebb 256 szintű programozható prioritási szintet definiál
- Az **STM32F446RE** MCU 4-bites megszakítási prioritás beállítást implementál



Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Implemented				Not Implemented			

Külső megszakítások (EXTI) kezelése

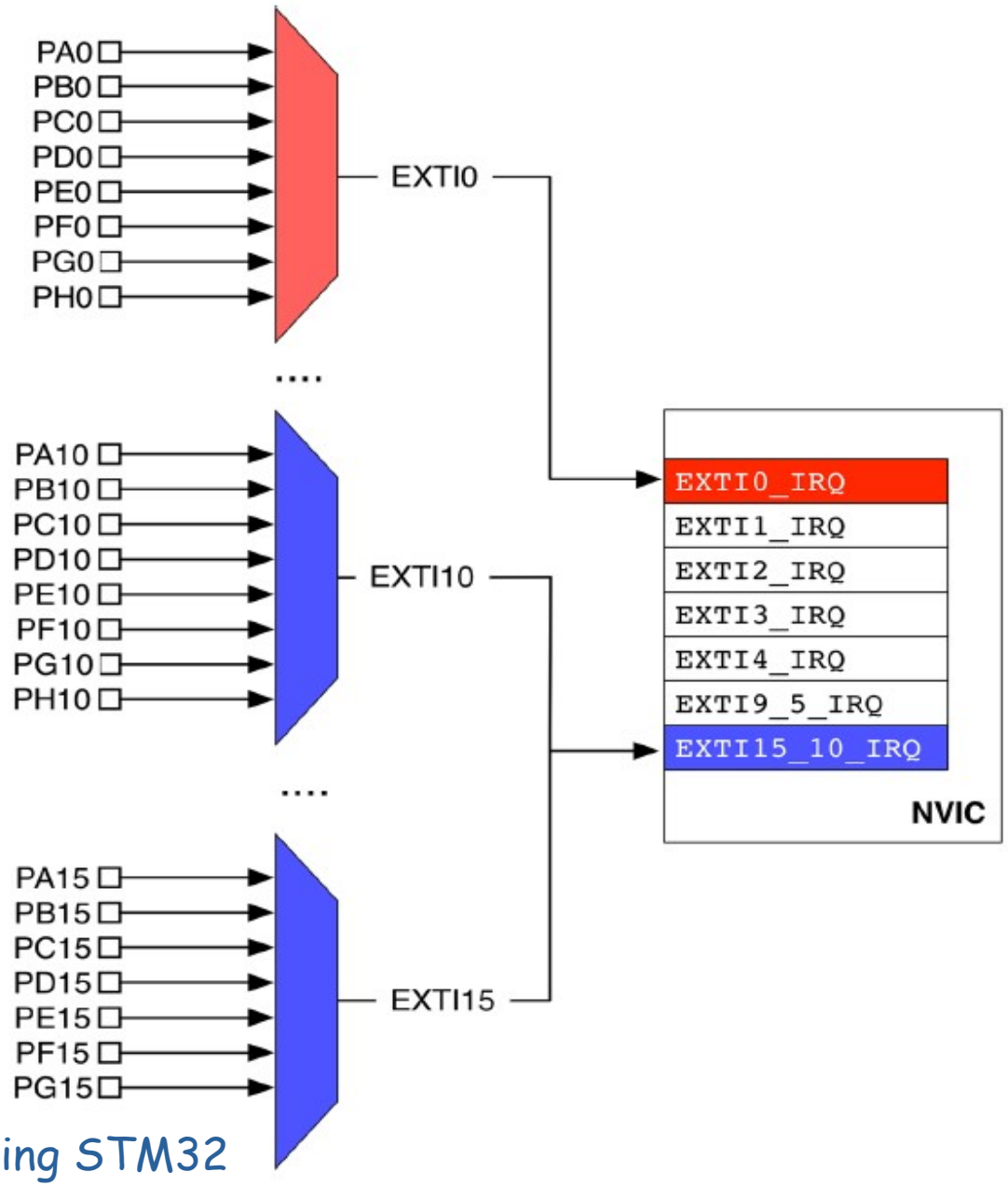
- Az **EXTI** modul a külső megszakítások vezérlője, 23 bemenete közül 16 a GPIO vonalakat fogadja (a portokat multiplexelve)
- Az EXTI16-EXTI22 vonalak egyéb eseményekre reagálnak
- A megszakításokat az élkiválasztást és az ébresztést az EXTI modul regiszterei vezérlik
- A megszakítási jel az **NVIC**-be jut



A kép forrása: https://www.cnblogs.com/shangdawei/p/4723789.html

A megszakítási vektorok kiosztása

- Az **NVIC** megszakításvezérlő csak 7 megszakítási vektort biztosít, az **EXTI5 – EXTI9** és az **EXTI10 – EXTI15** vonalak csak egy-egy vektort kaptak
- Engedélyezett megszakítás esetén az **EXTI_PR** regiszter megfelelő bitje jelzi a megszakítási kérelmet, ennek alapján azonosítható a közös vektorba „beeső” megszakítás eredete



Az ábra forrása: Carmine Noviello, Mastering STM32

Megszakítások használata mbed API-val

- A külső megszakítási kéréseket (a digitális bemenetek állapotváltozási eseményeit), az **InterruptIn** objektumosztály segítségével kezelhetjük, s a le-, illetve felfutáshoz ún. visszahívási függvényeket rendelhetünk

```
InterruptIn my_button(BUTTON1);
my_button.mode(PullUp);
my_button.fall(&my_ISR);
my_button.enable_irq();
```

Függvény	Használat
InterruptIn név(pin**)**	Létrehoz egy *"név"* nevű **InterruptIn** objektumot és a *"pin"* paraméterrel megadott digitális bemenet megszakításinak kezelését az objektumhoz rendelei.
mode(pinmode**)**	Az objektumhoz tartozó digitális bemenet felhúzási módjának (**PullUp, PullNone**) beállítása .
enable_irq()	Az objektumhoz tartozó megszakítás engedélyezése.
rise(*fptr**)**	A felfutáshoz tartozó megszakítást engedélyezi és visszahívandó függvényt rendel hozzá (*fptr: függvény pointer)
fall(*fptr**)**	A lefutáshoz tartozó megszakítást engedélyezi és visszahívandó függvényt rendel hozzá (*fptr: függvény pointer)
disable_irq()	Az objektumhoz tartozó megszakítás letiltása.

Lab04_button_interrupt/main.cpp

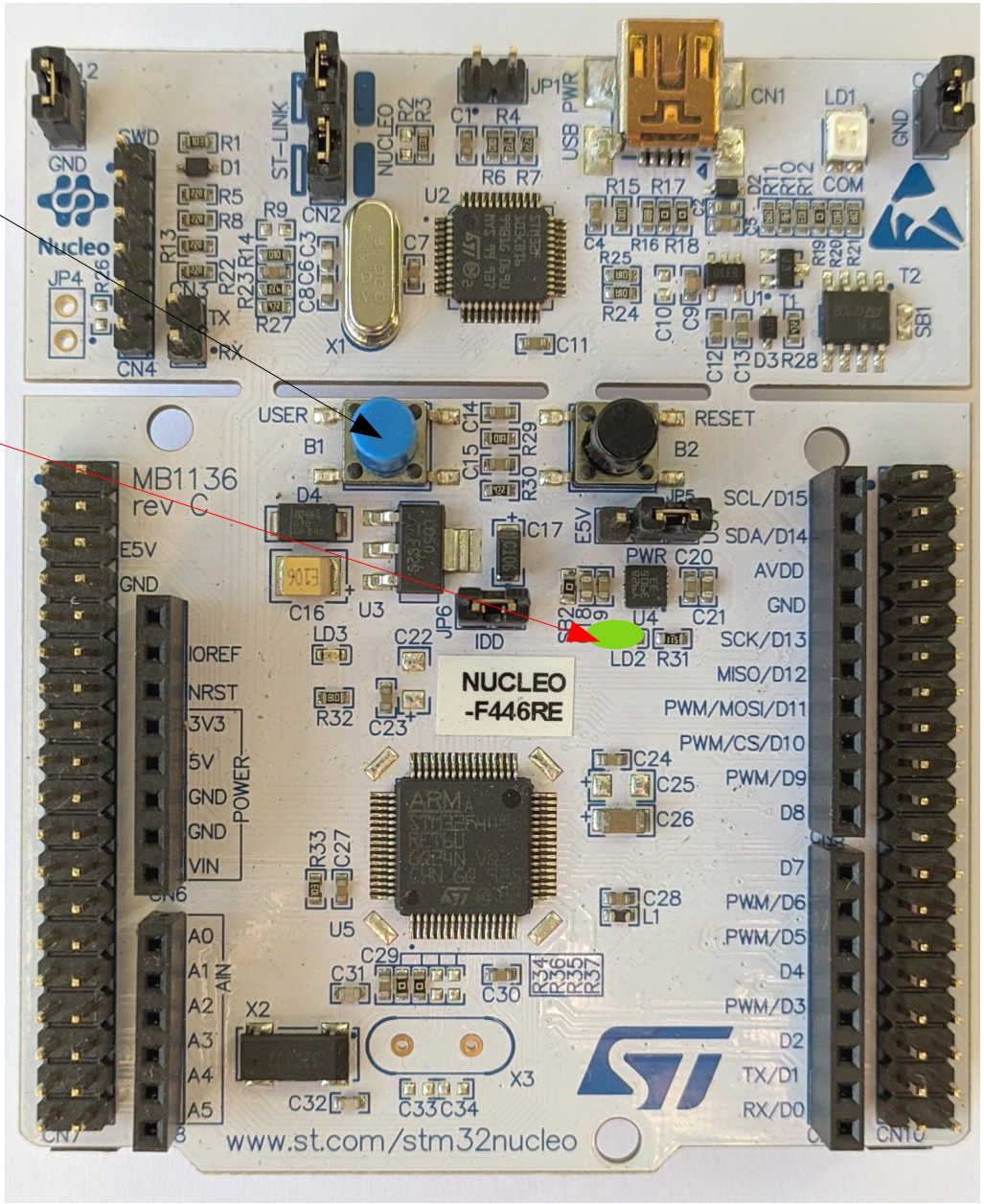
- Az alábbi programban a beépített nyomógombbal kapcsolgatjuk a beépített LED-et
- A megoldás „szépséghibája”, hogy a gomb pergése nincs lekezelve

