STM32 mikrovezérlők programozása STM32CubeIDE környezetben

7. Digitál-analóg konverter – 2. rész

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**

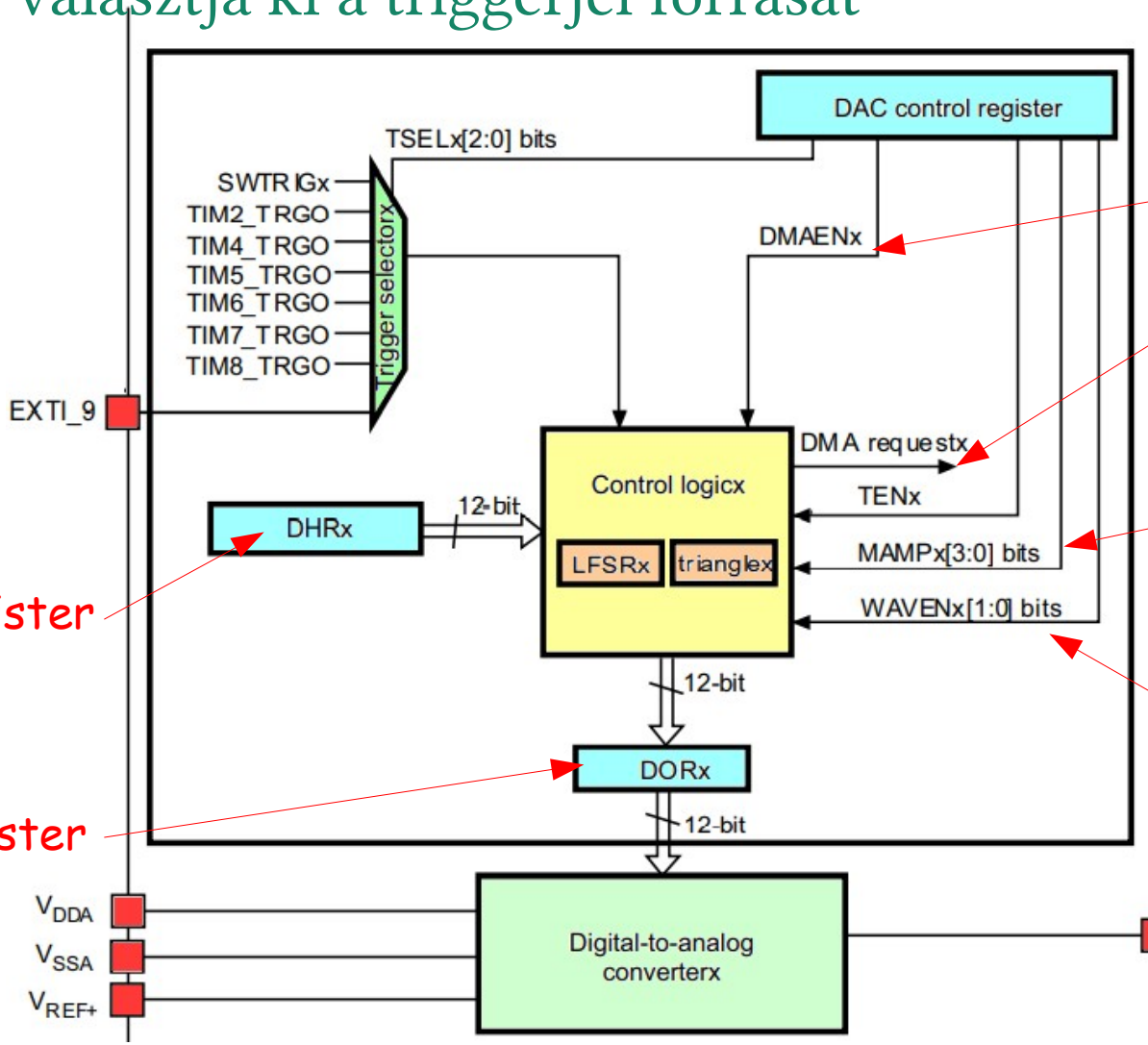
- Khaled Magdy (DeepBlue):
  STM32 DAC Tutorial – Example HAL Code & Analog Signal Generation

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**

Emlékeztető: A digitál-analóg átalakító felépítése

- A **DMAEN*x*** bit engedélyezi a DMA átvitelt, a **TSEL*x*[2:0]** bitcsoport választja ki a triggerjel forrását



DMA átvitelt engedélyező bit
DMA átviteli kérelem

MAMPx – mask vagy amplitudó beállítás

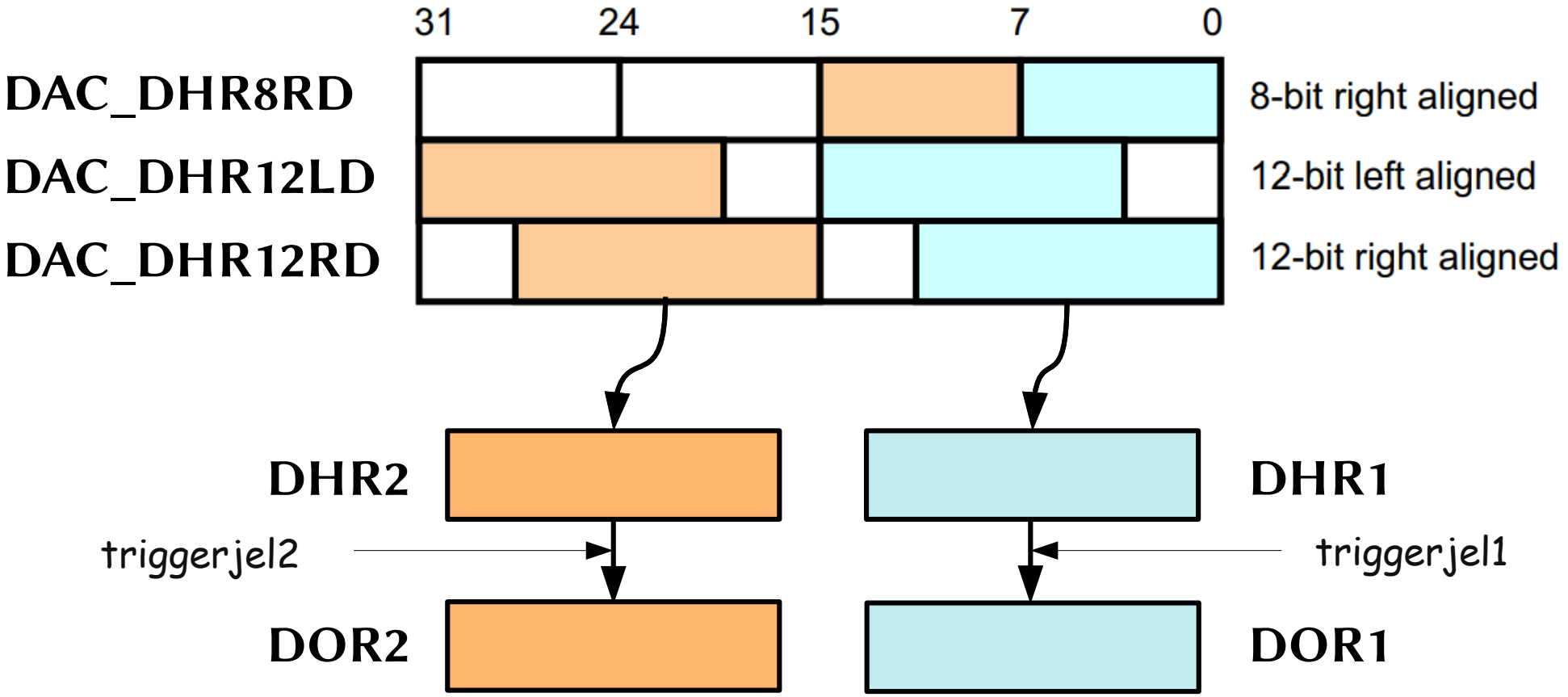
WAVENx – jelgenerátor mód választás

Data Holding Register (közvetlenül nem címezhető)

Data Output Register

Emlékeztető: DAC adatformátum

- Az adatbeírás nem közvetlenül, hanem a jobbra-balra igazítást, ill. helyiérték-eltolást szervező regisztereken keresztül történik, amelyekből három készlet van (1. és 2. csatorna, illetve duál mód)
- Itt a duál módú beíráshoz tartozó regisztereket tüntettük fel



Emlékeztető: a DAC vezérlő regisztere

- **DAC_CR** a két DAC csatorna közös vezérlő regisztere

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		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.	DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **EN*x*** – engedélyezés, **BOFF*x*** – buffer engedélyezés
- **TEN*x*** – trigger engedélyezés, **TSEL*x*** – trigger jelforrás választás
- **WAVE*x*[1:0]** – jelgenerátor engedélyezés/választás
- **MAMP*x*[3:0]** – maszk, ill. amplitúdó beállítás
- **DMAEN*x*** – DMA átvitel engedélyezése
- **DMAUDRIE*x*** – DMA aláfutási megszakítás engedélyezése

Emlékeztető: DMA átvitel

- A **DAC** csatornák adatfeltöltése a **DMA** Stream 5 és Stream 6 adatfolyammal történhet
- A DMA adatátviteli kérelem a 7. csatornát használja

Table 28. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	SPDIFRX_DT	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	SPDIFRX_CS	SPI3_TX
Channel 1	I2C1_RX	I2C3_RX	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	FMPI2C1_RX	TIM4_CH2	-	FMPI2C1_RX	TIM4_UP	TIM4_CH3
Channel 3	-	TIM2_UP TIM2_CH3	I2C3_RX	-	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	-	-	TIM3_CH4 TIM3_UP	-	TIM3_CH1 TIM3_TRIG	TIM3_CH2	-	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

HAL_DAC és HAL_DACEx modulok

- A **HAL_DAC** modul az alapvető API-t biztosítja:
  - **HAL_DAC_Init** – a DAC inicializálása
  - **HAL_DAC_MspInit** – a DAC kivezetés(ek) inicializálása (PA4/PA5)
  - **HAL_DAC_ConfigChannel** – DAC csatorna konfigurálása
  - **HAL_DAC_Start** – A DAC modul indítása
  - **HAL_DAC_Start_DMA** – a DAC modul és a DMA csatorna indítása
  - **HAL_DAC_SetValue** – DAC csatorna értékének szoftveres beállítása
- A **HAL_DACEx** modul további függvényeket biztosít:
  - **HAL_DACEx_TriangleWaveGenerate** – háromszögjel generátor
  - **HAL_DACEx_NoiseWaveGenerate** – álvéletlen zaj generátor
  - **HAL_DACEx_DualSetValue** – szoftveres adatbeírás duál módban
- Amint látjuk, nincs **HAL_DAC_Start_Dual_DMA()** függvény – majd írunk egyet magunknak!

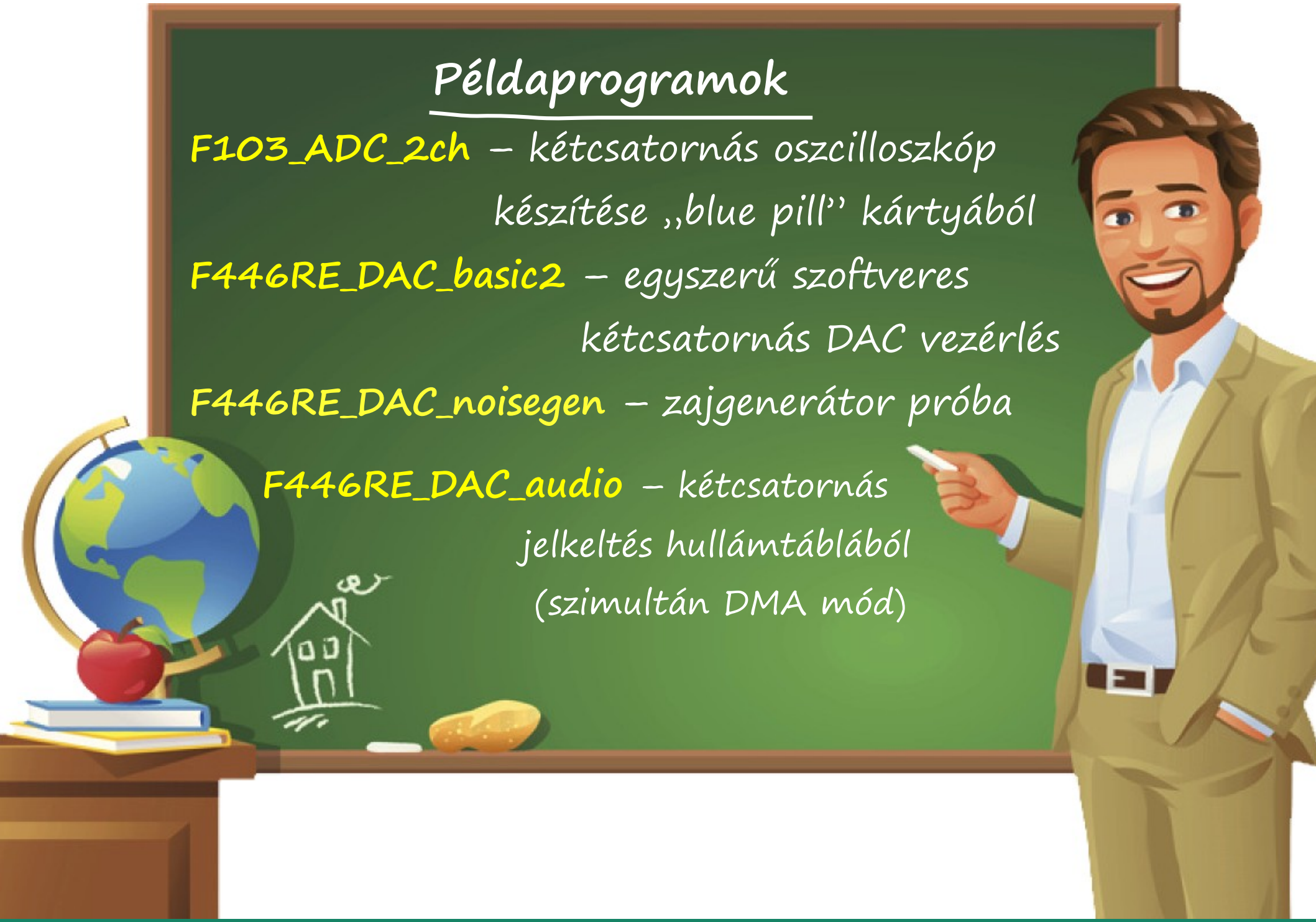
Példaprogramok

F103_ADC_2ch – kétcsatornás oszcilloszkóp készítése ,,blue pill'' kártyából

F446RE_DAC_basic2 – egyszerű szoftveres kétcsatornás DAC vezérlés

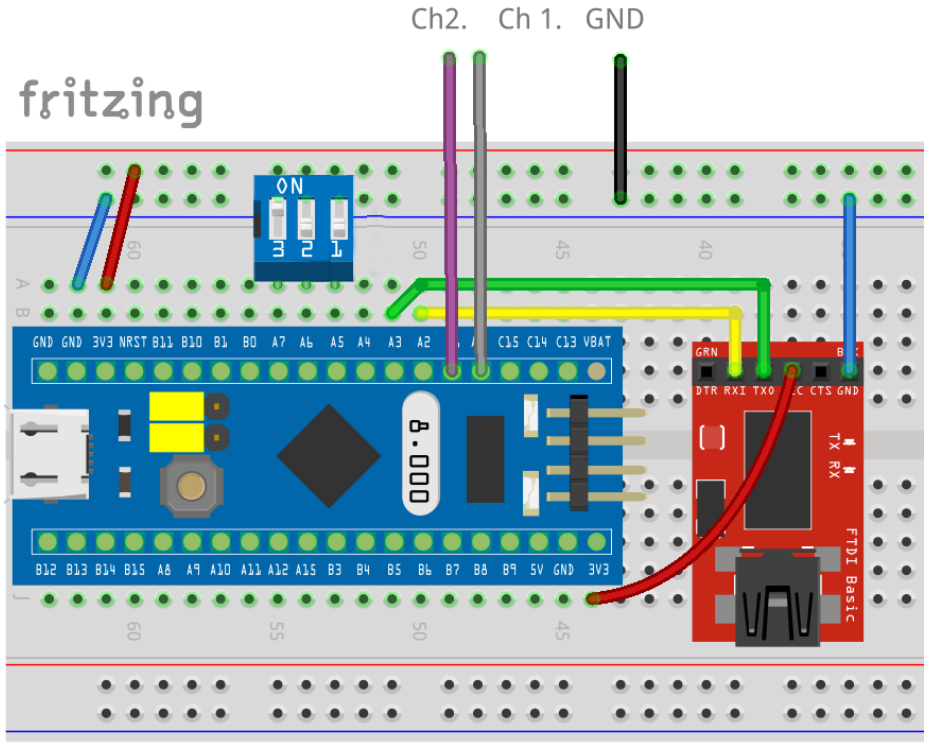
F446RE_DAC_noisegen – zajgenerátor próba

