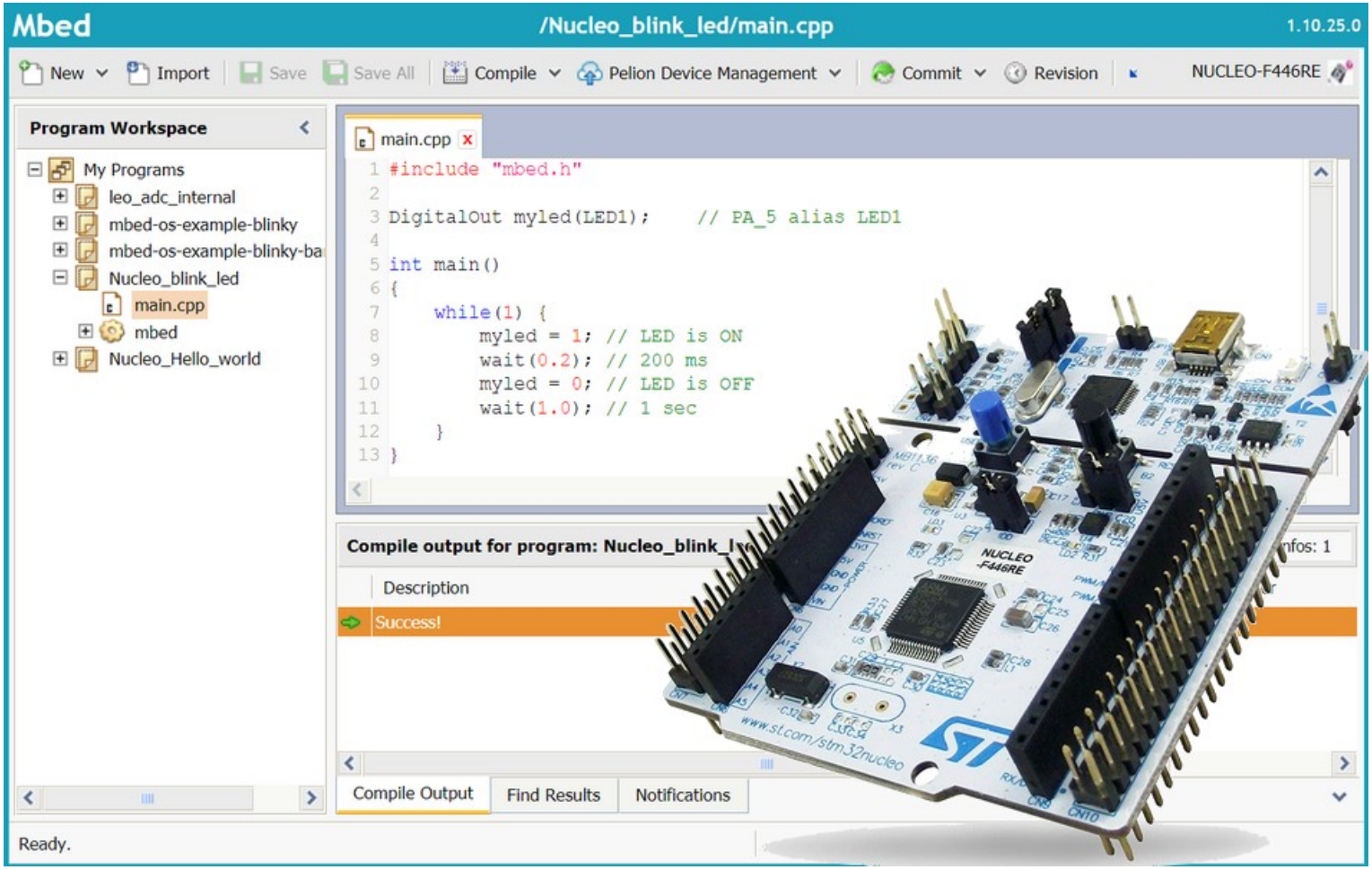
STM32 mikrovezérlők programozása ARM mbed környezetben



9. RTOS időzítők, megszakítások

Felhasznált és ajánlott irodalom

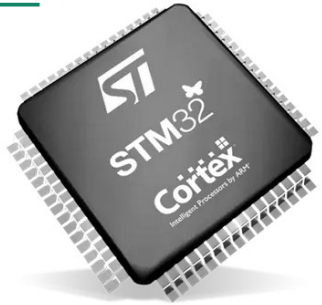
- Cserny István: **A FRDM-KL25Z kártya programozása mbed környezetben**

- Rob Toulson, Tim Wilmhurst:
  Fast and Effective Embedded Systems Design: Applying the ARM mbed
- Perry Xiao: Designing Embedded Systems and the IoT eith 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: **https://os.mbed.com/handbook/RTOS**
  - ARM mbed forráskód: https://github.com/ARMmbed/mbed-os

Adatlapok:

