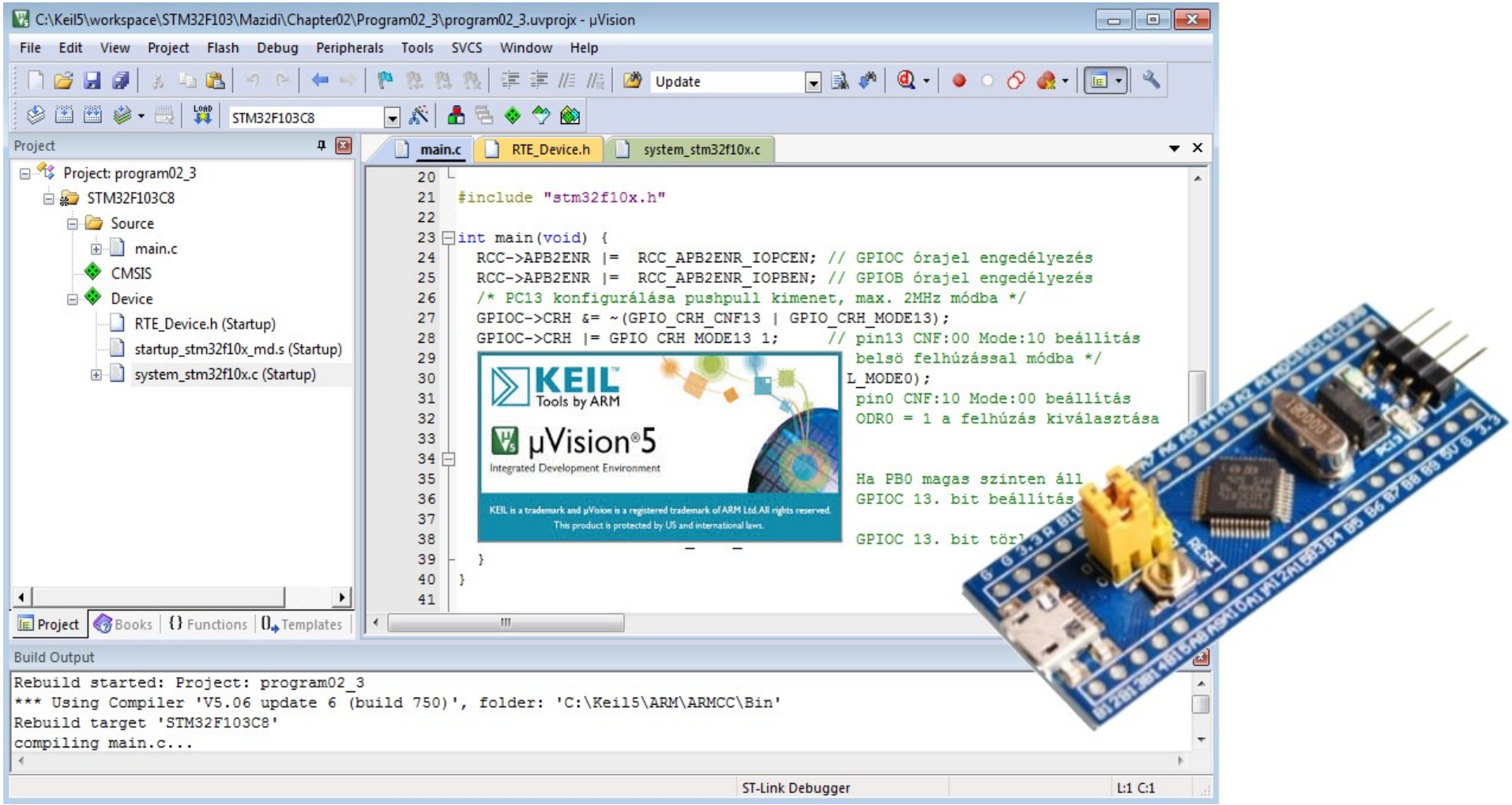
STM32 mikrovezérlők programozása ARM Keil környezetben



6. Aszinkron soros kommunikáció (USART) – 2. rész

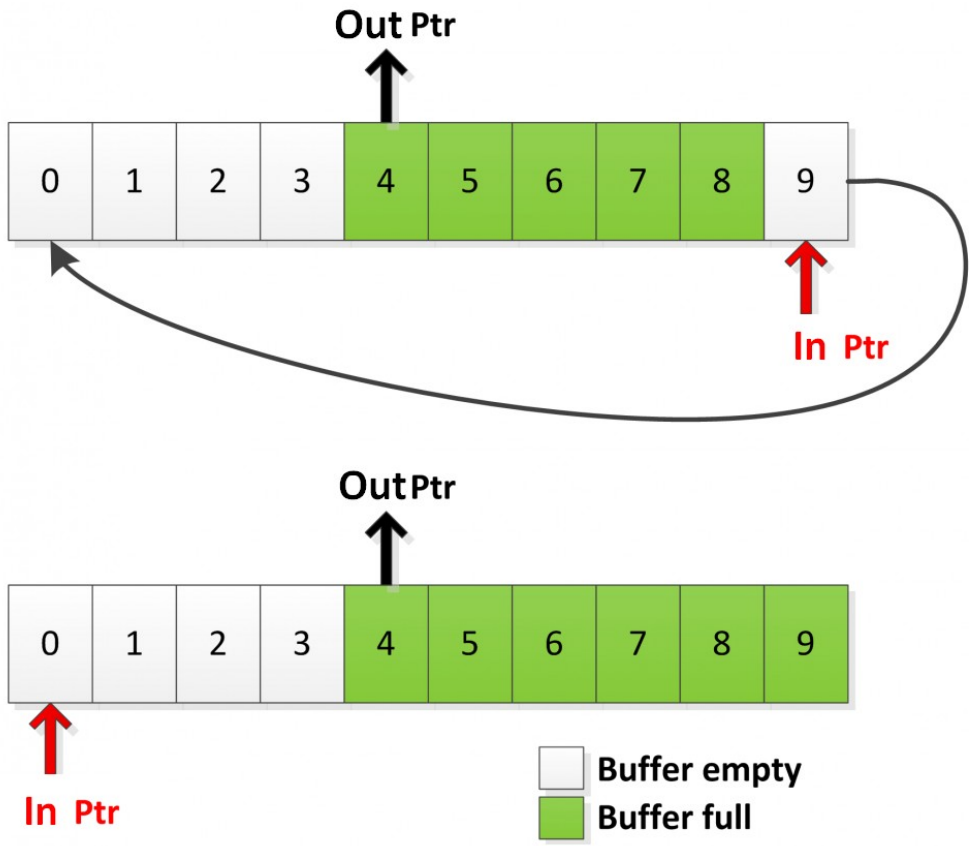
Felhasznált és ajánlott irodalom

- Joseph Yiu: Cortex-M for Beginners
- Joseph Yiu: The Definitive Guide To The ARM CORTEX-M3
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: STM32 Arm Programming for Embedded Systems
- Alexander Tarasov: **Курс «Штурмуем STM32»**
- Warren Gay: Beginning STM32 - Developing with FreeRTOS, libopencm3 and GCC
- ARM Keil MDK Getting started
- **STM32F103C8** adatlap és termékinfo
- **STM32F103** Family Reference Manual