F446RE_DAC_audio – kétcsatornás jelkeltés hullámtáblából (szimultán DMA mód)



F103_ADC_2ch: kétcsatornás oszcilloszkóp

- A korábbi előadásokban szerzett ismereteket felhasználva az **STM32F103C8** mikrovezérlőjének két **ADC**-je segítségével (**PA0** és **PA1** használatával) kétcsatornás tárolós oszcilloszkópot készítünk
- Fizikai korlátok:
  - Mintavételezési gyakoriság: 1 µs – 200 µs (a kapcsolókkal választható)
  - Működés módja: 500 adatpont gyűjtése, majd kiírás az **UART2** porton megjelenítés az ARDUINO IDE ***Serial Plotter*** ablakában
  - Bemenő jel: 0 – 3,3 V
  - Felbontás: 12 bit (0 – 4095)
- **Áramköri elrendezés:**
  Az **UART2** port kimenetét egy FTDI USB – UART átalakítón keresztül kötjük a számítógépre (Használhatnánk az **STM32F103C8** USB perifériáját is, de előbb meg kellene ismerkedni a használatával)



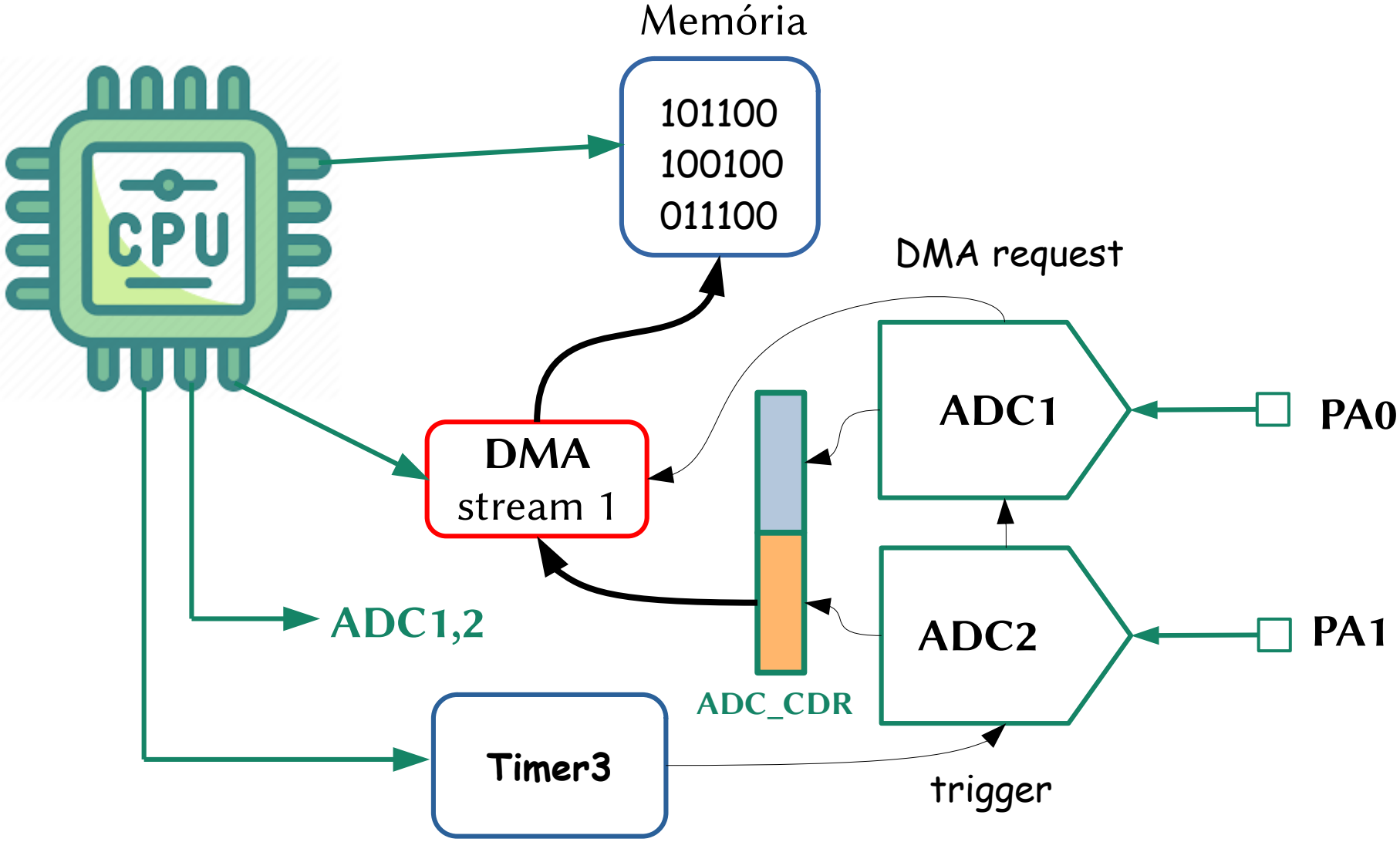
F103_ADC_2ch: kétcsatornás oszcilloszkóp

- A mintavételezés gyakoriságát a **PA7, PA6, PA5** bemenetekre kötött kapcsolósorral állíthatjuk be (belső felhúzást használunk, a kapcsolók a földre húznak)
- Változtatás esetén újra kell indítani a programot (RESET)

Switch	Tim6 Period	Sample period	Full scale
000	0	1 µs	0,5 ms
001	1	2 µs	1 ms
010	4	5 µs	2,5 ms
011	9	10 µs	5 ms
100	19	20 µs	10 ms
101	49	50 µs	25 ms
110	99	100 µs	50 ms
111	199	200 µs	100 ms

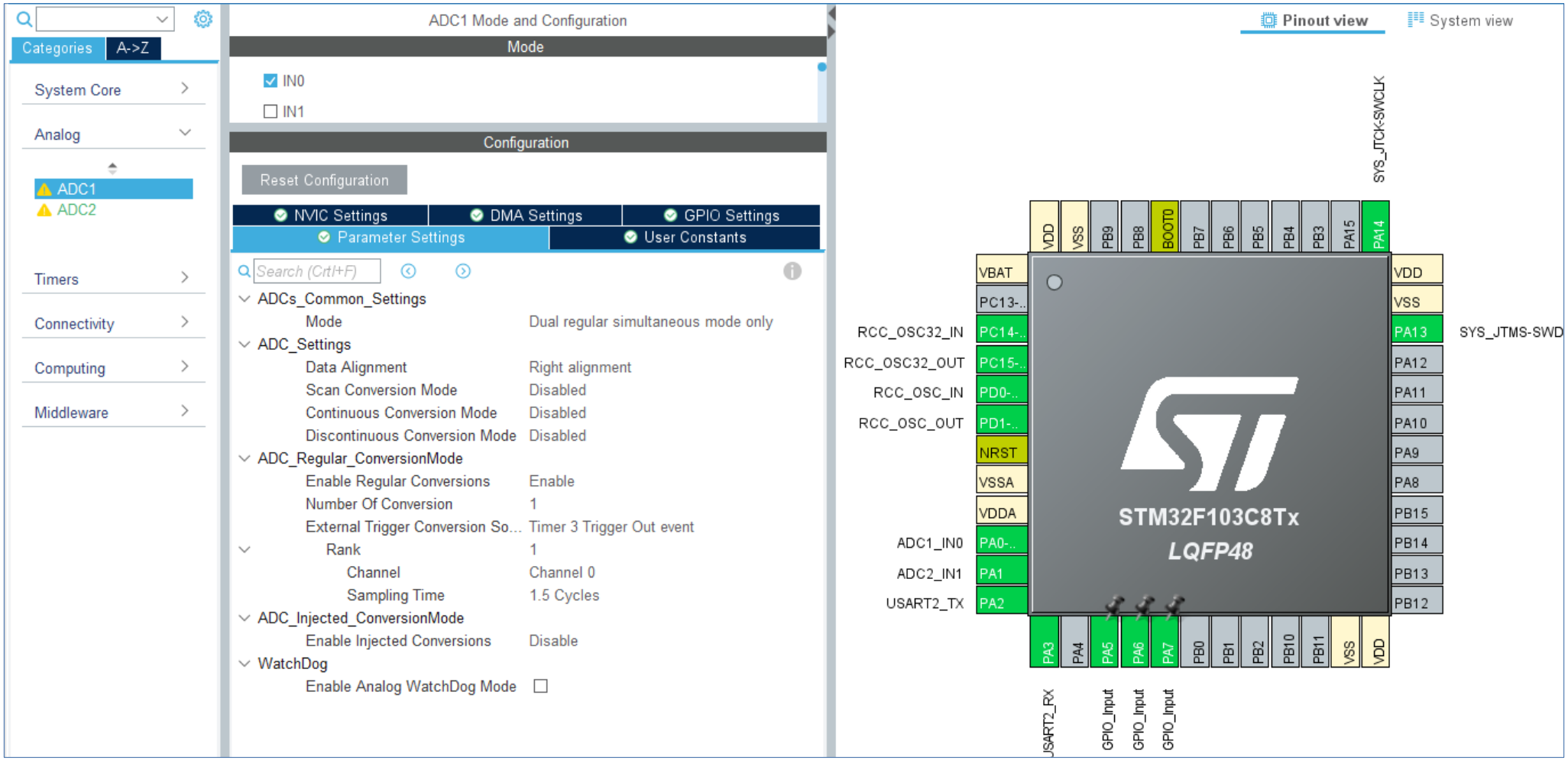
Az adatáramlás sematikus vázlata

- **Timer3** triggereli az összekapcsolt **ADC-ket,** közös, 32 bites adatregiszterüket **DMA1** olvassa ki és teszi a memóriába az adatot



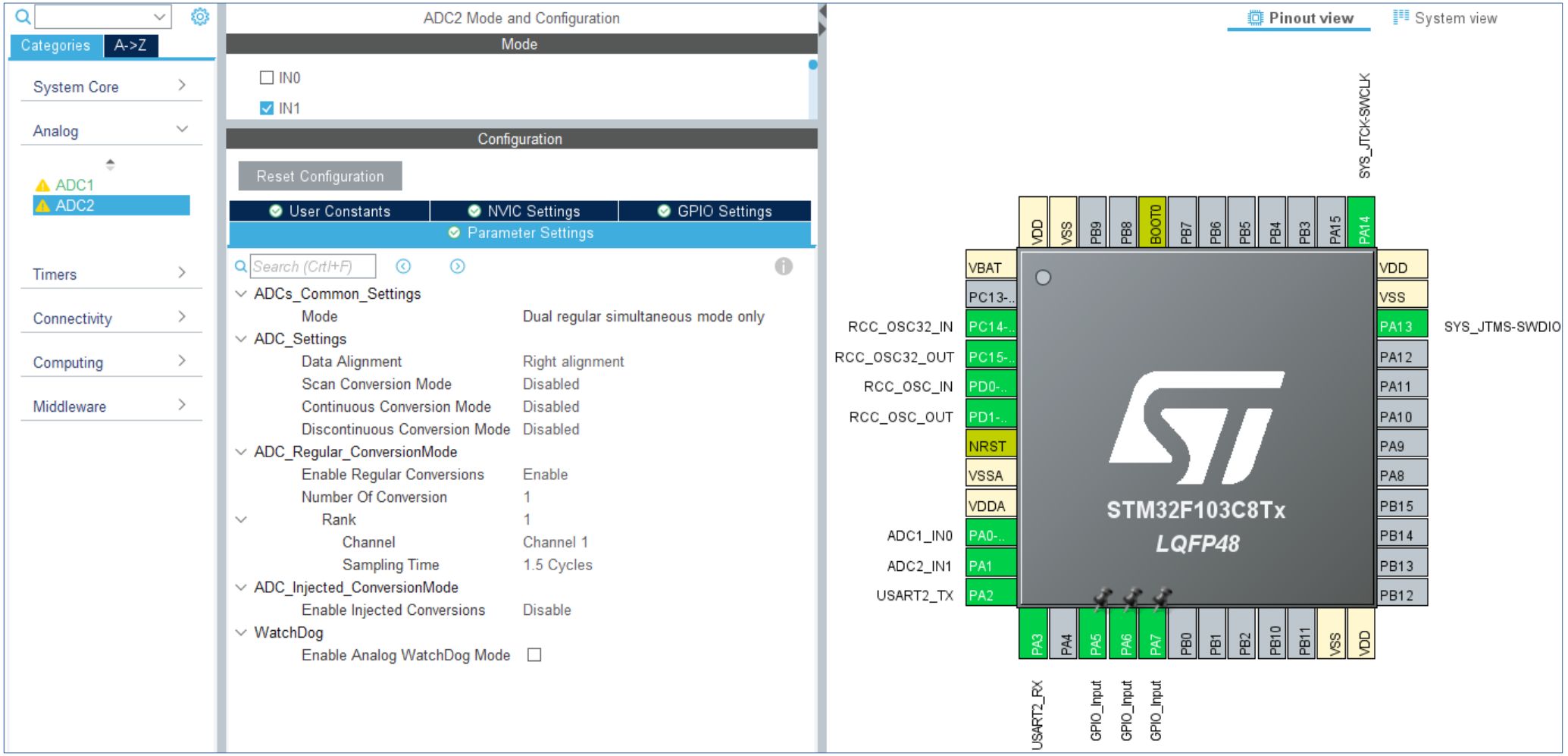
ADC1 konfigurálása

- Duál mód, 1 db. reguláris csatornával (**IN0**), nem folytonos a konverzió
- **Timer 3** TRGO a triggerforrás, **DMA**-val olvassuk ki az adatokat
- A leggyorsabb mintavételezést választottuk (sample+conversion 14 cy)