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



Példaprogramok

Lab09_ledblink – egyszerű RTOS timer példa

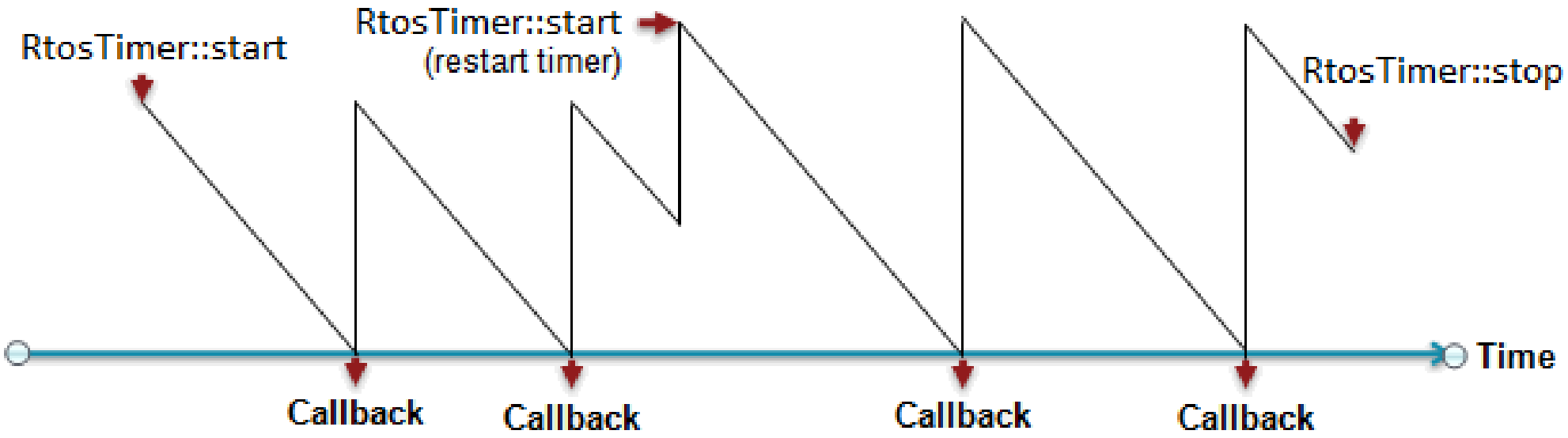
Lab09_rtos_timer – több RtosTimer kezelése

Lab09_rtos_interrupts – RTOS megszakítások

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

RTOS időzítők

- Az **RtosTimer** objektumosztály segítségével virtuális időzítőket hozhatunk létre, amelyek beállíthatók egyszeri vagy ismétlődő időzítésre és a túlcsordulási eseményhez visszahívási függvényt rendelhetünk
- Eltérés az mbed API **Ticker** és **Timeout** osztályokhoz képest az, hogy **RtosTimer** visszahívási függvénye nem megszakítási szinten fut, hanem az **osTimerThread** programszál alatt



Virtuális időzítő létrehozása

- Virtuális időzítő létrehozása: az **RtosTimer** objektum példányosításával történik

RtosTimer *name* (*void(*)(void const *argument)* ***task***,
os_timer_type ***type*** = *osTimerPeriodic*,
void * ***argument*** = *NULL*)

- **Paraméterek:**

  task - a visszahívási függvény

  type - az időzítő típusa (*osTimerOnce:* egyszeri, *osTimerPeriodic:* periodikus), alapértelmezetten periodikus.

  argument - a visszahívási függvénynek átadott, a hívást egyedivé tevő mutató *(alapértelmezetten NULL).*

Virtuális időzítő (újra)indítása

- Az előzőleg létrehozott **RtosTimer** virtuális időzítő indítása az objektumpéldány **start()** metódusával végezhető:

> ***osStatus* start** (*uint32_t time_delay*)

- **Paraméter:**
  - *time_delay* - a késleltetési idő ezredmásodpercben megadva *(max. 0xFFFE)*
- **Visszatérési érték:**

  A függvény visszatérési értéke az alábbi státusz, vagy hibakódok valamelyike lehet:
  - **osOK:** a megadott idejű késleltetés elindult (vagy újraindult)
  - **osErrorISR:** a függvény megszakítási szinten nem hívható meg
  - **osErrorParameter:** érvénytelen paraméter
- **Megjegyzés:** Ez a függvény megszakításból nem hívható!

