STM32 mikrovezérlők programozása STM32CubeIDE környezetben

4. Analóg Digitális átalakító (ADC)

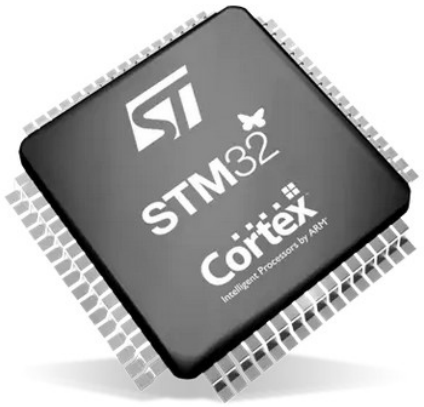
Felhasznált és ajánlott irodalom

- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi:
  STM32 Arm Programming for Embedded Systems

- Carmine Noviello: **Mastering STM32**

  A könyv mintapéldái: **github.com/cnoviello/mastering-stm32**
- Agus Kurniawan: **Getting Started With STM32 Nucleo Development**
- DeepBlue: **STM32 ADC read example**

Adatlapok, kézikönyvek:



- **STM32F103C8 adatlap és termékinfo**
- **STM32F103 Family Reference Manual**
- **STM32F446RE adatlap és termékinfo**
- **STM32F446 Family Reference Manual**

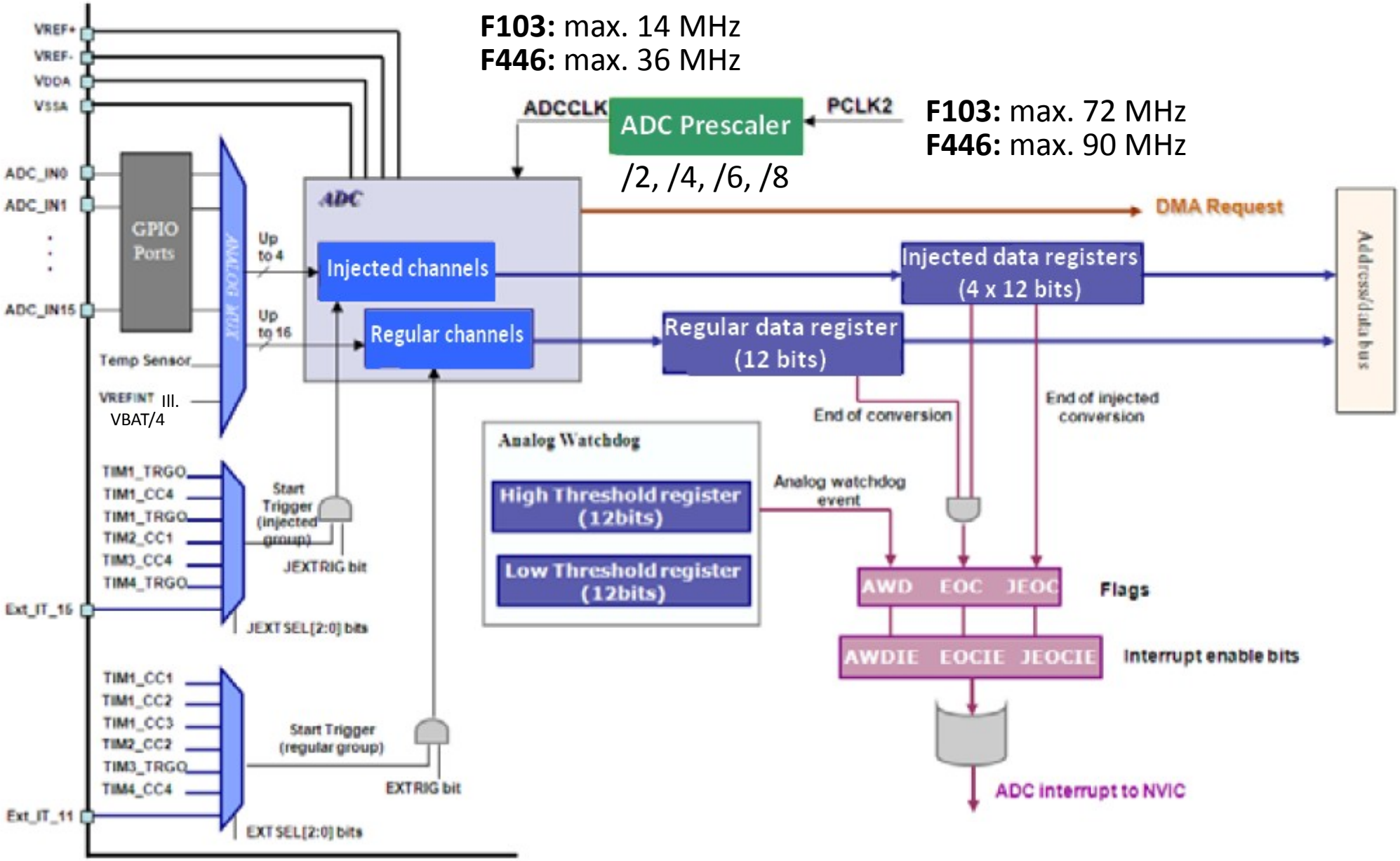
- STmicro: **Description of STM32F4 HAL and low-layer drivers**
- STmicro: **AN3116 – STM32's ADCs modes and their application**

Az STM32 ADC-k főbb jellemzői

	STM32F446RE	STM32F103C8
ADC	3 db (ADC1, ADC2, ADC3)	2 db (ADC1, ADC2)
Felbontás	6, 8, 10, 12 bit	12 bit
Üzemmód	független, duál, tripla	független, duál
VDDA	1.7 – 3.6 V	2.4 – 3.6 V
VREF	(VDDA-1.2 V) – VDDA	2.4 V – VDDA
fADC (VDDA ≥ 2.4 V)	0.6 – 36 MHz (typ: 30)	0.6 – 14 MHz
tconv	0.5 µs @ 12 bit 0.3 µs @ 6 bit	1 µs @ 12 bit

- Szingli és folytonos konverziós módok, önkalibrálás
- Pásztázó mód több csatorna automatikus méréséhez
- Csatornánként beállítható sorrend és mintavételezési idő
- Injektált csatornák közbevetett mérése (max. 4 db)
- Külső triggerelés a reguláris és az injektált csatornák méréséhez
- DMA adatátviteli kérelem generálása konverzió végén

Az ADC-k blokkvázata



STM32F446RE ADC-k további jellemzői

- Az **STM32F446** ADC-k némileg eltérnek az **STM32F103** mikrovezérlő **keil19_07 és keil19_08** előadásokban ismertetett ADC-itől
- **ADC** ki-/bekapcsolás: az **ADC*x*_CR2** regiszter **ADON** bitjével
- ADC konverzió indítása: **SWSTART** vagy **JSWSTART** bit 1-be
- **ADC órajel: APB2** busz **PCLK2** órajeléből leosztással (/2, /4, /6, /8)
- **Csatorna kiválasztás:** 16+2 csatorna áll rendelkezésre, melyek csoportokba rendezhetők:
  - **reguláris csatornák:** max. 16+2 pásztázandó csatorna, melyek száma és tetszőleges sorrendje az **ADC*x*_SQR*n*** regiszterekben adható meg
  - **injektált csatornák:** legfeljebb 4 db pásztázandó csatorna, melyek száma és sorrendje az **ADC*x*_JSQR** regiszterben adható meg
- **Belső referencia:** (1.21±0.03) V, **ADC1_IN17** csatornán érhető el
  Belső hőmérő: az **ADC1_IN18** csatornán érhető el

Temperature (°C) = (V_{SENSE} – V_{25})/Avg_Slope + 25 (V25 = 0.76 V, Avg_Slope = 2.5 mV/°C)

Az analóg bemenetek kiosztása

- Az alábbi táblázatban összefoglaltuk, hogy melyik analóg bemenet melyik GPIO portkivezetéshez csatlakozik
- **STM32F108C8** 48 pin **STM32F446RE** 64 pin

I/O Pin	ADC	I/O Pin	ADC
PA0	AIN[0]	**PC0**	AIN[10]
PA1	AIN[1]	**PC1**	AIN[11]
PA2	AIN[2]	**PC2**	AIN[12]
PA3	AIN[3]	**PC3**	AIN[13]
PA4	AIN[4]	**PC4**	AIN[14]
PA5	AIN[5]	**PC5**	AIN[15]
PA6	AIN[6]		
PA7	AIN[7]		
PB0	AIN[8]		
PB1	AIN[9]		

Az ADC-k üzemmódjainak áttekintése

- **Független módok:**
  - Egy csatorna, szingli konverzió
  - Pásztázás (több csatorna), szingli konverzió
  - Egy csatorna, folyamatos mód
  - Pásztázás, folyamatos mód
  - Injektált csatornák
- **Duál/Tripla módok:**
  - Duál, reguláris, szimultán mód
  - Duál, gyors átlapolásos mód
  - Duál, lassú átlapolásos mód
  - Duál alternáló triggerelésű mód
  - Duál kombinált: reguláris/injektált szimultán mód
  - Duál kombinált: injektált szimultán + átlapolásos mód

Ajánlott olvasmány:

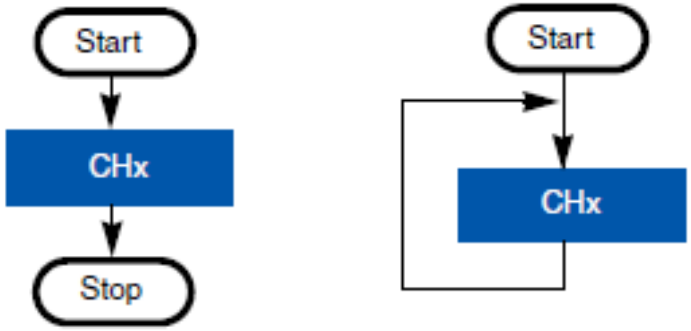
STmicro Application Note:
AN3116 – STM32's ADCs modes and their application

AN3116 mintaprogramok:
STSW-STM32028

Független módok

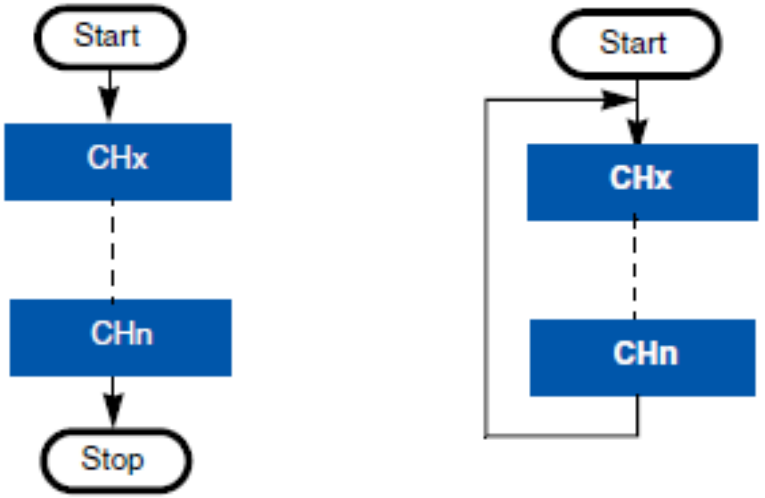
Egy csatorna

szingli konverzió folyamatos konverzió



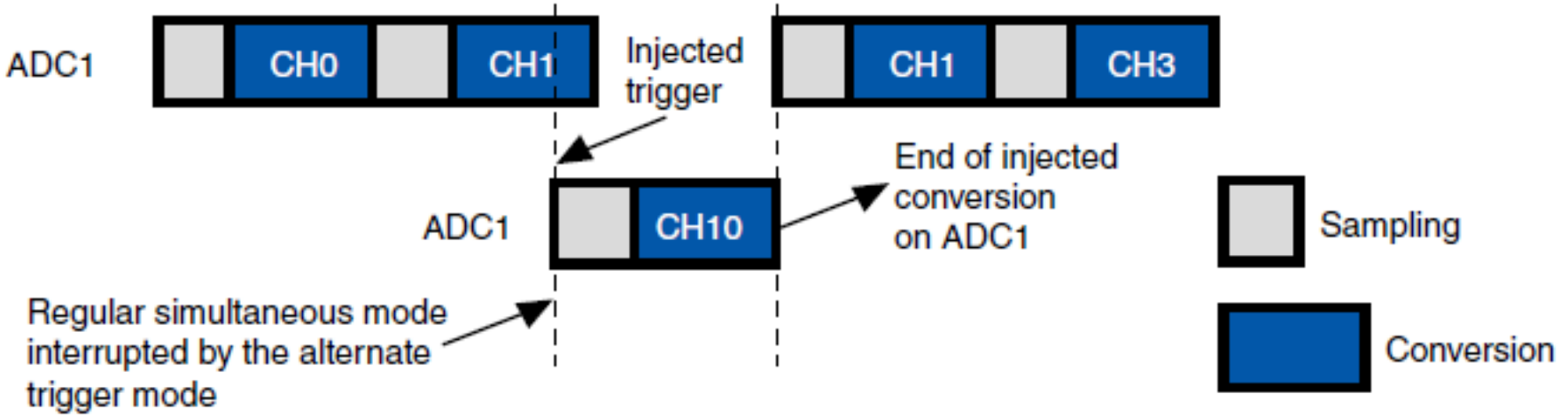
Pásztázás (több csatorna) (csak DMA-val lehetséges)

szingli konverzió folyamatos konverzió



Injektált konverziós mód

(legfeljebb 4 csatorna injektálható)



HAL_ADC és HAL_ADCEx modulok

- **HAL_ADC:** egyszerű kezelés, **HAL_ADCEx:** injektálás, multimód
- Az ADC-k kezeléséhez a HAL az alábbi struktúrát definiálja

```
typedef struct {
  ADC_TypeDef *Instance;          // Pointer to ADC descriptor
  ADC_InitTypeDef Init;           // ADC initialization parameters
  __IO uint32_t NbrOfCurrentConversionRank; // ADC number of current conversion rank
  DMA_HandleTypeDef *DMA_Handle; // Pointer to the DMA Handler
  HAL_LockTypeDef Lock;           // ADC locking object
  __IO uint32_t State;            // ADC communication state
  __IO uint32_t ErrorCode;        // Error code
} ADC_HandleTypeDef;
```

ahol:

Instance – mutató az ADC példány leírójához (pl. ADC1)
Init – `ADC_InitTypeDef` típusú struktúra az ADC inicializálásához
NbrOfCurrentConversionRank – az aktuális csatorna sorszáma
DMA_Handle – fogantyú a DMA módú használathoz
Lock – az ADC példány lefoglalására szolgál
State – az ADC állapotát jelző változó
ErrorCode – hiba esetén a hiba kódját tartalmazza