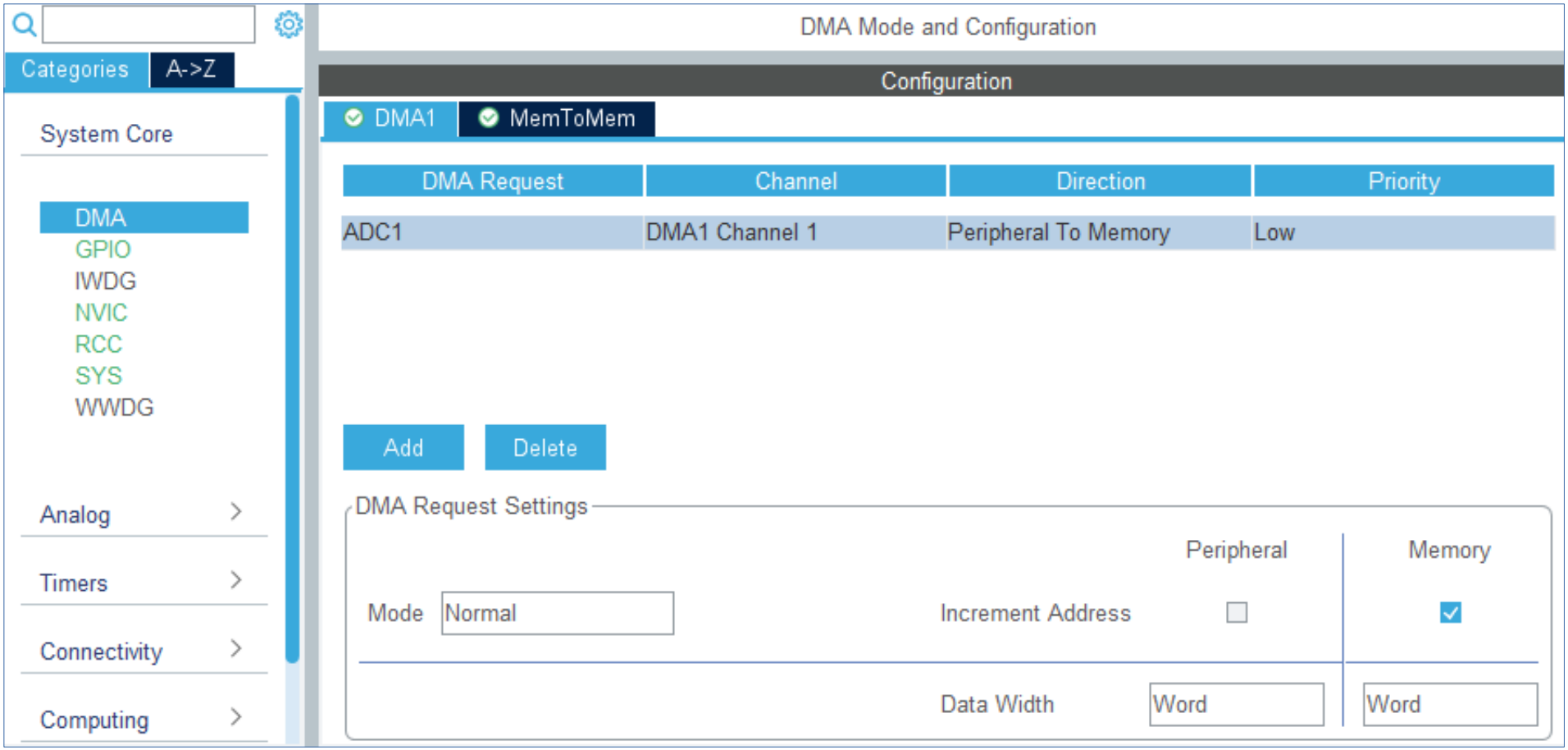
ADC2 konfigurálása

- Duál módban **ADC2** alárendelt szerepet játszik (slave)
- Itt is egy reguláris csatornát kell választanunk (**IN1**)
- A mintavételezés meg kell, hogy egyezzen az **ADC1** beállításával



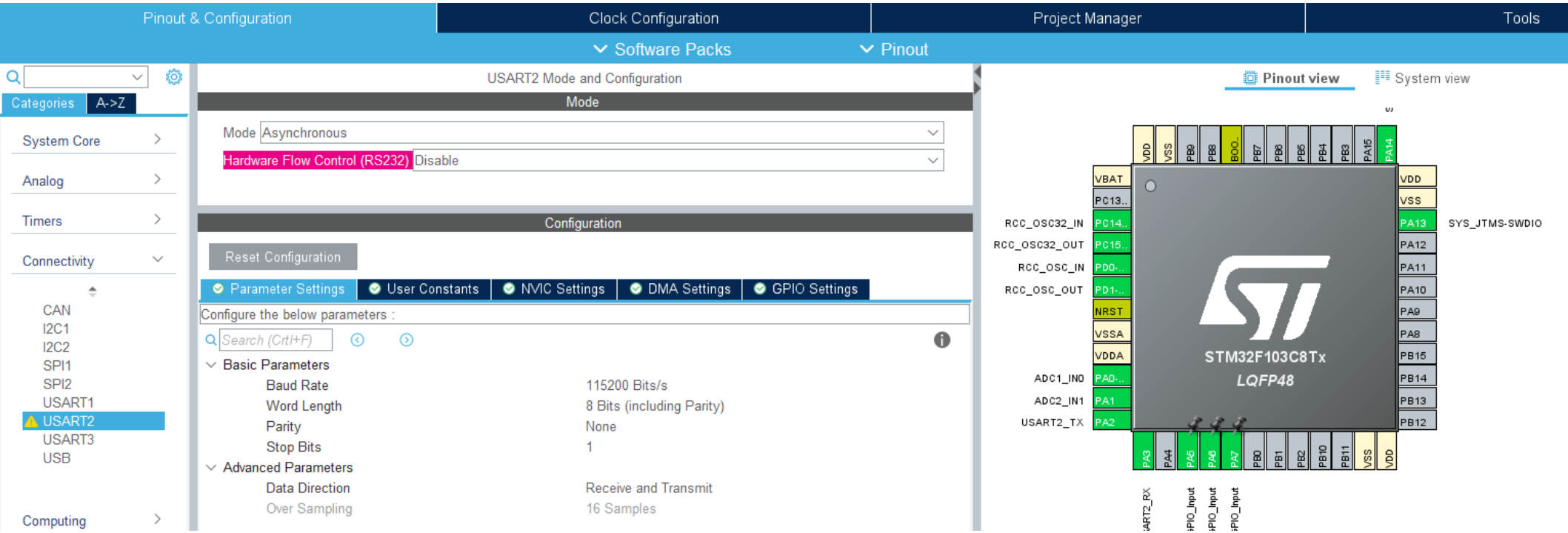
DMA konfigurálása

- Az **ADD** gombra kattintva **ADC1** DMA kérelmet kell beállítani
- Normál üzemmód (nem circular) kell, az átvitel pedig 32 bites (word), mert az **ADC1** és **ADC2** eredménye egy 32 bites közös regiszterbe kerül
- Az átvitelenkénti címléptetés értelemszerűen csak a memóriánál kell



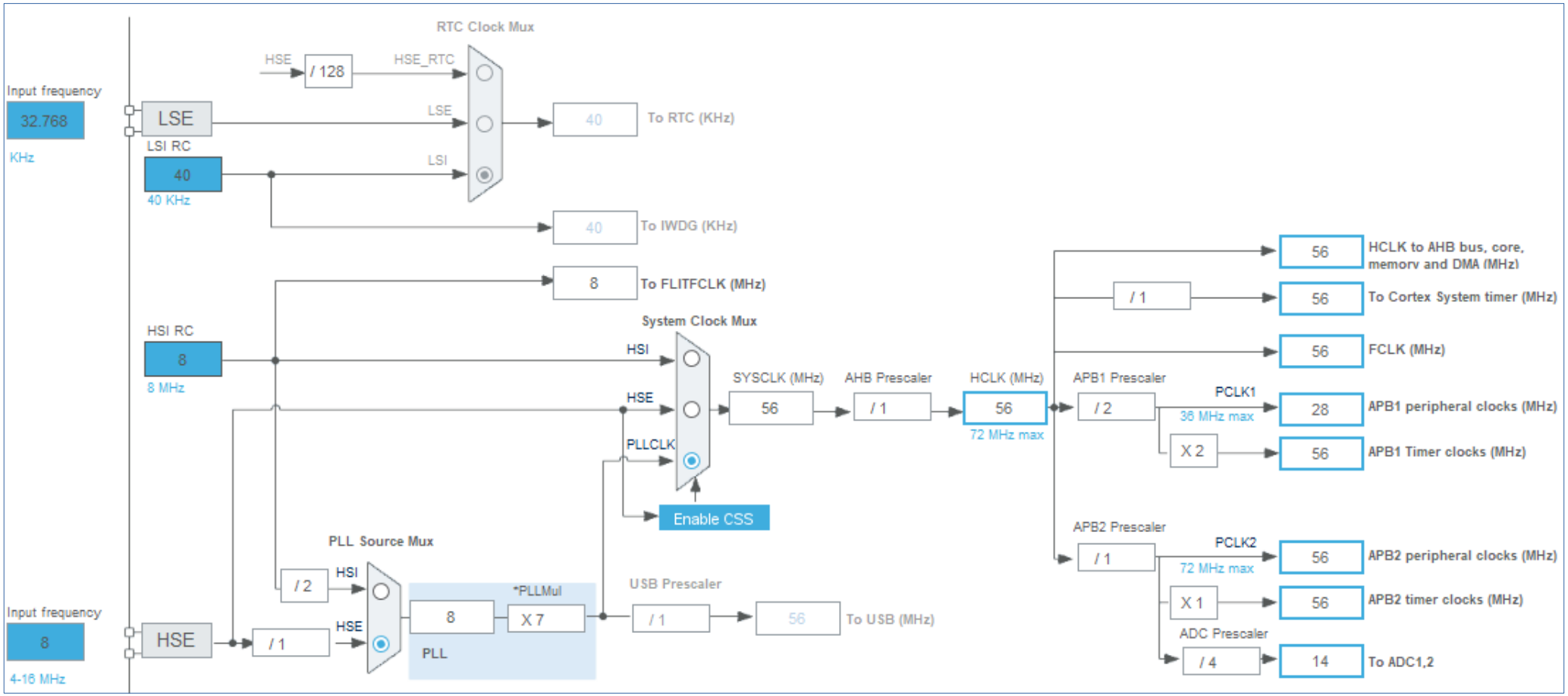
USART2 konfigurálása

- A szokásos beállításokat használjuk: aszinkron mód, 115 200 bps, 8-bit, 1 stop bit, nincs paritás, kétirányú (full duplex) átvitel, bár a vételt most nem használjuk, 16-szoros túlmintavételezés
- A **PA2** (Tx) és **PA3** (Rx) kivezetéseket használjuk, ezeket az **STM32Cube MX** automatikusan konfigurálja számunkra



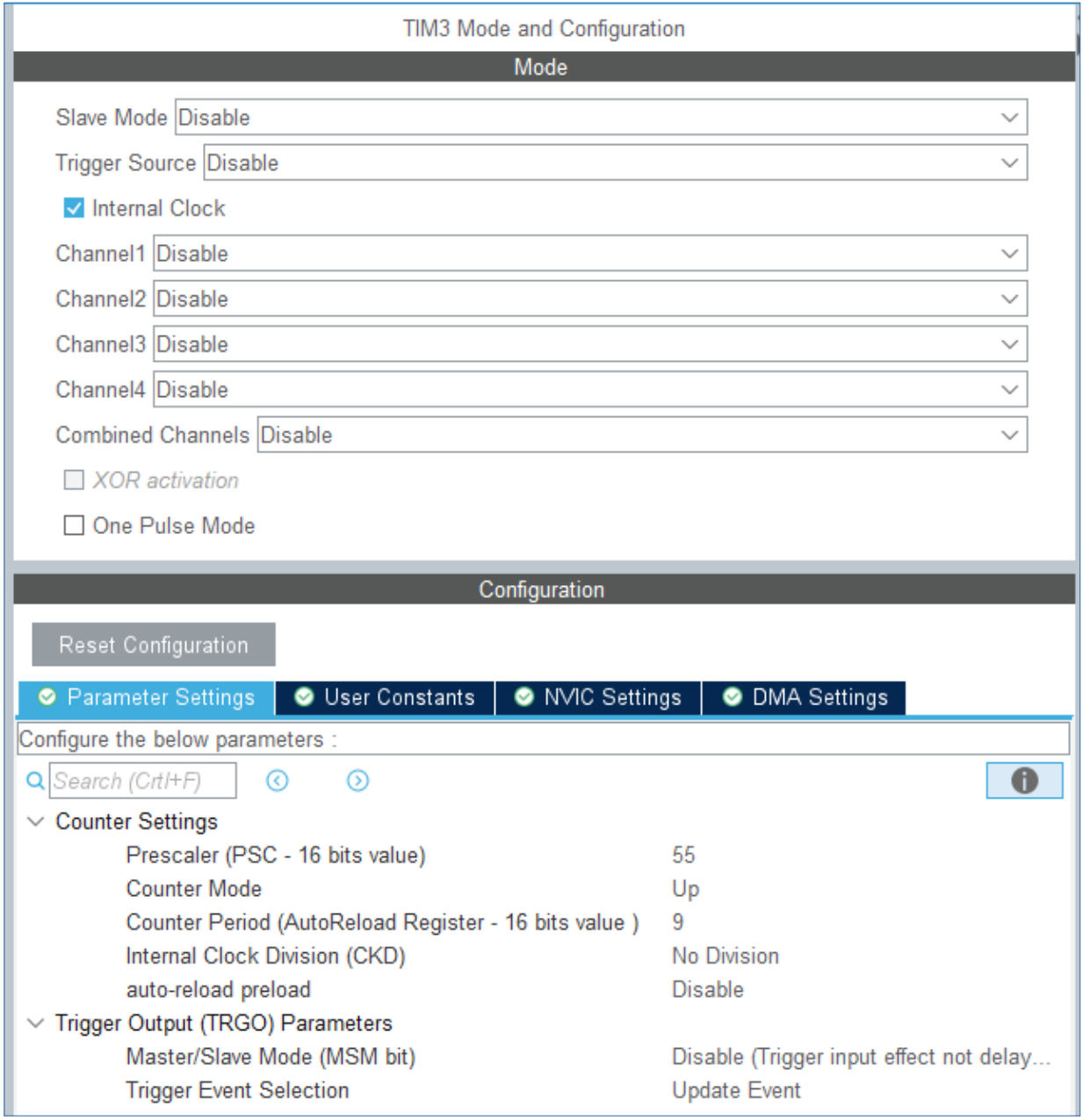
Az órajelek konfigurálása

- A 8 MHz-es kvarc jeléből a PLL 56 MHz-et állít elő, ezt 4-gyel osztva kapjuk a 14 MHz-es maximális ADC frekvenciát (ez biztosítja a maximális 1 MHz-es mintavételezési frekvenciát)
- Az időzítők (timers) bemenőjele is 56 MHz lesz