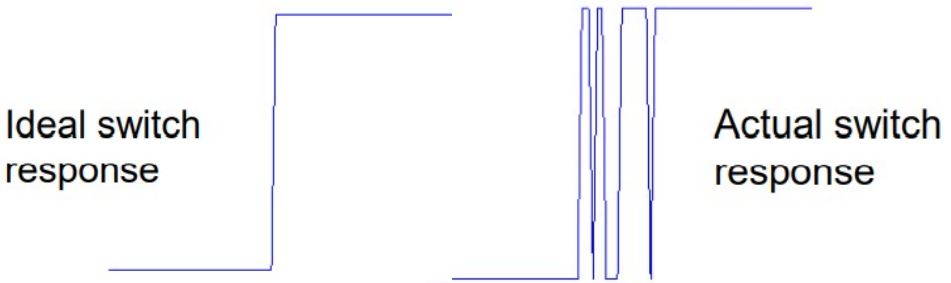
Import into Compiler

```
#include "mbed.h"

InterruptIn button(BUTTON1);          // Pusbutton input (PC_13)
DigitalOut led(LED1);                 // LED output (PA_5)

void button_pressed() {
    led = !led;                       // LED state changed at every button press
}

int main() {
    button.mode(PullUp);              // Enable internal pullup
    button.fall(&button_pressed);     // Attach function to falling edge
    while (true) {
        wait(0.2f);                   // Nothing to do. Just wait
    }
}
```

Lab04_button_interrupt projekt futtatás

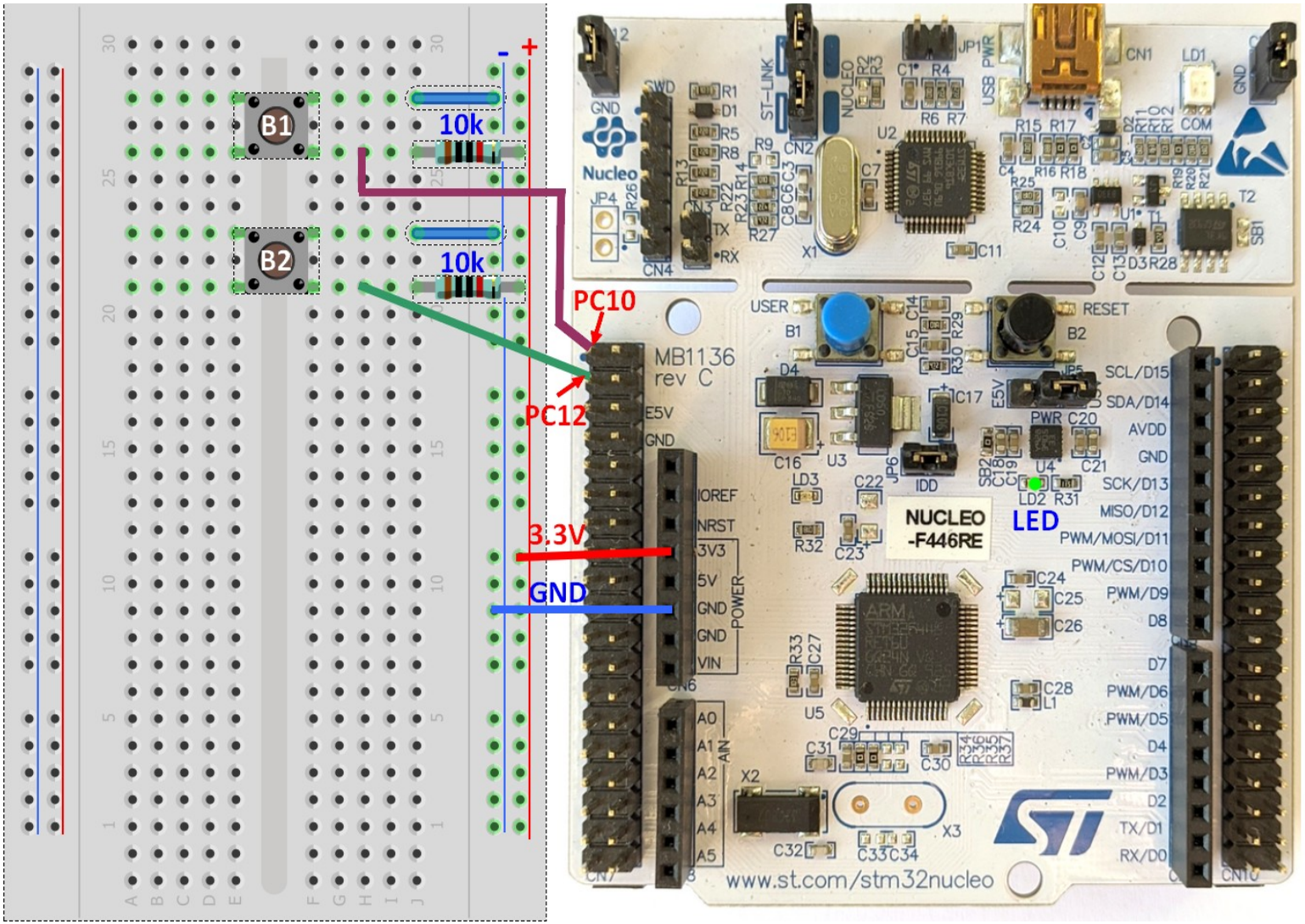
- A kék nyomógomb minden lenyomásakor az **LD2** LED állapotot vált (sajnos, néha többször is…)
- A nyomógomb „pergése” azt jelenti, hogy lenyomáskor, vagy felengedéskor egynél több állapotváltozás következik be a nyomógomb bemeneten





Lab04_b2int projekt

- A **NUCLEO-F446RE** kártya beépített **LD2** LED-jét (**PA5**) kapcsolgatjuk ki-be, két nyomógombbal, megszakításban
- A **B1** (PC_10) nyomógomb bekapcsolja
- A **B2** (PC_12) nyomógomb kikapcsolja
- A **PC_10** és **PC_12** bemenetek a **CN7** csatlakozó 1. és 3. tüskéjén érhetők el



Lab04_b2int/main.cpp

Import into Compiler

```
#include "mbed.h"

InterruptIn b_on(PC_10);        // Pusbutton input (PC_10)
InterruptIn b_off(PC_12);       // Pusbutton input (PC_12)
DigitalOut led(LED1);           // LED output (PA_5)

void b_on_pressed() {
    led = 1;                    // LED on
}

void b_off_pressed() {
    led = 0;                    // LED off
}

int main() {
    b_on.mode(PullUp);          // Enable internal pullup
    b_on.fall(&b_on_pressed);   // Attach function to falling edge
    b_off.mode(PullUp);         // Enable internal pullup
    b_ff.fall(&b_off_pressed); // Attach function to falling edge
    while (true) {
        wait(0.2f);             // Nothing to do. Just wait
    }
}
```

