STM32 mikrovezérlők programozása STM32CubeIDE környezetben

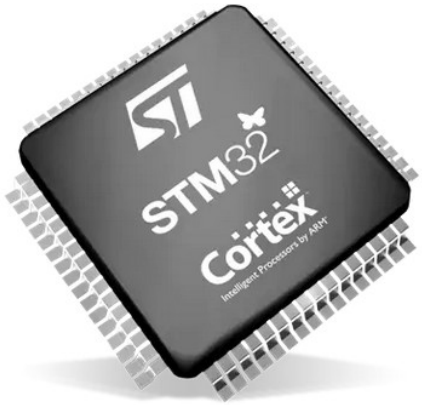
3. Aszinkron soros kommunikáció (UART)

Felhasznált és ajánlott irodalom

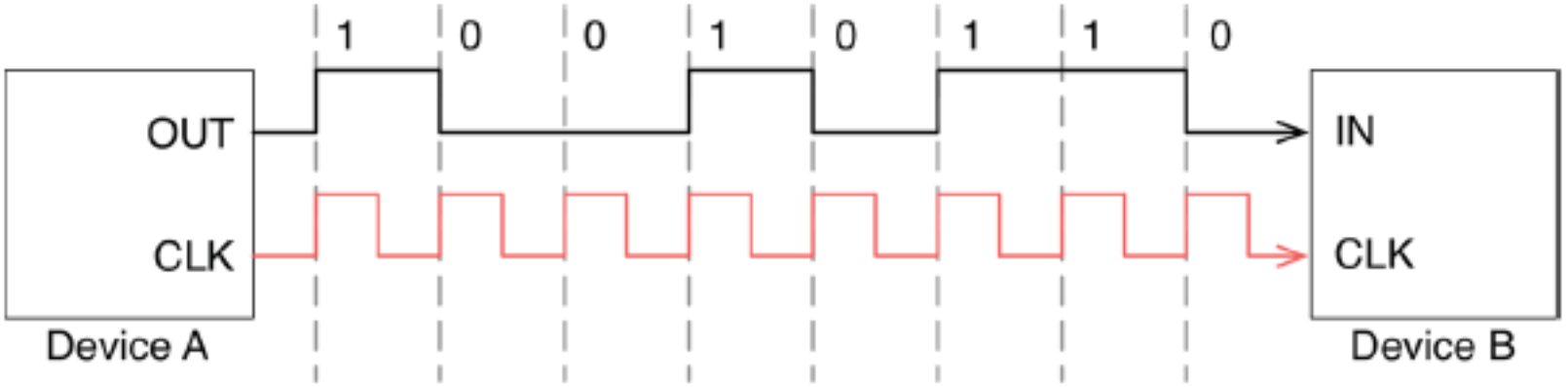
- Joseph Yiu: **The Definitive Guide To The ARM Cortex-M3 and Cortex-M4**
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: **STM32 Arm Programming for Embedded Systems**
- Carmine Noviello: **Mastering STM32**
- A könyv mintapéldái: **github.com/cnoviello/mastering-stm32**
- EMCU: **First embedded program for STM32 mcu using STM32CubeIDE**

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**



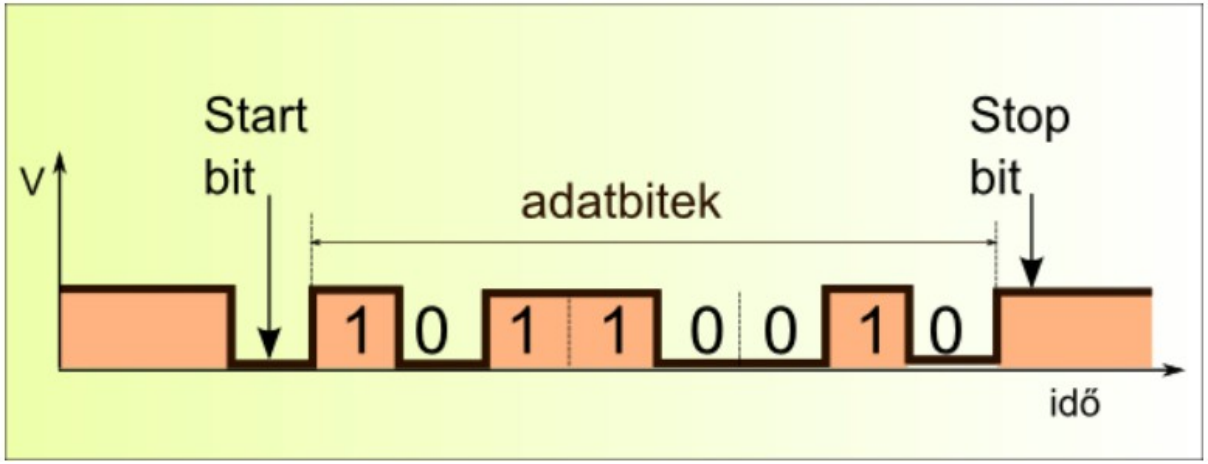
Szinkron soros kommunikáció



Az ábra forrása: Carmine Noviello - Mastering STM32

- Az adó és a vevő közös órajelet használ (az adó szolgáltatja)
- A közös órajel egyúttal azt is jelzi, hogy mikor kell mintavételezni a jelet. Amikor az órajel megjelenik a CLK kimeneten, az jelzi, hogy soros adatküldés kezdődik
- **Szinkron adatküldésnél** az adatküldés sebességét és tartamát az órajel szabja meg. Az órajel frekvenciája meghatározza, hogy mennyi idő alatt tudunk egy adatbitet kiküldeni
- Ha az adó és a vevő ***előzetesen megegyeznek*** az adatküldés sebességében, s hogy mikor kezdődik és végződik az adatbitek mintavételezése, elhagyható az órajelet továbbító vonal – ez az **aszinkron** adatküldés

Aszinkron soros kommunikáció

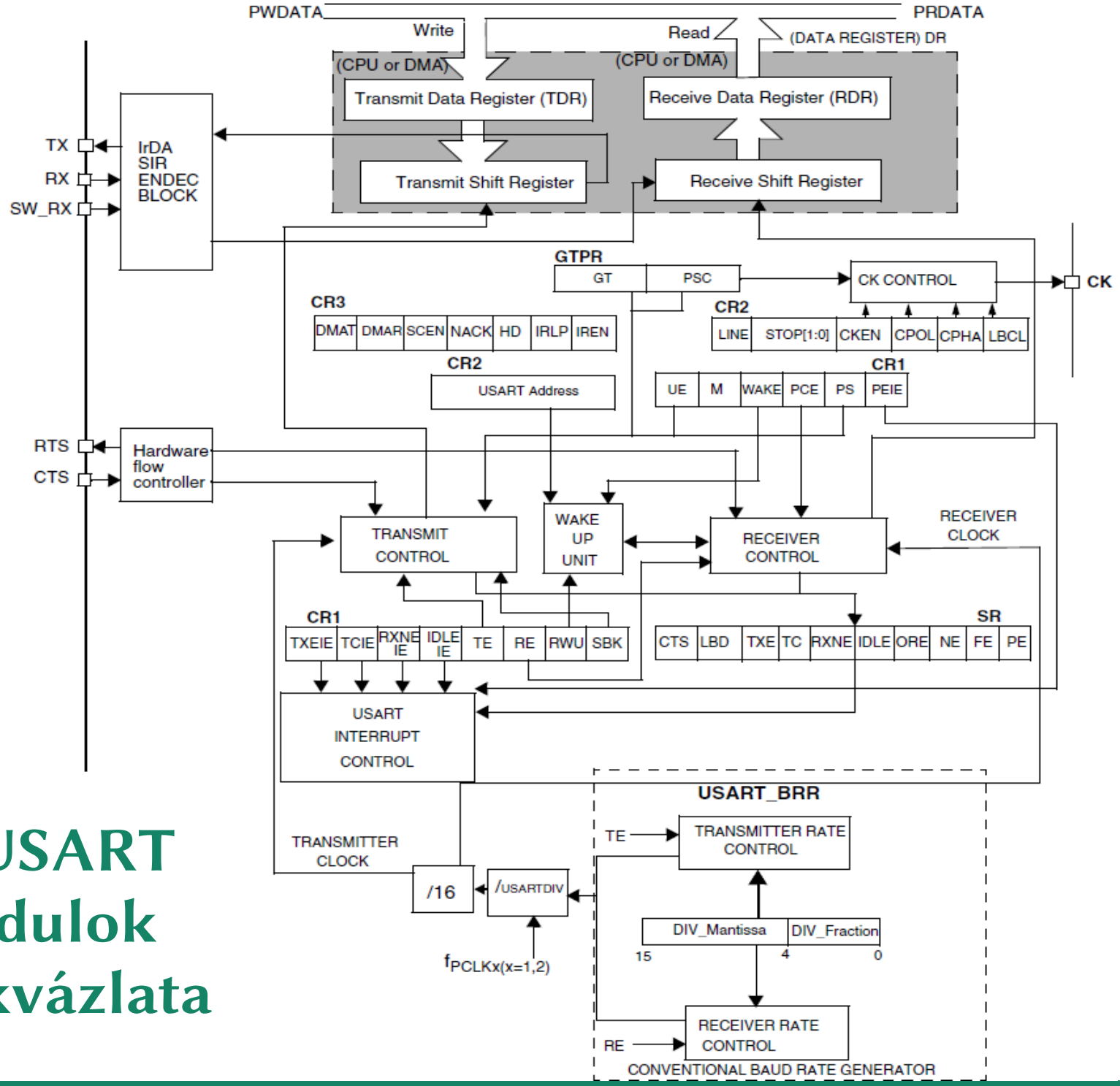


- Nincs szükség órajel továbbításra!
- Az adóban és a vevőben helyileg kell órajelet előállítani
- Az adó egy állandó értékű start bitet kell, hogy küldjön minden adategység előtt az adás kezdetének jelzésére
- A vevő detektálja a start bit homlokélét, és ehhez viszonyítva jelöli ki a mintavételezési időket, az N-edik adatbit számára (T_{bit}*N+1.5) időeltolással
- Stop bitet is használunk az időzítési hibák detektálása érdekében

