STM32 mikrovezérlők programozása STM32CubeIDE környezetben

5. Többcsatornás ADC mérések, DMA használata

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:

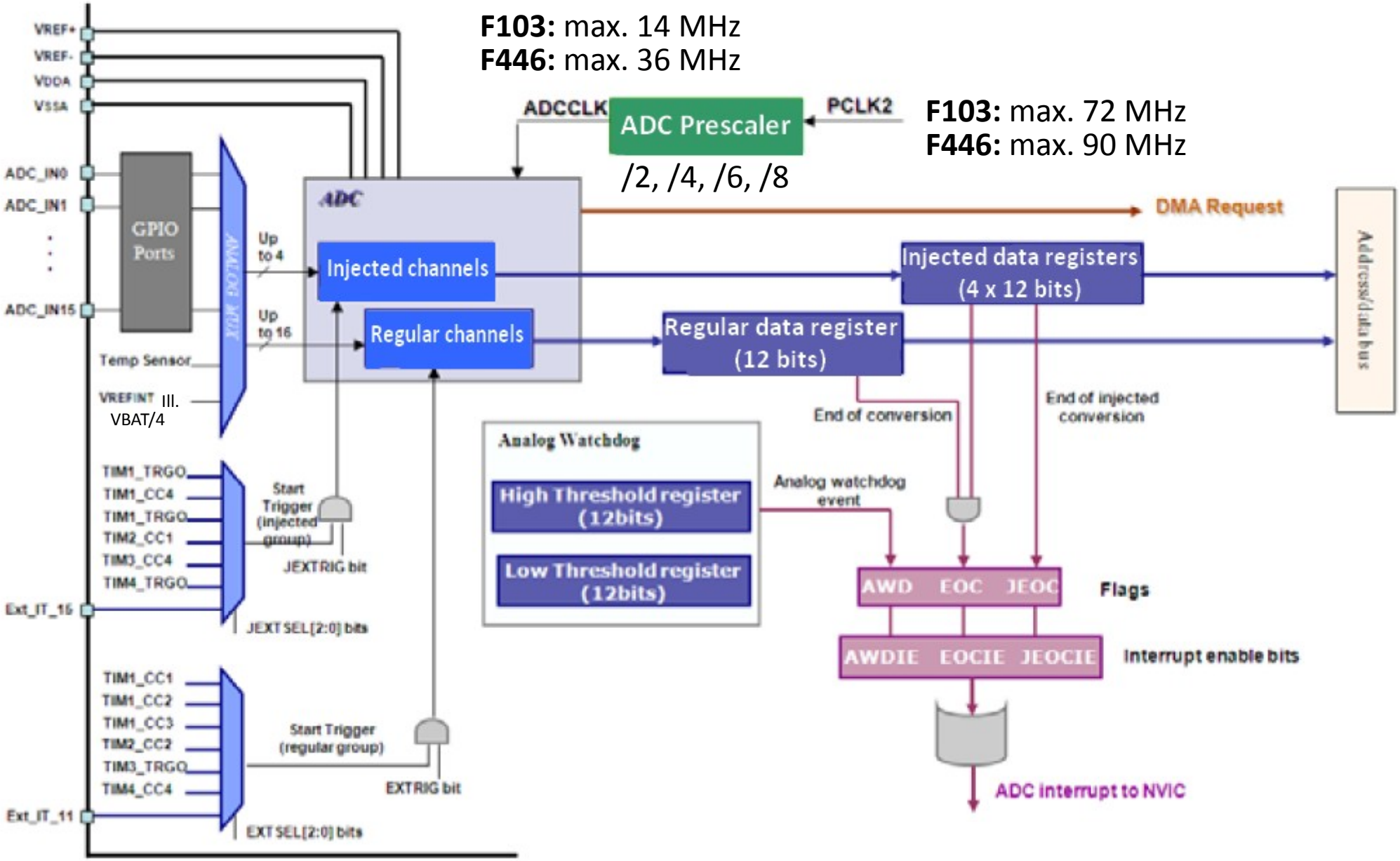
- **STM32F103C8 adatlap és termékinfo**
- **STM32F103 Family Reference Manual**
- **STM32F446RE adatlap és termékinfo**
- **STM32F446 Family Reference Manual**
- STmicro: **Description of STM32F4 HAL and low-layer drivers**
- STmicro: **AN3116 – STM32's ADCs modes and their application**

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



Az ADC-k üzemmódjainak áttekintése

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

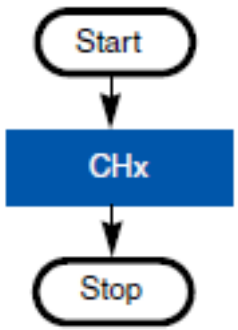
Ajánlott olvasmány:

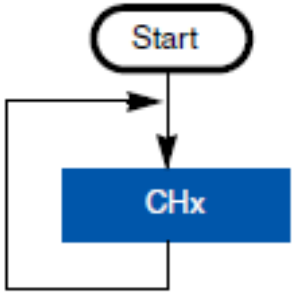
AN3116 – STM32's ADCs modes and their application

Független módok

Egy csatorna

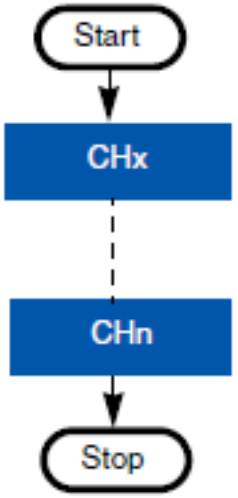
szingli konverzió folyamatos konverzió

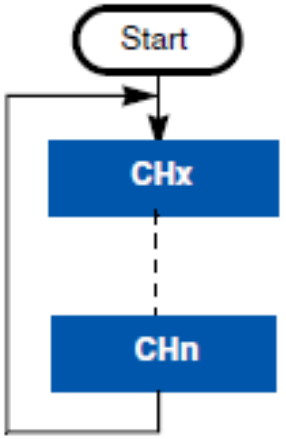




Pásztázás (több csatorna) (csak DMA-val) ✓

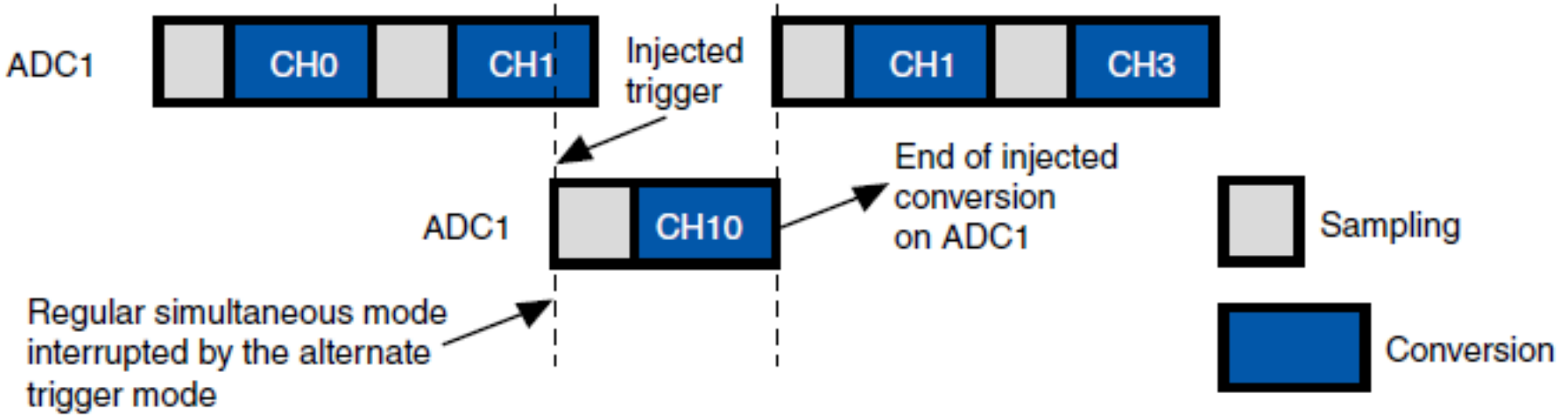
szingli konverzió folyamatos konverzió





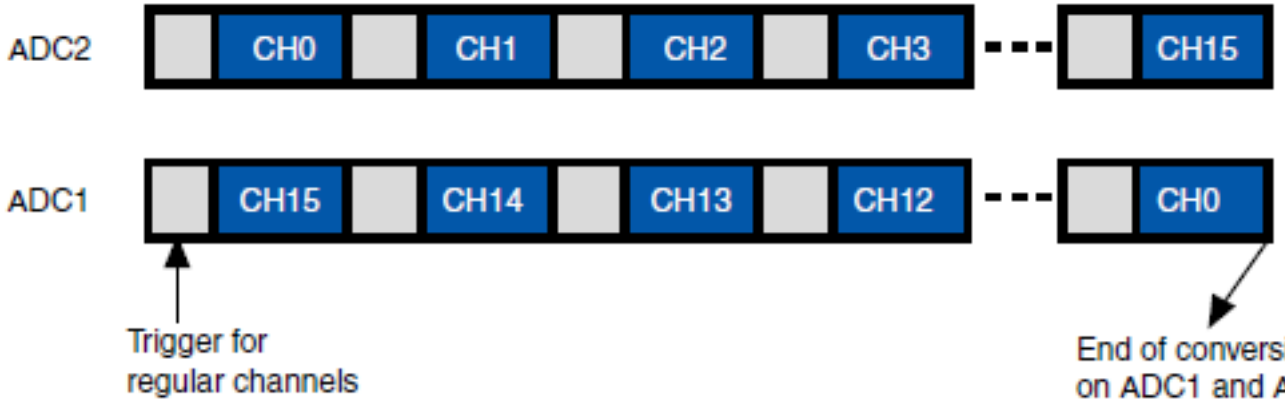
Injektált konverziós mód ✓

(legfeljebb 4 csatorna injektálható)

