STM32 mikrovezérlők programozása STM32CubeIDE környezetben

6. Digitál-analóg konverter, egyszerű időzítők

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**

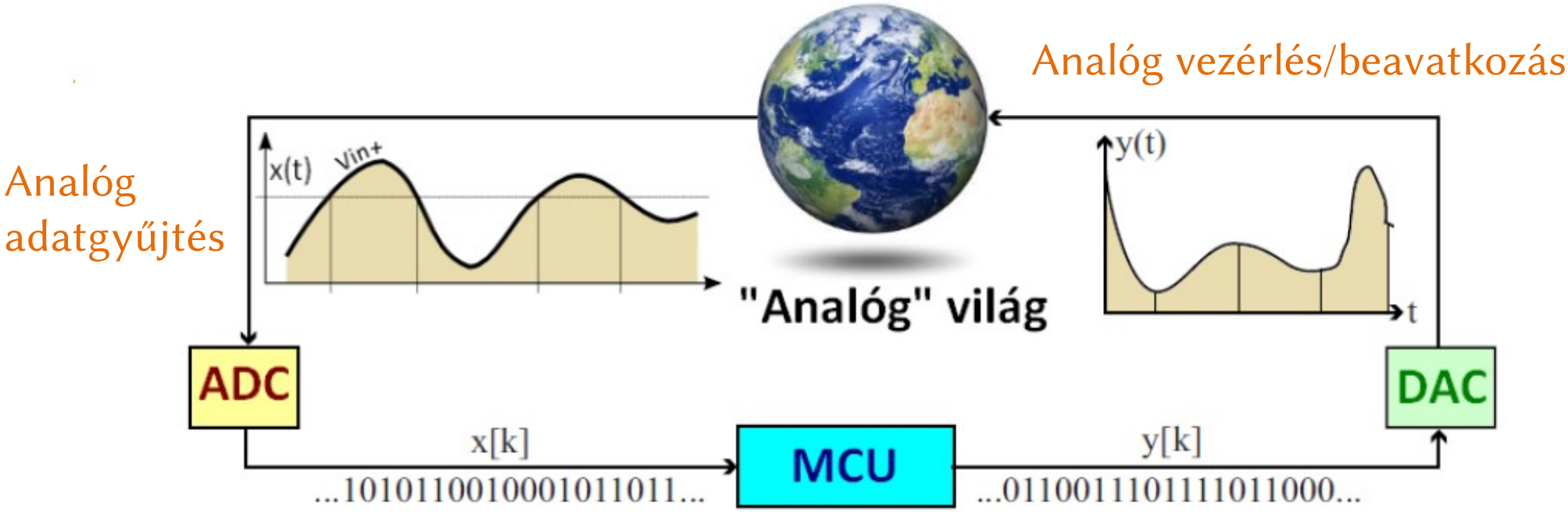
- Khaled Magdy (DeepBlue):
  STM32 DAC Tutorial – Example HAL Code & Analog Signal Generation

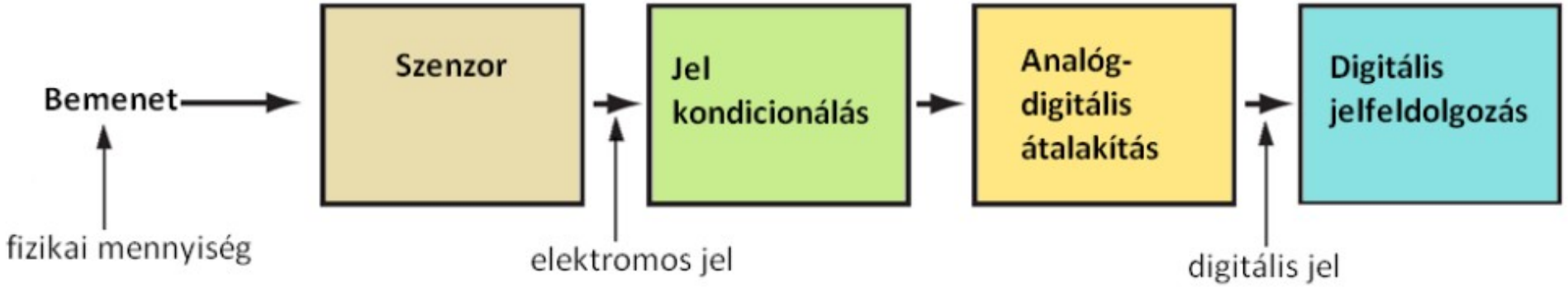
Adatlapok, kézikönyvek:

- **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**

Analóg jelfeldolgozás

- Analóg világban élünk, de digitális mikrovezérlővel dolgozunk…



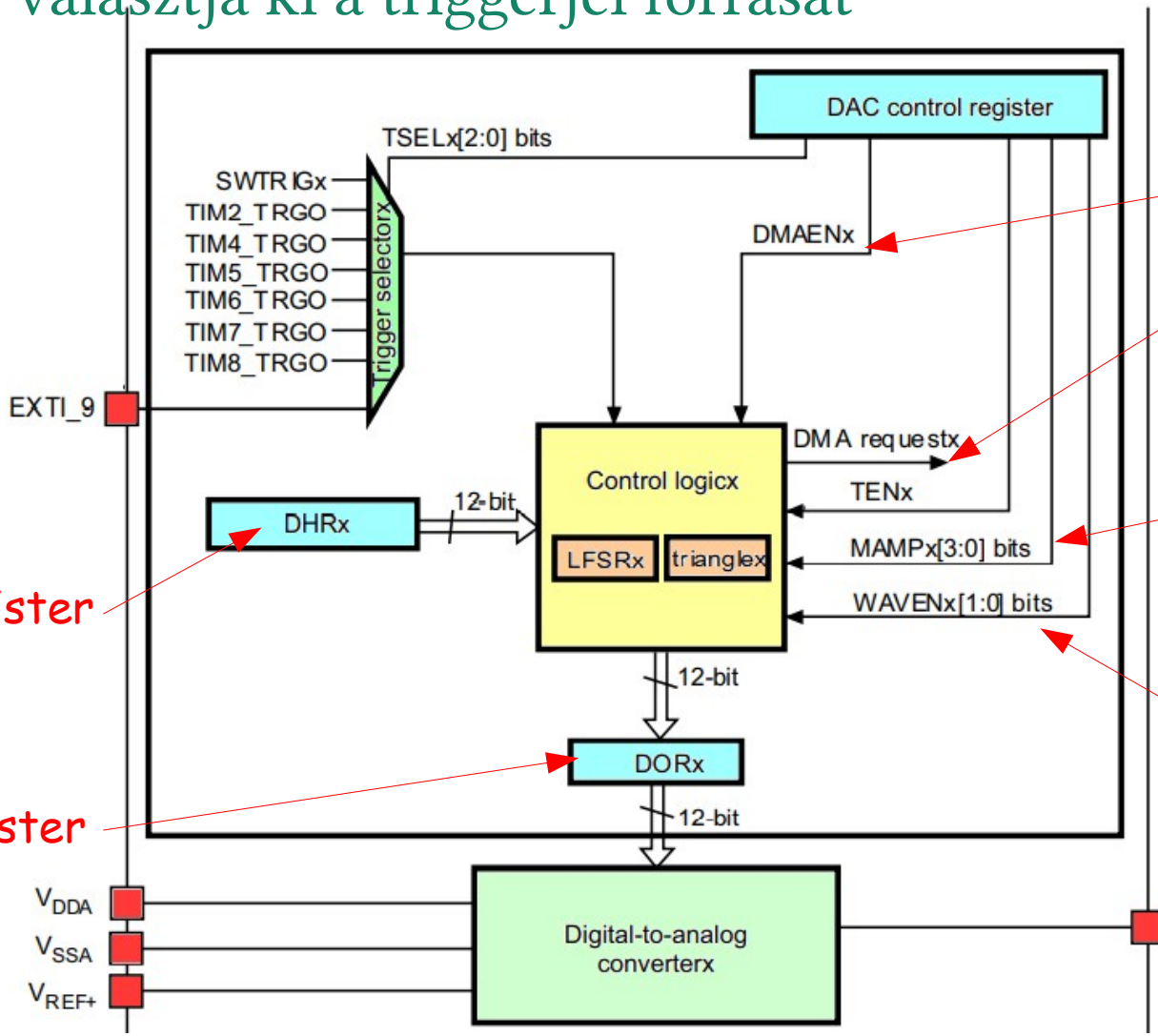


Digitális-analóg átalakító (DAC)

- Nem minden mikrovezérlő rendelkezik digitális-analóg átalakítóval, így pl. az **STM32F103C8** mikrovezérlő sem (azt a korábbi előadásokban egy külső, **SPI DAC**-kal egészítettük ki - lásd itt a 2020. március 26-i és április 2-i előadásokat!)
- Az **STM32F446RE** mikrovezérlő két, beépített 12 bites DAC csatornával rendelkezik, amelyek kimenetei a **PA4** és **PA5** kivezetések (nem áthelyezhető). Ezek használatakor a kivezetéseket analóg módba kell konfigurálni
- A két **DAC** csatorna működhet függetlenül, vagy szinkronizáltan
- DMA-képes mindkét csatorna
- Külső és belső triggerjel is használható a konverzió indításához
- Feszültségreferencia bemenet (1.8 V ≤ VREF+ ≤ VDDA)
- Beépített zaj- illetve háromszögjel generátorok

A digitális-analóg átalakító felépítése

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



Engedélyező bit

DMA átviteli kérelem

MAMPx – mask vagy amplitudó beállítás

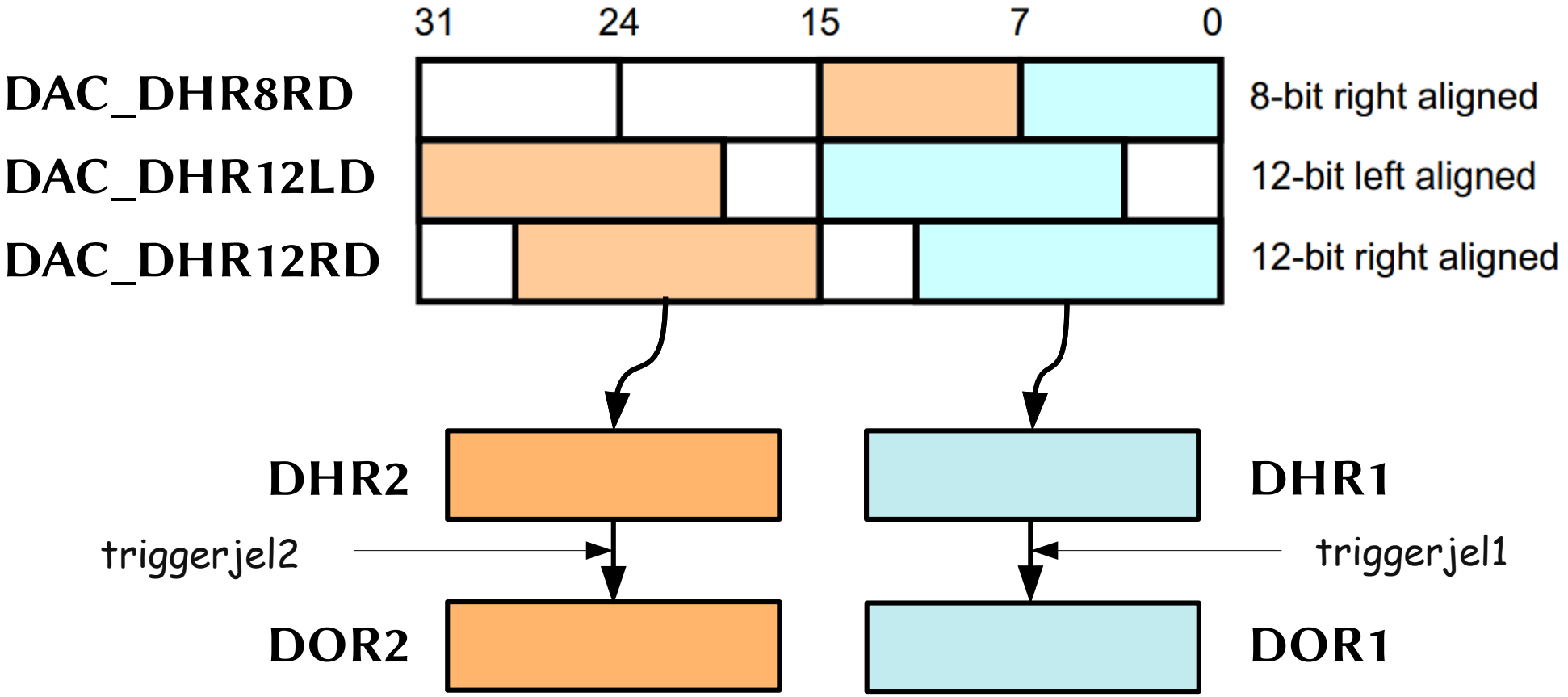
WAVENx – jelgenerátor mód választás

Data Holding Register (közvetlenül nem címezhető)