UART kommunikáció jellemzői

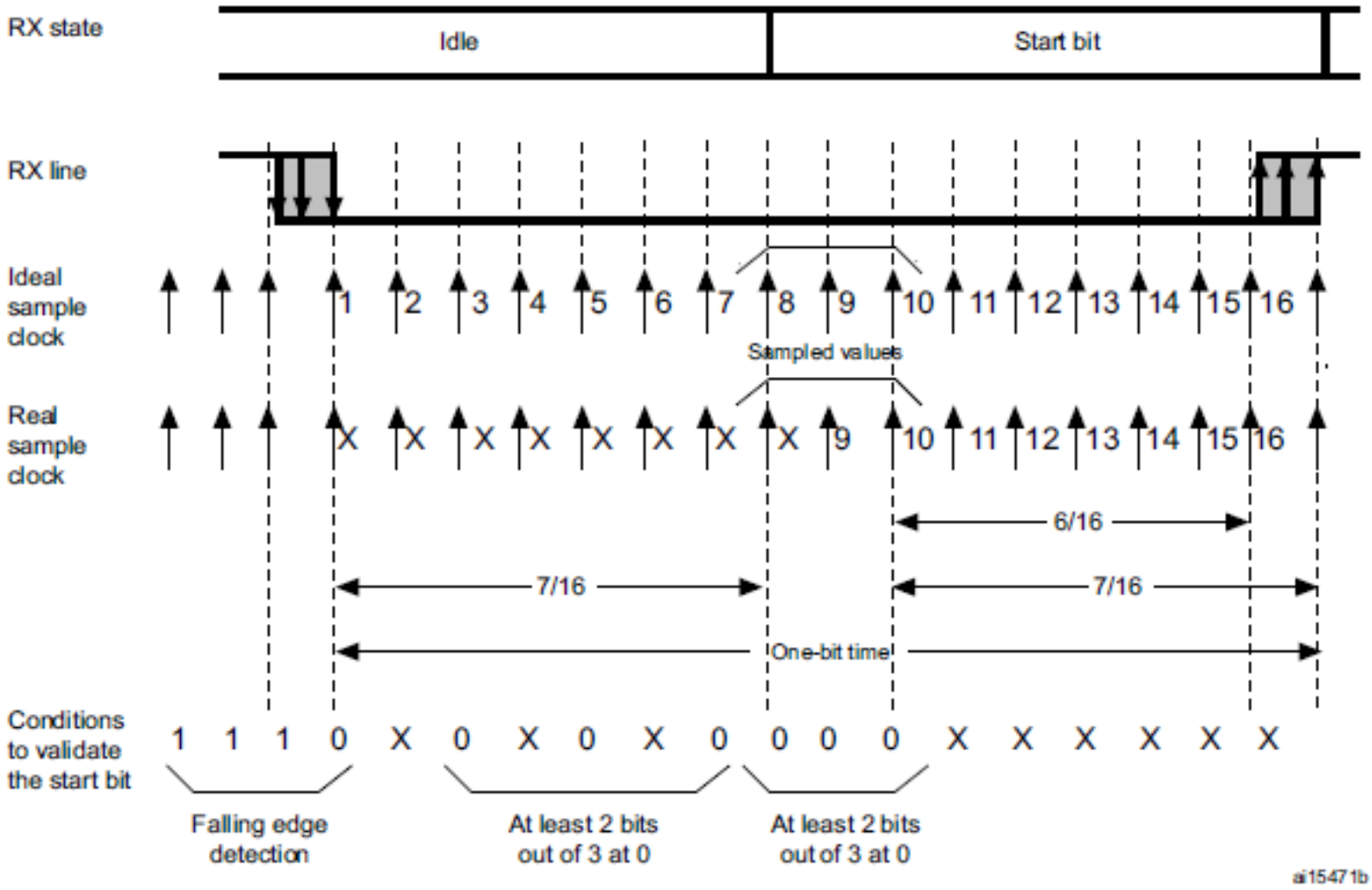
- **Adatkeret:**
  - Start bit (1 bit)
  - Adat (LSB vagy MSB sorrend, adatméret – 7, 8, 9 bit)
  - Opcionális paritásbit használható az adatban szereplő egyesek páros vagy páratlan számának jelzésére.
  - Stop bit (egy vagy két stop bit használható)
- **Az adó és a fogadó fél meg kell, hogy egyezzen:**
  - Kommunikációs sebesség (300 baud, 600, 1200, 2400, 9600, 14 400, 19 200, stb.)
- **Hálózati protokollok üzenetenként további adatokat tartalmazhatnak:**
  - Médium hozzáférés – ha több egység kapcsolódik a buszra, arbitrációra van szükség annak eldöntésére, hogy melyikük küldhet adatot
  - Címzési információ – melyik csomópontnak szól az üzenet?
  - Nagyobb méretű keretezett üzenetcsomag
  - Erősebb hibadetektáláshoz vagy hibajavításhoz szükséges információ (pl. CRC)
  - Kérelem azonnali válaszért

Az USART modulok blokkvázlata



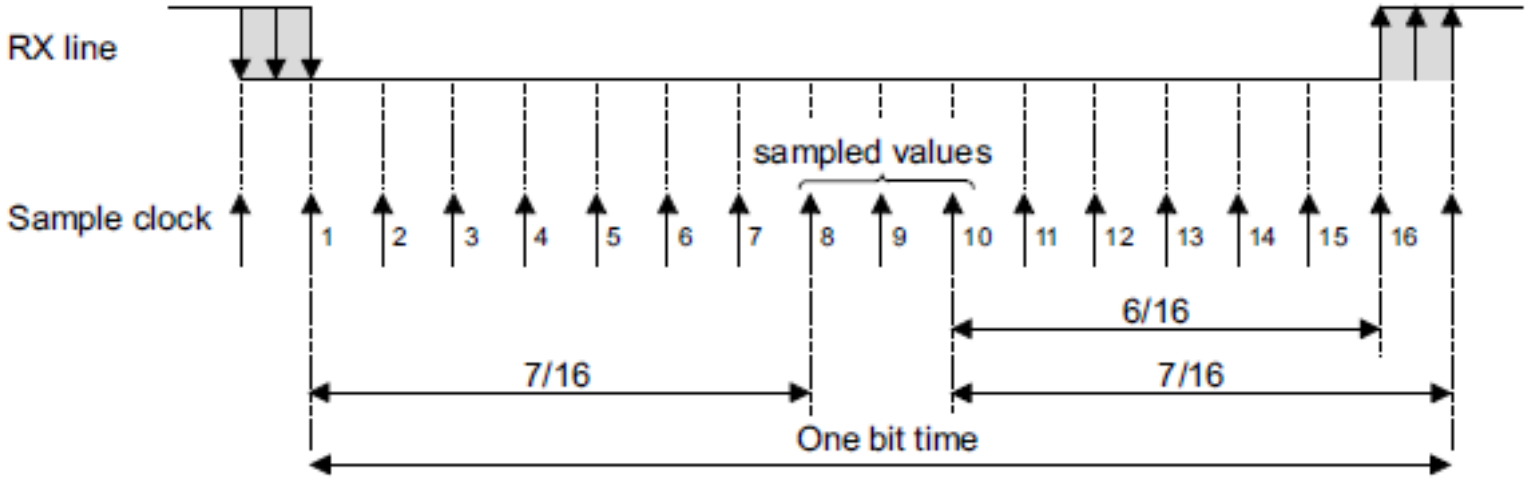
START bit detektálás túlmintavételezéssel

- A mintavételezést többnyire a bitidőhöz tartozó órajelfrekvencia 16-szorosával végezzük



Adatfogadás túlmintavételezéssel

- A mintavételezést az órajel frekvencia 16-szorosával végezzük
- Az *n.* adatbit mintavételezési helye a START bit homlokéléhez képest (n+1)*16 eltolással a 8., 9., és 10. bit
- Az STM32F446 USART*x*_CR3 regiszterének ONEBIT bitje két mód közül választ (STM32F103 esetén csak az első mód létezik):
  - Az adatbit értéke az, ami a három minta többségi értéke (az F jelzőbit bebillen, ha a három minta nem egyezik meg (ONEBIT = 0)
  - Az adatbit értéke az, amit a 9. órajelnél mintavételeztünk (ONEBIT = 1)



STM32F103C8 soros portok jellemzői

- Universal synchronous asynchronous receiver transmitter (USART)
- Full duplex vagy half duplex kétirányú kapcsolat
- 8 vagy 9 bites adathossz, 1 vagy 2 stop bit, paritásbit kezelés
- 10 megszakítási forrás: CTS change, LIN break, Transmit data register empty, Transmission complete, Receive data register full, Idle line, Overrun error, Framing error, Noise error, Parity error)
- Üzemmódok:
  - Aszinkron adatátvitel
  - Szinkron adatátvitel
  - half-duplex egyvezetékes
  - HW adatfolyam-vezérlés
  - multibuffer (DMA)
  - multiprocessor kommunikáció (pl. RS-485, RS-422)
  - SmartCard emuláció (ISO 7816-3)
  - IrDA SIR Encoder/Decoder
  - LIN

Az STM32F103C8 soros portok kivezetései

- Az **STM32F103C8** mikrovezérlő esetén az alacsony lábszám miatt csupán az **USART1**-re korlátozódik az átkonfigurálás lehetősége
- Az **RTSn, CTSn, CKn** kivezetéseket itt nem tüntettük fel, a pin diagramról leolvasható a kiosztásuk
- A beépített bootloader **USART1**-et használja, alapértelmezett hozzárendeléssel (RX=PA10, TX=PA9)

Port	RX	TX	AFIO_MAPR
USART1	**PA10**	**PA9**	USART1_REMAP = 0
	PB7	**PB6**	USART1_REMAP = 1
USART2	**PA3**	**PA2**	USART2_REMAP = 0
	- - -	**- - -**	USART2_REMAP = 1
USART3	**PB11**	**PB10**	USART3_REMAP = 00
	- - -	**- - -**	USART3_REMAP = xx

STM32F446RE soros portok jellemzői

- Az **STM32F446RE** mikrovezérlő 4 db univerzális szinkron/ aszinkron soros adó/vevőt (USART1, USART2, USART3 és USART6), valamint 2 db univerzális aszinkron soros adó/vevőt (UART4 és UART5) tartalmaz

Table 8. USART feature comparison(1)

USART name	Standard features	Modem (RTS/CTS)	LIN	SPI master	irDA	Smartcard (ISO 7816)	Max. baud rate in Mbit/s		APB mapping
							Oversampling by 16	Oversampling by 8	
USART1	X	X	X	X	X	X	5.62	11.25	APB2 (max. 90 MHz)
USART2	X	X	X	X	X	X	2.81	5.62	APB1 (max. 45 MHz)
USART3	X	X	X	X	X	X	2.81	5.62	
UART4	X	X	X	-	X	-	2.81	5.62	
UART5	X	X	X	-	X	-	2.81	5.62	
USART6	X	X	X	X	X	X	5.62	11.25	APB2 (max. 90 MHz)

1. X = feature supported.

STM32F446RE soros port kivezetések

Port	CTS	RTS	TX	RX	CK	AF settings
USART1	PA11	PA12	PA9	PA10	PA8	AF7
			PB6	PB7		AF7
USART2	PA0	PA1	PA2	PA3	PA4	AF7
	PD3	PD4	PD5	PD6	PD7	AF7
USART3	PB13	PB14	PB10	PB11	PB12	AF7
	PD11	PD12	PD8 PC10	PD9 PC11 PC5	PD10 PC12	AF7
UART4	PB0	PA15	PA0	PA1	–	AF8
			PC10	PC11	–	AF8
UART5			PE8	PE7	–	AF8
	PC9	PC8	PC12	PD2	– –	AF8 AF7
USART6	PG13	PG12	PG14	PG9	PG7	AF8
	PG15	PG8	PC6	PC7	PC8	AF8

A NUCLEO-64 kártyák **UART2** portja a PA2, PA3 lábakon át az ST-Link soros portjára csatlakozik

Alternatív funkciók kiválasztása

- **STM32F103C8** esetén a **GPIOx_CRL/H** regiszterben alternatív funkciót, az **AFIO_MAPR** regiszterben alternatív kivezetést választhatunk

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN_REMAP[1:0]		TIM4_REMAP	TIM3_REMAP[1:0]		TIM2_REMAP[1:0]		TIM1_REMAP[1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- **STM32F446RE** esetén a GPIOx_MODER regiszterben alternatív funkciót, a **GPIOx_AFRL/H** regiszterekben alternatív periféria kivezetést (**AF0 – AF15**) választhatunk

GPIOx AFRH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	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
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx AFRL

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	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
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