Megszakításvezérelt körkörös bufferelés

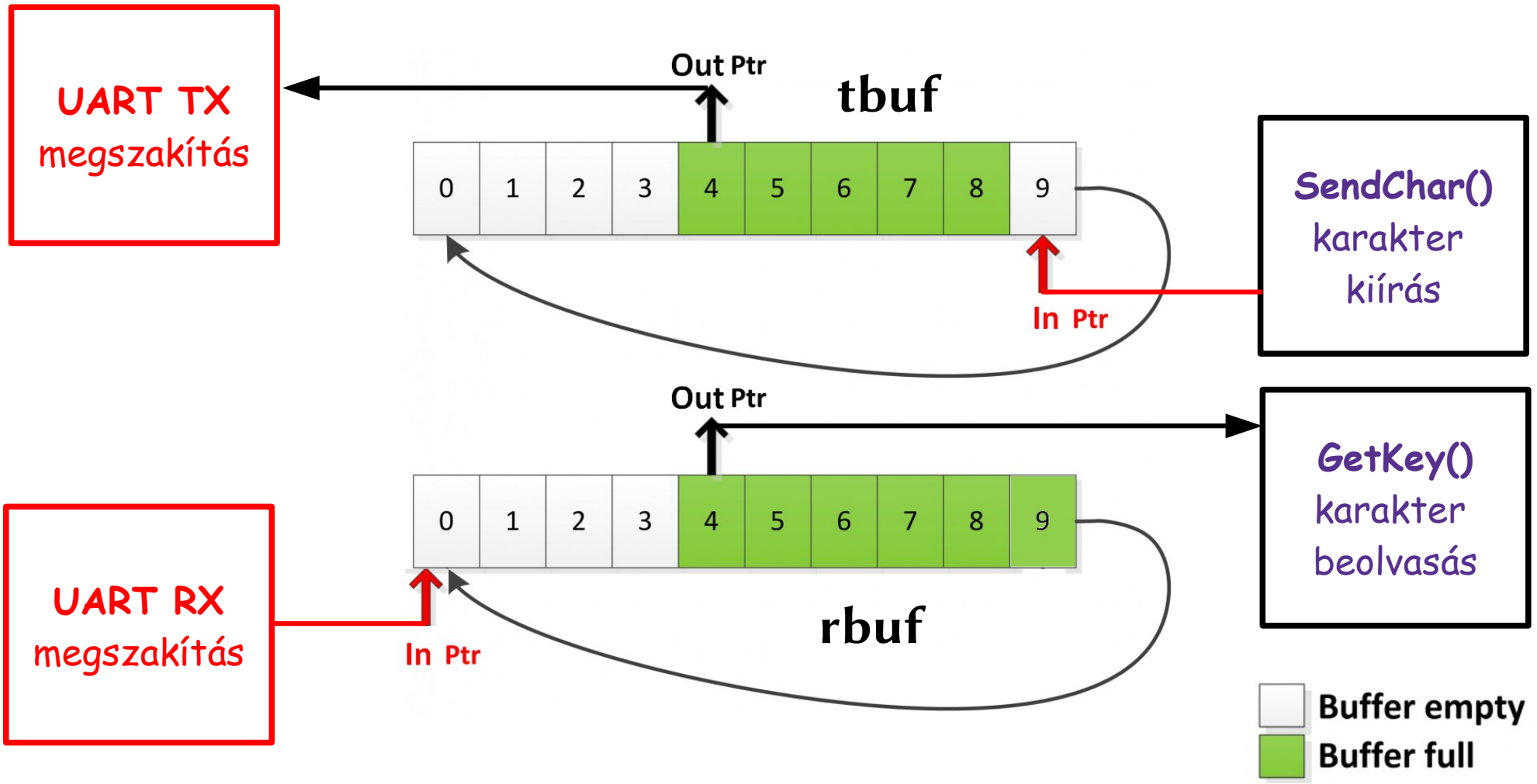
- Amikor az UART egy karaktert fogad, a főprogram nem feltétlenül tudja azonnal feldolgozni ⟶ ideiglenesen el kell tárolni
- Amikor a főprogram karaktereket küld, ne várjuk meg azok soros kiléptetését az UART kimenetén ⟶ ideiglenesen el kell tárolni

- 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



A bufferek definiálása

```
#define TBUF_SIZE   1024     // Must be a power of 2 (2,4,8,16,32,64,128,256,512,...)
#define RBUF_SIZE   256      // Must be a power of 2 (2,4,8,16,32,64,128,256,512,...)

#if TBUF_SIZE < 2
#error TBUF_SIZE is too small.  It must be larger than 1.
#elif ((TBUF_SIZE & (TBUF_SIZE-1)) != 0)
#error TBUF_SIZE must be a power of 2.
#endif

#if RBUF_SIZE < 2
#error RBUF_SIZE is too small.  It must be larger than 1.
#elif ((RBUF_SIZE & (RBUF_SIZE-1)) != 0)
#error RBUF_SIZE must be a power of 2.
#endif

struct buf_st {
  unsigned int in;                              // Next In Index
  unsigned int out;                             // Next Out Index
  char buf [TBUF_SIZE];                         // Buffer
};

static struct buf_st rbuf = { 0, 0, };
#define SIO_RBUFLEN ((unsigned short)(rbuf.in - rbuf.out))
static struct buf_st tbuf = { 0, 0, };
#define SIO_TBUFLEN ((unsigned short)(tbuf.in - tbuf.out))
static unsigned int tx_restart = 1;           // NZ if TX restart is required
```

Egy Keil mintapélda:
STM32 USART interrupt mode example
nyomán **<link>**

Buffer inicializálás, karakter küldés/fogadás

```
void buffer_Init (void) {
  tbuf.in = 0; tbuf.out = 0; tx_restart = 1;
  rbuf.in = 0; rbuf.out = 0;
}

//--- SendChar - transmit a character ------
int SendChar (int c) {
  struct buf_st *p = &tbuf;
  if (SIO_TBUFLEN >= TBUF_SIZE)
    return (-1);                          // If buffer is full, return an error code
  p->buf [p->in & (TBUF_SIZE - 1)] = c;   // Add data to the transmit buffer.
  p->in++;
  if (tx_restart) {                       // If transmit interrupt should be reenabled
    tx_restart = 0;
    USART1->CR1 |= USART_SR_TXE;          // enable TX interrupt
  }
  return (0);
}

//--- GetKey - receive a character -------
int GetKey (void) {
  struct buf_st *p = &rbuf;
  if (SIO_RBUFLEN == 0)
    return (-1);                          // If buffer is empty, return an error code
  return (p->buf[(p->out++) & (RBUF_SIZE-1)]);
}
```