Data Output Register

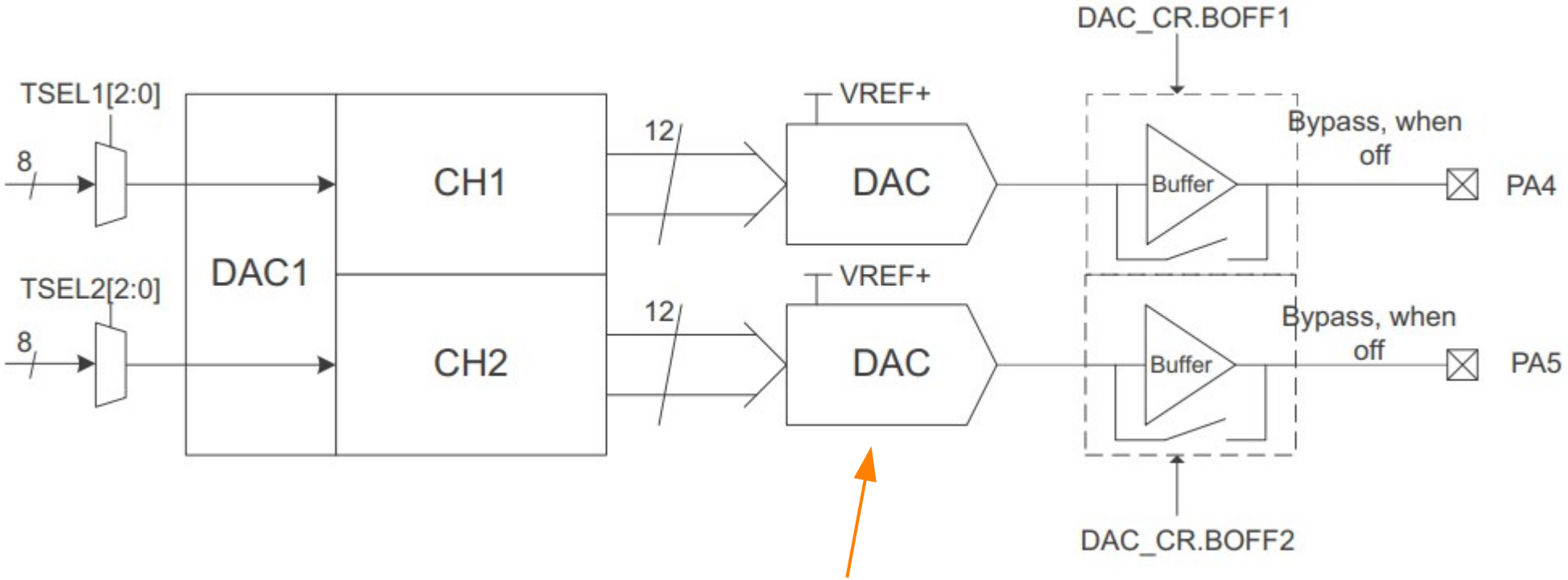
DAC adatformátum

- 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



Kimeneti buffer

- A **DAC** kimeneteihez egy-egy buffer erősítő is tartozik, amelyek engedélyezhetők, vagy áthidalhatók, szerepük a kimeneti ellenállás csökkentése – a nagyobb áramfogyasztás árán



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

DAC_CR: a DAC vezérlő regisztere

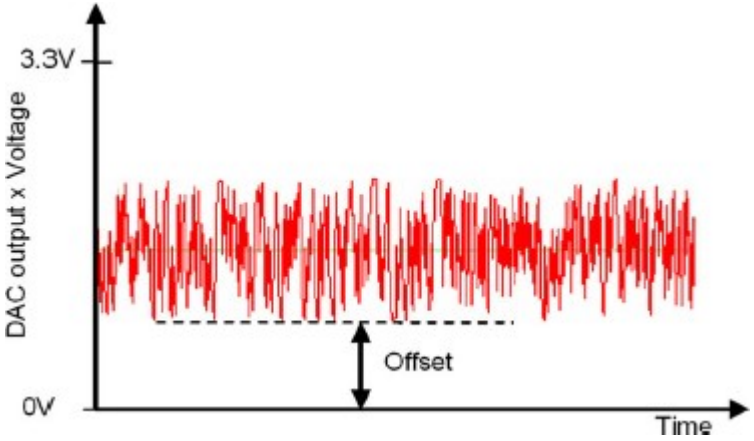
- **DAC_CR** a két DAC csatorna közös vezérlő regisztere

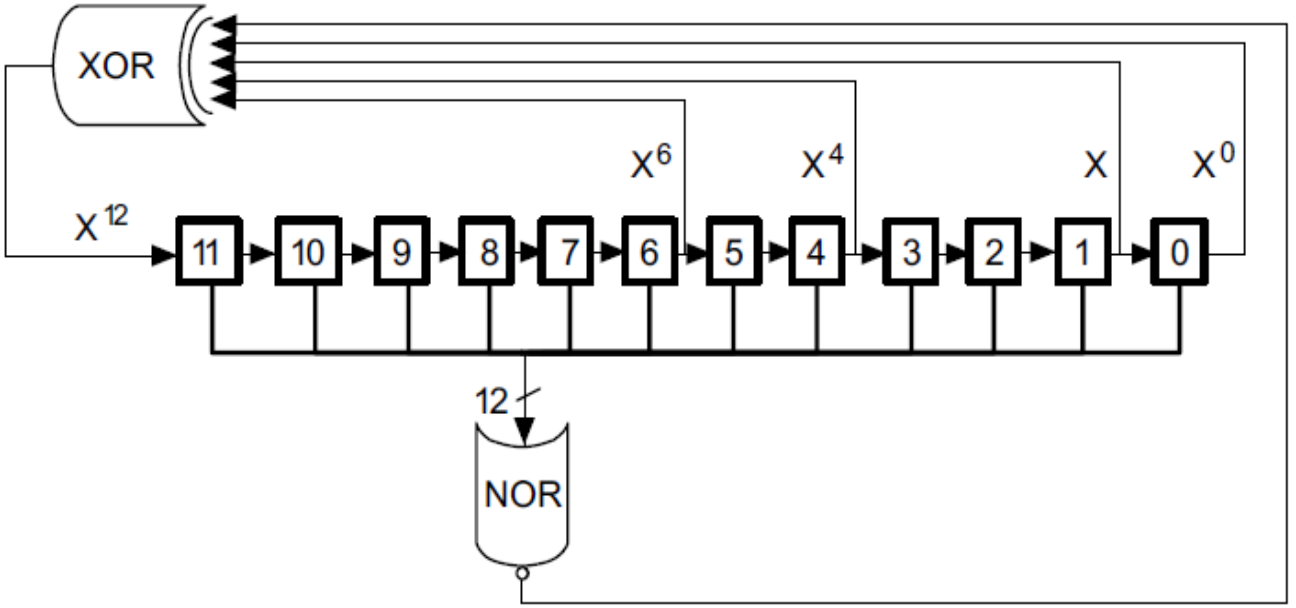
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	DMAUDRIE2	DMAEN2	MAMP2[3:0]				WAVE2[1:0]		TSEL2[2:0]			TEN2	BOFF2	EN2
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	DMAUDRIE1	DMAEN1	MAMP1[3:0]				WAVE1[1:0]		TSEL1[2:0]			TEN1	BOFF1	EN1
		rw	rw	rw	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

Zajgenerátor mód

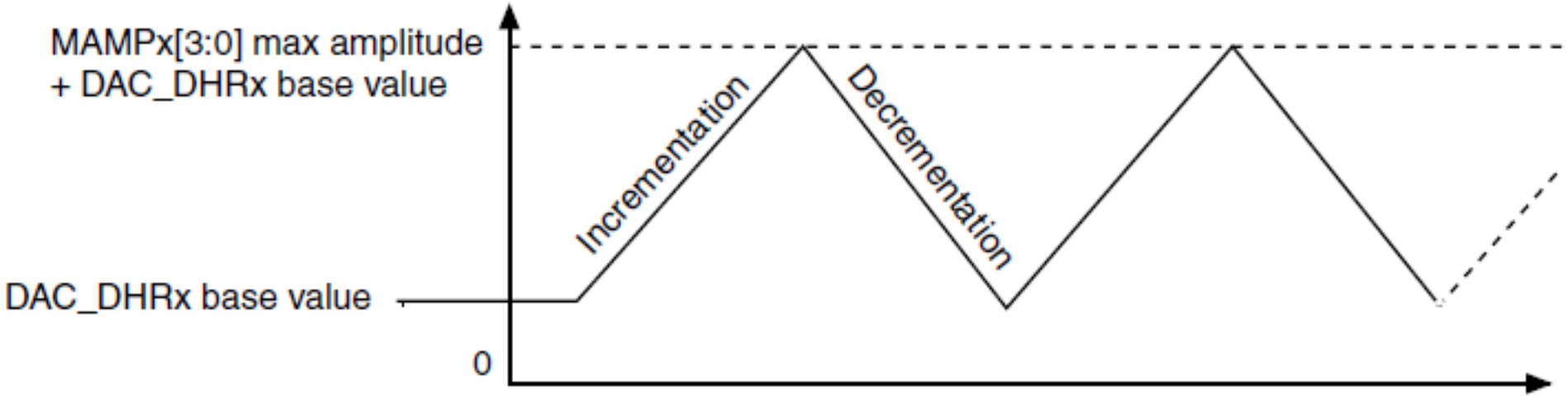
- A **WAVE*x***[1:0] = 01 beállítás bekapcsolja a zajjel generátort, amely egy álvéletlen számot ad hozzá a **DHR*x*** regiszter tartalmához (az *offset-hez*), s ez az összeg kerül a kimeneti adatregiszterbe
- Az álvéletlen számot egy lineárisan visszacsatolt shift regiszter (LFSR) állítja elő
- Az amplitúdót (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!





Háromszögjel generálás

- A **WAVE*x***[1:0] = 1*x* beállítás bekapcsolja a háromszögjel generátort, amely egy fel-le számláló regiszter tartalmát adja hozzá a **DHR*x*** regiszter tartalmához, s ez jut a kimeneti adatregiszterbe
- A számlálás felső határát a vezérlő regiszter **MAMP*x***[3:0] bitcsoportja határozza meg: 0000⟶**1**, 0001⟶**3**, 0010⟶**7**, 0011⟶**15**, 0100⟶**31**, 0101⟶**63**, 0110⟶**127**, 0111⟶**255**, 1000⟶**511**, 1001⟶**1022**, 1010⟶**2047**, ≥ 1011⟶**4095**



DMA átvitel

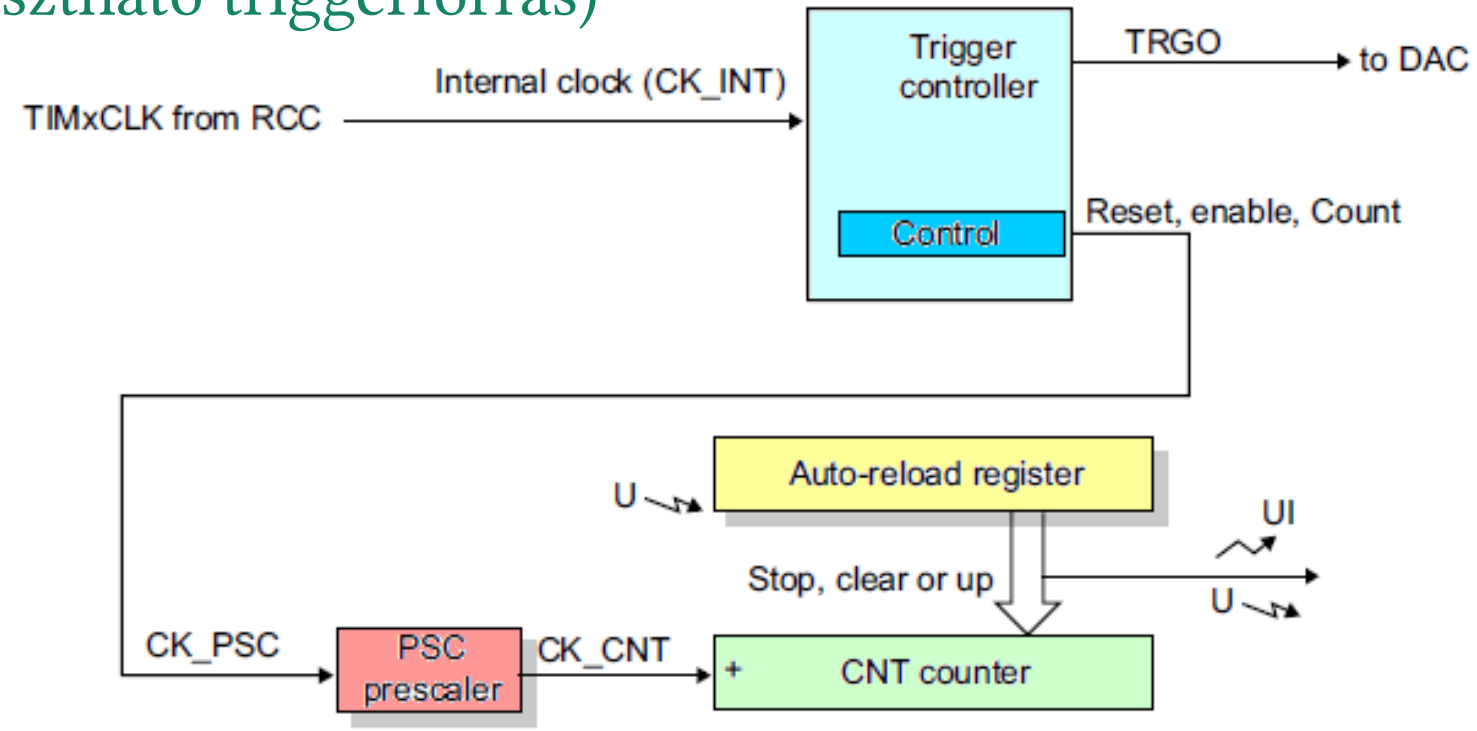
- A **DAC** csatornák adatfeltöltése a **DMA** Stream 5 és Stream 6 adatfolyammal történhet
- A DMA adatátviteli kérelem a 7. csatornát használja

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

Egyszerű időzítők (basic timers)

- **Timer6** és **Timer7** 16 bites, automatikusan újratöltő számlálóból állnak, amelyet egy programozható előosztó hajt meg
- Az előosztó bemenőjele a **PBCLK1** buszfrekvencia kétszerese (esetünkben 90 MHz)
- A **TRGO** trigger kimenetük a **DAC** egységhez van kötve (választható triggerforrás)