Az ADC konfigurálása

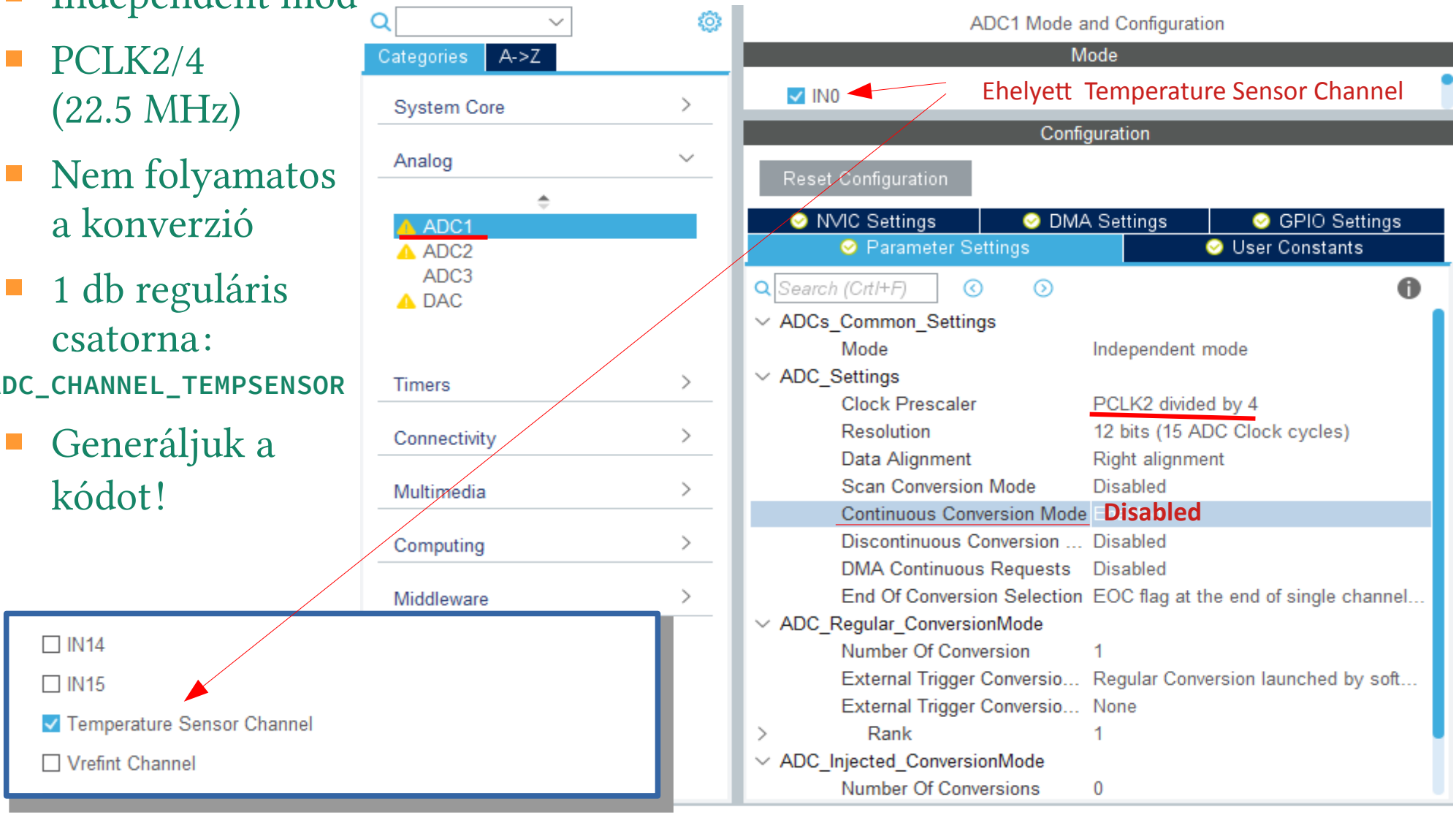
- **Az ADC engedélyezése:** `HAL_ADC_MspInit()` az alacsonyszintű konfigurálás helye
  - RCC szintű órajel engedélyezés (pl. `__ADC1_CLK_ENABLE()`)
  - órajel frekvencia beállítása
  - ADC GPIO kivezetések konfigurálása (`HAL_GPIO_Init()` segítségével)
- **ADC megszakítás engedélyezése** – ha szükséges, és hívjuk meg a `HAL_ADC_IRQHandler()`függvényt az `ADCx_IRQHandler()` rutinból
- **DMA csatorna konfigurálás** – ha szükséges
  - DMA csatorna konfigurálás a HAL_DMA_Init() függvénnyel
  - DMA csatorna megszakítás engedélyezés `(DMAx_Channelx_IRQn)`
  - `HAL_DMA_IRQHandler()` beszúrás `DMA2_Stream0_IRQHandler()`-be
- **ADC inicializálása** (`HAL_ADC_Init()` segítségével)
- **ADC csatornák konfigurálása** (`HAL_ADC_ConfigChannel()` segítségével)

Egy csatorna szingli konverziója

- Első mintapéldánkban (**F446RE_ADC1** projekt):
  - A belső hőmérő jelét mérjük meg (**ADC1** 18. csatorna)
  - Az eredményt kiíratjuk az **UART2** soros porton keresztül
- Hozzunk létre egy új projektet **F446RE_ADC1** névvel, és konfiguráljuk az órajeleket, az órabemeneteket és az SWD kivezetéseket úgy, mint a legelső előadáson! (CPU: 180 MHz, PCLK1: 45 MHz, PCLK2: 90 MHz)
- Konfiguráljuk az **UART2** soros portot a 3. előadásban leírtak szerint! (Aszinkron mód, nincs adatfolyam-vezérlés, 115 200 bps, 8-bit, no parity, 1 stop bit, full duplex, kétirányú forgalom)
- Konfiguráljuk **ADC1**-et a következő oldalon bemutatott módon, de az **ADC_CHANNEL_TEMPSENSOR** csatornát jelöljük ki **IN0** helyett és a **Continuous Conversion** (folyamatos konverzió) legyen letiltva (Disabled)

ADC konfigurálás STM32Cube MX alatt

- Az ***Analog*** szekcióban válasszuk ki **ADC1**-et majd konfiguráljuk
- Independent mód
- PCLK2/4 (22.5 MHz)
- Nem folyamatos a konverzió
- 1 db reguláris csatorna: `ADC_CHANNEL_TEMPSENSOR`
- Generáljuk a kódot!



stm32f4xx_hal_msp.c

- Az **ADC1** előző oldalon bemutatott konfigurálása és a kódgenerálás után az **stm32f4xx_hal_msp.c** forrásállomány ezekkel a függvényekkel bővül:

```
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc) {
  if(hadc->Instance==ADC1) {
  /* Peripheral clock enable */
    __HAL_RCC_ADC1_CLK_ENABLE();
  }
}

void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc) {
  if(hadc->Instance==ADC1) {
    /* Peripheral clock disable */
    __HAL_RCC_ADC1_CLK_DISABLE();
  }
}
```

ADC inicializálás a main.c állományban

```
static void MX_ADC1_Init(void) {
  ADC_ChannelConfTypeDef sConfig = {0};
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
      Error_Handler();
  /** Configure for the selected ADC regular channel */
  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
      Error_Handler();
}
```

A mintavételezési időt növeljük meg!
Lehetséges értékek:
3, 15, 28, 56, 84, 112, 144, 480

Szingli konverziós mód, lekérdezéssel

- Első mintapéldánkban (**F446RE_ADC1** projekt) a szingli módot használjuk, amikor az ADC egyetlen konverziót végez. A konverzió végét blokkoló módon, lekérdezéssel (polling) várjuk meg
- A konverzió elindítása esetünkben szoftveresen, történik, a `HAL_ADC_Start(&hadc1);` függvényhívással
- A konverzió végére blokkoló módon a `HAL_ADC_PollForConversion(&hadc1, 100);` függvényhívással várhatunk, ahol a második paraméter a ***timeout*** értéke (ms-ban)
- A konverzió eredményét a `HAL_ADC_GetValue(&hadc1)` függvény visszatérési értékeként kapjuk meg
- A zajok és bizonytalanságok ingadozásait átlagolással javíthatjuk. Itt most 1000 mérést átlagolunk, az eredményt kiíratjuk és tartunk 1 másodperc szünetet (hogy a kiírás ne legyen túl gyors)

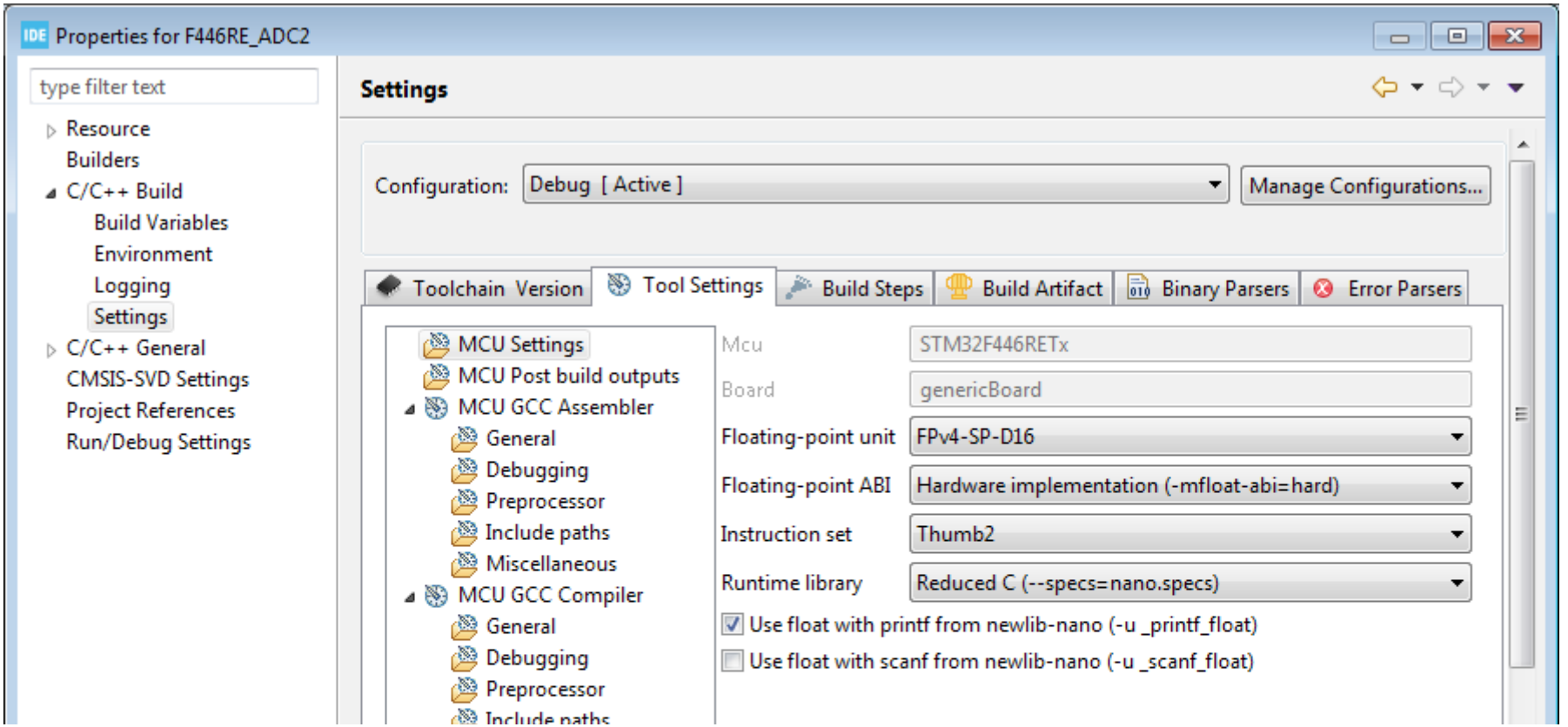
F446RE_ADC1 projekt: main.c

A programnak itt most csak az ADC1 használatával kapcsolatos releváns sorait mutatjuk
Az USART2 soros port konfigurálását és kezelését az előző előadásban mutattuk be

```
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;

int main(void) {
  char msg[80];
  float tempC, voltage;
  uint32_t val;
  HAL_Init(); SystemClock_Config(); MX_GPIO_Init(); MX_ADC1_Init(); MX_USART2_UART_Init();
  while (1) {
    for(int i = 0; i<1000; i++) {
      HAL_ADC_Start(&hadc1);                        // Start ADC1 conversion
      HAL_ADC_PollForConversion(&hadc1, 100);       // Wait for EOC
      val += HAL_ADC_GetValue(&hadc1);              // Read value
    }
    val = val/1000;
    voltage = (float)val * 3300 / 4096;             // Vref = 3.3V, resolution = 4096
    tempC = ((voltage - 760) / 2.5) + 25.0;
    sprintf(msg, "ADC= %d Voltage = %4.0f mV Temp = %4.1f C\r\n",(int)val,voltage,tempC);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(1000);
  }
}
```

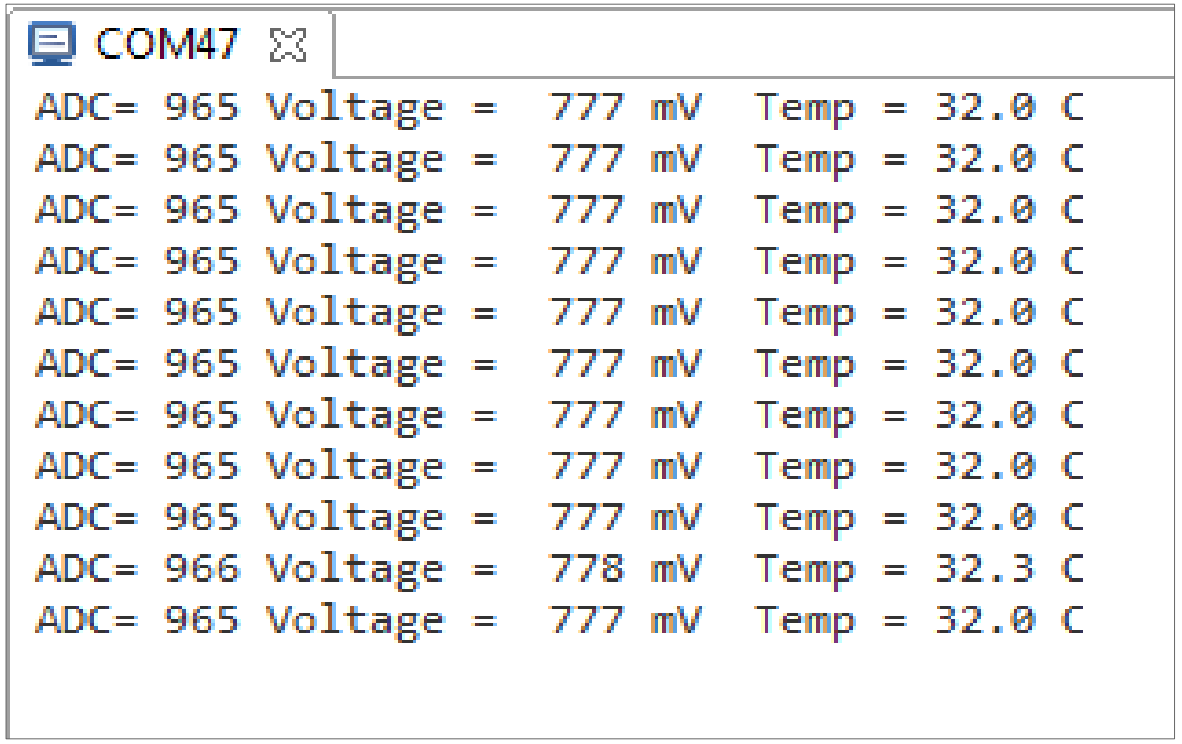
Lebegőpontos kiírás engedélyezése

- A projekt lefordítása előtt a **Project → Properties** menüben a **C/C++ Build → Settings** pontban a **Tool Settings** lapon tegyünk pipát a **Use float with printf from newlib-nano** opcióhoz!
- Utána kattintsunk az **Apply and Close** gombra!



