

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

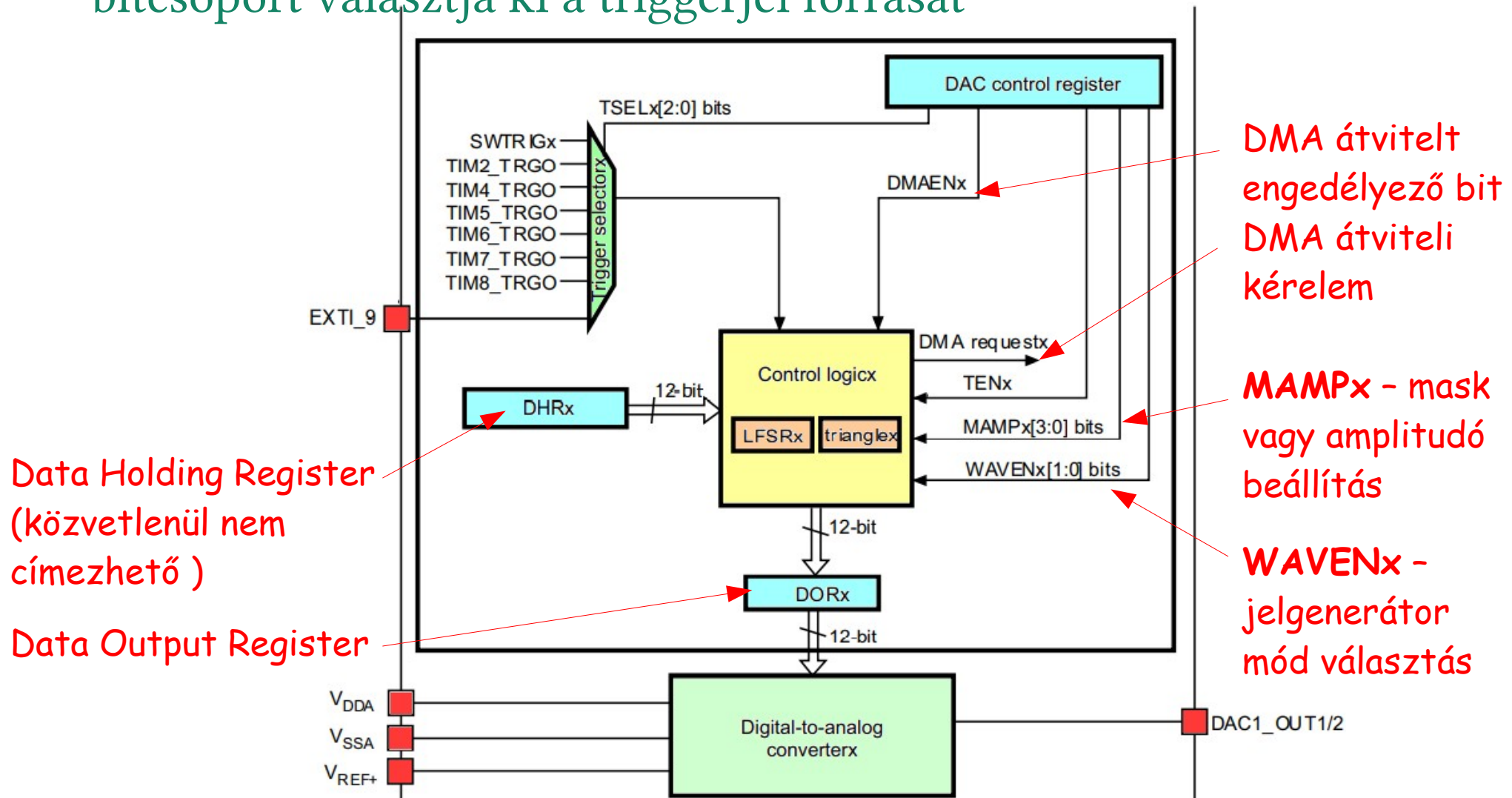
Adatlapok, kézikönyvek:

- 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



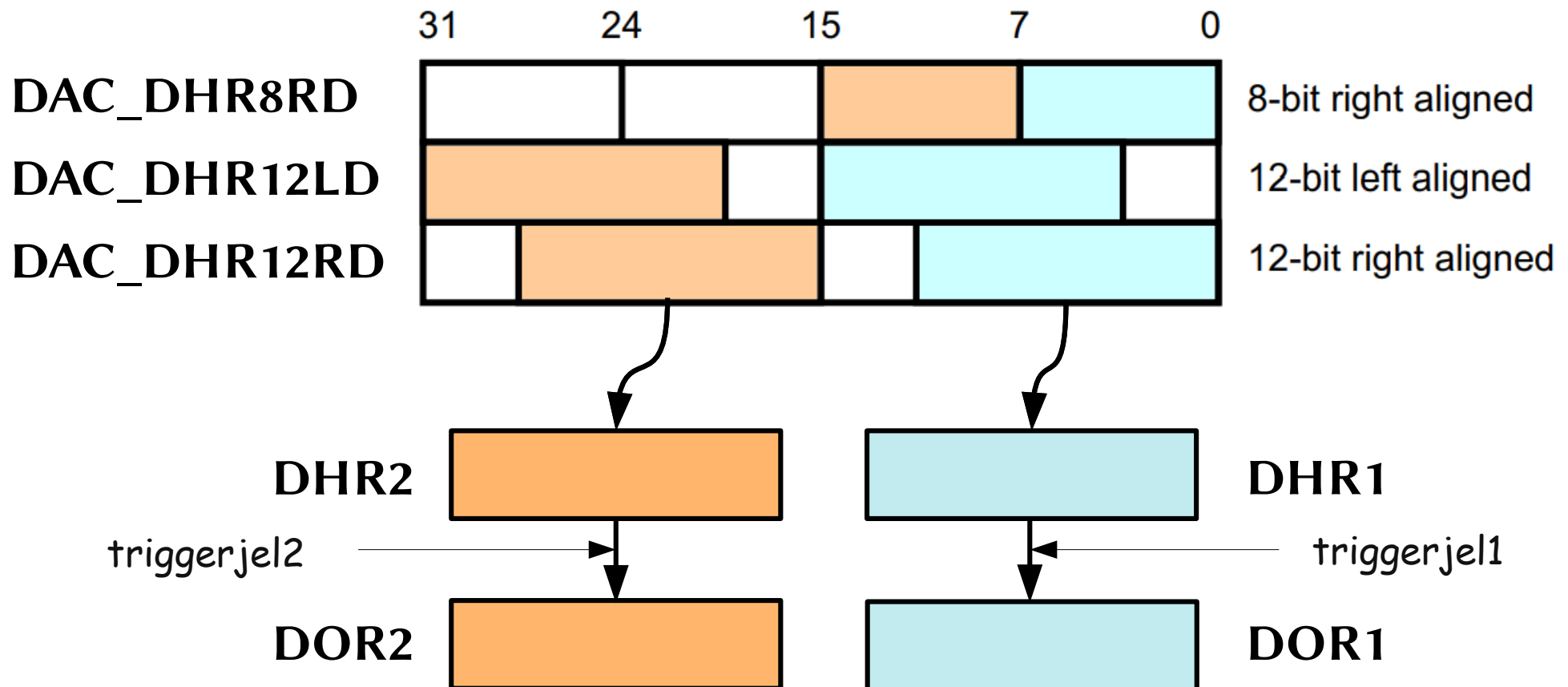
Emlékeztető: A digitál-analóg átalakító felépítése

- A **DMAENx** bit engedélyezi a DMA átvitelt, a **TSELx[2:0]** bitsorozat választja ki a triggerjel forrását



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.	DMAU DRIE2	DMA EN2	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.	DMAU DRIE1	DMA EN1	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 kérés 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!