HAL_DAC és HAL_DACEx modulok

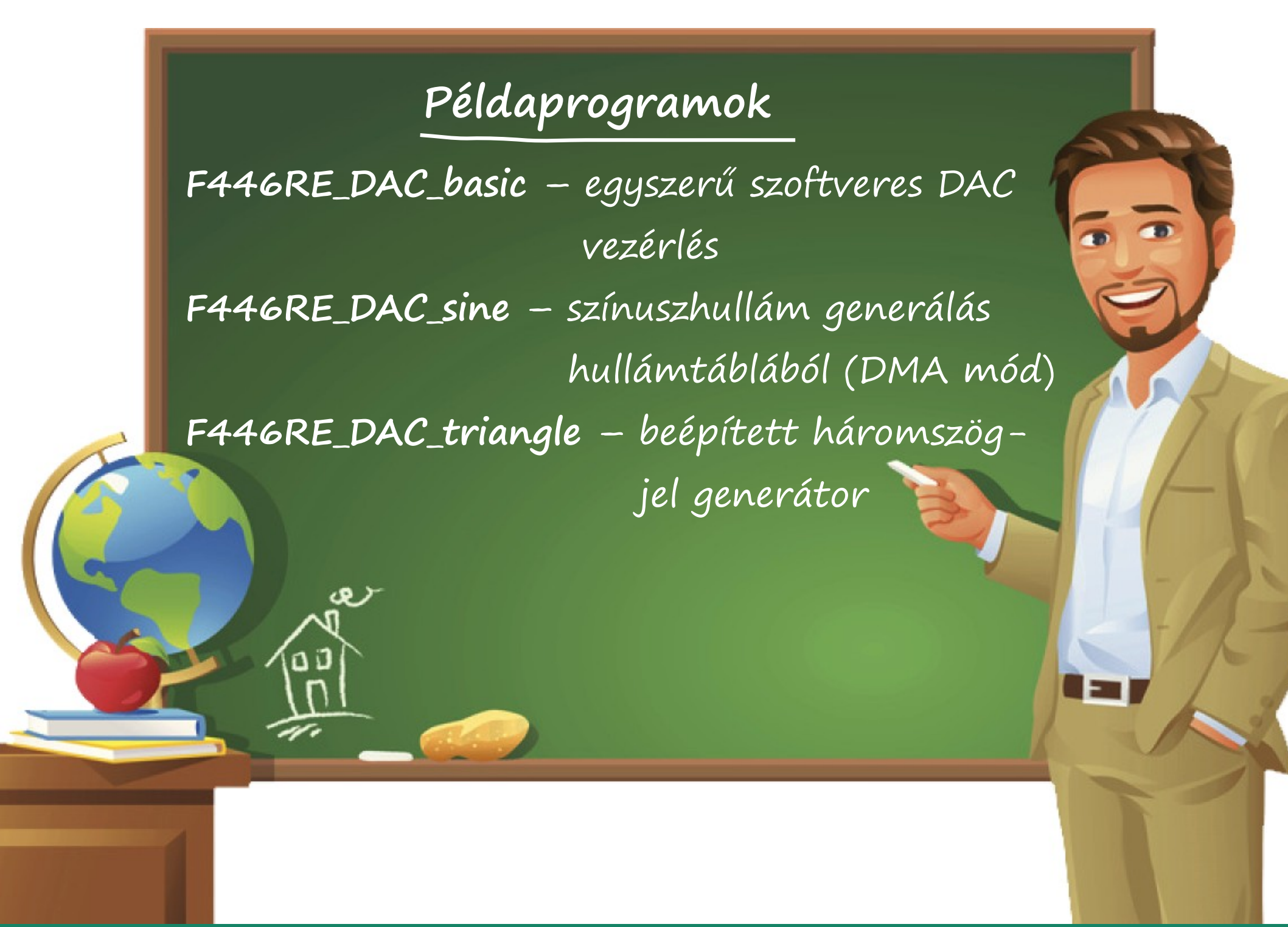
- A **HAL_DAC** modul az alapvető API-t biztosítja:
  - **HAL_DAC_Init** – a DAC inicializálása
  - **HAL_DAC_MspInit** – a DAC kivezetés(ek) inicializálása (PA4/PA5)
  - **HAL_DAC_ConfigChannel** – DAC csatorna konfigurálása
  - **HAL_DAC_Start** – A DAC modul indítása
  - **HAL_DAC_Start_DMA** – a 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

Példaprogramok

F446RE_DAC_basic – egyszerű szoftveres DAC vezérlés

F446RE_DAC_sine – színuszhullám generálás hullámtáblából (DMA mód)

F446RE_DAC_triangle – beépített háromszög-jel generátor

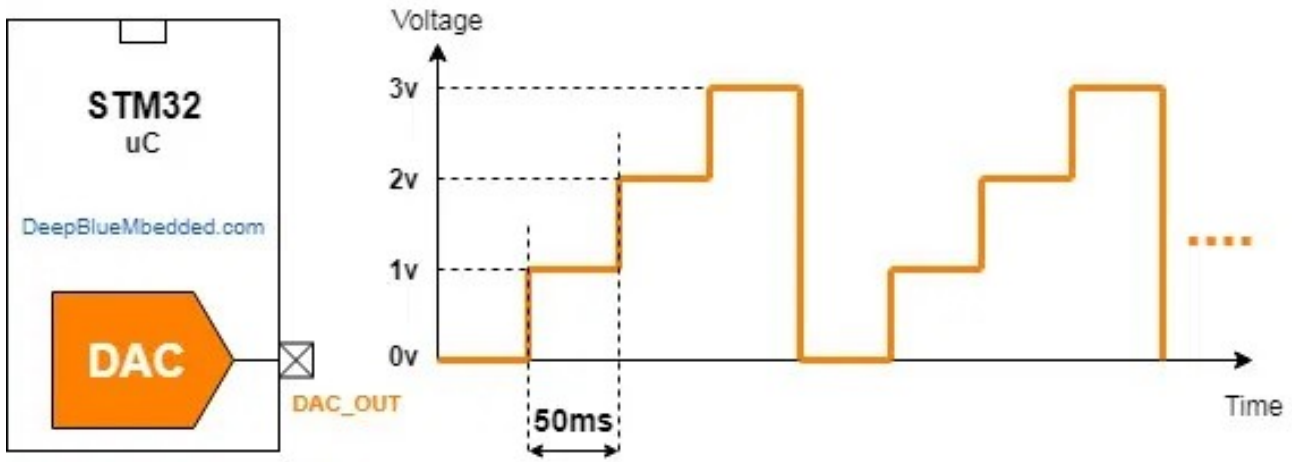


F446RE_DAC_basic

- Ebben a programban a **DAC** 1. csatornáját (**PA4** kivezetés) szoftveresen vezéreljük a **HAL_DAC_SetValue** függvénnyel
- A kiírásokat 50 ms-onként végezzük, és sorban 0 V, 1 V, 2 V, 3 V kimenő feszültséget állítunk be, s ezt a sorozatot ismételgetjük
- Az adott *U* feszültséghez kiküldendő adatok, az képlet alapján: 0, 1241, 2482, illetve 3723

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

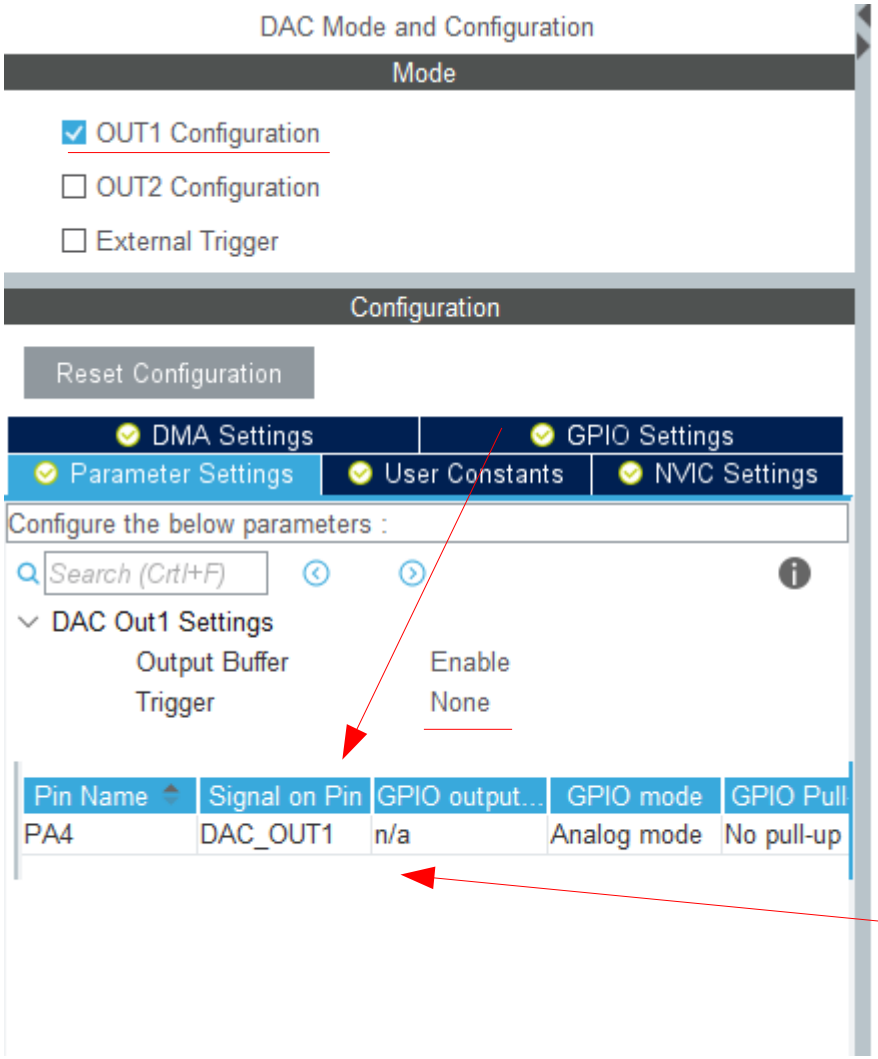
Megjegyzés: esetünkben VREF a tápfeszültség, azaz 3,3 V

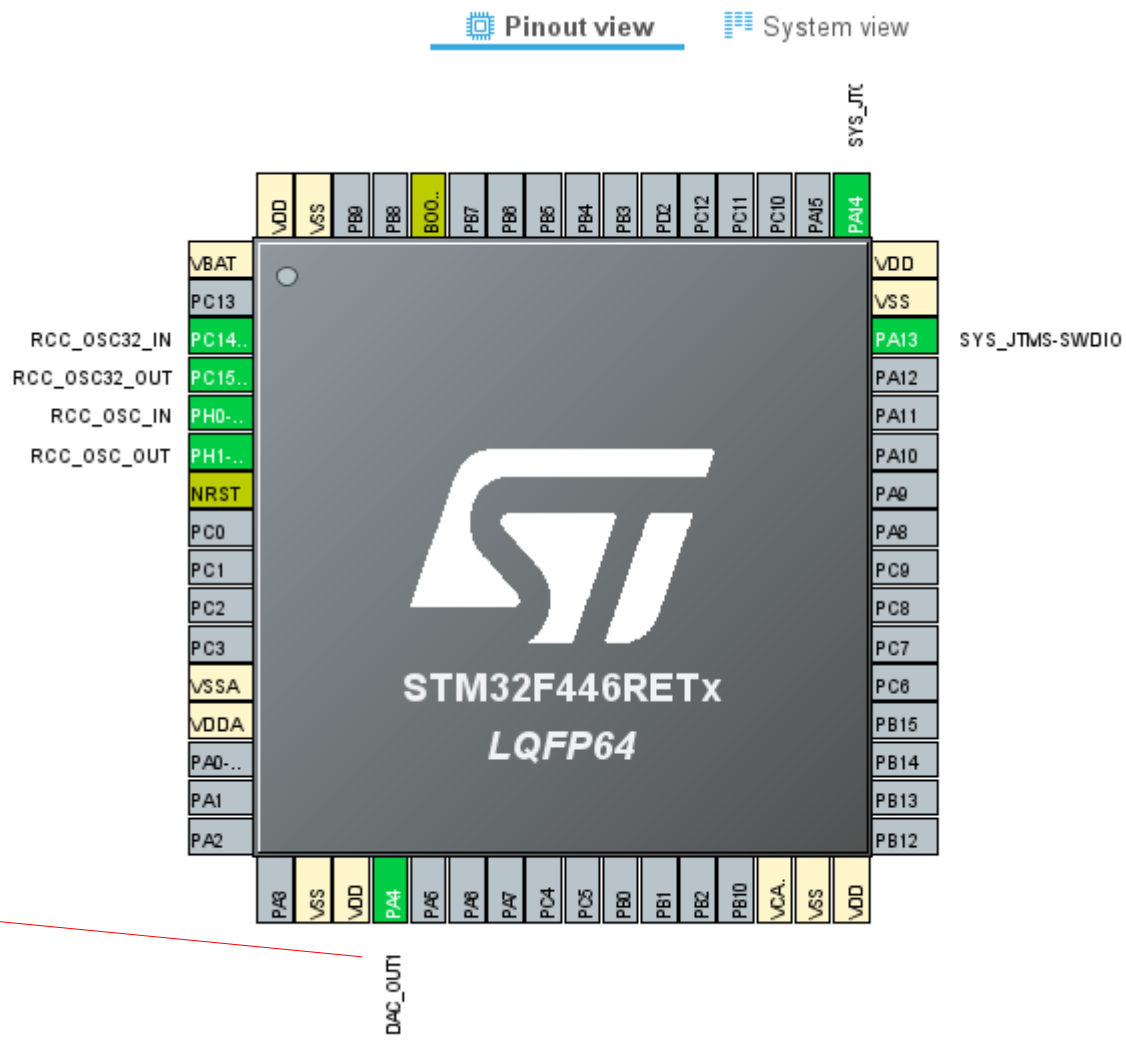


- Programötlet és az ábra forrása: Khaled Magdy : STM32 DAC Tutorial

A DAC modul konfigurálása

- Az 1. csatornát konfiguráljuk (**PA4** kivezetés)
- A kimeneti buffert engedélyezzük, triggerforrást nem definiálunk