F446RE_ADC1 projekt futási eredmény

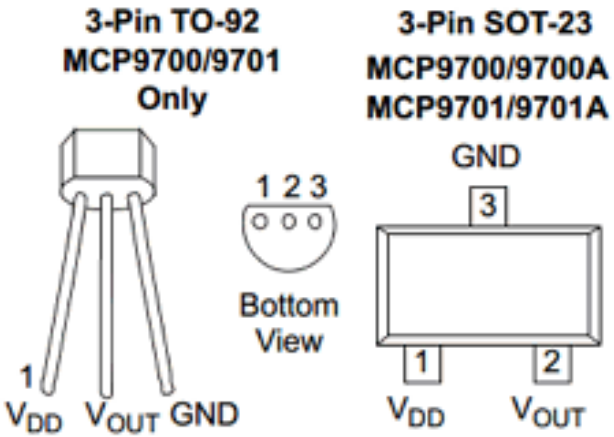
- A **NUCLEO-F446RE** kártyán az **UART2** soros port a **PA2, PA3** lábakon át a kártyára épített **ST-Link** programozó soros porti **Rx, Tx** kivezetéseire van kötve, így USB-n csatlakozik a számítógéphez is
- Egy terminál emulátor programmal, vagy az **STM32CubeIDE TM Terminal** bővítményével nyissunk egy terminál ablakot és ellenőrizzük a program eredményét! (Az adatsebesség 115 200 bps)

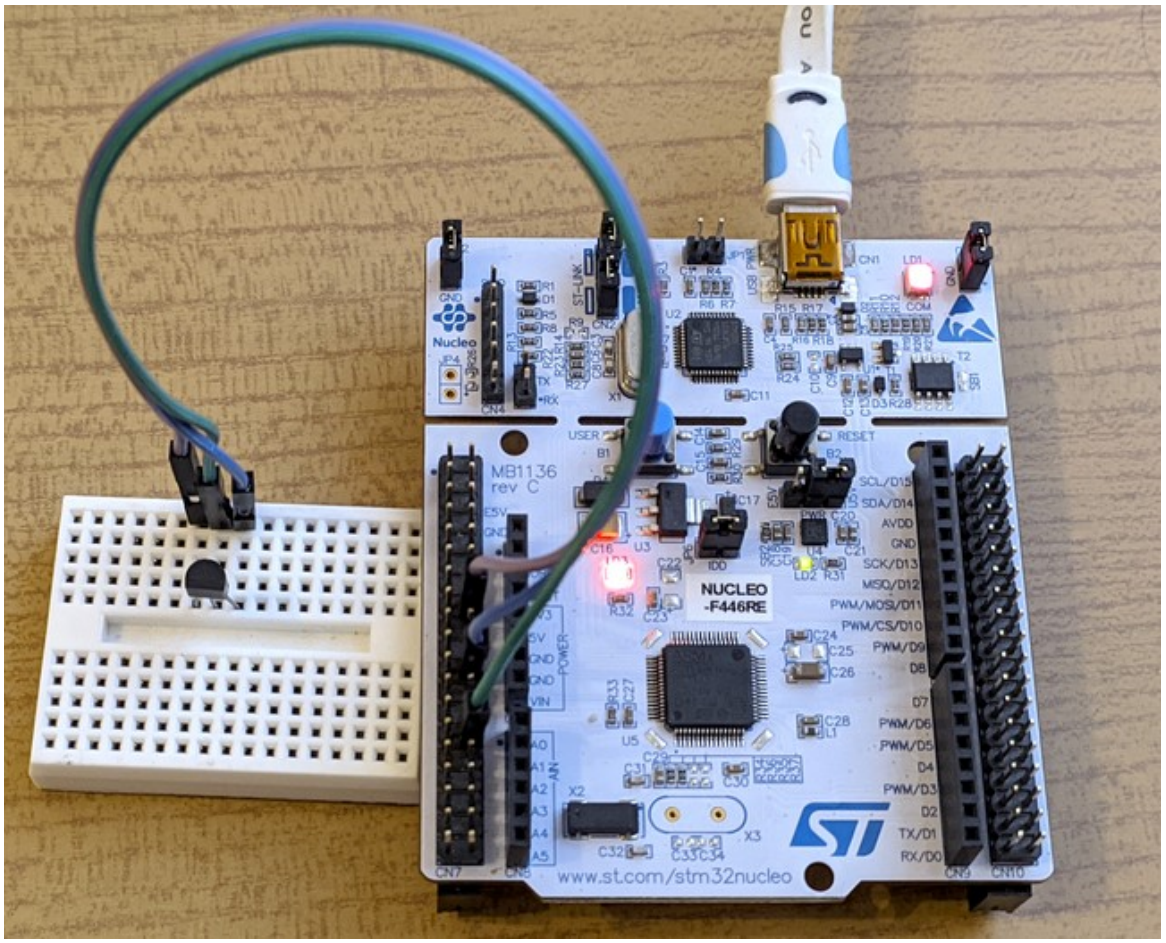


Egy csatorna folyamatos konverziója

- Második mintapéldánkban (**F446RE_ADC2** projekt):
  - Egy külső hőmérő jelét mérjük (**ADC2** AIN0. csatorna, **PA0** bemenet)
  - Az eredményt kiíratjuk az **UART2** soros porton keresztül a **B1** nyomógomb állapotától függő formában (hőmérséklet vagy feszültség)
  - A konverzió idejére felkapcsoljuk az **LD2** LED-et
- **MCP9700** paraméterei:

VDD = 3,3 – 5 V
VOUT = 500 mV @ 0 ℃
slope = 10 mV/℃

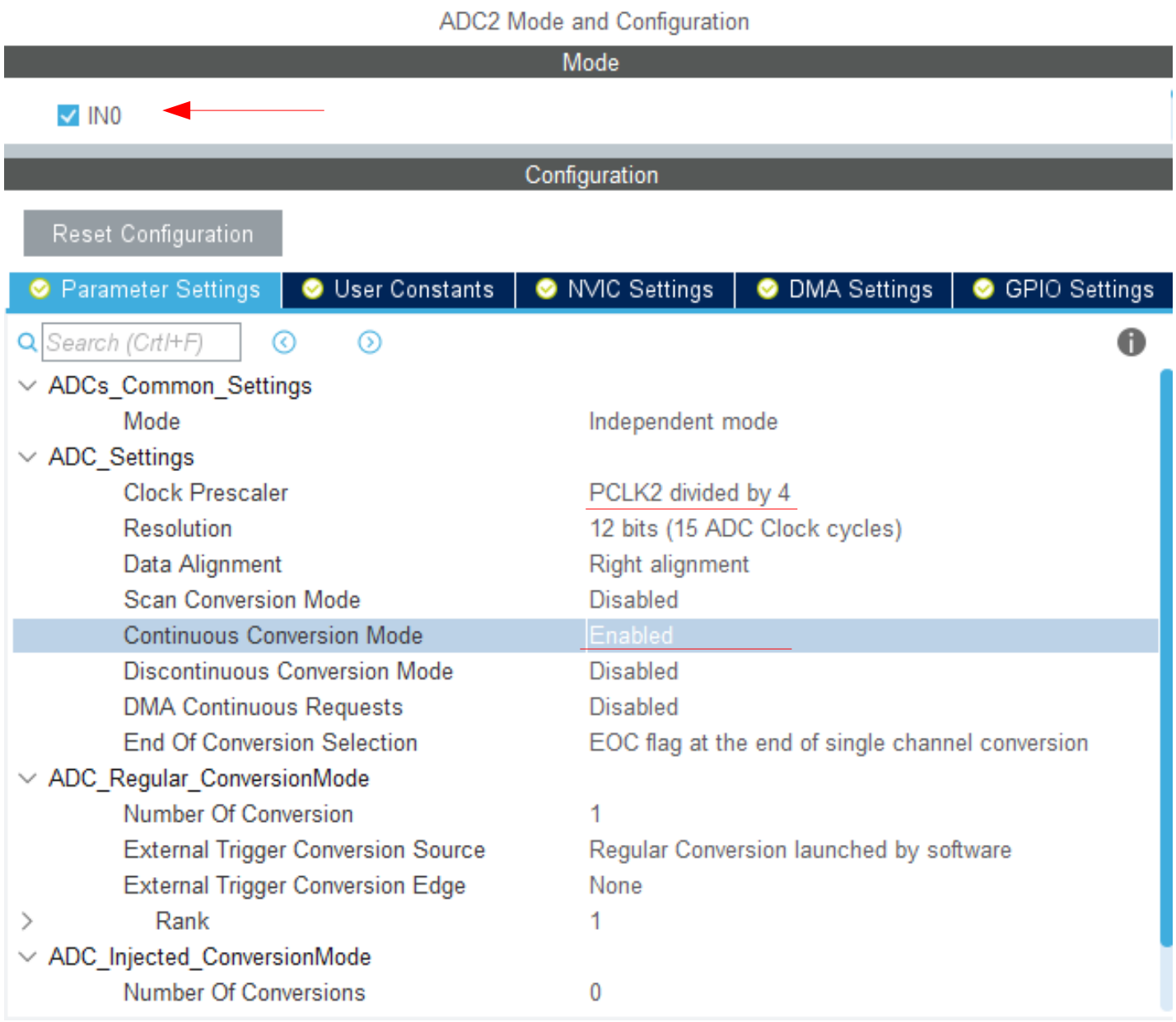




F446RE_ADC2 projekt konfigurálása

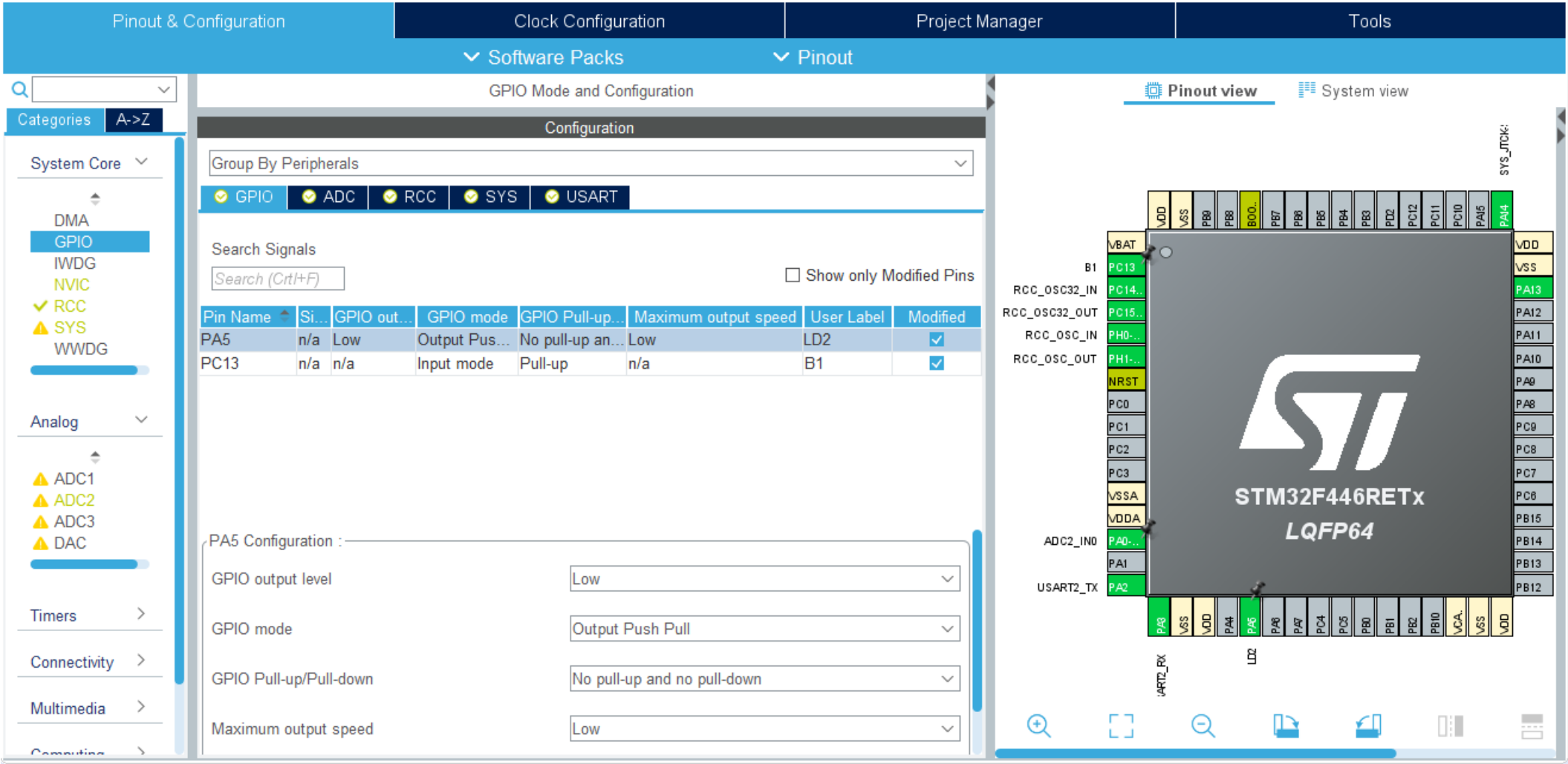
- Hozzunk létre egy új projektet **F446RE_ADC2** névvel, és konfiguráljuk az órajeleket, az órabemeneteket és az SWD programozó kivezetéseket úgy, mint a legelső előadáson!
  (CPU: 180 MHz, PCLK1: 45 MHz, PCLK2: 90 MHz)
- **GPIO** kivezetések konfigurálása:
  - **LD2** beépített LED (**PA5**) push-pull digitális kimenet
  - **B1** beépített nyomógomb (**PC13**) digitális bemenet, belső felhúzással
- Az **UART2** soros portot a 3. előadásban leírtak szerint konfiguráljuk!
  (Aszinkron mód, nincs adatfolyam-vezérlés, 115 200 bps,
  8-bit, no parity, 1 stop bit, full duplex, kétirányú forgalom)
- Konfiguráljuk **ADC2**-t:
  - az **IN0** csatornát jelöljük ki
  - válasszuk a **PCLK2**/4 órajelet
  - **Continuous Conversion Mode** legyen ***Enabled***

F446RE_ADC2 projekt: ADC konfigurálás



F446RE_ADC2 projekt: GPIO konfigurálás

- Az **NUCLEO-F446RE** kártya beépített **B1** nyomógombjának (**PC13**) és **LD2** LED-jének (**PA5**) konfigurálása ugyanúgy történik, mint a korábbi előadásokban



stm32f4xx_hal_msp.c

- Az **stm32f4xx_hal_msp.c** állományban az előző projekthez képest bővült az ADC alacsony szintű konfigurálása, mivel most nem csak **ADC2** órajelét kell engedélyezni, hanem a külső forrásból származó jel méréséhez a hozzá kapcsolódó GPIO bemenetet (**PA0**) is konfigurálni kell analóg bemenetként (az STM32Cube MX kódgenerátor ezeket a sorokat is beszúrja automatikusan)

```
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC2) {
    /* Peripheral clock enable */
    __HAL_RCC_ADC2_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    /* ADC2 GPIO Configuration PA0-WKUP ---> ADC2_IN0  */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  }
}
```

PA0 konfigurálása

ADC inicializálás a main.c állományban

```
static void MX_ADC2_Init(void) {
  ADC_ChannelConfTypeDef sConfig = {0};
  hadc2.Instance = ADC2;
  hadc2.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc2.Init.Resolution = ADC_RESOLUTION_12B;
  hadc2.Init.ScanConvMode = DISABLE;
  hadc2.Init.ContinuousConvMode = ENABLE;
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.NbrOfConversion = 1;
  hadc2.Init.DMAContinuousRequests = DISABLE;
  hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
      Error_Handler();
  /** Configure for the selected ADC regular channel */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_56CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
      Error_Handler();
}
```

