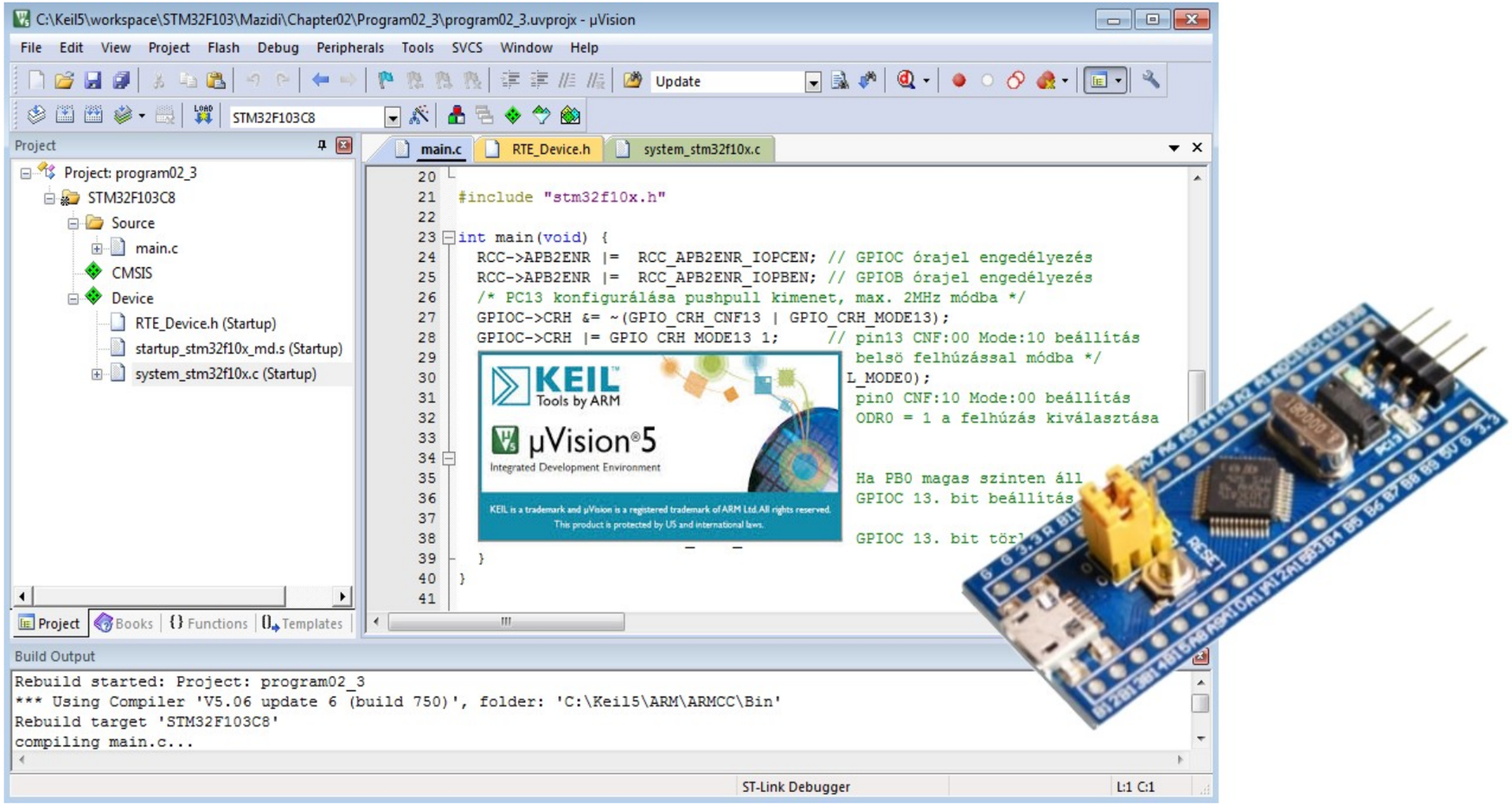
STM32 mikrovezérlők programozása ARM Keil környezetben