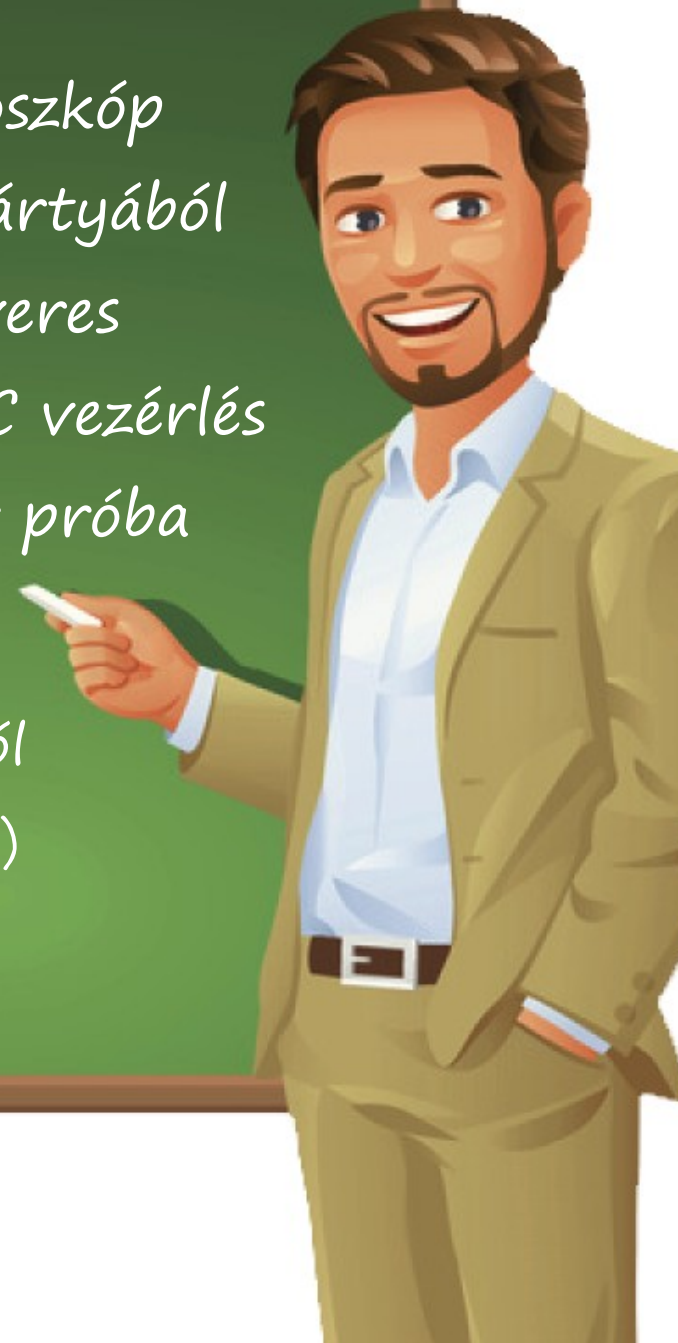
Példaprogramok

F103_ADC_2ch – kétcsatornás oszcilloszkóp
készítése „blue pill” kártyából

F446RE_DAC_basic2 – egyszerű szoftveres
kétcsatornás DAC vezérlés

F446RE_DAC_noisegen – zajgenerátor próba

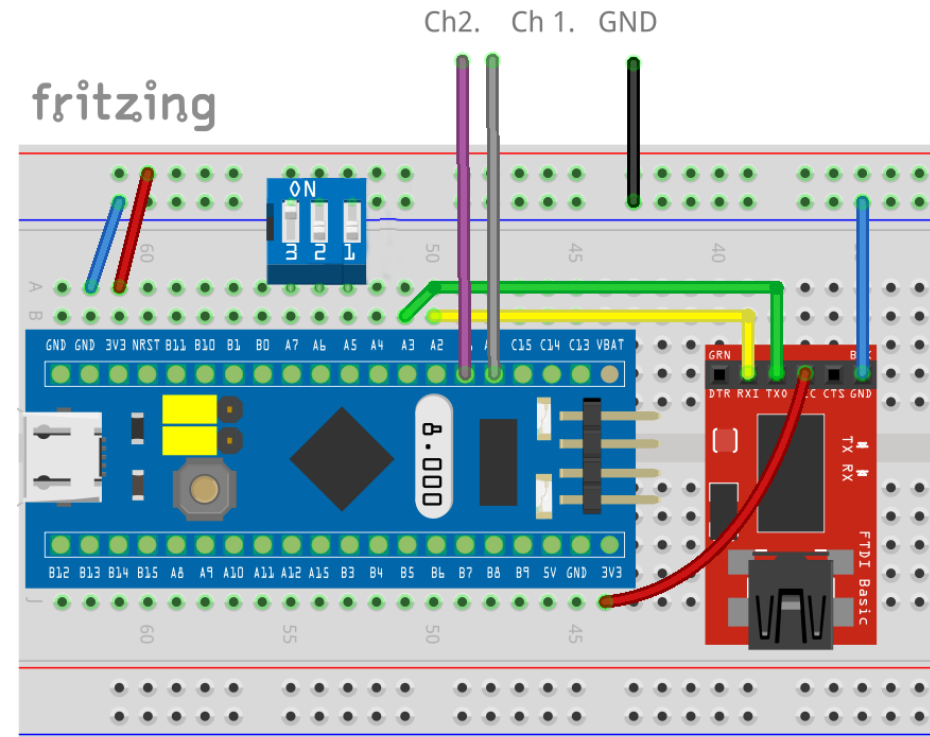
F446RE_DAC_audio – kétcsatornás
jelkeltés hullámtáblából
(szimultán DMA mód)



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: $1\ \mu\text{s}$ – $200\ \mu\text{s}$ (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)



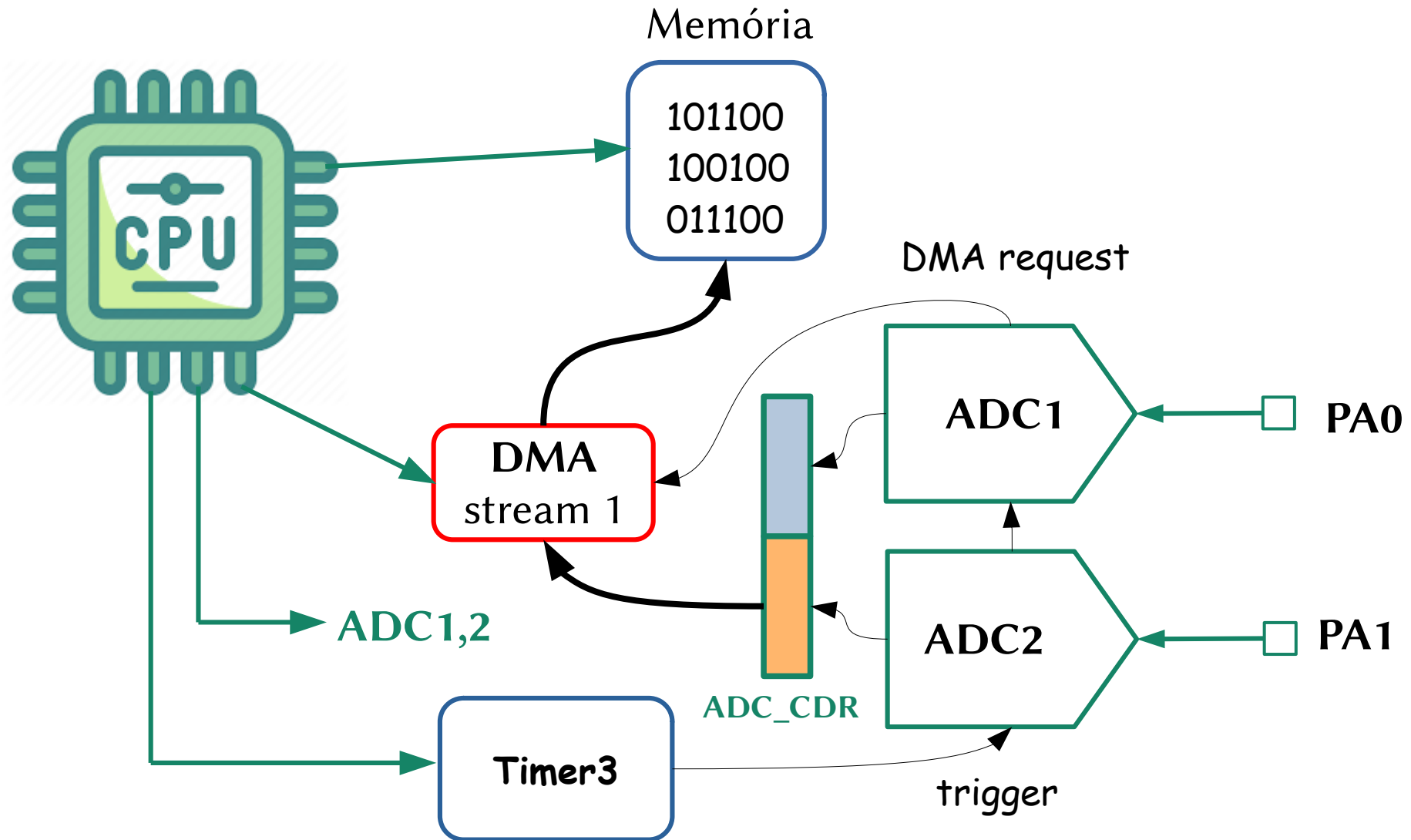
F103_ADC_2ch: kétcsatornás oszcilloszkóp

- 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 μ s	0,5 ms
001	1	2 μ s	1 ms
010	4	5 μ s	2,5 ms
011	9	10 μ s	5 ms
100	19	20 μ s	10 ms
101	49	50 μ s	25 ms
110	99	100 μ s	50 ms
111	199	200 μ s	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)

The image displays the STM32CubeMX software interface for configuring the ADC1 peripheral on an STM32F103C8Tx LQFP48 microcontroller. The left sidebar shows the project tree with 'ADC1' selected under the 'Analog' category. The main window shows the 'ADC1 Mode and Configuration' settings.

ADC1 Mode and Configuration:

- Mode:** ☒ IN0, ☐ IN1
- Configuration:**
 - Reset Configuration:** [Reset Configuration]
 - Parameter Settings:** ☒ NVIC Settings, ☒ DMA Settings, ☒ GPIO Settings, ☒ User Constants
- Search (Ctrl+F):** [Search]
- ADCs_Common_Settings:**
 - Mode:** Dual regular simultaneous mode only
- ADC_Settings:**
 - Data Alignment:** Right alignment
 - Scan Conversion Mode:** Disabled
 - Continuous Conversion Mode:** Disabled
 - Discontinuous Conversion Mode:** Disabled
- ADC_Regular_ConversionMode:**
 - Enable Regular Conversions:** Enable
 - Number Of Conversion:** 1
 - External Trigger Conversion Source:** Timer 3 Trigger Out event
 - Rank:** 1
 - Channel:** Channel 0
 - Sampling Time:** 1.5 Cycles
- ADC_Injected_ConversionMode:**
 - Enable Injected Conversions:** Disable
- WatchDog:**
 - Enable Analog WatchDog Mode:** ☐

Pinout Diagram: The diagram shows the STM32F103C8Tx LQFP48 package with pins labeled. The top row includes VDD, VSS, PB9, PB8, BOOT0, PB7, PB6, PB5, PB4, PB3, PA15, PA14, and SYS_JTCK-SWCLK. The bottom row includes VBAT, PC13, PC14, PC15, PD0, PD1, NRST, VSSA, VDDA, ADC1_IN0, ADC2_IN1, USART2_TX, PA3, PA4, PA5, PA6, PA7, PB0, PB1, PB2, PB10, PB11, VSS, and VDD. The right side shows VDD, VSS, PA13, PA12, PA11, PA10, PA9, PA8, PB15, PB14, PB13, and PB12. The left side shows RCC_OSC32_IN, RCC_OSC32_OUT, RCC_OSC_IN, RCC_OSC_OUT, and USART2_RX. The bottom row also includes GPIO_Input, GPIO_Input, and GPIO_Input.

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

The image displays the STM32CubeMX configuration interface for an STM32F103C8Tx LQFP48 microcontroller. The left sidebar shows the project configuration tree with 'ADC2' selected under the 'Analog' category. The main window shows the 'ADC2 Mode and Configuration' settings.

ADC2 Mode and Configuration

- Mode:** ☐ IN0, ☒ IN1
- Configuration:**
 - Reset Configuration:** [Reset]
 - User Constants:** ☒
 - NVIC Settings:** ☒
 - GPIO Settings:** ☒
 - Parameter Settings:** ☒
- Search (Ctrl+F):** [Search]
- ADCs_Common_Settings:**
 - Mode:** Dual regular simultaneous mode only
- ADC_Settings:**
 - Data Alignment:** Right alignment
 - Scan Conversion Mode:** Disabled
 - Continuous Conversion Mode:** Disabled
 - Discontinuous Conversion Mode:** Disabled
- ADC_Regular_ConversionMode:**
 - Enable Regular Conversions:** Enable
 - Number Of Conversion:** 1
 - Rank:** 1
 - Channel:** Channel 1
 - Sampling Time:** 1.5 Cycles
- ADC_Injected_ConversionMode:**
 - Enable Injected Conversions:** Disable
- WatchDog:**
 - Enable Analog WatchDog Mode:** ☐

Pinout view

The pinout view shows the STM32F103C8Tx LQFP48 package with the following pins and connections:

- VDD:** VDD
- VSS:** VSS
- PB9:** PB9
- PB8:** PB8
- BOOT0:** BOOT0
- PB7:** PB7
- PB6:** PB6
- PB5:** PB5
- PB4:** PB4
- PB3:** PB3
- PA15:** PA15
- PA14:** PA14
- PA13:** PA13
- PA12:** PA12
- PA11:** PA11
- PA10:** PA10
- PA9:** PA9
- PA8:** PA8
- PB15:** PB15
- PB14:** PB14
- PB13:** PB13
- PB12:** PB12
- VSS:** VSS
- VDD:** VDD
- PA3:** PA3
- PA4:** PA4
- PA5:** PA5
- PA6:** PA6
- PA7:** PA7
- PB0:** PB0
- PB1:** PB1
- PB2:** PB2
- PB10:** PB10
- PB11:** PB11
- VSS:** VSS
- VDD:** VDD
- USART2_RX:** USART2_RX
- GPIO_Input:** GPIO_Input
- GPIO_Input:** GPIO_Input
- GPIO_Input:** GPIO_Input
- USART2_TX:** USART2_TX
- ADC1_IN0:** ADC1_IN0
- ADC2_IN1:** ADC2_IN1
- RCC_OSC32_IN:** RCC_OSC32_IN
- RCC_OSC32_OUT:** RCC_OSC32_OUT
- RCC_OSC_IN:** RCC_OSC_IN
- RCC_OSC_OUT:** RCC_OSC_OUT
- NRST:** NRST
- VSSA:** VSSA
- VDDA:** VDDA
- PA0-...:** PA0-...
- PA1:** PA1
- PA2:** PA2
- PC13-...:** PC13-...
- PC14-...:** PC14-...
- PC15-...:** PC15-...
- PD0-...:** PD0-...
- PD1-...:** PD1-...
- SYSTICK-SWCLK:** SYSTICK-SWCLK
- SYS_JTMS-SWDIO:** SYS_JTMS-SWDIO

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

The screenshot shows the 'DMA Mode and Configuration' window in STM32CubeMX. On the left is a sidebar with a search bar and a tree view under 'System Core' containing DMA, GPIO, IWDG, NVIC, RCC, SYS, and WWDG. The 'DMA' item is selected. The main area is titled 'DMA Mode and Configuration' and has a 'Configuration' tab. It shows two checked items: 'DMA1' and 'MemToMem'. Below this is a table with columns: DMA Request, Channel, Direction, and Priority. The table contains one entry: ADC1, DMA1 Channel 1, Peripheral To Memory, and Low. Below the table are 'Add' and 'Delete' buttons. Under 'DMA Request Settings', there are two sections: 'Peripheral' and 'Memory'. In the 'Peripheral' section, 'Mode' is set to 'Normal' and 'Increment Address' is unchecked. In the 'Memory' section, the checkbox is checked. At the bottom, 'Data Width' is set to 'Word' for both Peripheral and Memory.

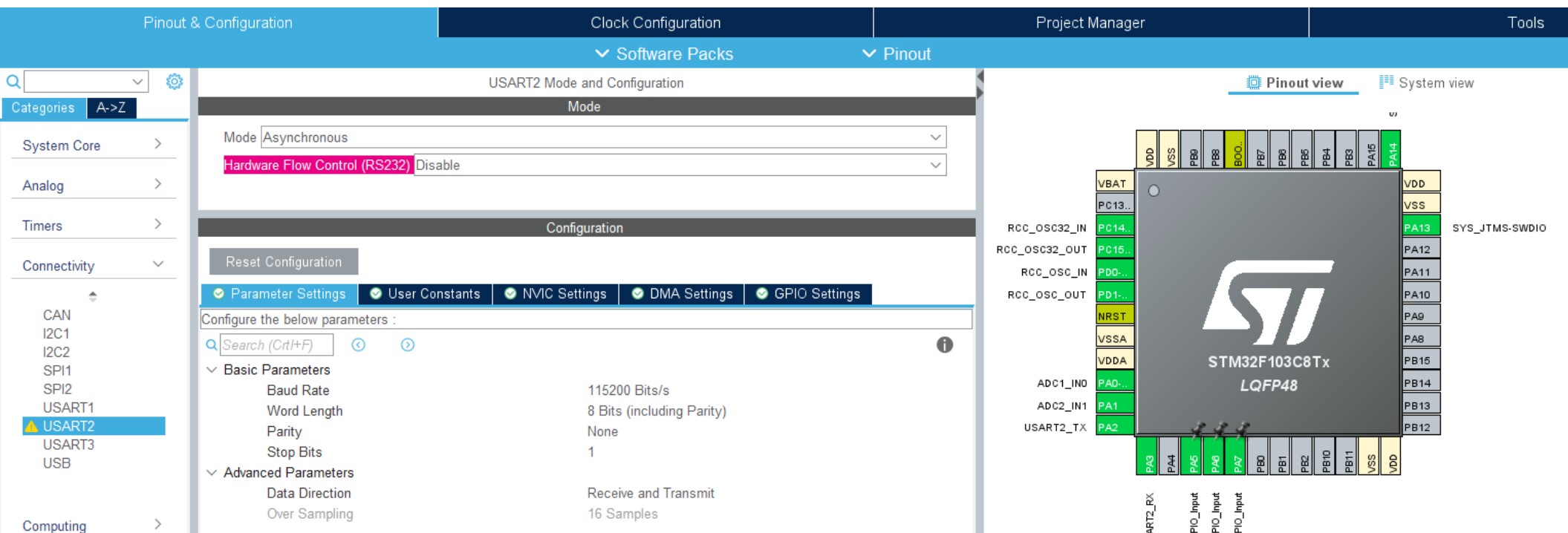
DMA Request	Channel	Direction	Priority
ADC1	DMA1 Channel 1	Peripheral To Memory	Low

DMA Request Settings

Peripheral		Memory
Mode: Normal	Increment Address: ☐	☒
Data Width: Word		Word

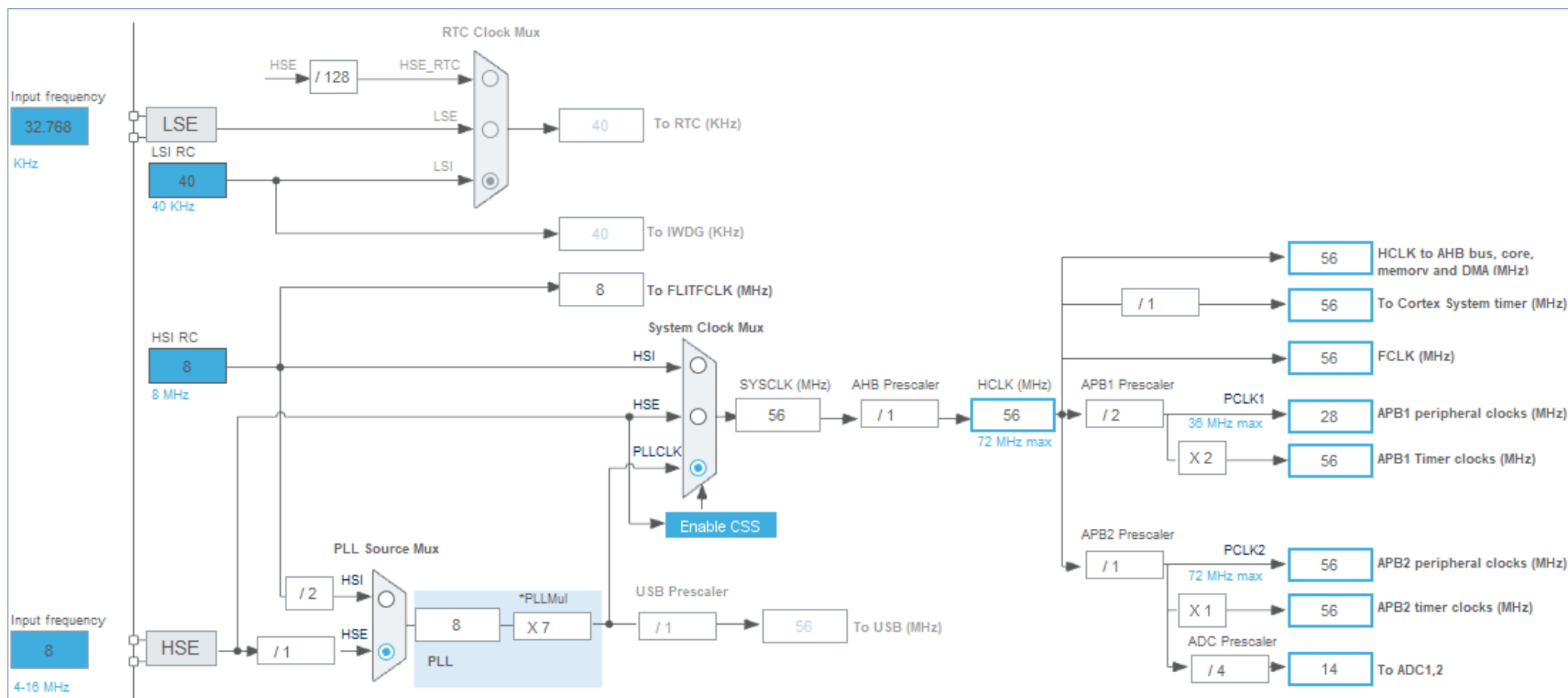
USART2 konfigurálása

- 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 összeköttetéseket használ...

The screenshot shows the 'TIM3 Mode and Configuration' window in STM32CubeMX. It is divided into two main sections: 'Mode' and 'Configuration'.

Mode Section:

- Slave Mode: Disable
- Trigger Source: Disable
- ☒ Internal Clock
- Channel1: Disable
- Channel2: Disable
- Channel3: Disable
- Channel4: Disable
- Combined Channels: Disable
- ☐ XOR activation
- ☐ One Pulse Mode

Configuration Section:

- Reset Configuration button
- Tabs: Parameter Settings (selected), User Constants, NVIC Settings, DMA Settings
- Configure the below parameters :
- Search (Ctrl+F) field
- Counter Settings:
 - Prescaler (PSC - 16 bits value): 55
 - Counter Mode: Up
 - Counter Period (AutoReload Register - 16 bits value): 9
 - Internal Clock Division (CKD): No Division
 - auto-reload preload: Disable
- Trigger Output (TRGO) Parameters:
 - Master/Slave Mode (MSM bit): Disable (Trigger input effect not delay...)
 - Trigger Event Selection: Update Event

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)

The screenshot displays the STM32CubeMX software interface. On the left, the 'System Core' sidebar shows various peripherals, with 'GPIO' selected. The main window is titled 'GPIO Mode and Configuration' and shows a 'Configuration' tab. Under 'Group By Peripherals', the 'GPIO' checkbox is checked. Below this, a table lists the configured pins:

Pin Name	Signal on ...	GPIO	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PA5	n/a	n/a	Input mode	Pull-up	n/a		✓
PA6	n/a	n/a	Input mode	Pull-up	n/a		✓
PA7	n/a	n/a	Input mode	Pull-up	n/a		✓

To the right of the table is a 'Search Signals' field and a checkbox for 'Show only Modified Pins'. On the far right, a 'Pinout view' shows the physical pinout of the STM32F103C8Tx LQFP48 package. The pins are color-coded: green for digital I/O, yellow for power, and grey for other functions. The pins are labeled as follows:

- Top: VDD, VSS, PB9, PB8, BOOT0, PB7, PB6, PB5, PB4, PB3, PA15, PA14
- Right: VDD, VSS, PA13, PA12, PA11, PA10, PA9, PA8, PB15, PB14, PB13, PB12
- Bottom: PA3, PA4, PA5, PA6, PA7, PB0, PB1, PB2, PB10, PB11, VSS, VDD
- Left: VBAT, PC13-..., PC14-..., PC15-..., PD0-O..., PD1-O..., NRST, VSSA, VDDA, ADC1_IN0, ADC2_IN1, USART2_TX

F103_ADC-2ch: main.c (részlet)

```
#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);

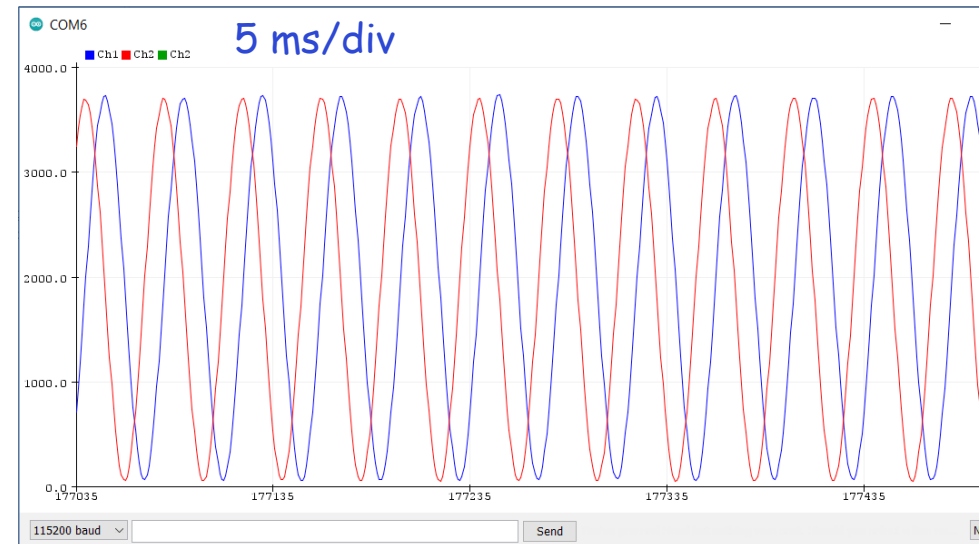
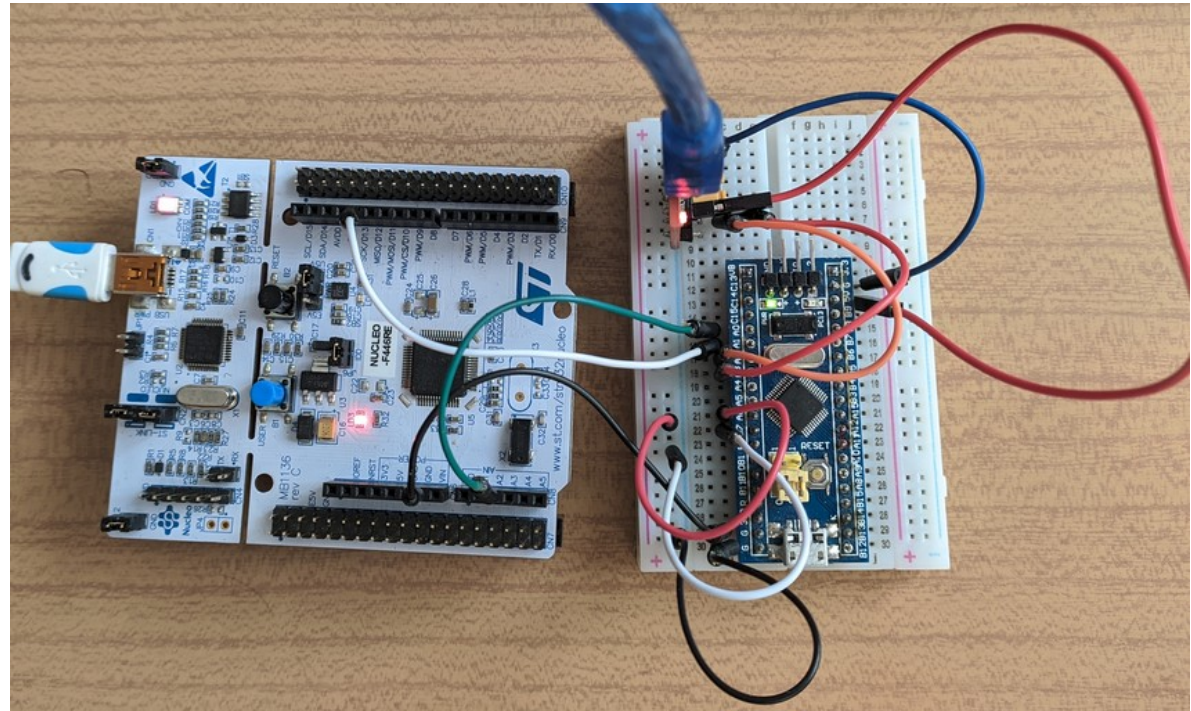
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) {
    data_ready = 1; // Jelzi, hogy vége a mérésnek
}
```

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] & 0xFFFF;
            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);
    }
}
```

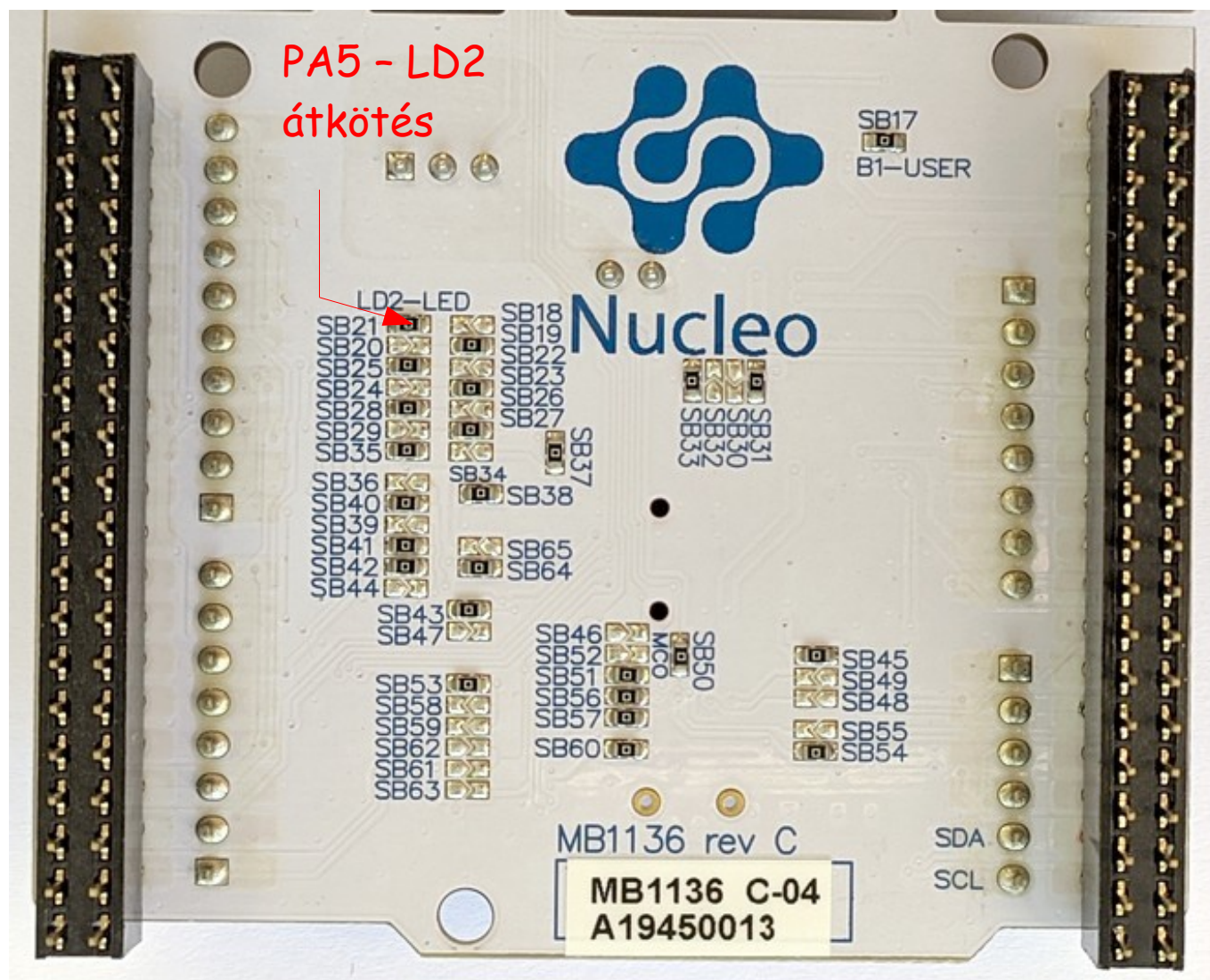
F103_ADC-2ch futási eredmény

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



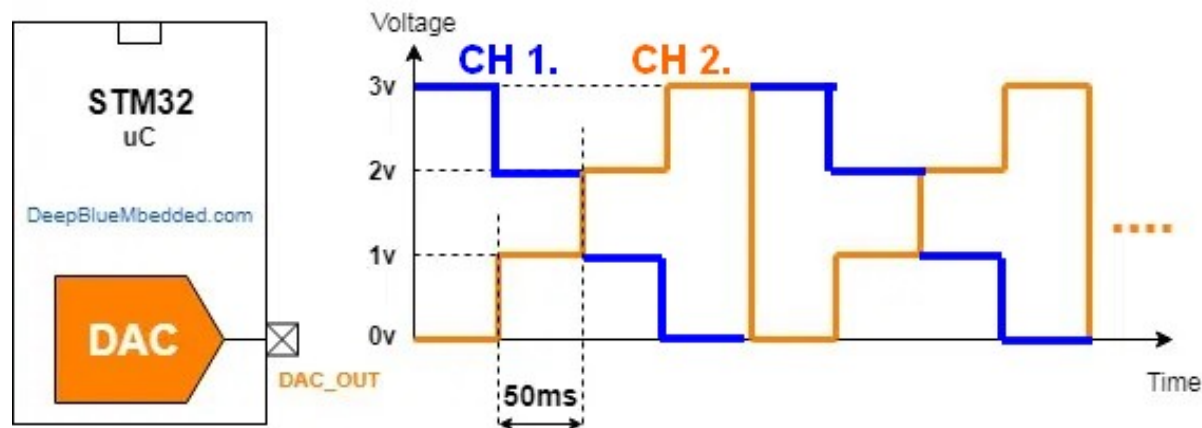
A DAC 2. csatorna „felszabadítása”

- 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()** függvénnnyel
- 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 sorrendben, s egy kicsi eltolással, s ezt a sorozatot ismételtetjük
- 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 V_{REF} 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

The screenshot displays the STM32CubeMX Pinout & Configuration window. The left sidebar shows the 'Categories' list with 'DAC' selected. The main area is titled 'DAC Mode and Configuration'. Under the 'Mode' section, 'OUT1 Configuration' and 'OUT2 Configuration' are checked, while 'External Trigger' is unchecked. The 'Configuration' section includes tabs for 'Reset Configuration', 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings' (selected), and 'User Constants'. Below these, a search bar is present, and the 'Configure the below parameters' section shows 'DAC Out1 Settings' and 'DAC Out2 Settings'. For both, 'Output Buffer' is set to 'Enable' and 'Trigger' is set to 'None'. To the right, a pinout diagram of the STM32F446RETx LQFP64 package is shown, with pins PA4 and PA5 highlighted in green and labeled 'DAC_OUT1' and 'DAC_OUT2' respectively. The package also shows other pins like VDD, VSS, PB9, PB8, BOO, PB7, PB6, PB5, PB4, PB3, PD2, PC12, PC11, PC10, PA15, PA14, VBAT, PC13, PC14, PC15, PH0, PH1, NRST, PC0, PC1, PC2, PC3, VSSA, VDDA, PA0, PA1, PA2, PA3, VSS, VDD, PA4, PA5, PA6, PA7, PC4, PC5, PB0, PB1, PB2, PB10, VCA, VSS, and VDD.

F446RE_DAC_basic2: main.c releváns sorai