A bufferek inicializálása

Egy Keil mintapélda:
STM32 USART interrupt mode example
nyomán <link>

A megszakítások kiszolgálása

Az USART portok csak egy megszakítási vektorral rendelkeznek, így vizsgálni kell a megszakítás forrását!

```
void USART1_IRQHandler (void) {
  volatile unsigned int IIR;
  struct buf_st *p;
  IIR = USART1->SR;                       // Státuszregiszter másolat
  if (IIR & USART_SR_RXNE) {              // Ha van beérkezett karakter
    USART1->SR &= ~USART_SR_RXNE;         // jelzőbit törlése
    p = &rbuf;                            // Eltárolás rbuf-ban
    if (((p->in - p->out) & ~(RBUF_SIZE-1)) == 0) {
      p->buf [p->in & (RBUF_SIZE-1)] = (USART1->DR & 0x1FF);
      p->in++;
    }
  }
  if (IIR & USART_SR_TXE) {
    USART1->SR &= ~USART_SR_TXE;          // clear interrupt
    p = &tbuf;
    if (p->in != p->out) {                // Ha nem üres a tbuf küldő buffer
      USART1->DR = (p->buf [p->out & (TBUF_SIZE-1)] & 0x1FF);
      p->out++;
      tx_restart = 0;
    }
    else {                                // Ha üres a buffer ...
      tx_restart = 1;
      USART1->CR1 &= ~USART_SR_TXE;       // disable TX interrupt if nothing to send
    }
  }
}
```

Egy Keil mintapélda:
STM32 USART interrupt mode example
nyomán <link>

USART1 és PB6/PB7 inicializálása

- **USART1** és a **PB6/PB7** kivezetések inicializálását **Program4_4**-ből emeltük át, csak a pirossal bejelölt sorokkal kellett kibővíteni

```
void USART1_init (void) {
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN;      // USART1 engedélyezése
  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;        // AFIO órajel engedélyezés
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;        // GPIOB órajel engedélyezés
  AFIO->MAPR |= AFIO_MAPR_USART1_REMAP;      // Beállítja USART1 REMAP bitjét
  //--- PB6 konfigurálása USART1 Tx digitális kimenetként -----------
  GPIOB->CRL &= ~(GPIO_CRL_MODE6|GPIO_CRL_CNF6); // PB6 CNF és Mode bitek törlése
  GPIOB->CRL |=  GPIO_CRL_MODE6_0 |          // PA2 pushpull alt. kimenet, max. 10 MHz
                 GPIO_CRL_CNF6_1;            // CNF: 10  MODE: 01
  //--- PB7 konfigurálása digitális bemenetként  -----------
  GPIOB->CRL &= ~(GPIO_CRL_MODE7|GPIO_CRL_CNF7); // PB7 CNF és Mode bitek törlése
  GPIOB->CRL |=  GPIO_CRL_CNF7_0;            // PB7 digitális bemenet CNF: 01  MODE: 00

  //--- USART1 konfigurálása: Tx&Rx, 9600  8 N 1 módba -------------
  USART1->BRR = 7500;                        // 9600 baud @ 72 MHz (APB2!)
  USART1->CR1 = USART_CR1_TE | USART_CR1_RE| // Tx, Rx engedélyezés, 8-bit adat
                USART_CR1_TXEIE |            // TXE  megszakítás engedélyezése
                USART_CR1_RXNEIE;            // RXNE megszakítás engedélyezése
  NVIC_EnableIRQ(USART1_IRQn);               // USART1 megszakítás engedélyezése
  USART1->CR2 = 0x0000;                      // 1 stop bit
  USART1->CR3 = 0x0000;                      // nincs adatfolyam-vezérlés
  USART1->CR1 |= USART_CR1_UE;               // USART1 engedélyezés
}
```

uart_irq/retarget2.c

- A **printf(), scanf()** függvények átirányítása a soros portra ugyanúgy történik, ahogyan korábban **Program04_5**-ben csináltuk
- Az előző oldalakon bemutatott **SendChar()** és **GetKey()** függvények felhasználásával fölül kell definiálnunk a **fputc()** valamint a **fgetc()** függvényeket
- Ne feledjük el becsatolni a **stdio.h** fejléc állományt!

```
#include <stdio.h>
#pragma import(__use_no_semihosting_swi)

extern int  SendChar(int ch);
extern int  GetKey(void);

struct __FILE {
  int handle;
};
FILE __stdin  = {0};
FILE __stdout = {1};
FILE __stderr = {2};

int fputc(int ch, FILE *f) {
  return (SendChar(ch));
}

int fgetc(FILE *f) {
  int ch;
  do {
    ch = GetKey ();
  }
  while (ch == -1);
  SendChar(ch);
  return (ch);
}
```