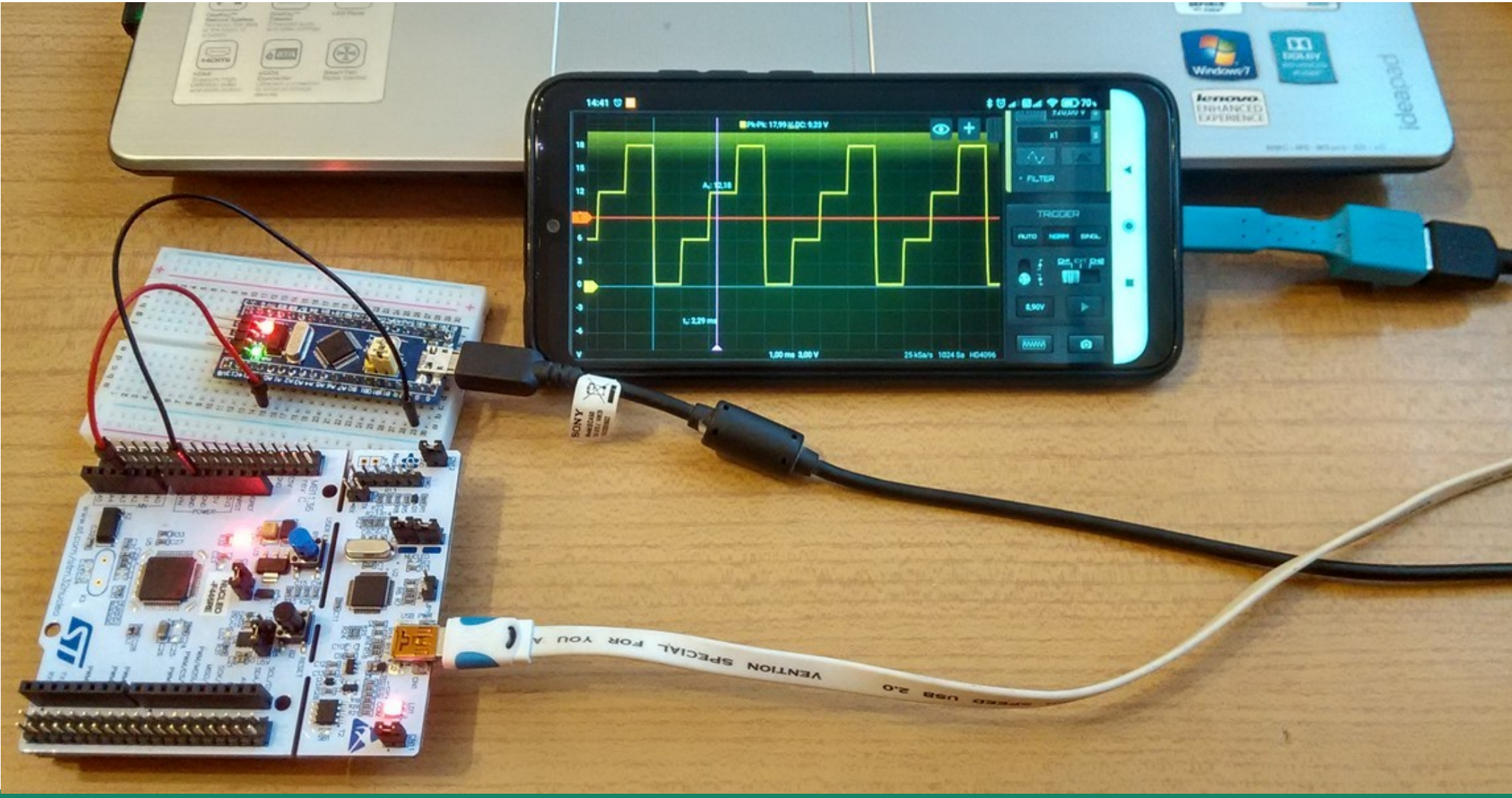


F446RE_DAC_basic: main.c releváns sorai

```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);
  while (1) {
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, DAC_OUT[i++]);
    i = i & 0x03;                               // csak 0, 1, 2 vagy 3 lehet!
    HAL_Delay(50);
  }
}
```

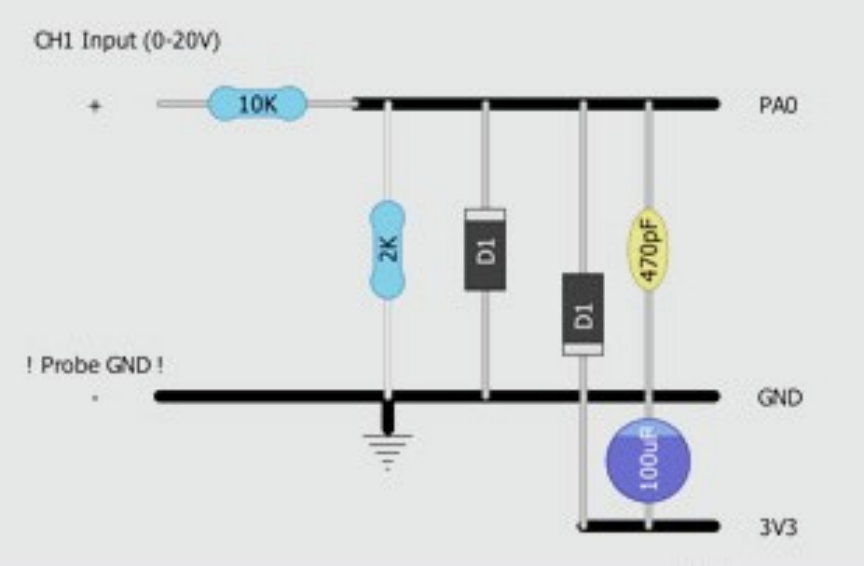
F446RE_DAC_basic: futási eredmény

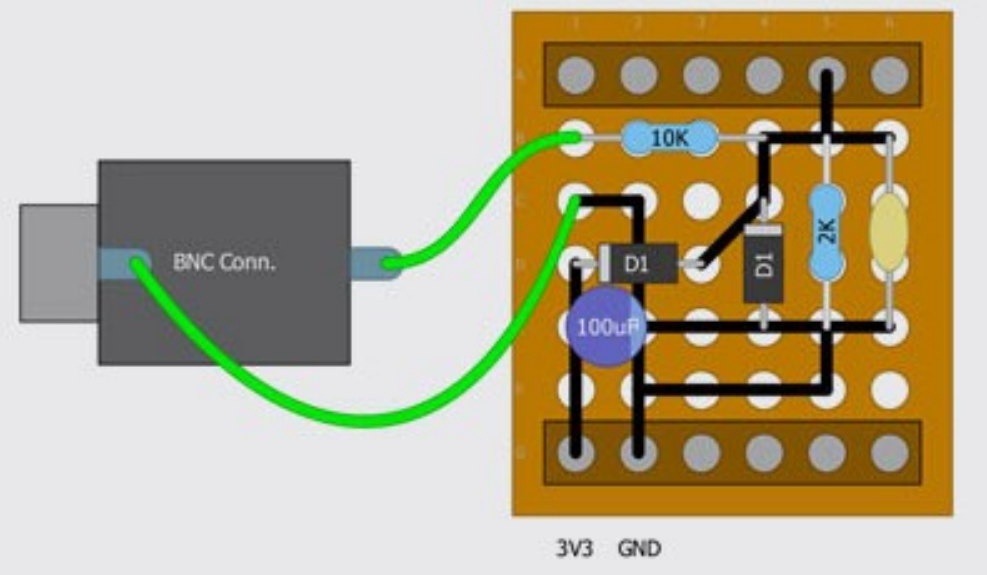
- Az ábrához *1 ms*-ra csökkentettük a kiírások közti késleltetést,
  Megjegyzés: ehhez `HAL_Delay(0);`-t kellett írni!



Oszcilloszkóp házilag

- Az előző oldalon bemutatott jelvizsgálóhoz két dolog kell:
  - Martin Loren Hscope nevű Android alkalmazása (ingyenes változata korlátozással)
  - Egy támogatott digitális (USB) oszcilloszkóp (vásárolt, vagy saját készítésű)
- A HS101 egycsatornás készülék a „Blue pill”
- Építési leírása: hscope.martinloren.com/HS101-oscilloscope.html
- Firmaware letöltés: github.com/martinloren/HScope/tree/master/HS 10X



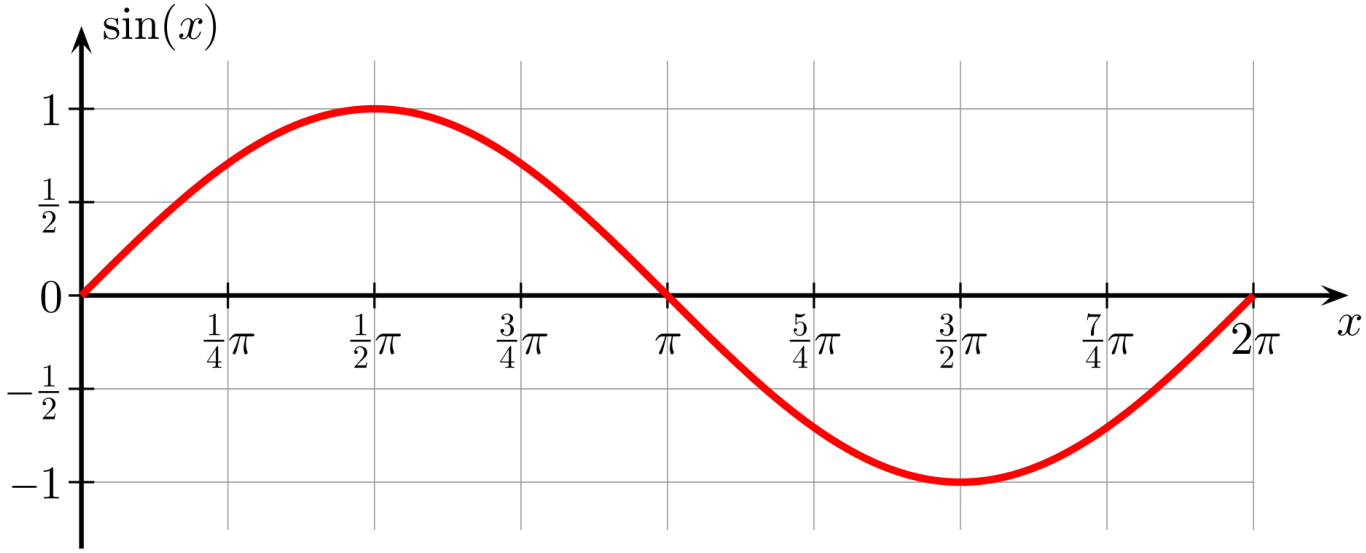


F446RE_DAC_sine

- A második programban a **DAC** 1. csatornáját (**PA4** kivezetés) egy hullámtáblából töltögetjük a **DMA1** modul segítségével (stream 5), az ütemezésről pedig **Timer6** gondoskodik
- Terv szerint a kiírásokat 10 µs-onként végeztük volna…
  a 200 adatot tartalmazó wavetable tömbben egy periódusnyi szinuszhullámot tárolunk (a függvényértékeket 0 – 4095 közé normálva)