```
#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

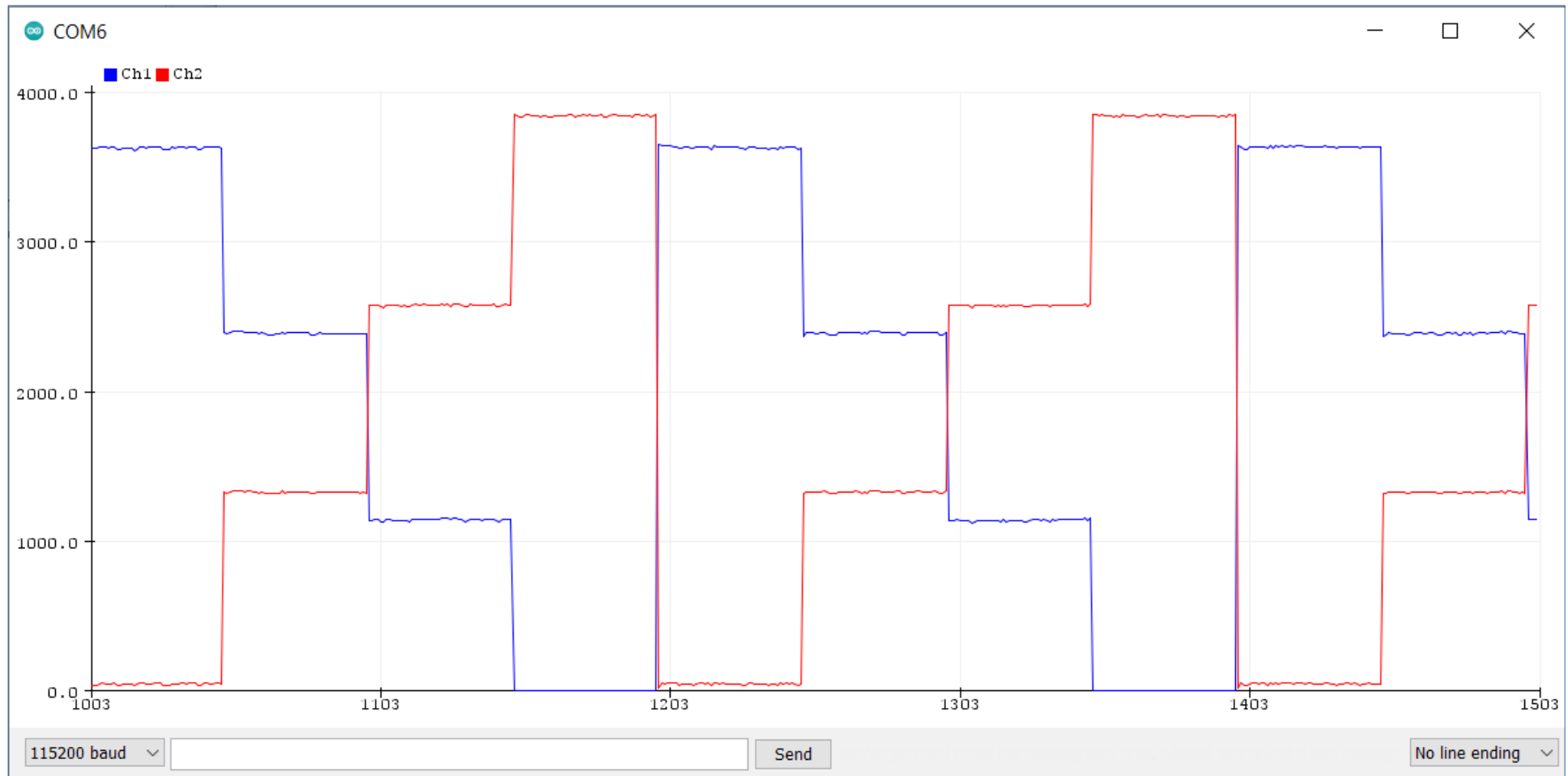
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DAC_Init();

    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 μ s-onként történt a mintavételezés, így egy osztás (100 adat) 2 ms, a teljes ábra pedig 10 ms időtartamot fed le

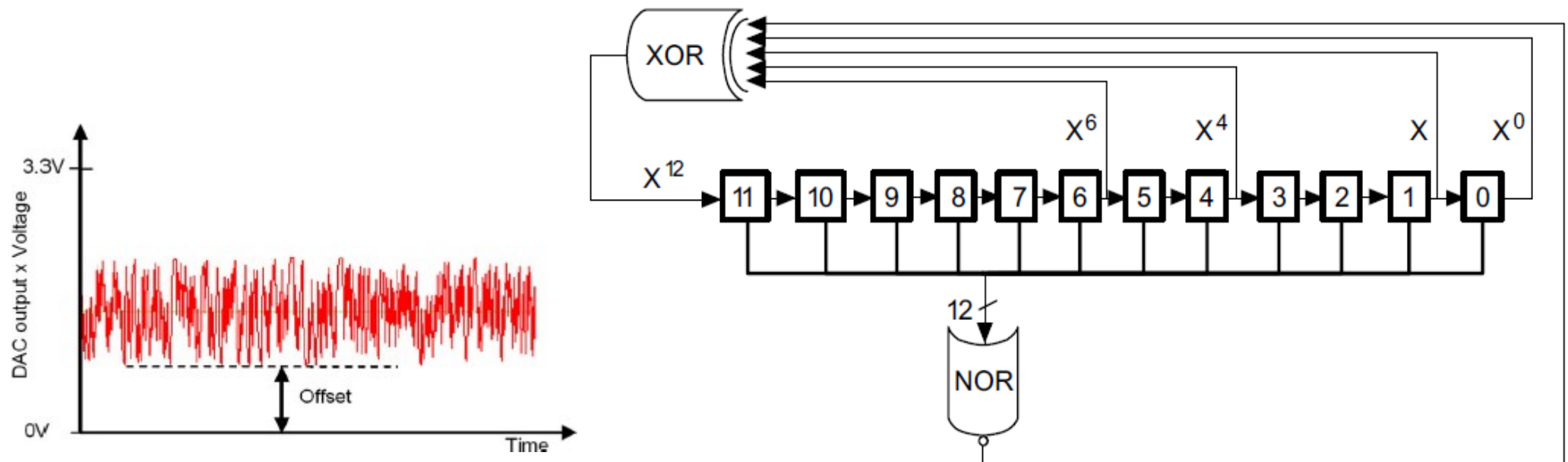


Zajongjunk! F446RE_DAC_noise

- 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 μ s 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 μ s-onként vált értéket)

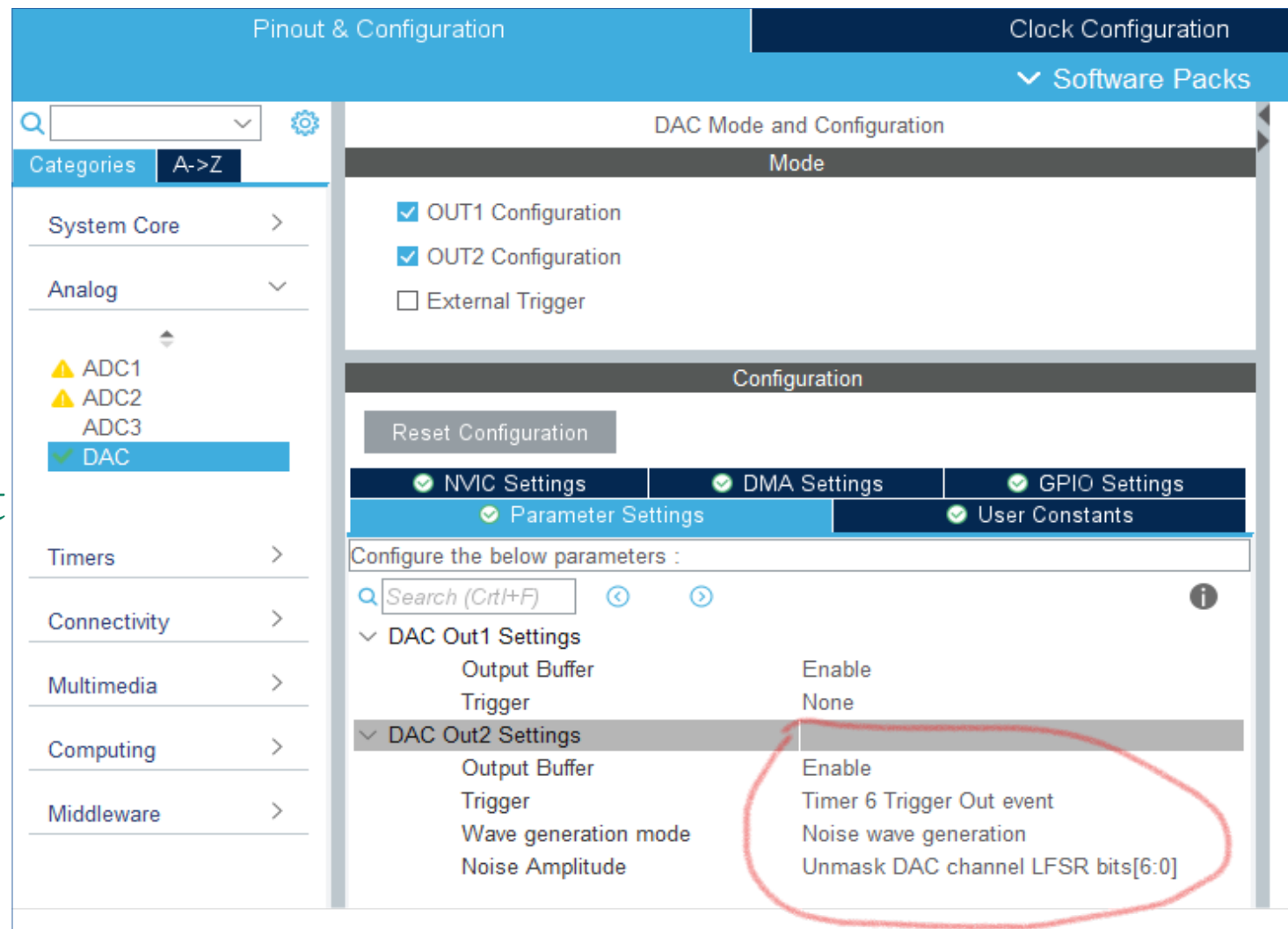
Emlékeztető: Zajgenerátor mód

- A DAC vezérlő regiszterében $WAVEx[1:0] = 01$ beállítás bekapcsolja a zajjel generátort, amely egy álvéletlen számot ad hozzá a $DHRx$ 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 $MAMPx[3:0]$ bitcsoport beállítása szabja meg
- Az *offset* és a maximális amplitúdó összege 4095-nél nem lehet nagyobb!