STM32F446RE alternatív funkciók táblázata *(részlet)*

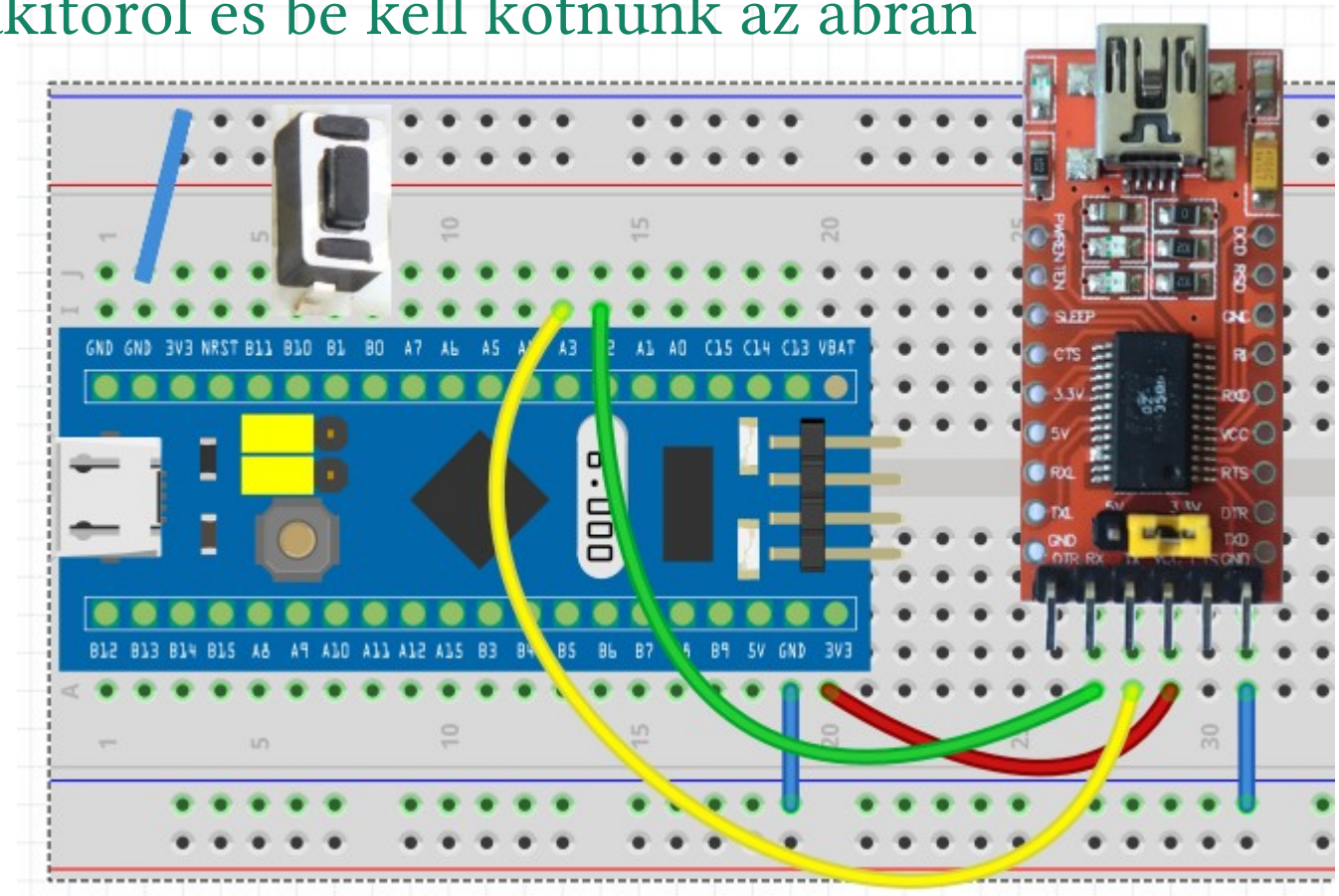
Table 11. Alternate function

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7	AF8	AF9	AF10	AF11	AF12	AF13	AF14	AF15
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10/11/CEC	I2C1/2/3/4/CEC	SPI1/2/3/4	SPI2/3/4/SAI1	SPI2/3/USART1/2/3/UART5/SPDIFRX	SAI/USART6/UART4/5/SPDIFRX	CAN1/2 TIM12/13/14/QUADSPI	SAI2/QUADSPI/OTG2_HS/OTG1_FS	OTG1_FS	FMC/SDIO/OTG2_FS	DCMI	-	SYS
Port A	PA0	-	TIM2_CH1/TIM2_ETR	TIM5_CH1	TIM8_ETR	-	-	-	USART2_CTS	UART4_TX	-	-	-	-	-	-	EVENTOUT
	PA1	-	TIM2_CH2	TIM5_CH2	-	-	-	-	USART2_RTS	UART4_RX	QUADSPI_BK1_IO3	SAI2_MCLK_B	-	-	-	-	EVENTOUT
	PA2	-	TIM2_CH3	TIM5_CH3	TIM9_CH1	-	-	-	USART2_TX	SAI2_SCK_B	-	-	-	-	-	-	EVENTOUT
	PA3	-	TIM2_CH4	TIM5_CH4	TIM9_CH2	-	-	SAI1_FS_A	USART2_RX	-	-	OTG_HS_ULPI_D0	-	-	-	-	EVENTOUT
	PA4	-	-	-	-	-	SPI1_NSS/I2S1_WS	SPI3_NSS/I2S3_WS	USART2_CK	-	-	-	-	OTG_HS_SOF	DCMI_HSYNC	-	EVENTOUT
	PA5	-	TIM2_CH1/TIM2_ETR	-	TIM8_CH1N	-	SPI1_SCK/I2S1_CK	-	-	-	-	OTG_HS_ULPI_CK	-	-	-	-	EVENTOUT
	PA6	-	TIM1_BKIN	TIM3_CH1	TIM8_BKIN	-	SPI1_MISO	I2S2_MCK	-	-	TIM13_CH1	-	-	-	DCMI_PIXCLK	-	EVENTOUT
	PA7	-	TIM1_CH1N	TIM3_CH2	TIM8_CH1N	-	SPI1_MOSI/I2S1_SD	-	-	-	TIM14_CH1	-	-	FMC_SDNWE	-	-	EVENTOUT
	PA8	MCO1	TIM1_CH1	-	-	I2C3_SCL	-	-	USART1_CK	-	-	OTG_FS_SOF	-	-	-	-	EVENTOUT
	PA9	-	TIM1_CH2	-	-	I2C3_SMBA	SPI2_SCK/I2S2_CK	SAI1_SD_B	USART1_TX	-	-	-	-	-	DCMI_D0	-	EVENTOUT
	PA10	-	TIM1_CH3	-	-	-	-	-	USART1_RX	-	-	OTG_FS_ID	-	-	DCMI_D1	-	EVENTOUT
	PA11	-	TIM1_CH4	-	-	-	-	-	USART1_CTS	-	CAN1_RX	OTG_FS_DM	-	-	-	-	EVENTOUT
	PA12	-	TIM1_ETR	-	-	-	-	-	USART1_RTS	SAI2_FS_B	CAN1_TX	OTG_FS_DP	-	-	-	-	EVENTOUT
	PA13	JTMS-SWDIO	-													-	EVENTOUT
	PA14	JTCK-SWCLK	-													-	EVENTOUT

forrás: STM32F446RE adatlap és termékinfo

Kapcsolódás a számítógéphez

- A **NUCLEO-F446RE** kártya esetén az **USART2** port a **PA2, PA3** kivezetésen keresztül alapértelmezetten össze van kötve a kártyára épített **ST-Link** programozó és hibavadász **Rx/Tx** kivezetéseivel
- Az **STM32F103C8** kártya esetén nekünk kell gondoskodni egy USB-UART TTL átalakítóról és be kell kötnünk az ábrán látható módon
- Az átalakító **RX** kivezetése a **PA2**, a **TX** kivezetése pedig a **PA3** lábra (Rx/Tx „keresztbe” kötés)
- A GND és a 3.3 V lábakat közösítsük!



UART HAL firmware library

- Az **STM32Cube HAL** programkönyvtárban szét van választva az **UART** és az **USART** kezelés:
  - Minden **USART** függvény és C típus **HAL_USART** előtaggal kezdődik és az ***stm32xxxx_hal_usart.c*** és ***stm32xxxx_hal_usart.h*** állományokban található
  - Az **UART** kezeléssel kapcsolatos függvények és C típusok pedig a **HAL_UART** előtaggal kezdődnek és az ***stm32xxxx_hal_uart.c*** és ***stm32xxxx_hal_uart.h*** állományokban találhatók
- Mivel mindkét modul hasonló felépítésű, s mivel az **UART** mód használata az elterjedtebb, ezért az alábbiakban csak ezzel foglalkozunk
- Természetesen az **USART** portok képesek **UART** módban is működni

Az UART HAL firmware library használata

- Deklarálni kell egy **UART_HandleTypeDef** típusú struktúrát, pl.
  `UART_HandleTypeDef huart2;  //"fogantyú" a soros porthoz`
- A **HAL_UART_MspInit()** függvény implementálásával inicializálni kell az UART alacsonyszintű erőforrásait:
  - Az érintett **USART** és **GPIO** portok órajelének engedélyezése
  - **GPIO** lábak konfigurálása (Alternate Function push-pull módba)
  - **NVIC** konfigurálása – csak megszakításos mód esetén kell
  - **DMA** konfigurálása – csak DMA mód esetén kell
- Az **UART** port konfigurálása (bitsebesség, adatformátum, üzemmód, stb) megadásával és a **HAL_UART_Init()** függvény meghívásával
- **Megjegyzés:** Az **STM32CubeIDE** környezetből meghívott, vagy a függetlenül használható **STM32Cube MX** grafikus konfiguráló eszköz használata esetén a fenti konfigurációs lépések automatikusan beépülnek a projektünkbe