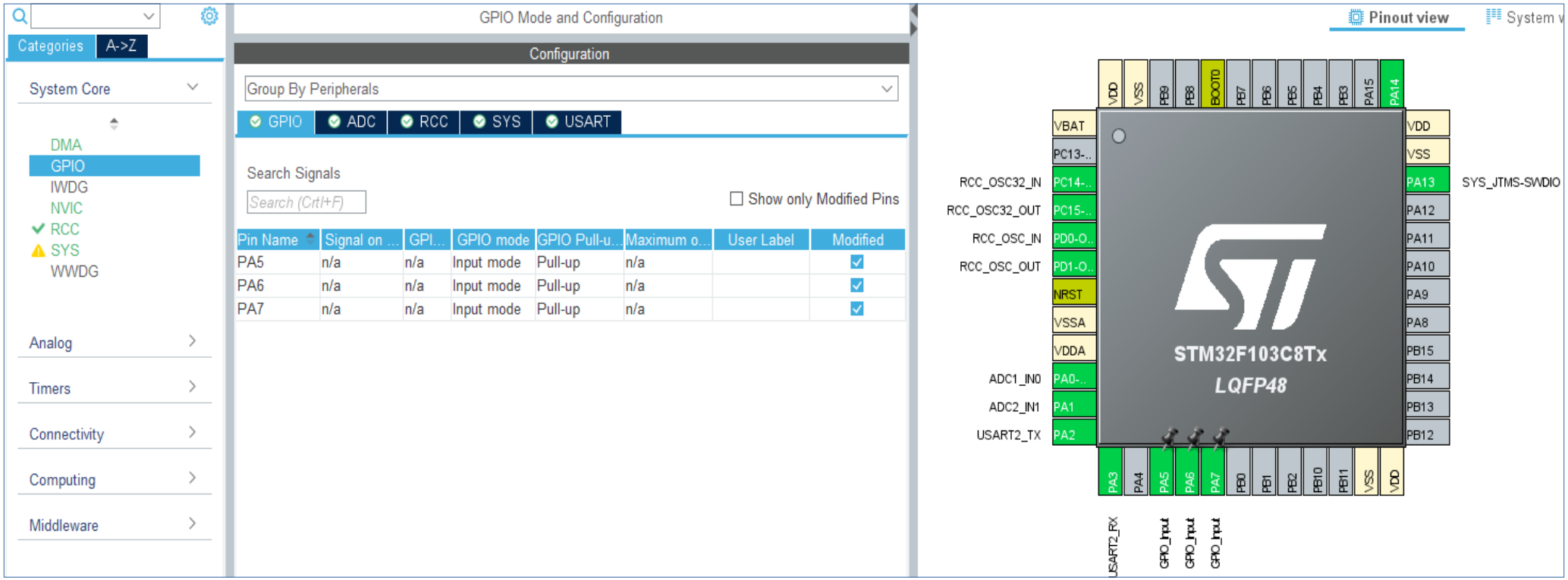
Timer3 konfigurálása

- Belső órajel (56 MHz)
- Előosztás: /56
- Periódus: /10 (ezt majd felülírjuk a kapcsolók beállítása szerint)
- Trigger kimenet: Update Event
- Timer3 itt nem használ mikrovezérlő kivezetést, belső összekötéseket használ…



GPIO kivezetések konfigurálása

- A **PA5, PA6** és **PA7** kivezetéseket digitális bemenetnek konfiguráljuk, belső felhúzással
- A kapcsolókat majd a bemenetek és a föld közé kötjük (lehúzás)



F103_ADC-2ch: main.c (részlet)

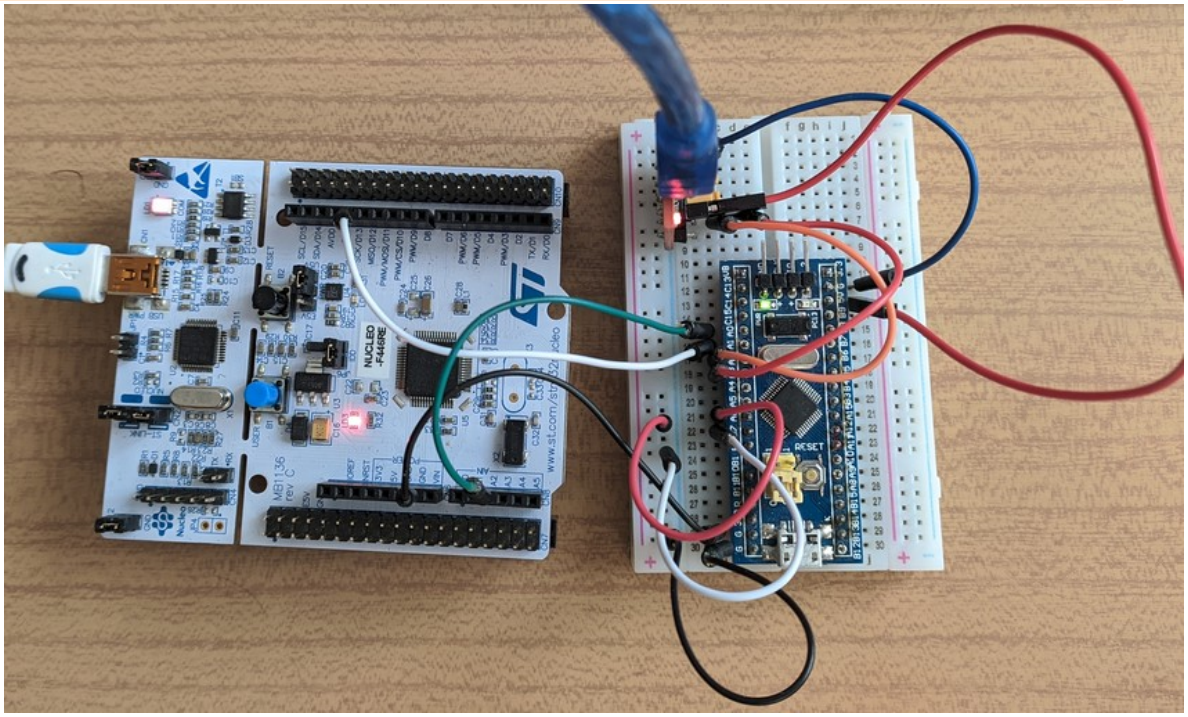
```c
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
DMA_HandleTypeDef hdma_adc1;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart2;
#define NDATA 500                                       // Mintavételek száma
const uint16_t period[] = {0,1,4,9,19,49,99,199};  // Tim3 period of 1, 2, 5, 10 ... us
uint16_t sw = 9;                                        // Alapértelmezett periódus (10 us)
volatile uint8_t data_ready = 0;
volatile uint32_t data[NDATA];
char msg[80];
uint32_t status;
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_TIM3_Init(void);
static void MX_USART2_UART_Init(void);

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
  data_ready = 1;                                        // Jelzi, hogy vége a mérésnek
}
```

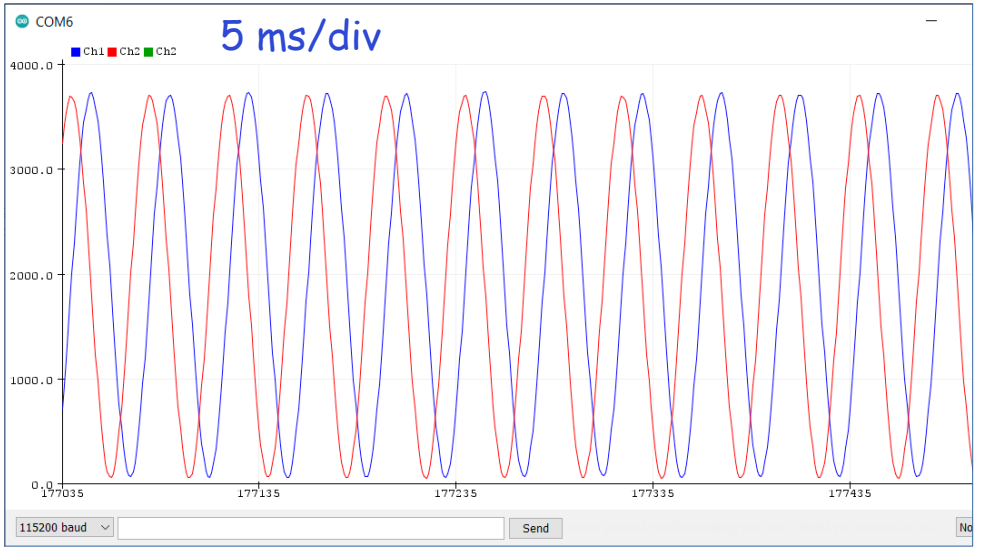
F103_ADC-2ch: main.c (részlet)

```
int main(void) {
  int d1, d2;
  HAL_Init();
  SystemClock_Config(); MX_GPIO_Init(); MX_DMA_Init();
  MX_ADC1_Init(); MX_ADC2_Init(); MX_TIM3_Init(); MX_USART2_UART_Init();
  sw = GPIOA->IDR;                    // A kapcsolók leolvasása
  sw = (sw>>5) & 7;                   // bit 5 -> bit 0
  TIM3->ARR = period[sw];
  HAL_TIM_Base_Start(&htim3);
  while (1) {
    data_ready = 0;
    HAL_ADC_Start(&hadc2);
    HAL_ADCEx_MultiModeStart_DMA(&hadc1, (uint32_t*)data,NDATA);
    while(!data_ready);
    status = HAL_ADCEx_MultiModeStop_DMA(&hadc1);
    HAL_ADC_Stop(&hadc2);
    sprintf(msg, "Ch1  Ch2\r\n");
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    for(int i = 0; i<NDATA; i++) {
      d1 = data[i] & 0x0FFF;
      d2 = data[i] >> 16;
      sprintf(msg, "%d  %d\r\n", d1, d2);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    HAL_Delay(5000);
  }
}
```

F103_ADC-2ch futási eredmény

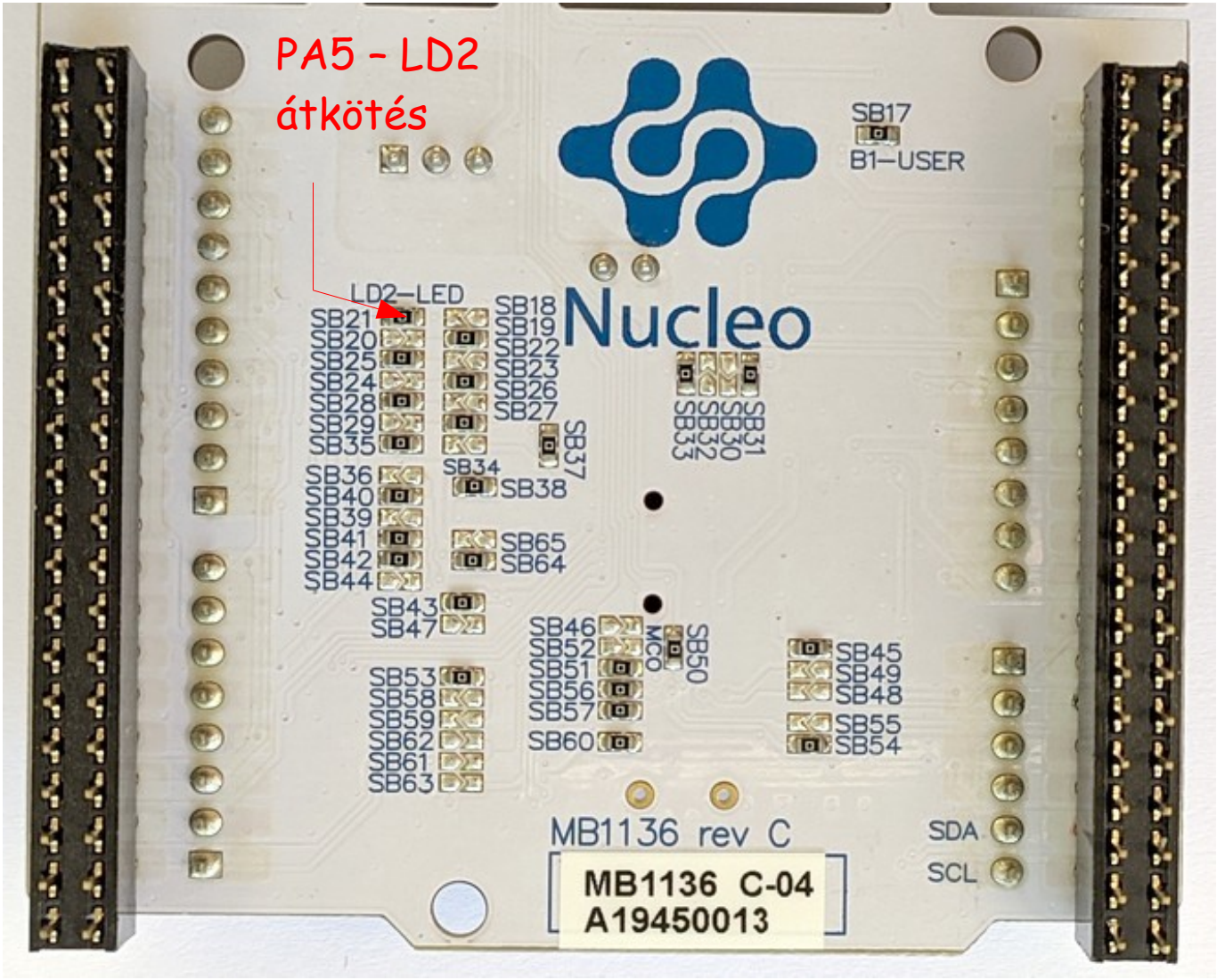
- Ebben az összeállításban kapcsolók helyet vezetékek átdugdosásával történik a mintavételi sebesség beállítása
- Az alábbi ábrákon az **F446RE_DAC_audio** program eredményét vizsgáltuk





A DAC 2. csatorna „felszabadítása”

- A **DAC** 2. csatornája arra a **PA5** kivezetésre csatlakozik, amelyre az **LD2** LED is rá van kötve. A LED-et az **SB21** átkötés leforrasztásával tudjuk leválasztani
- Az **SB** elnevezés egyébként az angol *solder bridge* rövidítése

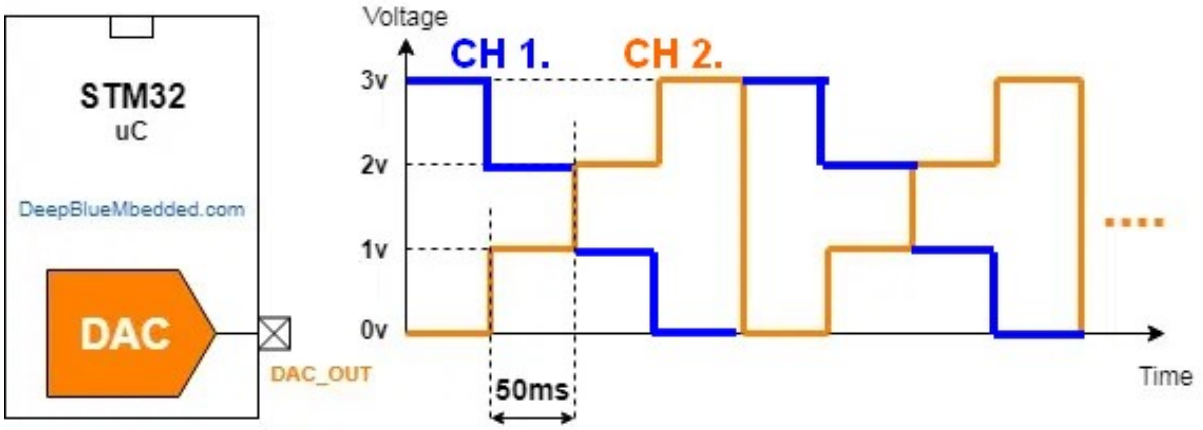


F446RE_DAC_basic2

- Ebben a programban a **DAC** 1. és 2. csatornáját (**PA4** és **PA5** kivezetés) szoftveresen vezéreljük a **HAL_DACEx_DualSetValue()** függvénnyel
- A kiírásokat 1 ms-onként végezzük, és az 1. csatornában a 3 V, 2 V, 1V, 0 V kimenő feszültséget állítunk be, a 2. csatornában pedig ugyanezt fordított sorrenben, s egy pici eltolással, s ezt a sorozatot ismételgetjük
- Az adott *U* feszültséghez kiküldendő adatok, az képlet alapján: 0, 1241, 2482, illetve 3723