Lab04_button_bounce/main.cpp

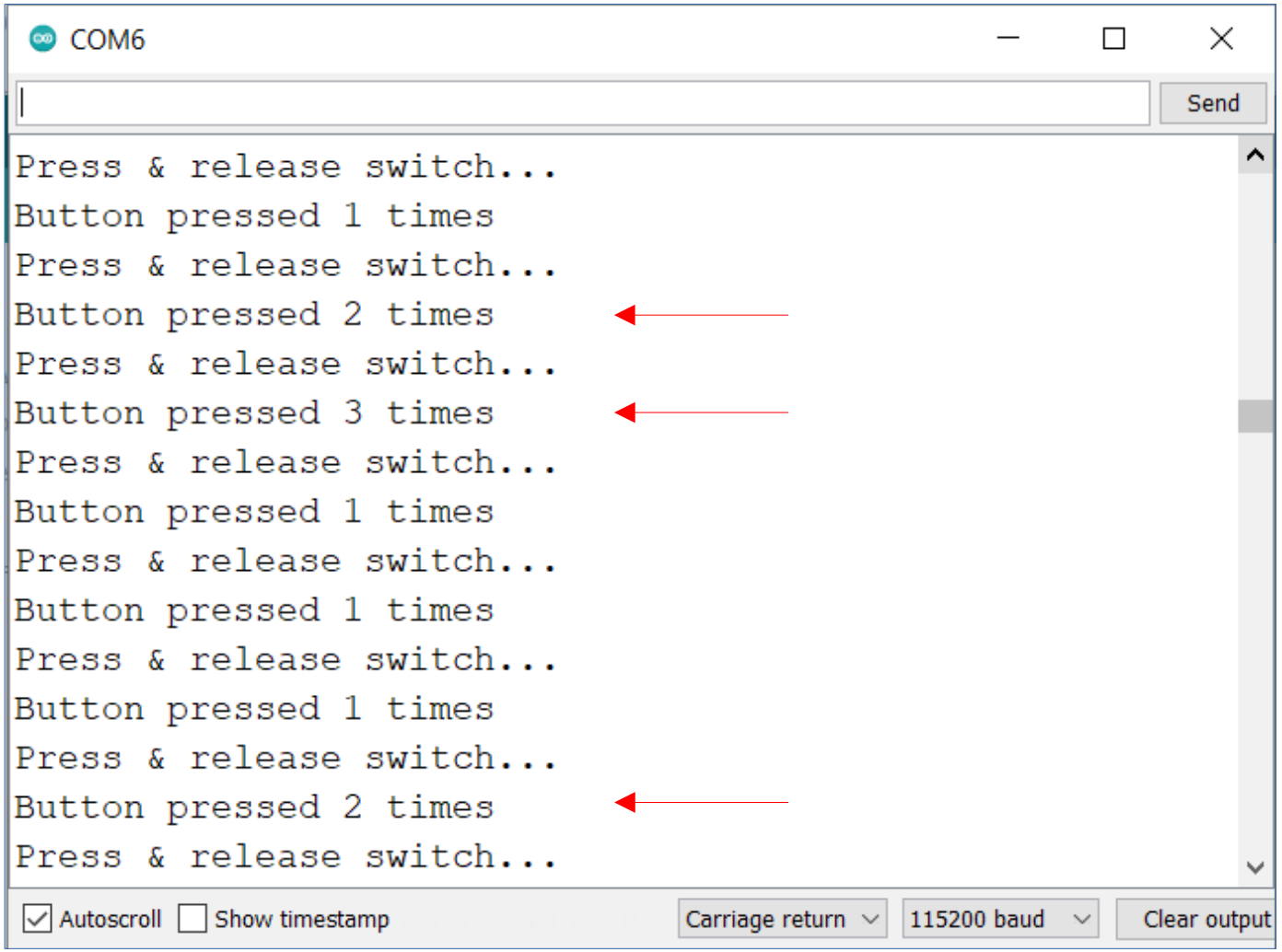
- Nyomógomb pergését vizsgáljuk, az eredményt soros porton kiíratjuk

Import into Compiler

```
#include "mbed.h"
DigitalIn mybutton(BUTTON1,PullUp); // Pushbutton input (PC_13)
InterruptIn button(BUTTON1);        // Pusbutton interrupt
Serial pc(USBTX,USBRX);             // UART0 via OpenSDA
volatile uint16_t counts;           // counter variable

void button_pressed() {
    counts++;                       // counts button presses
}

int main() {
    pc.baud(115200);                // set baudrate for UART
    button.mode(PullUp);            // Enable internal pullup
    button.fall(&button_pressed);   // Attach function to falling edge
    while (true) {
        counts = 0;                 // Clear counter
        pc.printf("Press & release switch... \r\n");
        while (mybutton);           // Wait for button press
        wait_ms(20);                // Debounce delay
        while (!mybutton);          // Wait for button release
        wait_ms(20);                // Debounce delay
        pc.printf("Button pressed %d times\r\n",counts);
    }
}
```

Lab04_button_bounce futási eredmény

- Az alábbi ábrán egy futási eredmény látható. Minden egynél nagyobb szám pergést jelez

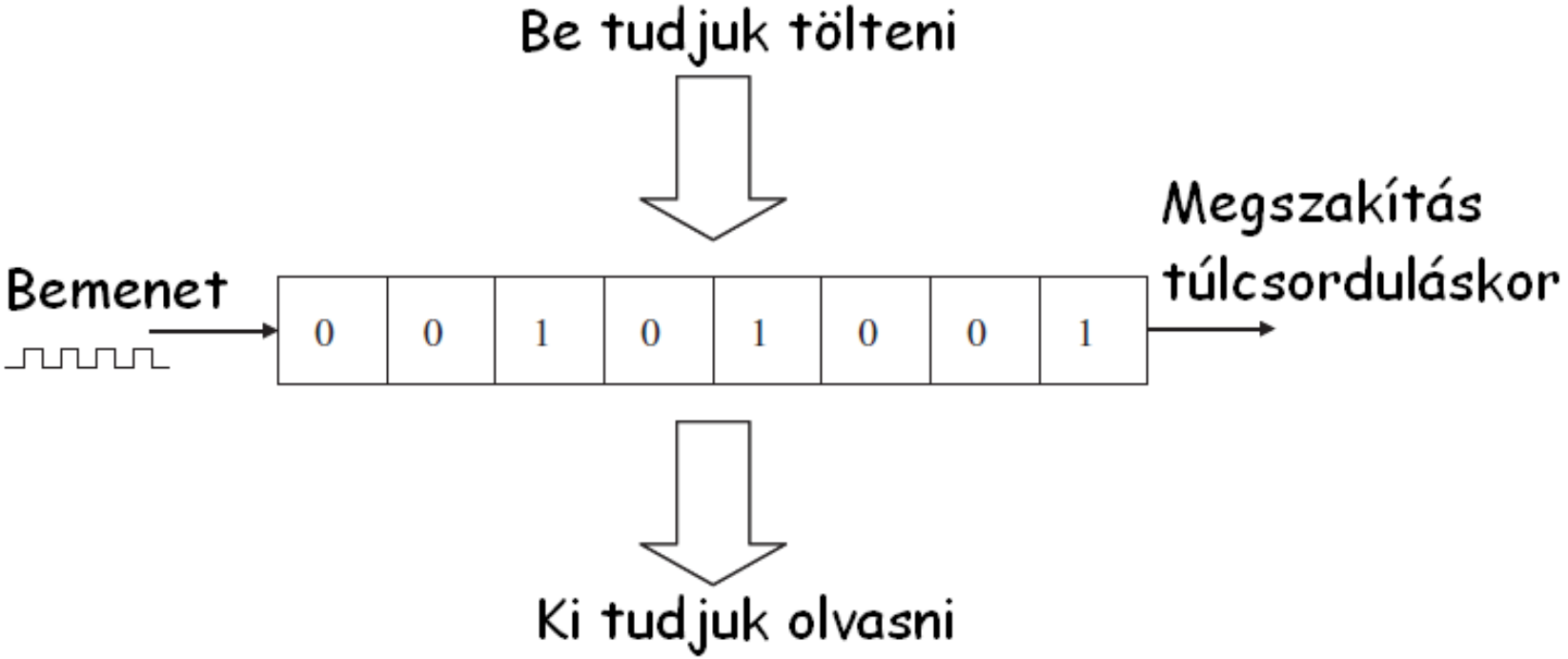


Az időzítés szerepe beágyazott rendszereknél

- A beágyazott rendszerek időben kell, hogy reagáljanak a bekövetkező eseményekre, vagy előre megadott ütemezés szerint kel, hogy végezzék feladataikat (pl. adatgyűjtés/adatmegjelenítés azonos időközönként), ezért képeseknek kell lenniük az alábbiakra:
  - Időtartam mérése
  - Idő-alapú események generálása, ami lehet egyszeri vagy ismétlődő
  - Elfogadható időn belül reagálni az előre nem meghatározható időben bekövetkező eseményekre
- A **számláló** (counter) olyan digitális áramkör, amellyel feszültségimpulzusokat tudunk leszámlálni. Ha az impulzusok nem külső forrásból származnak, hanem állandó frekvenciájú jelet vezetünk a számláló bemenetére, akkor pedig **időzítőről** (timer) beszélünk, amely lehetővé teszi számunkra az idő mérését, illetve a feladatok ütemezését.

Számlálók

- **Számlálókat** gyakran használnak a digitális áramkörökben. Ezek többnyire sorbakötött bistabil billenőáramkörök, melyek mindegyike egy-egy bitnyi információt tárol.
- Egy n-bites számláló 2^n-1 állapotot képes felvenni. Például az alábbi ábrán látható 8 bites számláló 0000 0000_b-tól 1111 1111_b-ig, azaz 0-tól 255-ig számol