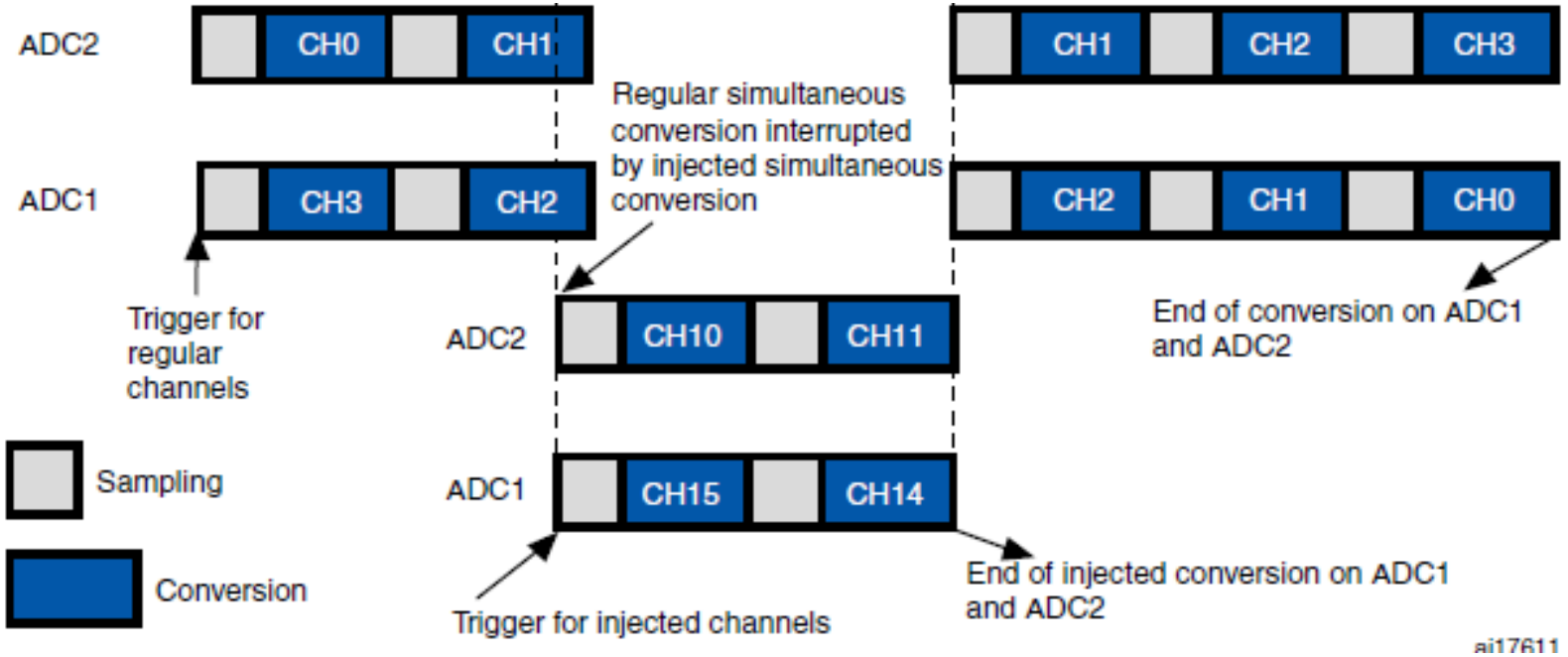


Duál/Tripla szimultán mód

- Szimultán módban több ADC egyszerre mintavételez és konvertál

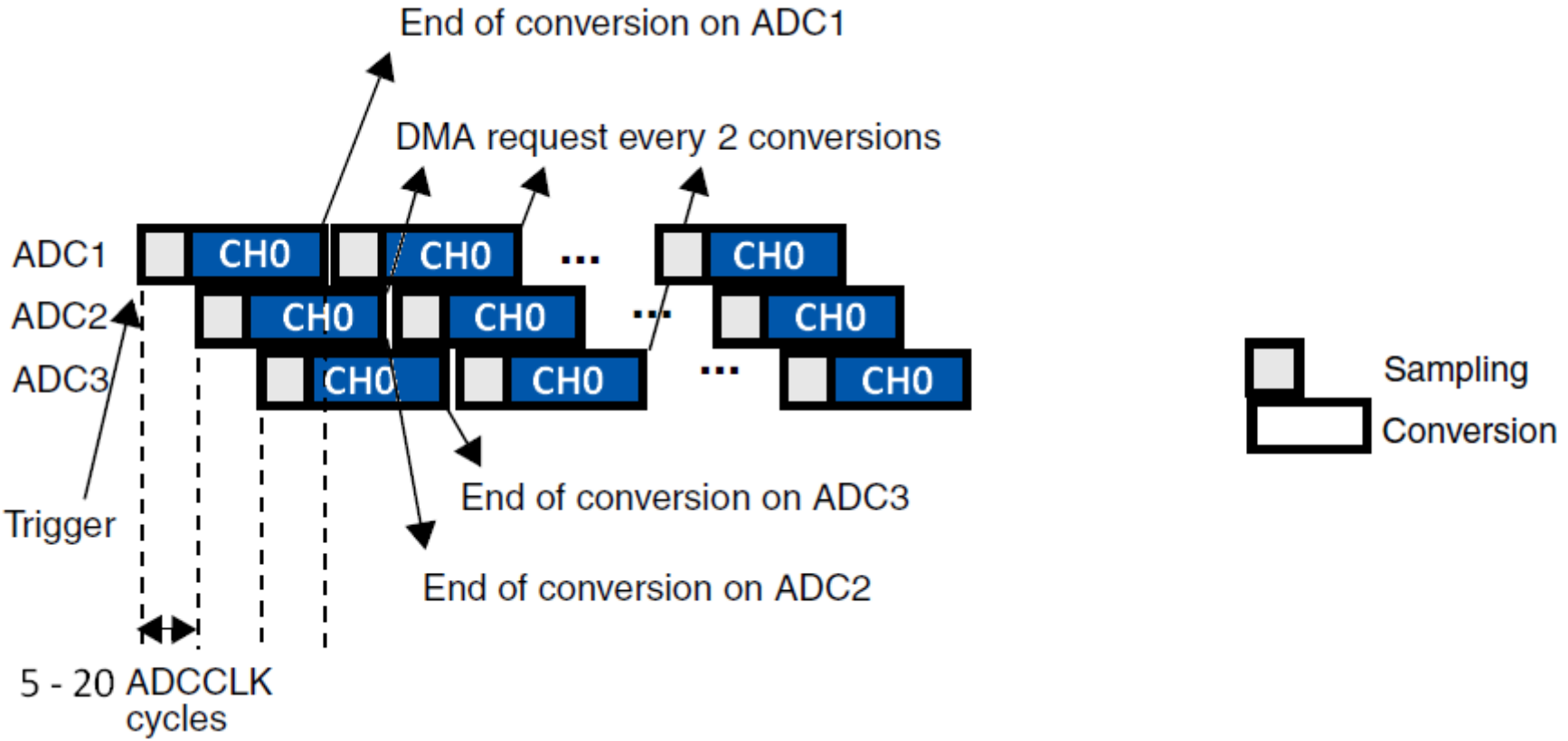


- Injektált csatornákkal is kombinálható



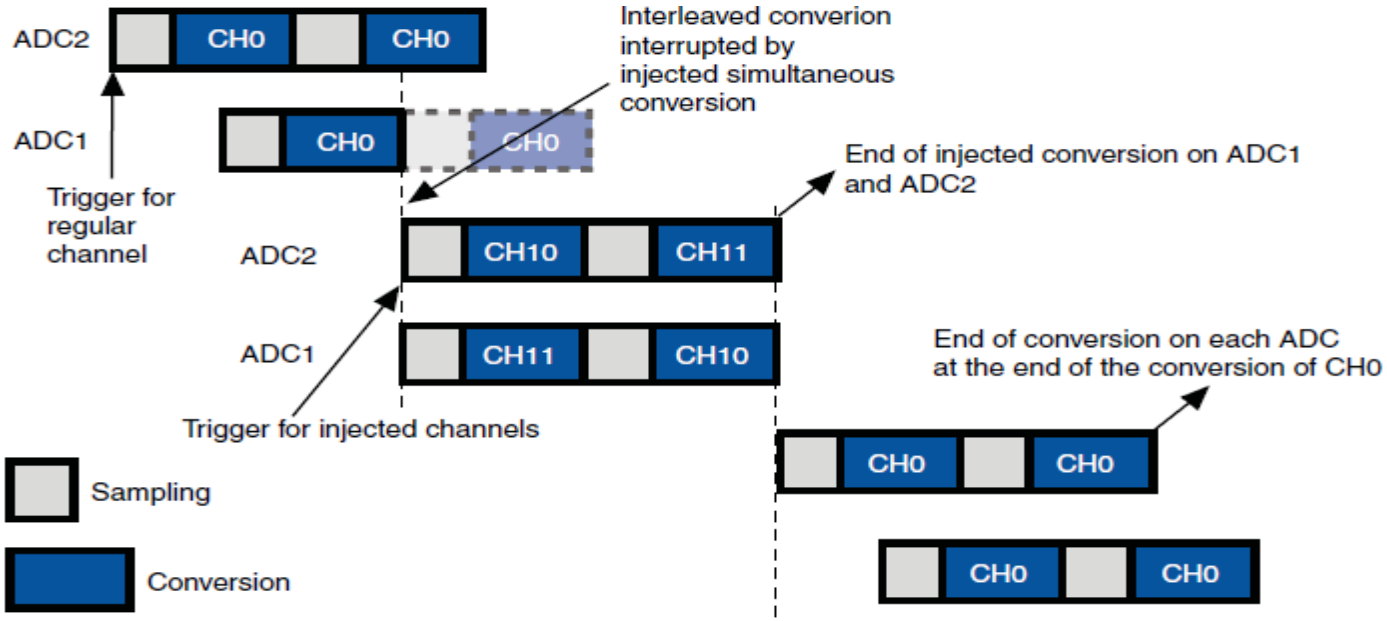
Duál/Tripla átlapolásos mód ✓

- Az ADC-k indítása 5-20 órajelciklus eltolással indul
- **ADC1** a master, ebben konfigurálhatók a paraméterek
- Adatkiolvasás 2-2 konverziónként egy közös, 32 bites regiszterből
- **STM32F103C8:** csak duál mód, 7 vagy 14 ciklus eltolással



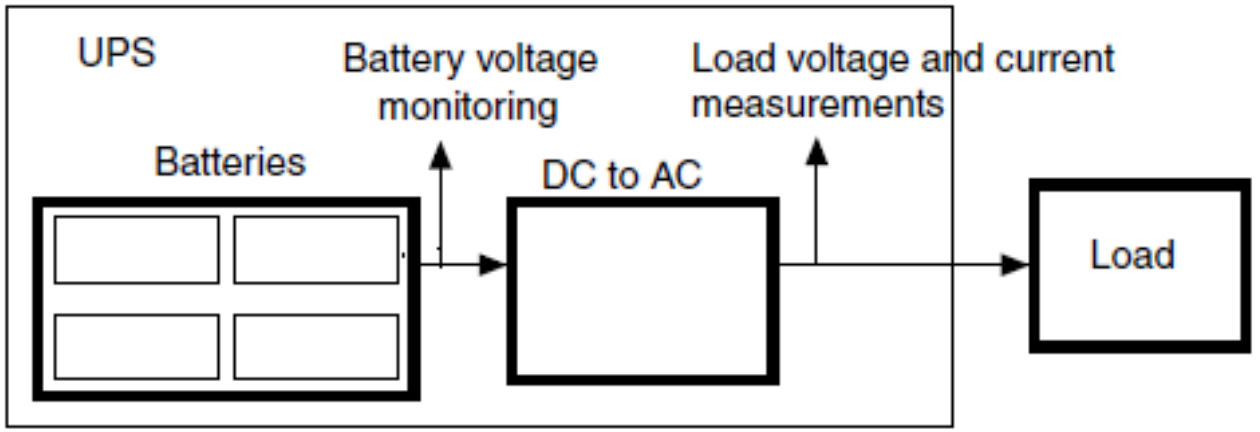
Duál, szimultán injektált, átlapolásos mód

- A reguláris csatorná(k)ban átlapolásos, az injektált csatornákban szimultán méréssel



Alkalmazási példa: UPS

- Reguláris csatorna: **VBAT** monitorozás
- Injektált csatornák: fogyasztó V és I mérés



Példaprogramok STM32 446RE mikrovezérlőre

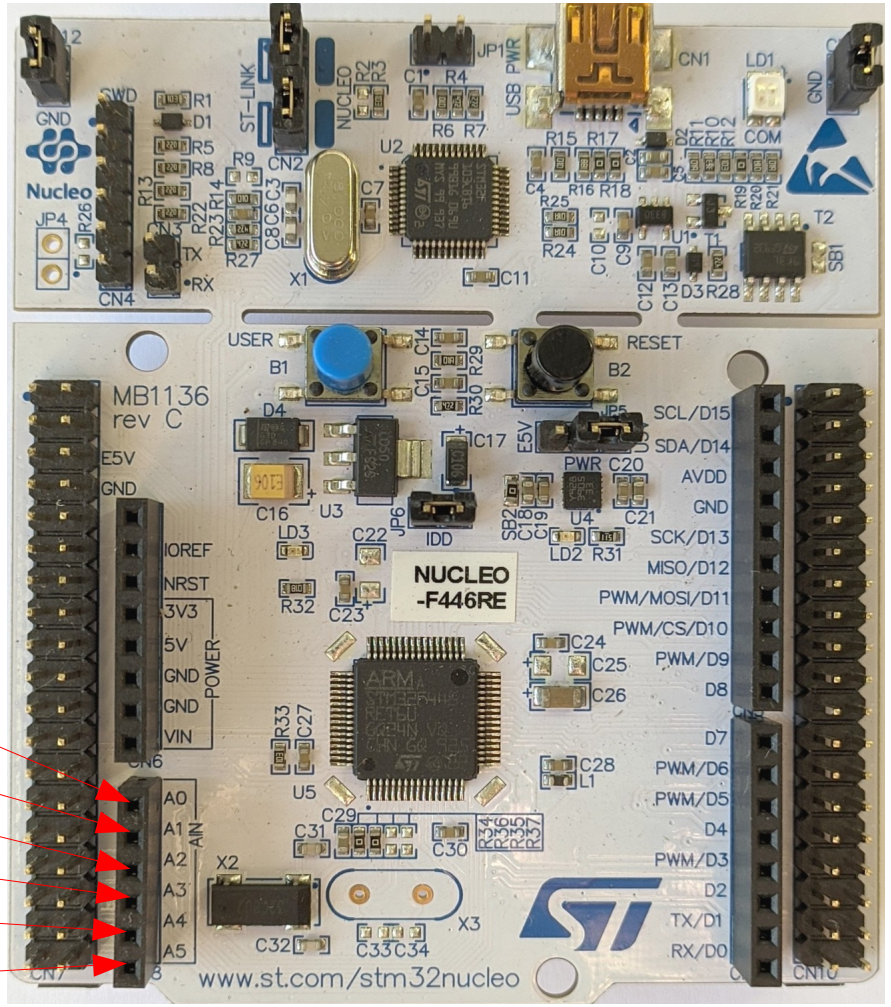
- **F446RE_ADC_injected:** mérés egy reguláris és négy injektált csatornában – szoftveres indításokkal (PA0, PA1, PA4 bemenetek, valamint a belső hőmérő és a belső referencia jelét mérjük)
- **F446RE_ADC_injected_avg:** ugyanaz, mint az előző projekt, csak 1000 mérésenként átlagolunk
- **F446RE_ADC_DMA:** mérés négy reguláris csatornában, DMA átvitellel (PA0, PA1, PA4 bemenetek, valamint a belső hőmérő jelét mérjük)
- **F446RE_ADC_interleaved:** Tripla átlapolásos módú mérés, három ADC-vel (a PA0 bemenet jelét mérjük, 7.2 Msamples/s mintavételezéssel)
  Megjegyzés: az utolsó projektben a szokásostól eltérő rendszer-órajel beállítást kell használnunk: SYSCLK=144MHz, PCLK1=36MHz, PCLK2=72MHz, ADCCLK=36MHz