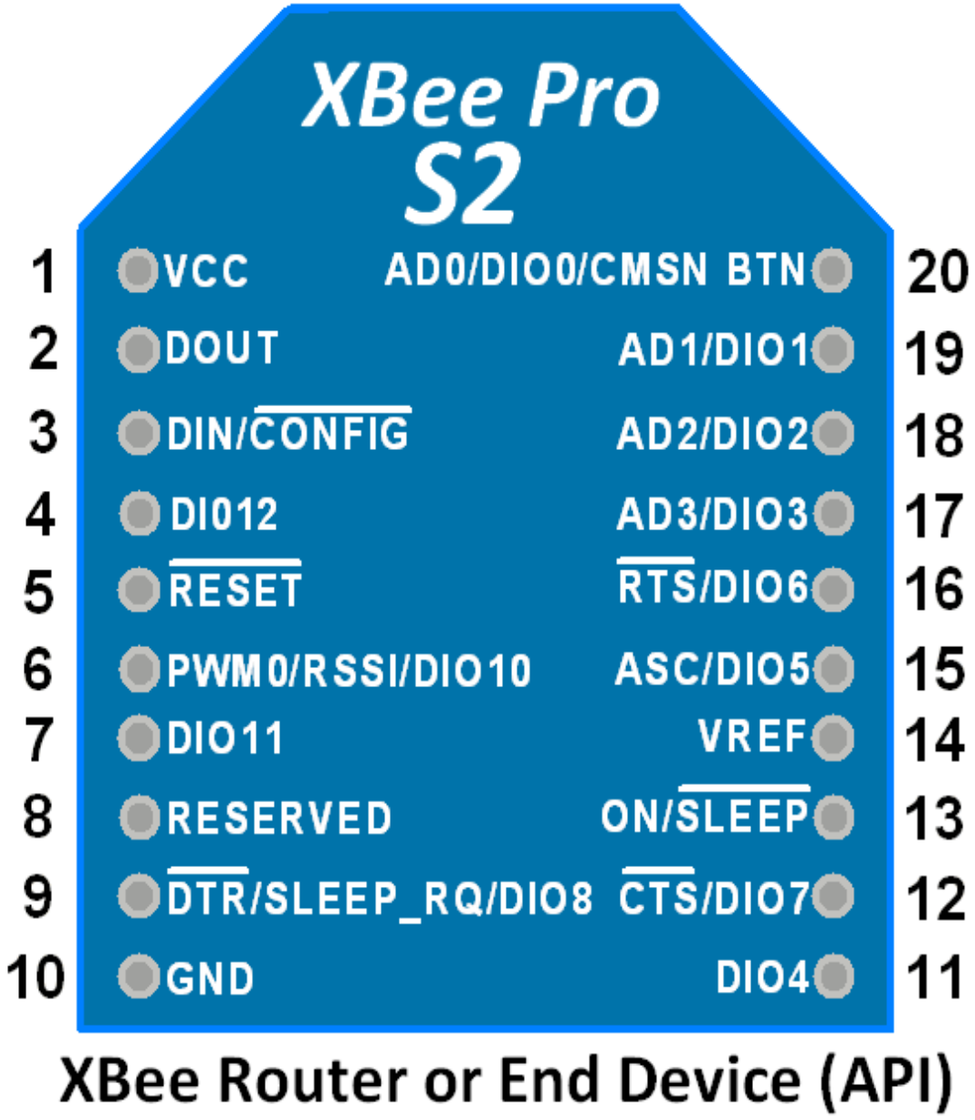
16. Vezeték nélküli kommunikáció (*Xbee/ZigBee*) – 2. rész

Felhasznált és ajánlott irodalom

- 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»**
- ARM Keil MDK Getting started
- **STM32F103C8** adatlap és termékinfo
- **STM32F103** Family Reference Manual
- Matthijs Kooijman: Building Wireless Sensor Networks Using Arduino
- Robert Faludi: Building Wireless Sensor Networks: With Zigbee, Xbee, Arduino, And Processing
- DigiKey (Scott Schmit): XBee API Mode - Read Remote ADC Example
- **DIGI International:** Zigbee RF Modules XBEE2, XBEEPRO2, PRO S2B
- **DIGI International:** XTCU Configuration and Test Utility Software

Az Xbee modul kivezetései – 1. oldal

Pin	Name	dir	default	Fuction
1	**VCC**	PWR	–	Power supply
2	**DOUT**	out	out	UART data out
3	**DIN**/CONFIG	in	in	UART data in
4	DIO12	both	out	Digital I/O 12
5	RESET	both	OC w Pup	Module reset
6	RSSI PWM/ DIO10	both	out	RSSI indicator/ I/O
7	DIO11	both	in	Digital I/O 11
8	Reserved		disabled	Do not connect
9	DTR/DIO8/ Sleep_RQ	both	in	Sleep control or I/O8
10	**GND**	PWR	–	Ground



XBee Router or End Device (API)

Az Xbee modul kivezetései – 2. oldal

Pin	Name	dir	default	Fuction
20	AD0/DIO0	both	disabled	analog in/digital I/O 0. comm button
19	AD1/DIO1	both	disabled	analog in/digital I/O 1.
18	AD2/DIO2	both	disabled	analog in/digital I/O 2
17	AD3/DIO3	both	disabled	analog in/digital I/O 3.
16	RTS/DIO6	both	in	RTS/digital I/O 6.
15	ASC/DIO5	both	out	associated LED/DIO 5.
14	VREF	in	-	not used
13	ON/SLEEP	out	out	status indicator/DIO 9.
12	CTS/DIO7	both	out	CTS/digital I/O 7.
11	DIO4	both	disabled	digital I/O 4.