A Timer objektumosztály

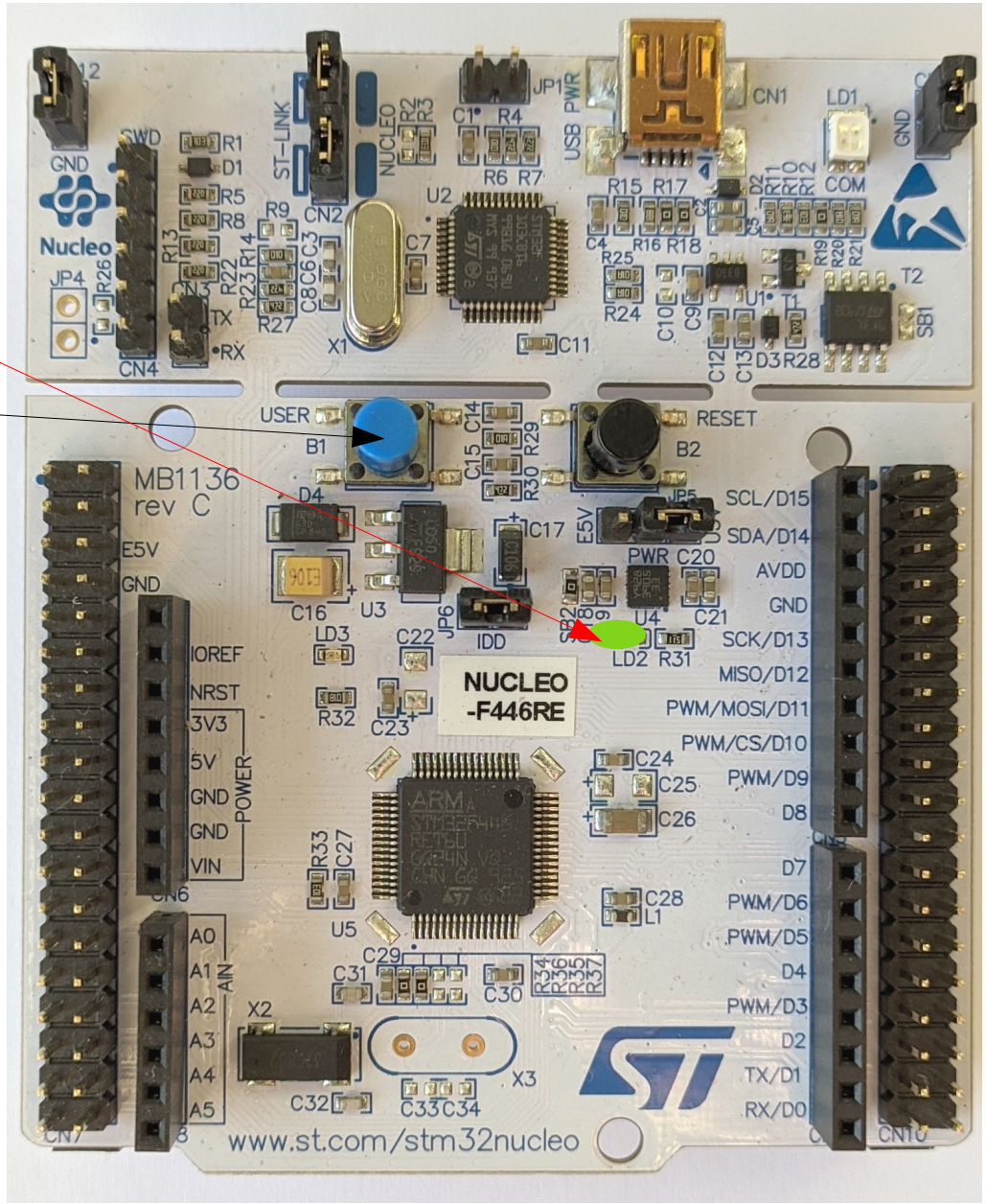
- **Timer** objektumok segítségével absztrakt időmérőket hozhatunk létre, amellyel pl. a két esemény között eltelt i mérhetjük meg, mikroszekundumos felbontással
- A **Timer** objektumosztály tagfüggvényeit az alábbi táblázatban foglaltuk össze:

Függvény	Használat
Timer név	Létrehoz egy *"név"* nevű absztrakt **Timer** objektumot.
start()	Elindítja az időmérést
stop()	Megállítja az időmérést
reset()	Nullázza az időmérőt
read()	Másodpercekben adja meg az eltelt időt.
read_ms()	Ezredmásodpercekben adja meg az eltelt időt.
read_us()	Milliomod másodpercekben adja meg az eltelt időt.

- **Megjegyzés:** A **read_ms()** és **read_us()** tagfüggvények visszatérési értéke ***int*** típusú, ezért az absztrakt időzítők $2^{31}-1$ µs-nál (kb. 2147 s) hosszabb időtartam mérésére nem alkalmasak.

Reakcióidő mérése Timer használatával

- A Lab04_reaction_time nevű programban a **Timer** absztrakt időzítőt stopperként használjuk, s a **LED1** beépített LED felgyulladása és a válaszként lenyomott **BUTTON1** nyomógomb bekapcsolása között eltelt időt mérjük vele (a felhasználó reakcióideje).
- Az eredményt az alapértelmezett soros porton keresztül íratjuk ki, amelynek sebességét 115 200 baudra állítjuk be



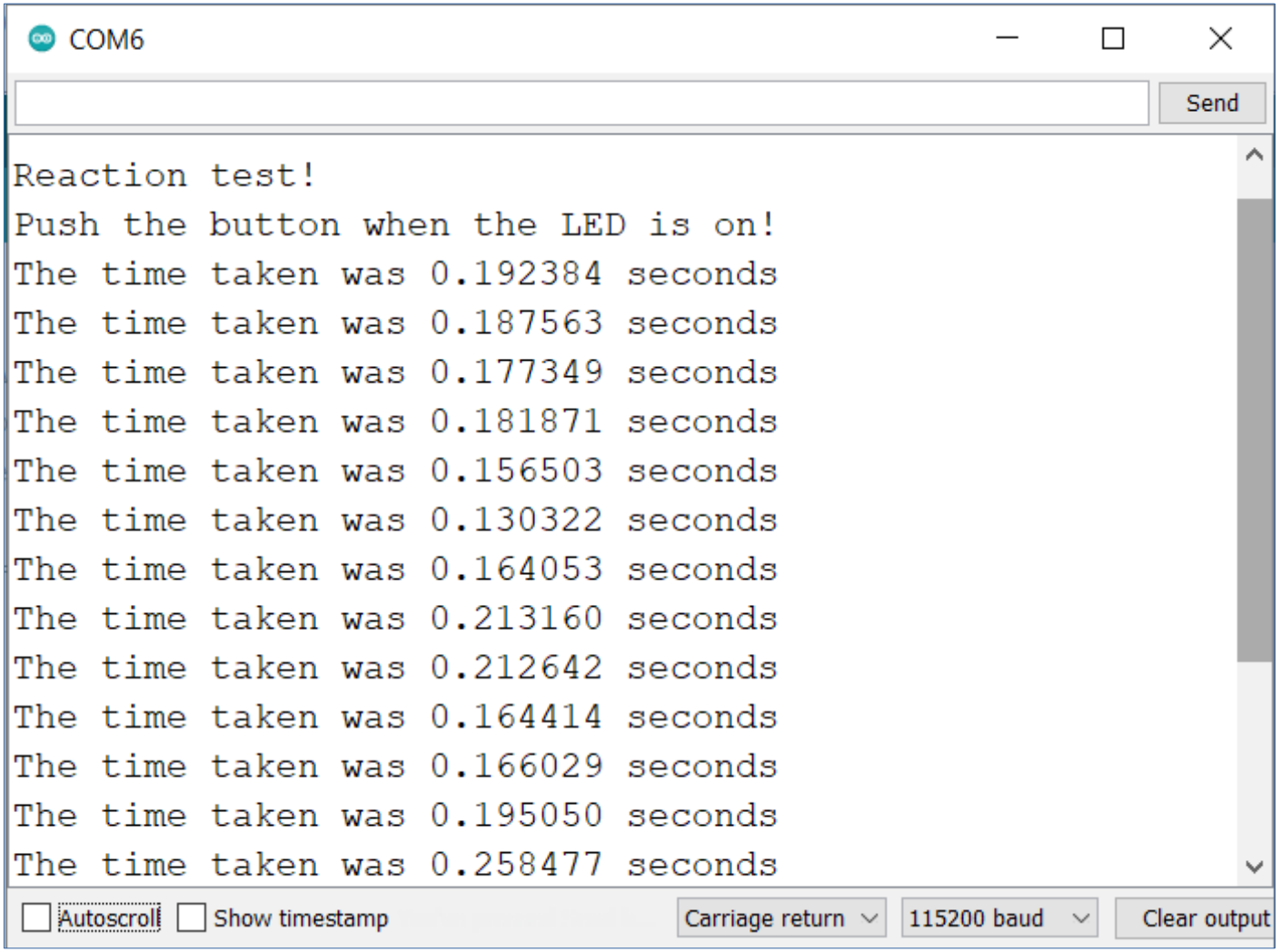
Lab04_reaction_time/main.cpp

Import into Compiler

```
#include "mbed.h"
Serial pc(USBTX,USBRX);
DigitalOut myled(LED1);                 //define LED
DigitalIn mybutton(BUTTON1,PullUp); //define pushbutton input with pull-up
Timer t;                                //define timer
int x;                                  //x for random number storage

int main() {
    pc.baud(115200);                    // Set baudrate for UART
    myled = 0;                          //LED is initially off
    pc.printf("\r\nReaction test!\r\n");
    pc.printf("Push the button when the LED is on!\r\n");
    while(1) {
        t.reset();                      //reset the timer
        while(!mybutton);               //wait for button release
        x=rand()%2000;                  //generate random number 0-2000
        wait_ms(1000+x);                //wait random time between 1 and 3 seconds
        t.start();                      //start the timer
        myled=1;                        //LED on
        while(mybutton);                //wait for press
        t.stop();                       //stop the timer
        myled=0;                        //LED off
        wait_ms(20);                    //debounce delay
        printf("The time taken was %f seconds\r\n", t.read());
    }
}
```

Lab04_reaction_time futási eredmény

- A képen egy programfutás eredménye látható



A TimeOut objektumosztály

- A **Timeout** objektumosztály segítségével ébresztő-órához hasonló időmérőt hozhatunk létre, amellyel adott idő elteltével visszahívási (callback) függvényt aktiválhatunk, a főprogram futását megszakítva
- **Timeout** tagfüggvényei:

Függvény	Használat
Timeout név	Létrehoz egy *"név"* nevű **Timeout** objektumot.
attach(*fptr,time)	A **Timeout** objektumhoz rendel egy visszahívási függvényt és megadja a visszahívási időt. Az időt lebegőpontos formátumban, másodpercben adjuk meg.
attach_us(*fptr,time)	A **Timeout** objektumhoz rendel egy visszahívási függvényt és megadja a visszahívási időt. Az időt *unsigned int* formátumban, mikroszekundumokban adjuk meg.
detach()	Megszünteti a visszahívási függvény hozzárendelését az időzítőhöz

- **Megjegyzés:** a visszahívási függvény programmegszakítási szinten aktiválódik, ügyeljünk rá, hogy ne tartalmazzon blokkoló várakozást, terjedelmes adatfeldolgozást, **printf** kiíratást, memória allokációt.