Virtuális időzítő leállítása

- Az előzőleg létrehozott és elindított virtuális időzítő leállítása az **RtosTimer** objektumpéldány stop() metódusával végezhető:

> ***osStatus* stop** (*void*)

- **Paraméterek:** a függvénynek nincs paramétere
- **Visszatérési érték:** A függvény visszatérési értéke az alábbi státusz, vagy hibakódok valamelyike lehet:
  - **osOK:** az időzítő leállítása sikeresen megtörtént
  - **osErrorISR:** a függvény megszakítási szinten nem hívható meg
  - **osErrorParameter:** hibás vagy érvénytelen paraméter
  - **osErrorResource:** a leállítani kívánt időzítőt előtte nem indítottuk el
- **Megjegyzés:** Ez a függvény megszakításból nem hívható

Lab09_ledblink/main.cpp

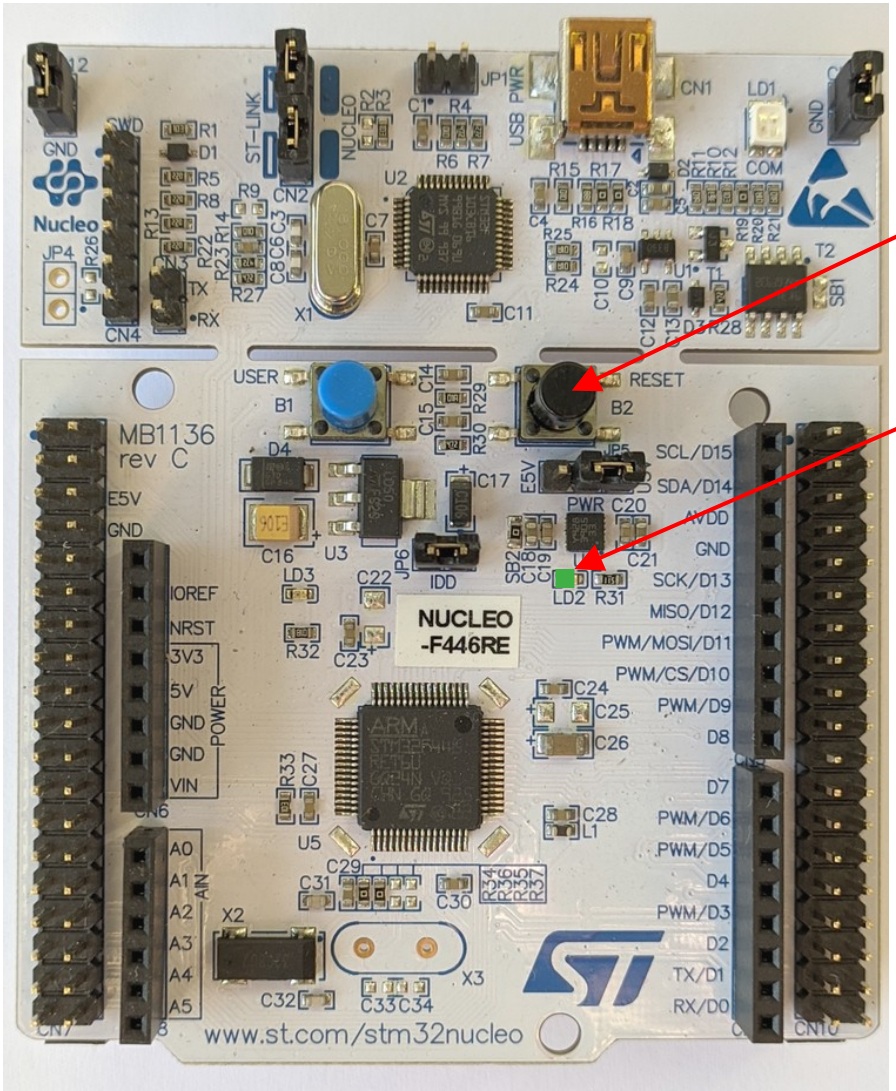
- Az alábbi programban az elindított időzítő addig villogtatja a beépített LED1-et, amíg a main() függvényben le nem állítjuk azt – jelen esetben tehát 5 másodpercig

  Forrás: Dogan Ibrahim, ARM-Based microcontroller projects using MBED

```
#include "mbed.h"
#include "rtos.h"
DigitalOut led(LED1);

void Flash() {
    led = !led;                     // Flash the LED
}

int main() {
    led = 1;                        // LED off
    RtosTimer timer(&Flash);        // Create a timer
    timer.start(500);               // Start the timer
    Thread::wait(5000);             // Wait 5 seconds
    timer.stop();                   // Stop the timer
    Thread::wait(osWaitForever); // Wait forever
}
```