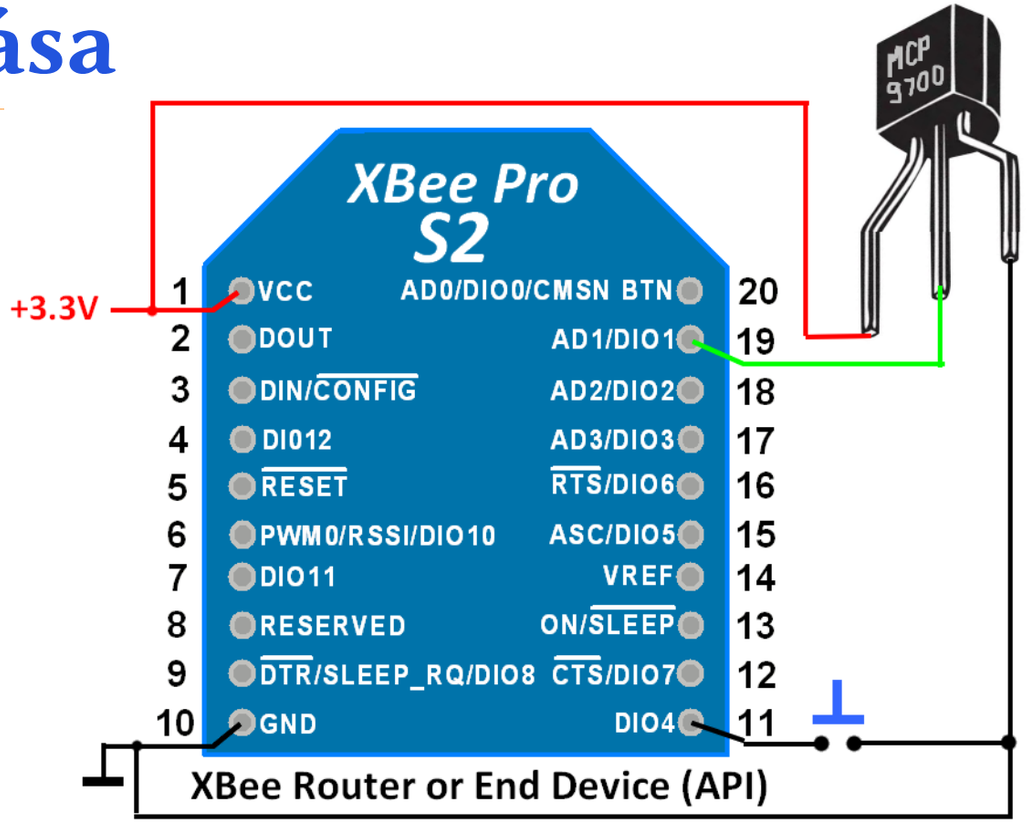
Az Xbee modul önálló alkalmazása

- Az Xbee moduloknak elég tápfeszültséget adni, hogy működőképesek legyenek
- Egy nagyon egyszerű példa az önálló működésre:
  - Az **AD1/DIO1** lábon egy **MCP9700** analóg hőmérő
  - A **DIO4** lábon egy nyomógomb
- A 10 bites ADC 1,20 V-os belső referenciát használ, így a millivoltokban mért feszültség:

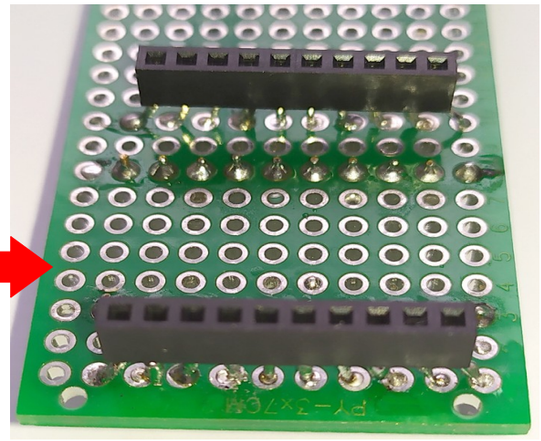
$$U[mV] = \frac{N_{ADC} \cdot 1200}{1024}$$

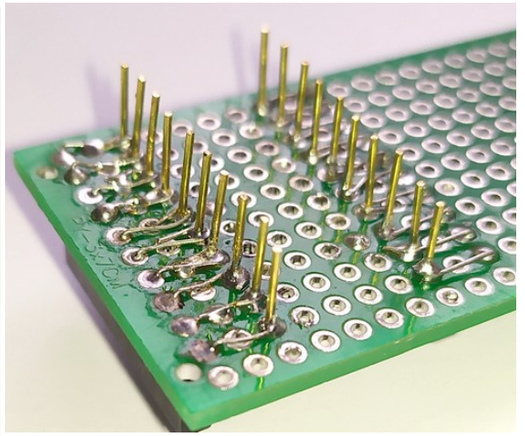
- A hőmérő 10 mV/°C érzékenységű, 500 mV nullapont-eltolással