Lab04_Timeout_demo/main.cpp

BUTTON1 minden lenyomásakor két másodperces LED villogtatás indul

```
#include "mbed.h"

Timeout wecker;                                 //create a Timeout, and name it "wecker"
DigitalIn button(BUTTON1);
DigitalOut led(LED1);                           //blinks with main while(1) loop
volatile int state = 0;

void noblink() {                                // called at the end of the Timeout

    state = 0;                                  // Stop blinking
}

int main() {
    while(1) {
        if(button==0) {
            wecker.attach(&noblink,2.0); //attach noblink function to Response
                                                //Timeout, to occur after 2 seconds
            state = 1;                          // Start binking LED
        }
        if(state) {                             // blink the led if state = 1
            led=!led;
            wait(0.2);
        } else {
            led = 0;                            // Switch off LED id state = 0
        }
    }
}
```

Import into Compiler

A Ticker objektumosztály

- A **Ticker** objektum segítségével periodikusan, adott időközönként aktiválhatjuk az objektumhoz rendelt visszahívási (callback) függvényt
- A visszahívási függvény ne tartalmazzon blokkoló várakozást, terjedelmes adatfeldolgozást, **printf** kiíratást, vagy memória allokációt!
- **Ticker** tagfüggvényei:

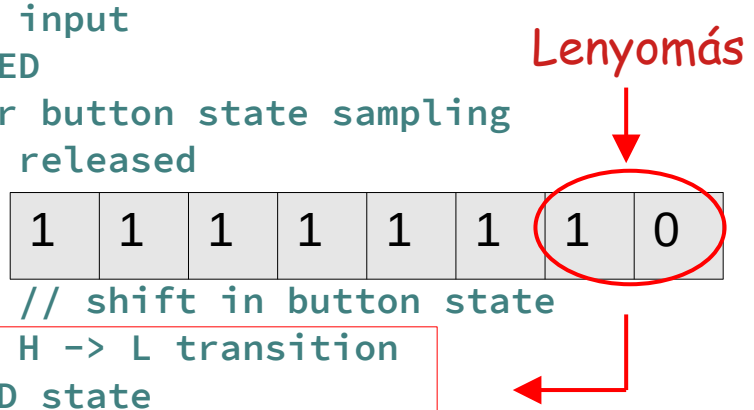
Függvény	Használat
Ticker név	Létrehoz egy *"név"* nevű **Ticker** objektumot.
attach(*fptr,time)	A **Ticker** objektumhoz rendel egy visszahívási függvényt és megadja a visszahívási időt. Az időt lebegőpontos formátumban, másodpercben adjuk meg.
attach_us(*fptr,time)	A **Ticker** objektumhoz rendel egy visszahívási függvényt és megadja a visszahívási időt. Az időt *unsigned int* formátumban, mikroszekundumokban adjuk meg.
detach()	Megszünteti a visszahívási függvény hozzárendelését az időzítőhöz

Lab04_button_debouncer/main.cpp

- Az alábbi programban pergésmentesítetté tesszük a nyomógombos LED kapcsolgatást (a pergésmentesítés itt úgy történik, hogy csak 20 ms-onként vizsgáljuk a nyomógomb állapotát)

Import into Compiler

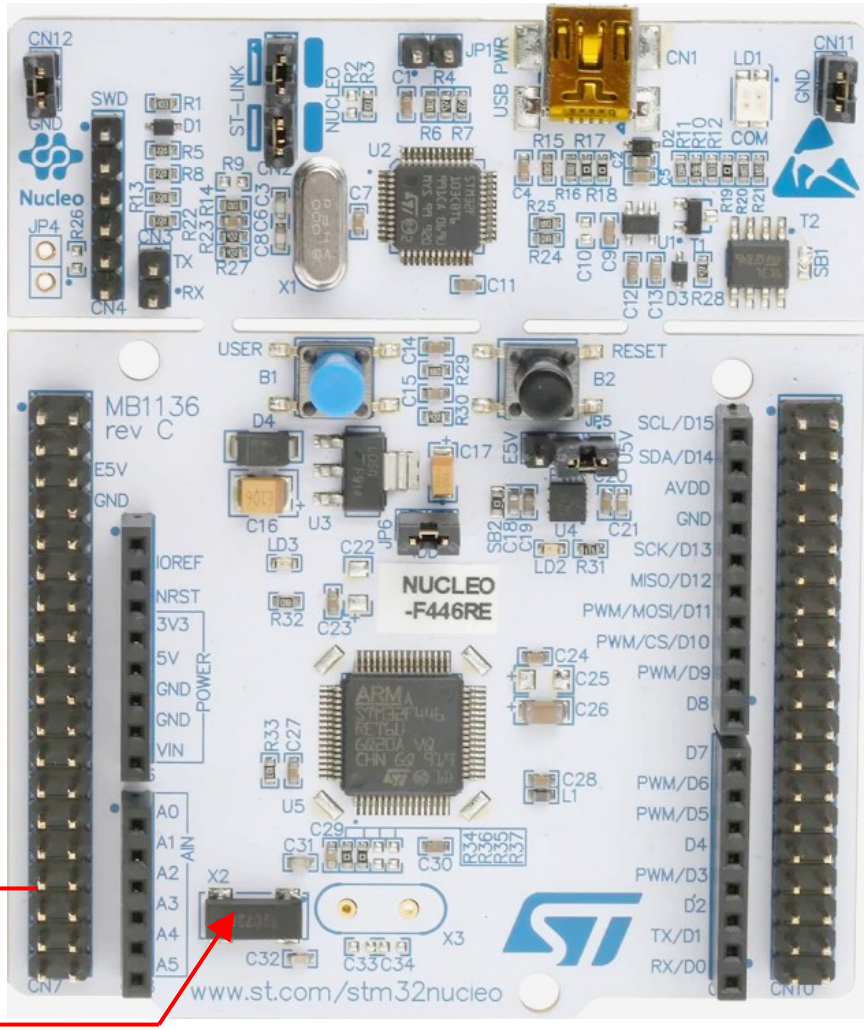
```
#include "mbed.h"

DigitalIn button(BUTTON1,PullUp);        // Pusbutton input
DigitalOut led(LED1);                    // Builtin LED
Ticker sampler;                          // Ticker for button state sampling
volatile uint8_t button_state = 1;       // Initially released

void button_check() {
    button_state = (button_state<<1) | (button & 1); // shift in button state
    if((button_state & 3)==2) {          // Check for H -> L transition
        led = !led;                      // Switch LED state
    }
}

int main() {
    led = 0;                             // LED off
    sampler.attach(&button_check,0.02); // sample button state in each 20 ms
    while (true) {
        wait(1);                         // do nothing
    }
}
```



A LED kapcsolgatása a főprogramban is lehetne

Real Time Clock (RTC)

- Az **STM32F446RE** mikrovezérlő valós idejű órával (RTC) is rendelkezik, amelynek saját, kisfrekvenciás órajel forrása van (32 kHz)
- Az **RTC** akkor is működhet, amikor a mikrovezérlő energiatakarékos, vagy kikapcsolt állapotban van
- Az **RTC** a **VBAT** kivezetésen keresztül kap tápellátást, ami alapértelmezetten a **VDD**-re van kötve (**SB45 átkötés**)
- Az átkötés eltávolítása után a **CN7** csatlakozó 33. tüskéjén vihetjük be VBAT tápellátását (elem / akkumulátor)



Az RTC használat támogatása

- Az **RTC** kezeléséhez az mbed API nem tartalmaz C++ objektumosztályt. A **Time** programkönyvtár a standard C library-ból örökölt C függvényekből áll, amelyekhez a céláramkörök **RTC** hardverének kezelésére (idő beírása, ill. kiolvasása) szolgáló C függvények társulnak (megszakításban ne használjuk ezeket!)
- Az aktuális UTC idő az **1970. január 1.** óta eltelt másodperceket jelenti (UNIX timestamp), a tárolása **time_t** "timestamp" (időbélyeg) típusú változókban történik, ami **uint32_t** típust jelent
- Az idő kiíratáshoz, emberi léptékű kezeléshez a **tm** sruktúra típust használjuk, amelyben külön-külön változókban szerepel az év, a hónap, a nap, az óra, a perc és a másodpercek.
- **Year 2038 problem:** azoknál a rendszereknél, ahol **int32_t** típusú változóban tárolják/kezelik az UTC időt, **2038. január 19**-én 03:14:07-kor túlcsordulás következik be