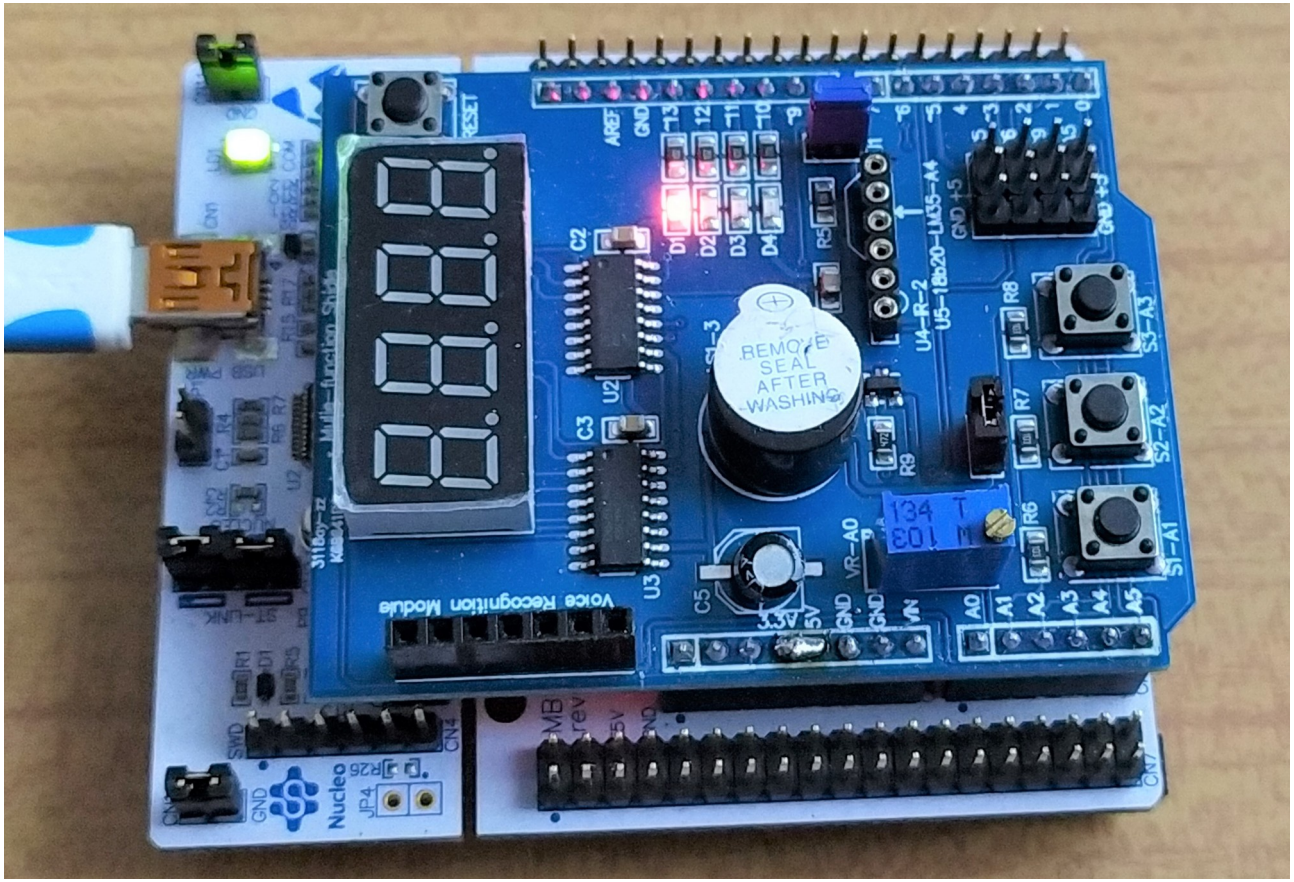
Lab09_ledblink

- A program újraindítása (RESET) után a beépített LED ötször felvillan



Lab09_rtos_timer

- A következő programban négy timerrel négy LED-et villogtatunk (**D10, D11, D12, D13** kimenetek), különböző időzítésekkel
- A főprogramnak az inicializálás és az időzítők elindítása után nincs mit tennie, ezért végtelen ciklus helyett végtelen várakozásba helyezzük



Lab09_rtos_timer/main.cpp

```
#include "mbed.h"
#include "rtos.h"

DigitalOut LEDs[4] = {DigitalOut(D13), DigitalOut(D12), DigitalOut(D11),
DigitalOut(D10)};

void blink(void const *n) {
    LEDs[(int)n] = !LEDs[(int)n];
}

int main(void) {
    RtosTimer led_1_timer(blink, osTimerPeriodic, (void *)0);
    RtosTimer led_2_timer(blink, osTimerPeriodic, (void *)1);
    RtosTimer led_3_timer(blink, osTimerPeriodic, (void *)2);
    RtosTimer led_4_timer(blink, osTimerPeriodic, (void *)3);

    led_1_timer.start(2000);
    led_2_timer.start(1000);
    led_3_timer.start(500);
    led_4_timer.start(250);

    Thread::wait(osWaitForever);
}
```

RTOS megszakítások

- A **programmegszakítások** kiszolgálása **RTOS** környezetben is ugyanúgy zajlik mint egyébként
- Az **RTOS API** hívásoknál az alábbi korlátozások vannak:
  - Programmegszakításban nem használhatók a **mutexek**, a **várakozások**, a **dinamikus tárterület** lefoglalások
  - **Szemaforok wait()** tagfüggvénye nem hívható megszakításból, csak a **release()** tagfüggvény
  - A programszálak jelzőbitjeit (Signals) kezelő függvények közül egyedül a **signal_set()** metódus hívható meg megszakításból
  - A **Queue** üzenetsor **put()** és **get()** metódusai egyaránt meghívhatók megszakításból, de csak nulla várakozási idő megadásával
  - **Mail** esetén az **alloc()** és **get()** metódus csak 0 várakozással hívható
  - **Queue** és **Mail** esetén a **get()** metódusnál nem nulla az alapértelmezett várakozási idő, ezért a nulla értéket muszáj megadni
  - Az **RTOS Timer** metódusai megszakításból nem hívhatók