$$T[^{o}C] = \frac{U - 500}{10}$$



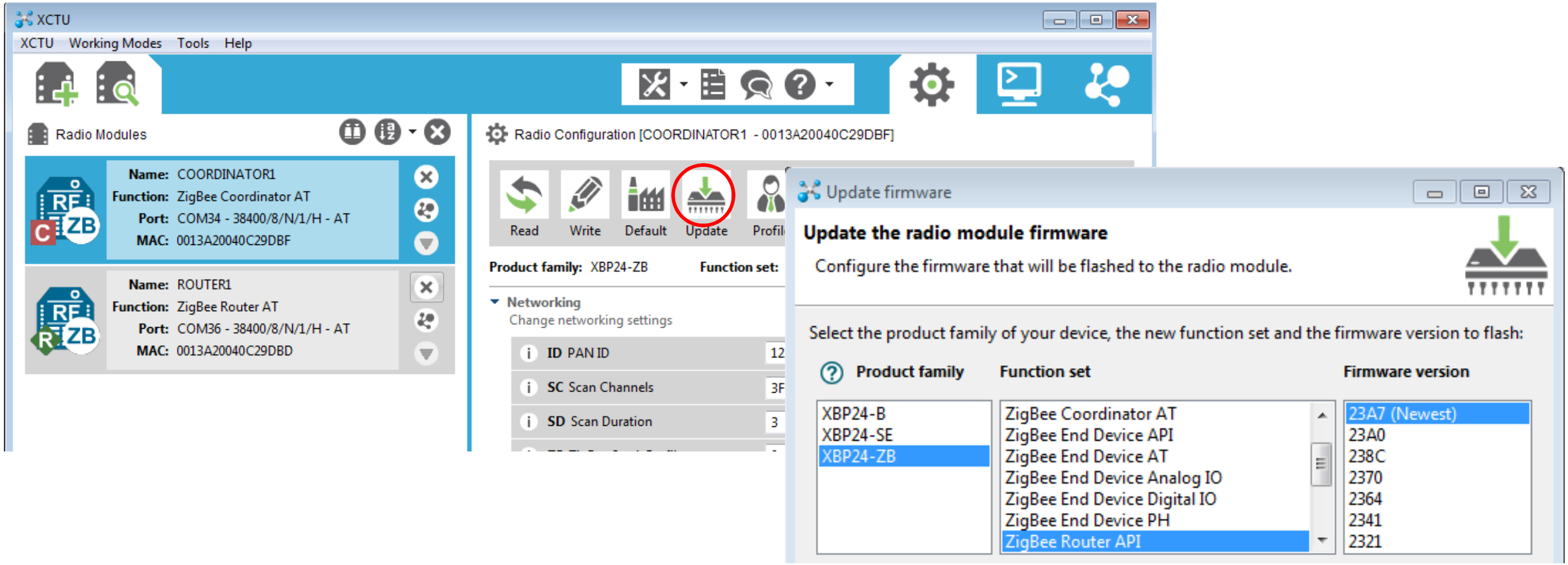
Az Xbee modul lábai 2 mm-re vannak, a dugaszolós panelhoz csak adapterrel tudjuk csatlakoztatni!