Az RTC használat támogatása

- A Time programkönyvtár függvényeit az alábbi táblázat mutatja

Függvény	Használat
time(NULL**)**	Az aktuális idő kiolvasása. A visszatérési érték timestamp típusú
set_time(time_t t**)**	Az aktuális idő beállítása
mktime(struct tm * t**)**	Egy tm struktúra átalakítása timestamp formátumra. A paraméter tm struktúrára mutató pointer, a vissztérési érték timestamp típus.
localtime(time_t * t**)**	Egy timestamp átalakítása tm struktúrába (a visszatérési érték pointer!)
ctime(time_t * t**)**	Egy timestamp olvasható szöveggé történő konverziója. Pl. "Wed Oct 28 11:35:37 2009\n"
sfrtime(char * buffer, size_t max, char * format, struct tm * t**)**	Egy tm struktúra testreszabható formátumú szöveggé alakítása

- Példa **set_time()** és **time()** használatára

```
#include "mbed.h"
int main() {
    set_time(1256729737);  // Set RTC time to Wed, 28 Oct 2009 11:35:37
    time_t seconds = time(NULL);
    printf("Time as seconds since January 1, 1970 = %d\n", seconds);
. . .   }
```

Az RTC használat támogatása

- **mktime** - Egy **tm** struktúrát UNIX időbélyeggé alakít, emellett a **tm** struktúra **tm_wday** (a hét napja) és **tm_yday** (az év napja) elemeit is aktualizálja (**Megjegyzés:** az évszámokat 1900-hoz képest kell megadni, s a hónapok számozása 0-val kezdődik)
- Példa **mktime**() használatára:

```
#include "mbed.h"

int main() {
    // setup time structure for Wed, 28 Oct 2009 11:35:37
    struct tm t;
    t.tm_sec = 37;     // 0-59
    t.tm_min = 35;     // 0-59
    t.tm_hour = 11;    // 0-23
    t.tm_mday = 28;    // 1-31
    t.tm_mon = 9;      // 0-11
    t.tm_year = 109;   // year since 1900
    // convert to timestamp and display (1256729737)
    time_t seconds = mktime(&t);
    printf("Time as seconds since January 1, 1970 = %d\n", seconds);
. . . }
```

Az RTC használat támogatása

- **localtime** – egy **time_t** típusú időbélyeg értékét egy (statikusan allokált) **tm** stuktúrába konvertálja

```
#include "mbed.h"

int main() {
    time_t seconds = 1256729737;
    struct tm *t = localtime(&seconds);
. . . }
```

- **ctime**() – egy időbélyeg értékét olvasható szöveggé alakít. Az eredmény ilyen formátumú lesz: “Wed Oct 28 11:35:37 2009\n”.

```
#include "mbed.h"

int main() {
    time_t seconds = time(NULL);
    printf("Time as a string = %s", ctime(&seconds));
  . . .
}
```

Az RTC használat támogatása

- **strftime** - egy **tm** struktúra szöveggé alakítása

```
#include "mbed.h"
int main() {
    time_t seconds = time(NULL); char buffer[32];
    strftime(buffer, 32, "%I:%M %p\n", localtime(&seconds));
    printf("Time as a formatted string = %s", buffer);
}
```

Az **sfrtime()** függvényben használható formátumelemek táblázata

%S	Second (másodperc 00-59)
%M	Minute (perc 00-59)
%H	Hour (óra 0-23)
%d	Day (nap 01-31)
%m	Month (hónap 01-12)
%Y/%y	Year (év 2009, illetve 09 formátumban)
%A/%a	Weekday name (hét napjának neve Monday, illetve Mon formátumban)
%B/%b	Month name (hónap neve January, illeve Jan formátumban)
%I	12 Hour format (12 órás kijelzés 00-12)
%p	AM vagy PM
%X	Time (idő 14:55:02 formátumban)
%x	Date (dátum 02/05/09 formában February 5, 2009 esetén)

Lab04_rtc_time/main.cpp 3/1.

Import into Compiler

Ebben a példában a soros porton keresztül beállíthatjuk és 5 másodpercenként alapértelmezett formátumban kiíratjuk az időt

```
#include "mbed.h"
Serial pc(USBTX,USBRX);
Ticker myticker;
time_t mytime;
volatile uint8_t myflag = 0;
#define DATE_20200222_222222    1582377742  // 2020/2/22 22:22:22

void setflag(void) { myflag = 1; }          // Ticker Callback function

int main() { pc.baud(115200);
    time_t seconds;
    seconds = time(NULL);
    if (seconds < DATE_20200222_222222) {
      set_time(DATE_20200222_222222);       // Set RTC only once
    }
    myticker.attach(&setflag,5);            // Set flag in each 5th second
    while(1) {
        if(pc.readable()) {
            ProcessSerialCommand();         // Process command if any...
        }
        if(myflag) {
            mytime = time(NULL);            // Print time in default format
            pc.printf("RTC time: %s\r\n",ctime(&mytime));
            myflag = 0;
        }
    }
}
```

Lab04_rtc_time/main.cpp 3/2.

Az RTC beállításához egy **TYYMMDDhhmmss** parancsot kell kiküldenünk a terminálból

T - a kötelező parancsbetű (T mint Time)
YY - az évszám utolsó két jegye (2016-ban ez 16)
MM - a hónap sorszáma (Január esetében 01)
DD - a nap (01 - 31 közötti szám)
hh - az óra (00 - 23 közötti szám)
mm - a perc (00 - 59 közötti szám)
ss - a másodperc (00 - 59 közötti szám)

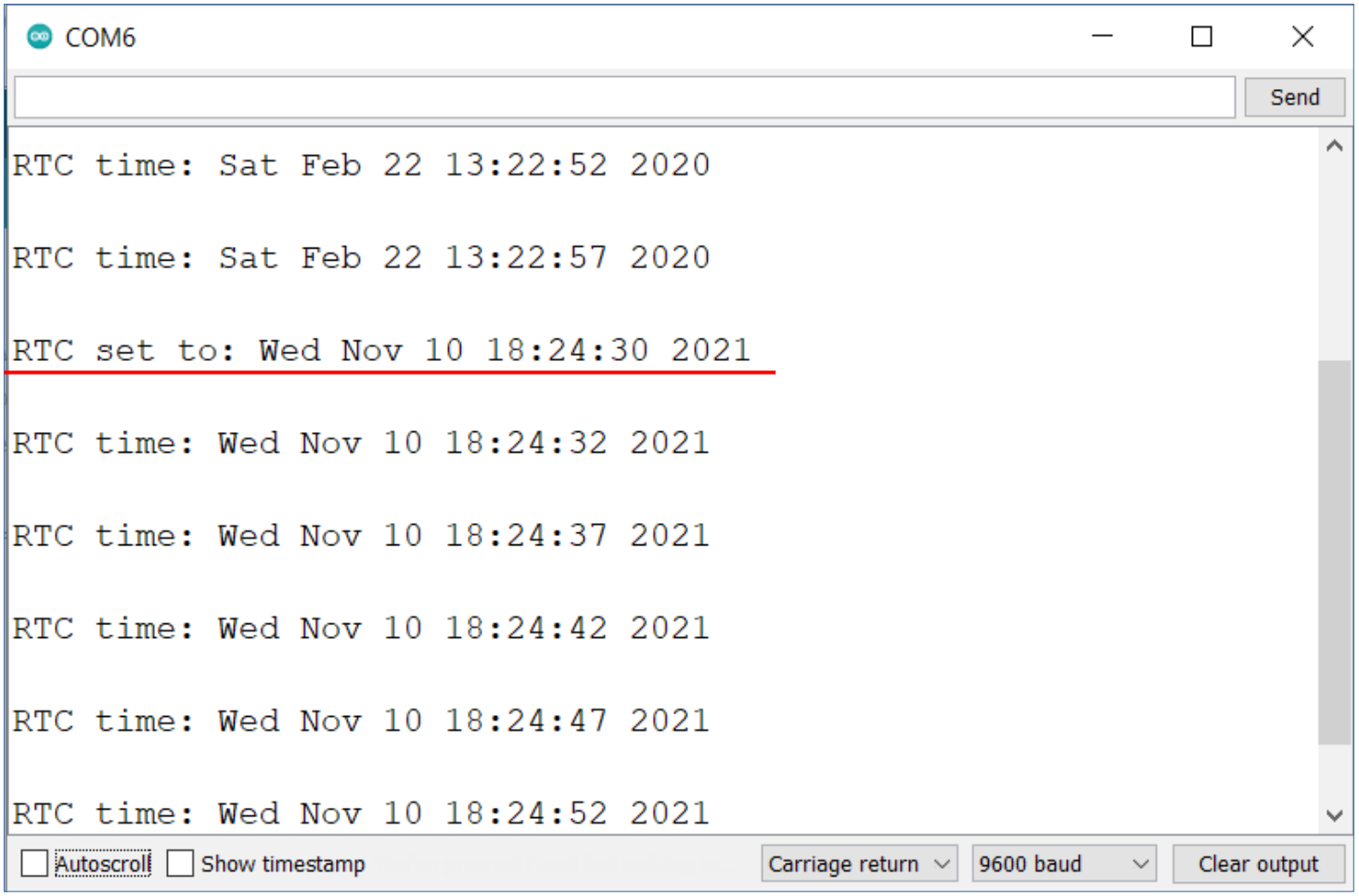
```
void processSerialCommand() {
    char c = pc.getc();
    switch(c) {
        case 'T':
            // Command to set RTC time
            // Command format: TYYMMDDHHMMSS
 //Example: 2012 Oct 21 1:23pm is T121021132300
            struct tm tme;
            time_t newTime;

            // Parse incomming 12 ASCII charaters into time_t
            // no error checking for numeric values in YYMDDHHMMSS fields, be careful!
            c = pc.getc();
            tme.tm_year = c - '0';
            c = pc.getc();
            tme.tm_year = 10*tme.tm_year;
            tme.tm_year += c-'0';
            tme.tm_year += 100;             //Years are counted from 1900!
            c = pc.getc();
            tme.tm_mon = c - '0';
            c = pc.getc();
            tme.tm_mon = 10*tme.tm_mon;
            tme.tm_mon += c-'0'-1;          //corrected by -1 due to a stupid error
            c = pc.getc();
            tme.tm_mday = c - '0';
```