LAB19_rtos_interrupt

- Programmegszakításokkal már a korábbi előadásokban is találkoztunk: az **InterruptIn**, a **Timeout,** a **Ticker,** illetve a **Wakeup** objektumosztályok kapcsán
- Ezúttal viszont azt mutatjuk meg, hogy az **NVIC**, illetve a periféria regiszterek programozásával hogyan kelthetünk és kezelhetünk olyan megszakítást, amelyre az **mbed API** nem nyújt megoldást
- A következő programban az ADC-vel végzünk méréseket. A mérési csatornát kijelölő és a konverziót elindító függvényben azonban nem blokkoló módon várakozunk, hanem egy üzenetsorra (Queue) várunk, melynek üzeneteit az ADC megszakítást kiszolgáló függvény küldi el
- Ehhez természetesen konfigurálnunk és engedélyeznünk kell az ADC megszakításokat, s egy kiszolgáló függvényt is kell rendelnünk hozzá
- Emellett azt is megmutatjuk, hogy az ADC belső hőérzékelőjével (ADC 18. csatorna) hogyan mérhetünk hőmérsékletet

LAB19_rtos_interrupt

- Az **ADC** kezeléséhez nem írtunk teljes inicializálást végző eljárást, csupán felülírjuk az **mbed API AnalogIn** konstruktor függvényének inicializálását (csak azokat a regisztereket írjuk felül, amelyeket módosítani akarunk)
- Az inicializálás felülírása az **adc_read()** függvényben történik az alábbiak szerint:
- Az **adc_read()** függvény paramétereként adhatjuk meg a mérendő analóg csatorna sorszámát (*ch*), amit az **ADC1->SQR3** regiszter legalsó bitjeibe kell beírni (A4 = 11, ADC_TEMP_SENSOR = 18)
- Az **ADC1->SMPR2** regiszter alsó bitjeibe írjuk a mintavételezési időt
- **ADC1->CR1** regiszter **EOCIE** bitje engedélyezi az ADC megszakítást
- **ADC1->CR2** regiszter **SWSTART** bitje indítja a konverziót
- A konverzió elindítása után nem blokkoló várakozást indítunk, hanem a **queue** néven példányosított üzenetsorra várunk (amely majd a megszakításból kap új adatot)