Az analóg bemenetek kiosztása

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

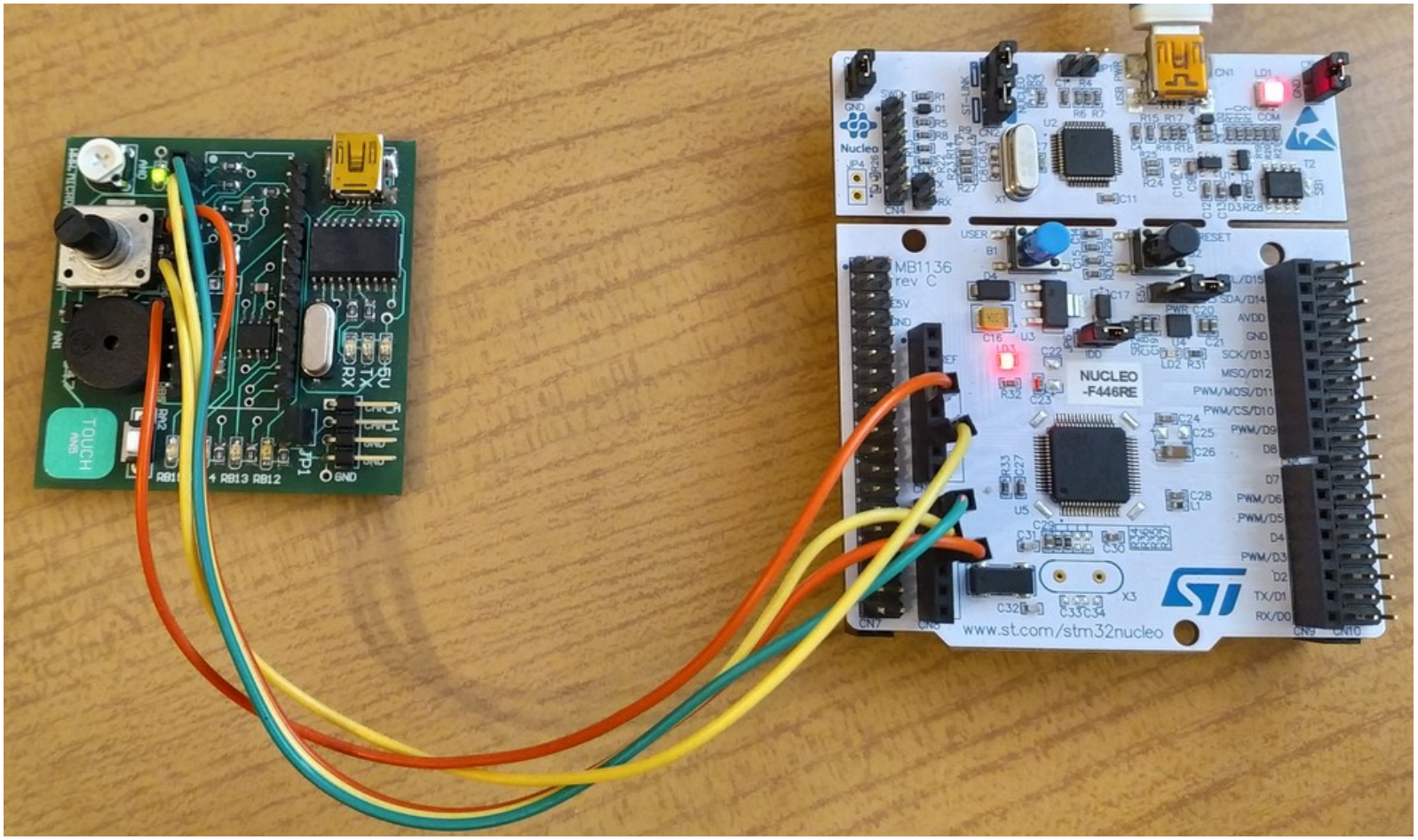
I/O Pin	ADC	I/O Pin	ADC
PA0	AIN[0]	PC0	AIN[10]
PA1	AIN[1]	PC1	AIN[11]
PA2	AIN[2]	PC2	AIN[12]
PA3	AIN[3]	PC3	AIN[13]
PA4	AIN[4]	PC4	AIN[14]
PA5	AIN[5]	PC5	AIN[15]
PA6	AIN[6]		
PA7	AIN[7]		
PB0	AIN[8]		
PB1	AIN[9]		

PA0 IN[0]
PA1 IN[1]
PA4 IN[4]
PB0 IN[8]
PC1 IN[11]
PC0 IN[10]



A kapcsolási elrendezés

- A mérendő jeleket itt egy **MicrostickPlus** kártyáról vettük, de a kapcsolás próbapanelon is megépíthető, ill. más forrás jele is mérhető
- **PA0** ← potméter, **PA1** ← TC1047 hőmérő, **PA4** ← 2.5 V referencia fentieken kívül a belső hőmérő és a belső referencia jelét mérjük



F446RE_ADC_injected

- Az első projektben egy reguláris csatorna és 4 db injektált csatorna jelét mérjük **ADC1** segítségével, a méréseket szoftveresen indítjuk
- Az injektált csatornák segítségével **DMA** nélkül is végezhetünk pásztázó mérést (legfeljebb négy csatornában), mert mindegyik injektált csatornához külön adatregiszter tartozik
- A feszültségre, ill. hőmérsékletre átszámolt értékeket az **UART2** porton (**PA2, PA3** kivezetéseken) keresztül kiíratjuk
- **A felhasznált csatornák:**
  0. **IN[0]** (PA0) lesz a reguláris csatorna (potméter jele)
  1. **IN[1]** (PA1) lesz az 1. injektált csatorna (külső hőmérő jele)
  2. **IN[4]** (PA4) lesz a 2. injektált csatorna (2.5 V külső referencia)
  3. **Vrefint** lesz a 3. injektált csatorna (1.2 V belső referencia)
  4. **temp_sensor** lesz a 4. injektált csatorna (belső hőmérő)
- A program fő előnye az egyszerűség és az áttekinthetőség

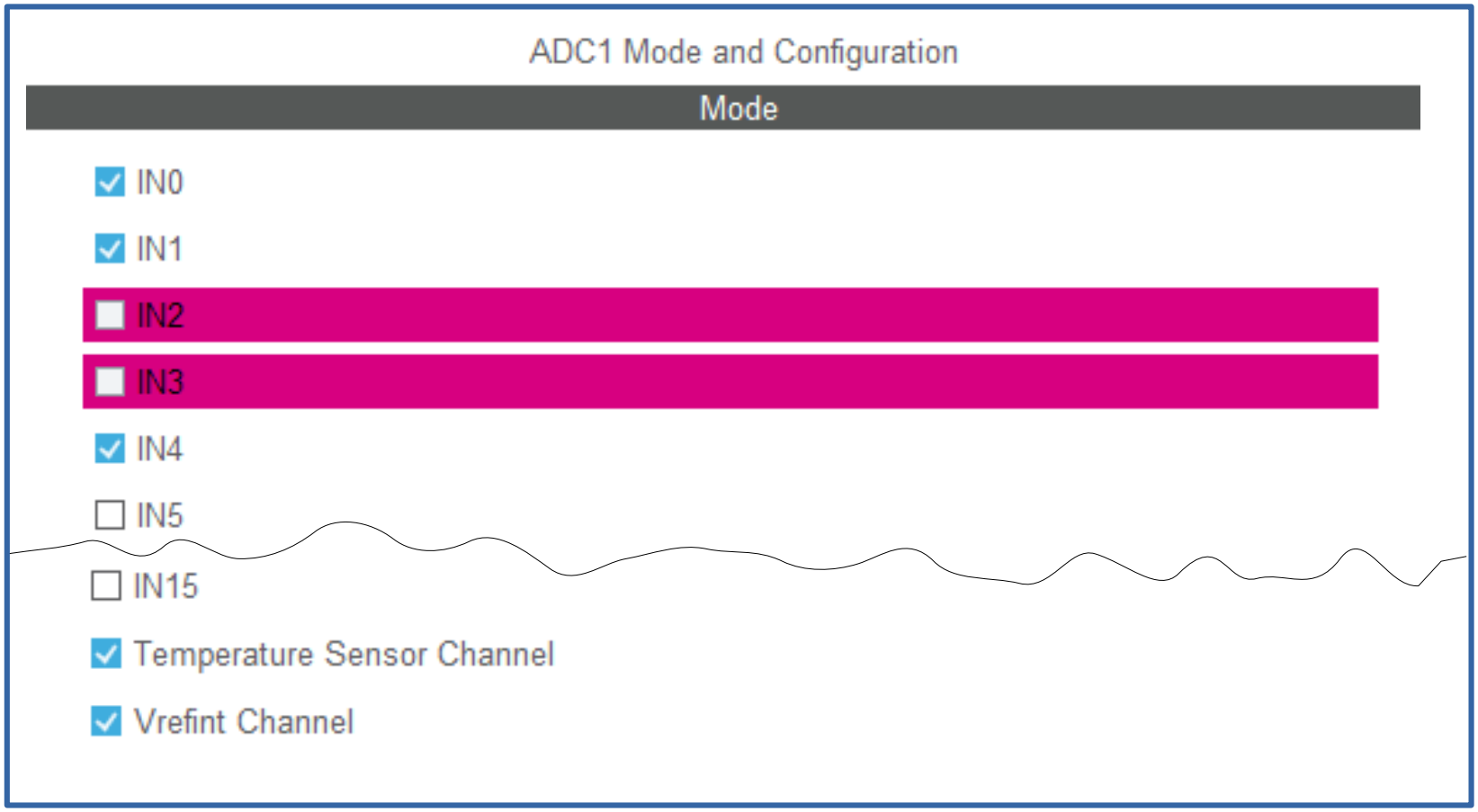
Injektált csatornák kezelése lekérdezéssel

- Az injektált csatornák konverziójának indítása történhet:
  - Automatikusan (a reguláris csatornák konverziója végén)
  - Szoftveres indítással *(mi most ezt fogjuk használni)*
  - Hardveres triggereléssel

Csatorna	Reguláris (1)	Injektált (1-4)
HAL package	**HAL_ADC**	**HAL_ADCEx**
Konverzió indítása	`HAL_ADC_Start(&hadc1)`	`HAL_ADCEx_InjectedStart(&hadc1)`
Várakozás a konverzió végére	`HAL_ADC_PollForConversion (&hadc1, timeout)`	`HAL_ADCEx_InjectedPollFor Conversion(&hadc1, timeout)`
Eredmény kiolvasása	`val=HAL_ADC_GetValue(&hadc1)`	`val=HAL_ADCEx_Injected GetValue(&hadc1,i)  (i=1..4)`
Konverzió leállítása	`HAL_ADC_Stop(&hadc1)`	`HAL_ADCEx_InjectedStop(&hadc1)`
Csatorna konfigurálása	`HAL_ADC_ConfigChannel (&hadc1, sConfig)`	`HAL_ADCEx_InjectedConfigChannel (&hadc1, sConfigInjected)`

ADC1 konfigurálása: csatornák kiválasztása

- ADC1 konfigurálásánál az alábbi csatornákat kell aktívvá tenni: IN0, IN1, IN4, Temperature Sensor, Vrefint
- Amint látjuk, **IN2** (PA2) és **IN3** (PA3) nem használható, mert ezeket a kivezetéseket **UART2** lefoglalja