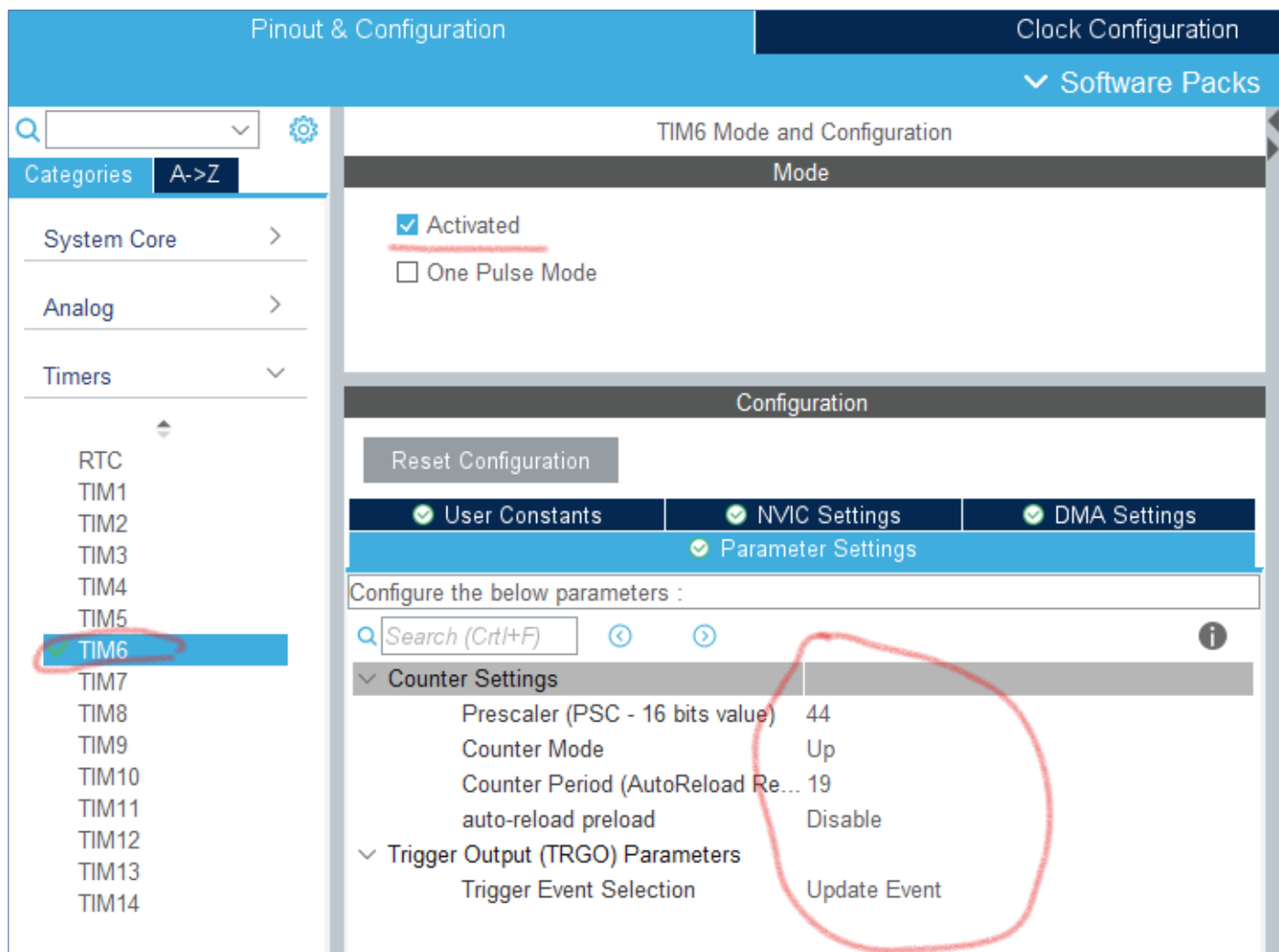
A DAC konfigurálása

- 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_noise: main.c (részlet)

```
#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_noise: futási eredmény

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

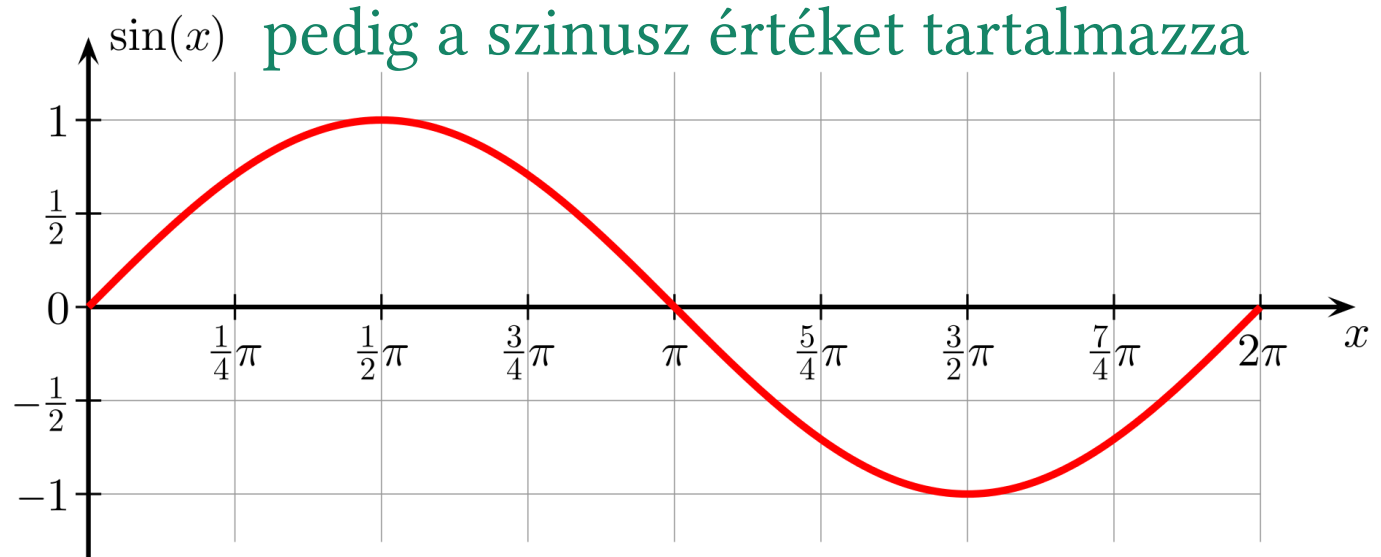


F446RE_DAC_audio

- 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 μ s-onként végezzük, a 200 adatot tartalmazó **wavetable** tömbben kombináltan egy periódusnyi szinusz- illetve koszinusz hullámot tárolunk (a függvényértékeket 0 – 4095 közé normálva). A 32 bites adatok felső fele a koszinusz, az alsó fele pedig a szinusz értéket tartalmazza

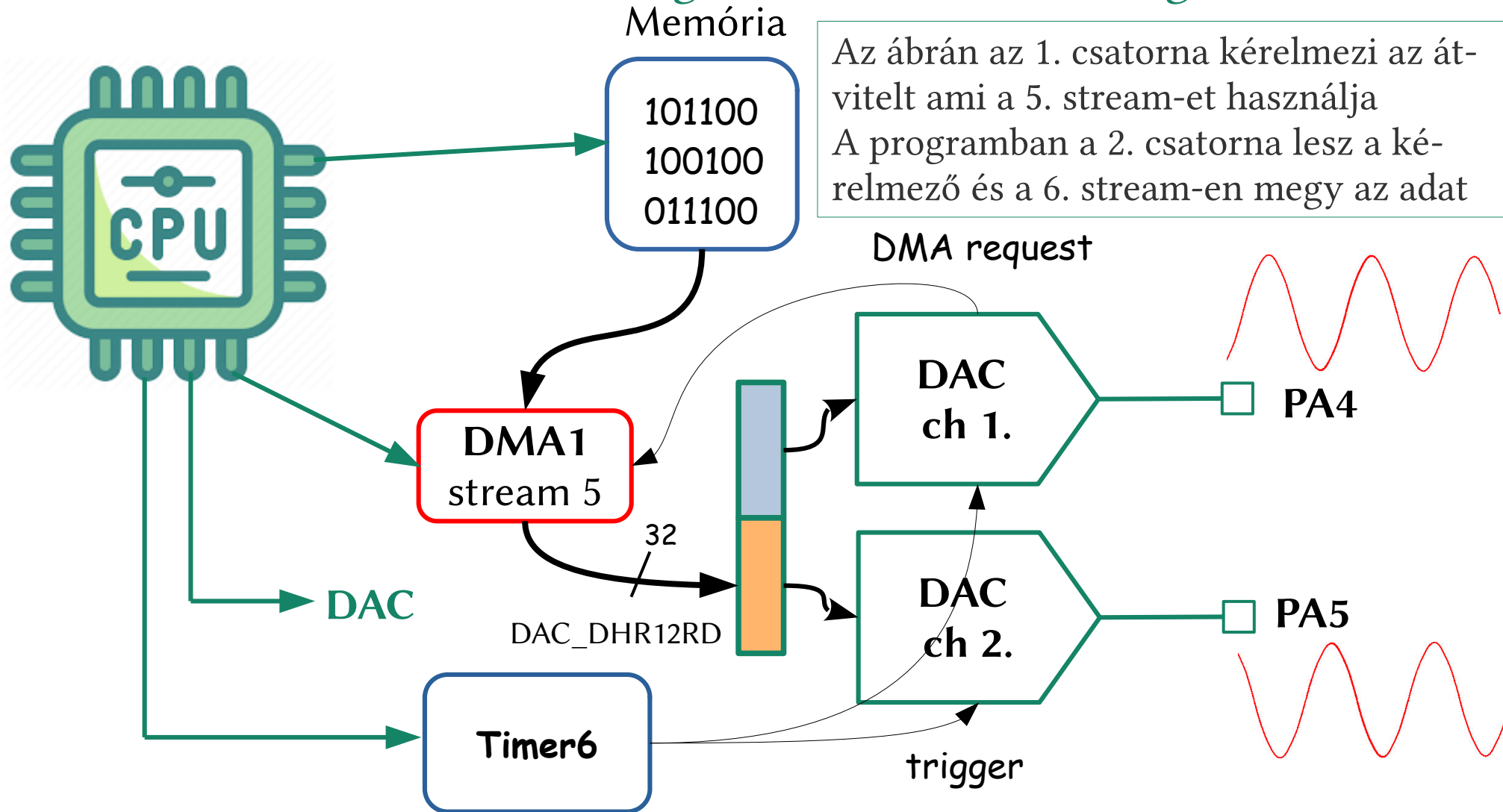
$$Freq = \frac{F_{CPU}}{Psc \cdot Period \cdot NDATA}$$

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



Az adatáramlás sematikus vázlat

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



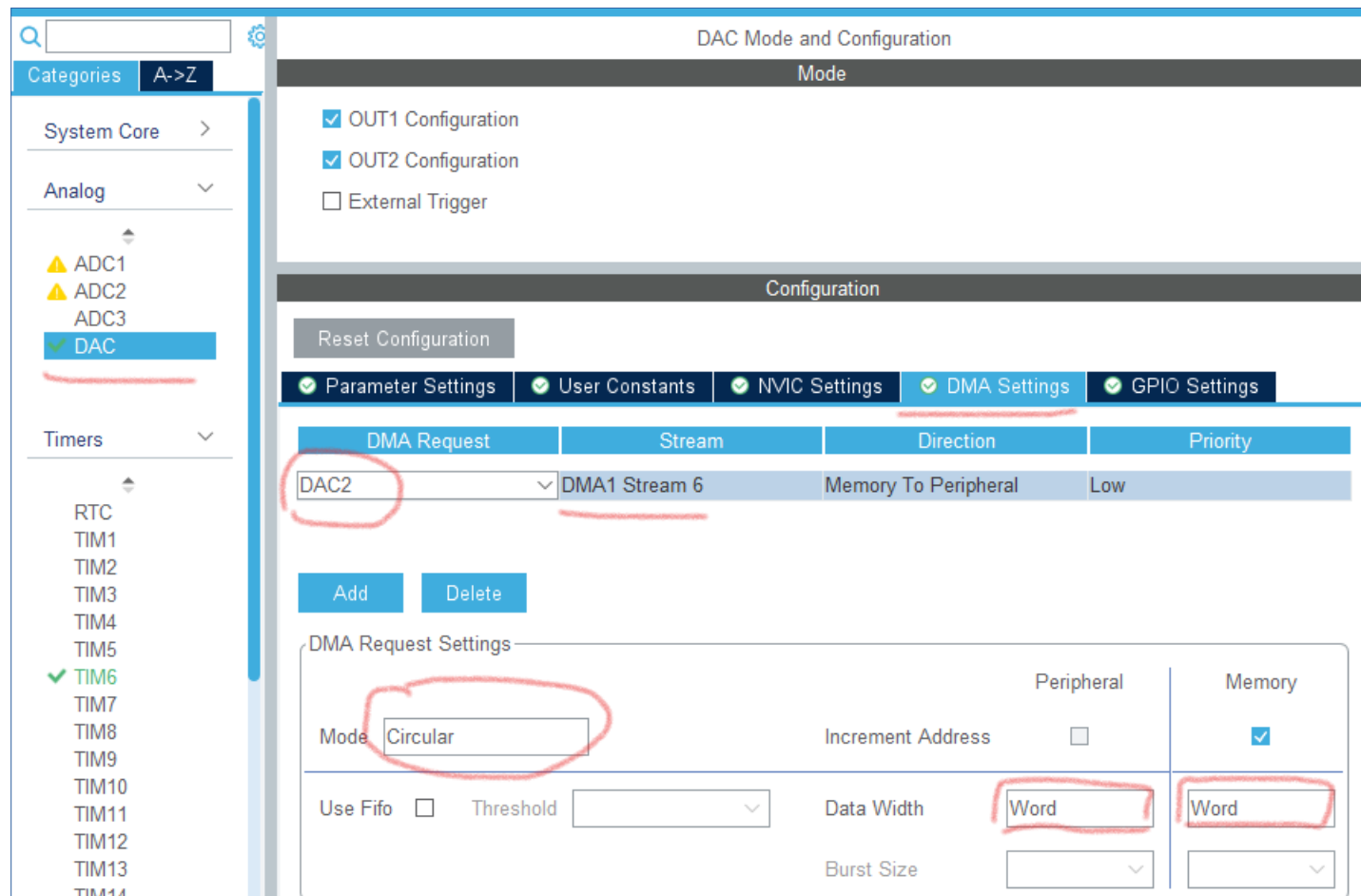
A DAC konfigurálása

- 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

The screenshot displays the STM32CubeMX Pinout & Configuration window. The left sidebar shows the 'Categories' list with 'DAC' selected. The main window is titled 'DAC Mode and Configuration'. Under the 'Mode' section, 'OUT1 Configuration' and 'OUT2 Configuration' are checked, while 'External Trigger' is unchecked. The 'Configuration' section shows 'Reset Configuration' and tabs for 'NVIC Settings', 'DMA Settings', 'GPIO Settings', 'Parameter Settings', and 'User Constants'. The 'Parameter Settings' tab is active, showing 'Configure the below parameters :'. Under 'DAC Out1 Settings', 'Output Buffer' is 'Enable', 'Trigger' is 'Timer 6 Trigger Out event', and 'Wave generation mode' is 'Disabled'. Similarly, under 'DAC Out2 Settings', 'Output Buffer' is 'Enable', 'Trigger' is 'Timer 6 Trigger Out event', and 'Wave generation mode' is 'Disabled'. The pinout diagram on the right