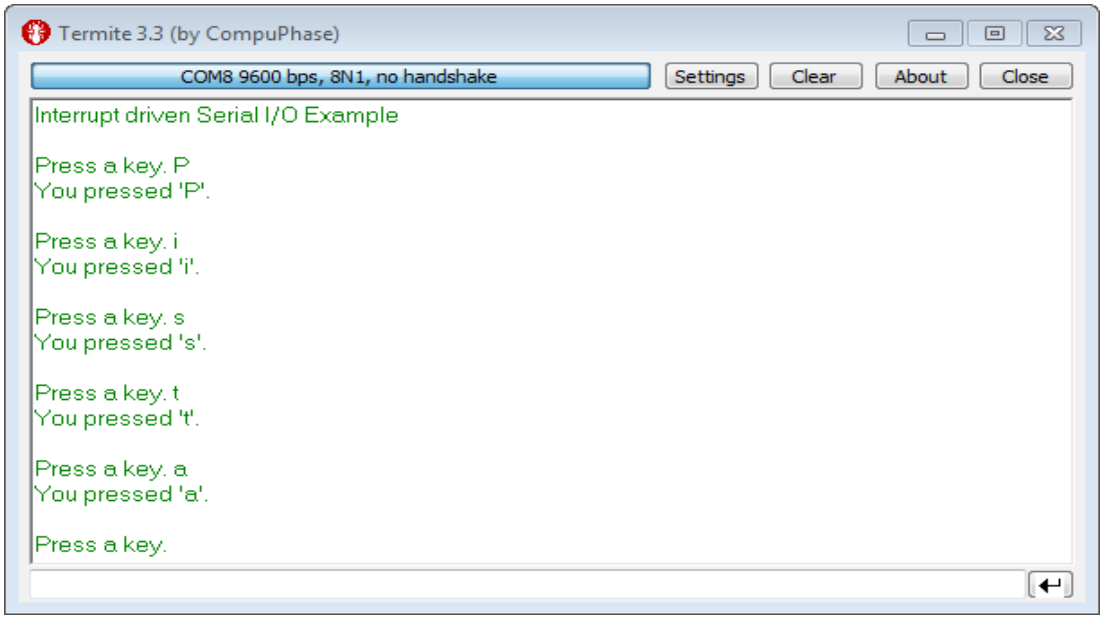
uart_irq/main.c

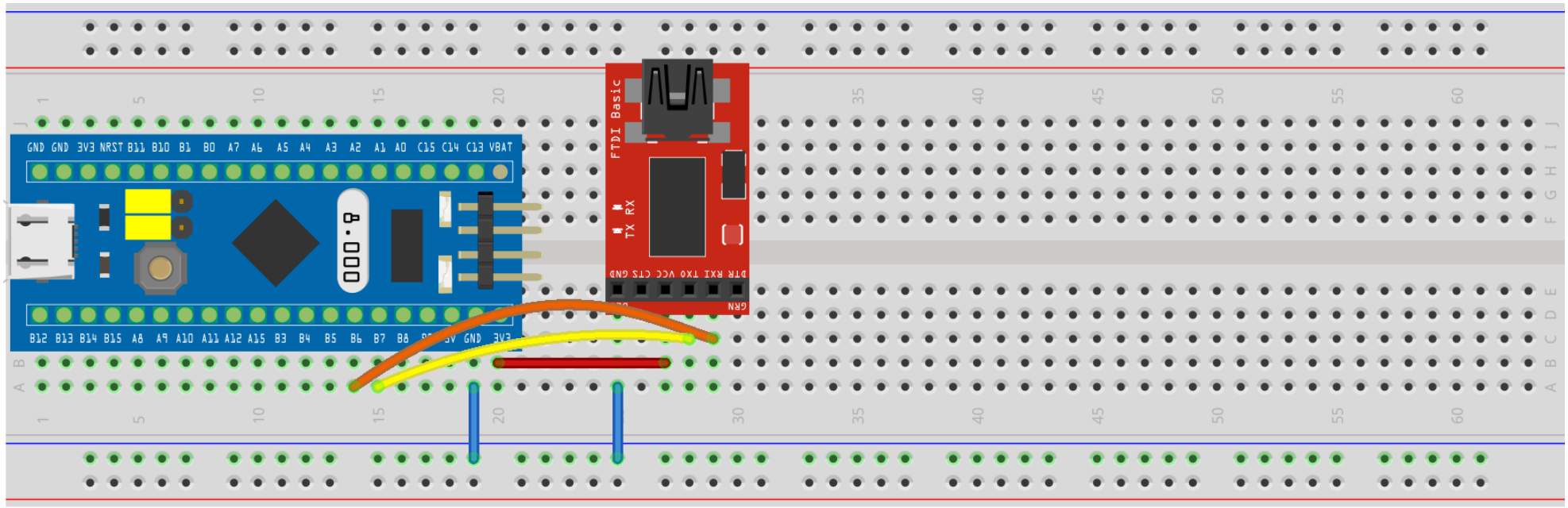
- A főprogramban az inicializálások után ne feledkezzünk meg a megszakítások globális engedélyezéséről sem!

```
#include "stm32f10x.h"
#include <stdio.h>

extern void buffer_Init (void);
extern void USART1_init (void);

/*----------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------*/
int main (void) {
  buffer_Init();                                // RX / TX bufferek inicializálása
  USART1_init();                                // USART1 configuration
  __enable_irq();                               // Megszakítások globális engedélyezése
  printf ("Interrupt driven Serial I/O Example\r\n\r\n");
  while (1) {                                   // végtelen ciklus
    unsigned char c;
    printf ("Press a key. ");
    c = getchar ();                             // Beolvasunk egy karaktert
    printf ("\r\n");
    printf ("You pressed '%c'.\r\n\r\n", c); // Kiírjuk az eredményt
  }
}
```

uart_irq: kapcsolás és eredmény

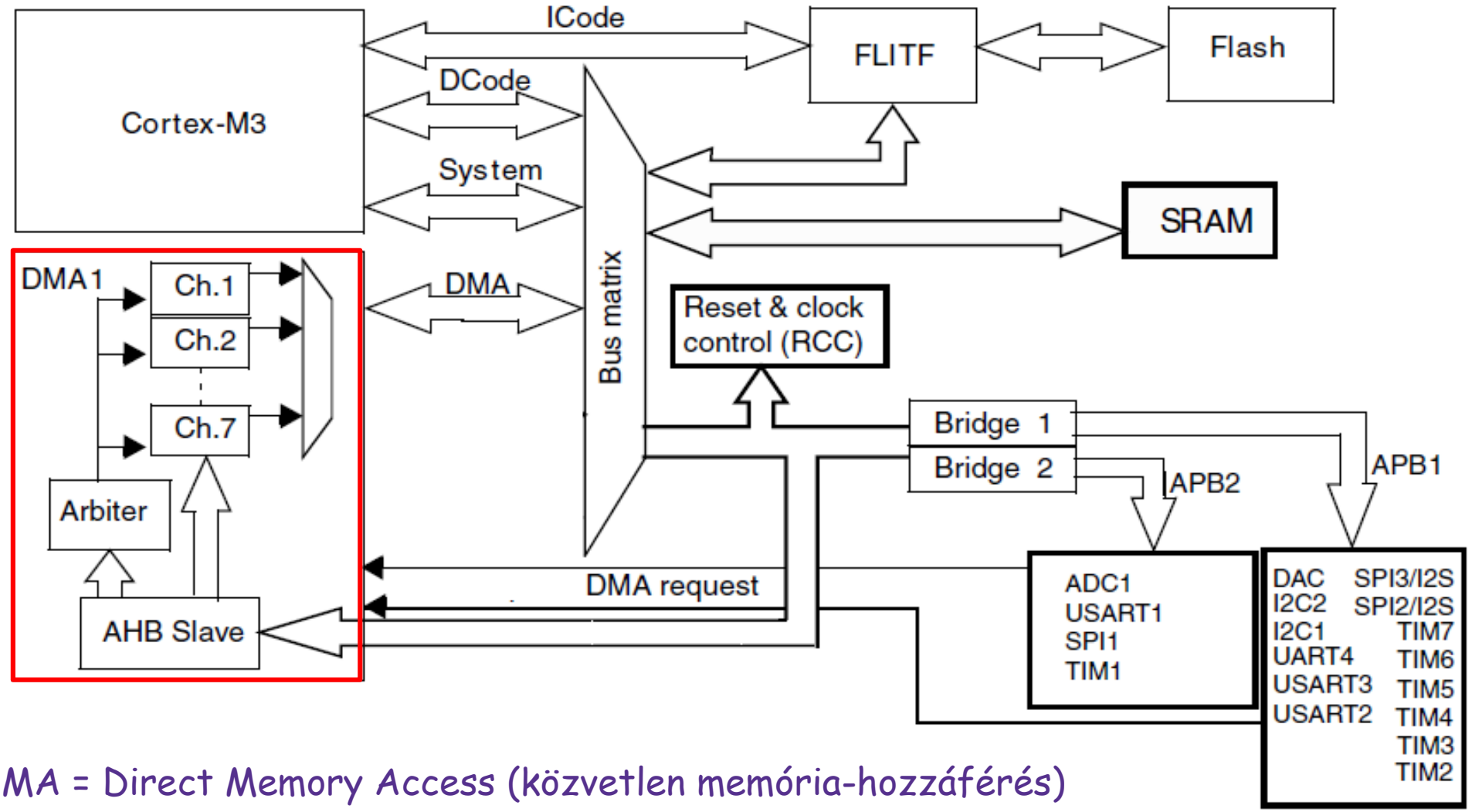
- **PB6** az átalakító **Rx** lábára, **PB7** az átalakító **Tx** lábára legyen kötve
- A terminált állítsuk 9600 bps-re!