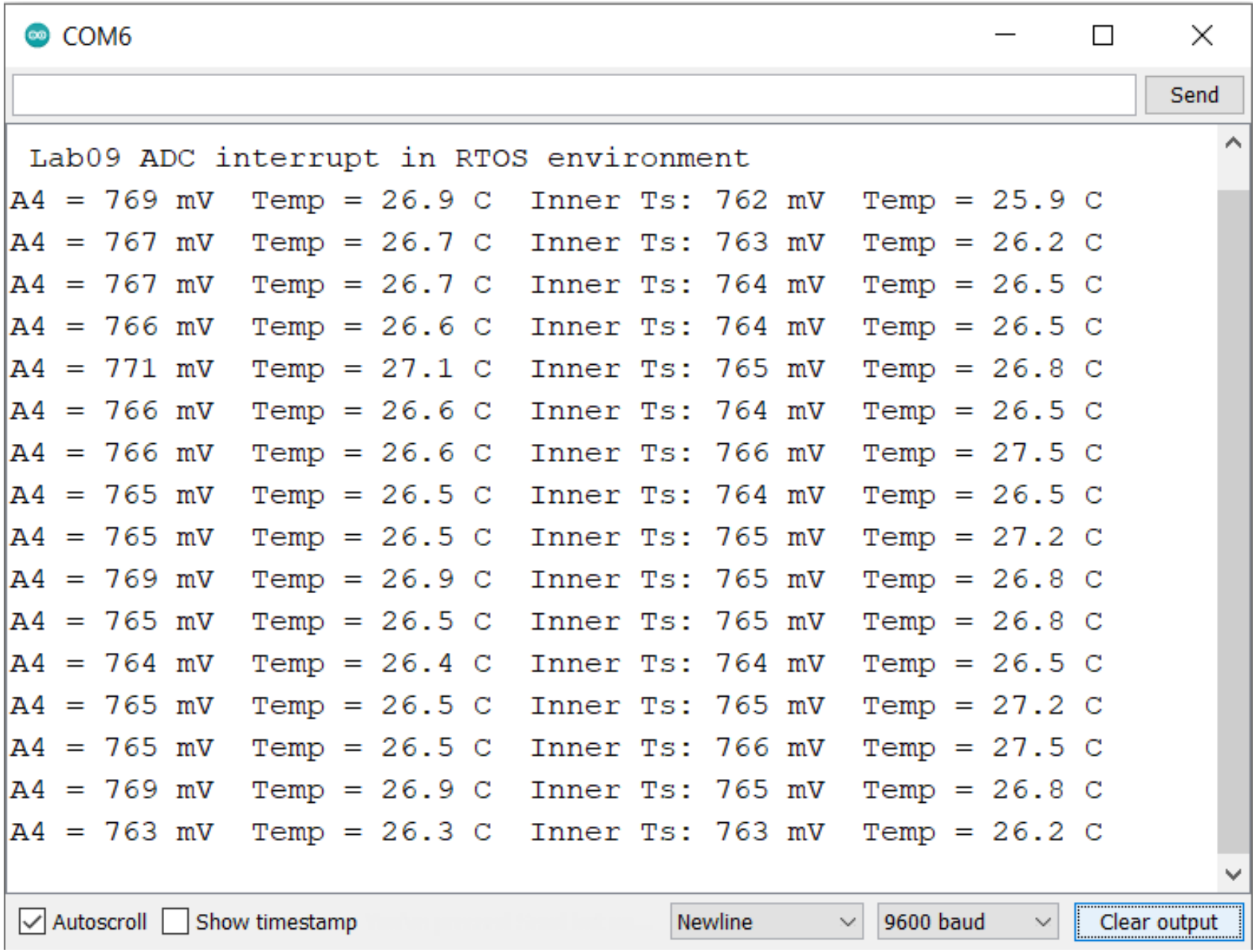
Lab09_rtos_interrupt/main.cpp

```
#include "mbed.h"
#include "rtos.h"

AnalogIn adc(A4);
DigitalOut led1(LED1);

typedef uint32_t message_t;                 // Message data type
Queue <message_t, 4> queue;                 // Message queue with direct data

extern "C" void ADC1_IRQHandler() {         //--- ADC Interrupt handler
    NVIC_ClearPendingIRQ(ADC_IRQn);         // Clear ADC Interrupt Request Flag
    uint16_t raw = ADC1->DR;
    queue.put((message_t*)raw);             // Send result through a Queue
}

uint16_t adc_read(uint32_t ch) {            //--- Start conversion
    ADC1->SQR3 = ch;                        // set conversion channel
    ADC1->SMPR2 = 7;                        // Sample time =480
    ADC1->CR1 |= ADC_CR1_EOCIE;             // enable interrupt
    ADC1->CR2 |= ADC_CR2_SWSTART;           // Start conversion
    osEvent evt = queue.get();              // Wait for a message
    return (uint16_t)evt.value.v;           // Return obtained value
}
```