$$N_{DAC} = \frac{U \cdot 4096}{V_{REF}}$$

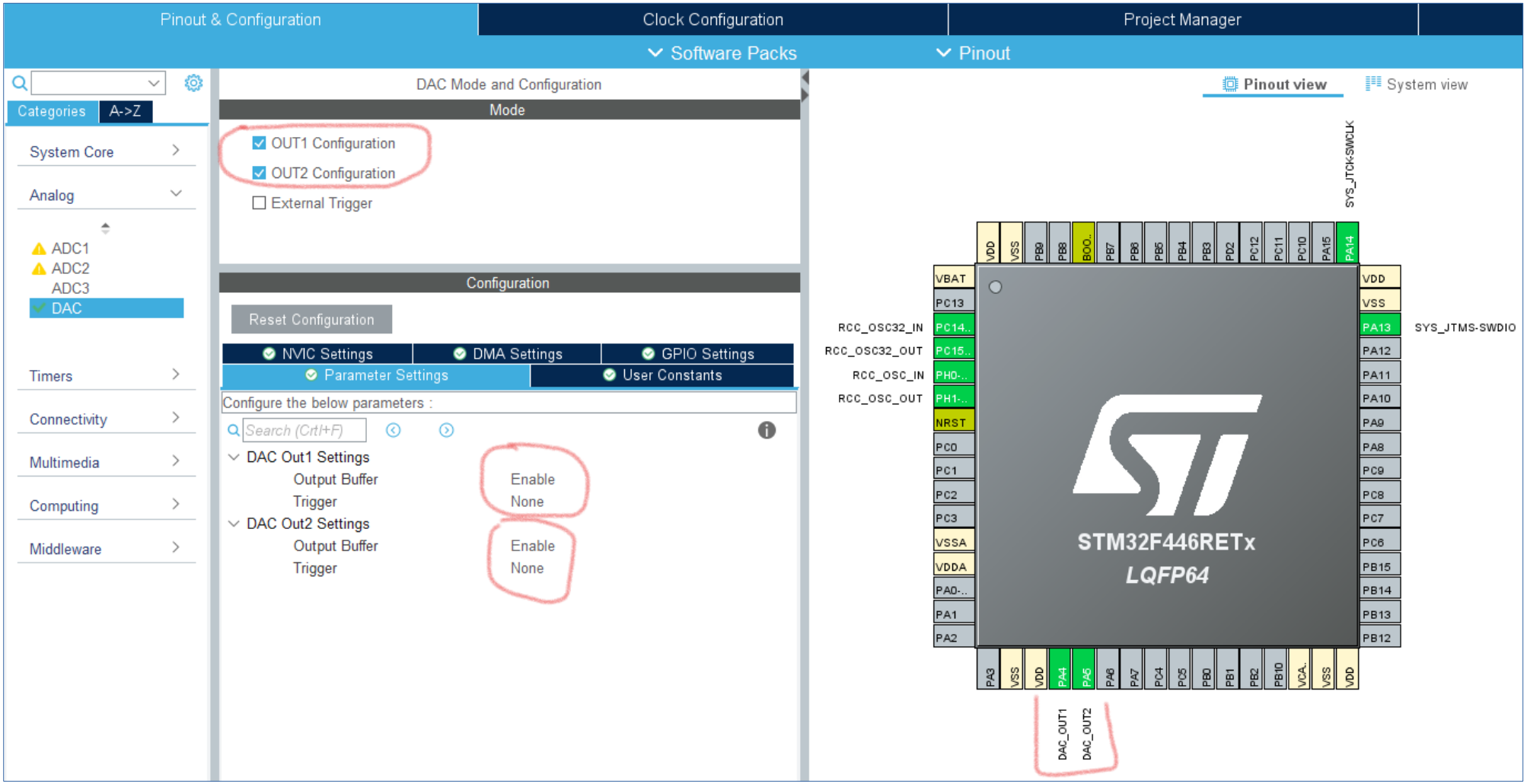
- Az eltolás 200 (kb. 0,16V). Esetünkben VREF a tápfeszültség, azaz 3,3 V



- Programötlet: Khaled Magdy : STM32 DAC Tutorial

A DAC modul konfigurálása

- Mindkét csatornát konfiguráljuk (**PA4** és **PA5** kivezetések)
- A kimeneti buffert engedélyezzük, triggerforrást nem definiálunk

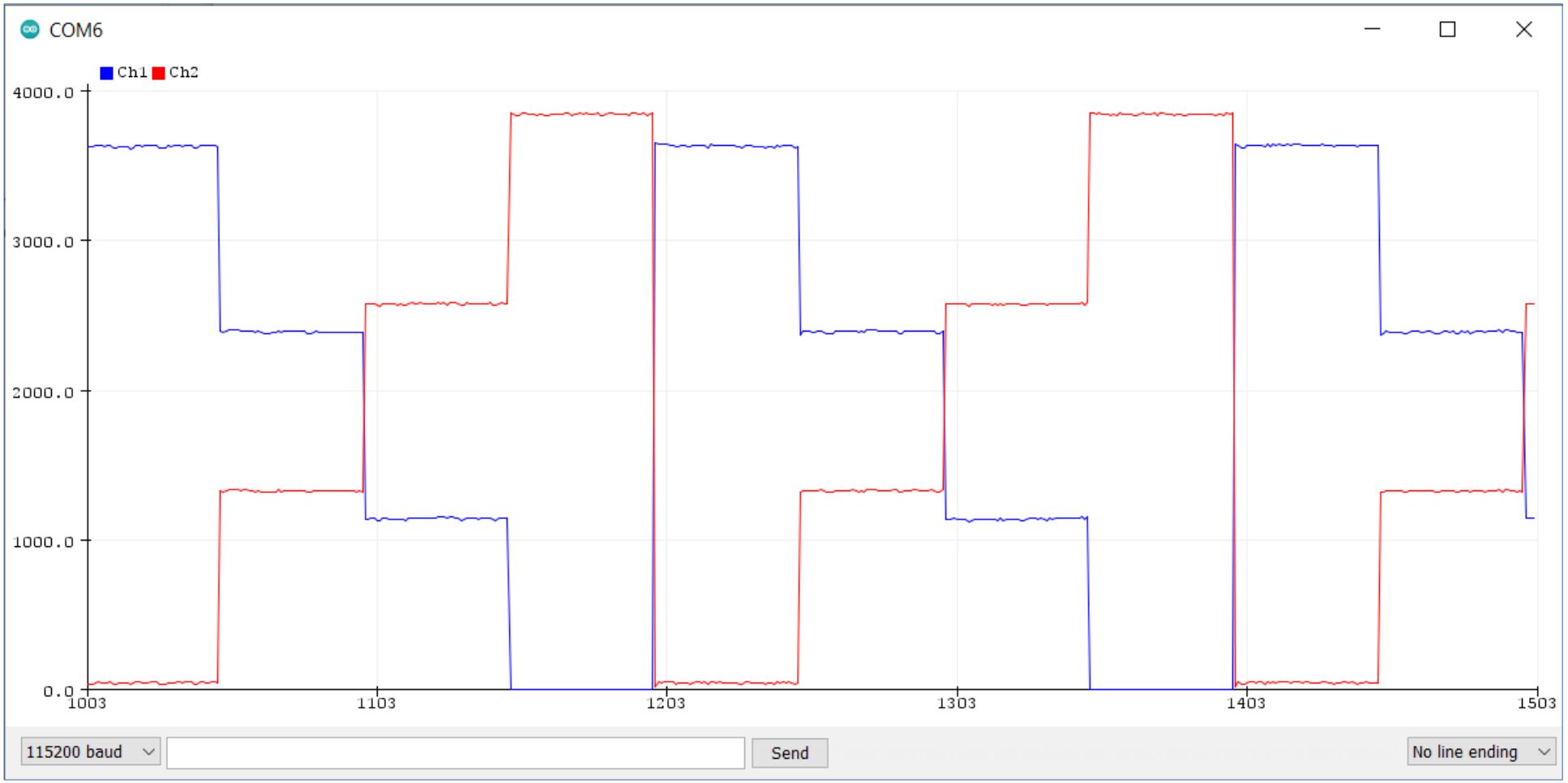


F446RE_DAC_basic2: main.c releváns sorai

```c
#include "main.h"
DAC_HandleTypeDef hdac;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DAC_Init(void);

int main(void) {
  uint32_t DAC_OUT[4] = {0, 1241, 2482, 3723};  // 0V, 1V, 2V, 3V
  uint8_t i = 0;                                // Számláló
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DAC_Init();
  HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
  HAL_DAC_Start(&hdac,DAC_CHANNEL_2);
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, 200);
  while (1) {
    HAL_DACEx_DualSetValue(&hdac, DAC_ALIGN_12B_R, DAC_OUT[3-i], DAC_OUT[i++] + 200);
    i = i & 0x03;                               // csak 0, 1, 2 vagy 3 lehet!
    HAL_Delay(0);                               // 1 ms delay!!!
  }
}
```

F446RE_DAC_basic2: futási eredmény

- Az ábrán 20 µs-onként történt a mintavételezés, így egy osztás (100 adat) 2 ms, a teljes ábra pedig 10 ms időtartamot fed le

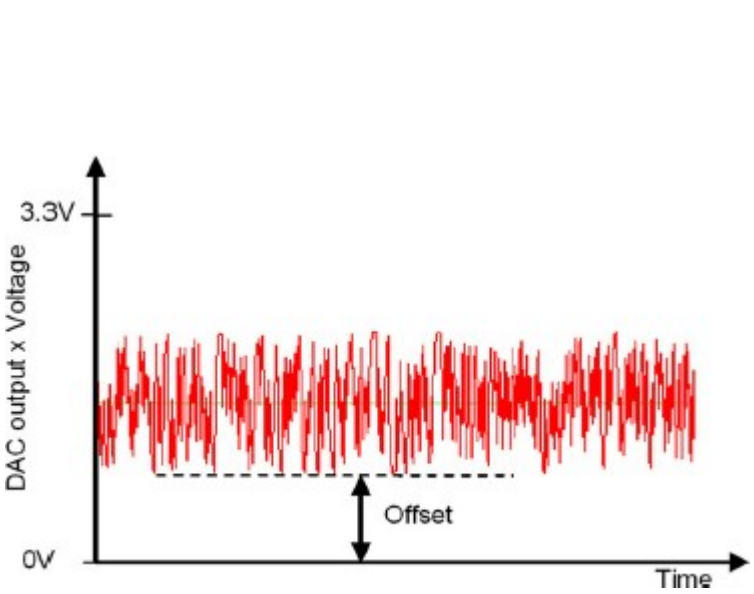


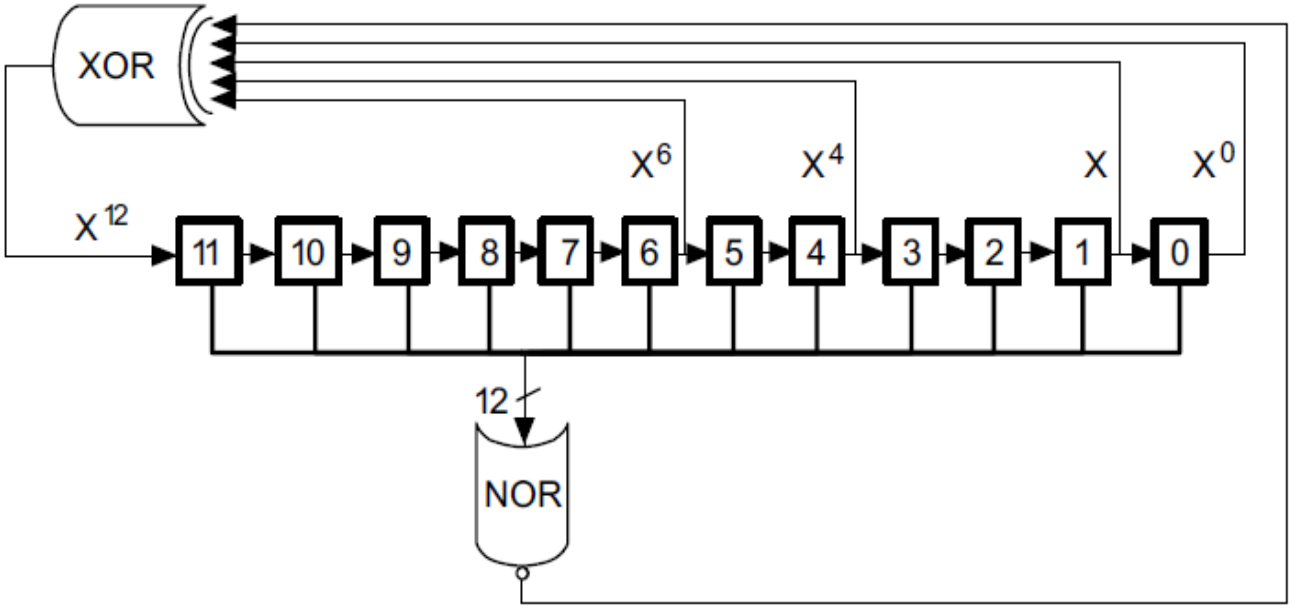
Zajongjunk! F446RE_DAC_noisegen

- Az előző programot bővítsük azzal, hogy a **DAC** 2. csatorna jeléhez álvéletlen fehérzajt keverünk hozzá, 256-os amplitúdóval
- A **DAC** 1. csatorna jeléhez pedig hozzáadunk 128-at, hogy a zaj középvonalát kijelölje
- A zajgenerátor használatához azonban trigger jelforrásra is szükség van (ugyanúgy, mint az előző előadásban a háromszögjel generátornál) – erre **Timer6**-ot használjuk fel, 10 µs periódussal
- Végeredményben tehát a `HAL_DACEx_DualSetValue()` függvényhívással 1 ms-onként szimultán beírjuk a 0 V, 1V, 2 V, 3 V feszültségnek megfelelő értéket (az 1. csatornában 128-cal megnövelve), amelyhez mint alapjelhez a 2. csatornában hozzáadódik a zajgenerátor jele (ami 10 µs-onként vált értéket)