DMA vezérlő az STM32F103C8 MCU-ban

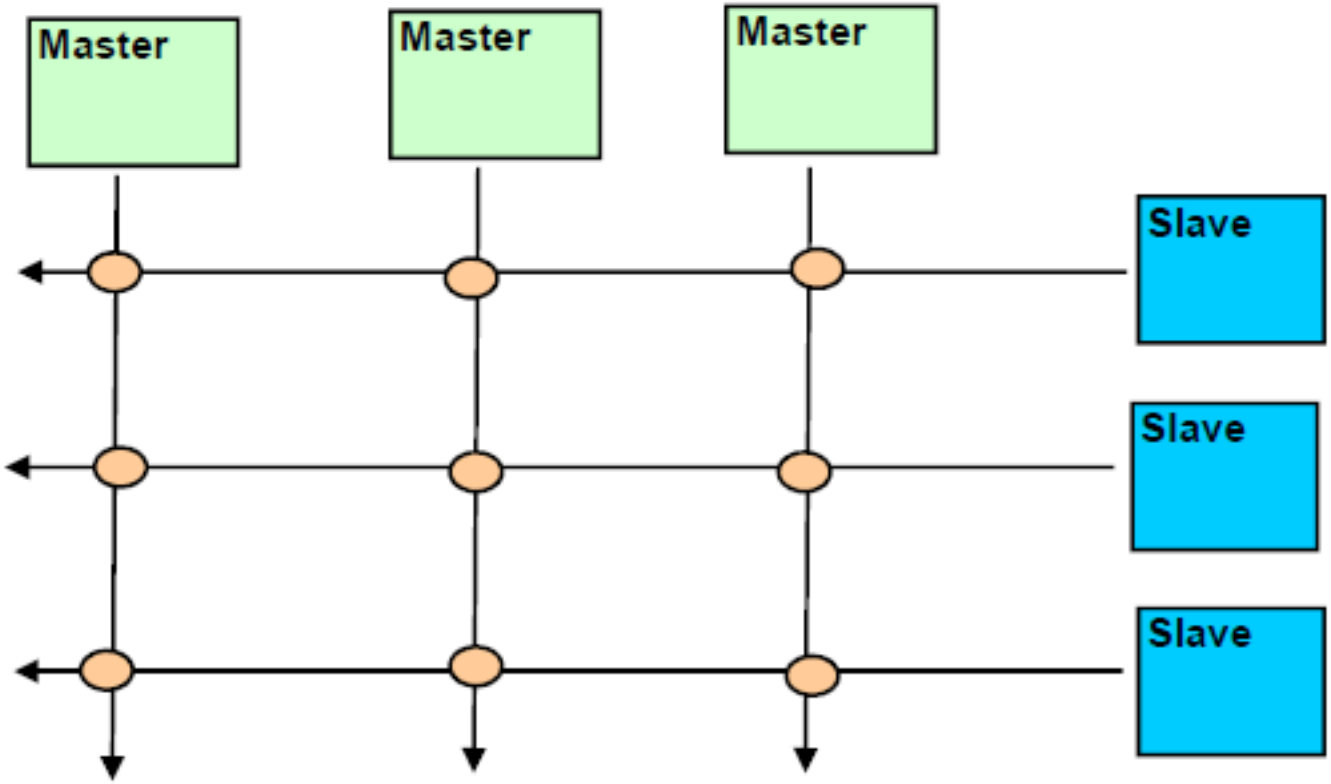
- DMA vezérlő: a CPU-hoz hasonlóan busz műveleteket indíthat
- DMA átvitel indítása: hardveres kérelmekkel vagy szoftveresen



DMA = Direct Memory Access (közvetlen memória-hozzáférés)

AHB Bus Matrix

- Az **AHB** (advanced high-performance bus) párhuzamos hozzáférést enged a master eszközöknek, ha nem ugyanazt a periféria- vagy memória buszt használják



A DMA csatornák kiosztása

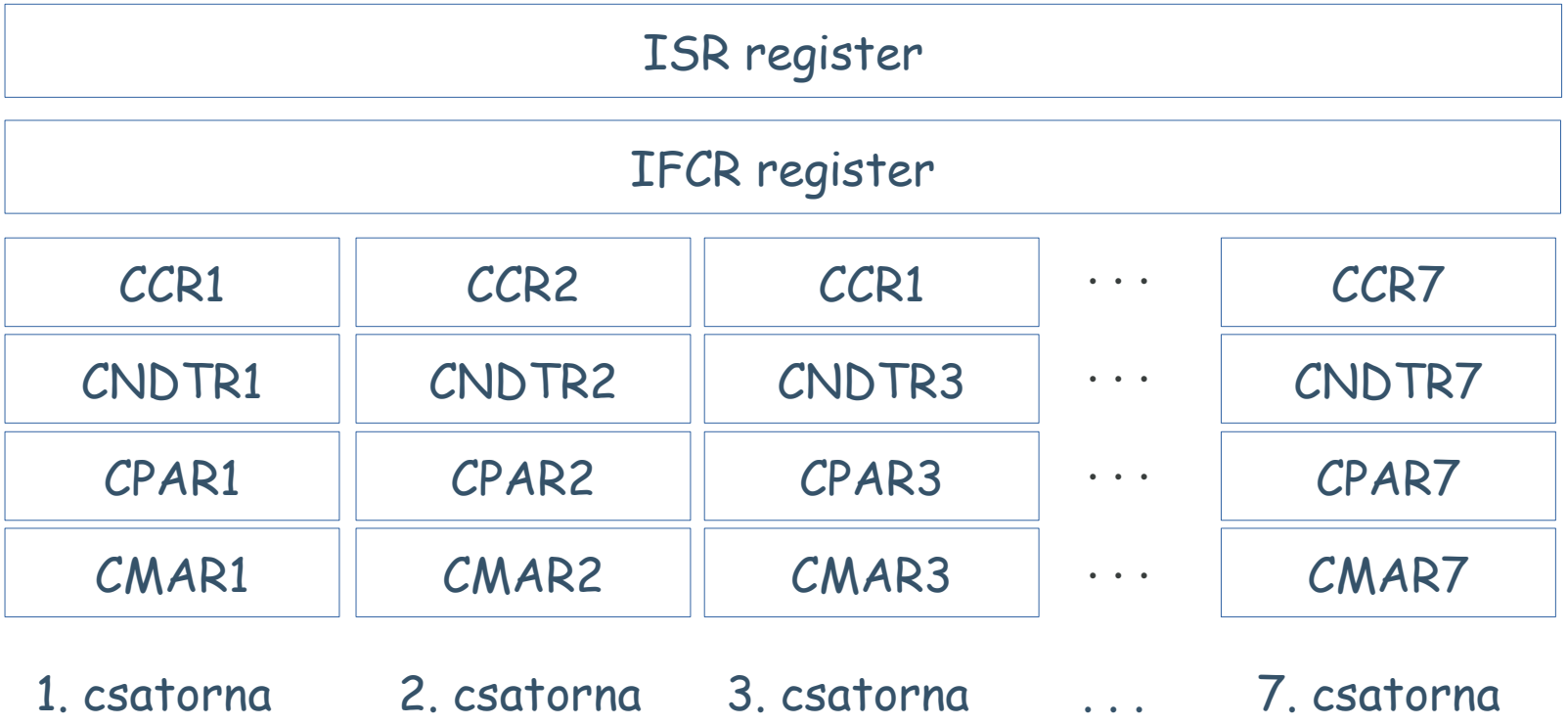
- Az **STM32F103C8** mikrovezérlő egy **DMA** vezérlője van, amely 7 csatornával rendelkezik
- Az alábbi táblázatban látható, hogy melyik periféria melyik csatornát használhatja