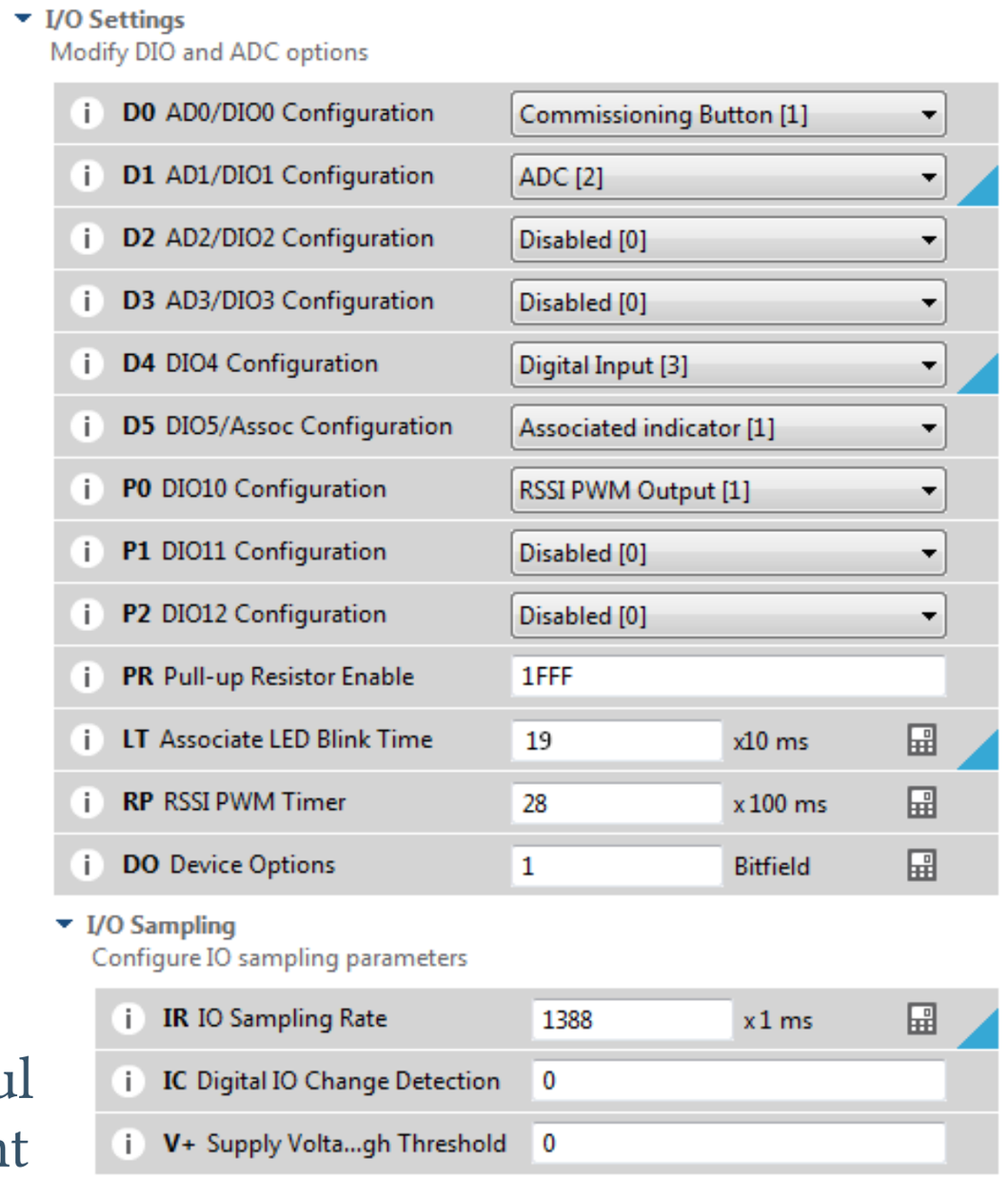
Az XBee modul konfigurálása API módba

- Az **XBee** modulok I/O lehetőségeinek kiaknázását csak az úgynevezett **API módban** tudjuk megvalósítani. Ehhez a moduljaink firmware-ét le kell cserélni az **XCTU** alkalmazás **Update** funkciójával. Az önálló modulhoz a ZigBee Router API firmware-t telepítsük



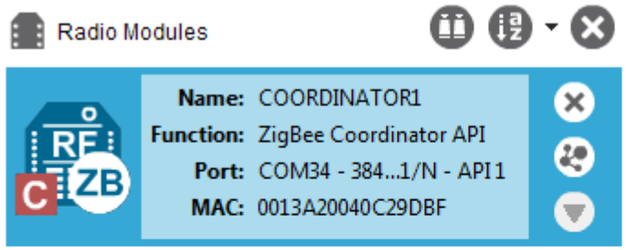
Az XBee modul konfigurálása

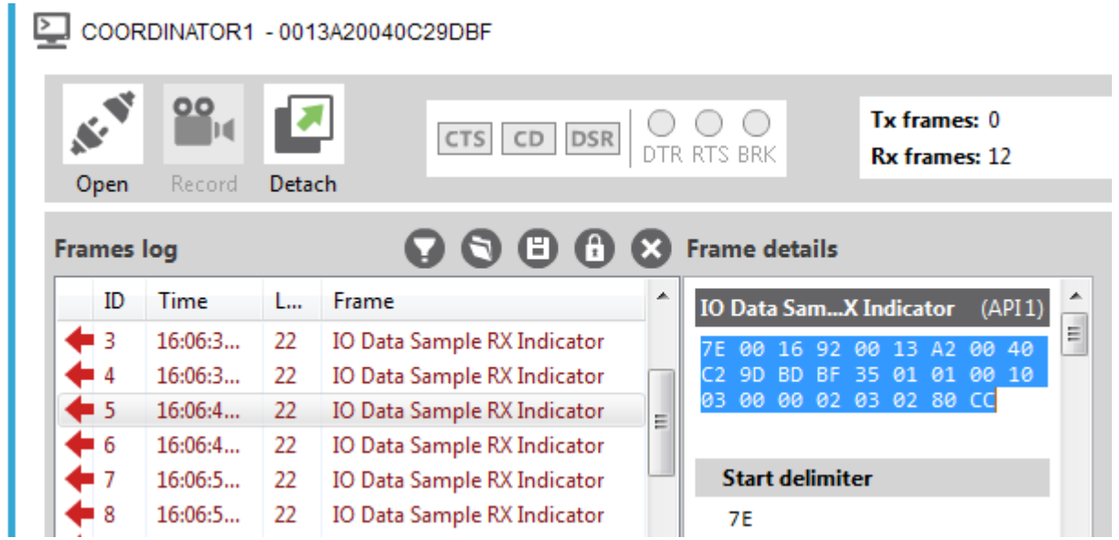
- **Javasolt beállítások:**
  - **PAN ID = 1234** (hálózat azonosító)
  - **JV** és **JN** = Enabled (1)
  - **AP** = 1 (API 1 mód)
  - **BD** =38 400 bps (baudrate)
  - **D6/D7** = Disabled (nics adatfolyam vezérlés)
- **I/O konfigurálás:**
  - **AD1** Analóg mód engedélyezés
  - **DIO4** = digitális bemenet
  - **LT** = 19 (DIO5 25x10 ms villogási idő)
  - **IR** = 1388 (I/O mintavételezés 5000 ms-onként)
- Ha **DH, DL** 0-án maradt, akkor a koordinátor modul kapja meg az I/O mintavétel eredményeit 5 s-onként



A hőmérő modul ellenőrzése az XCTU programmal

- Csatlakoztassuk a **COORDINATOR1** modult az **XCTU** program **Console** módjában, akkor a terminál ablakban láthatjuk a beérkező üzeneteket
- Az üzenetek értelmezéséhez vegyük igénybe az XCTU segítségét!