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

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

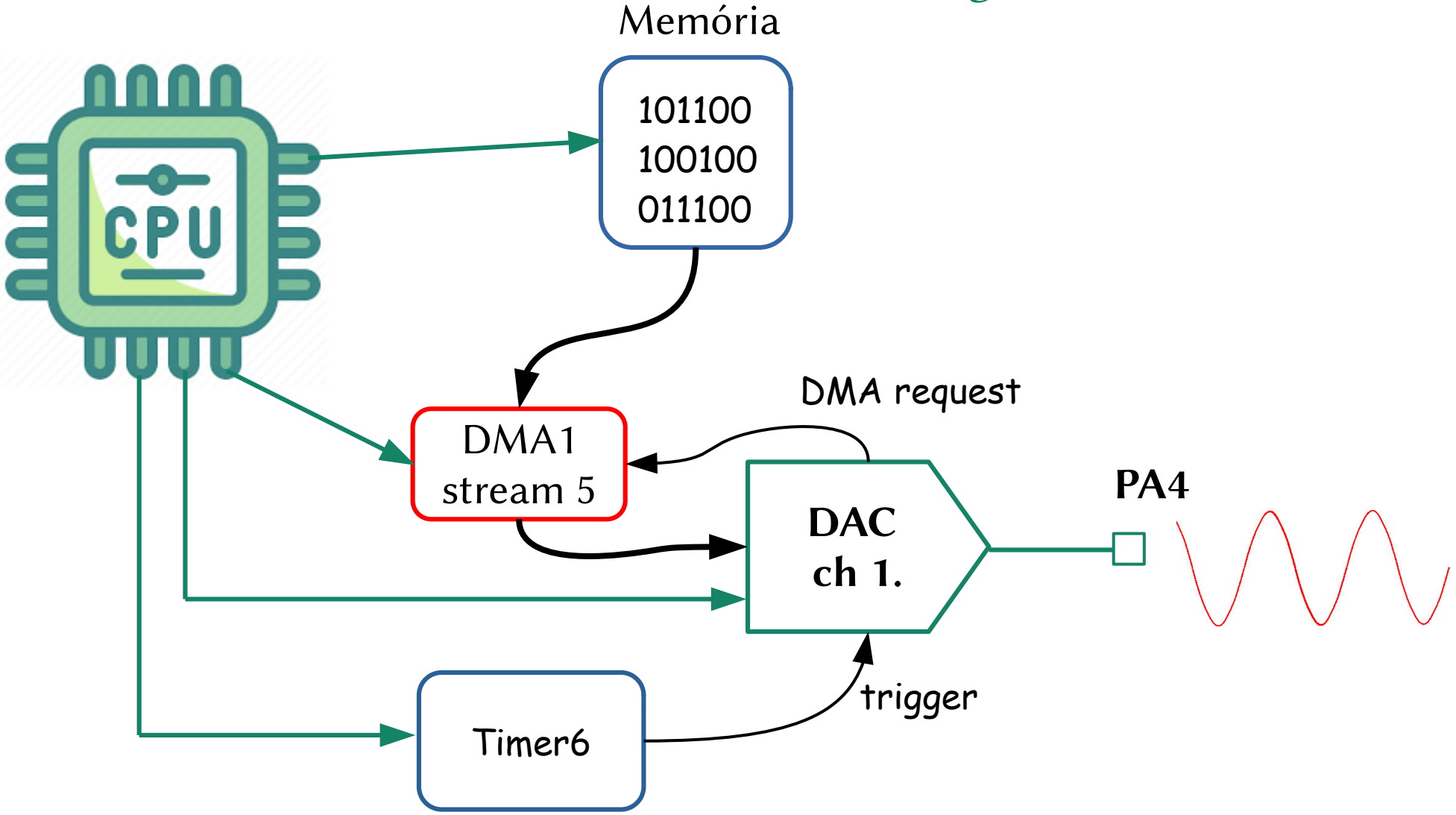


- Programötlet: Carmine Noviello: **Mastering STM32**

  A könyv mintapéldái: **github.com/cnoviello/mastering-stm32**

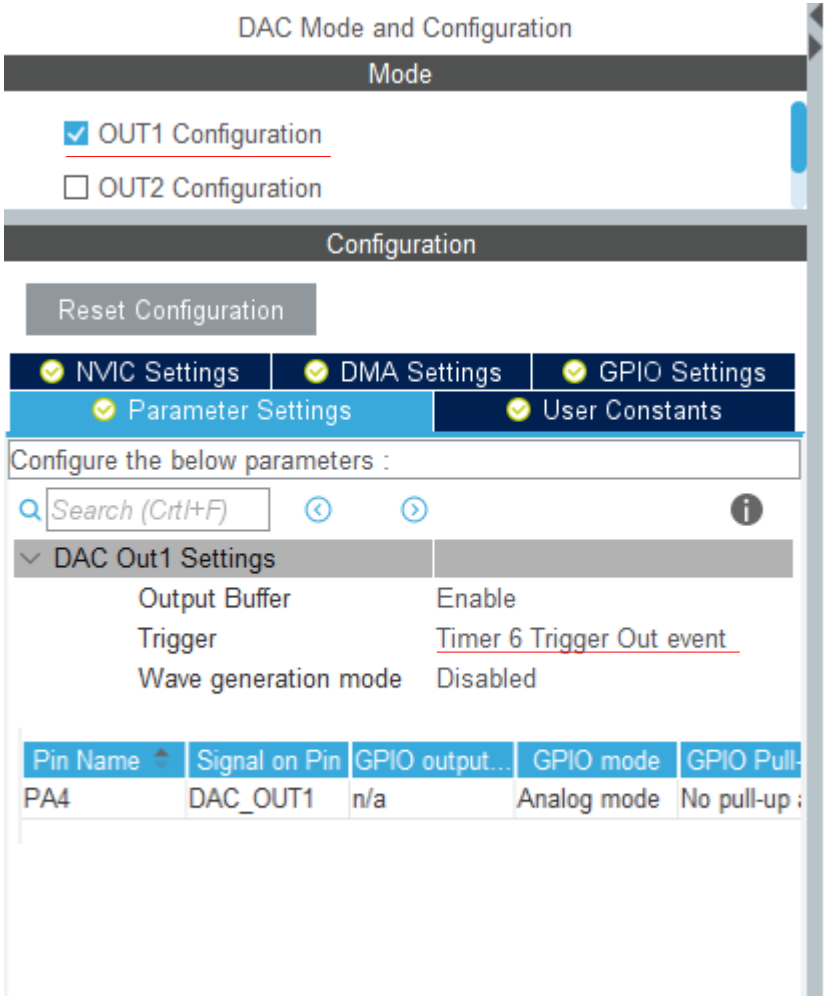
Az adatáramlás sematikus vázlata

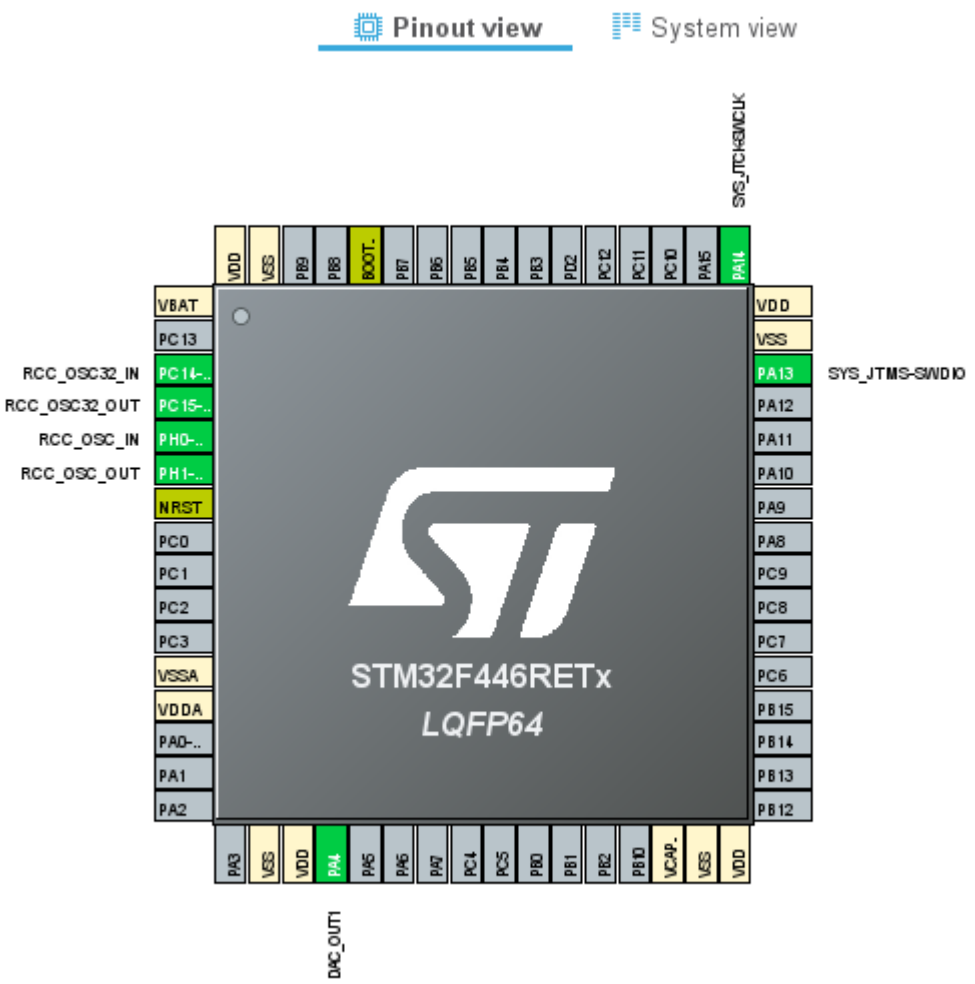
- A konverziót **Timer6** triggereli, a DAC pedig adatot kér a DMA-tól
- A DMA a memóriából 16 bites adatokat mozgat a DAC felé



A DAC konfigurálása

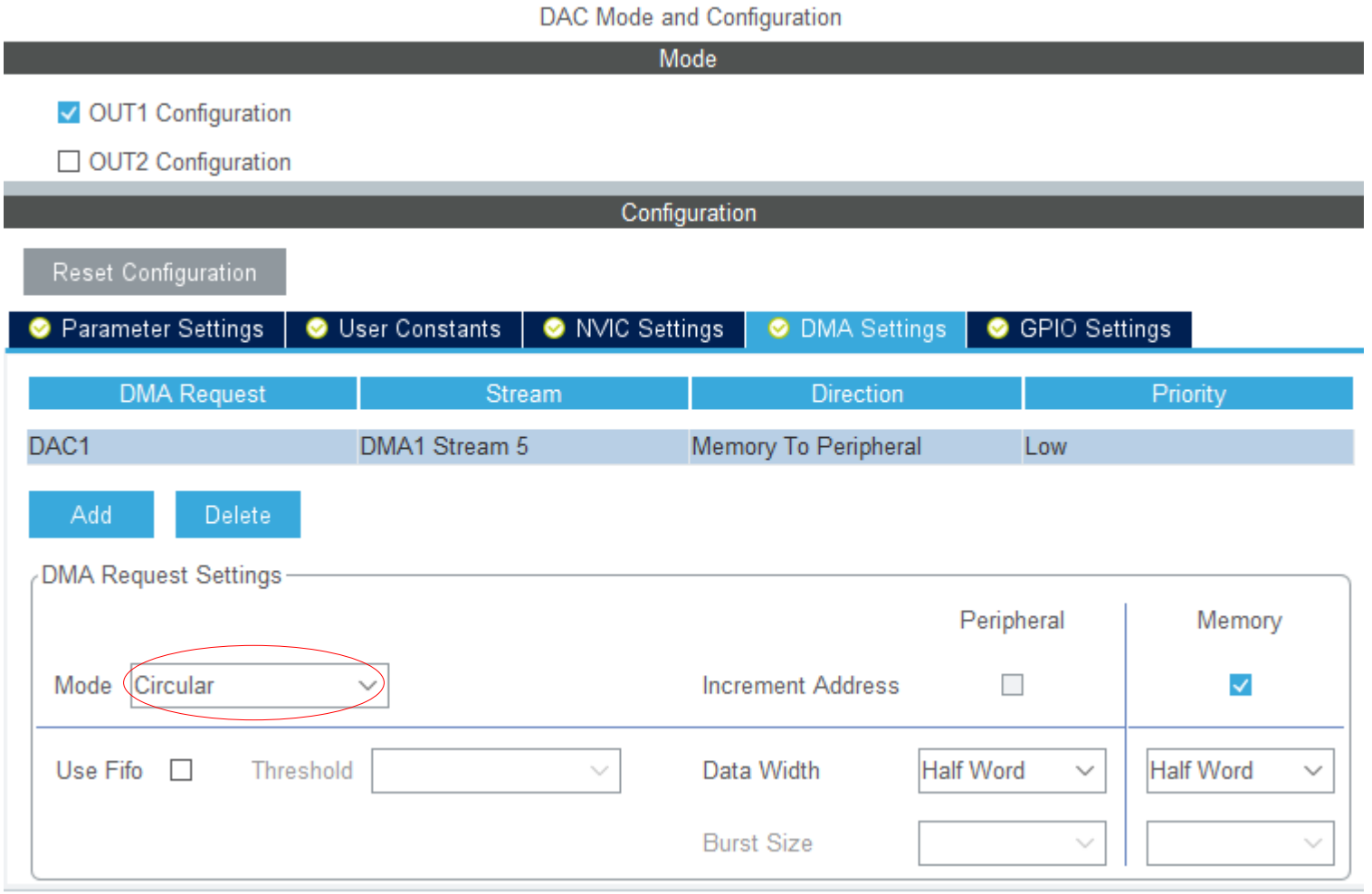
- Változás az előző programhoz képest, hogy trigger jelforrást is kell választani, majd a *DMA Settings* fülre kattintva aktiválni kell a *DMA1 Stream 5* csatornát





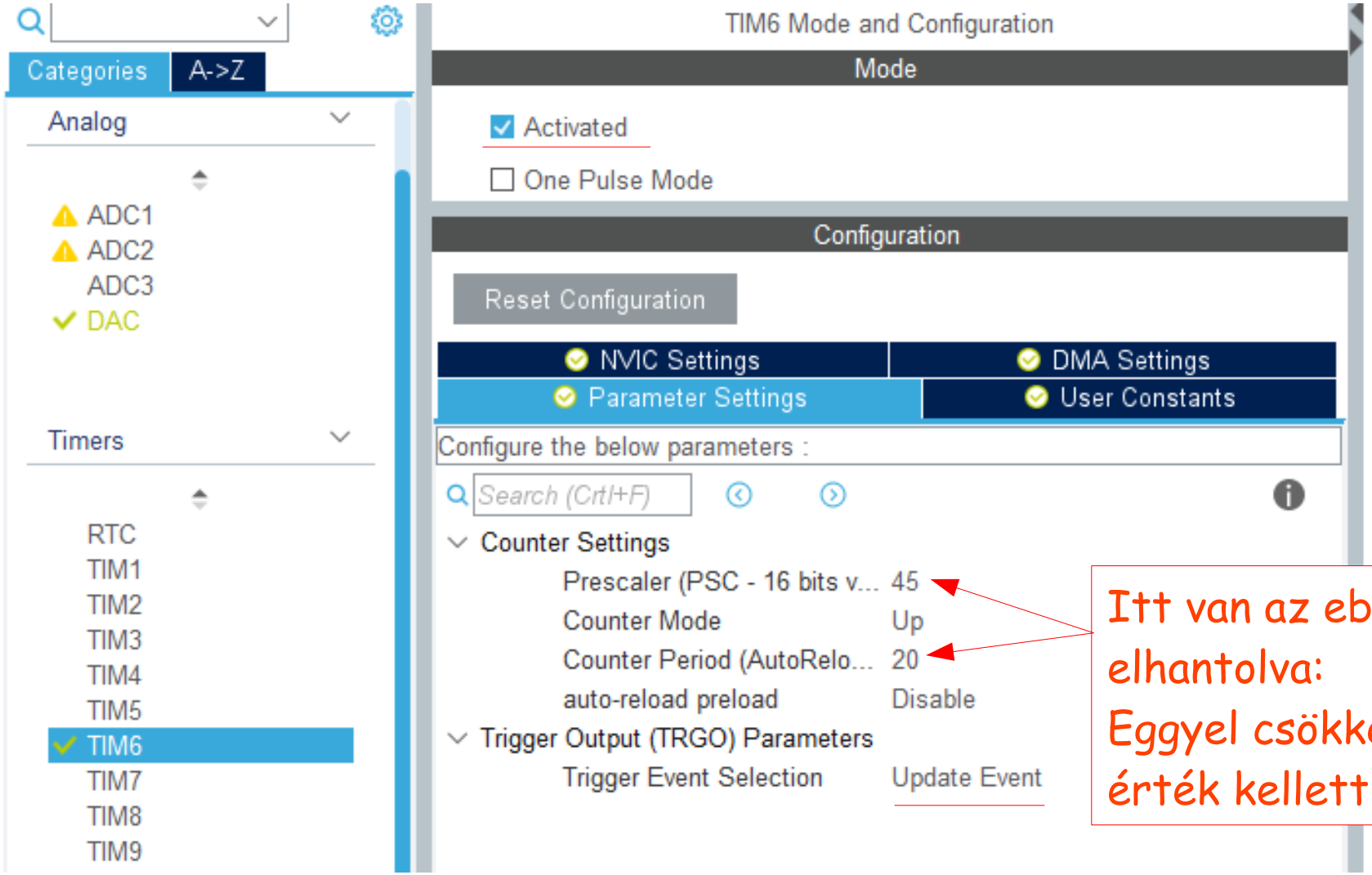
DAC – DMA kapcsolat konfigurálása

- Az **ADD** gombbal aktiváljuk a *DAC1 stream 5*-öt (*circular* mód!)
- Az átvitel **Memória → periféria** irányú, 16-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