Table 78. Summary of DMA1 requests for each channel

Peripherals	Channel 1	Channel 2	Channel 3	Channel 4	Channel 5	Channel 6	Channel 7
ADC1	ADC1	-	-	-	-	-	-
SPI/I^2S	-	SPI1_RX	SPI1_TX	SPI2/I2S2_RX	SPI2/I2S2_TX	-	-
USART	-	USART3_TX	USART3_RX	USART1_TX	USART1_RX	USART2_RX	USART2_TX
I^2C	-	-	-	I2C2_TX	I2C2_RX	I2C1_TX	I2C1_RX
TIM1		TIM1_CH1	-	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
TIM2	TIM2_CH3	TIM2_UP	-	-	TIM2_CH1	-	TIM2_CH2 TIM2_CH4
TIM3	-	TIM3_CH3	TIM3_CH4 TIM3_UP	-	-	TIM3_CH1 TIM3_TRIG	-
TIM4	TIM4_CH1	-	-	TIM4_CH2	TIM4_CH3	-	TIM4_UP

A DMA vezérlő regiszterei

- A DMA vezérlő állapotjelző és megszakításkérést törlő regiszterei közösek
- Minden csatornához tartozik egy-egy vezérlő (**CCR*x***), egy adatszám (**CNDTR*x***) egy perifériacím (**CPAR*x***) és egy memóriacím (**CMAR*x***) regiszter



DMA megszakításkérő bitek

- **DMA1_ISR** – interrupt status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TEIF7	HTIF7	TCIF7	GIF7	TEIF6	HTIF6	TCIF6	GIF6	TEIF5	HTIF5	TCIF5	GIF5
				r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TEIF4	HTIF4	TCIF4	GIF4	TEIF3	HTIF3	TCIF3	GIF3	TEIF2	HTIF2	TCIF2	GIF2	TEIF1	HTIF1	TCIF1	GIF1
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

- **TEIFx** – Channel x Transfer Error Interrupt Flag
- **HTIFx** – Channel x Half Transfer Flag
- **TCIFx** – Channel x Transfer Complete Flag
- **GIFx** – Channel x Global interrupt flag (GIF = TEIF | HTIF | TCIF)

DMA megszakítási kérelmek törlése

- **DMA1_IFCR** – interrupt flag clear register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				CTEIF7	CHTIF7	CTCIF7	CGIF7	CTEIF6	*CHTIF6*	CTCIF6	CGIF6	CTEIF5	CHTIF5	CTCIF5	CGIF5
				w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEIF4	CHTIF4	CTCIF4	CGIF4	CTEIF3	CHTIF3	CTCIF3	CGIF3	CTEIF2	CHTIF2	CTCIF2	CGIF2	CTEIF1	CHTIF1	CTCIF1	CGIF1
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

- **CTEIFx** – Channel x Clear Transfer Error Interrupt Flag
- **CHTIFx** – Channel x Clear Half Transfer Flag
- **CTCIFx** – Channel x Clear Transfer Complete Flag
- **CGIFx** – Channel x Clear Global interrupt flag

- **Megjegyzések:** A megszakításkérés törléséhez 1-et kell írni a **DMA1_IFCR** regiszter megfelelő bitjébe. A **CGIFx** bit 1-be állítása az x. csatorna többi jelzőbitjét is törli.

DMA csatorna konfigurációs regiszter

- **DMA1_CCR*x*** – channel *x* configuration register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	MEM2MEM	PL[1:0]		MSIZE[1:0]		PSIZE[1:0]		MINC	PINC	CIRC	DIR	TEIE	HTIE	TCIE	EN
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1

- **MEM2MEM** – memória-memóriaátvitelt (0: tilt, 1: engedélyez)
- **PL[1:0]** – Prioritási szint
- **MSIZE** – memóriacím léptetés (0: 8, 1: 16, 2: 32 bitenként)
- **PSIZE** – periféria cím léptetés (0: 8, 1: 16, 2: 32 bitenként)
- **MINC** – Memóriacím léptetés (0: tilt, 1: enged)
- **PINC** – Periféria cím léptetés (0: tilt, 1: enged)
- **CIRC** – körkörös buffer mód (0: tilt, 1: enged)
- **DIR** – adatmozgatás iránya (0: periféria ⟶ mem, 1: mem⟶ periféria
- **TEIE, HTIE, TCIE** – megszakítások engedélyezése
- **EN** – a DMA vezérlő működésének engedélyezése

uart_dma/main.c

- **USART1** szövegküldés az **RB6** lábon, **DMA** segítségével

```
#include "stm32f10x.h"

void USART1_init(void);
void DMA1_init(void);
void DMA1_ch4_setup(unsigned int src, unsigned int dst, int len);

volatile int done = 1;
char alphabets[] = "abcdefghijklmnopqrstuvwxyz";
char message[80];
int size = sizeof(alphabets);

int main (void) {
  USART1_init();
  DMA1_init();
  __enable_irq();                                  // Megszakítások globális engedélyezése

  while (1) {
    for (int i = 0; i < size; i++)                 // az üzenet előkészítése
      message[i] = alphabets[i];
    while (done == 0) {}                           // Várakozás, amíg a done jelző=0
    done = 0;                                      // Jelzo törlése
    DMA1_ch4_setup((unsigned int)message, (unsigned int)&USART1->DR, size);
  }
}
```