A mintavételezési időt megnöveltük. Lehetséges értékek: 3, 15, 28, 56, 84, 112, 144, 480

F446RE_ADC2 projekt: main.c (részlet)

```
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

ADC_HandleTypeDef hadc2;
UART_HandleTypeDef huart2;

MX_GPIO_Init();
MX_ADC2_Init();
MX_USART2_UART_Init();

int main(void) {
  char msg[80];                 // SPRINTF buffer
  float tempC;                  // Temperature in C degs
  uint32_t val, voltage;        // temporary data and voltage in mV
  /* Initialisations */
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC2_Init();
  MX_USART2_UART_Init();
```

A programnak itt most csak az **ADC2** használatával kapcsolatos sorait mutatjuk Az **USART2** soros port konfigurálását és kezelését az előző előadásban mutattuk be

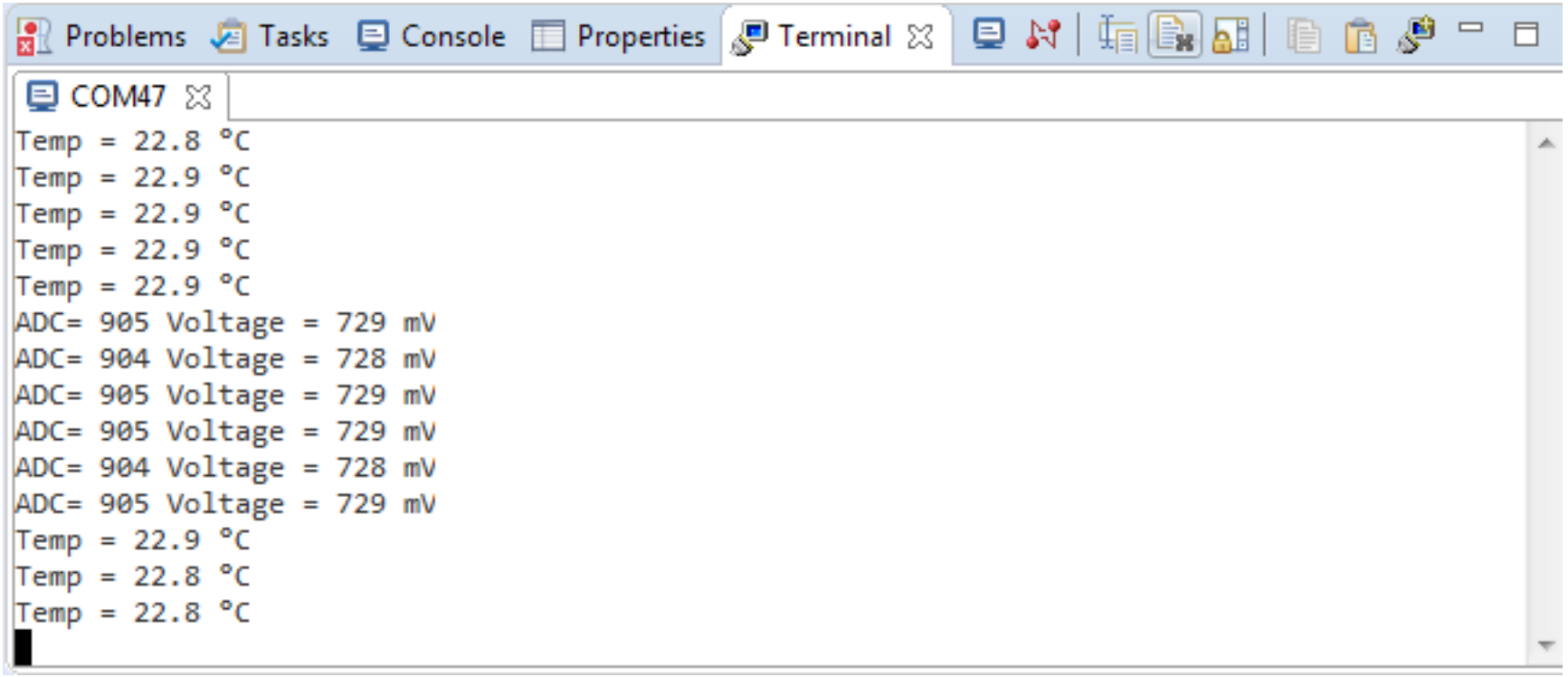
F446RE_ADC2 projekt: main.c (részlet)

A while ciklusban 200 mérést átlagolunk, s **B1** elengedett állapotában a hőmérsékletet, lenyomott állapotában a feszültséget írjuk ki

```
  while (1) {
    val = 0;
    HAL_ADC_Start(&hadc2);                              // Start ADC2 conversion
    HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);          // LED On
    for(int i = 0; i<200; i++) {                        // Make 200 measurements
      HAL_ADC_PollForConversion(&hadc2, 100);           // Wait for EOC
      val += HAL_ADC_GetValue(&hadc2);                  // Read value
    }
    HAL_ADC_Stop(&hadc2);                               // Stop ADC2 conversion
    HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);          // LED Off
    val = (val + 50)/200;
    voltage = val * 3300 / 4096;                        // Vref = 3300 mV, resolution = 4096
    tempC = ((float)voltage - 500.0) / 10.0;
    if(HAL_GPIO_ReadPin(B1_GPIO_Port,B1_Pin)) {         // Test B1 status (released or pressed?)
      sprintf(msg, "Temp = %4.1f \x0B0\x043\r\n", tempC);
    } else {
     sprintf(msg, " ADC= %d Voltage = %d mV\r\n", (int)val,(int)voltage);  // if released
    }
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY); // if pressed
    HAL_Delay(1000);
  }
}
```

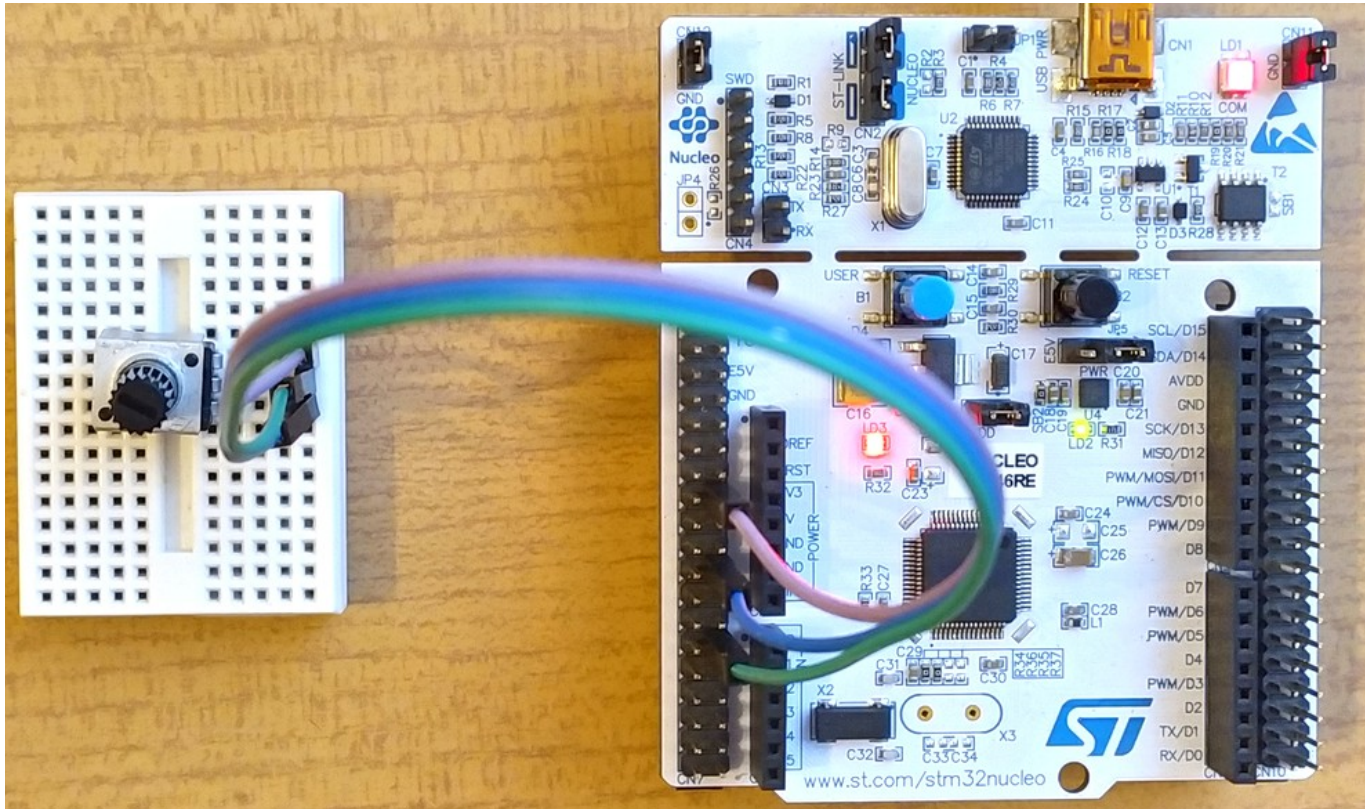
F446RE_ADC2 projekt futási eredmény

- A **NUCLEO-F446RE** kártyán az **UART2** soros port a **PA2, PA3** lábakon át a kártyára épített **ST-Link** programozó soros porti **Rx, Tx** kivezetéseire van kötve, így USB-n csatlakozik a számítógéphez is
- Egy terminál emulátor programmal, vagy az **STM32CubeIDE TM Terminal** bővítményével nyissunk egy terminál ablakot és ellenőrizzük a program eredményét! (Az adatsebesség 115 200 bps)



ADC kezelése megszakítással

- Az **F446RE_ADC_int** projektben az **ADC3** megszakításos kezelését mutatjuk be, amikor az ADC a konverzió végén (**EOC**) megszakítással jelzi, hogy kiolvashatjuk az eredményt
  - Potméterrel leosztott feszültséget mérünk (**AIN0** csatorna, **PA0** bemenet)
  - Az eredménnyel négyzetes arányban TIMER2 CH1 (**PA5**) PWM kitöltését, azaz a beépített LED fényerejét változtatjuk

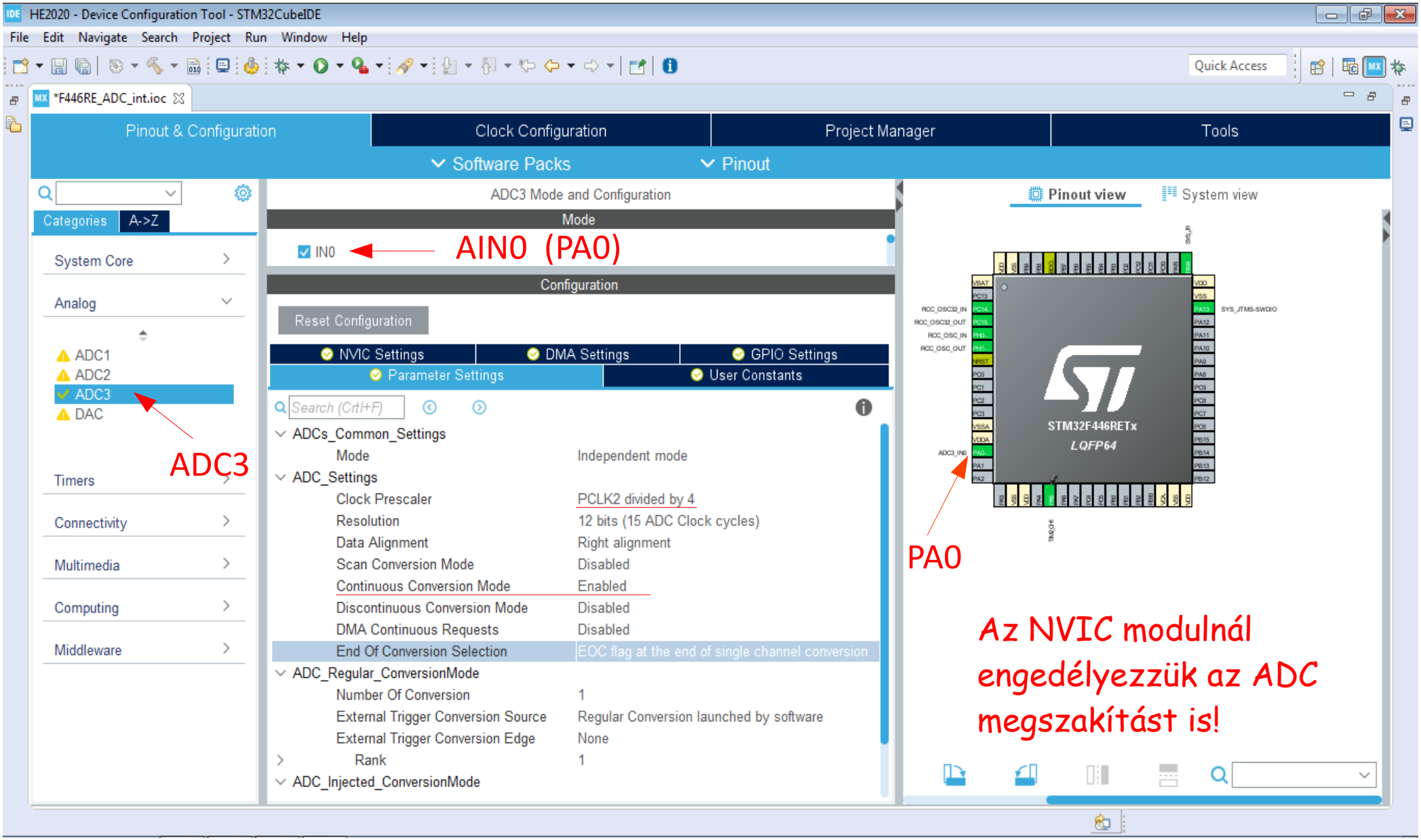


F446RE_ADC_int projekt konfigurálása

- Hozzunk létre a **F446RE_ADC_int** projektet, és konfiguráljuk az órajeleket, az órabemeneteket és az SWD kivezetéseket úgy, mint a legelső előadáson! (CPU: 180 MHz, PCLK1: 45 MHz, PCLK2: 90 MHz)
- Konfiguráljuk **ADC3**-at:
  - az **IN0** csatornát (**PA0**) jelöljük ki
  - válasszuk ki a **PCLK2/4** órajelet
  - **Continuous Conversion Mode** legyen ***Enabled***
- Az **NVIC** modulnál engedélyezzük az **ADC_IRQn** megszakítási vektort (prioritást is rendelhetünk hozzá)
- Konfiguráljuk **TIMER2 CH1** PWM kimenetét (**PA5**) úgy, hogy kb. 500 Hz-es ismétlődési frekvenciát kapjunk
  - Előosztó = 64
  - Periódus = 2800