F446RE_DAC_sine: main.c releváns sorai

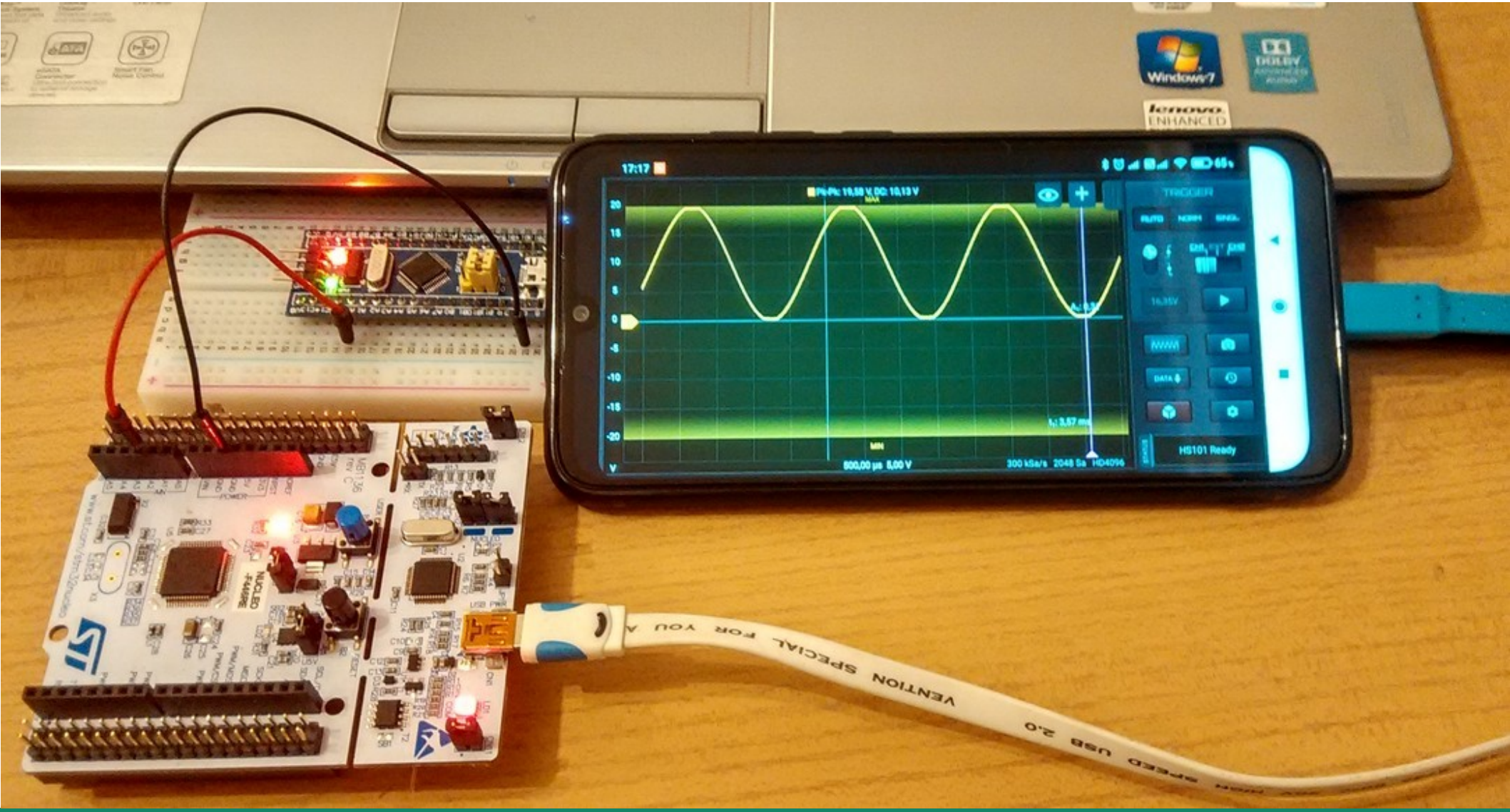
```
#include "main.h"
#include <string.h>
#include <math.h>
DAC_HandleTypeDef hdac;
DMA_HandleTypeDef hdma_dac1;
TIM_HandleTypeDef htim6;
#define PI    3.14159
#define NDATA 200
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_DAC_Init(void);
static void MX_TIM6_Init(void);

int main(void) {
  uint16_t wavetable[NDATA], value;
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  ...
```

F446RE_DAC_sine: main.c releváns sorai

```
  MX_GPIO_Init();
  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)+1)*2048);
    wavetable[i] = value < 4096 ? value : 4095;  // levágás 4095-nél
  }
  HAL_DAC_Init(&hdac);
  HAL_TIM_Base_Start(&htim6);
  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)wavetable,
                    NDATA, DAC_ALIGN_12B_R);
  while (1) {
  }
}
```

F446RE_DAC_sine: futási eredmény

- A kimenőjelet oszcilloszkóp segítségével ellenőrizhetjük



Miért nem stimmel a frekvencia?

- **Timer6** paramétereit rosszul adtuk meg, ezért a frekvencia nem 500 Hz lett, hanem:

$$Freq = \frac{90\,MHz}{46 \cdot 21 \cdot 200} = 465.8\,Hz$$

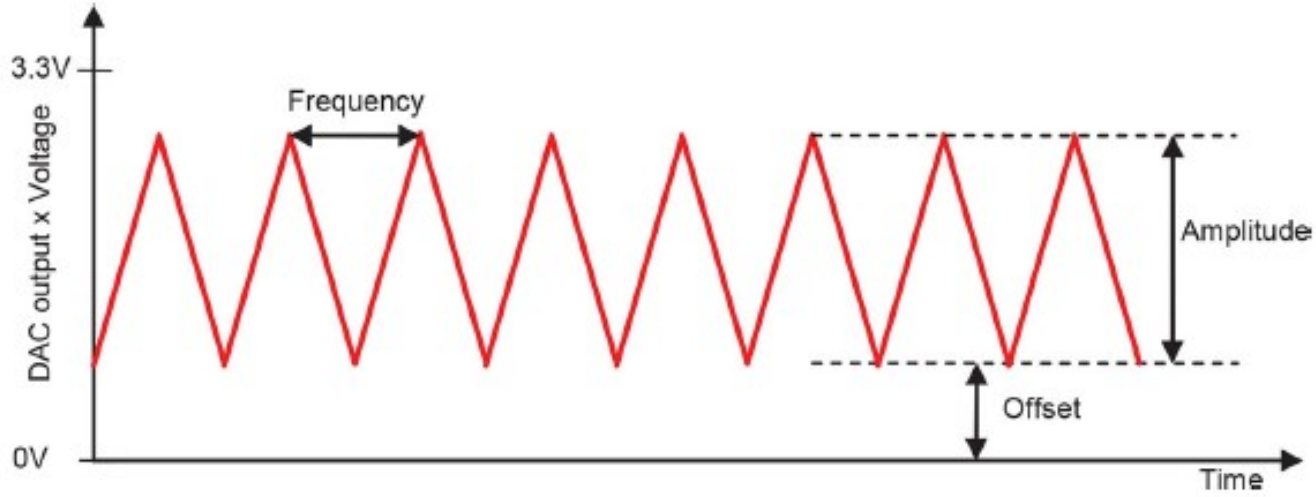
- A fenti értéket elég jól megközelíti a szemre beállított markerpozíciókból kapott érték:
  Freq = 464.6 Hz

F446RE_DAC_triangle

- A harmadik programban a **DAC** beépített háromszögjel generátorát használjuk, eltolt nullapontú háromszögjel (az adatregiszter 2000 – 4047 közötti értékeket vesz fel) keltésére, a **DAC** 1. csatornáján (**PA4** kivezetés)
- Az ütemezésről most is **Timer6** gondoskodik
- A kimenő jelet **ADC1** segítségével mintavételezzük, folyamatos konverzióval, s az adatokat **DMA2**-vel mozgatjuk, s 500 adatonként **UART2** segítségével kiíratjuk

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

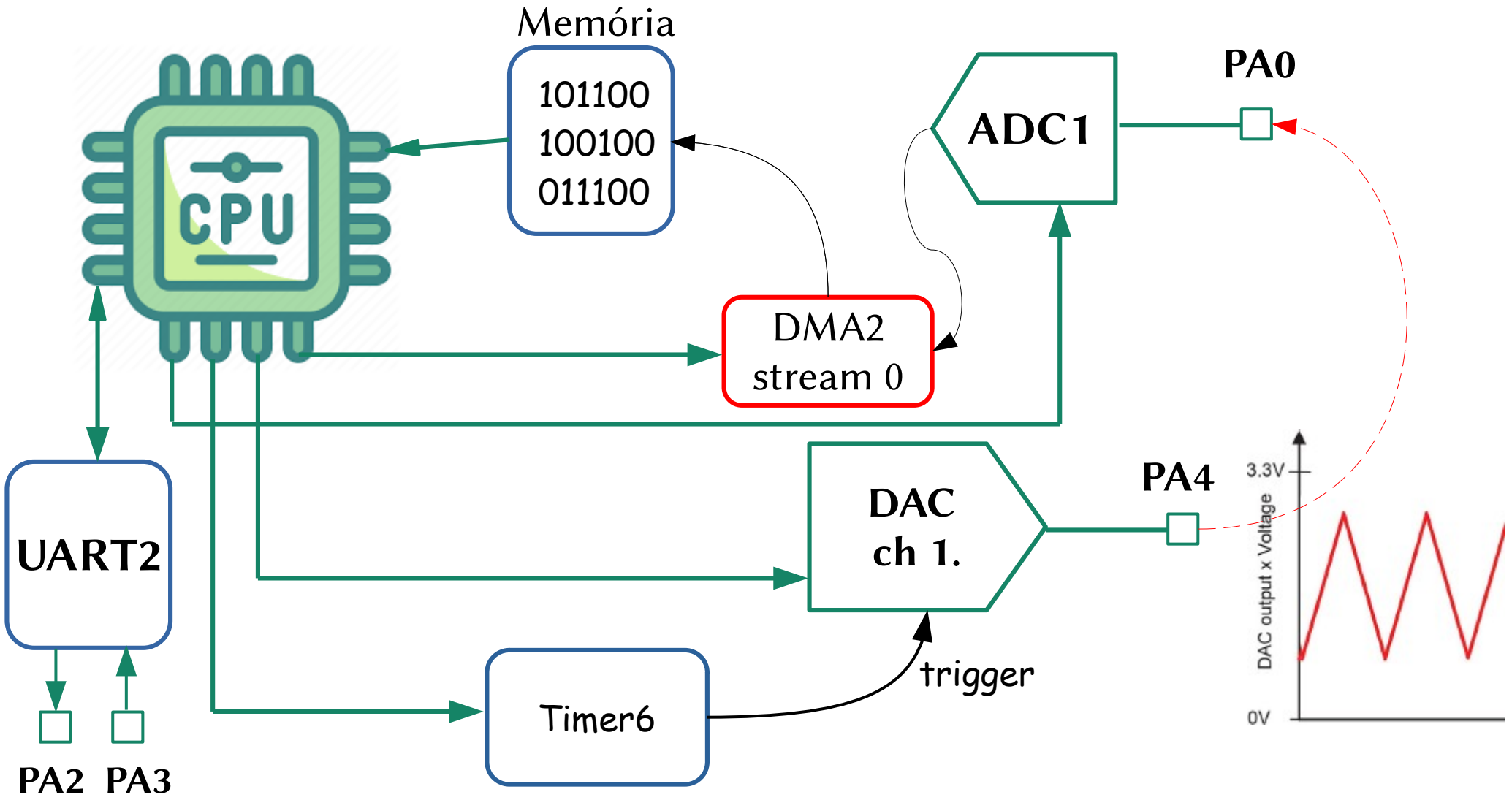
$$Freq = \frac{90\,MHz}{1 \cdot 11 \cdot 4095} = 1998\,Hz$$