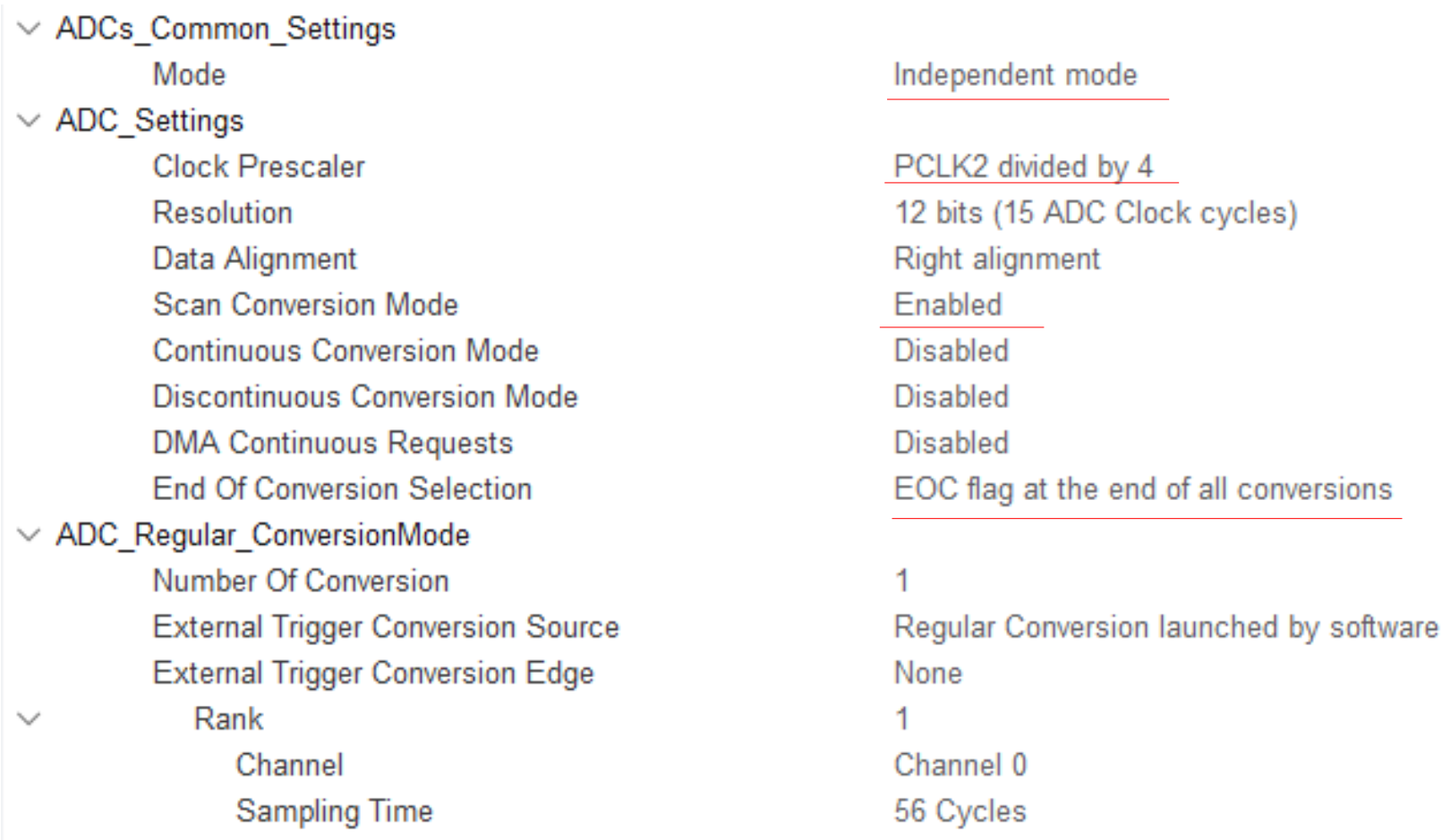


ADC1 konfigurálás: csatornák konfigurálása

ADC_Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 0
Sampling Time	56 Cycles
ADC_Injected_ConversionMode	
Number Of Conversions	4
External Trigger Source	Injected Conversion launched by software
External Trigger Edge	None
Injected Conversion Mode	None
Injected Rank	1
Channel	Channel 1
Sampling Time	56 Cycles
Injected Offset	0
Injected Rank	2
Channel	Channel 4
Sampling Time	56 Cycles
Injected Offset	0
Injected Rank	3
Channel	Channel Vrefint
Sampling Time	480 Cycles
Injected Offset	0
Injected Rank	4
Channel	Channel Temperature Sensor
Sampling Time	480 Cycles
Injected Offset	0

ADC1 konfigurálás: paraméterek beállítása

- A *Scan Conversion* mód az egyetlen reguláris csatornához nem kellene, de az injektált csatornák pásztázása miatt *Enabled* kell
- Az injektált csatornáknál csak a sorozat végén kérünk *EOC* jelet



F446RE_ADC_injected: main.c 2/1

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

A kiíratás miatt kellenek...

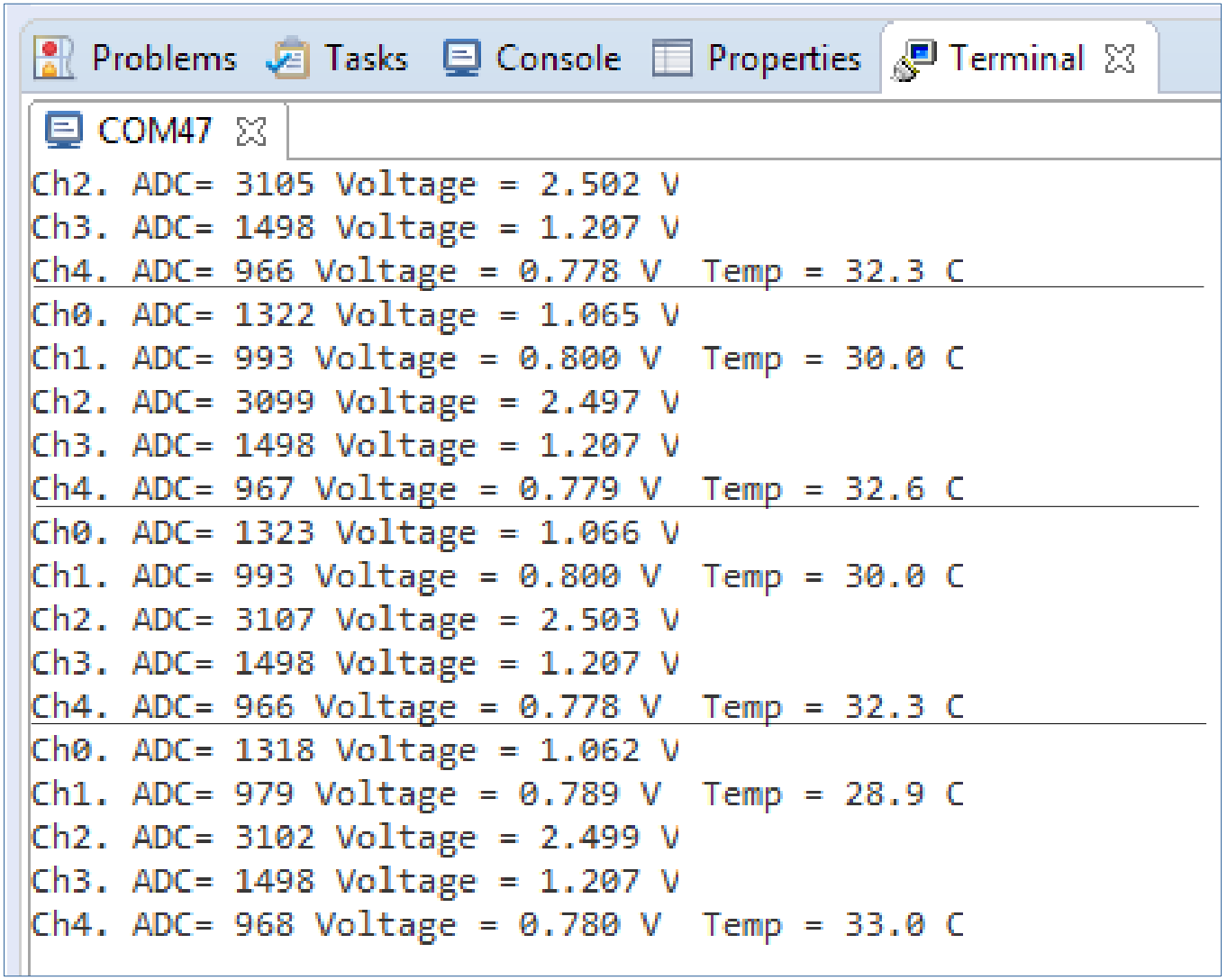
```c
ADC_HandleTypeDef hadc1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);

void display_data(int Nadc, float voltage, int i)  {
  char msg[80];
  float tempC;
  if (i == 1) {
    tempC = (voltage - 0.5) / 0.01;
    sprintf(msg, "Ch1. ADC= %d Voltage = %5.3f V Temp = %4.1f C\r\n",Nadc,voltage,tempC);
  } else if (i == 4) {
    tempC = ((voltage - 0.760) / 0.0025) + 25.0;
    sprintf(msg, "Ch4. ADC= %d Voltage = %5.3f V Temp = %4.1f C\r\n",Nadc,voltage,tempC);
  } else {
    sprintf(msg, "Ch%d. ADC= %d Voltage = %5.3f V\r\n", i, Nadc,voltage);
  }
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
}
```

F446RE_ADC_injected: main.c 2/2

```
int main(void) {
  float voltage;
  int val;

  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();
  while (1) {
      HAL_ADC_Start(&hadc1);                      // ADC1 indítása
      HAL_ADC_PollForConversion(&hadc1, 100); // EOC jelre várunk
      val = HAL_ADC_GetValue(&hadc1);          // A reguláris csatorna kiolvasása
      voltage = (float)val * 3.3 / 4096;       // Vref = 3.3 V, resolution = 4096
      display_data(val, voltage, 0);           // Az eredmény kiíratása
      HAL_ADCEx_InjectedStart(&hadc1);         // ADC1 injektált csat. konverzió indítása
      HAL_ADCEx_InjectedPollForConversion(&hadc1, 100); // EOC jelre várunk
      for(int i = 1; i<5; i++) {
          val = HAL_ADCEx_InjectedGetValue(&hadc1,i);   // Az i-edik csatorna kiolvasása
          voltage = (float)val * 3.3 / 4096;            // Vref = 3.3 V, res. = 4096
          display_data(val, voltage, i);                // Az eredmény kiíratása
      }
//    HAL_ADC_Stop(&hadc1);                         // ADC1 leállítása
      HAL_Delay(1000);
  }
}
```

F446RE_ADC_injected: futási eredmény

- A futási eredmények erős szórást mutatnak, érdemes lenne több mérést átlagolni a zaj okozta ingadozások kiátlagolására!