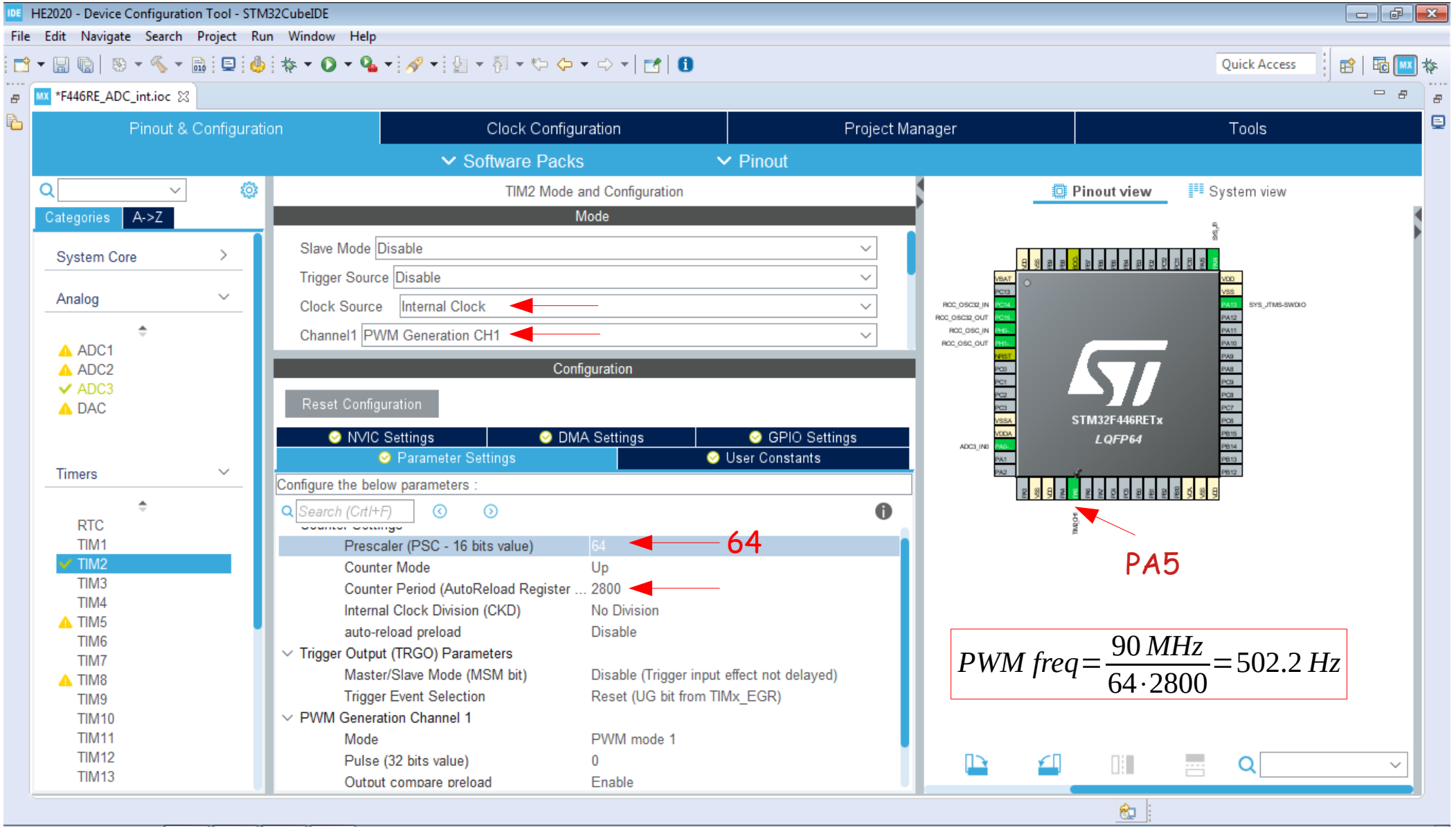
$$PWM\ freq = \frac{90\ MHz}{64 \cdot 2800} = 502.2\ Hz$$

F446RE_ADC_int projekt: ADC konfigurálás



F446RE_ADC_int projekt: Timer2 CH1 konfigurálás

- A **Timer2 CH1** PWM csatorna az alábbiak szerint könnyen konfigurálható



stm32f4xx_hal_msp.c

- Az **stm32f4xx_hal_msp.c** állományban az ADC alacsony szintű konfigurálása újabb sorokkal bővült: a megszakítás prioritásának beállításával és a megszakítási vektor engedélyezésével

```
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(hadc->Instance==ADC3) {
    /* Peripheral clock enable */
    __HAL_RCC_ADC3_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    /* ADC3 GPIO Configuration PA0-WKUP ---> ADC3_IN0  */
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /* ADC3 interrupt Init */
    HAL_NVIC_SetPriority(ADC_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(ADC_IRQn);
  }
}
```

PA0 konfigurálása

Megszakítás engedélyezés

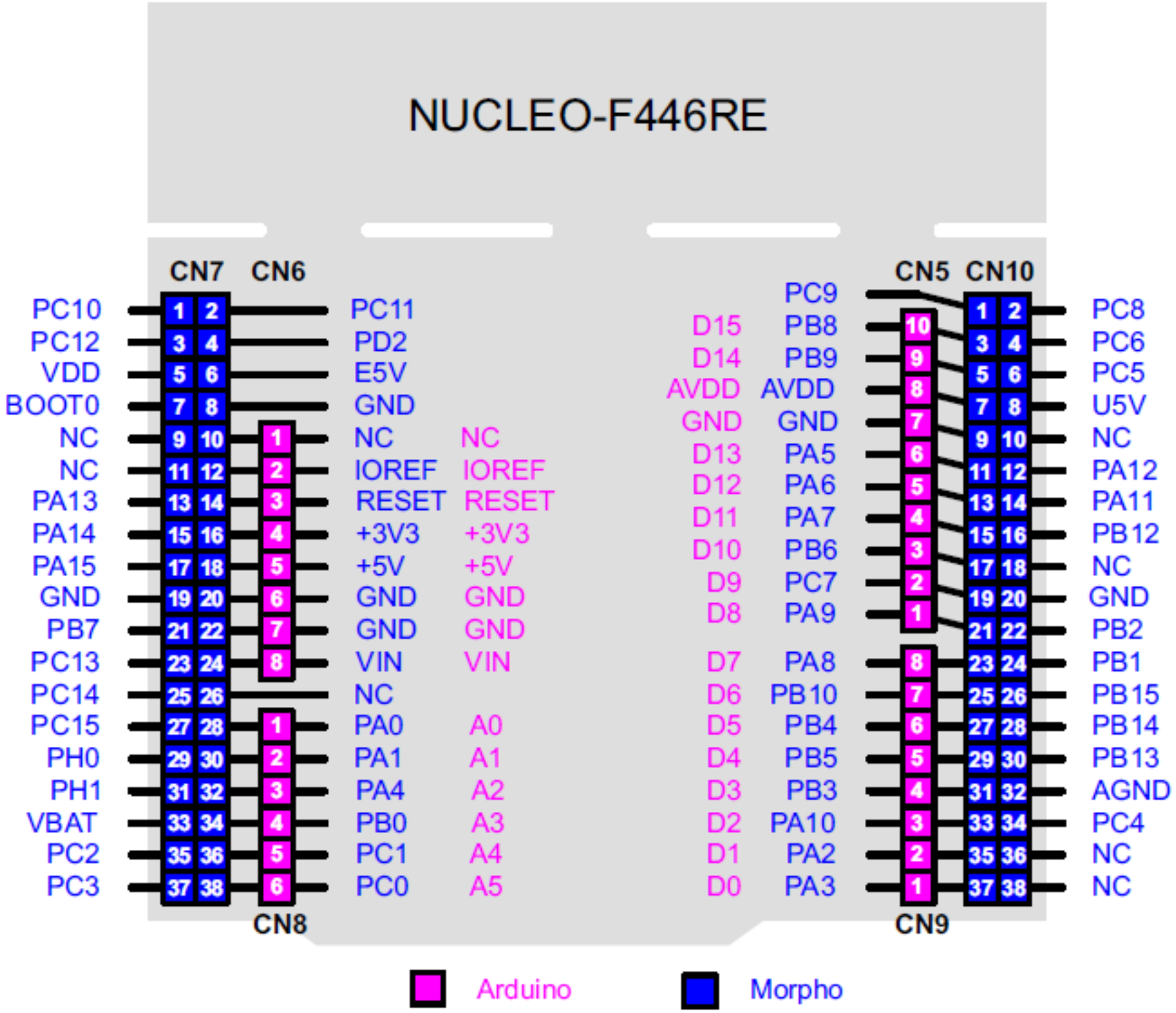
F446RE_ADC_int projekt: main.c (részlet)

```c
#include "main.h"
ADC_HandleTypeDef hadc3;
TIM_HandleTypeDef htim2;
volatile uint16_t ADC_RES = 0;

/* ADC megszakítás visszahívási függvénye */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
  if(hadc->Instance==ADC3) {
    ADC_RES = HAL_ADC_GetValue(&hadc3);     // Read ADC result
  }
}

int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC3_Init();
  MX_TIM2_Init();
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // Start PWM channel
  HAL_ADC_Start_IT(&hadc3);                 // Start ADC conversions
  while (1) {
    uint32_t val = ADC_RES >> 6;
    TIM2->CCR1 = (uint16_t)(val*val);       // Set PWM duty cycle
    HAL_Delay(10);
  }
}
```

stm32f4xx_it.c – a megszakítások kiszolgálása

- ADC-k: egyetlen közös megszakítási vektoruk van (**ADC_IRQn**)
- A HAL programkönyvtár egy **HAL_ADC_IRQHandler(&hadc)** függvényt biztosít a megszakítások teljeskörű kiszolgálásához, ezt kell meghívni az **ADC_IRQHandler()** megszakításkiszolgáló rutinból, a megfelelő ADC *„fogantyúját”* megadva
- Ha a megnevezett ADC-től érkezett megszakítási kérelem, akkor a megszakítás kiszolgálása után automatikusan meghívásra kerül a **HAL_ADC_ConvCpltCallback(&hadc)** visszahívási függvény, amelyben az alkalmazás elvégezheti a szükséges teendőket (pl. az ADC adatregiszterének kiolvasása, stb.

```
void ADC_IRQHandler(void) {
  // HAL_ADC_IRQHandler(&hadc1);     // ADC1 megszakítás kiszolgálása
  // HAL_ADC_IRQHandler(&hadc2);     // ADC2 megszakítás kiszolgálása
  HAL_ADC_IRQHandler(&hadc3);        // ADC3 megszakítás kiszolgálása
}
```


NUCLEO-F446RE
CN7
CN6
PC10
PC12
VDD
BOOT0
NC
NC
PA13
PA14
PA15
GND
PB7
PC13
PC14
PC15
PH0
PH1
VBAT
PC2
PC3
PC11
PD2
E5V
GND
NC
IOREF
RESET
+3V3
+5V
GND
GND
VIN
NC
PA0
PA1
PA4
PB0
PC1
PC0
NC
IOREF
RESET
+3V3
+5V
GND
GND
VIN
A0
A1
A2
A3
A4
A5
CN8
CN5
CN10
PC9
D15
D14
AVDD
GND
D13
D12
D11
D10
D9
D8
D7
D6
D5
D4
D3
D2
D1
D0
PB8
PB9
AVDD
GND
PA5
PA6
PA7
PB6
PC7
PA9
PA8
PB10
PB4
PB5
PB3
PA10
PA2
PA3
PC8
PC6
PC5
U5V
NC
PA12
PA11
PB12
NC
GND
PB2
PB1
PB15
PB14
PB13
AGND
PC4
NC
NC
CN9
Arduino
Morpho

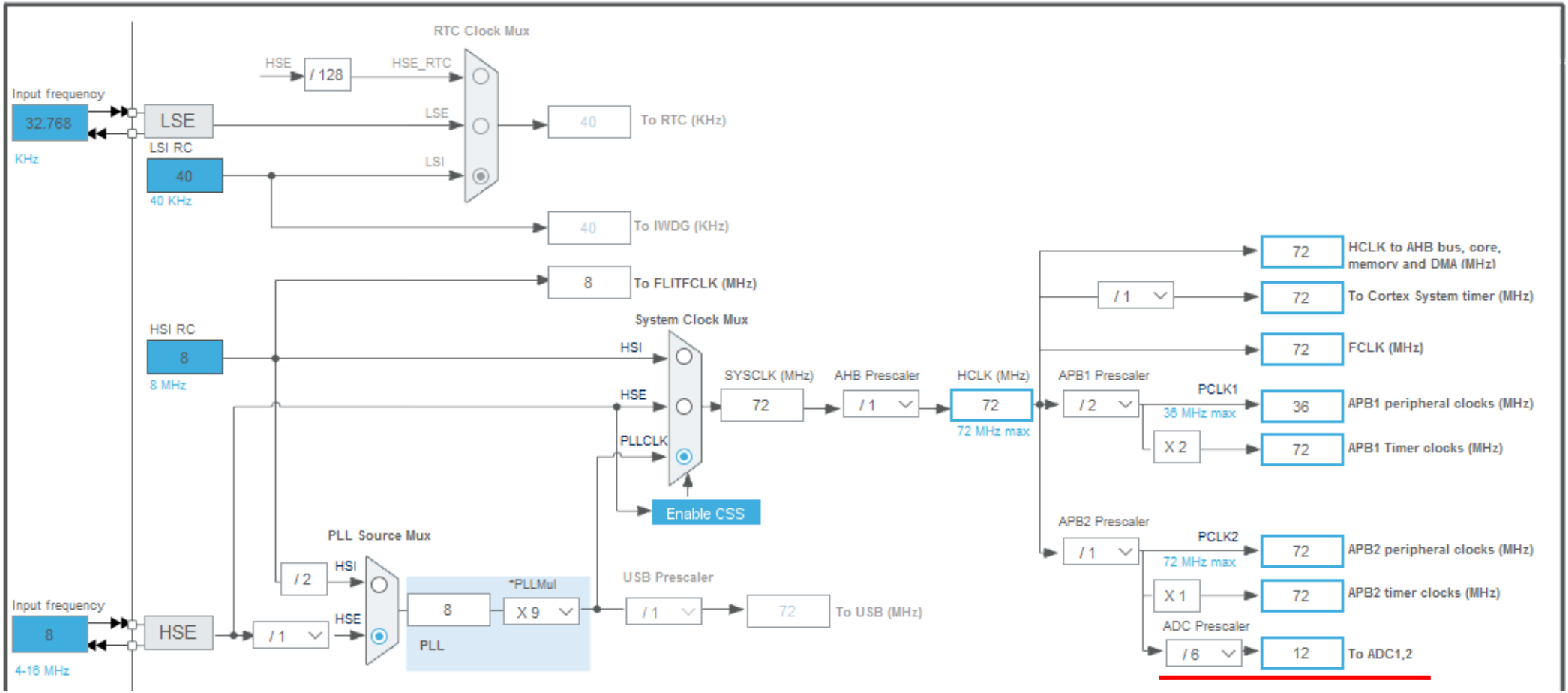
Példaprogramok az STM32F10C8 mikrovezérlőre

A mintaprojekteket **STM32F103C8** mikrovezérlőre is adaptáltuk:

- **F103_ADC1 projekt:** Egy USB-UART átalakítót kell csatlakoztanunk az **USART2** port **PA2** (Tx) és **PA3** (Rx) kivezetéseihez. A belső hőmérő paramétereit az adatlap tartalmazza
- **F103_ADC2 projekt:** Egy USB-UART átalakítót kell csatlakoztanunk az **USART2** port **PA2** (Tx) és **PA3** (Rx) kivezetéseihez. Az **LD2** beépített LED itt **PC13**-ra csatlakozik, a **B1** nyomógombot pedig **PB10** és GND közé kell kötni. Az **MCP9700** analóg hőmérő kimenetét **PA0**-ra kell kötni
- **F103_ADC_int projekt: ADC3** hiányában itt is **ADC1**-et használjuk. A potméter csúszkáját **PA0**-ra, a vezérelni kívánt LED-et pedig a **PA6** kimenetre kell kötni (**TIM3 CH1** PWM csatorna)
- A konfigurálás releváns részletei a következő oldalakon láthatók

Az ADC-k órajelének konfigurálása

- Az ADC-k előosztóját a **Clock Configuration** lapon állíthatjuk be
- **PCLK2** = 72 MHz esetén **/6** osztásra lesz szükség, ami 12 MHz-es órajelet biztosít az ADC-k számára (max. 14 MHz lehet)