uart_dma/main.c

```
//--- USART1 és a PB6 kivezetés inicializálása --------
void USART1_init (void) {
  RCC->APB2ENR |= RCC_APB2ENR_USART1EN;       // USART1 engedélyezése
  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;         // AFIO órajel engedélyezés
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN;         // GPIOB órajel engedélyezés
  AFIO->MAPR |= AFIO_MAPR_USART1_REMAP;       // Beállítja USART1 REMAP bitjét
  //--- PB6 konfigurálása USART1 Tx digitális kimenetként -----------
  GPIOB->CRL &= ~(GPIO_CRL_MODE6|GPIO_CRL_CNF6); // PB6 CNF és Mode bitek törlése
  GPIOB->CRL |=  GPIO_CRL_MODE6_0 |           // PA2 pushpull alt. kimenet, max. 10 MHz
                 GPIO_CRL_CNF6_1;             // CNF: 10  MODE: 01
  //--- USART1 konfigurálása: Tx only, 9600  8 N 1 módba --------------
  USART1->BRR = 7500;                         // 9600 baud @ 72 MHz (APB2!)
  USART1->CR1 = USART_CR1_TE;                 // Tx engedélyezés, 8-bit adat
  USART1->CR2 = 0x0000;                       // 1 stop bit
  USART1->SR = ~USART_SR_TC;                  // Töröljük a TC jelzőbitet
  USART1->CR1 |= USART_CR1_TCIE;              // Átvitel vége megszakítás engedélyezése
  NVIC_EnableIRQ(USART1_IRQn);                // USART1 megszakítás engedélyezése
  USART1->CR1 |= USART_CR1_UE;                // USART1 működés engedélyezése
}

//--- A DMA1 vezérlő 4. csatorna inicializálása --------
void DMA1_init(void) {
  RCC->AHBENR |= RCC_AHBENR_DMA1EN;           // DMA vezérlő engedélyezése
  DMA1->IFCR = DMA_IFCR_CGIF4;                // Törli 4. csat. megszakítási jelzőbitjeit
  NVIC_EnableIRQ(DMA1_Channel4_IRQn);         // DMA 4. csatorna megszakítás engedélyezés
}
```

uart_dma/main.c

- Az alábbi kód minden adatblokk indítása előtt lefut
- Beállítjuk a forrás és a cél címét, adatok számát, átvitel módját
- Szemléltetésül engedélyezzük a TE és a TC megszakítást de enélkül is működik a program mert, USART1 TCIF jelzi az átvitel végét

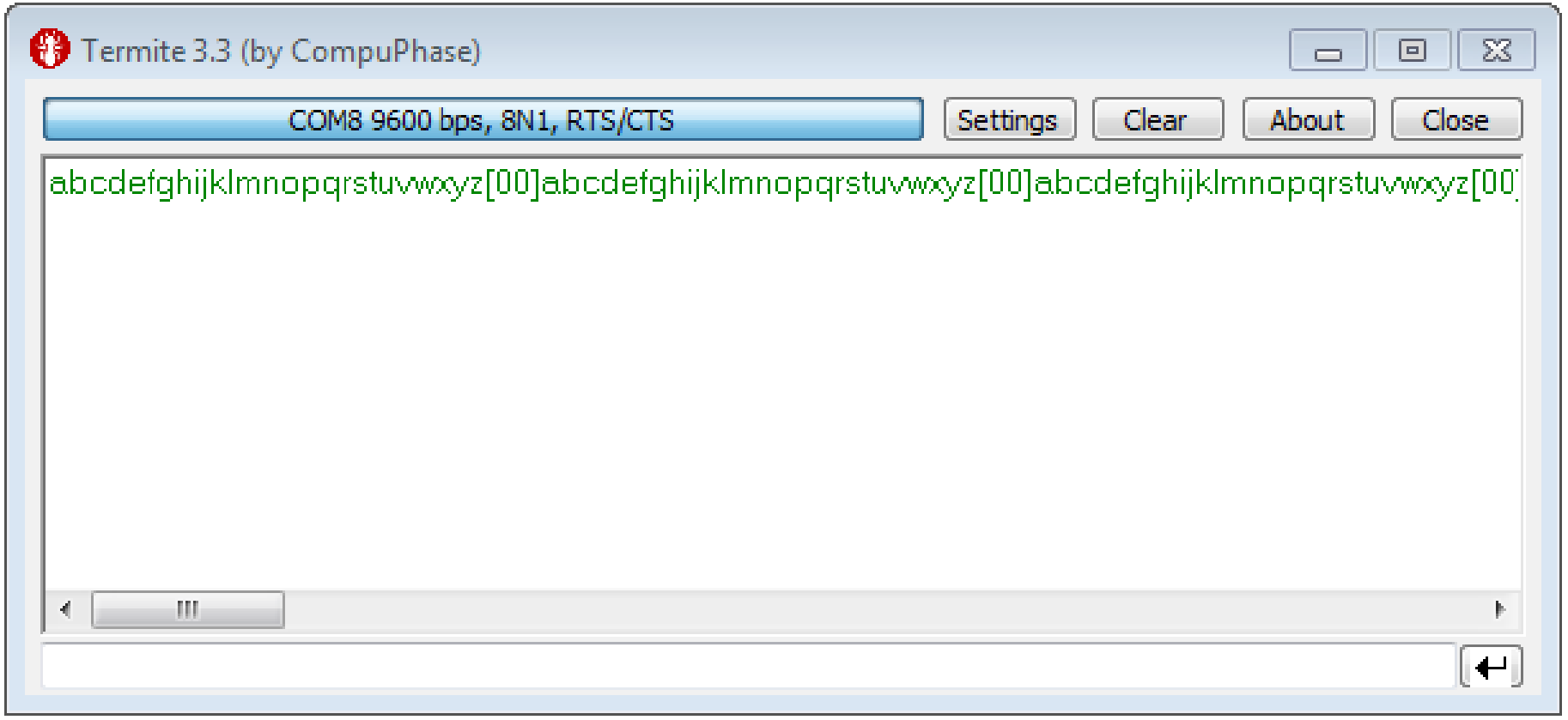
```
/*-----------------------------------------------------------------------
   DMA 4. csatorna --> UART1 Tx adatátvitel konfigurálása és indítása
 *-----------------------------------------------------------------------*/
void DMA1_ch4_setup(unsigned int src, unsigned int dst, int len) {
  DMA1_Channel4->CCR = 0;                    // DMA1 Channel 4 ideiglenes letiltása
  while (DMA1_Channel4->CCR & 1) {}          // Vár, amíg DMA1 Channel 4 aktív
  DMA1->IFCR |= DMA_IFCR_CGIF4;              // Törli 4. csat. megszakítási jelzőbitjeit
  DMA1_Channel4->CPAR = dst;                 // Destination  address  (cél)
  DMA1_Channel4->CMAR = src;                 // Source address  (forrás)
  DMA1_Channel4->CNDTR = len;                // length (adatblokk hossza)
  DMA1_Channel4->CCR |= 0x0090;              // MINC=1 PINC=0 CIRC=0 DIR=1 (mem→per)
  DMA1_Channel4->CCR |= DMA_CCR4_TEIE;       // Transfer Error megszakítés engedélyezés
  DMA1_Channel4->CCR |= DMA_CCR4_TCIE;       // Transfer Complete megszakítás engedése
  DMA1_Channel4->CCR |= DMA_CCR4_EN;         // DMA1 Channel 4 engedélyezése

  USART1->SR &= ~USART_SR_TC;                // USART átvitel vége jelzőbit törlése
  USART1->CR3 = USART_CR3_DMAT;              // DMA Transmit engedélyezés
}
```