UART kezelés lekérdezéses (polling) módban

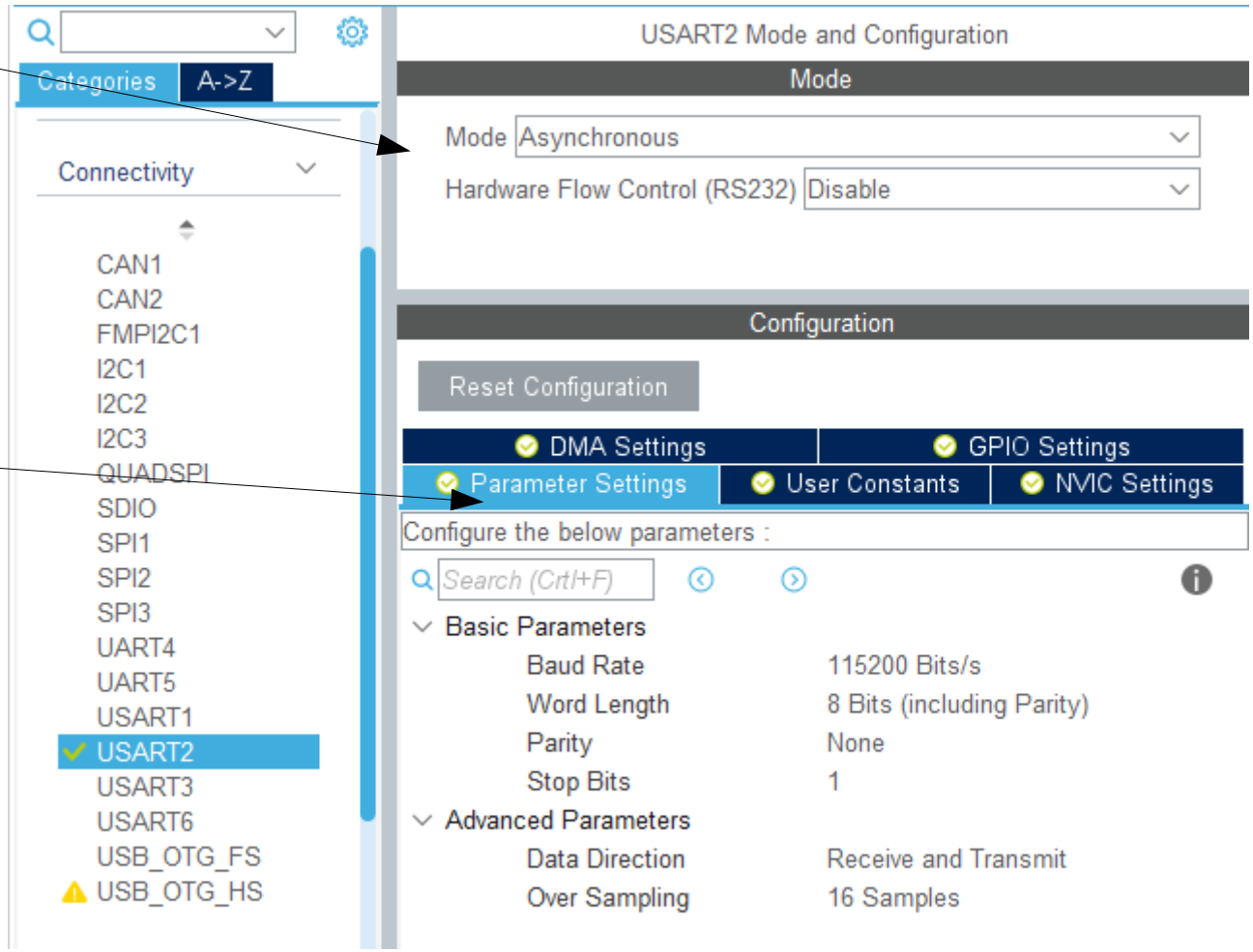
- Első projektünkben (**uart_proba**) az **UART2** soros porton keresztül a beépített **LD1** LED-et (**PA5**) vezéreljük, illetve a szintén beépített **B1** nyomógomb (**PC13**) állapotát kérdezzük le
- A legegyszerűbb, lekérdezéses módot használjuk, amely blokkoló várakozást jelent, amíg a várt darabszámú karakter átvitele be nem fejeződik, vagy amíg a megadott „türelmi idő” le nem jár
- **HAL_UART_Transmit** *(handle, pdata, size, timeout)*
  adatküldés blokkoló várakozással
- **HAL_UART_Receive** *(handle, pdata, size, timeout)*
  adatfogadás blokkoló várakozással

  ahol:

  handle – az UART porthoz definiált „fogantyú”
  pdata – az adattömbre mutató pointer
  size – az adatátvitel mérete bájtokban
  timeout – türelmi időtartam, vagy 0xFFFF FFFF korlátlan

Az UART2 port konfigurálása

- A projekt létrehozását, az órajelek, a LED és a nyomógomb konfigurálását itt nem mutatjuk be, új elemként csak az **USART2** port konfigurálást mutatjuk: Konfigurálásnál a **Connectivity** szekcióban találjuk meg az **USART2** perifériát, amit rákattintással aktiválunk
- Aszinkron üzemmódot használunk
- A hardver adatfolyam-vezérlést kikapcsoltuk (nincs bekötve, tehát nem is használhatnánk)
- A **Parameters** fülön állíthatjuk be a sebességet, adatformátumot, az üzemmódot és a túlmintavételezést
- **Receive and Transmit** kétirányú forgalmat (full duplex) jelent



uart_proba/main.c 3/1

A program forrása: Carmine Noviello: Mastering STM32 (Chapter 8, exc1.c)

```
#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
UART_HandleTypeDef huart2;        // Fogantyú az UART2 porthoz
void SystemClock_Config(void);    // Rendszer órajelek konfigurálása
static void MX_GPIO_Init(void);   // LED és nyomógomb GPIO konfigurálás
static void MX_USART2_UART_Init(void); // USART2 soros port konfigurálás

#define WELCOME_MSG "Welcome to the Nucleo management console\r\n"
#define MAIN_MENU   "Select the option you are interested in:\r\n\t1. Toggle LD2 LED\r\
n\t2. Read USER BUTTON status\r\n\t3. Clear screen and print this message "
#define PROMPT "\r\n> "

void printWelcomeMessage(void) { // Üdvözlő üzenet és a menü kiírása
  HAL_UART_Transmit(&huart2, (uint8_t*)"\033[0;0H", strlen("\033[0;0H"), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t*)"\033[2J", strlen("\033[2J"), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t*)WELCOME_MSG, strlen(WELCOME_MSG), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t*)MAIN_MENU, strlen(MAIN_MENU), HAL_MAX_DELAY);
}

uint8_t readUserInput(void) {     // Prompt kiírása, felhasználói válasz beolvasása
    char readBuf[1];
    HAL_UART_Transmit(&huart2, (uint8_t*)PROMPT, strlen(PROMPT), HAL_MAX_DELAY);
    HAL_UART_Receive(&huart2, (uint8_t*)readBuf, 1, HAL_MAX_DELAY);
    return atoi(readBuf);
}
```

uart_proba/main.c 3/2

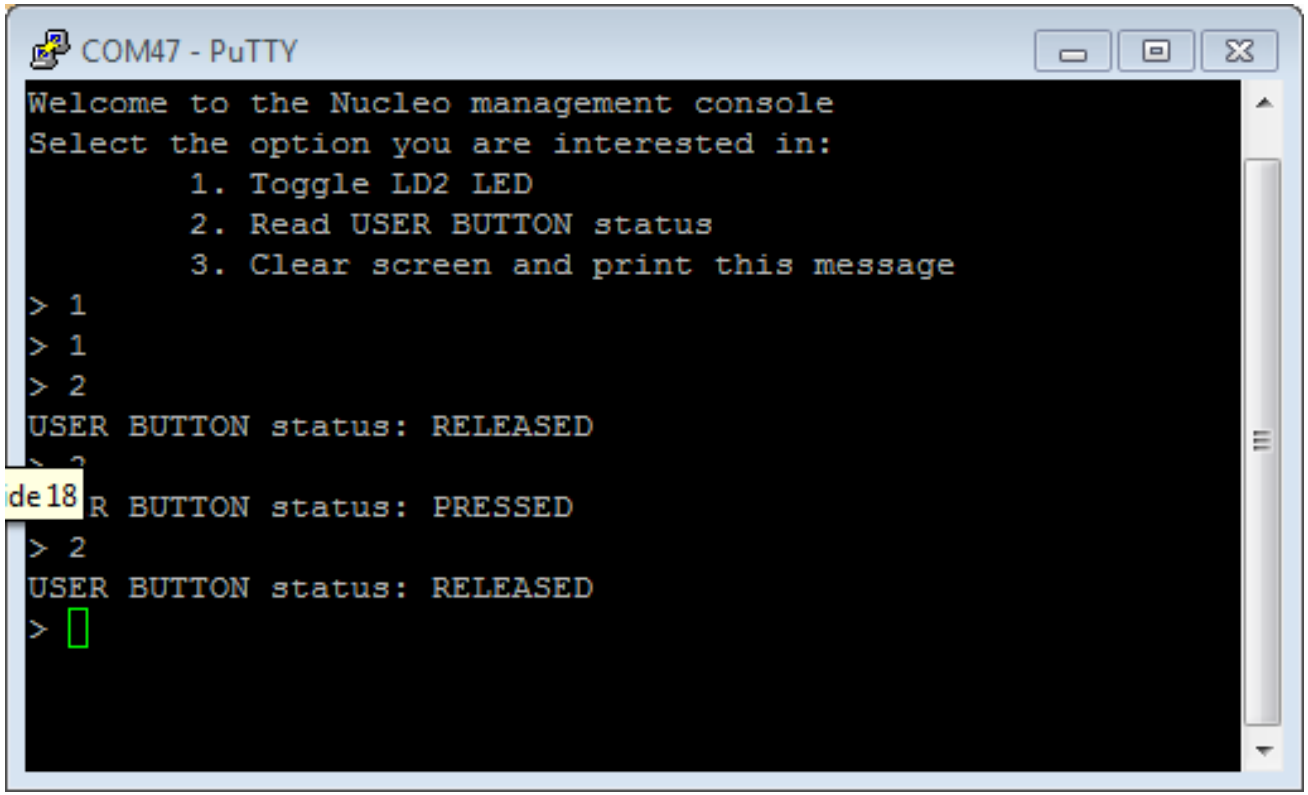
```
uint8_t processUserInput(uint8_t opt) {
  char msg[30];
  if(!opt || opt > 3) return 0;
  sprintf(msg, "%d", opt);
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
  switch(opt) {
    case 1:
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
      break;
    case 2:
      sprintf(msg, "\r\nUSER BUTTON status: %s", HAL_GPIO_ReadPin(GPIOC,
              GPIO_PIN_13) == GPIO_PIN_RESET ? "PRESSED" : "RELEASED");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
      break;
    case 3:
      return 2;
  };
  return 1;
}
```

uart_proba/main.c 3/3

```
int main(void) {
  uint8_t opt = 0;            // Ebbe olvassuk be a felhasználói parancsot
  HAL_Init();                 // HAL inicializálás
  SystemClock_Config();       // Rendszer órajelek beállítása
  MX_GPIO_Init();             // LED és nyomógomb konfigurálása
  MX_USART2_UART_Init();      // UART2 soros port konfigurálása
  PrintMessage:               // Címke a visszaugráshoz
    PrintWelcomeMessage();    // Üdvözlő képernyő és a menü kiírása
  while (1) {
    opt = readUserInput();    // Parancs beolvasása
    processUserInput(opt);    // Parancs feldolgozása
    if(opt == 3)              // 3-as parancs esetén
      goto printMessage;      // visszaugrás az elejére
  }
}
...
```