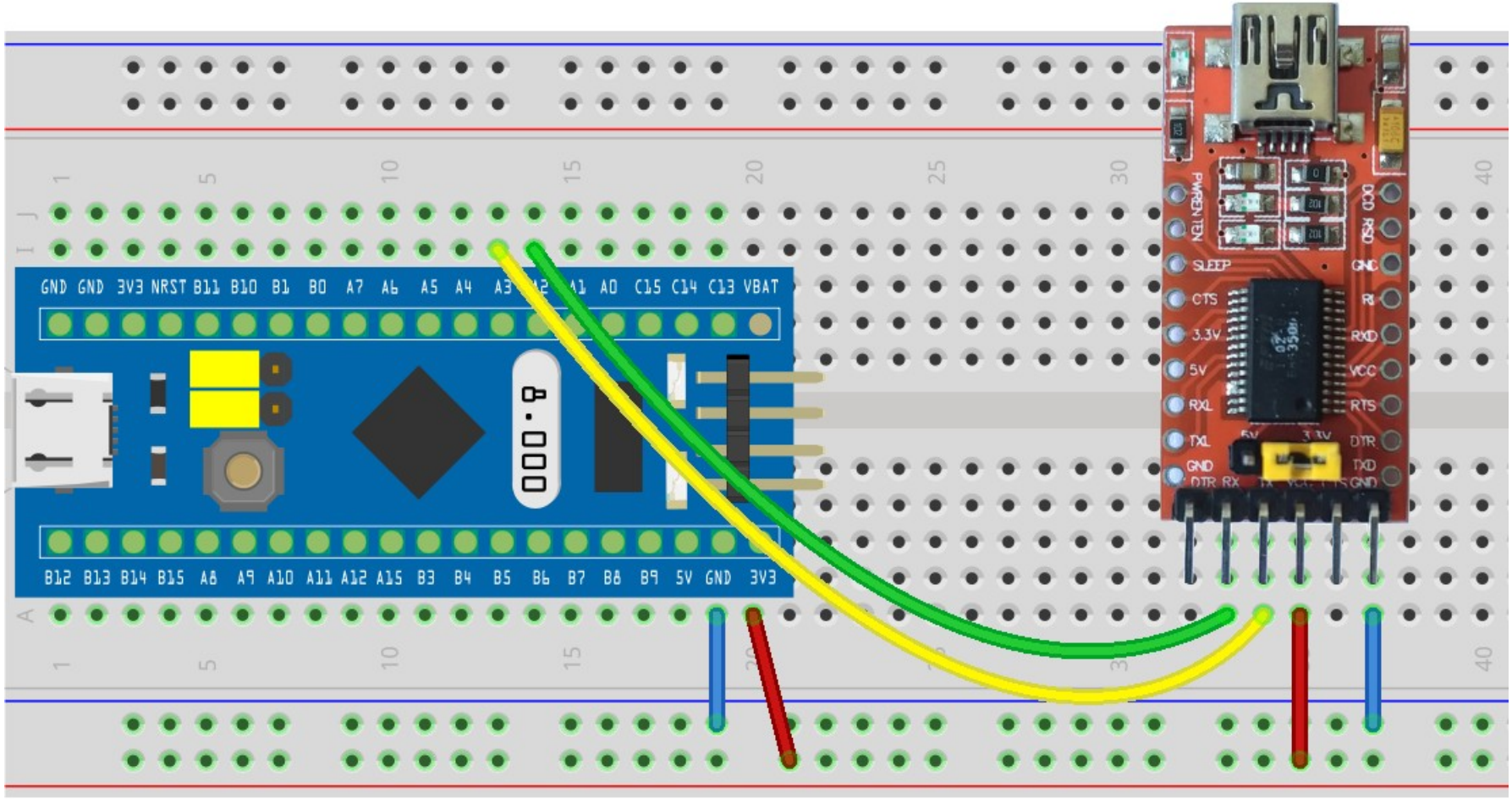


F103_ADC1 projekt

- Soros átalakító bekötése **PA2** ⟶ FT232 **Rx, PA3** ⟵ FT232 **Tx**
- A belső hőmérő **ADC1** 16. csatornájára csatlakozik

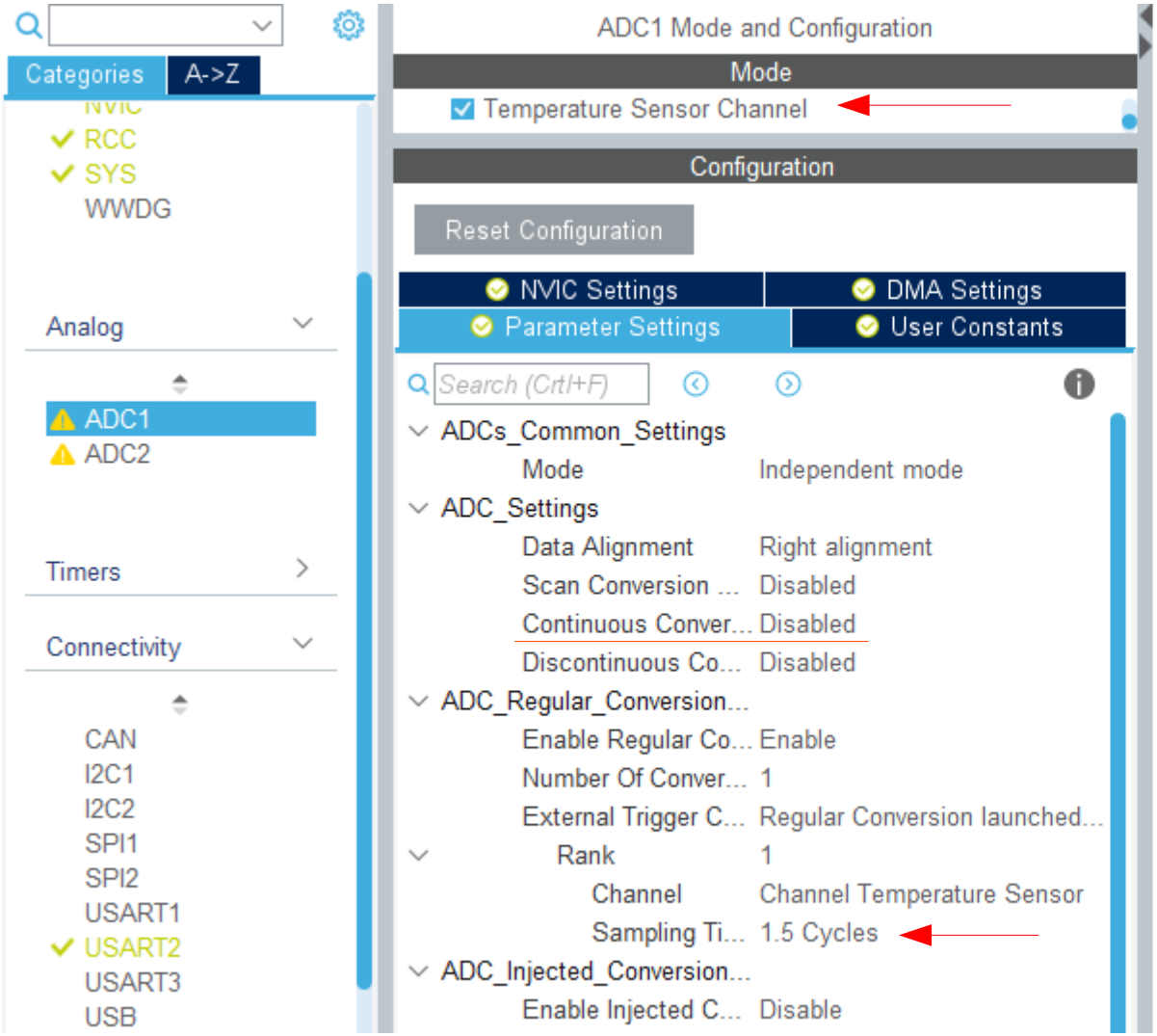
Temperature (°C) = (VSENSE– V25)/Slope + 25

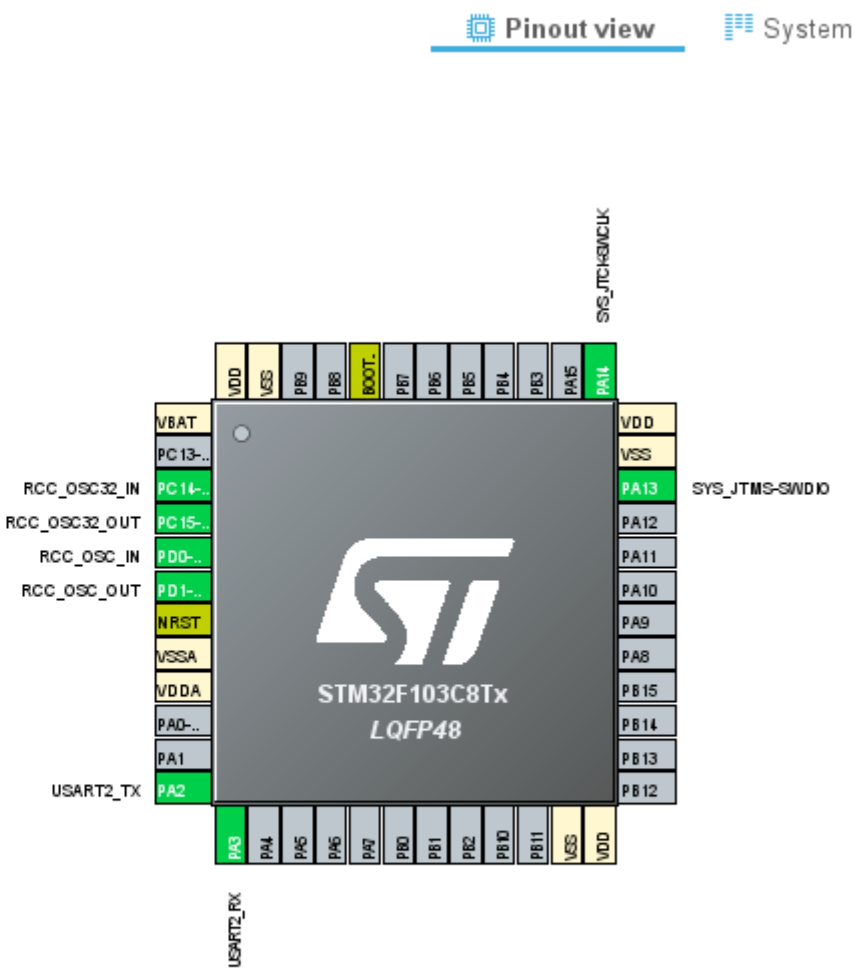
Az **STM32F103C8** MCU adatlapja szerint: V_{25} = 1.43 V, Slope = 4.3 mV/°C



F103_ADC1 projekt: ADC1 konfigurálása

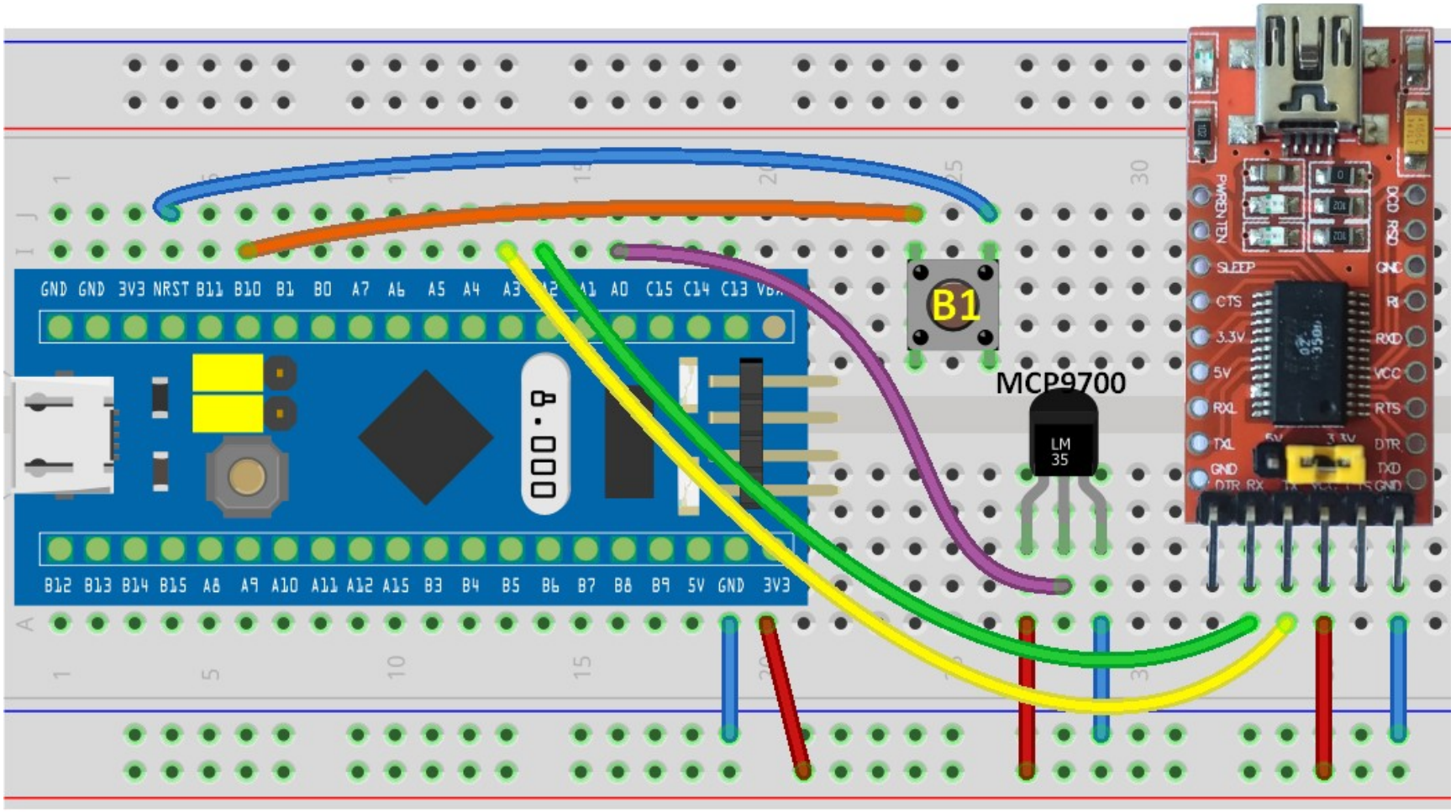
- Hőmérő csatorna kiválasztása, folytonos konverzió letiltása hosszabb mintavételezési idő választása