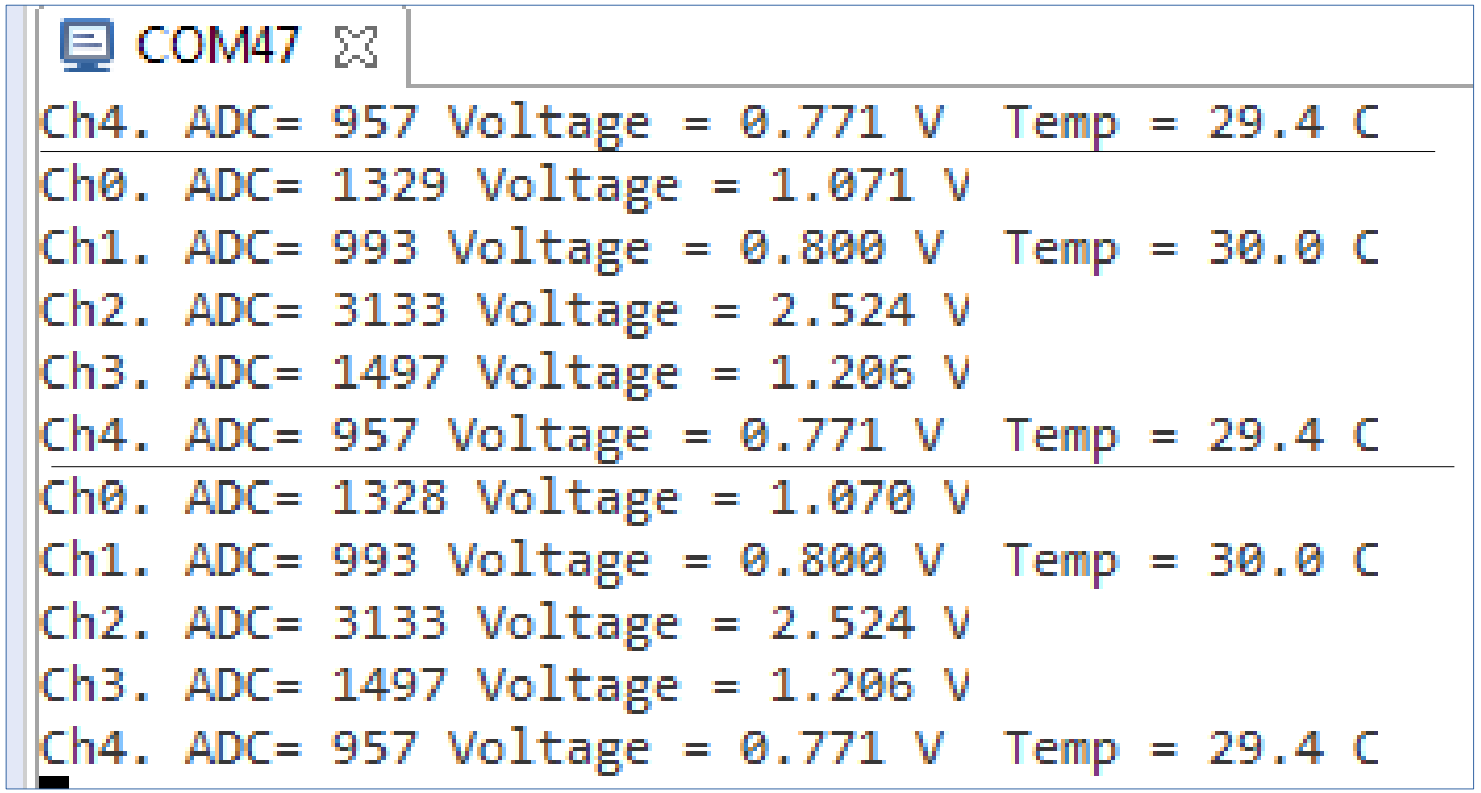


F446RE_ADC_injected_avg: main.c (részlet)

```
int main(void) {
  float voltage;
  int val;
  uint32_t data[5];
  HAL_Init(); MX_GPIO_Init(); MX_ADC1_Init(); MX_USART2_UART_Init();
  while (1) {
    for(int i = 0; i<5; i++) data[i] = 0;          // Összegző változók törlése
    for(int j=0; j<1000; j++) {
      HAL_ADC_Start(&hadc1);                       // reguláris csat. konverzió indítása
      HAL_ADC_PollForConversion(&hadc1, 100);      // EOC jelre várunk
      data[0] += HAL_ADC_GetValue(&hadc1);         // A reguláris csatorna kiolvasása
      HAL_ADCEx_InjectedStart(&hadc1);             // injektált csatornák konverzió indítás
      HAL_ADCEx_InjectedPollForConversion(&hadc1, 100);   // EOC jelre várunk
      for(int i = 1; i<5; i++) {
        data[i] += HAL_ADCEx_InjectedGetValue(&hadc1,i);  // i. csatorna kiolvasása
      }
    }
    for(int i = 0; i<5; i++) {
      val = data[i]/1000;                          // Átlagolás
      voltage = (float)val * 3.3 / 4096;           // Vref = 3.3 V, resolution = 4096
      display_data(val, voltage, i);               // Az eredmény kiíratása
    }
//  HAL_ADC_Stop(&hadc1);                          // ADC1 leállítása
    HAL_Delay(1000);
  }
}
```

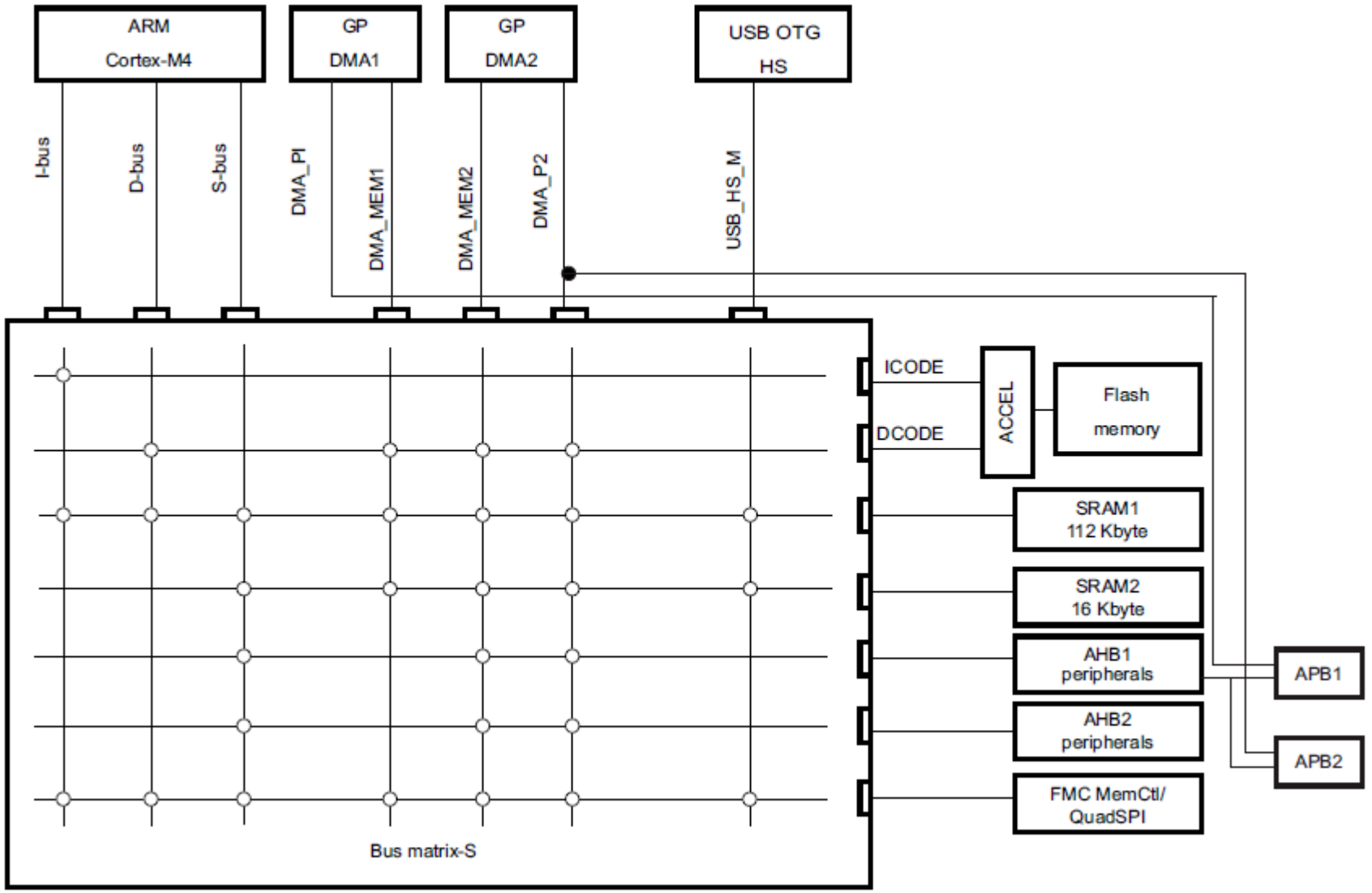
F446RE_ADC_injected_avg: futási eredmény

- A futási eredmények az alábbi ábrán láthatók
- Észrevehető, hogy sokkal kisebb lett az adatok szórása

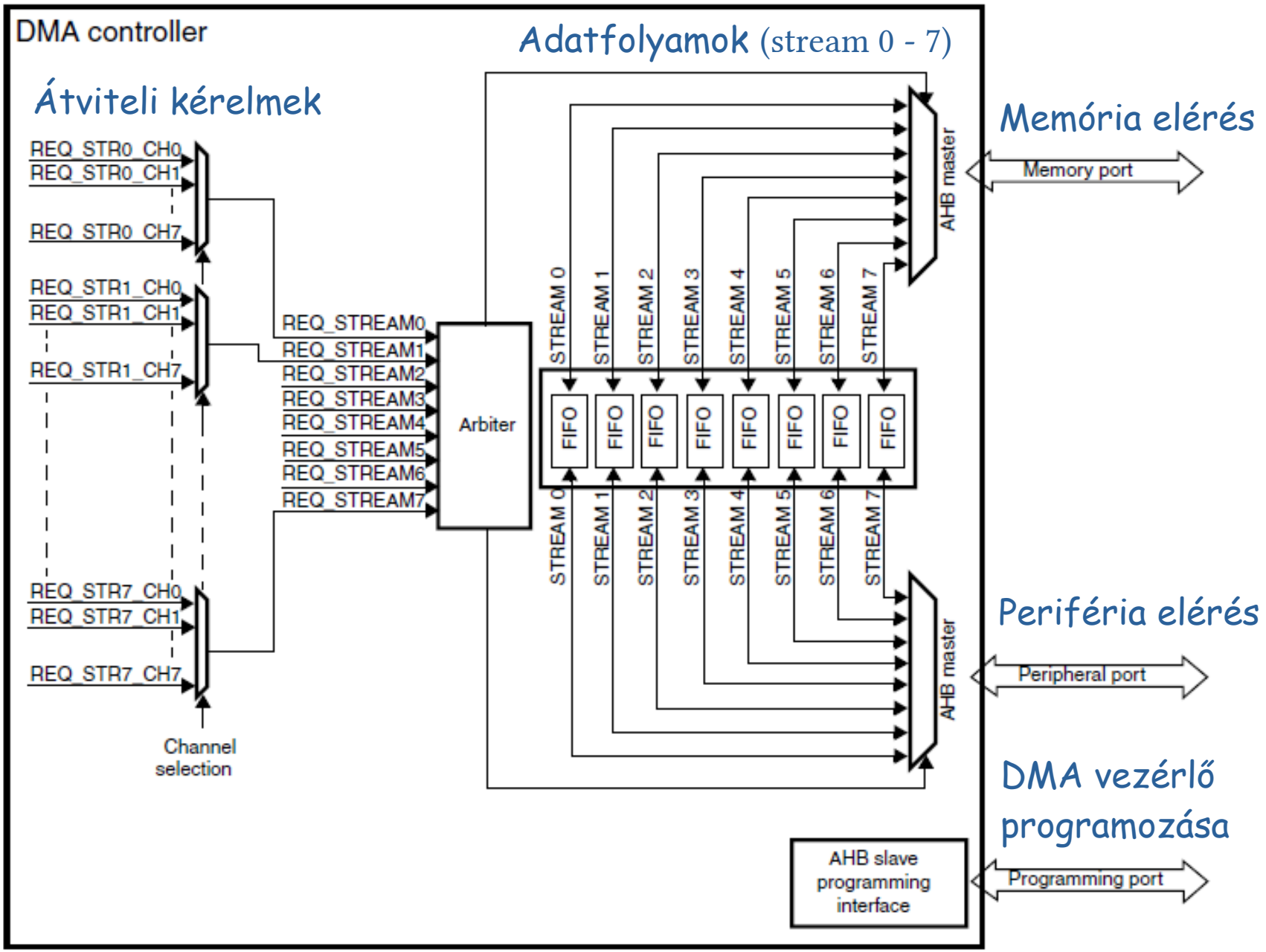


STM32F446RE MCU rendszerfelépítés

- Két **DMA** vezérlő van, külön az **APB1** és az **APB2** perifériákhoz



A DMA egység blokkvázlata



DMA2 csatornák és adatfolyamok

- Az **STM32F446RE** mikrovezérlőben az **ADC**-k a **DMA2** vezérlőhöz kapcsolódnak

Table 29. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A	ADC1	SAI1_B	TIM1_CH1 TIM1_CH2 TIM1_CH3	SAI2_B
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B	-	-	DCMI
Channel 2	ADC3	ADC3	-	-	-	-	-	-
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	SAI2_A	SPI1_TX	SAI2_B	QUADSPI
Channel 4	SPI4_RX	SPI4_TX	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX	SPI4_TX	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	-	-	TIM8_CH4 TIM8_TRIG TIM8_COM

DMA HAL függvények

- **HAL_DMA_Init()** – a DMA modul konfigurálása

Lekérdezéses módú használat:

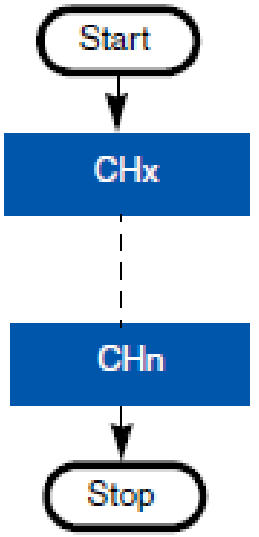
- **HAL_DMA_Start()** – az előzőleg konfigurált átvitel indítása
- **HAL_DMA_PollForTransfer()** – a DMA átvitel végére vár

Megszakításos módú használat:

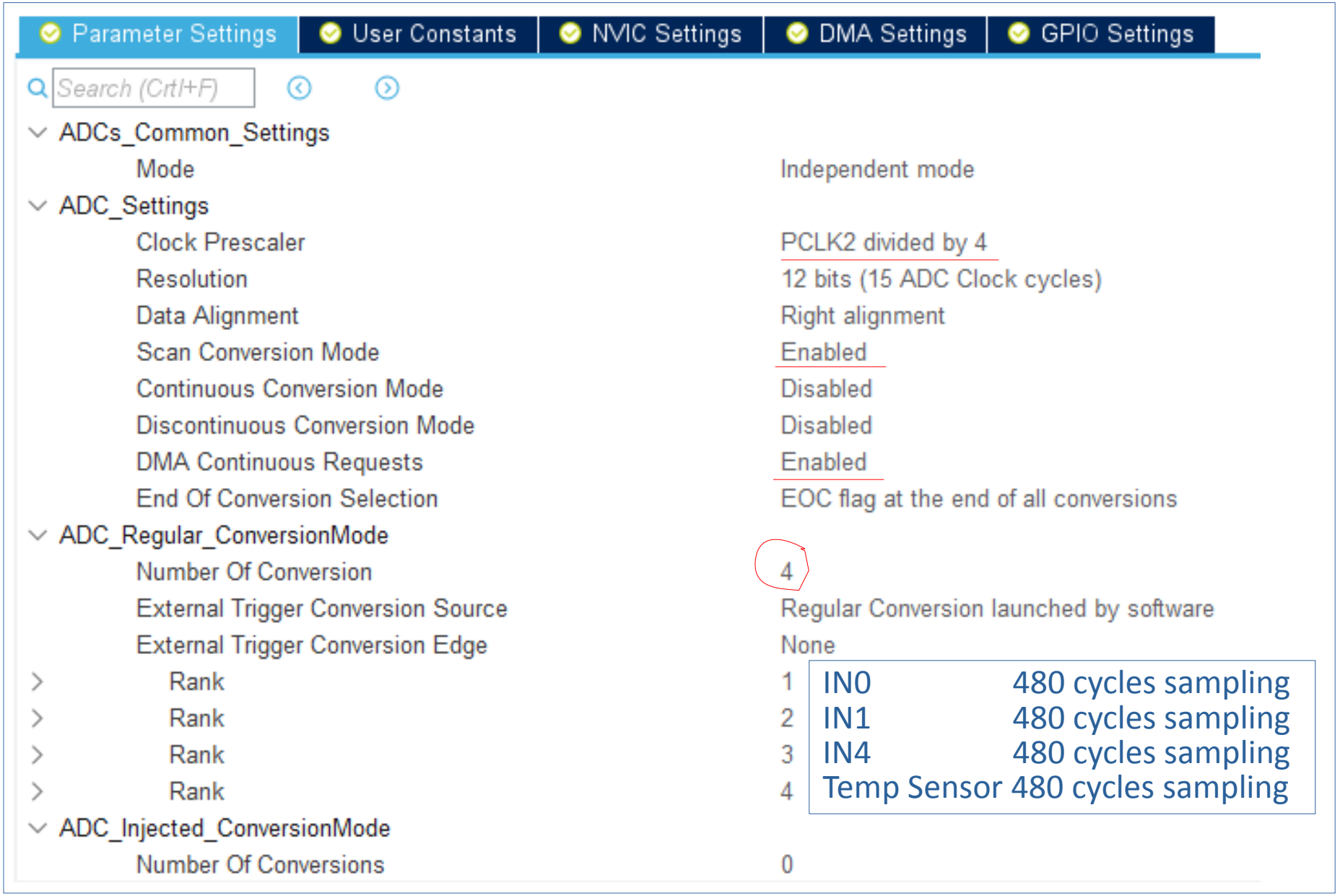
- **HAL_DMA_Start_IT()** – DMA megszakítás és átvitel engedélyezése
- **HAL_DMA_IRQHandler()** – DMA megszakításkiszolgáló függvény

F446RE_ADC_DMA projekt

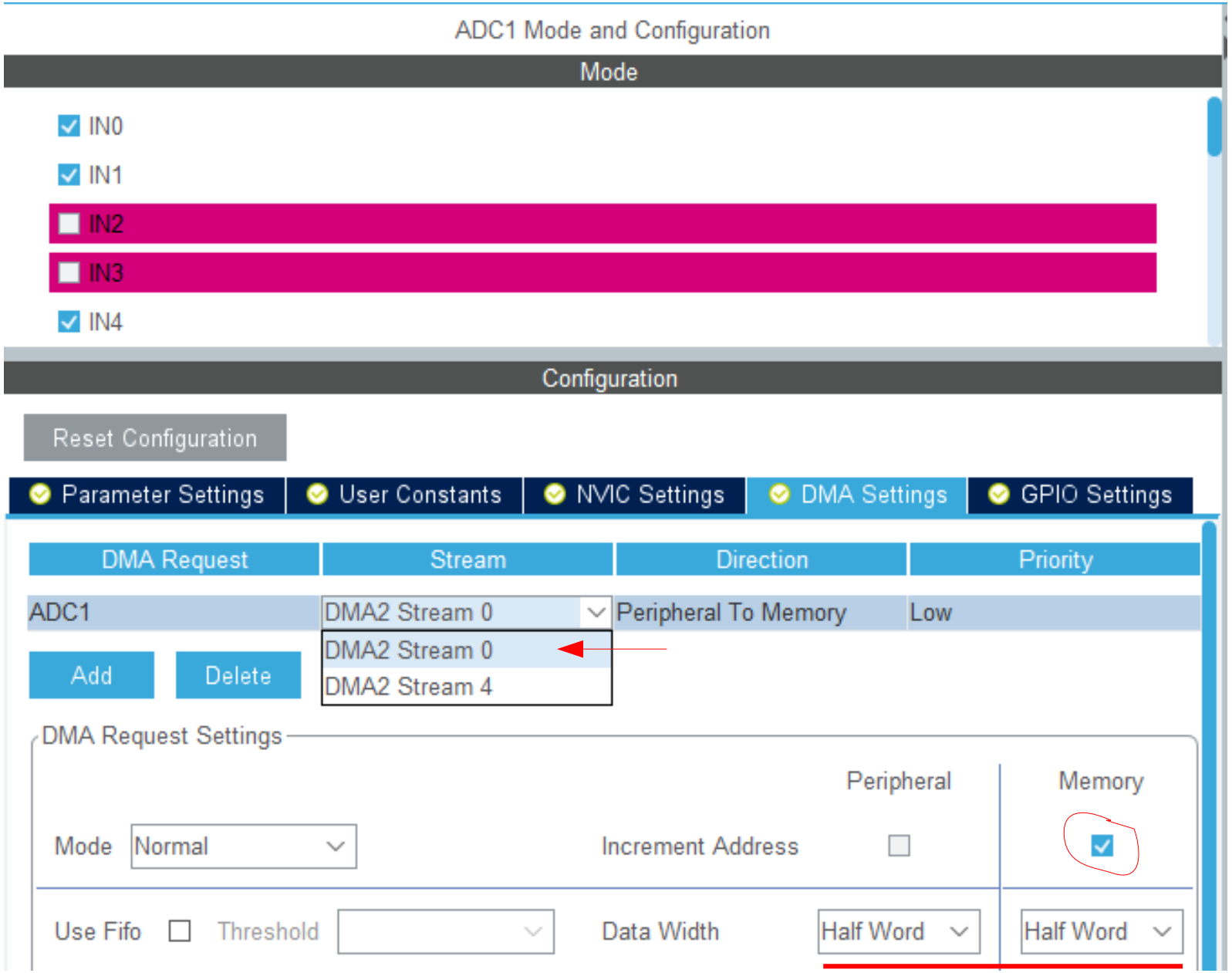
- Ebben a projektben 4 db reguláris csatorna jelét mérjük **ADC1** segítségével, scan (csatornapásztázó) módban, s az adatokat a **DMA2** egység segítségével olvassuk ki és pakoljuk át a memóriába
- A feszültségre, ill. hőmérsékletre átszámolt eredményeket az **UART2** porton (**PA2, PA3** kivezetéseken) keresztül kiíratjuk
- **A felhasznált csatornák:**
  IN[0] (PA0) egy potméter jele
  IN[1] (PA1) egy külső hőmérő jele (TC1047/MCP9700)
  IN[4] (PA4) 2.5 V külső referencia feszültség
  IN[18] (**temp_sensor)** a belső hőmérő jele
- A program egyszerű példát mutat a DMA használatára



F446RE_ADC_DMA: ADC1 konfigurálása



F446RE_ADC_DMA: ADC1/DMA2 konfigurálása



F446RE_ADC_DMA: main.c 3/1. oldal

```c
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
UART_HandleTypeDef huart2;
volatile uint8_t data_ready = 0;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_USART2_UART_Init(void);

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
  data_ready = 1;                          // Átvitel végének jelzése
}

void HAL_ADC_ErrorCallback(ADC_HandleTypeDef *hadc) {
  asm("BKPT #0");                          // Hiba esetén leállunk
}
```

F446RE_ADC_DMA: main.c 3/2. oldal

- A kiíratás hasonlóan történik, mint az előző programban, csak most a belső hőmérőhöz tartozó csatorna sorszáma nem 4, hanem 3

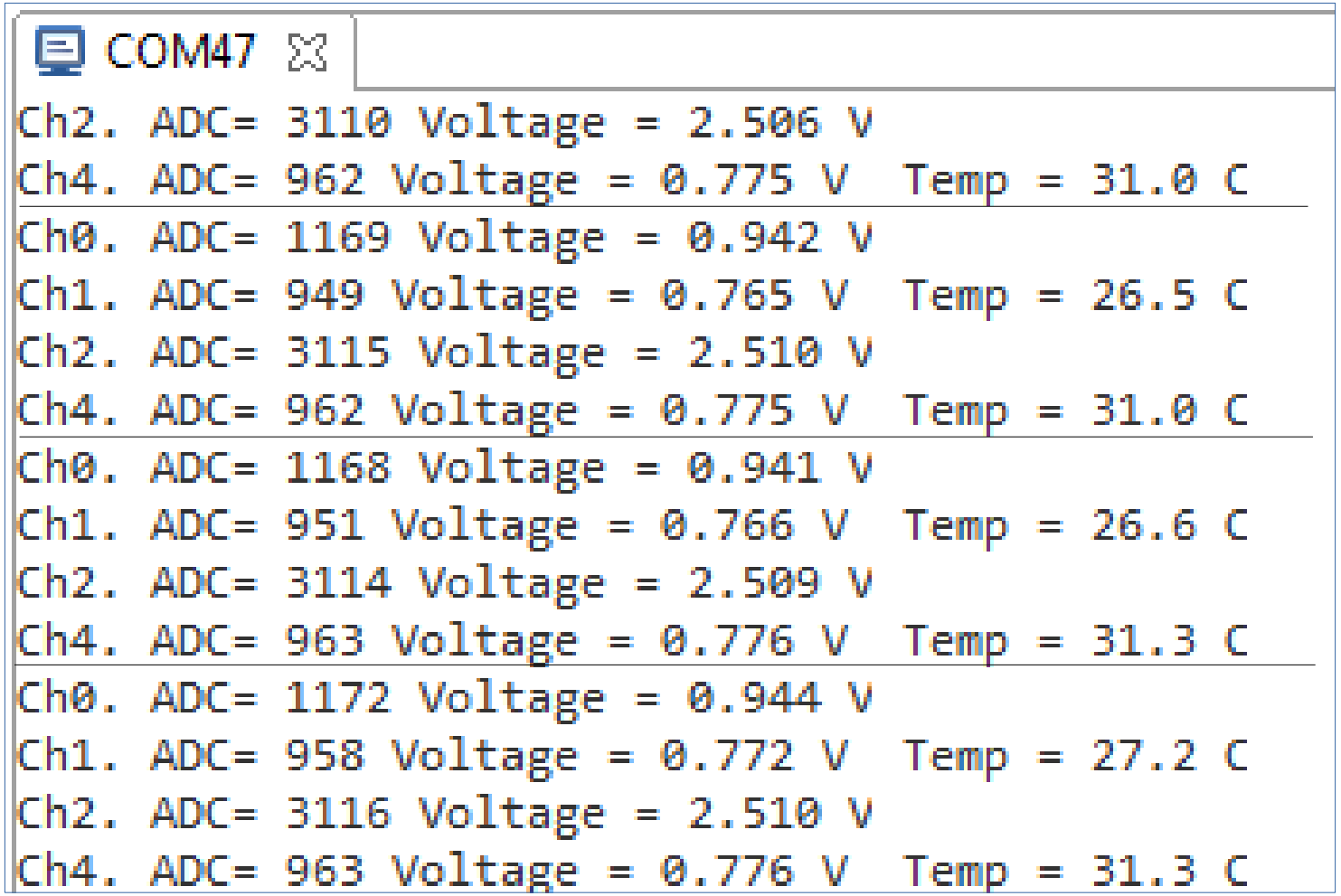
```
void display_data(int Nadc, float voltage, int i)  {
  char msg[80];
  float tempC;
  if (i == 1) {
    tempC = (voltage - 0.5) / 0.01;
    sprintf(msg, "Ch%d. ADC= %d Voltage = %5.3f V Temp = %4.1f C\r\n",\
            i, Nadc,voltage,tempC);
  }
  else if (i == 3) {
    tempC = ((voltage - 0.760) / 0.0025) + 25.0;
    sprintf(msg, "Ch4. ADC= %d Voltage = %5.3f V Temp = %4.1f C\r\n",\
            Nadc,voltage,tempC);
  }
  else {
    sprintf(msg, "Ch%d. ADC= %d Voltage = %5.3f V\r\n", i, Nadc,voltage);
  }
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
}
```