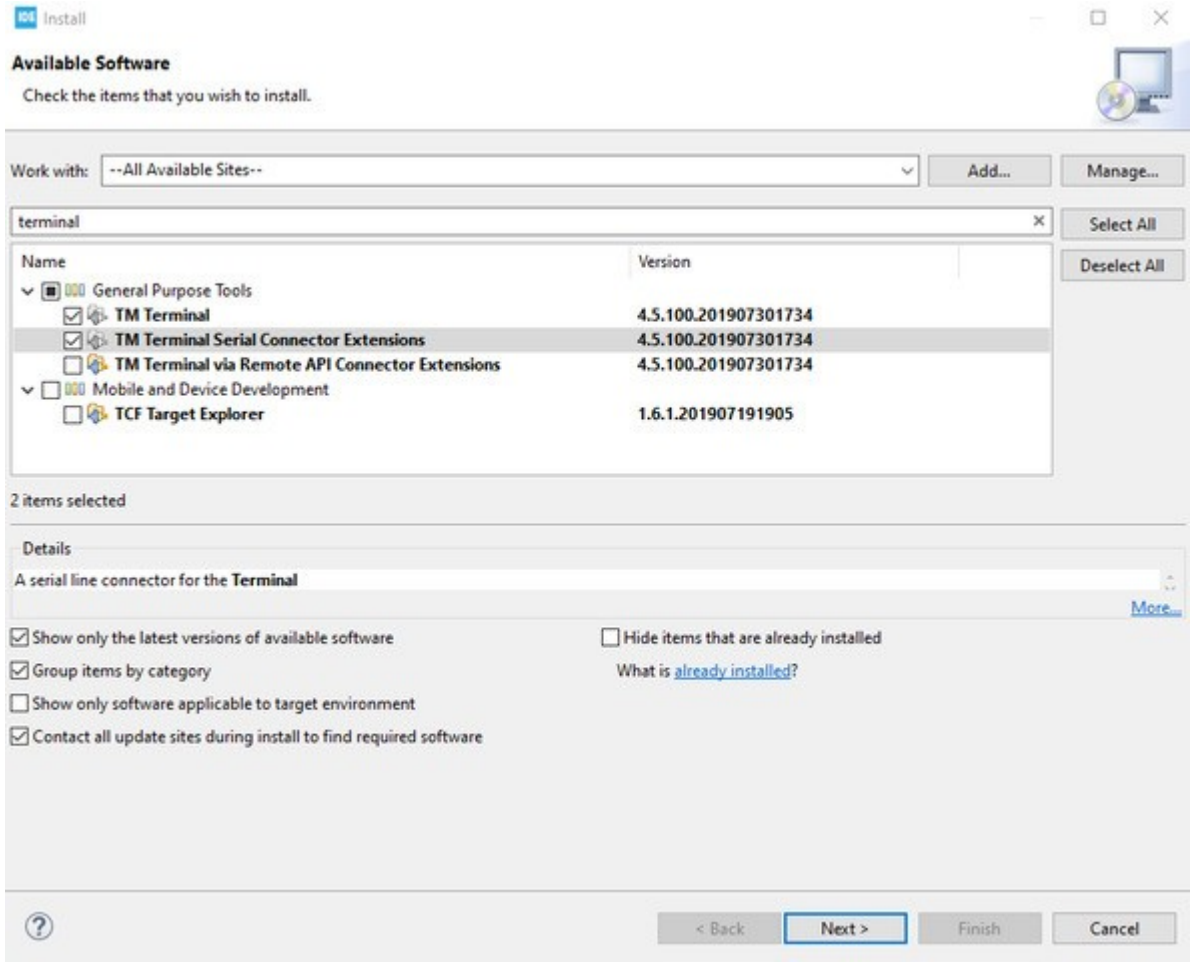
Az uart_proba program futtatása

- A programunkat bármelyik terminél emulátor program, például a **PuTTY.exe** segítségével is futtathatjuk
- Válasszuk ki a virtuális soros portot és állítsuk 115 200 baudra!
- A parancsok **1:** LED állapot átbillentése, **2:** a nyomógomb állapotának kiolvasása, **3:** képernyőtörlés és újrakezdés



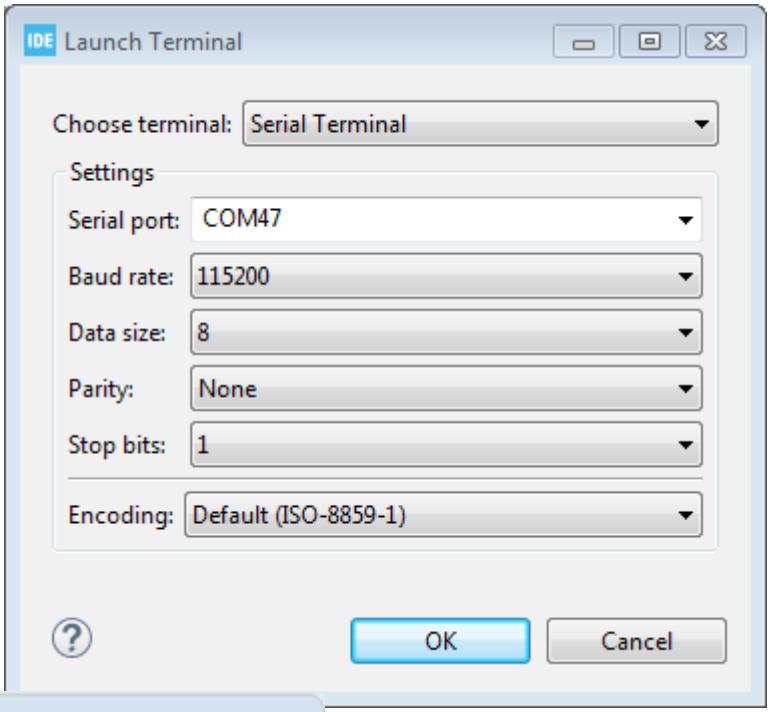
Terminál ablak Eclipse alatt

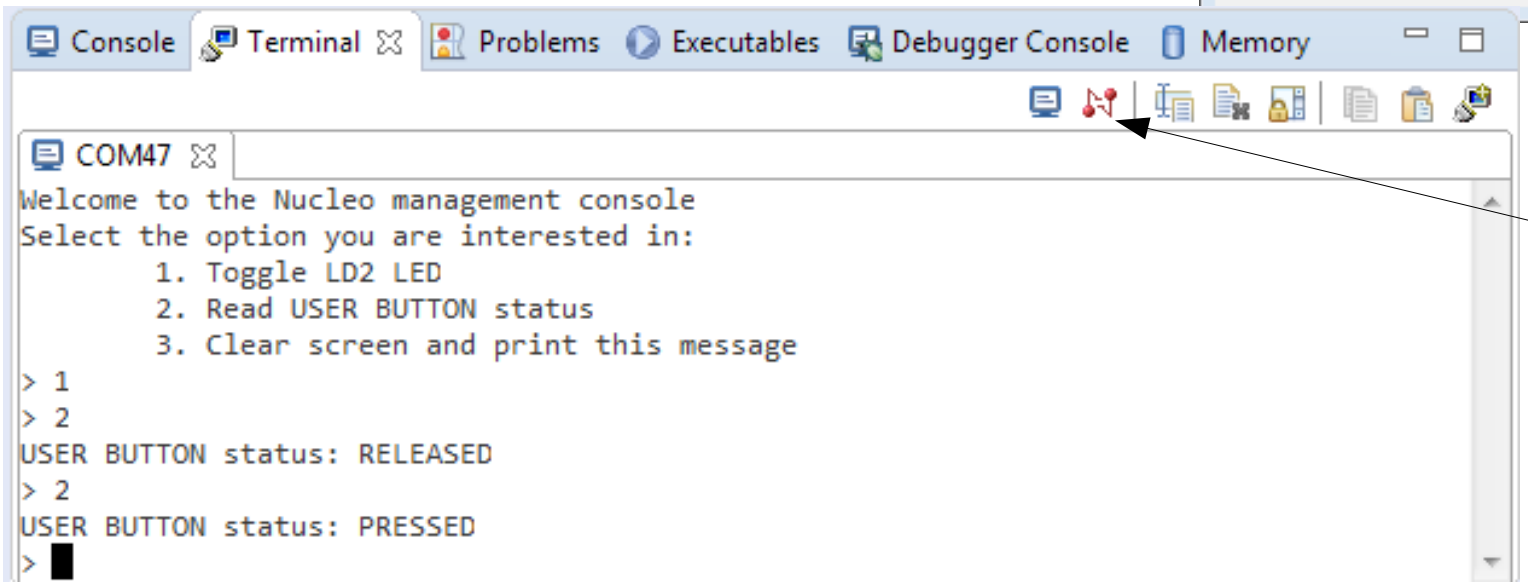
- Az **STM32CubeIDE** fejlesztői környezet az **Eclipse** (Java alapú) keretrendszerre épül, amelynek **TM Terminal** bővítménye lehetővé teszi, hogy egy terminál ablakot nyissunk (soros porti, SSH, Telnet vagy lokális kapcsolattal)
- **A telepítés menete:**
  Help → Install New Software
  Work with: All available sites
  Search: terminal
  Jelöljük ki telepítésre:
  - TM Terminal
  - Serial Connection Extension
  Kattintsunk a NEXT gombra
- Indítsuk újra az **STM32CubeIDE** környezetet!



Terminál ablak Eclipse alatt

- A terminál ablak ***Shift+ Ctrl+Alt+T*** gombnyomásra indul, a felugró ablakban beállíthatjuk a paramétereket
- Válasszuk a Serial Terminal módot!
- Válasszuk ki a soros port sorszámát!
- Állítsunk be 115 200 bps sebességet és 8-bit, No parity, 1 Stop bit módot!





Kapcsolat megszakítása

USART2 konfigurálásának részletei

- Az alábbi inicializáló függvényben szereplő **HAL_UART_Init()** függvény automatikusan meghívja a **HAL_UART_MspInit()** függvényt is *(MSP = MCU support package)*

```
UART_HandleTypeDef huart2;                          // Fogantyú az UART2 porthoz

// Az USART2 port konfigurálása
static void MX_USART2_UART_Init(void) {
  huart2.Instance = USART2;                         // A port kiválasztása
  huart2.Init.BaudRate = 115200;                    // Adatsebesség
  huart2.Init.WordLength = UART_WORDLENGTH_8B;      // Adatformátum: 8 bit
  huart2.Init.StopBits = UART_STOPBITS_1;           // Keretezés: 1 stop bit
  huart2.Init.Parity = UART_PARITY_NONE;            // Nincs paritásbit
  huart2.Init.Mode = UART_MODE_TX_RX;               // Duplex mód (adó/vevő)
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;      // Nincs adatfolyam vezérlés
  huart2.Init.OverSampling=UART_OVERSAMPLING_16;    // 16x túlmintavételezés
  if (HAL_UART_Init(&huart2) != HAL_OK) {           // Az inicilizáló fv. hívása
    Error_Handler();                                // Hibára fut, ha nem sikerül
  }
}
```

USART2 konfigurálásának részletei

- A **HAL_UART_MspInit()** függvény az MCU függő hardver konfigurálást (GPIO, NVIC, DMA) végzi el
- Ez a függvény az **stm32f4xx_hal_msp.c** állományban található

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2) {
    __HAL_RCC_USART2_CLK_ENABLE();      // Peripheral clock enable
    __HAL_RCC_GPIOA_CLK_ENABLE();
    // PA2     ------> USART2_TX
    // PA3     ------> USART2_RX
    GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  }
}
```

UART kezelés megszakítás (interrupt) módban

- Gondoljuk végig, hogy mi történik, ha az előző programunkat egy további feladattal bővítjük, például így:

```
while (1) {
  opt = readUserInput();  // Parancs beolvasása
  processUserInput(opt);  // Parancs feldolgozása
  if(opt == 3)            // 3-as parancs esetén
    goto printMessage;    // visszaugrás az elejére
  PerformCriticalTasks(); // Újabb feladat
}
```

- Ebben az esetben nem blokkolhatjuk a program futását a felhasználói bemenetre várva, mert addig az új feladat nem kerül végrehajtásra
- Az egyik lehetséges megoldás a *timeout* értékének csökkentése, de adatvesztést kockáztathatjuk
- A másik megoldás a **megszakításos kezelés (interrupt)**, amikor a beérkező karakter megszakítja a program futását és lehetőséget kapunk az adat eltárolására – ezt próbájuk ki az **uart_interrupt** projektben

USART porttal kapcsolatos megszakítások

- Az alábbi táblázatban összefoglaltuk az **USART** porttal kapcsolatos megszakításokat, a megszakításjelző, valamint a megszakítást engedélyező regiszterbitek nevét

Interrupt Event	Event Flag	Enable Control Bit
Transmit Data Register Empty	TXE	TXEIE
Clear To Send (CTS) flag	CTS	CTSIE
Transmission Complete	TC	TCIE
Received Data Ready to be Read	RXNE	RXNEIE
Overrun Error Detected	ORE	RXNEIE
Idle Line Detected	IDLE	IDLEIE
Parity Error	PE	PEIE
Break Flag	LBD	LBDIE
Noise Flag, Overrun error and Framing Error in multi buffer communication	NF or ORE or FE	EIE

UART kezelés megszakítás (interrupt) módban

- Második projektünkben (**uart_interrupt**) is a beépített **LD1** LED-et vezéreljük, illetve a szintén beépített **B1** nyomógomb állapotát kérdezzük le, az **UART2** soros porton keresztül
- Most azonban a blokkoló várakozást mellőzve, **megszakításos módban** kezeljük a soros portot (legalábbis vételi oldalon)
- **HAL_UART_Transmit_IT** *(handle, pdata, size)*
  adatküldés blokkoló várakozás nélkül
- **HAL_UART_Receive_IT** *(handle, pdata, size)*
  adatfogadás kezdeményezése, blokkoló várakozás nélkül
- **HAL_UART_RxCpltCallback** *(handle)*
  visszahívási függvény (adatbeérkezés után)

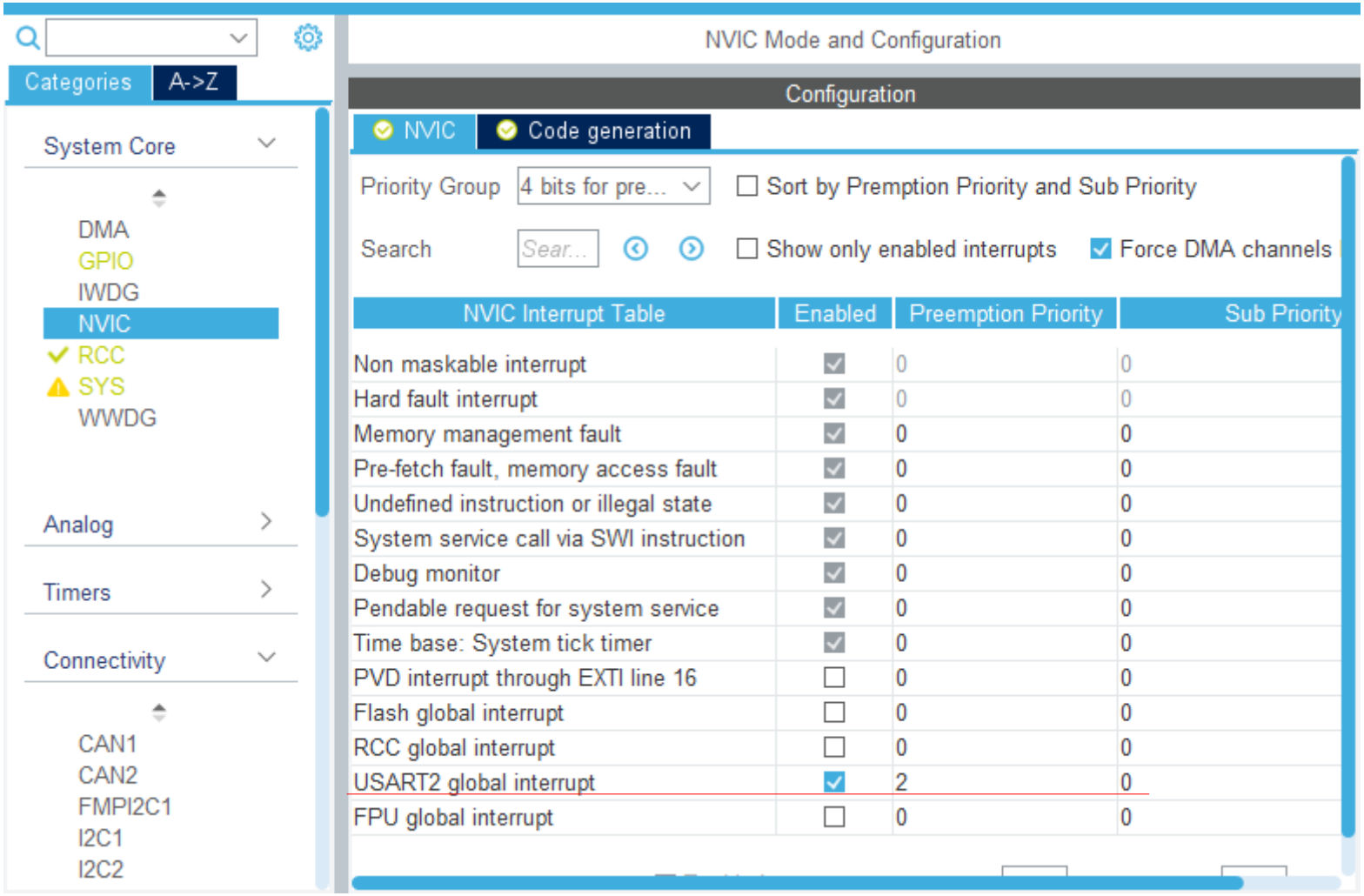
  ahol:
  handle – az UART porthoz definiált „fogantyú”
  pdata – az adattömbre mutató pointer
  size – az adatátvitel mérete bájtokban

UART kezelés megszakítás (interrupt) módban

- A megszakításos kezeléséhez engedélyezni kell az **USART*x*_IRQ*n*** megszakítást és implementálni kell az **USART*x*_IRQHandler()** megszakítást kiszolgáló függvényt (ISR)
- Az **USART*x*_IRQHandler()** belsejében meg kell hívni az **HAL_UART_IRQHandler()** függvényt, amely lekezeli a megszakítást
- Használjuk az adatforgalomhoz a **HAL_UART_Transmit_IT()** és a **HAL_UART_Receive_IT()** függvényeket (ezek automatikusan engedélyezik az UART modul megszakítását is)!
- Az átvitel végéről a **HAL_UART_RxCpltCallback(),** illetve a **HAL_UART_TxCpltCallback()** visszahívási függvények fognak értesíteni bennünket
- Szervezzük át a programot az aszinkron események kezeléséhez!
- Fentiek nagy részéről az **STM32Cube MX** kódgenerátora már gondoskodik, kivéve a visszahívási függvényeket

NVIC konfigurálás

- Konfigurálásnál a **System Core** szekcióban kattintsunk az **NVIC** feliratra és engedélyezzük az **USART2** megszakítását, illetve állítsuk be a megszakítás prioritási szintjét!



Az NVIC konfigurálás hatása

- Az előző oldalon bemutatott **NVIC** konfigurálás hatására a **HAL_UART_MspInit()** függvény két újabb sorral bővül

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(huart->Instance==USART2) {
    __HAL_RCC_USART2_CLK_ENABLE();      // Peripheral clock enable
    __HAL_RCC_GPIOA_CLK_ENABLE();
    // PA2 ------> USART2_TX,  PA3  ------> USART2_RX
    GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    /* USART2 interrupt Init */
    HAL_NVIC_SetPriority(USART2_IRQn, 2, 0);  // Prioritás beállítása
    HAL_NVIC_EnableIRQ(USART2_IRQn);          // Megszakítás engedélyezése
  }
}
```

Az UART2 megszakítások kiszolgálása

- Az **STM32Cube MX** által generált kód **STM32F4xx_it.c** állománya tartalmazza megszakítások alapértelmezett kiszolgáló függvényeit, így az **USART2_IRQHandler()** is itt található

```
extern UART_HandleTypeDef huart2;

void USART2_IRQHandler(void) {
  HAL_UART_IRQHandler(&huart2);
}
```

- A meghívott **HAL_UART_IRQHandler()** függvény a **HAL** részeként teljeskörűen lekezeli a megszakítást és folytatja a megszakítást korábban engedélyező **HAL_UART_Transmit_IT()**, vagy **HAL_UART_Receive_IT()** függvényhívásban előírt feladat végrehajtását (kiírás, vagy beolvasás), átvitel végén pedig aktiválja a megfelelő visszahívási függvényt

UART megszakítás visszahívási függvényei

- **HAL_UART_TxCpltCallback** *(huart)* – kiírás vége visszahívási függvény. Akkor aktiválódik, amikor az adott hosszúságú karakterfüzér kiírása véget ért
- **HAL_UART_RxCpltCallback** *(huart)* – beolvasás vége visszahívási függvény. Akkor aktiválódik, amikor az adott hosszúságú karakterfüzér beolvasása véget ért
- **HAL_UART_ErrorCallback** *(huart)* – hibajelző visszahívási függvény. Akkor aktiválódik, amikor az átvitel során valamilyen hiba keletkezik a hiba kódját a `huart->ErrorCode` változóból olvashatjuk ki
- A visszahívási függvényeknek egy-egy üres változatát a HAL tartalmazza, de ezeket felüldefiniálhatjuk, ha szükségünk van valamelyikre. Például:

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle) {
 UartReady = SET;
}
```

UART hibakezelés

- A **HAL_UART** programkönyvtár fel van készítve a hibák detektálására és figyelmeztet, ha hiba történt
- Csupán a **HAL_UART_ErrorCallback()** függvényt kell implementálnunk az alkalmazásban.
- Azt hogy milyen hiba történt, a **huart->ErrorCode** kiolvasásával tudhatjuk meg. A lehetséges hibakódokat az alábbi táblázat mutatja

UART Error Code	Description
HAL_UART_ERROR_NONE	No error occurred
HAL_UART_ERROR_PE	Parity check error
HAL_UART_ERROR_NE	Noise error
HAL_UART_ERROR_FE	Framing error
HAL_UART_ERROR_ORE	Overrun error
HAL_UART_ERROR_DMA	DMA Transfer error

uart_interrupt/main.c 4/1

A program forrása: Carmine Noviello: Mastering STM32 (Chapter 8, exc2.c)

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

UART_HandleTypeDef huart2;         // Fogantyú az UART2 porthoz
char readBuf[1];                   // Bemeneti buffer
__IO ITStatus UartReady = SET;     // Megszakítási állapotjelző

void SystemClock_Config(void);     // Rendszer órajelek konfigurálása
static void MX_GPIO_Init(void);    // LED és nyomógomb GPIO konfigurálás
static void MX_USART2_UART_Init(void); // USART2 soros port konfigurálás

#define WELCOME_MSG "Welcome to the Nucleo management console\r\n"
#define MAIN_MENU   "Select the option you are interested in:\r\n\t1. Toggle LD2 LED\r\
n\t2. Read USER BUTTON status\r\n\t3. Clear screen and print this message "
#define PROMPT "\r\n> "

void printWelcomeMessage(void) {
  char *strings[] = {"\033[0;0H", "\033[2J", WELCOME_MSG, MAIN_MENU, PROMPT};
  for (uint8_t i = 0; i < 5; i++) {
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)strings[i], strlen(strings[i]));
    while (HAL_UART_GetState(&huart2) == HAL_UART_STATE_BUSY_TX ||
           HAL_UART_GetState(&huart2) == HAL_UART_STATE_BUSY_TX_RX);
  }
}
```

Megszakítással sem könnyű az élet, ellenőrizni kell, hogy befejeződött-e már a kiírás. (Ez itt blokkoló várakozás)

uart_interrupt/main.c 4/2

```
int8_t readUserInput(void) {
  int8_t retVal = -1;

  if(UartReady == SET) {
    UartReady = RESET;
    HAL_UART_Receive_IT(&huart2, (uint8_t*)readBuf, 1);
    retVal = atoi(readBuf);
  }
  return retVal;
}
/* Vevőoldali visszahívási függvény, az átvitel végén hívódik meg */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle) {
 /* Set transmission flag: transfer complete*/
 UartReady = SET;    // Jelzőbit beállítása, ha befejeződött a vétel
}

void performCriticalTasks(void) {
  HAL_Delay(100);
}
```

uart_interrupt/main.c 4/3

```
uint8_t processUserInput(uint8_t opt) {
  char msg[30];
  if(!opt || opt > 3) return 0;
  sprintf(msg, "%d", opt);
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
  switch(opt) {
    case 1:
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
      break;
    case 2:
      sprintf(msg, "\r\nUSER BUTTON status: %s", HAL_GPIO_ReadPin(GPIOC,
              GPIO_PIN_13) == GPIO_PIN_RESET ? "PRESSED" : "RELEASED");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
      break;
    case 3:
      return 2;
  }
  HAL_UART_Transmit(&huart2,(uint8_t*)PROMPT,strlen(PROMPT), HAL_MAX_DELAY);
  return 1;
}
```

Mivel a kiírás kiszámíthatóan gyors, nem volt indokolt a megszakításos kezeléssel bonyolítani a programot.