Az ábra forrása: Carmine Noviello: **Mastering STM32**

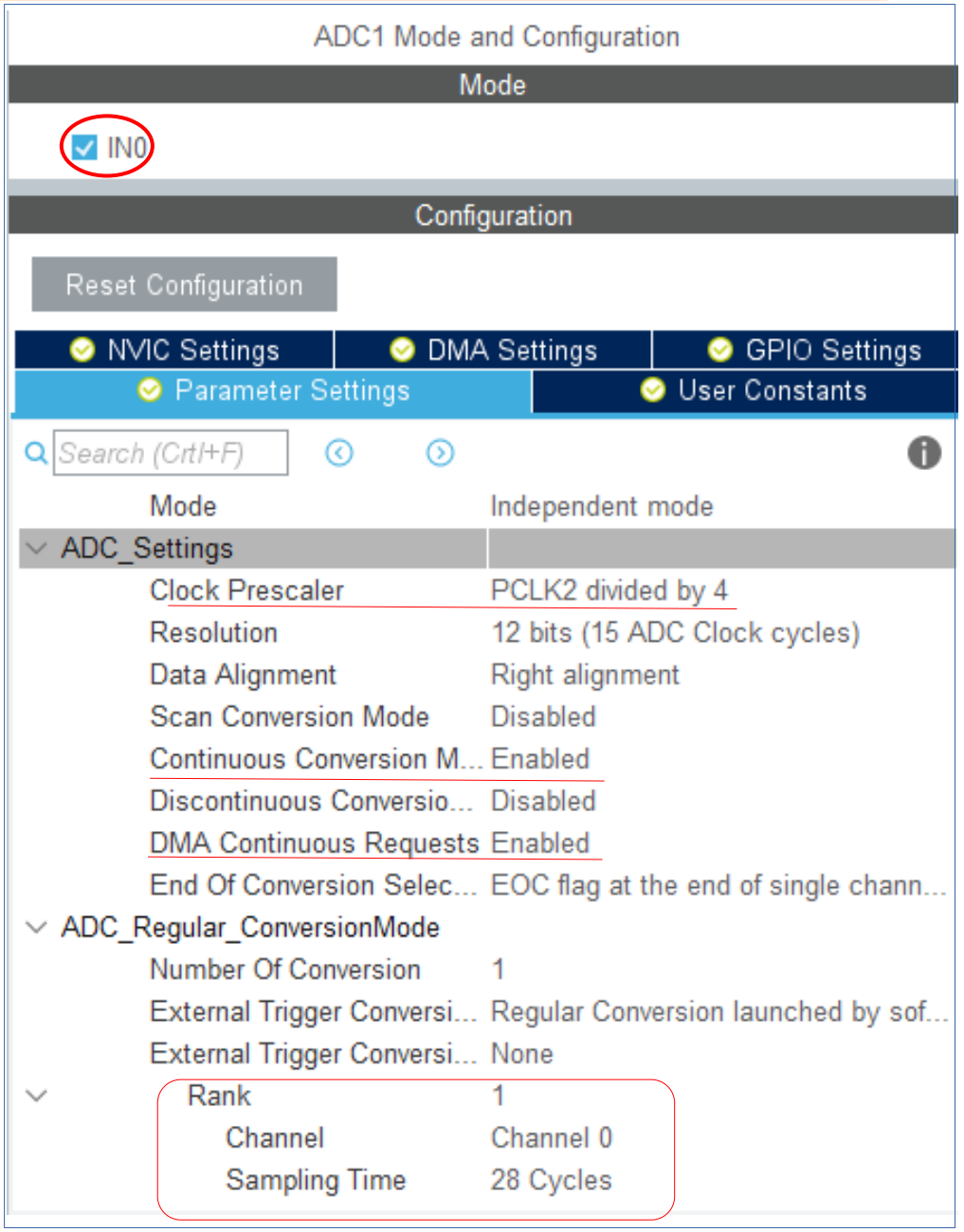
Az adatáramlás sematikus vázlata

- **Timer6** triggereli a **DAC**-ot. az **ADC** pedig szabadonfutó módba konvertál, s a **DMA** tárolja el az adatokat a memóriába

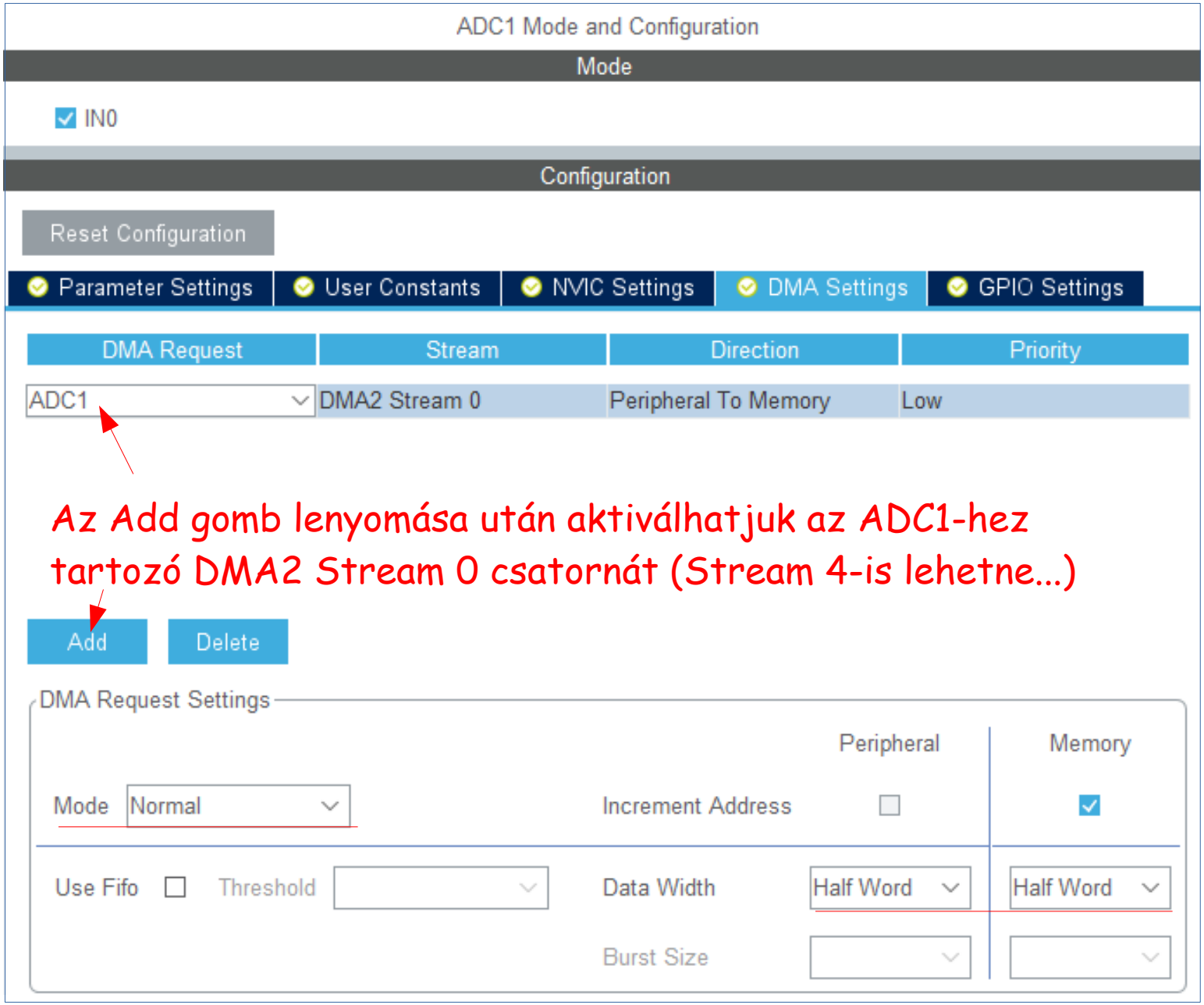


ADC1 konfigurálása

- Az **IN0** csatornát használjuk (**PA0** bemenet)
- Bemenő órajel: PCLK2/4=22.4 MHz
- Mintavételezés + konverzió: 40 ciklus, azaz 562 500 minta/s (T = 1.78 µs)
- A folytonos konverziót ne felejtsük el engedélyezni!

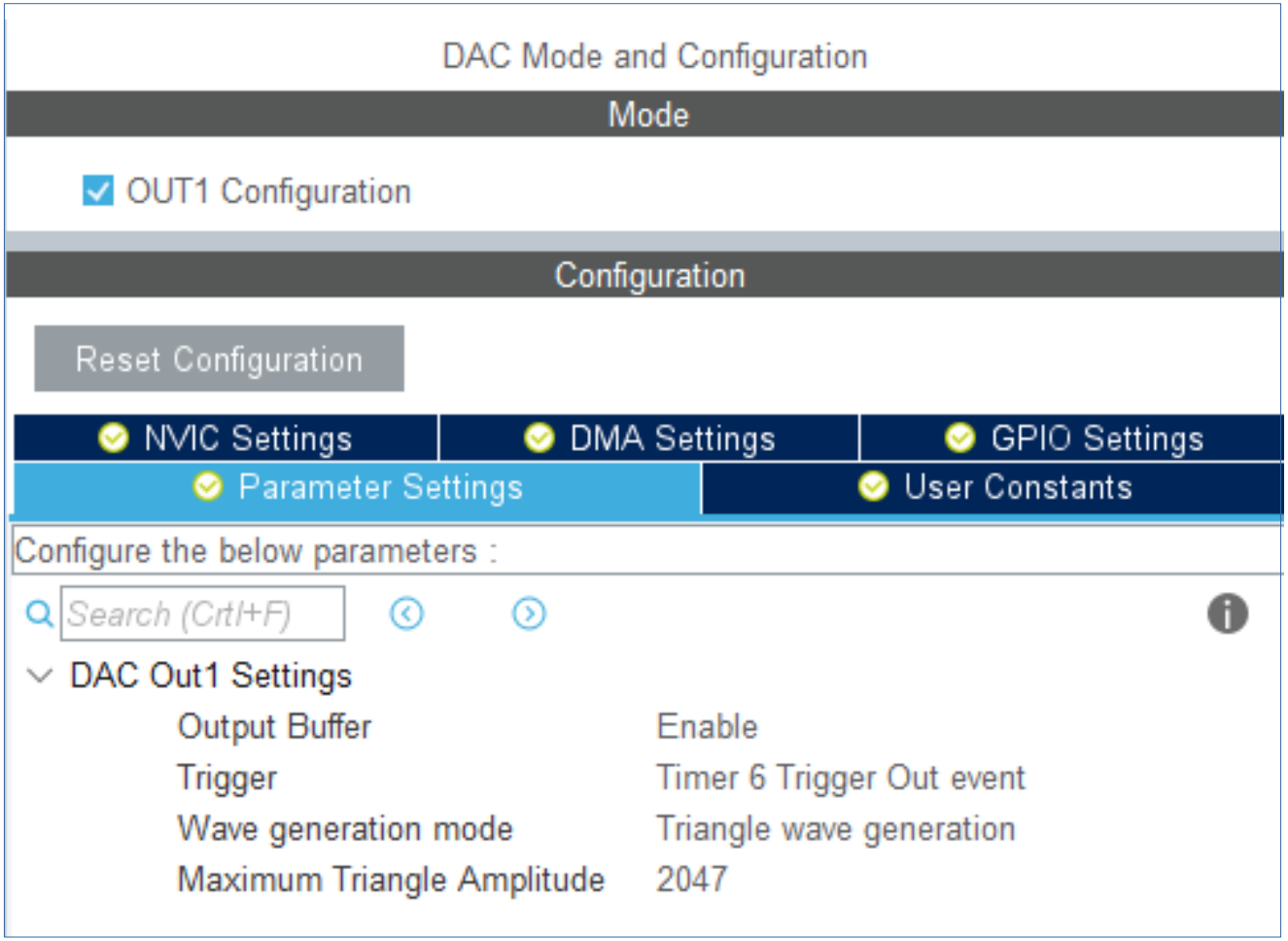


Az ADC1 – DMA átvitel konfigurálása



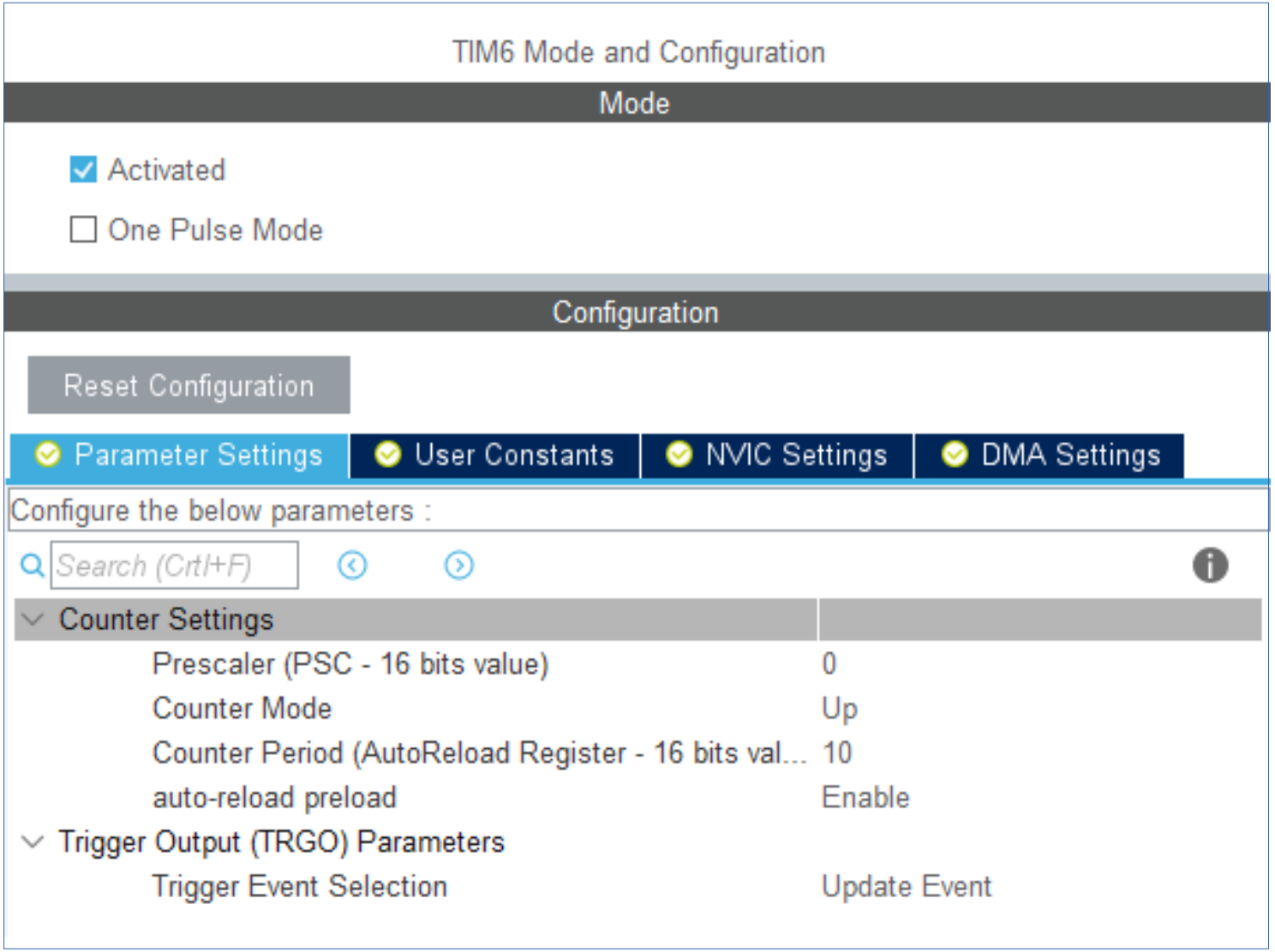
A DAC konfigurálása

- Aktiváljuk a belső háromszögjel generátort, s az amplitúdót 2047-re állítjuk (az *offset* majd 2000 lesz, így beleférünk 4095-be!)
- A triggerjelet **Timer6** fogja szolgáltatni