shows the STM32F446RETx LQFP64 package. Pins PA13 and PA14 are highlighted in green, corresponding to DAC_OUT1 and DAC_OUT2. The pinout diagram also shows other pins like VDD, VSS, PB9, PB8, PB7, PB6, PB5, PB4, PB3, PD2, PC12, PC11, PC10, PA15, PA14, PA13, PA12, PA11, PA10, PA9, PA8, PC9, PC8, PC7, PC6, PB15, PB14, PB13, PB12, PA3, VSS, VDD, PA4, PA5, PA6, PA7, PC4, PC5, PB0, PB1, PB2, PB10, VCA, VSS, and VDD.

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

The screenshot shows the STM32CubeMX Pinout & Configuration window. The left sidebar lists various components: System Core, Analog (ADC1, ADC2, ADC3, DAC), and Timers (RTC, TIM1, TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM8, TIM9). TIM6 is selected and highlighted. The main window displays the 'TIM6 Mode and Configuration' settings. The 'Mode' section has 'Activated' checked and circled in red. The 'Configuration' section has tabs for 'Parameter Settings', 'User Constants', 'NVIC Settings', and 'DMA Settings'. The 'Parameter Settings' tab is active, showing 'Counter Settings' and 'Trigger Output (TRGO) Parameters'. The 'Counter Settings' section has 'Prescaler (PSC - 16 bits value)' set to 44, 'Counter Mode' set to 'Up', and 'Counter Period (AutoReload Register - 16 bits...)' set to 19. The 'Trigger Output (TRGO) Parameters' section has 'Trigger Event Selection' set to 'Update Event', which is also circled in red. A yellow box on the right contains the text '200 adat esetén:' and the formula
$$Freq = \frac{90\text{ MHz}}{45 \cdot 20 \cdot 200} = 500\text{ Hz}$$
.

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 *Alignment* paramétertől függően a megfelelő regisztercím (`DHR8RD`, `DHR12RD`, `DHR12LD` valamelyike) legyen a periféria cím

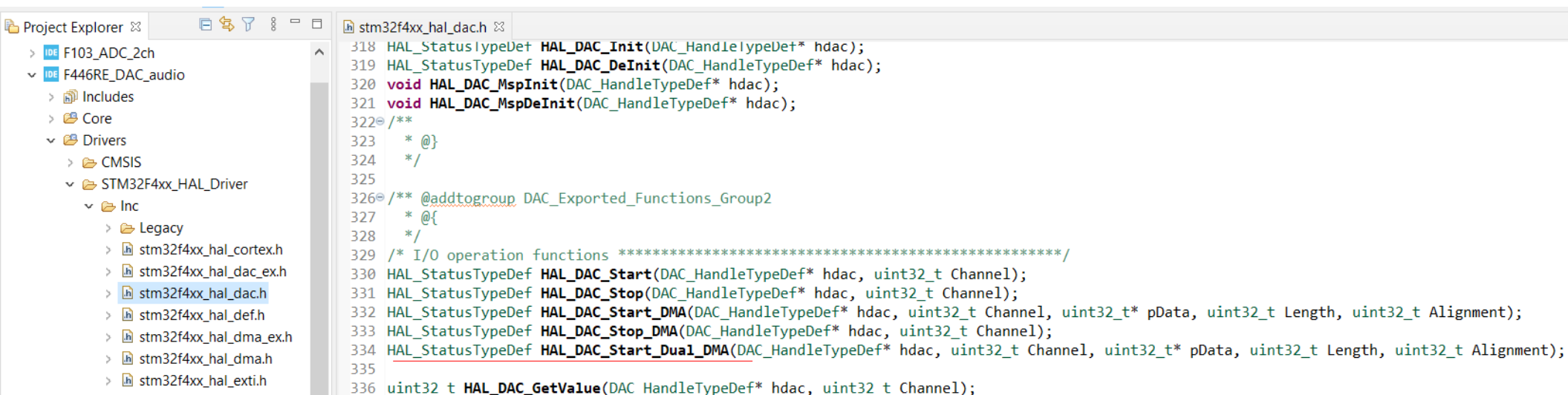
- Szintaxis:

```
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
- ❖ *Length* – 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!