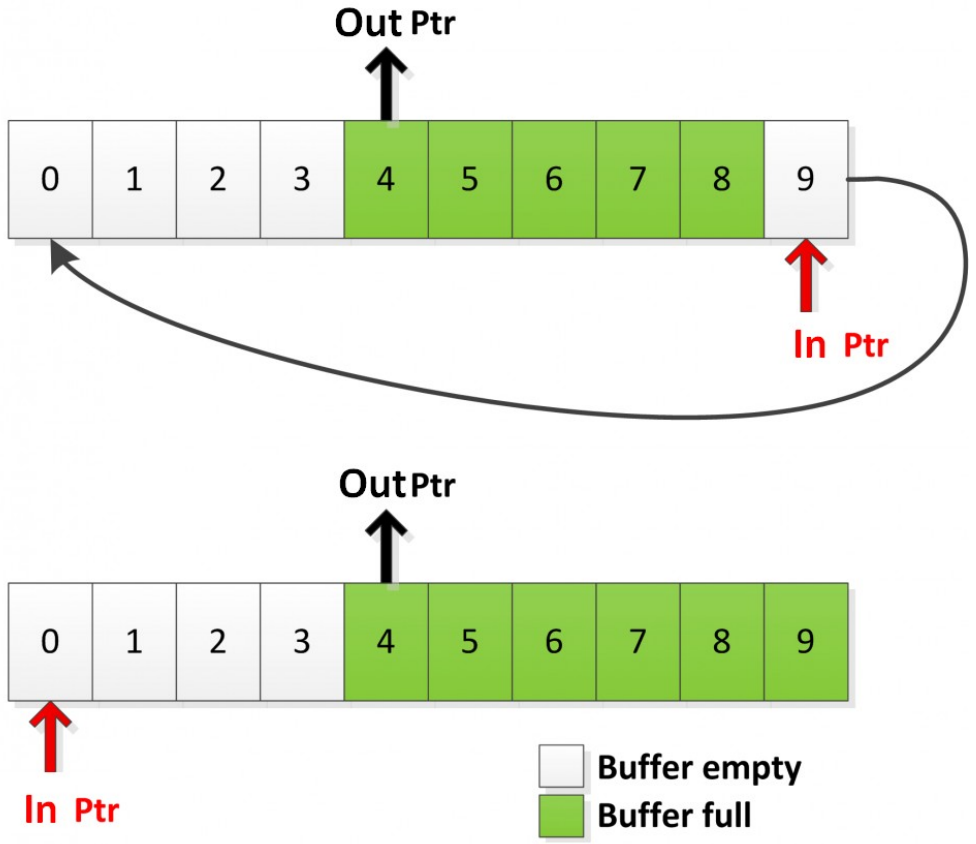
uart_interrupt/main.c 4/4

```
int main(void) {
  uint8_t opt = 0;              // Ebbe olvassuk be a felhasználói parancsot
  HAL_Init();                   // HAL inicializálás
  SystemClock_Config();         // Rendszer órajelek beállítása
  MX_GPIO_Init();               // LED és nyomógomb konfigurálása
  MX_USART2_UART_Init();        // UART2 soros port konfigurálása
  PrintMessage:                 // Címke a visszaugráshoz
    PrintWelcomeMessage();      // Üdvözlő képernyő és a menü kiírása
  while (1) {
    opt = readUserInput();      // Parancs beolvasása
    if(opt > 0) {               // Nincs beérkezett parancs ha opt = -1
      processUserInput(opt);// Parancs feldolgozása
      if(opt == 3)              // 3-as parancs esetén
        goto printMessage;      // visszaugrás az elejére
    }
    PerformCriticalTasks(); // Új feladat, amely nem tűr halasztást
  }
}
...
```

Ez a program is ugyanazt az eredményt produkálja, mint az előző. A különbség csak az, hogy itt a **PerformCriticalTasks()** függvény is lefut minden ciklusban.

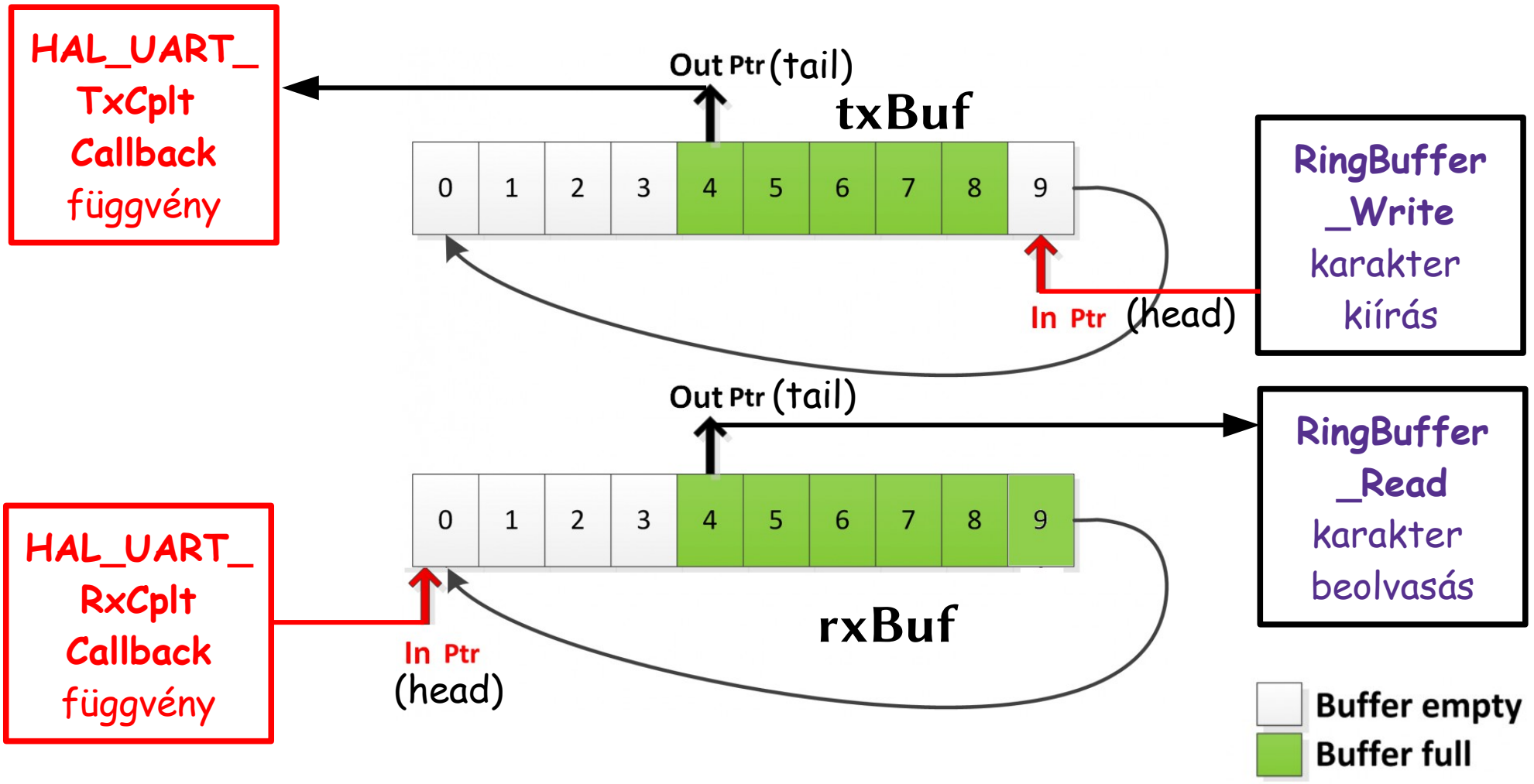
Megszakításvezérelt körkörös bufferelés

- Amikor az **UART** egy karaktert fogad, a főprogram nem feltétlenül tudja azonnal feldolgozni ⟶ ideiglenesen el kell tárolni
- Amikor a főprogram karaktereket küld, ne várjuk meg azok soros kiléptetését az **UART** kimenetén ⟶ ideiglenesen el kell tárolni
- A körkörös buffer egy FIFO tár, a karaktereket a beérkezés sorrendjében vesszük elő
- A beérkező karaktereket az ***In*** mutatóval jelezett helyre tesszük, s a mutatót léptetjük
- A felhasználásnál az ***Out*** mutatóval jelzett helyről vesszük elő az adatot, s a mutatót léptetjük



Adó és vevőoldali bufferelés

- Kétirányú kommunikációhoz két bufferre lesz szükségünk: egy az UART Tx adó, egy pedig az UART Rx vevő számára



UART bufferelt átvitel gyűrűs tárral

- Harmadik projektünkben (**uart_ringbuffer**) a körkörös FIFO tár egyszerű implementációja ***Carmine Noviello: Mastering STM32*** könyvének mintapéldájából való, s külön forrásállományban található (*ringbuffer.c, ringbuffer.h*)
- A bemenetnél, mivel csak 1-1 karakteres parancsokat várunk, nem használjuk a FIFO tárat, de a kimeneti bufferelés mintájára könnyen átírható lenne
- Kiírásnál (**UART_Transmit()** függvény) először megpróbáljuk a kiíratást a **HAL_UART_Transmit_IT()** függvénnyel, s ha ez nem sikerül, akkor a körkörös FIFO tárban helyezzük el a kiírandó karaktereket a **RingBuffer_Write()** fügvényhívással
- A **HAL_UART_Transmit_IT()** függvénnyel indított kiírás végén, a **HAL_UART_TxCpltCallback()** visszahívási függvényben pedig ellenőrizzük, hogy van-e kiírandó karakter a FIFO tárban, s ha van, akkor kiíratjuk egy újabb **HAL_UART_Transmit_IT()** hívással

uart_ringbuffer/main.c 4/1

A program forrása: Carmine Noviello: Mastering STM32 (Chapter 8, exc3.c)

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

UART_HandleTypeDef huart2;        // Fogantyú az UART2 porthoz
char readBuf[1];                  // Bemeneti buffer
__IO ITStatus UartReady = SET;    // Megszakítási állapotjelző

uint8_t txData;
RingBuffer txBuf, rxBuf;          // Körkörös FIFO tárak


#define WELCOME_MSG "Welcome to the Nucleo management console\r\n"
#define MAIN_MENU   "Select the option you are interested in:\r\n\t1. Toggle LD2 LED\r\
n\t2. Read USER BUTTON status\r\n\t3. Clear screen and print this message "
#define PROMPT "\r\n> "

void printWelcomeMessage(void) {
  char *strings[] = {"\033[0;0H", "\033[2J", WELCOME_MSG, MAIN_MENU, PROMPT};
  for (uint8_t i = 0; i < 5; i++) {
    HAL_UART_Transmit_IT(&huart2, (uint8_t*)strings[i], strlen(strings[i]));
    while (HAL_UART_GetState(&huart2) == HAL_UART_STATE_BUSY_TX ||
           HAL_UART_GetState(&huart2) == HAL_UART_STATE_BUSY_TX_RX);
  }
}
```

uart_ringbuffer/main.c 4/2