AD1 = 0x280 = 640

U = 640*1200/1024 =
= 750 mV

T = 25,0 °C

```
Jelző          64 bites forráscím     16 bit cím  dig mask dig bits      AD1   check
7E 00 16 92 00 13 A2 00 40 C2 9D BD BF 35 01 01 00 10 03 00 00 02 03 02 80 CC
   Hossz  típus     (ROUTER1)              opció mintaszám an. mask  AD0
```

Programkönyvtárak és mintaprogramok

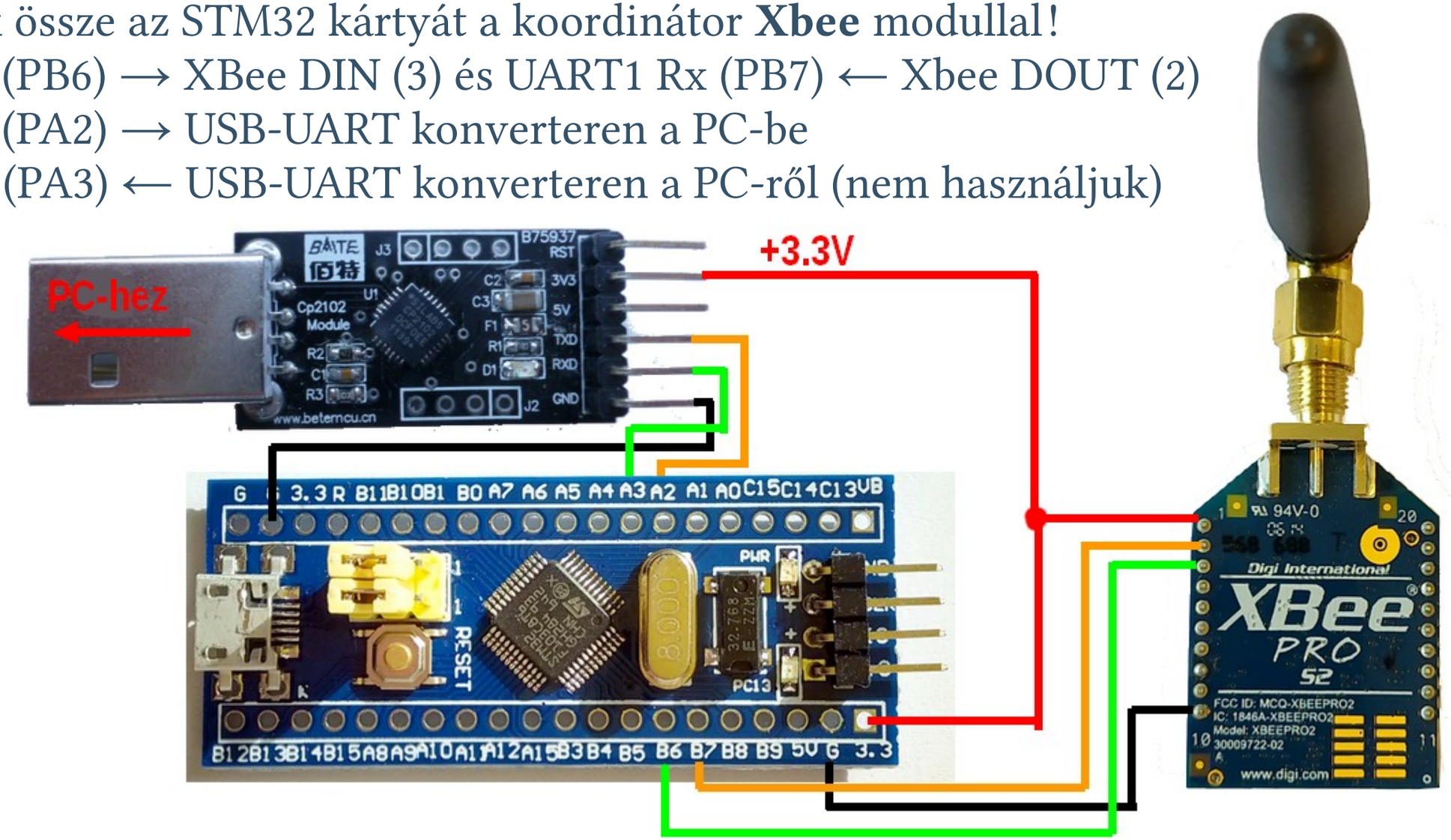
- Az előző példából láthattuk, hogy az API módú üzenetcsomagok kezelése nem egyszerű feladat, szerencsére lehet találni segédleteket hozzá.
- Támogatói programkönyvtárak
  - XBee library for Digi XBee ZB Modules in API Operation mode (STM32, mbed)
  - DIGI Xbee ANSI C library
  - DIGI Xbee Java library
  - XBee-api Node.js module
  - Xbee Arduino library
- Hasznos könyvek
  - Matthijs Kooijman: Building Wireless Sensor Networks Using Arduino
  - Robert Faludi: Building Wireless Sensor Networks: With Zigbee, Xbee, Arduino, And Processing
- Hasznos cikkek:
  - DigiKey (Scott Schmit): XBee API Mode - Read Remote ADC Example (PSOC 4)
  - Steven J. Erdmanczyk Jr.: XBee API Mode Tutorial Using Python and Arduino