Lab04_rtc_time/main.cpp 3/3.

```
            c = pc.getc();
            tme.tm_mday = 10*tme.tm_mday;
            tme.tm_mday += c-'0';
            c = pc.getc();
            tme.tm_hour = c - '0';
            c = pc.getc();
            tme.tm_hour = 10*tme.tm_hour;
            tme.tm_hour += c-'0';
            c = pc.getc();
            tme.tm_min = c - '0';
            c = pc.getc();
            tme.tm_min = 10*tme.tm_min;
            tme.tm_min += c-'0';
            c = pc.getc();
            tme.tm_sec = c - '0';
            c = pc.getc();
            tme.tm_sec = 10*tme.tm_sec;
            tme.tm_sec += c-'0';
            newTime = mktime(&tme);
            set_time(newTime);
            pc.printf("RTC set to: %s\r\n",ctime(&newTime));
    }
    while(pc.readable()) {
        pc.getc();    // clear serial buffer
    }
}
```

Lab04_rtc_time

- A program egy futási eredménye:



WakeUp – Mélyalvás és felébresztés

STM32 üzemmódok

- Az **STM32F446RE** mikrovezérlő az alábbi üzemmódokkal rendelkezik:

Run mode	A CPU és minden bekapcsolt periféria működik
Sleep mode	A CPU nem kap órajelet, a perifériák továbbra is működnek Ébresztés megszakítási jellel, vagy wakeup eseménnyel
Stop mode	A CPU és a perifériák nem kapnak órajelet, a RAM megőrzi tartalmát Ébresztés EXTI megszakítással, vagy wakeup eseménnyel
Standby mode	Az RTC kivételével minden lekapcsol, csak az RTC backup domain őrzi meg tartalmát (deepslep, azaz mélyalvás mód). Ébresztés WKUP bemenet felfutó jelre, RTC riasztásra, NRST jelre

- A legkisebb fogyasztású Standby (vagy deepsleep) módból csak újraindulással tudunk „felébredni”, mert a RAM és a regiszterek is elvesztik tartalmukat
- Kísérleteinkhez **Kenji Arai** **WakeUp_STM32** programkönyvtárát fogjuk felhasználni, amely az **RTC** modul segítségével végzi az ébresztést „mélyalvásból”

Lab04_WakeUP_STM32

- Az alábbi egyszerű programban néhányszor felvillantjuk a beépített **LED1**-et. Majd 30 másodpercre mélyalvásba merülünk
- A program minden ébredés után elölről indul

```
#include "mbed.h"
#include "WakeUp_STM32.h"
//See at: os.mbed.com/users/kenjiArai/code/WakeUp_STM32/
DigitalOut myled(LED1);

int main() {
    uint32_t loop_count = 1;
    while(true) {
        // In run mode Imax = 44 mA
        myled = !myled;
        wait(0.25);
        if (++loop_count > 4) {
            WakeUp::standby_then_reset(30000);  // 30sec
            // In standby mode I = 3.7 uA
            while(true) {;} // never executing this line
        }
    }
}
```

Import into Compiler

Az ehhez a programhoz is használt, általunk módosított **WakeUp_STM32** programkönyvtárat az alábbi gombra kattintva adhatjuk hozzá saját projektünkhöz

Import library into Compiler

Lab04_Check_Standby_Os2

- Kenji Arai a WakeUp_STM32 programkönyvtárhoz egy egyszerű **mintaalkalmazást** is készített, amelyben bemutatja, hogy hogyan végezhetünk ciklikus adatgyűjtést, a szünetekben mélyalvásos energiatakarékos módot használva
- A program csak **Mbed-os5.15.1** vagy **mbed 2.0.165** alatt fut
- A program 10 másodperces mélyalvás után ébred és a **BUTTON1** gomb lenyomásáig (de legfeljebb 20 alkalommal) másodpercenként adatgyűjtést végez, az eredményt időbélyeggel kiírja, majd elalszik
- Az idő YY HH MM hh mm ss formátumú adatbeírással állítható be
- A programot itt nem ismertetjük részletesen, csak a legfontosabb részeket mutatjuk be

Lab04_Check_Standby_Os2/main.cpp (részlet)

Import into Compiler

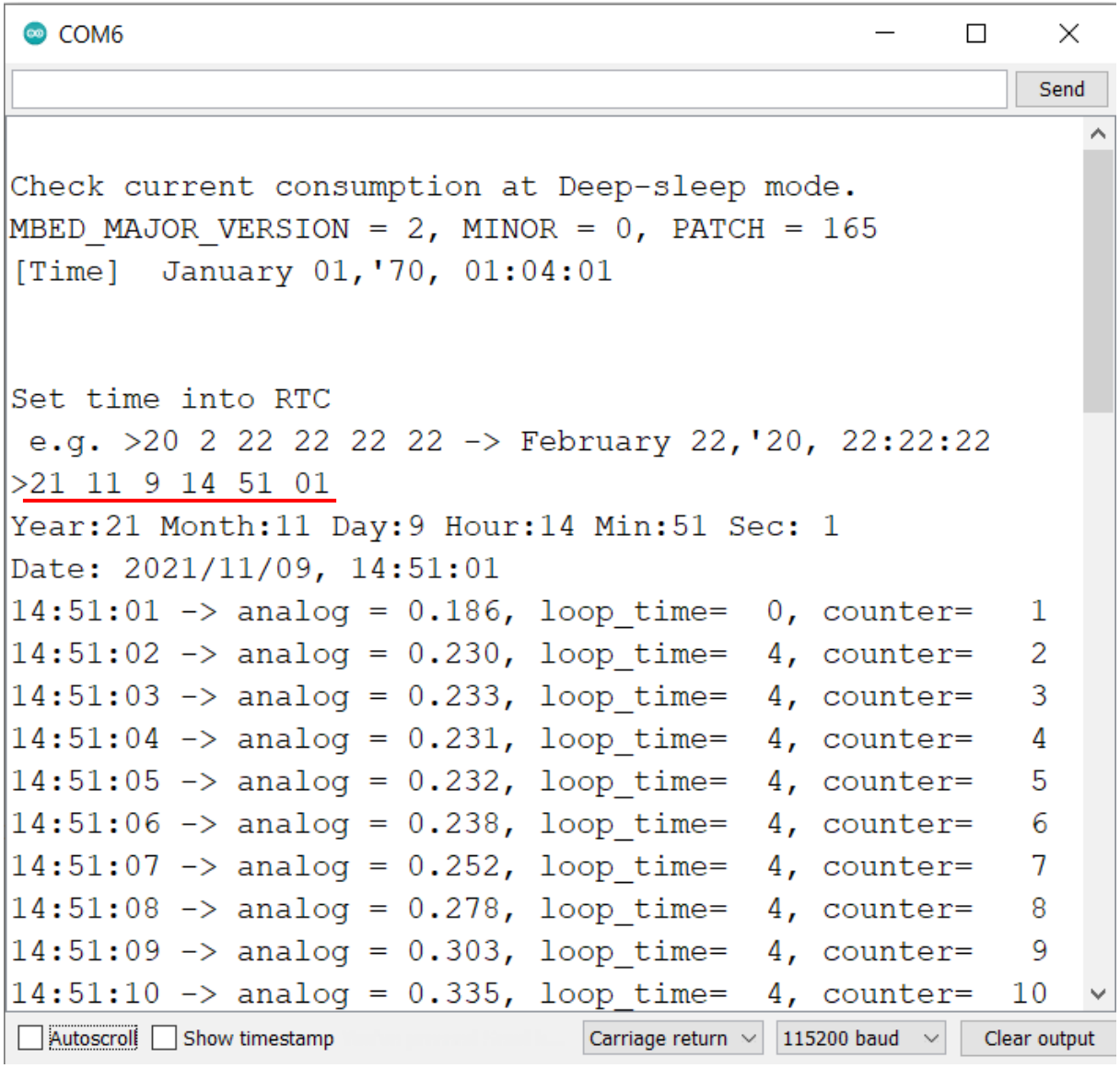
```
#include "mbed.h"
#include "WakeUp_STM32.h"
#define DATE_20200222_222222    1582377742      // 2020/2/22 22:22:22
DigitalIn   my_sw(USER_BUTTON);
DigitalOut  myled(LED1,1);
Serial      pc(USBTX, USBRX);
AnalogIn    a_in(A0);
Timer       t;

int main() {
    time_t seconds;
    char buf[64];
    uint32_t t_pass = 0;
    uint32_t loop_count = 1;
    float ain;
    printf("\r\nCheck current consumption at Deep-sleep mode.\r\n");
    print_revision();
    seconds = time(NULL);                   // RTC kiolvasás
    if (seconds < DATE_20200222_222222) {
        strftime(buf, 50, " %B %d,'%y, %H:%M:%S\r\n", localtime(&seconds));
        pc.printf("[Time] %s\r\n", buf);
        time_enter_mode();                  // RTC adatbekérés, ha még nem volt
    }
```