F446RE_ADC_DMA: main.c 3/3. oldal

```
int main(void) {
  float voltage;
  int val;
  uint16_t data[5];                                  // Ide pakolja az eredményeket a DMA
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USART2_UART_Init();

  while (1) {
    for(int i = 0; i<5; i++) data[i] = 0;            // Összegző változók törlése
    data_ready = 0;
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)data, 4);
    while(!data_ready);
//  HAL_ADC_Stop_DMA(&hadc1);                        // Akkor kell, ha nincs engedélyezve a
    for(int i = 0; i<4; i++) {                       // DMA Continuous Requests paraméter
      val = data[i];
      voltage = (float)val * 3.3 / 4096;             // Vref = 3.3 V, resolution = 4096
      display_data(val, voltage, i);                 // Az eredmény kiíratása
    }
    HAL_Delay(1000);
  }
}
```

F446RE_ADC_DMA: futási eredmény

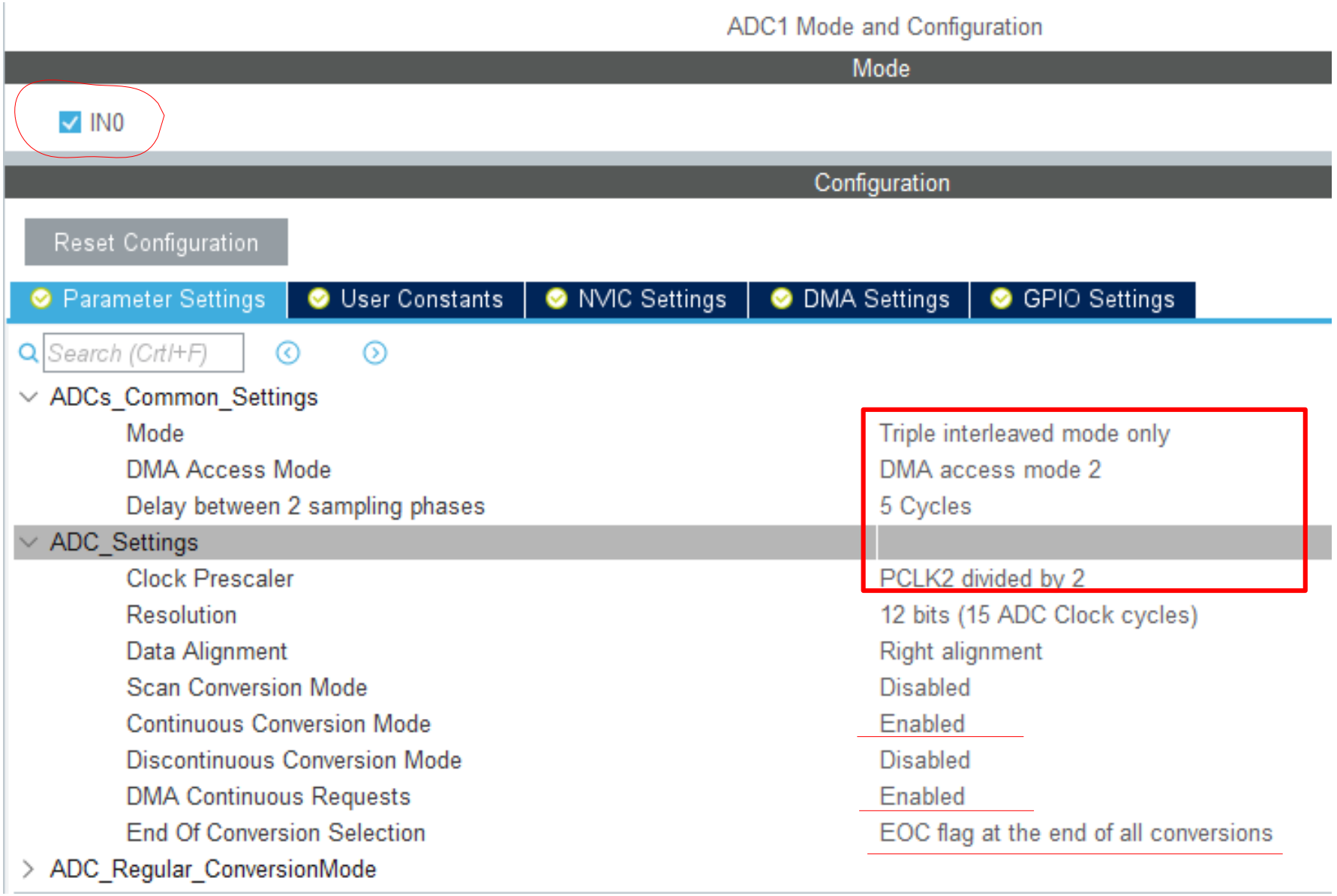
- A futási eredmények erős szórást mutatnak, érdemes lenne itt is több mérést átlagolni a zaj okozta ingadozások kiátlagolására!



F446RE_ADC_interleaved

- Az utolsó projektben az **IN0** (PA0) analóg csatorna jelét mérjük meg az **ADC1, ADC2** és **ADC3** átalakítók tripla átlapolásos módjában
- Az egyes ADC-k 5-5 órajelciklus késleltetéssel indulnak egymás után, így az időkeretbe pont belefér egy 3 órajelciklusos mintavétel és egy 12 órajelciklusos 12 bites konverzió
- A maximális mintavételi sebesség elérésére az ADC-ket 36 MHz-es órajellel hajtjuk, ehhez a rendszer órajelek konfigurálásánál el kell térnünk a szokásos beállítástól, most SYSCLK = 144 MHz, PCLK2 = 72 MHz, ADCCLK = 36 MHz (PCLK2/2) beállítás kell
- A fenti beállítással a mintavételi frekvencia = 36/5 = 7.2 MHz

ADC1 konfigurálása

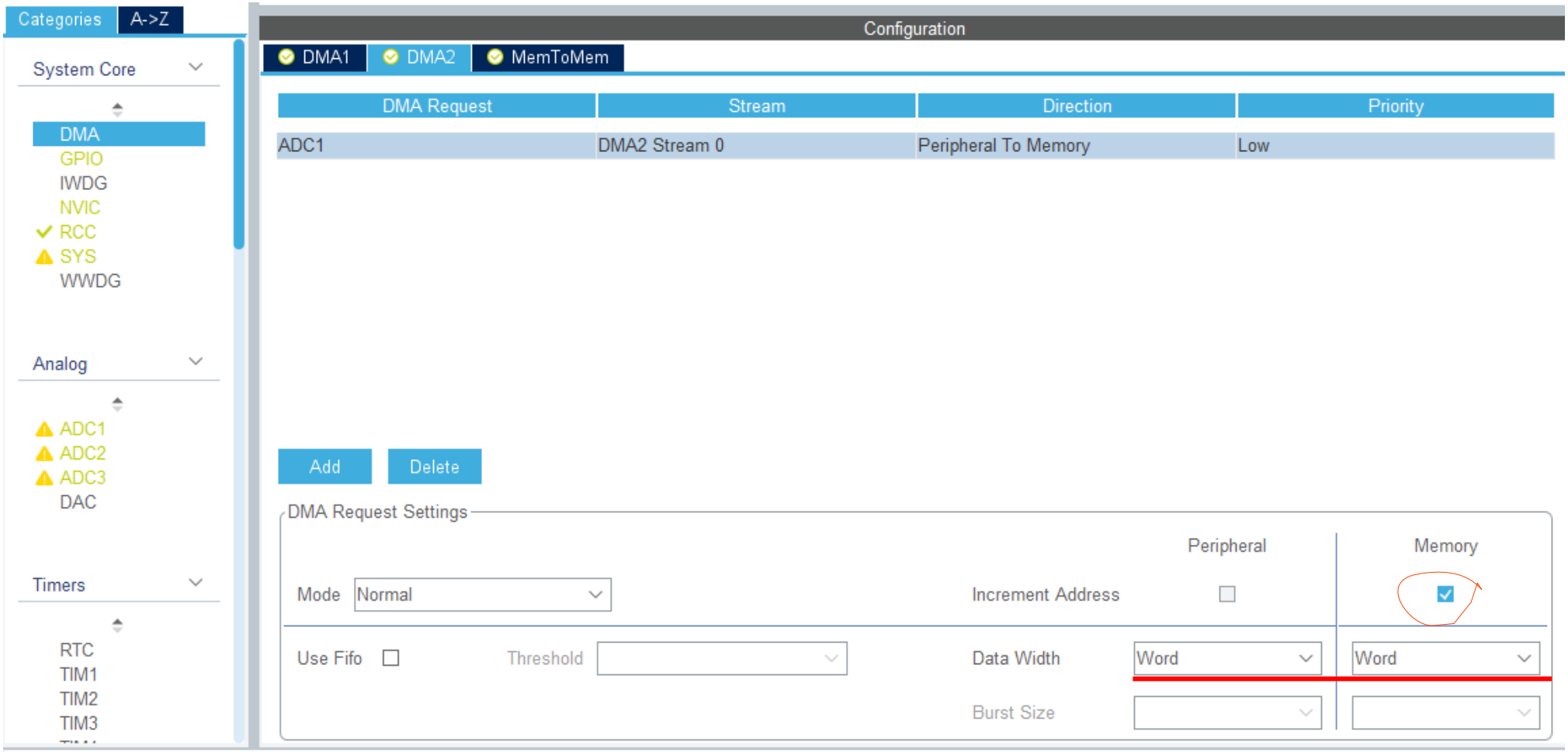


ADC bemeneti csatorna konfigurálása

- Ha az ADC-k ugyanazt a csatornát mérik, akkor a mintavételi idő *(Sampling Time)* legalább 2 órajelciklussal kevesebb legyen az előző oldalon beállított késleltetési időnél *(Delay between 2 sampling phases)*

A DMA adatfolyam konfigurálása

- Tripla átlapoló módnál csak **ADC1 DMA** adatfolyamát (itt **DMA2/Stream0**) kell konfigurálni
- Az adatszélesség 32 bites, mindig két-két adat kerül átvitelre!