```
uint8_t UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t
len) {
  if(HAL_UART_Transmit_IT(huart, pData, len) != HAL_OK) {
    if(RingBuffer_Write(&txBuf, pData, len) != RING_BUFFER_OK)
      return 0;
  }
  return 1;
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
  if(RingBuffer_GetDataLength(&txBuf) > 0) {  // Ha RingBuffer nem üres
    RingBuffer_Read(&txBuf, &txData, 1);      // akkor előveszünk egy bájtot
    HAL_UART_Transmit_IT(huart, &txData, 1);  // és kiküldjük soros porton
  }
}

void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart) {
  if(huart->ErrorCode == HAL_UART_ERROR_ORE)  // Ha ráfutás történt, akkor
    HAL_UART_Receive_IT(huart, (uint8_t *)readBuf, 1); // új beolvasás indul
}
```

uart_ringbuffer/main.c 4/3

```
int8_t readUserInput(void) {
  int8_t retVal = -1;
  if(UartReady == SET) {
    UartReady = RESET;
    HAL_UART_Receive_IT(&huart2, (uint8_t*)readBuf, 1);
    retVal = atoi(readBuf);
  }
  return retVal;
}

/* Vevőoldali visszahívási függvény, az átvitel végén hívódik meg */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *UartHandle) {
 /* Set transmission flag: transfer complete*/
 UartReady = SET;    // Jelzőbit beállítása, ha befejeződött a vétel
}

void performCriticalTasks(void) {
  HAL_Delay(100);
}
```

Ezek a függvények megegyeznek az előző programban bemutatottakkal

uart_ringbuffer/main.c 4/4

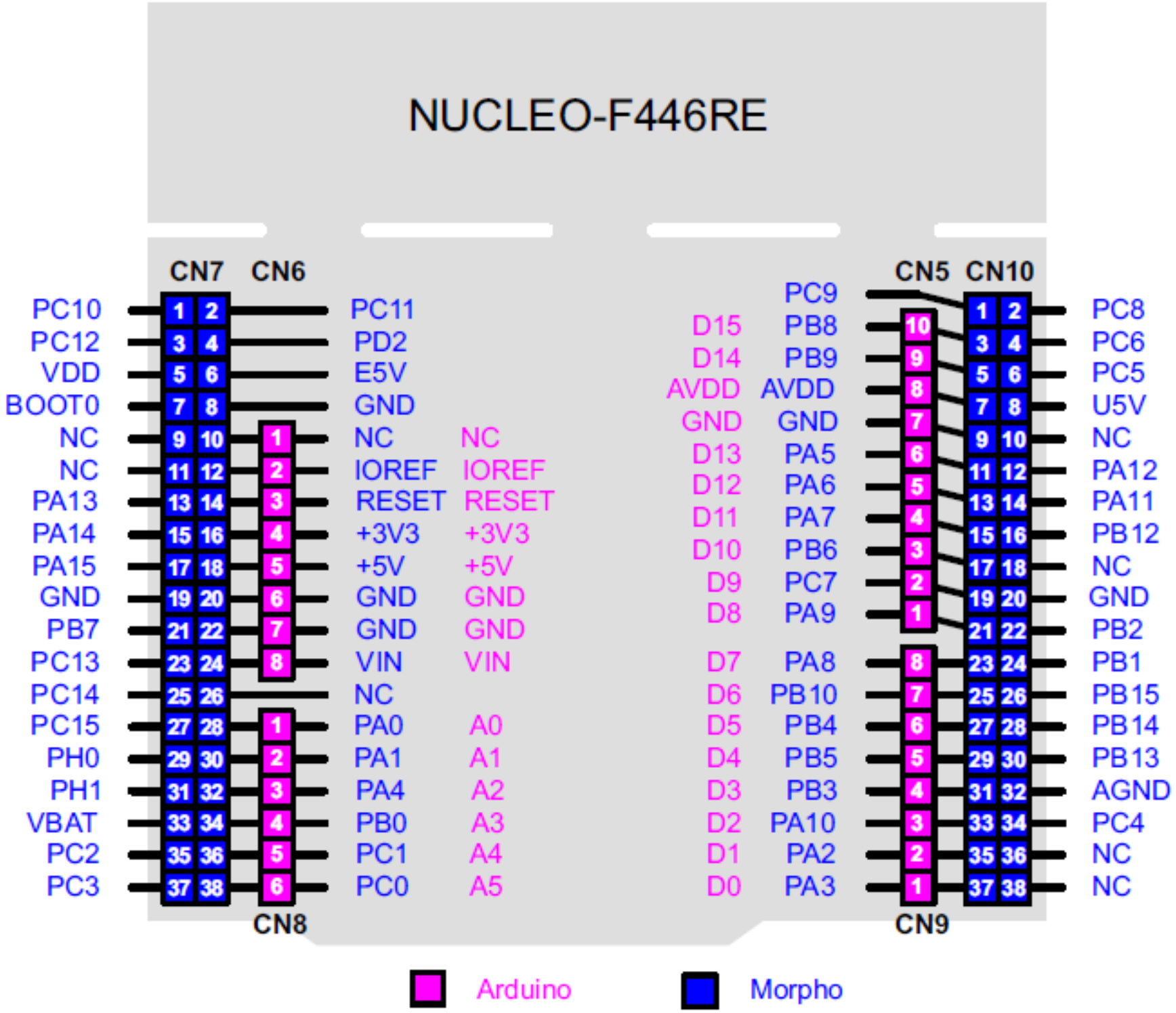
```
uint8_t processUserInput(uint8_t opt) {
  char msg[30];
  if(!opt || opt > 3) return 0;
  sprintf(msg, "%d", opt);
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
  switch(opt) {
    case 1:
      HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
      break;
    case 2:
      sprintf(msg, "\r\nUSER BUTTON status: %s", HAL_GPIO_ReadPin(GPIOC,
              GPIO_PIN_13) == GPIO_PIN_RESET ? "PRESSED" : "RELEASED");
      HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
      break;
    case 3:
      return 2;
  };
  HAL_UART_Transmit(&huart2,(uint8_t*)PROMPT,strlen(PROMPT), HAL_MAX_DELAY);
  return 1;
}
```

Ez a függvény megegyezik az előző programban találhatóval.

uart_interrupt/main.c 4/5

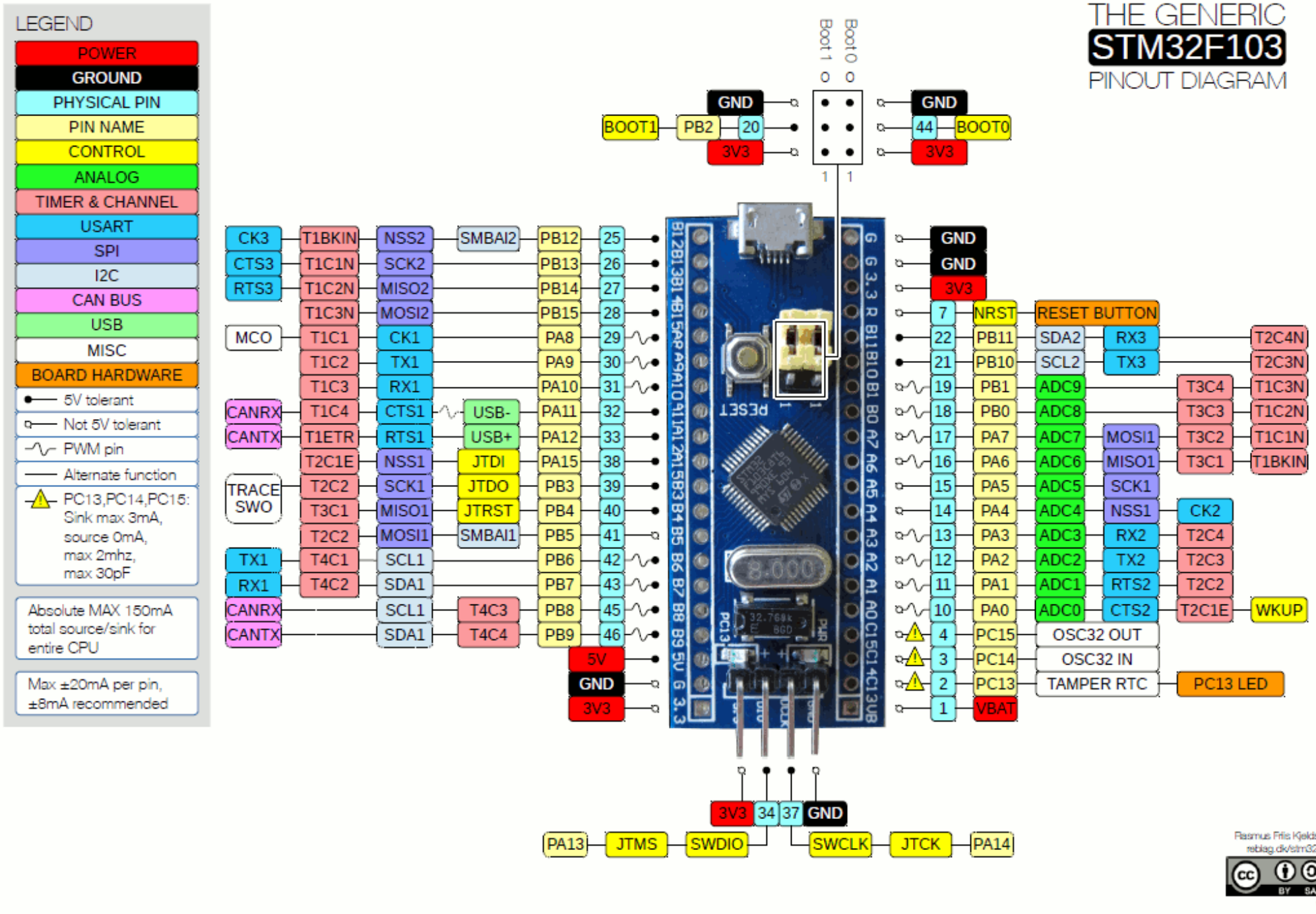
```
int main(void) {
  uint8_t opt = 0;              // Ebbe olvassuk be a felhasználói parancsot
  HAL_Init();                   // HAL inicializálás
  SystemClock_Config();         // Rendszer órajelek beállítása
  MX_GPIO_Init();               // LED és nyomógomb konfigurálása
  MX_USART2_UART_Init();        // UART2 soros port konfigurálása
  PrintMessage:                 // Címke a visszaugráshoz
    PrintWelcomeMessage();      // Üdvözlő képernyő és a menü kiírása
  while (1) {
    opt = readUserInput();      // Parancs beolvasása
    if(opt > 0) {               // Nincs beérkezett parancs ha opt = -1
      processUserInput(opt);// Parancs feldolgozása
      if(opt == 3)              // 3-as parancs esetén
        goto printMessage;      // visszaugrás az elejére
    }
    PerformCriticalTasks(); // Új feladat, amely nem tűr halasztást
  }
}
...
```

Ez a program is ugyanazt az eredményt produkálja, mint az előzőek.
A különbség a megvalósítás módjában van.


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

Lab03 projektek STM32F103C8-ra

- Az **STM32F103C8** mikrovezérlőre (Blue pill) is adaptáltuk az előadás mintapéldáit. Itt csak a hardver eltéréseket soroljuk fel.
- Az UART2 soros portot (**PA2:** Tx. **PA3:** Rx) egy USB-UART átalakítón keresztül kötjük a számítógéphez (a kapcsolási elrendezés a 15. oldalon látható)
- A beépített LED (**PC13**) lehúzásra világít, de ennek most nincs sok jelentősége, mert csak állapotváltó parancsot használunk
- A nyomógombot **PB10** és GND közé kötöttük, s az egyszerűség kedvéért a bemenetet belső felhúzással konfiguráltuk
- A projektek nevei **F103_** előtagot kaptak


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