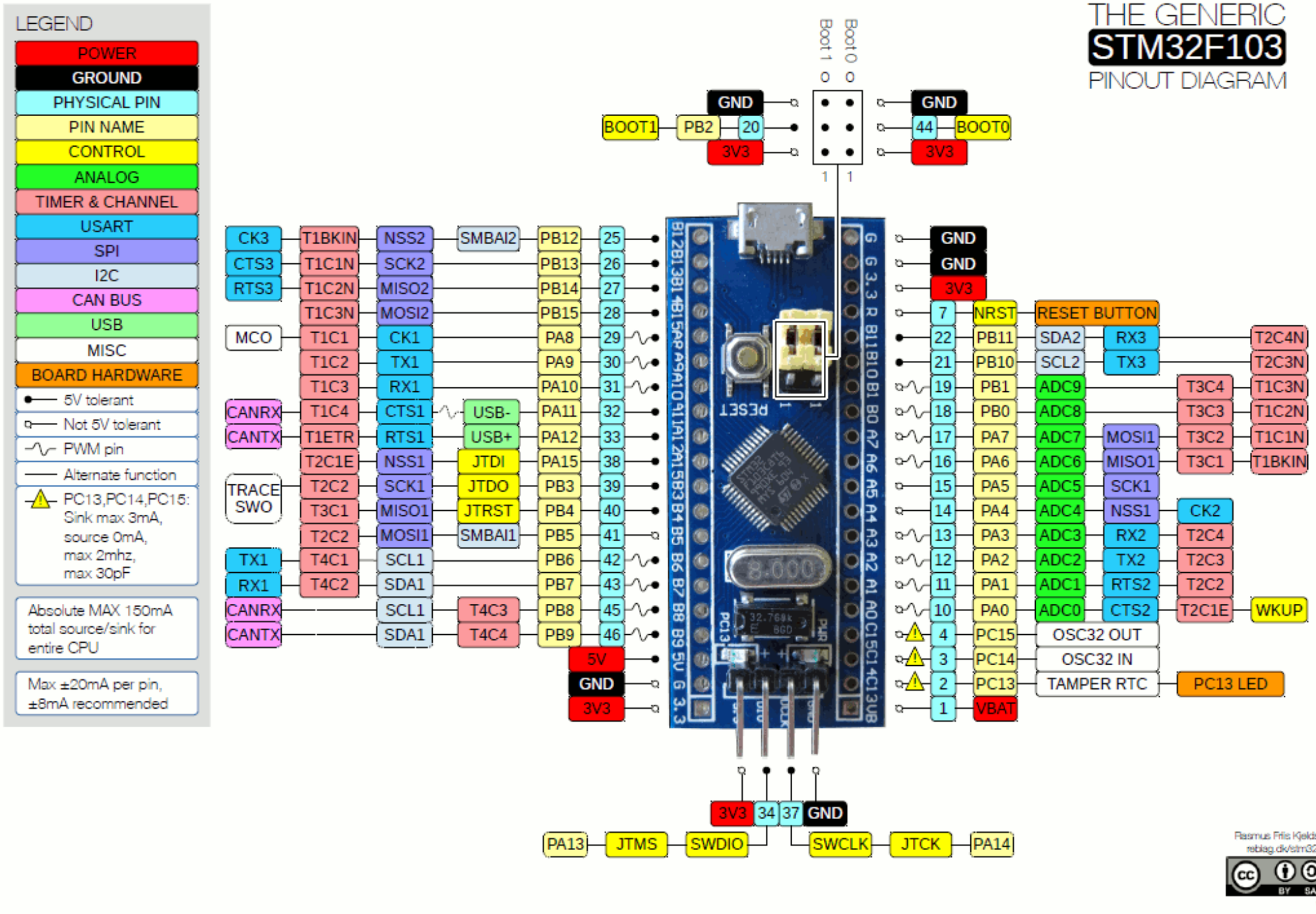
uart_dma/main.c

```
/*-------------------------------------------------------------------------
   DMA1 Channel4 megszakítás kiszolgálása
   Ez a függvény kezeli a DMA1 vezérlő 4. csatornájától érkező megszakításokat
   A hibajelző megszakítás itt most csak egy helykitöltő végtelen ciklus,
   melynek helyére a hiba esetén futtatandó kódot írhatjuk.
   Ha a megszakítás a DMA átvitel végét jelzi, a DMA megszakításokat letiltjuk
   s a megszakításkérő jelzőbiteket töröljük.
 *-------------------------------------------------------------------------*/
void DMA1_Channel4_IRQHandler(void) {
  if (DMA1->ISR & DMA_ISR_TEIF4)              // Ha átviteli hiba történt...
    while(1) {}                               // helyettesítsük ezt hibakezelo kóddal
  DMA1->IFCR |= DMA_IFCR_CGIF4;               // Törli 4. csatorna megszakításijelzőit
  DMA1_Channel4->CCR &= ~(DMA_CCR4_TCIE|      // Transfer Complete megszakítés letiltása
                          DMA_CCR4_TEIE);     // Transfer Error megszakítés letiltása
}

/*-------------------------------------------------------------------------
   USART1 megszakítás kiszolgálása
   Az USART1 modul Transmit Complete megszakítását használjuk az átvitel végének
   jelzésére. Ennek bebillenésekor állítjuk 1-be a szemaforként használt done
   változót, amit a főprogram figyel
 *-------------------------------------------------------------------------*/
void USART1_IRQHandler(void) {
    USART1->SR &= ~USART_SR_TC ;              // UART1 TC jelzobitjének törlése
    done = 1;                                 // A done jelzo 1-be állítása
}
```

uart_dma futási eredménye

- A program futási eredménye az alábbi ábrán látható
- A szövegblokkok végén 0 kódú karakterek is láthatók, ami a szövegkonstansok zárókaraktere. Ha el akarjuk kerülni, a szöveküldés hosszána ***size*** helyett ***size-1***-et írjunk!




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
Boot1
Boot0
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
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
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 Friis Kjeldsen
reblag.dk/stm32
V1.0