serial_dump

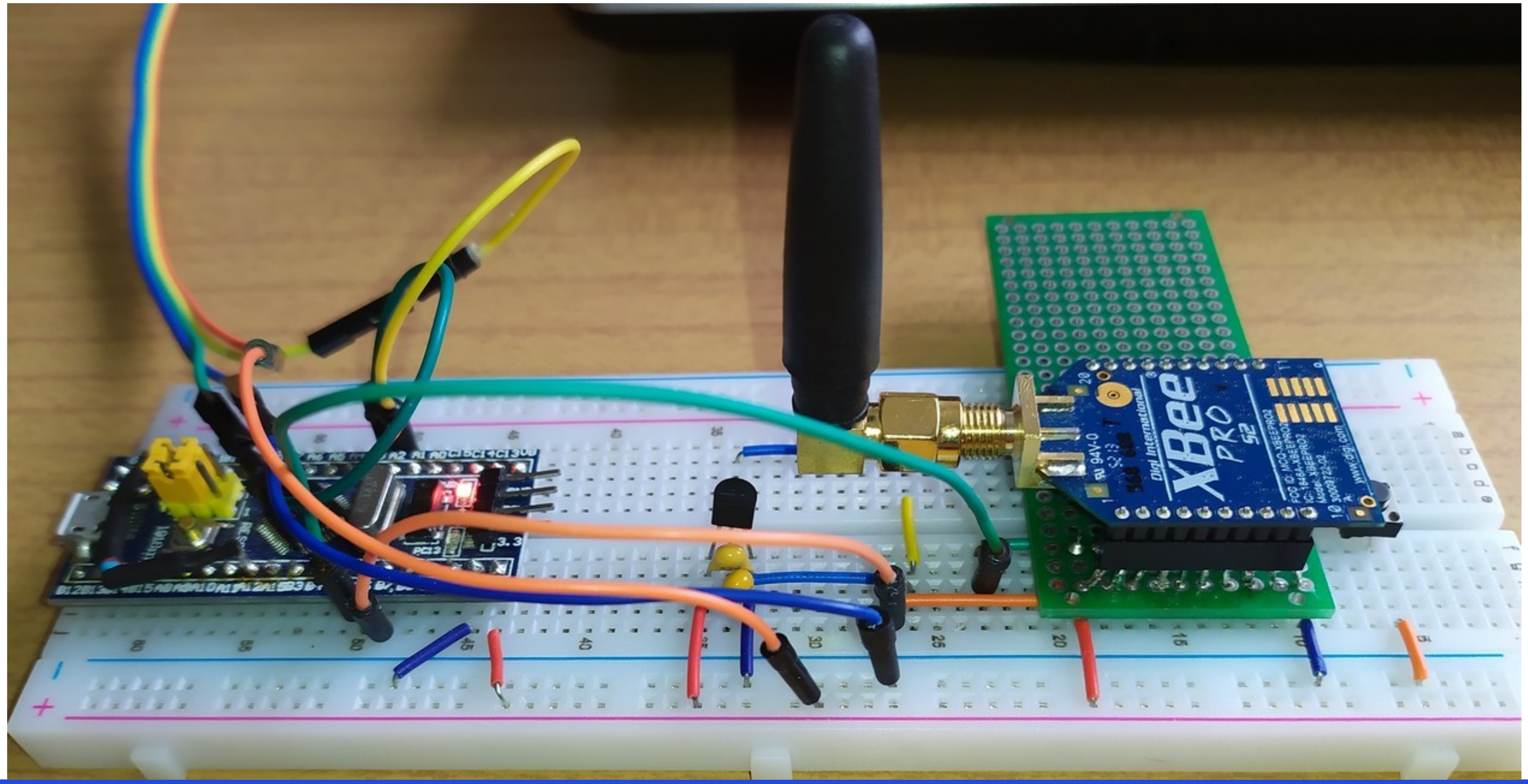
read2adc

Kísérleti elrendezés

- Kapcsoljuk össze az STM32 kártyát a koordinátor **Xbee** modullal!
  UART1 Tx (PB6) ⟶ XBee DIN (3) és UART1 Rx (PB7) ⟵ Xbee DOUT (2)
  UART2 Tx (PA2) ⟶ USB-UART konverteren a PC-be
  UART2 Rx (PA3) ⟵ USB-UART konverteren a PC-ről (nem használjuk)