Emlékeztető: Zajgenerátor mód

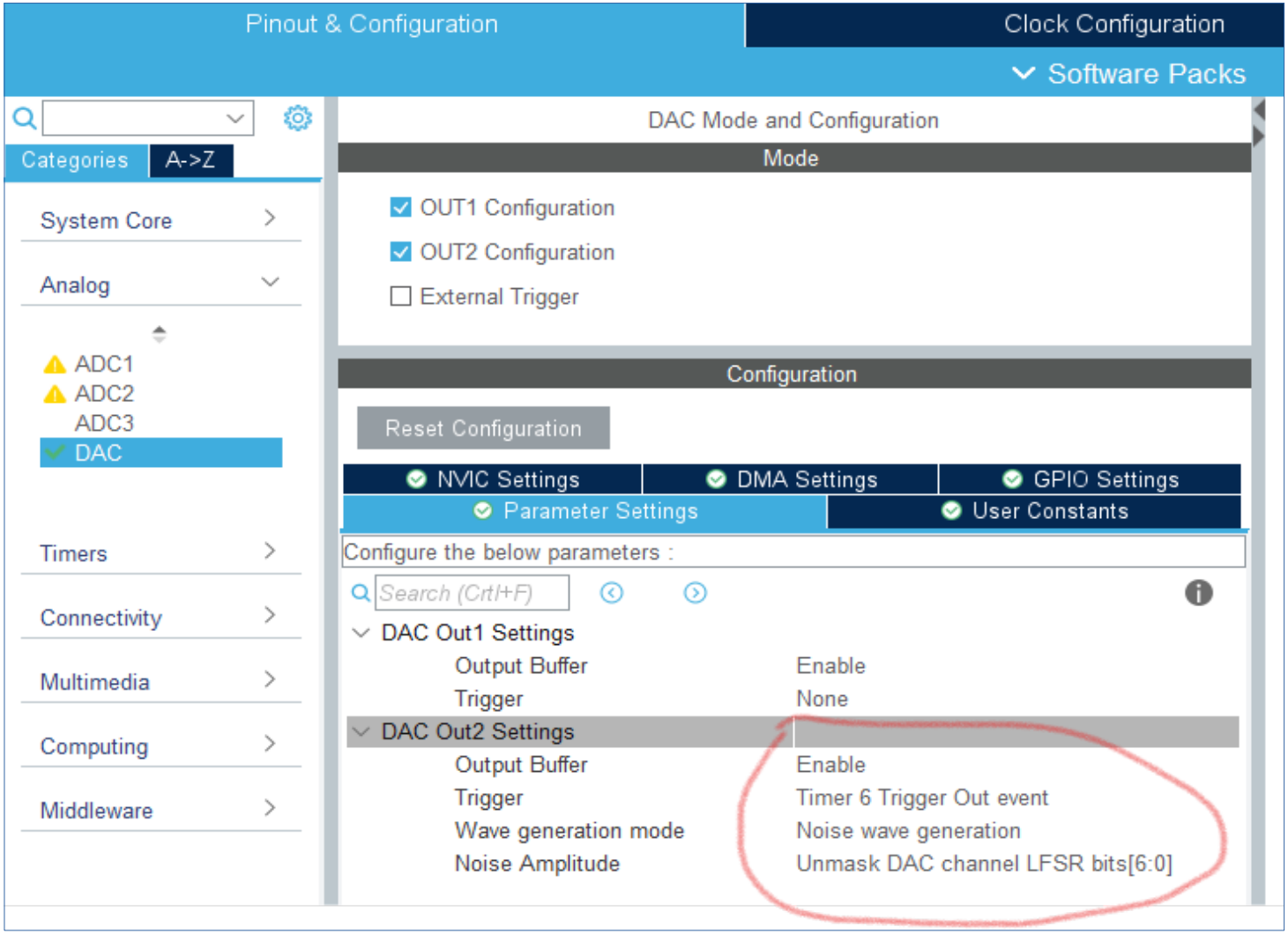
- A **DAC** vezérlő regiszterében **WAVE*x***[1:0] = 01 beállítás bekapcsolja a zajjel generátort, amely egy álvéletlen számot ad hozzá a **DHR*x*** regiszter tartalmához (az *offset-hez*), s ez az összeg kerül a kimeneti adatregiszterbe
- Az álvéletlen számot egy lineárisan visszacsatolt shift regiszter (LFSR) állítja elő
- Az amplitúdót (vagyis a felhasznált bitek számát) a **MAMP*x***[3:0] bitcsoport beállítása szabja meg
- Az *offset* és a maximális amplitúdó összege 4095-nél nem lehet nagyobb!





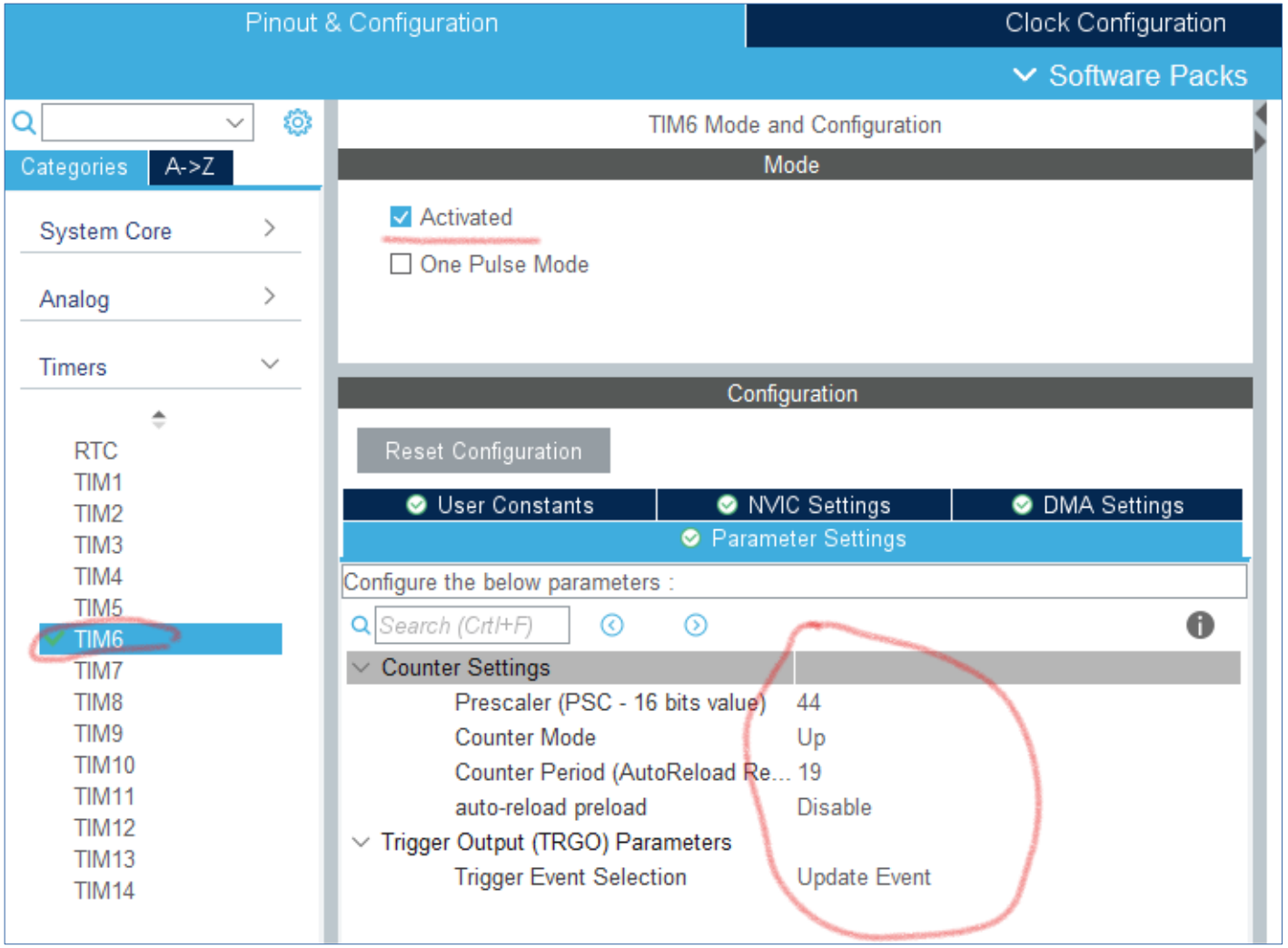
A DAC konfigurálása

- A DAC 2. csatornája esetében engedélyezni kell a trigger jelforrást
- Ha a zajjel generátort itt konfiguráljuk, annak beállítása automatikusan belekerül az `MX_DAC_Init()` függvénybe
- A triggerjelet szolgáltató egységet (**Timer6**) viszont nekünk kell elindítani a jelgenerátor beindításához



Timer6 konfigurálása

- Aktiváljuk, megadjuk az előosztást és a számlálási határt (periódus)
- A jelgenerátort triggerelő kimenetnek az ***Update eseményt*** adjuk meg



F446RE_DAC_noisegen: main.c (részlet)

```c
#include "main.h"
DAC_HandleTypeDef hdac;
TIM_HandleTypeDef htim6;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DAC_Init(void);
static void MX_TIM6_Init(void);

int main(void) {
  uint32_t DAC_OUT[4] = {0, 1241, 2482, 3723};
  uint8_t i = 0;
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DAC_Init();
  MX_TIM6_Init();
  HAL_DAC_Start(&hdac,DAC_CHANNEL_1);
  HAL_DAC_Start(&hdac,DAC_CHANNEL_2);
//HAL_DACEx_NoiseWaveGenerate(&hdac, DAC_CHANNEL_2,  DAC_LFSRUNMASK_BITS7_0);
  HAL_TIM_Base_Start(&htim6);
  while (1) {
    HAL_DACEx_DualSetValue(&hdac, DAC_ALIGN_12B_R, DAC_OUT[i]+128, DAC_OUT[i]);
    i = (i + 1) & 0x03;
    HAL_Delay(0);          // 1 ms delay!!!!
  }
}
```

F446RE_DAC_noisegen: futási eredmény

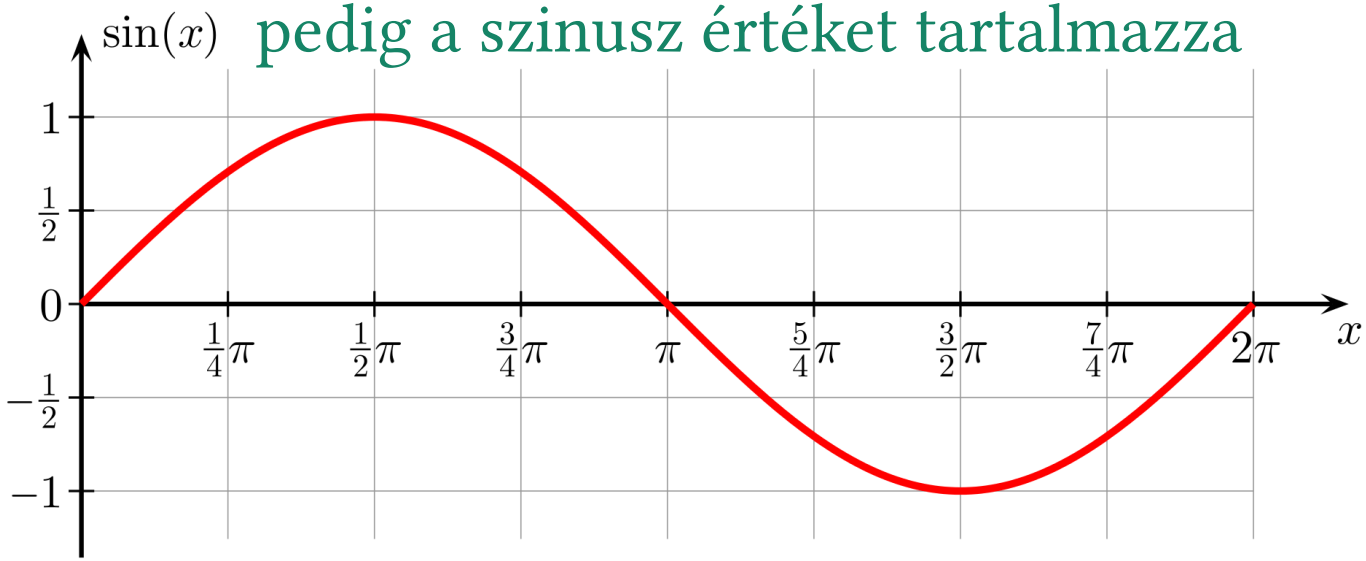
- Az ábrán az 5 µs-onként mintavételezett DAC kimenőjelek láthatók
- Az 1. csatorna jelét a zaj amplitúdó felével megemeltük (128)

F446RE_DAC_audio

- Az utolsó programban a **DAC** 1. és 2. csatornáját (**PA4** és **PA5** kivezetés) együttesen egy hullámtáblából töltögetjük a **DMA1** modul segítségével (stream 5 vagy stream 6), az ütemezésről pedig **Timer6** gondoskodik
- A kiírásokat 10 µs-onként végezzük, a 200 adatot tartalmazó **wavetable** tömbben kombináltan egy periódusnyi szinusz- illetve koszinuszhullámot tárolunk (a függvényértékeket 0 – 4095 közé normálva). A 32 bites adatok felső fele a koszinusz, az alsó fele pedig a szinusz értéket tartalmazza

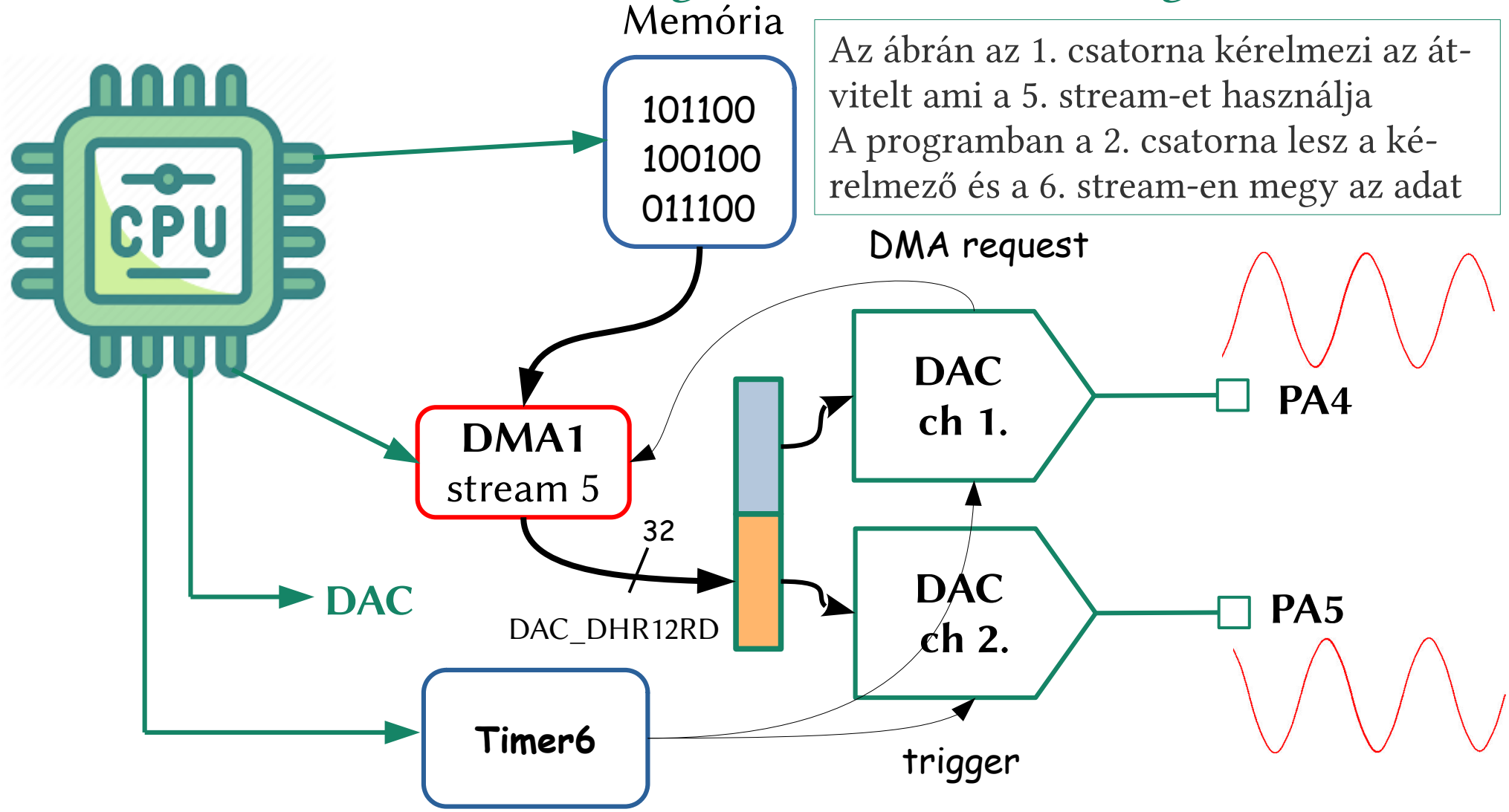
$$Freq = \frac{F_{CPU}}{Psc \cdot Period \cdot NDATA}$$