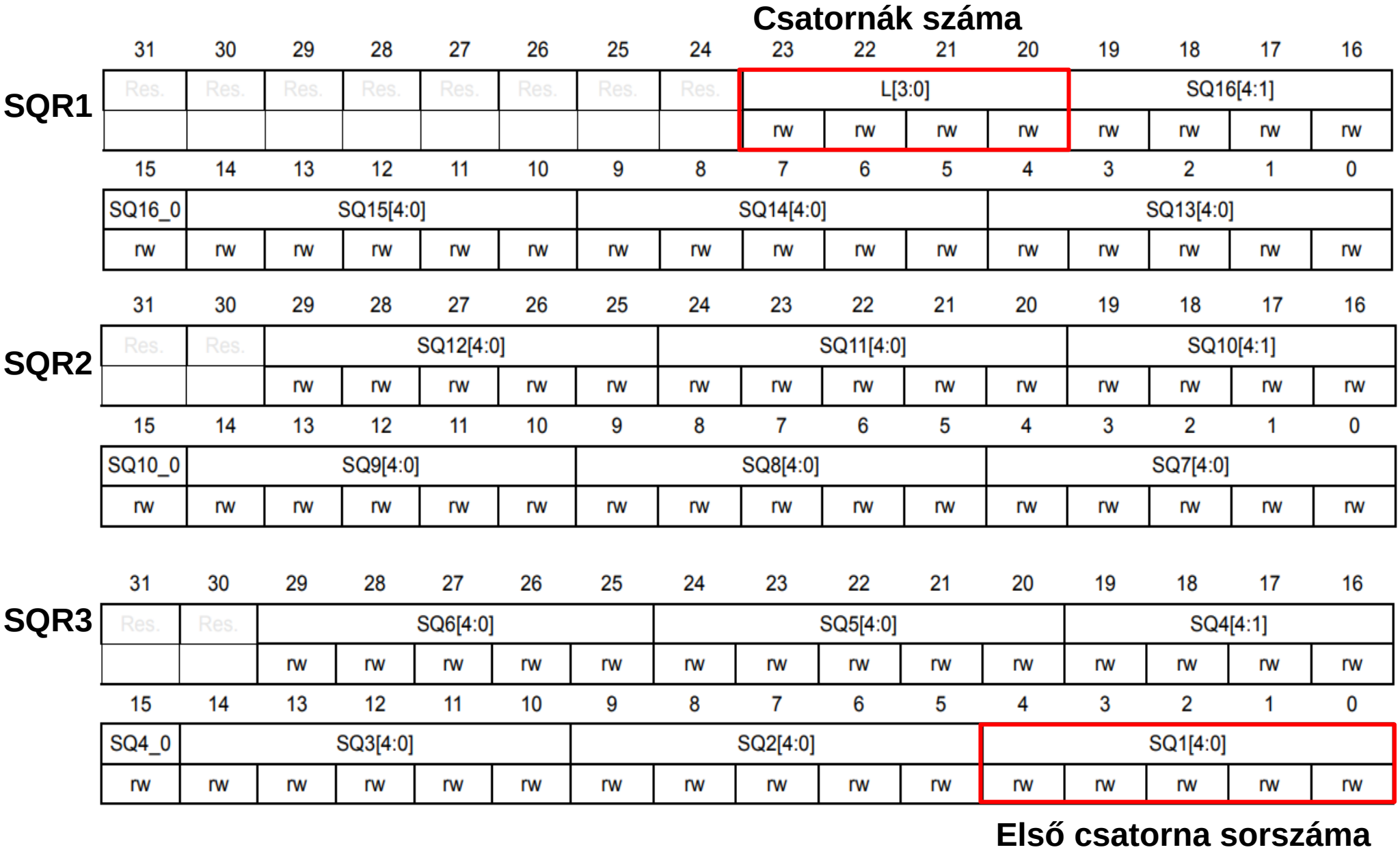
Lab09_rtos_interrupt/main.cpp

```
void led1_thread(void const *args) {
    while (true) {
        led1 = !led1; Thread::wait(1000);
    }
}

int main() {
  int32_t v25 = 760;                        //Voltage at 25C (in millivolts)
  float m = 2.5;                            //Slope  mV per degree)
    uint16_t dummy = adc.read();            //needed for ADC configuration
    ADC123_COMMON->CCR |= ADC_CCR_TSVREFE;      //Enable inner channels
    NVIC_SetVector(ADC_IRQn,(uint32_t)&ADC1_IRQHandler); //Attach ADC ISR
    NVIC_EnableIRQ(ADC_IRQn);               //Enable ADC interrupts
    Thread thread1(led1_thread);
    printf("\r\n Lab09 ADC interrupt in RTOS environment\r\n");
    while(true) {
      uint16_t a1 = adc_read(11);           //Measure voltage at A4 (PTC_1)
      uint16_t a2 = adc_read(18);           //Internal temperature sensor
      float v1 = a1*3300.0f/4096;           //Convert v1 to millivolts
      float temp1 = (v1-500)/10.0;          // MCP9700 temperature in Celsius
      float v2 = a2*3300.0f/4096;           //Convert v2 to millivolts
      float temp2 = 25.0f+(v2-v25)/m;       //Calculate temp in Celsius
      printf("A4= %.0f mV Temp = %.1f C
              Inner Ts: %.0f mV  Temp = %.1f C \n",v1,temp1,v2,temp2);
      Thread::wait(2000);
    }
}
```

LAB19_rtos_interrupt futási eredmény



ADC regular sequence registers



ADC sample time register 2

- Az **ADC_SMPR1** és **ADC_SMPR2** regiszterekben adható meg csatornánként, hogy hány órajelciklus legyen a mintavételezési idő (**0:** 3, **1:**15, **2:** 28, **3:** 56, **4:** 84, **5:** 112, **6:** 144, **7:** 480)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]			SMP5[2:1]	
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Belső hőmérő esetén 480 ciklus kell (7-et írunk **SMPR2**-be)

A belső hőmérő bekapcsolása

- A belső hőmérő bekapcsolásához az ADC-k közös **CCR** regiszterében (**ADC123_CCR**) 1-be kell állítani a **TSVREFE** bitet, a **VBATE** bitet pedig nullába kell törölni, mert a VBAT és a belső hőmérő mérése ugyanazon a 18-as analóg csatornán történik

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSVREFE	VBATE	Res.	Res.	Res.	Res.	ADCPRE	
								rw	rw					rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA[1:0]		DDS	Res.	DELAY[3:0]				Res.	Res.	Res.	MULTI[4:0]				
rw	rw	rw		rw	rw	rw	rw				rw	rw	rw	rw	rw

Calculate the temperature using the following formula:

Temperature (in °C) = $\{(V_{SENSE} - V_{25}) / Avg\_Slope\} + 25$

Where:

- V_{25} = V_{SENSE} value for 25° C
- Avg_Slope = average slope of the temperature vs. V_{SENSE} curve (given in mV/°C or µV/°C)