F446RE_ADC_interleaved: main.c

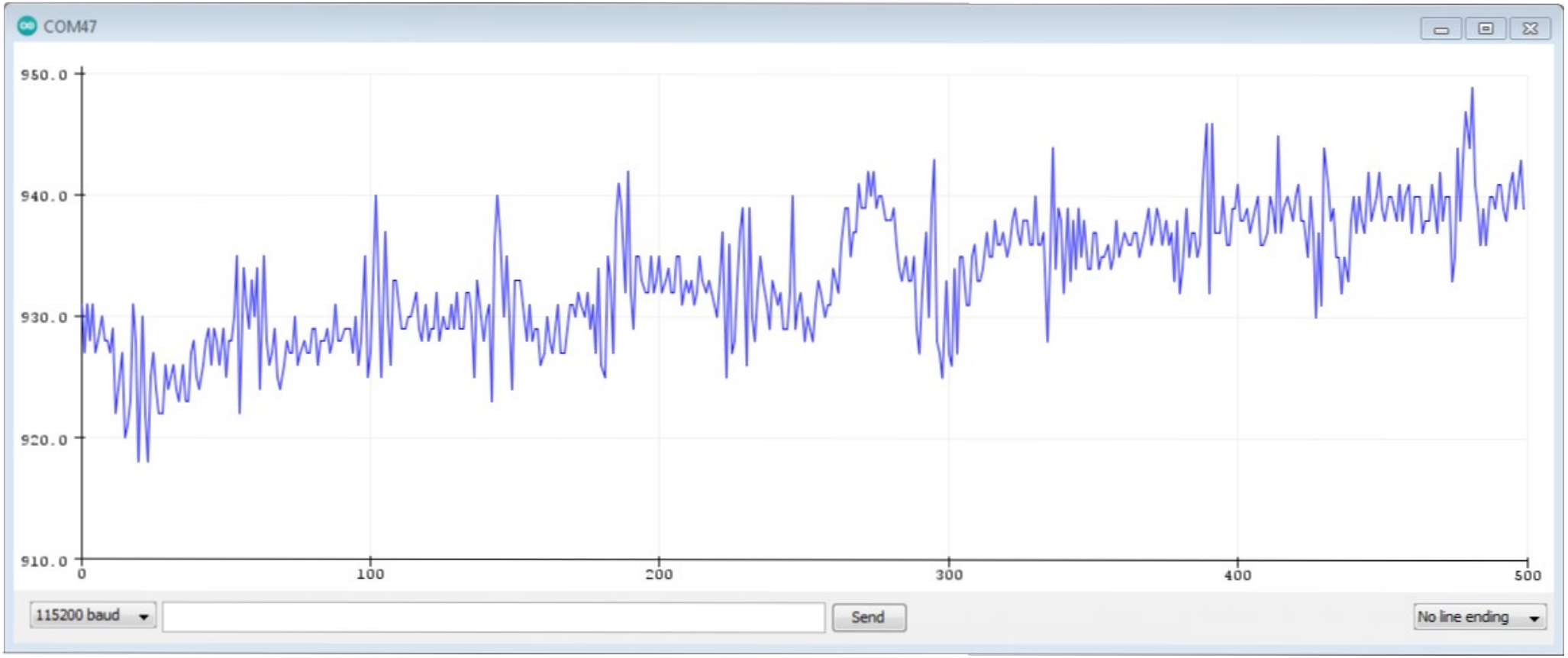
```c
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#define NDATA   250
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
ADC_HandleTypeDef hadc3;
DMA_HandleTypeDef hdma_adc1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);
static void MX_ADC3_Init(void);
static void MX_USART2_UART_Init(void);
volatile uint8_t data_ready = 0;
volatile uint32_t data[NDATA];
char msg[80];
uint32_t status;

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
  data_ready = 1;
}
```

F446RE_ADC_interleaved: main.c

```
int main(void) {
  int d1, d2;
  HAL_Init(); SystemClock_Config();
  MX_GPIO_Init();   MX_DMA_Init();
  MX_ADC1_Init();   MX_ADC2_Init();
  MX_ADC3_Init();   MX_USART2_UART_Init();
  while (1) {
    for(int i = 0; i<NDATA; i++) data[i] = 0;      // A törlés nyomkövetéshez kellett
    data_ready = 0;
    HAL_ADC_Start(&hadc3);
    HAL_ADC_Start(&hadc2);
    HAL_ADCEx_MultiModeStart_DMA(&hadc1, (uint32_t*)data,NDATA);
    while(!data_ready);
    status = HAL_ADCEx_MultiModeStop_DMA(&hadc1);  // staus = 1 hibával tér vissza!
    HAL_ADC_Stop(&hadc3);
    HAL_ADC_Stop(&hadc2);
    for(int i = 0; i<NDATA; i++) {
      d1 = data[i] & 0x0FFF;
      d2 = data[i] >> 16;
      sprintf(msg, "%d\r\n%d\r\n", d1, d2);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    HAL_Delay(5000);
    MX_ADC1_Init();                                // Enélkül nem lehetett újraindítani
  }
}
```

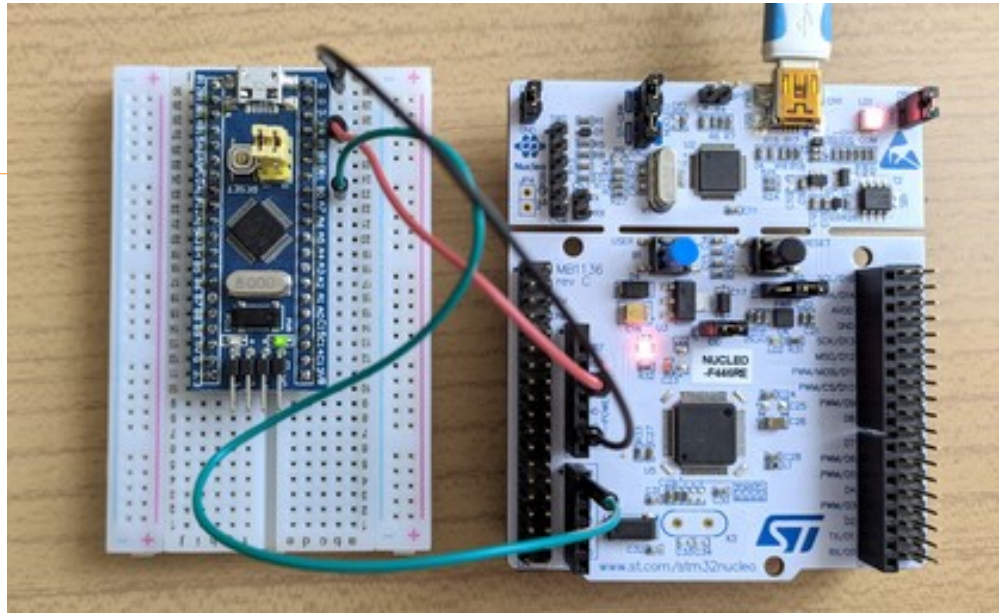
F446RE_ADC_interleaved: futási eredmény

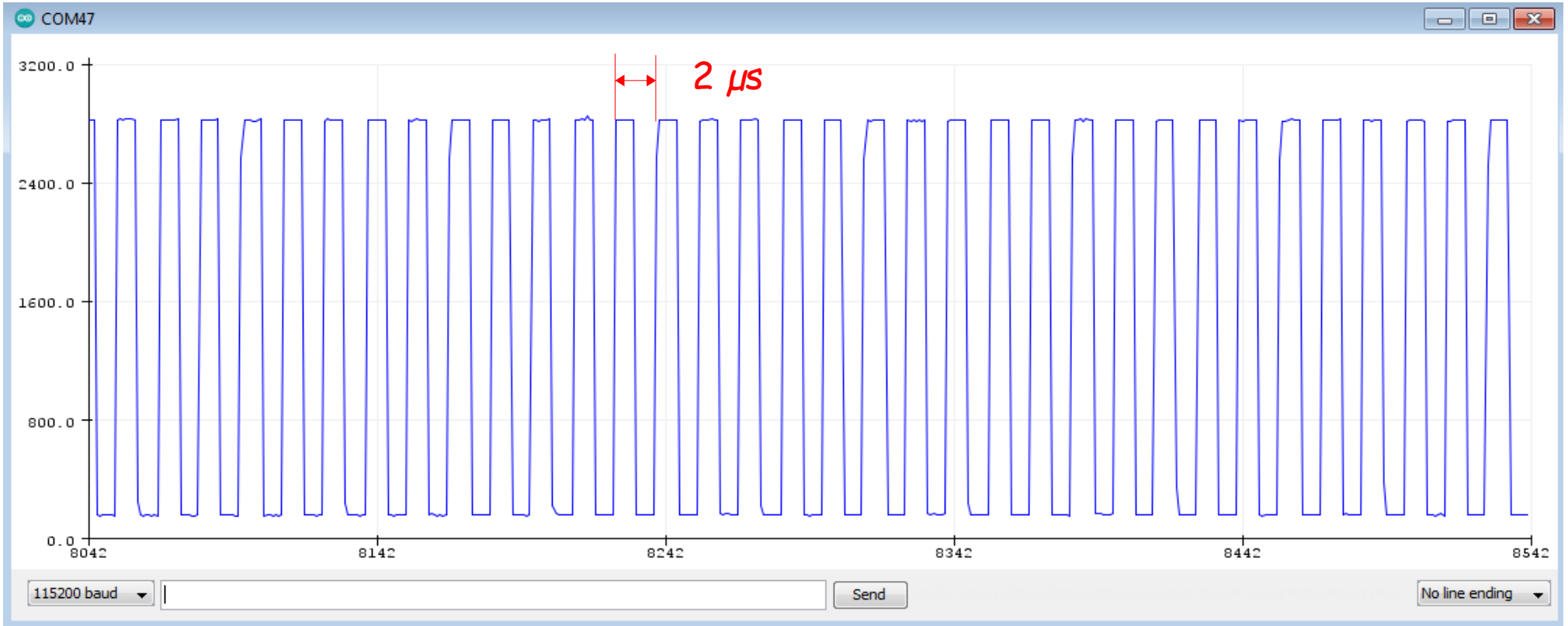
- Egy zajos egyenfeszültség mérése (TC1047 hőmérő jele)

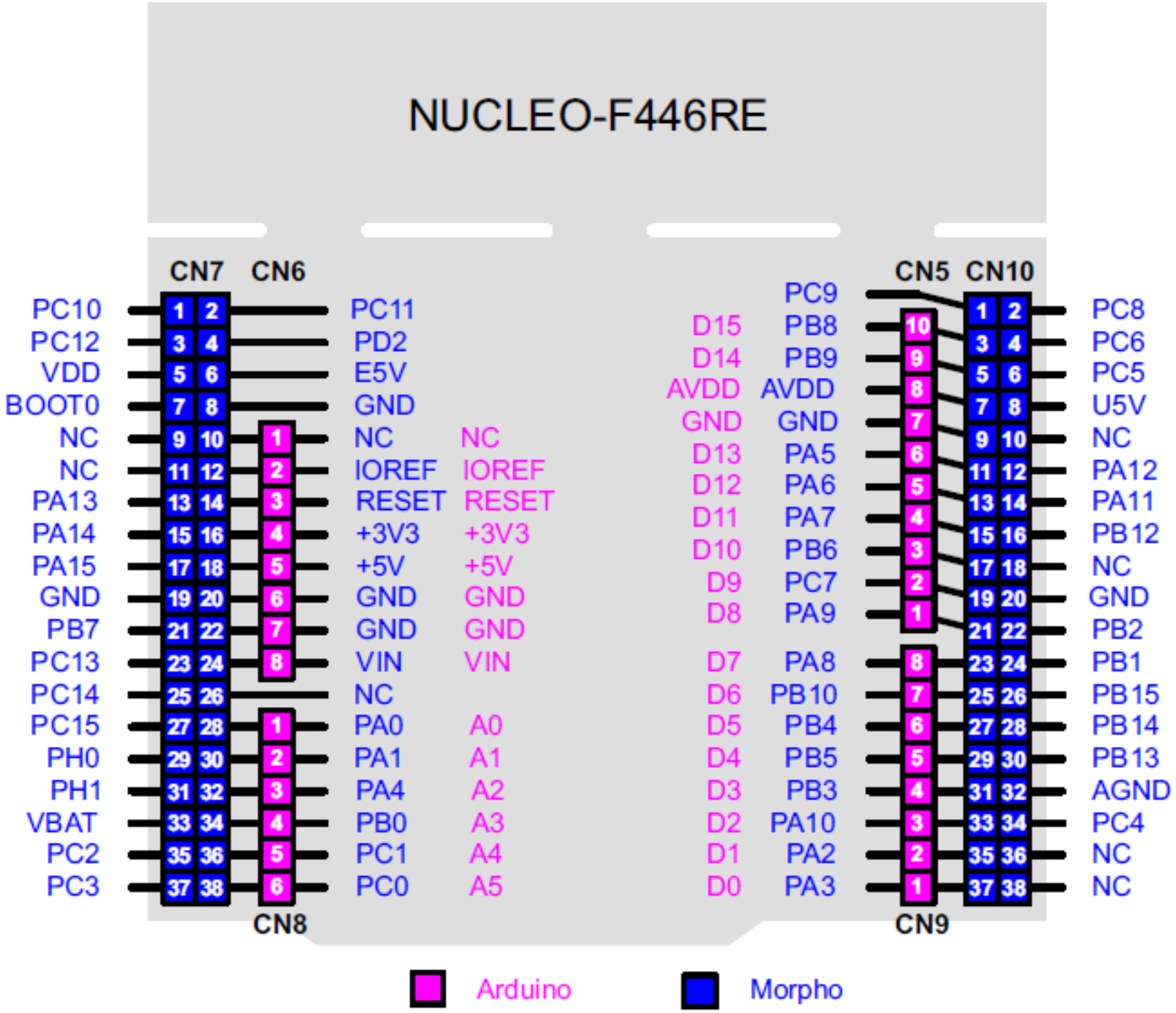


500 kHz négyszögjel

- A tavalyi **keil19_08** előadásban bemutatott **Program07_6b** mintapélda apró módosításával 500 kHz-es (1 µs félperiódus) jelet keltünk és vizsgálunk






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