$$Freq = \frac{90\,MHz}{45 \cdot 20 \cdot 200} = 500\,Hz$$



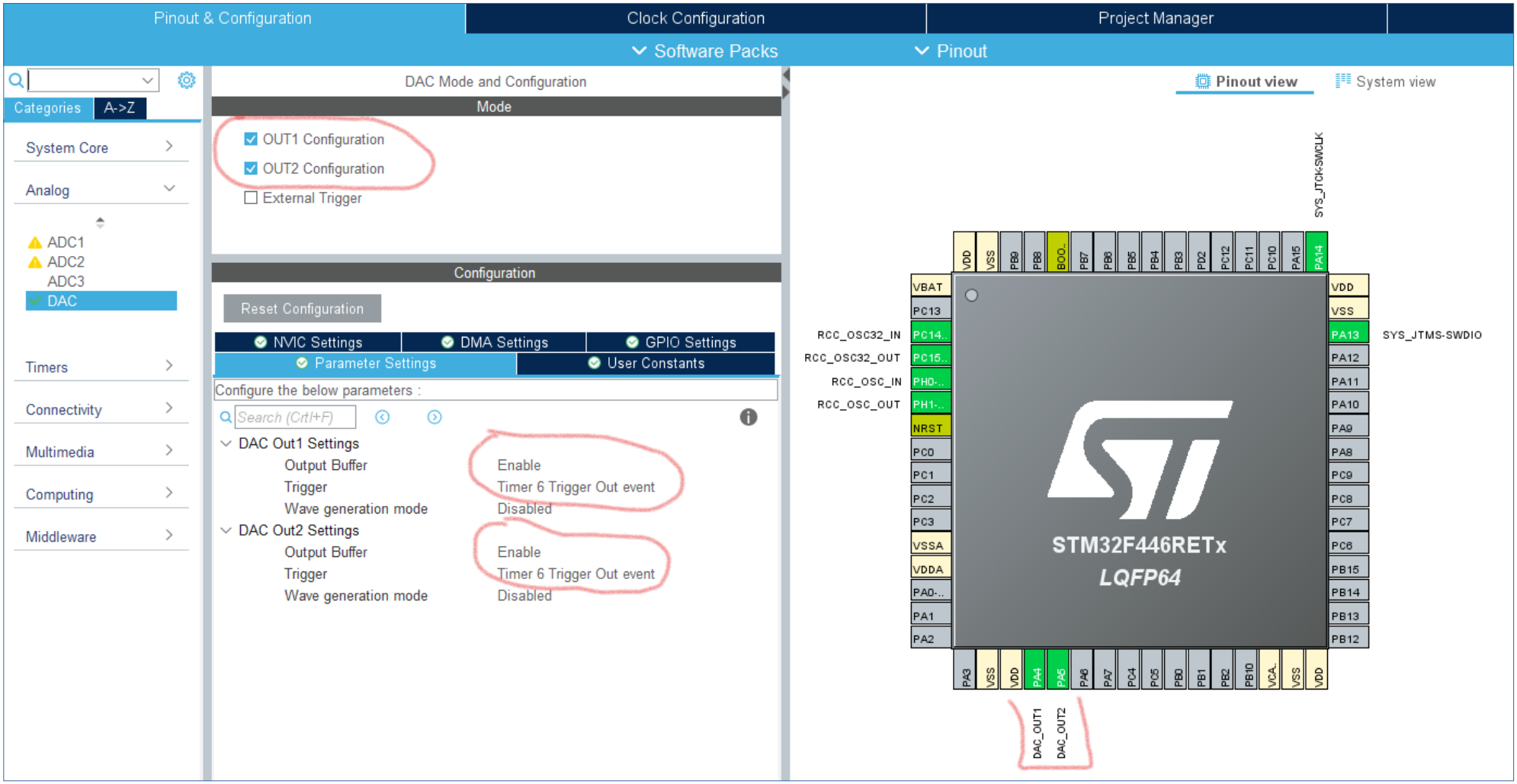
Az adatáramlás sematikus vázlata

- A konverziót **Timer6** triggereli, a DAC pedig adatot kér a DMA-tól
- A DMA 32 bites adatokat mozgat a **DAC_DHR12RD** regiszter felé



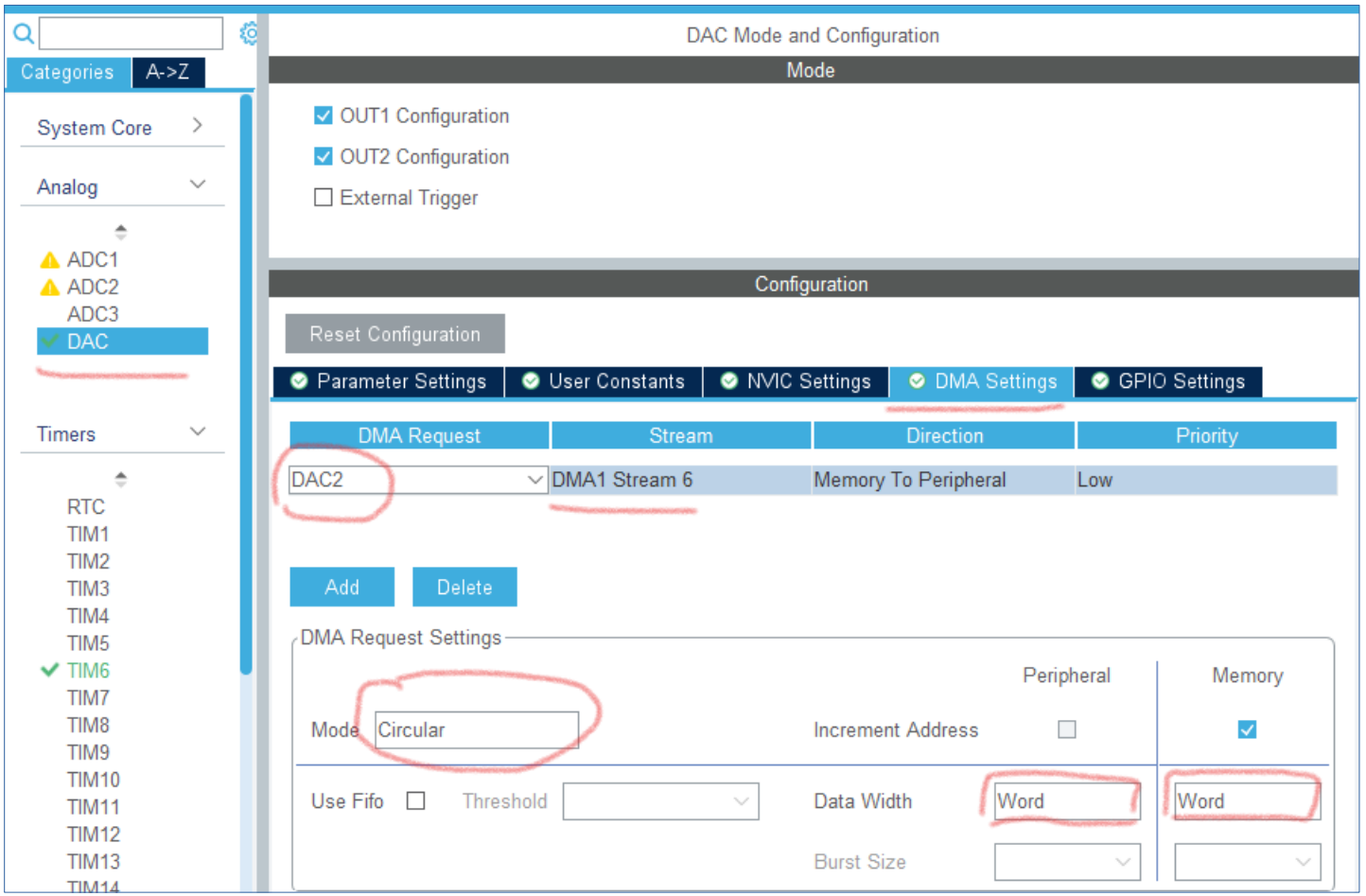
A DAC konfigurálása

- Változás az előző programhoz képest, hogy trigger jelforrást is kell választani, majd a *DMA Settings* fülre kattintva aktiválni kell a *DMA1 Stream 5, vagy Stream 6* csatornát



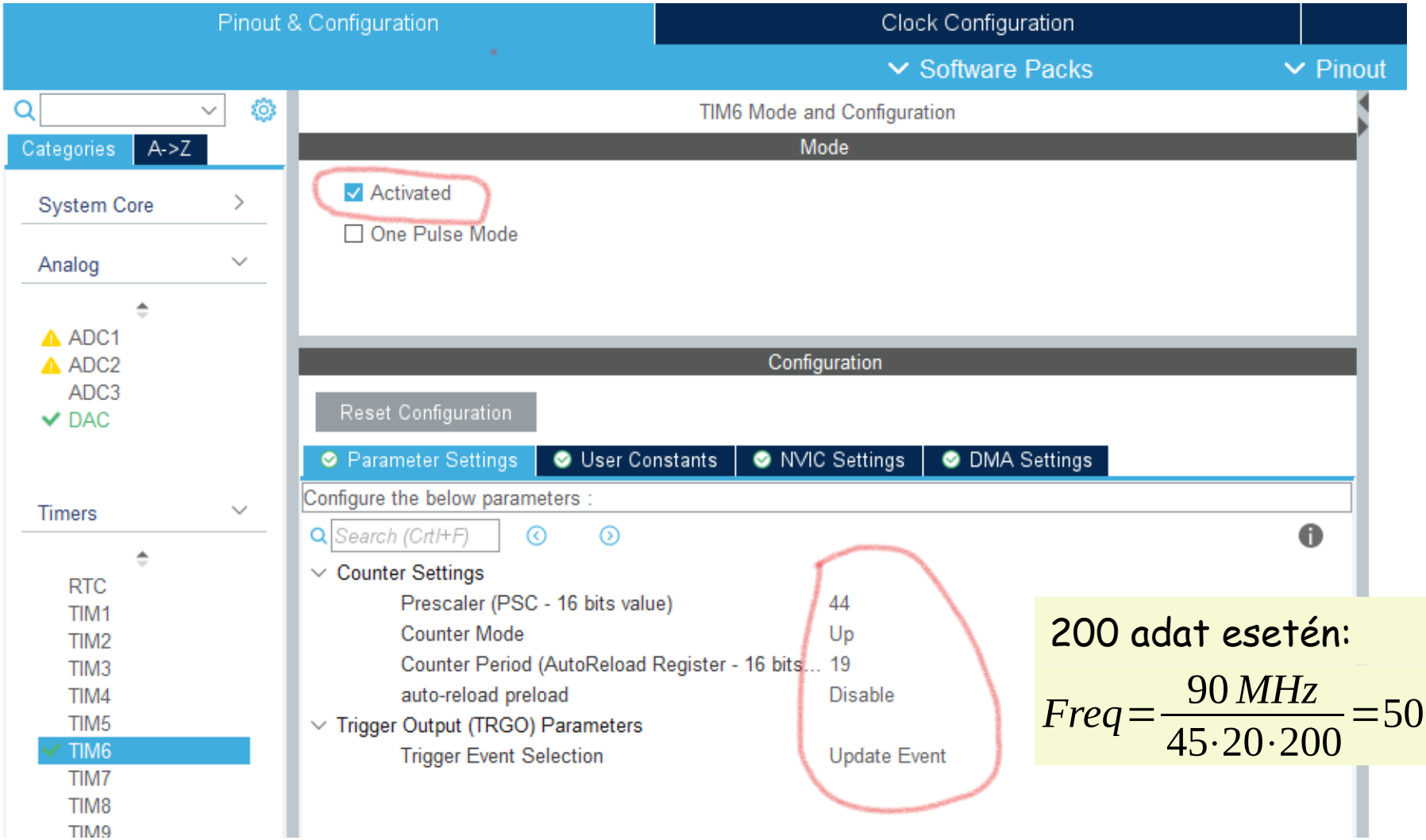
DAC – DMA kapcsolat konfigurálása

- Az **ADD** gombbal aktiváltuk a *DAC2 stream 6*-ot (*circular* mód!)
- Az átvitel **Memória → periféria** irányú, 32-bites adatszélességgel
- Címléptetés csak a memóriánál kell…



Timer6 konfigurálása

- Aktiváljuk, megadjuk az előosztást és a számlálási határt (periódus)
- A **DAC**-ot triggerelő kimenetnek az *Update eseményt* adjuk meg



200 adat esetén:

$$Freq = \frac{90\,MHz}{45 \cdot 20 \cdot 200} = 500\,Hz$$

F446RE_DAC_audio: main.c releváns sorai

```
#include "main.h"
#include <string.h>
#include <math.h>
DAC_HandleTypeDef hdac;
DMA_HandleTypeDef hdma_dac2;
TIM_HandleTypeDef htim6;
#define PI    3.14159
#define NDATA 200
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_DAC_Init(void);
static void MX_TIM6_Init(void);

int main(void) {
  uint32_t wavetable[NDATA], value, v1, v2;
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  ...
```

F446RE_DAC_audio: main.c releváns sorai

```
  MX_DMA_Init();
  MX_DAC_Init();
  MX_TIM6_Init();
  for (uint16_t i = 0; i < NDATA; i++) {
    value = (uint16_t) rint((sinf(((2*PI)/NDATA)*i)*1800) + 2000);
    v1 = value < 4096 ? value : 4095;
    value = (uint16_t) rint((cosf(((2*PI)/NDATA)*i)*1800) + 2000);
    v2 = value < 4096 ? value : 4095;
    wavetable[i] = v1 + (v2 << 16);
  }
  HAL_DAC_Init(&hdac);
  HAL_TIM_Base_Start(&htim6);
  HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
  HAL_DAC_Start_Dual_DMA(&hdac, DAC_CHANNEL_2, (uint32_t*)wavetable, NDATA,
DAC_ALIGN_12B_R);

  while (1)
  {
  }
}
```

A HAL_DAC_Start_Dual_DMA() függvény

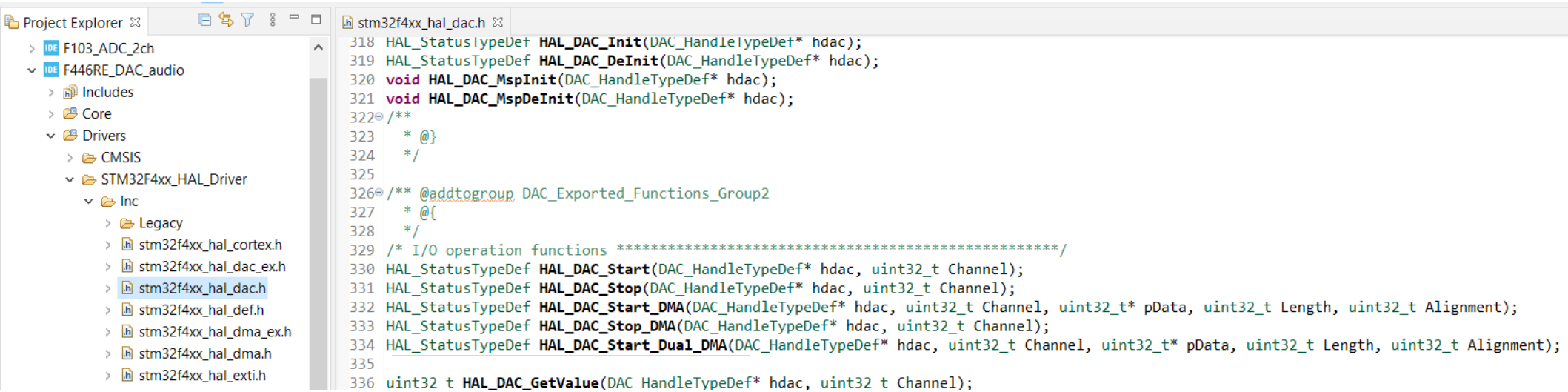
- A **hal_dac.c** nevű forrásállományban általunk létrehozott **HAL_DAC_Start_Dual_DMA()** függvény lényegében a **HAL_DAC_Start_DMA()** függvény másolata, amelyen csupán néhány apró változtatást eszközöltünk: azt kell elérni, hogy akár `DAC_CHANNEL_1`, akár `DAC_CHANNEL_2` szerepel híváskor a felbontást és jobbra-balra igazítást megadó ***Aligment*** paramétertől függően a megfelelő regisztercím (`DHR8RD`, `DHR12RD`, `DHR12LD` valamelyike) legyen a periféria cím
- Szintaxis:

```
HAL_StatusTypeDef HAL_DAC_Start_Dual_DMA( DAC_HandleTypeDef* hdac,
uint32_t Channel, uint32_t* pData, uint32_t Length, uint32_t Alignment)
```