A vezérlő regiszterek felülírása

- Az **ADC_CR1** regiszter **EOCIE** bitjének 1-be állítása engedélyezi, hogy az **ADC** a konverzió végén megszakítást kérjen

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	OVRIE	RES		AWDEN	JAWDEN	Res.	Res.	Res.	Res.	Res.	Res.
					rw	rw	rw	rw	rw						

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCEN	DISCEN	JAUTO	AWDSGL	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

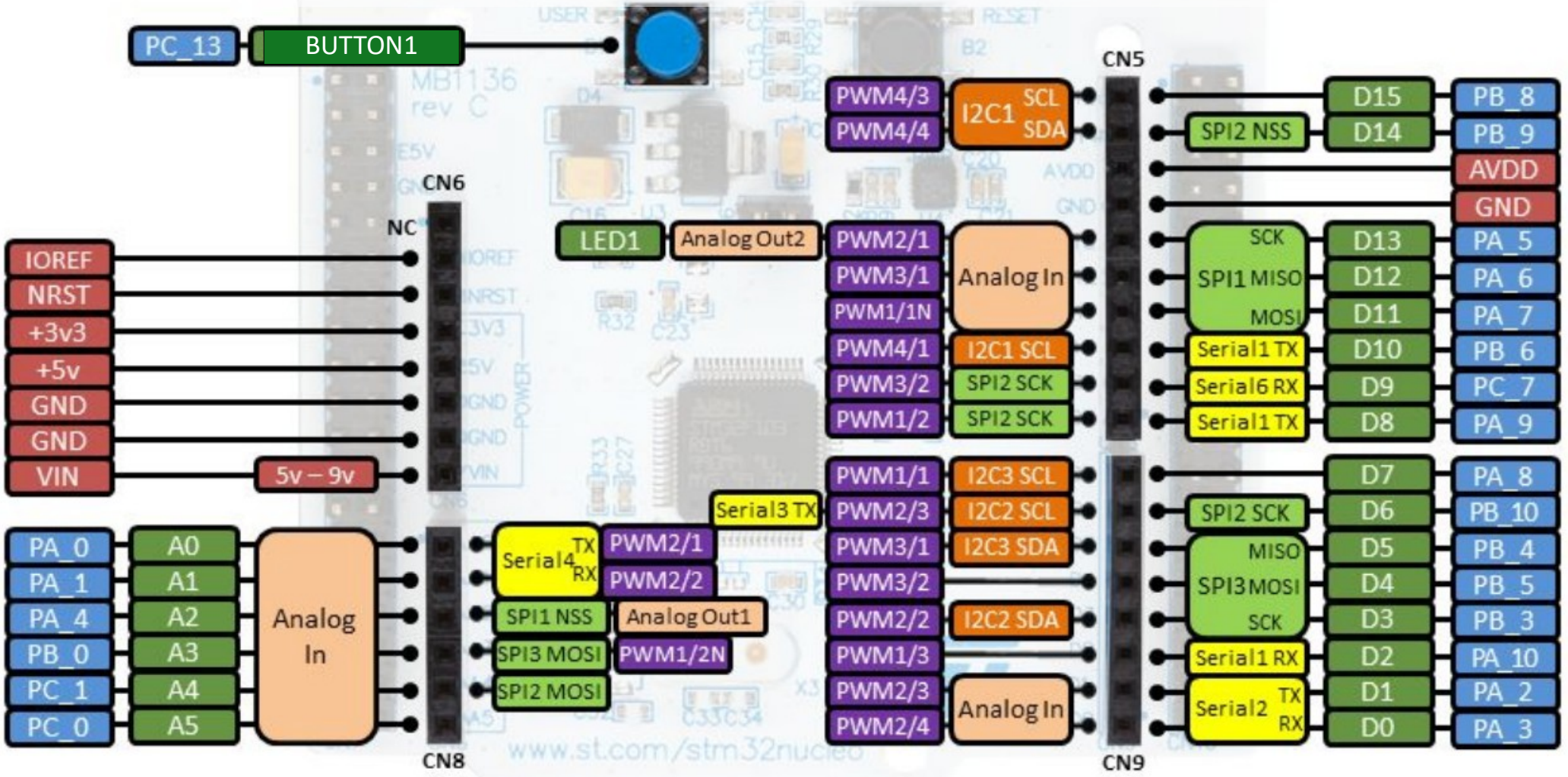
- Az **ADC_CR2** regiszter **SWSTART** bitjének 1-be állítása elindítja az adatgyűjtést és a konverziót

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	SWSTART	EXTEN		EXTSEL[3:0]				Res.	JSWSTART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	ALIGN	EOCS	DDS	DMA	Res.	Res.	Res.	Res.	Res.	Res.	CONT	ADON
				rw	rw	rw	rw							rw	rw

Arduino kompatibilis kivezetések

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



Morpho kivezetések

Nucleo F446RE

Morpho Headers