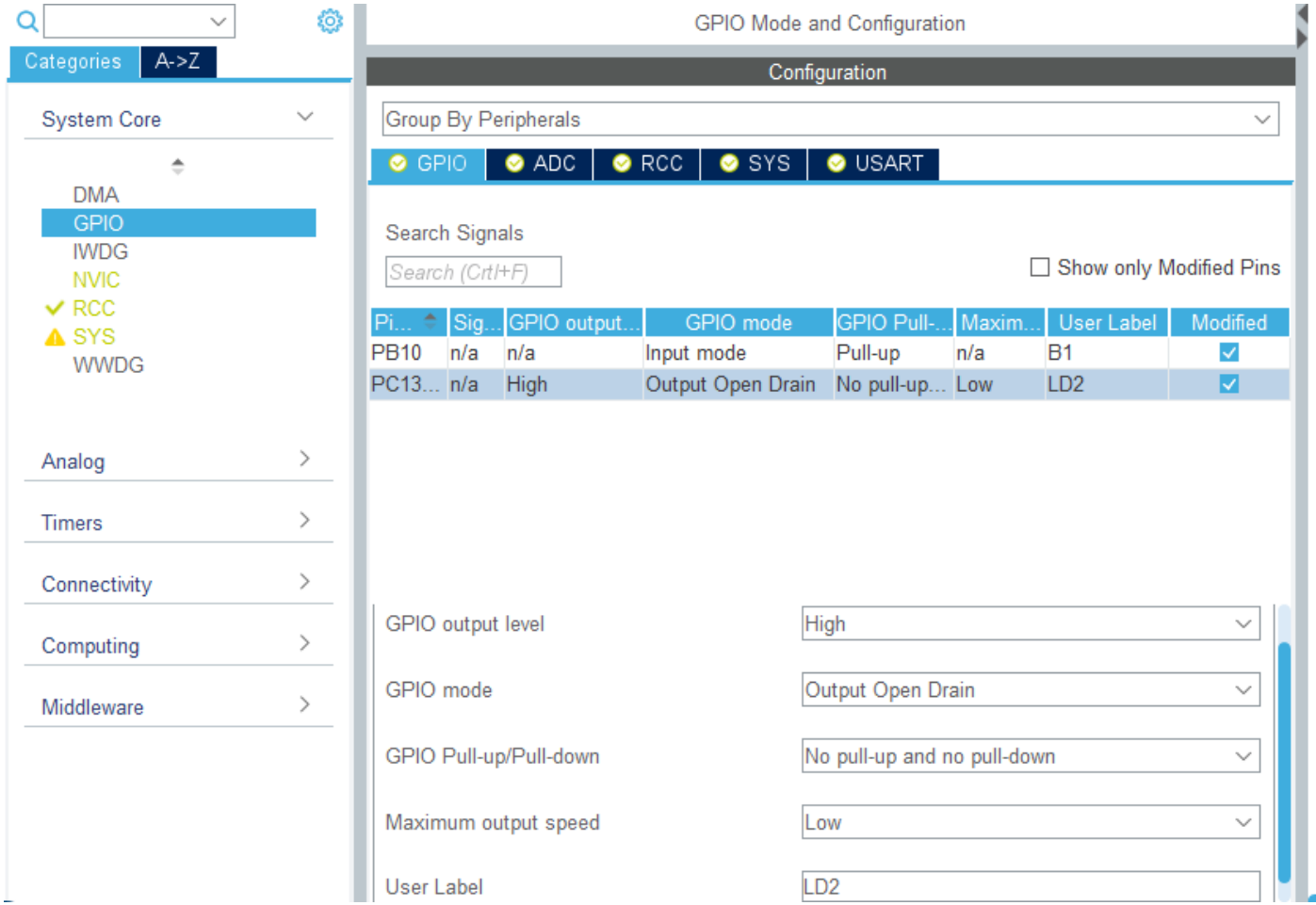
F103_ADC2 projekt

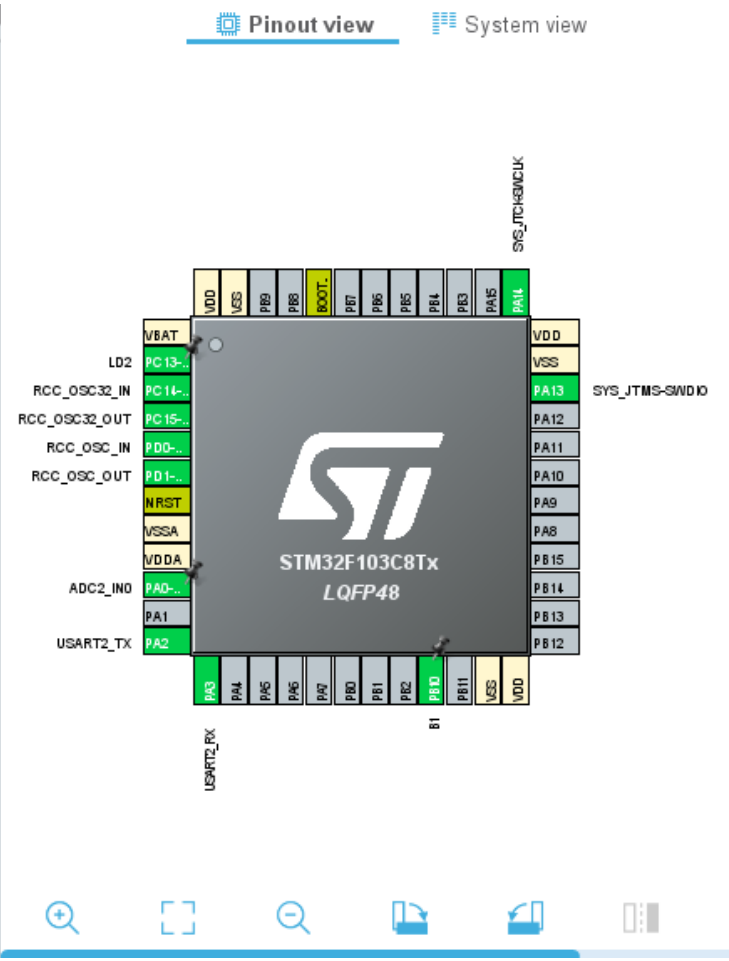
- Soros átalakító bekötése **PA2** ⟶ FT232 **Rx, PA3** ⟵ FT232 **Tx**
- A B1 nyomógomb PB10-re, az analóg hőmérő PA0-ra csatlakozik **MCP9700** 0 ℃-on VOUT = 500 mV, meredekség = 10 mV/℃



F103_ADC2 projekt: GPIO konfigurálás

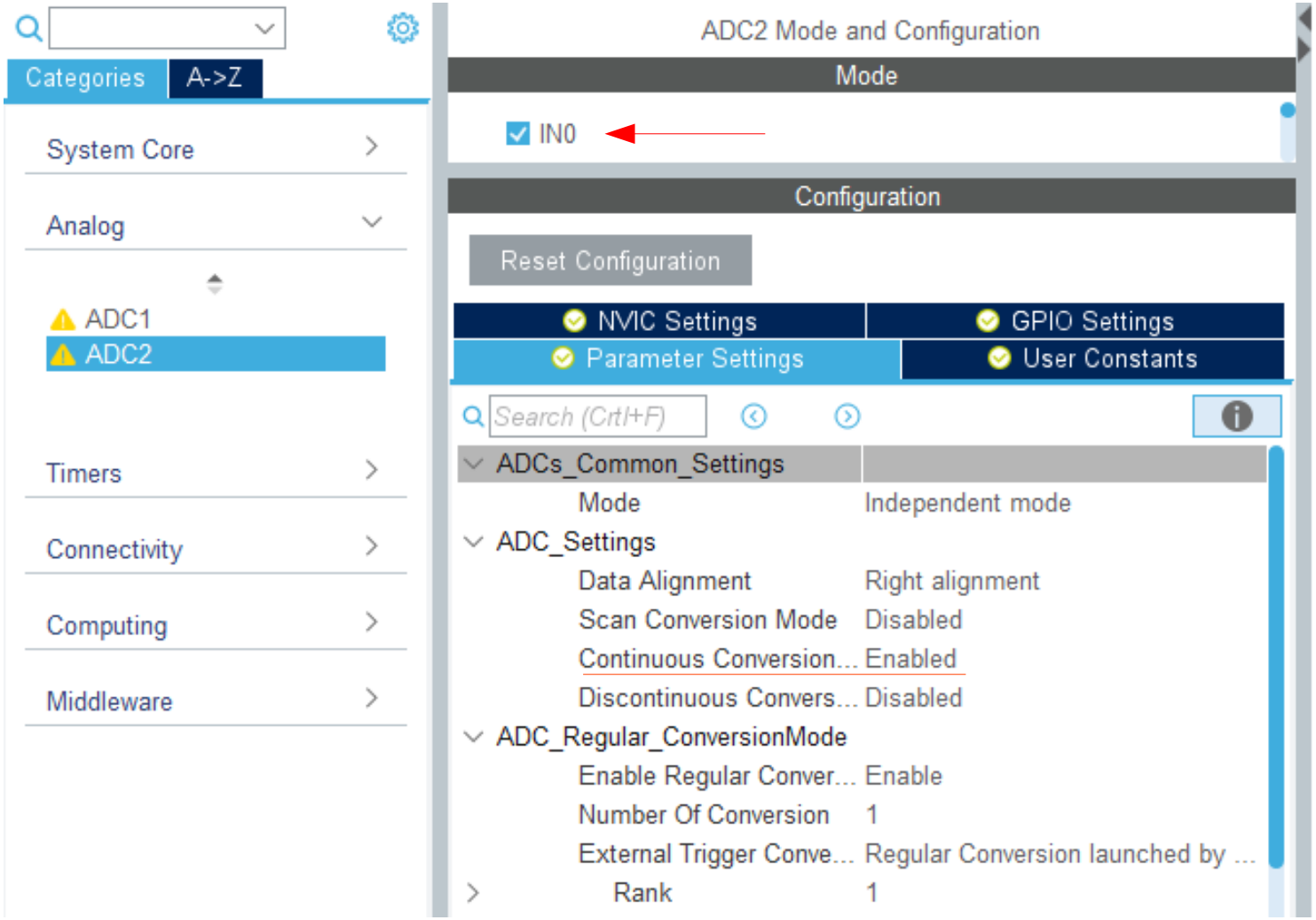
- **PC13** konfigurálása **LD2** néven, nyitott nyelőelektródás digitális kimenet, induló állapot: magas szint (nem világít)
- **PB10** konfigurálása **B1** néven, digitális bemenetként, belső felhúzással

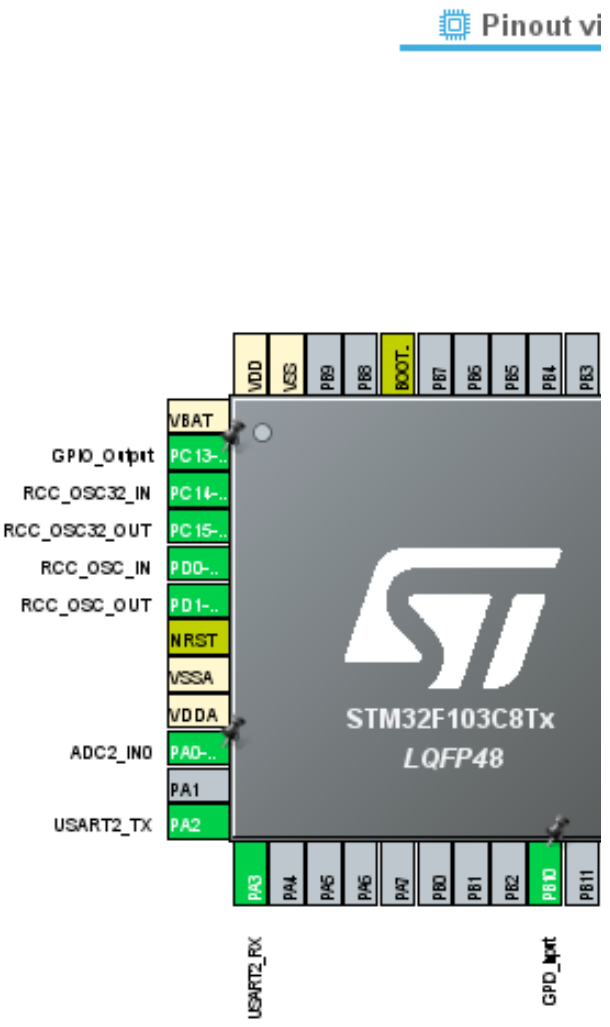




F103_ADC2 projekt: ADC2 konfigurálása

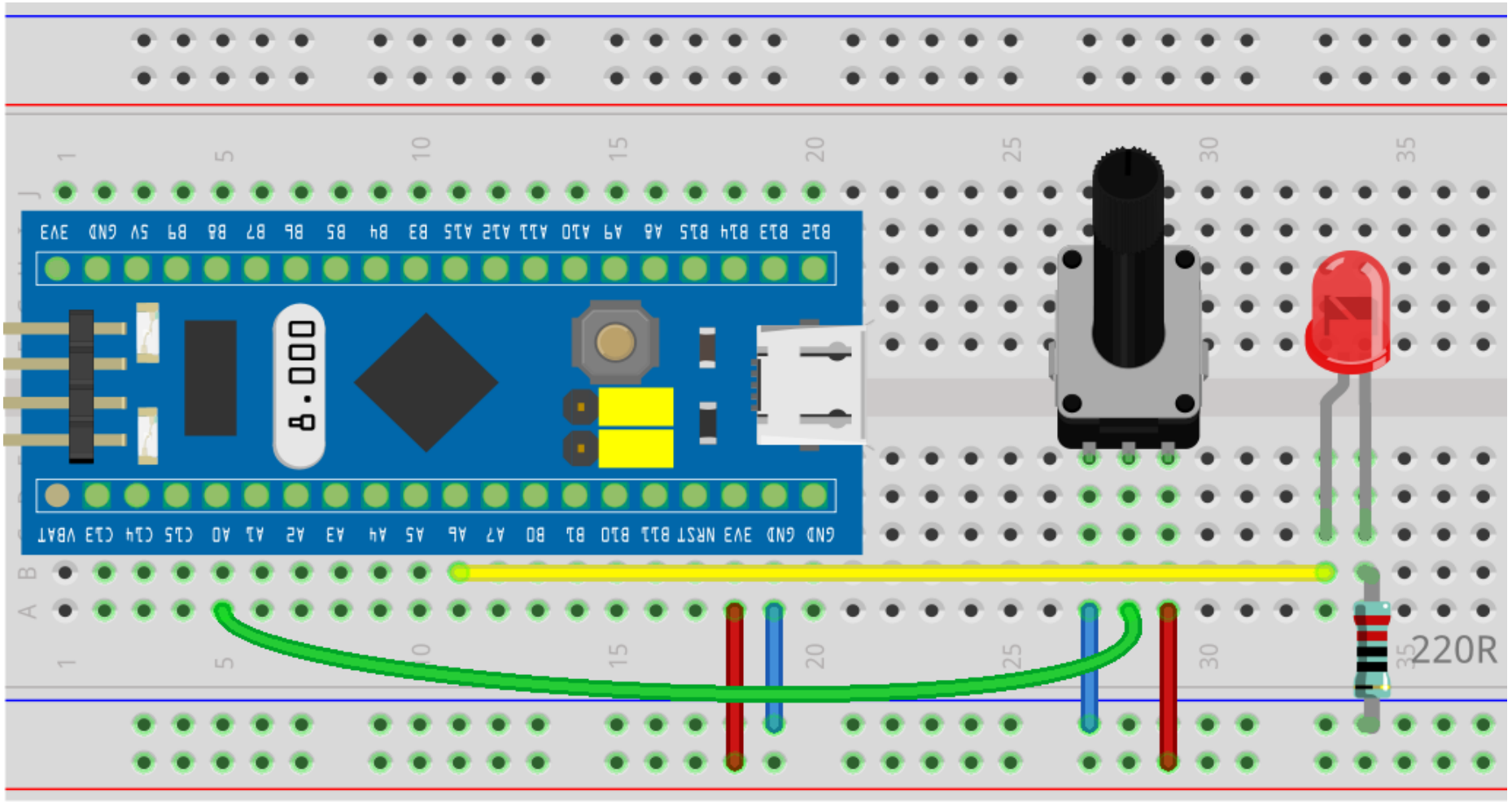
- IN0 csatorna kiválasztása, folytonos konverzió engedélyezése, esetleg hosszabb mintavételezési idő választása