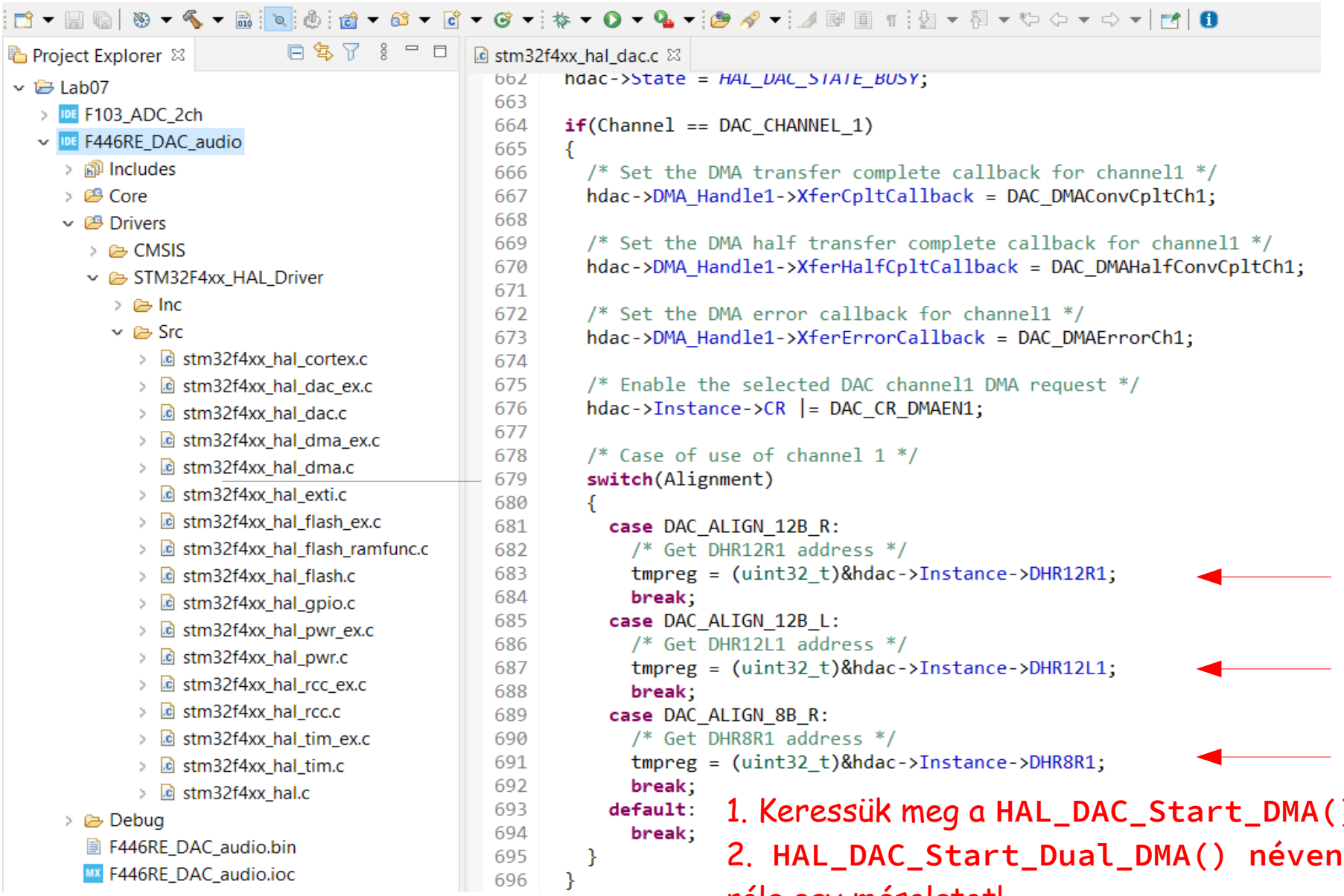
  - ***hdac*** – az eszköz (DAC) „fogantyúja”
  - ***Channel*** – valamelyik DAC csatorna
  - ***pData*** – az adatokra mutató pointer
  - ***Lenght*** – az adatok száma
  - ***Alignment*** – az igazítás (`DAC_ALIGN_12B_R`, `DAC_ALIGN_12B_L`, vagy `DAC_ALIGN_8B_R`)

A hal_dac.h állomány bővítése

- A projekt `Drivers` mappájában az `STM32F4xx_HAL_Driver/inc` almappába keressük meg a hal_dac.h fejléc állományt és deklaráljuk benne a `HAL_DAC_Start_Dual_DMA()` függvényt!



A hal_dac.c állomány bővítése



1. Keressük meg a `HAL_DAC_Start_DMA()` függvényt!
2. `HAL_DAC_Start_Dual_DMA()` néven készítsünk róla egy másolatot!
3. Írjuk át a regiszter hivatkozásokat!

HAL_DAC_Start_Dual_DMA() 2/1.

```
HAL_StatusTypeDef HAL_DAC_Start_Dual_DMA(DAC_HandleTypeDef* hdac,
    uint32_t Channel, uint32_t* pData, uint32_t Length, uint32_t Alignment) {
    uint32_t tmpreg = 0U;
  . . .
  if(Channel == DAC_CHANNEL_1) {
    . . .
    switch(Alignment) {
      case DAC_ALIGN_12B_R:
        /* Get DHR12R1 address */
        tmpreg = (uint32_t)&hdac->Instance->DHR12RD;
        break;
      case DAC_ALIGN_12B_L:
        /* Get DHR12L1 address */
        tmpreg = (uint32_t)&hdac->Instance->DHR12LD;
        break;
      case DAC_ALIGN_8B_R:
        /* Get DHR8R1 address */
        tmpreg = (uint32_t)&hdac->Instance->DHR8RD;
        break;
      default:
        break;
    }
  }
```

Kicseréltük a regiszterek nevét a duál módú beíráshoz

Megjegyzés: **tmpreg** a DMA átvitelhez a periféria címet tartalmazza

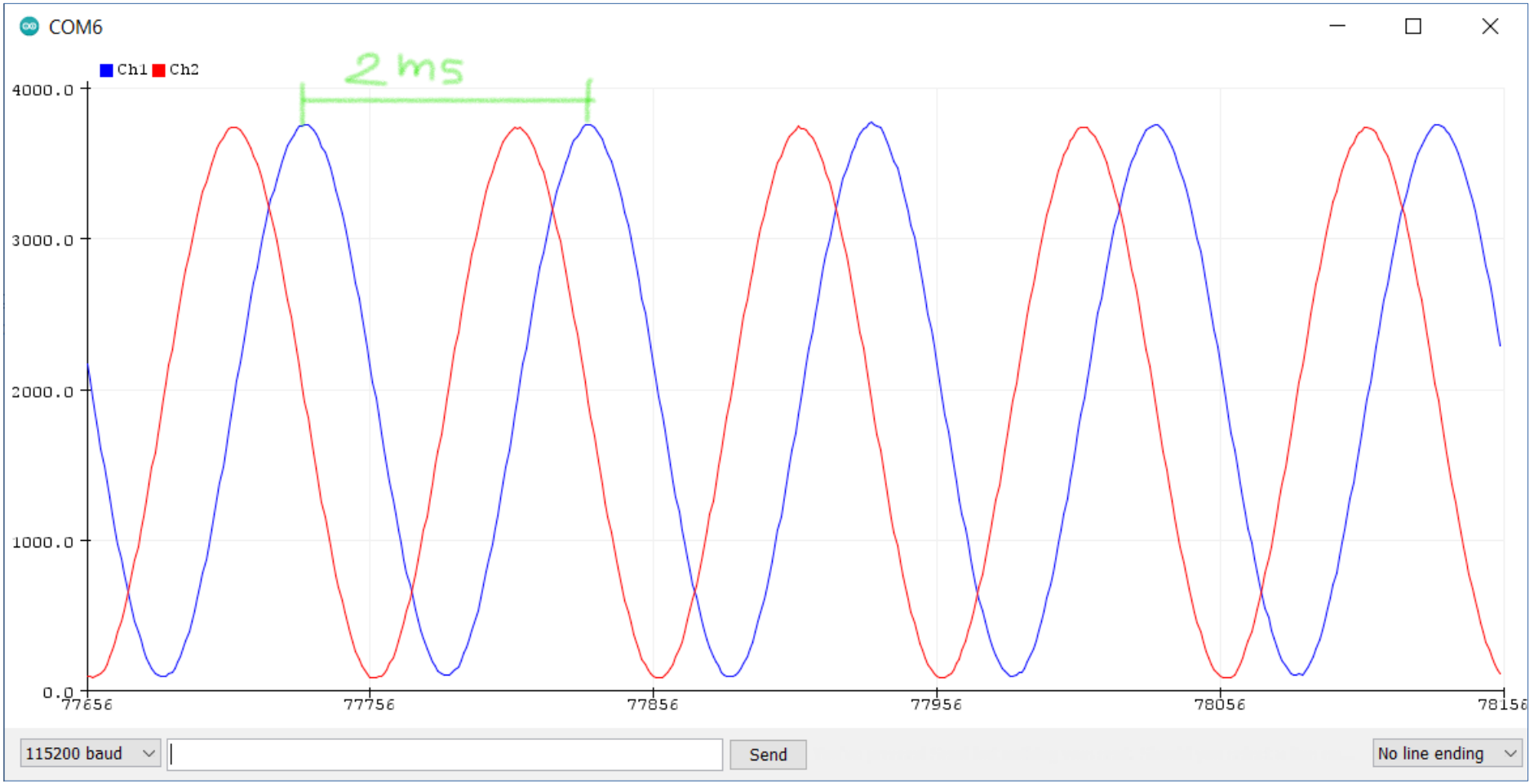
HAL_DAC_Start_Dual_DMA() 2/2.

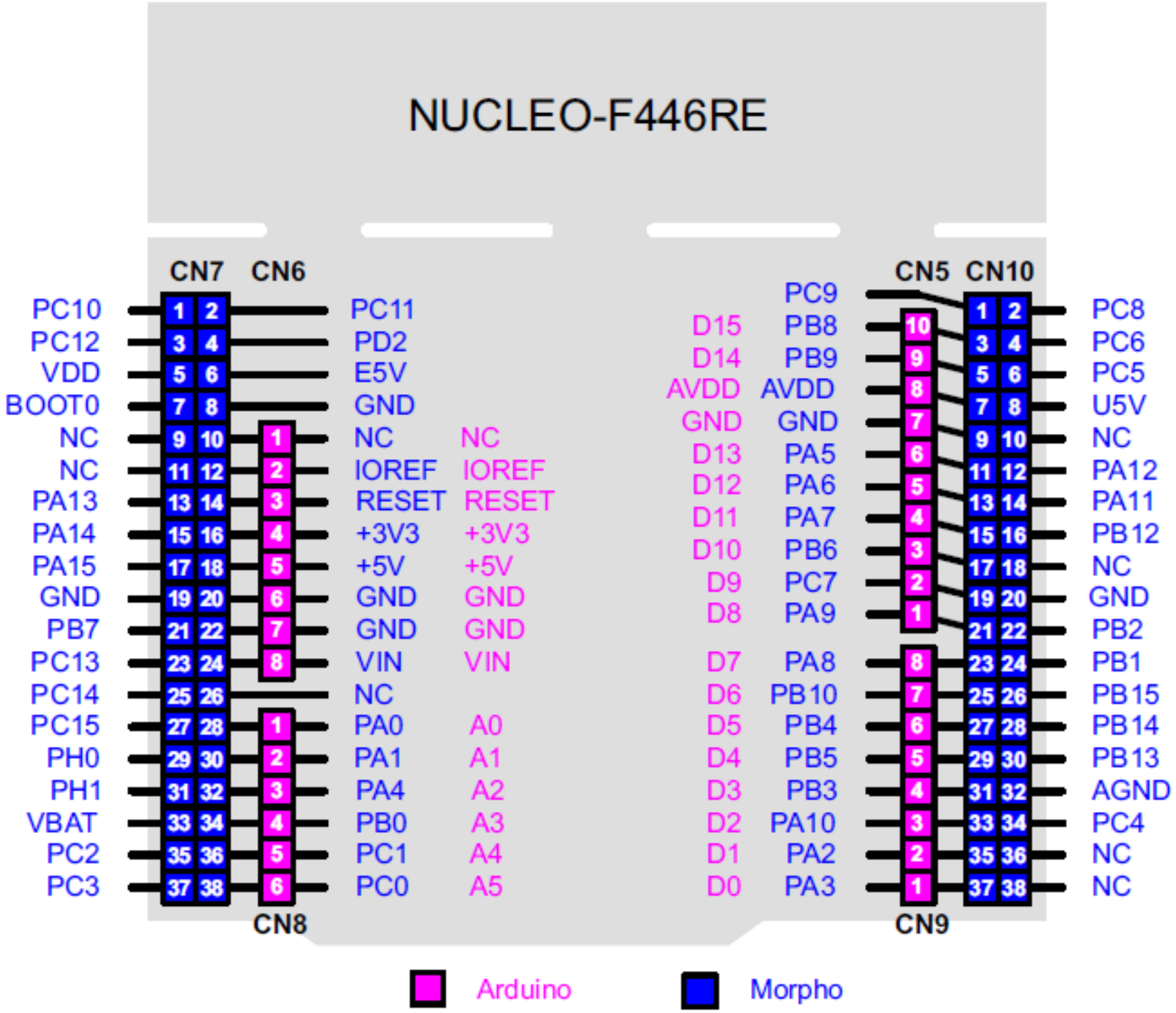
```
else  {
  . . .
  /* Case of use of channel 2 */
  switch(Alignment)
  {
    case DAC_ALIGN_12B_R:
      /* Get DHR12R2 address */
      tmpreg = (uint32_t)&hdac->Instance->DHR12RD;
      break;
    case DAC_ALIGN_12B_L:
      /* Get DHR12L2 address */
      tmpreg = (uint32_t)&hdac->Instance->DHR12LD;
      break;
    case DAC_ALIGN_8B_R:
      /* Get DHR8R2 address */
      tmpreg = (uint32_t)&hdac->Instance->DHR8RD;
      break;
    default:
      break;
  }
}
```

Kicseréltük a regiszterek nevét a duál módú beíráshoz

F446RE_DAC_audio: futási eredmény

- A kimenőjelet az első példaprogramban ismertetett „kétcsatornás oszcilloszkóp” segítségével ellenőrizhetjük
- A mintavételezés 20 µs-onként történt (2 ms / osztás)




NUCLEO-F446RE
CN7
CN6
CN5
CN10
CN8
CN9
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
PC9
D15 PB8
D14 PB9
AVDD AVDD
GND GND
D13 PA5
D12 PA6
D11 PA7
D10 PB6
D9 PC7
D8 PA9
D7 PA8
D6 PB10
D5 PB4
D4 PB5
D3 PB3
D2 PA10
D1 PA2
D0 PA3
PC8
PC6
PC5
U5V
NC
PA12
PA11
PB12
NC
GND
PB2
PB1
PB15
PB14
PB13
AGND
PC4
NC
NC
Arduino
Morpho