The screenshot shows an IDE interface. On the left, the Project Explorer displays a tree structure with the following items: F103_ADC_2ch, F446RE_DAC_audio, Includes, Core, Drivers, CMSIS, STM32F4xx_HAL_Driver, Inc, Legacy, stm32f4xx_hal_cortex.h, stm32f4xx_hal_dac_ex.h, stm32f4xx_hal_dac.h (highlighted), stm32f4xx_hal_def.h, stm32f4xx_hal_dma_ex.h, stm32f4xx_hal_dma.h, and stm32f4xx_hal_exti.h. The main editor window displays the content of stm32f4xx_hal_dac.h, showing lines 318 through 336. The code includes declarations for HAL_DAC_Init, HAL_DAC_DeInit, HAL_DAC_MspInit, HAL_DAC_MspDeInit, and HAL_DAC_GetValue. It also features a section for I/O operation functions, including HAL_DAC_Start, HAL_DAC_Stop, HAL_DAC_Start_DMA, HAL_DAC_Stop_DMA, and HAL_DAC_Start_Dual_DMA (highlighted with a red line).

```
318 HAL_StatusTypeDef HAL_DAC_Init(DAC_HandleTypeDef* hdac);
319 HAL_StatusTypeDef HAL_DAC_DeInit(DAC_HandleTypeDef* hdac);
320 void HAL_DAC_MspInit(DAC_HandleTypeDef* hdac);
321 void HAL_DAC_MspDeInit(DAC_HandleTypeDef* hdac);
322 /**
323  * @}
324  */
325
326 /** @addtogroup DAC_Exported_Functions_Group2
327  * @{
328  */
329 /* I/O operation functions *****/
330 HAL_StatusTypeDef HAL_DAC_Start(DAC_HandleTypeDef* hdac, uint32_t Channel);
331 HAL_StatusTypeDef HAL_DAC_Stop(DAC_HandleTypeDef* hdac, uint32_t Channel);
332 HAL_StatusTypeDef HAL_DAC_Start_DMA(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t* pData, uint32_t Length, uint32_t Alignment);
333 HAL_StatusTypeDef HAL_DAC_Stop_DMA(DAC_HandleTypeDef* hdac, uint32_t Channel);
334 HAL_StatusTypeDef HAL_DAC_Start_Dual_DMA(DAC_HandleTypeDef* hdac, uint32_t Channel, uint32_t* pData, uint32_t Length, uint32_t Alignment);
335
336 uint32_t HAL_DAC_GetValue(DAC_HandleTypeDef* hdac, uint32_t Channel);
```

A hal_dac.c állomány bővítése

```
662 hdac->State = HAL_DAC_STATE_BUSY;
663
664 if(Channel == DAC_CHANNEL_1)
665 {
666     /* Set the DMA transfer complete callback for channel1 */
667     hdac->DMA_Handle1->XferCpltCallback = DAC_DMAConvCpltCh1;
668
669     /* Set the DMA half transfer complete callback for channel1 */
670     hdac->DMA_Handle1->XferHalfCpltCallback = DAC_DMAHalfConvCpltCh1;
671
672     /* Set the DMA error callback for channel1 */
673     hdac->DMA_Handle1->XferErrorCallback = DAC_DMAErrorCh1;
674
675     /* Enable the selected DAC channel1 DMA request */
676     hdac->Instance->CR |= DAC_CR_DMAEN1;
677
678     /* Case of use of channel 1 */
679     switch(Alignment)
680     {
681         case DAC_ALIGN_12B_R:
682             /* Get DHR12R1 address */
683             tmpreg = (uint32_t)&hdac->Instance->DHR12R1;
684             break;
685         case DAC_ALIGN_12B_L:
686             /* Get DHR12L1 address */
687             tmpreg = (uint32_t)&hdac->Instance->DHR12L1;
688             break;
689         case DAC_ALIGN_8B_R:
690             /* Get DHR8R1 address */
691             tmpreg = (uint32_t)&hdac->Instance->DHR8R1;
692             break;
693         default:
694             break;
695     }
696 }
```

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

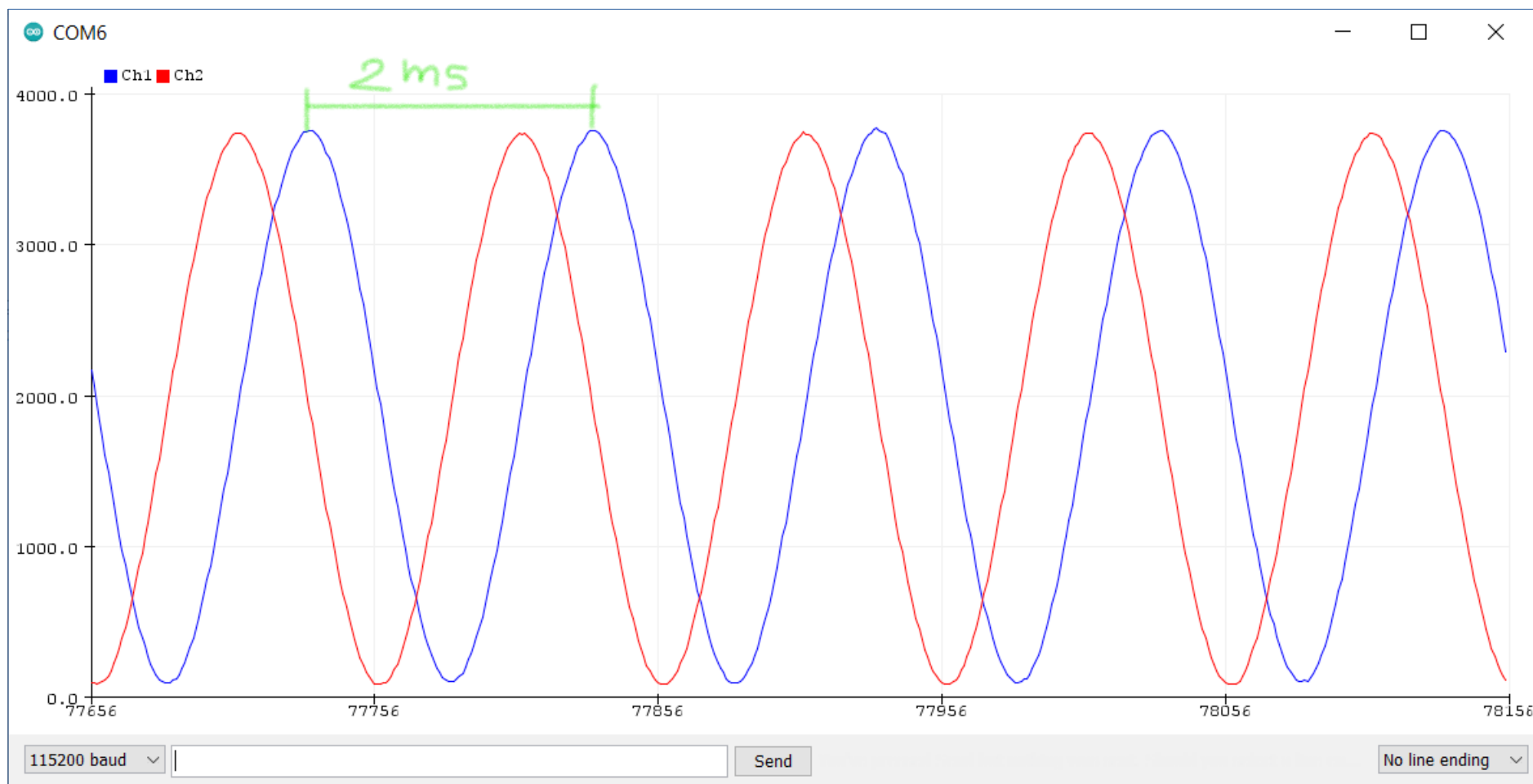
HAL_DAC_Start_Dual_DMA() 2/2.

```
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;  
    }  
}
```

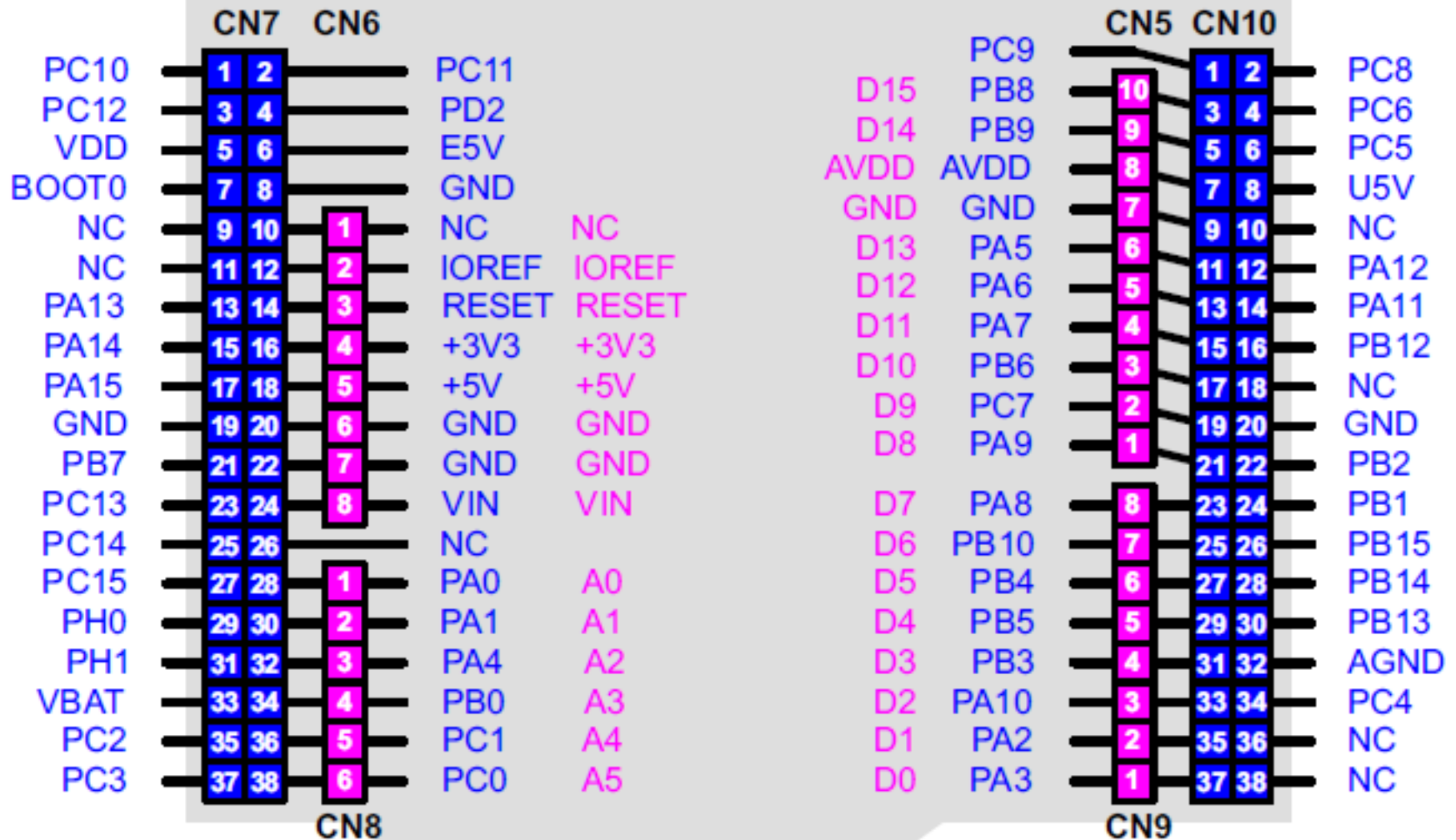
Kicseréltük a
regiszterek
nevét a duál
módú
beíráshoz

F446RE_DAC_audio: futási eredmény

- A kimenőjelet az első példaprogramban ismertetett „kétcsatornás oszcilloszkóp” segítségével ellenőrizhetjük
- A mintavételezés $20\text{ }\mu\text{s}$ -onként történt ($2\text{ ms} / \text{osztás}$)



NUCLEO-F446RE



Arduino

Morpho