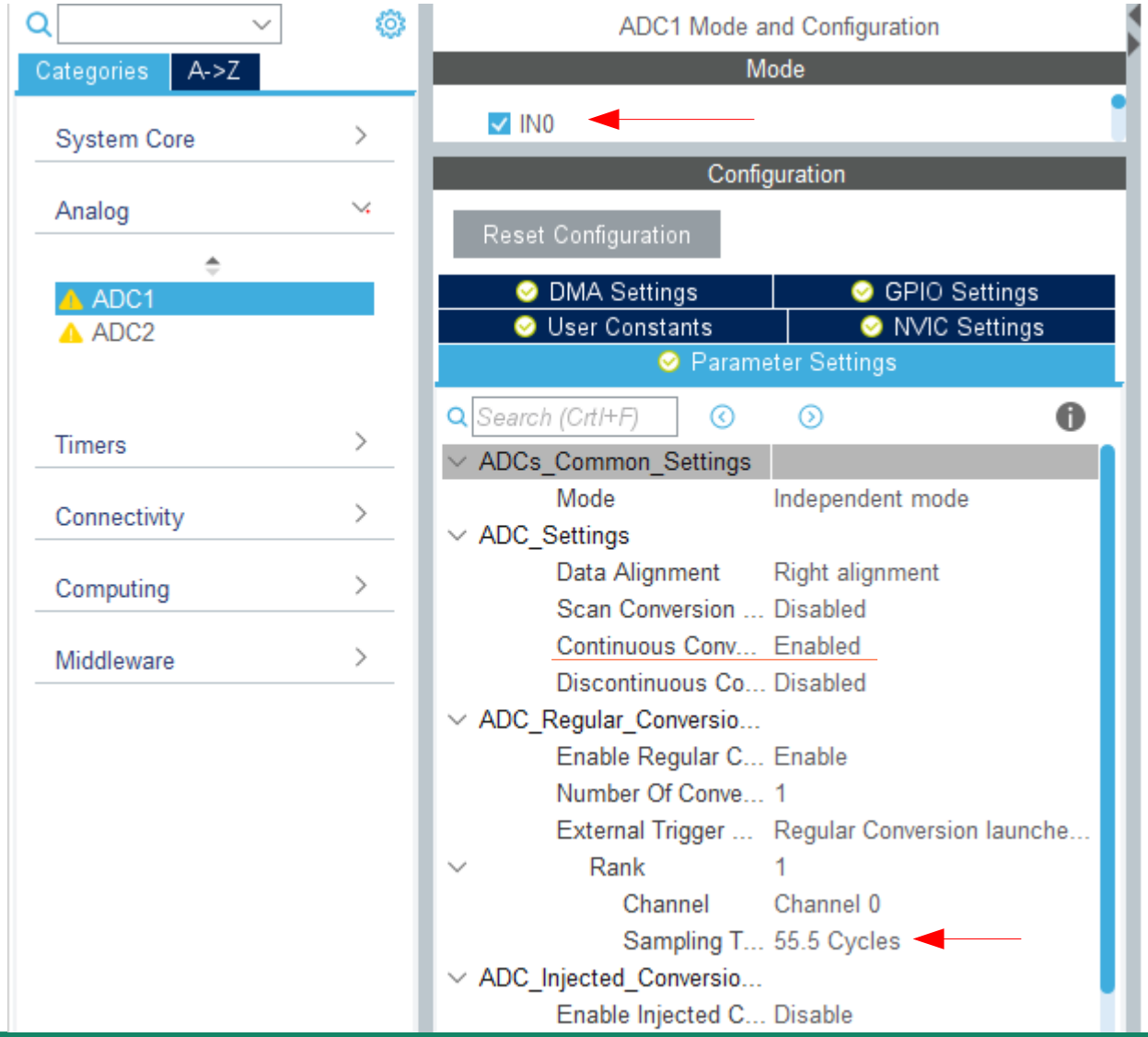
F103_ADC_in projekt

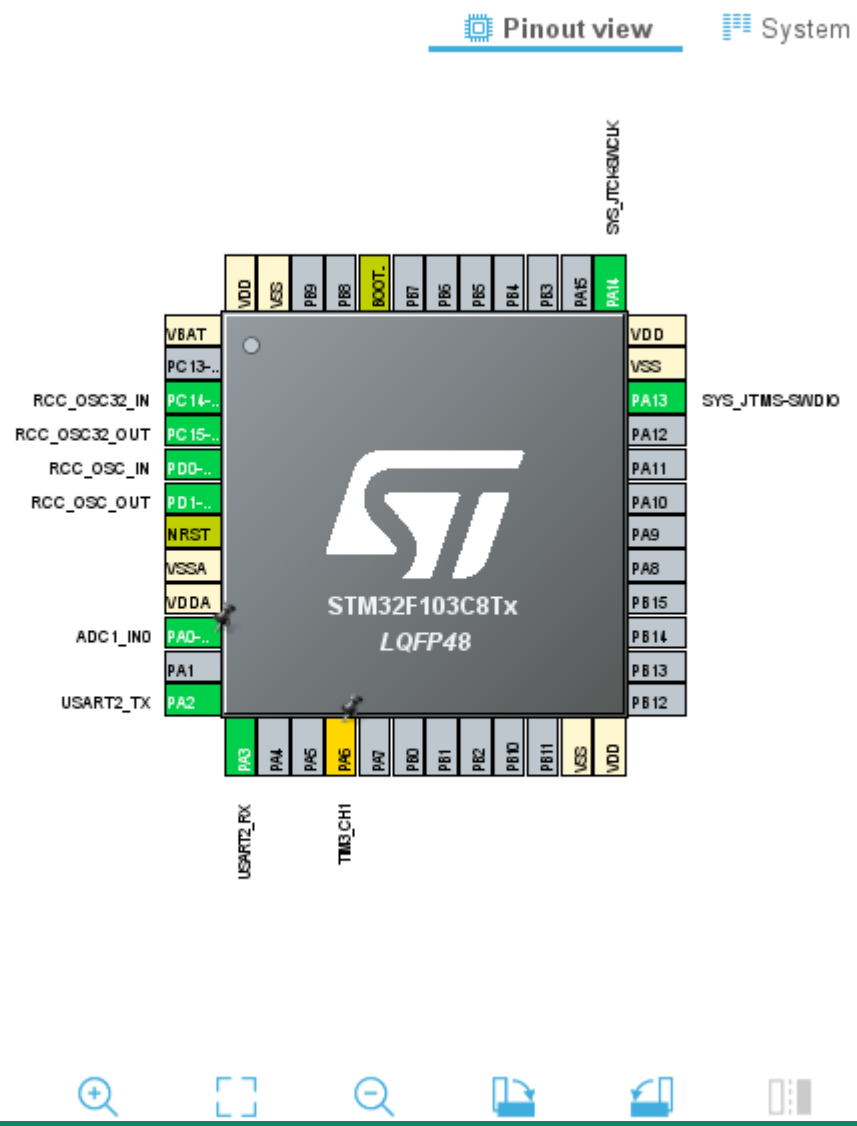
- A potméter csúszkája **PA0**-ra, az általa vezérelt LED pedig **PA6**-ra csatlakozik (a Blue pill kártya itt el van forgatva!)



F103_ADC_in projekt: ADC1 konfigurálása

- **IN0** csatorna kiválasztása, folytonos konverzió engedélyezése, hosszabb mintavételezési idő választása, megszakítás engedélyezése





F103_ADC_in projekt: TIM3 CH1 konfigurálása

- **Timer3** paraméterek beállítása:

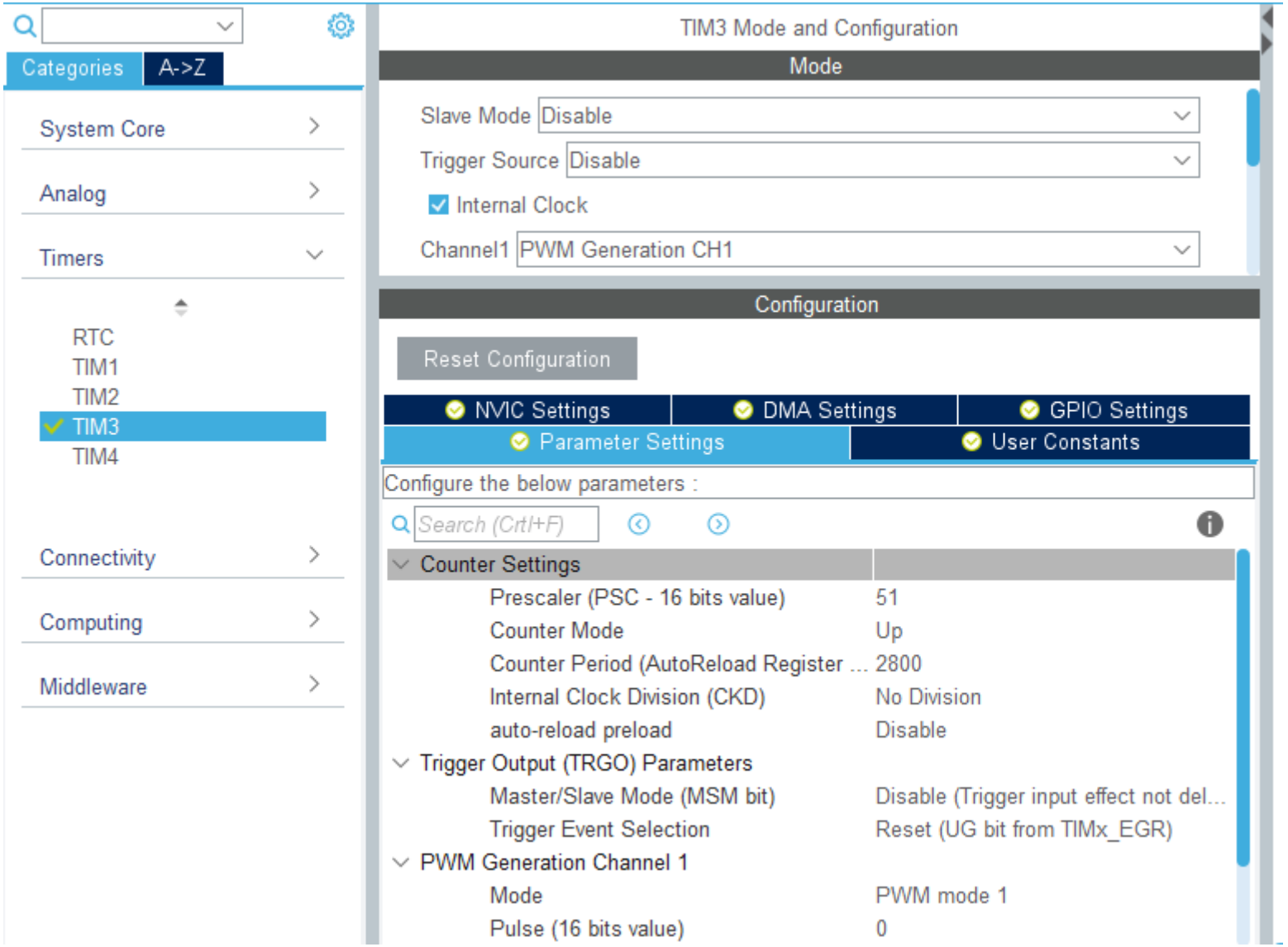
  Belső órajel

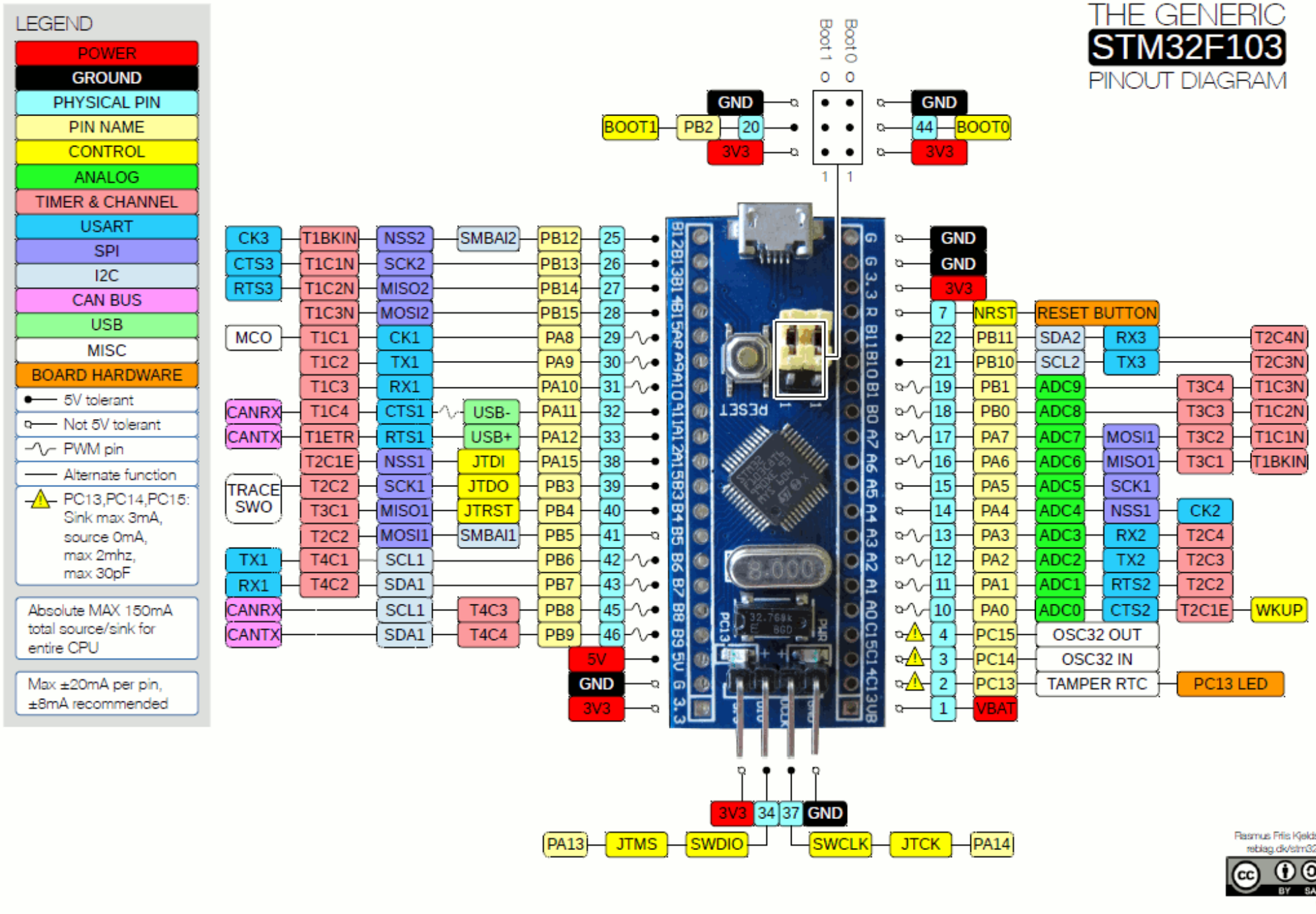
  CH1 PWM mód
- Időzítések:

  Prescaler : 51

  Periódus: 2800
- CH1 mód:

  PWM mode 1




THE GENERIC
STM32F103
PINOUT DIAGRAM
LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
Boot1
Boot0
GND
BOOT1
PB2
20
3V3
44
BOOT0
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T3C4
T3C3
T3C2
T3C1
CK2
T2C4
T2C3
T2C2
T2C1E
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
PC13 LED
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Fris Kjeldsen
reblag.dk/stm32
V1.0