Lab04_Check_Standby_Os2/main.cpp (részlet)

```
    while (my_sw == 0) {;}
    WAIT_MS(10);
    while (true) {
        t.reset(); t.start();
        if ((my_sw == 0) || (loop_count > 20)) {
            DigitalIn dmy0(LED1);
            DigitalIn dmy1(USBTX);
            DigitalIn dmy2(USBRX);
            WakeUp::standby_then_reset(10000); //--- I = 3.4 uA here

            while(true) {;}                    // never executing this
        }
        ain = a_in.read();                     //--- Imax = 50.4 mA here
        myled = !myled;
        seconds = time(NULL);
        strftime(buf, 50, "%H:%M:%S -> ", localtime(&seconds));
        pc.printf("%s", buf);
        pc.printf("analog = %4.3f, loop_time=%3d, counter=%4d\r\n",
                  ain, t_pass, loop_count++);
        t_pass = t.read_ms();
        WAIT_MS(1000 - t_pass);
    }
}
```

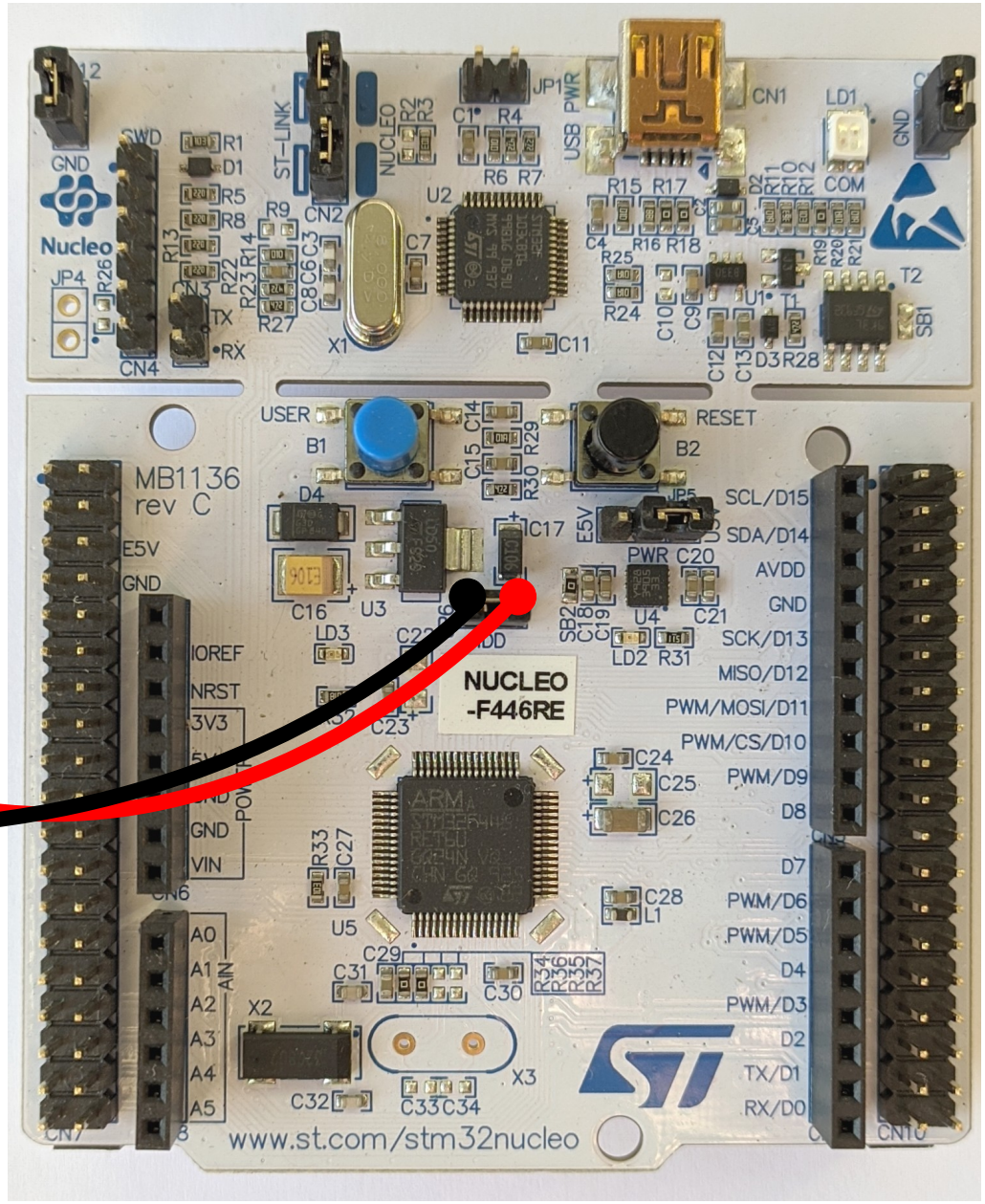
Lab04_Check_Standby_Os2

- A program egy futási eredménye az ábrán látható



A mikrovezérlő áramfelvételének mérése

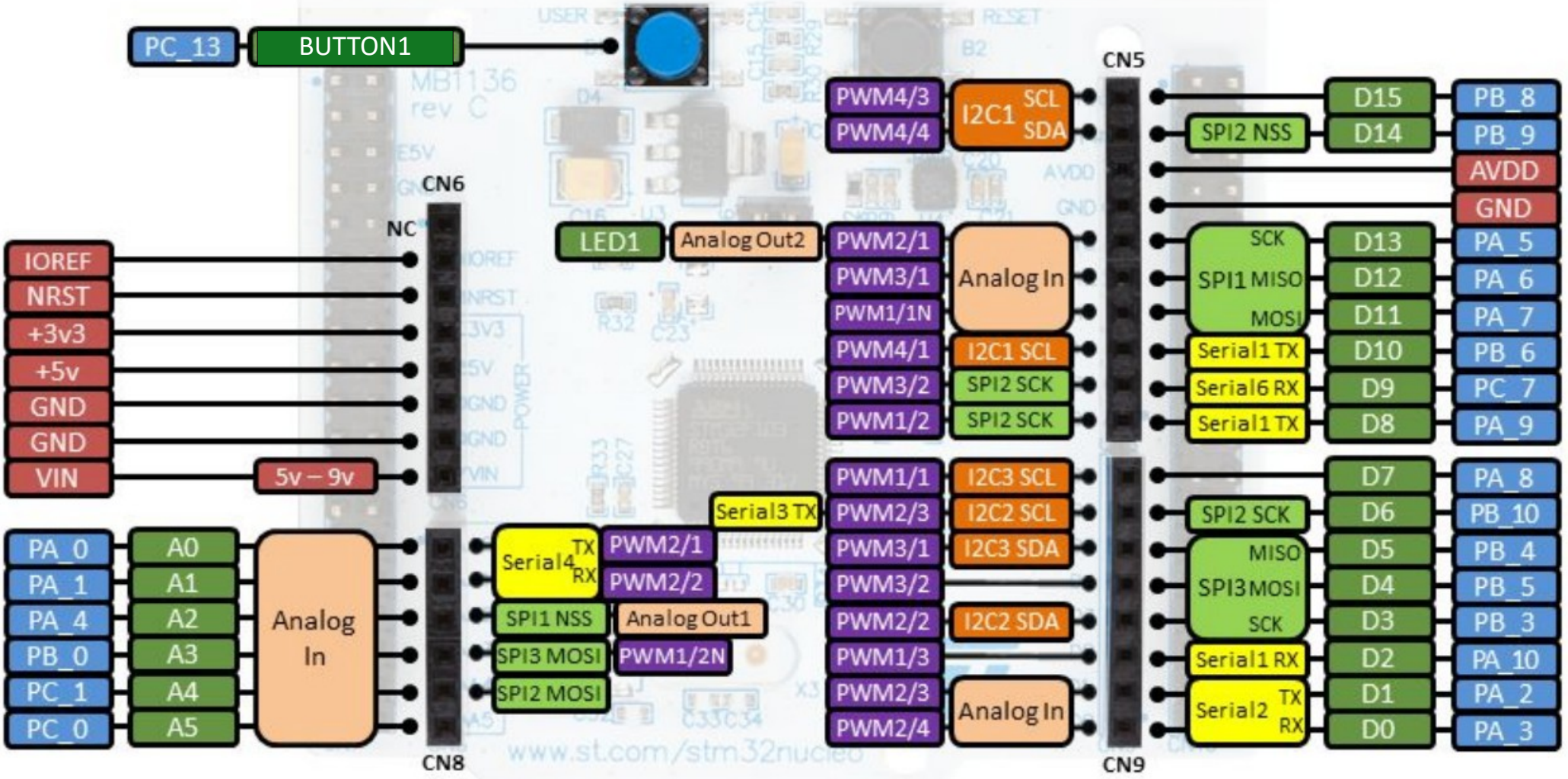
- A mikrovezérlő áramfelvételét az **IDD** átkötés helyén mérhetjük meg



- **Run** mód: 40 – 50 mA
- **Standby** mód: 3.4 µA

Arduino kompatibilis kivezetések

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



Morpho kivezetések

Nucleo F446RE
Morpho Headers