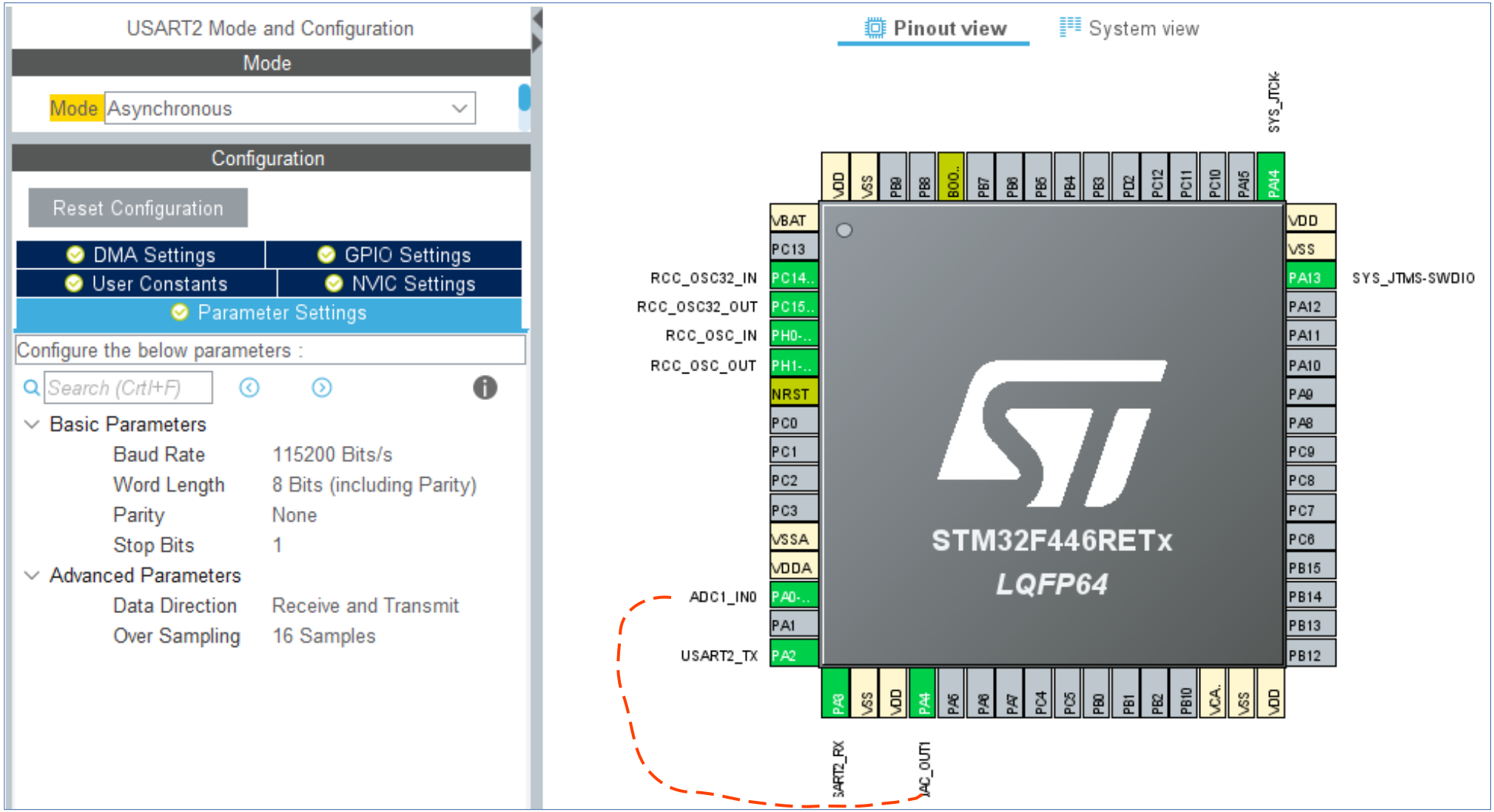
Timer6 konfigurálása

- Nincs előosztás, a periódus pedig 11, így a triggerjelek frekvenciája F_{TRIG} = 90 MHz / 11 = 8.18 MHz a jelfrekvencia pedig 1998 Hz



USART2 konfigurálása

- Aszinkron módú átvitelt használunk, 115 200 bit/s bitrátával a **PA2/PA3** kivezetések már össze vannak kötve az ST-Linkkel



F446RE_DAC_triangle

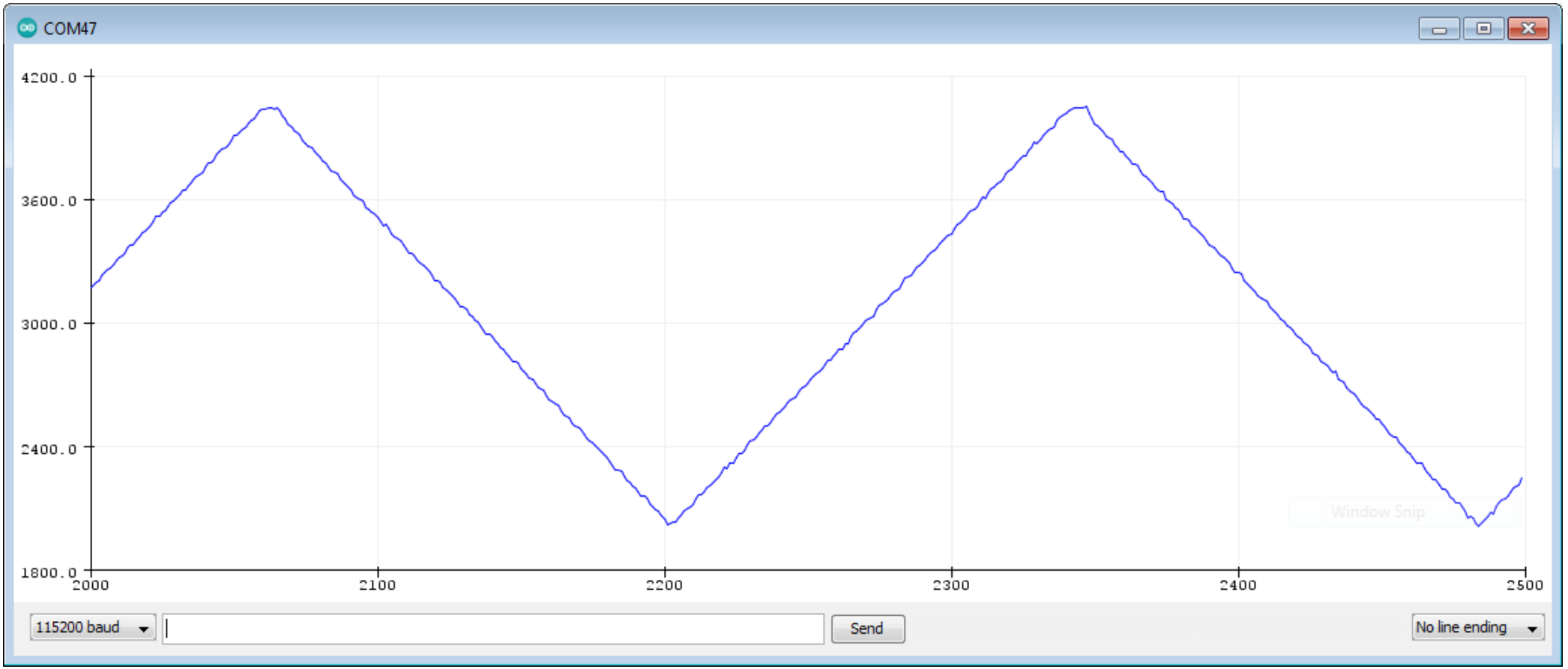
```c
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#define NDATA   500
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
DAC_HandleTypeDef hdac;
TIM_HandleTypeDef htim6;
UART_HandleTypeDef huart2;
volatile uint8_t data_ready = 0;
volatile uint16_t data[NDATA];
char msg[80];
uint32_t status;

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

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

F446RE_DAC_triangle

```
int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_DAC_Init();
  MX_TIM6_Init();
  MX_USART2_UART_Init();
  HAL_DAC_Start(&hdac,DAC_CHANNEL_1);                                    // A DAC indítása
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 2000); // Offset beállítása
  HAL_DACEx_TriangleWaveGenerate(&hdac, DAC_CHANNEL_1, DAC_TRIANGLEAMPLITUDE_2047);
  HAL_TIM_Base_Start(&htim6);                                            // Timer6 indítása
  while (1) {
    data_ready = 0;
    status = HAL_ADC_Start_DMA(&hadc1,(uint32_t*)data,NDATA);    // ADC mérés indítás
    while(!data_ready);                                                  // 500 adatra várunk
    for(int i = 0; i<NDATA; i++) {                                       // 500 adat kiküldése
      sprintf(msg, "%d\r\n", data[i]);
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    }
    HAL_ADC_Stop_DMA(&hadc1);
    HAL_Delay(5000);
  }
}
```

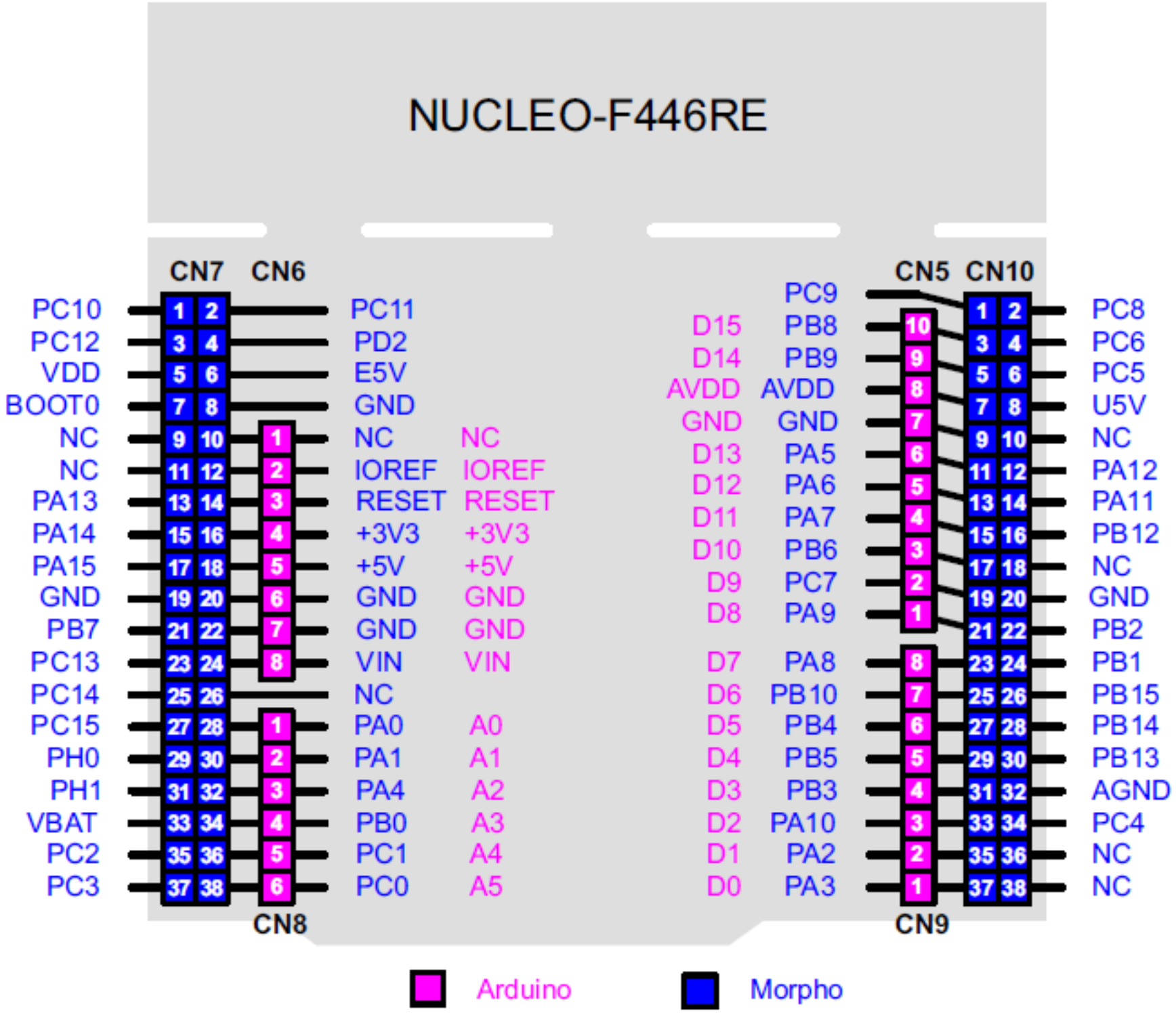
F446RE_DAC_triangle: eredmények

- Az ADC-vel mért adatokat az Arduino IDE Serial Plotter eszközével egyszerűen megjeleníthetjük
- Az ábrán kb. 280 adatpont (280 * 1.78 µs = 0.498 ms) a periódusidő, ami durván 2 kHz-es frekvenciát jelent



F446RE_DAC_triangle: eredmények

- A **Hscope** alkalmazással is hasonló eredményt kaptunk periódusidő: 498.64 µs, a frekvencia 2.01 kHz


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 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
CN9
Arduino
Morpho