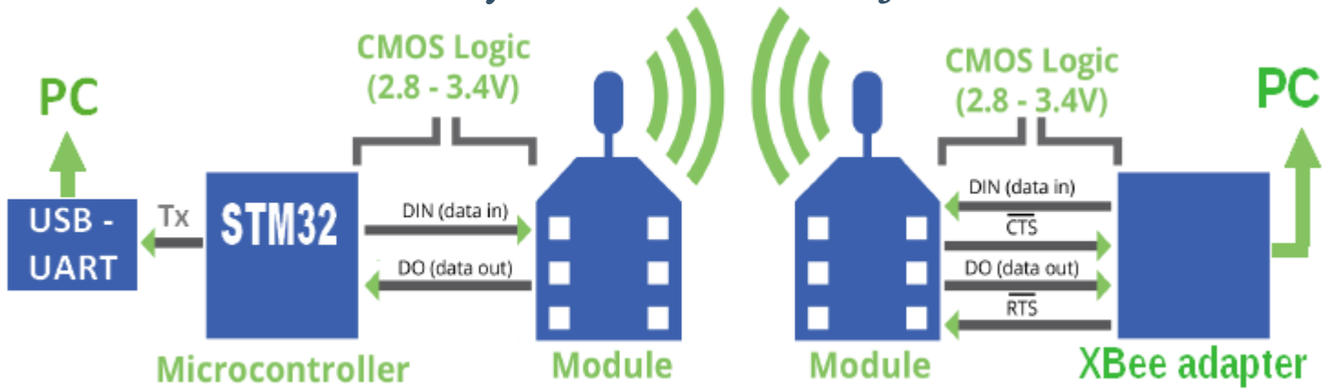


Kísérleti elrendezés



serial_dump

- Az alábbi program az **STM32** mikrovezérlőhöz kapcsolódó **XBee** modul által fogadott üzenetcsomagokat kilistázzuk, mellyel ellenőrizhetjük a kommunikáció működését



- A főprogramhoz *Mathijs Kooijman: Building Wireless Sensor Networks Using Arduino* c. könyve első fejezetének hasonló nevű mintapéldáját adaptáltuk.
- A soros portok kezelését a korábbi előadásokban bemutatott programokból emeltük át:
  - **UART1** kezelését a 2019. november 21-i előadás **uart_irq** mintaprogram mutatta be
  - **UART2** kezelését a 2019. november 7-i és előadás **Program04_1 és Program04_2** mutatta be
- Az **UART1** soros port kezelését néhány apróbb függvénnyel ki kellett egészíteni:
  - int **IsAvailable**(void) – a visszaadott szám jelzi, hogy van-e beérkezett karakter a bufferben
  - int **PeekKey**(void) – megmondja, hogy mi lesz a következő karakter a bemeneti bufferben

serial_dump/main.c

```
#include "stm32f10x.h"
#include <ctype.h>
extern void buffer_Init (void);                 // Ki- és bemeneti bufferek inicializálása
extern void USART1_init (void);                 // USART1 és a PB6, PB7 kivezetések inicializálása
extern  int IsAvailable(void);                  // Van beérkezett karakter?              Új függvény
extern  int PeekKey (void);                     // Mi lesz a következő karakter?         Új függvény
extern  int GetKey (void);                      // Karakter beolvasása (-1, ha nincs beérkezett)
extern  int SendChar (int c);                   // Karakter kiírása (-1, ha betelt a buffer)
extern void USART2_init (void);                 // USART2 és a PA2, PA3 kivezetések inicializálása
extern void USART2_write (int ch);              // Karakter kiírása (blokkoló függvény)
extern void USART2_putstring(char* s);          // Karakterfüzér kiírása                 Új függvény
extern void setTimeout(int n);                  // Homokóra indítása (1 - 1800 ms)       Új függvény
extern  int TimedOut(void);                     // Homokóra lejárt már? (0: nem 1: igen) Új függvény
int hexadigits[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F',};
#define data_size    8                          // Ennyi bájtot írunk ki egy sorban

int main (void) {
  int column, b, data[data_size];
  buffer_Init();                                // RX/TX bufferek inicializálása
  USART1_init();                                // XBee_UART konfigurálása (38400 bps)
  USART2_init();                                // PC_UART konfigurálása (9600 bps)
  __enable_irq();                               // Megszakítások globális engedélyezése
```

serial_dump/main.c

```
  while (1) {                                   // végtelen ciklus
    if (IsAvailable()) {
      for (column = 0; column < data_size; ++column) {
        setTimeout(1000);                       // Legfeljebb 1 másodpercig várunk
        while (!IsAvailable() && !TimedOut()) {}
        if (!IsAvailable())
          break;
        if (column && PeekKey() == 0x7E)        // Új csomag kezdetén új sort kezdünk
          break;
        b = GetKey();                           // Egy bájt beolvasása és kiíratása
        data[column] = isprint(b) ? b : '.';
        USART2_write(hexadigits[b >> 4]);
        USART2_write(hexadigits[b & 0xF]);
        USART2_write(' ');
      }

      for (int i=column; i < data_size; ++i)
        USART2_putstring("   ");  // Maradék sor feltöltése

      // Befejezzük a sort a nyomtatható karakterek kiírásával
      USART2_write(' ');
      for(int i=0; i<column; i++)
        USART2_write(data[i]);
      USART2_putstring("\r\n");
    }
  }
}
```

Egy tipikus kiírás pl. így néz ki:

```
7E 00 19 90 00 13 A2 00  ~.......
40 D8 5F 9D 00 00 02 48  @._....H
65 6C 6C 6F 2C 20 57 6F  ello, Wo
72 6C 64 21 3B           rld!;
```

Uart1.c bővítése

- Emlékeztető: **UART1** kezelése megszakítással, és bufferelléssel történik
- Az új függvényeket a **Getkey()** függvény mintájára készítettük el
- **PeekKey()** csak abban különbözik tőle, hogy a kiolvasott karaktert nem vesszük ki a bufferből, azaz nem léptetjük a **p->out** kimeneti mutatót
- **IsAvailable** egyszerűen a be- és kimeneti mutatók különbségét adja vissza a **SIO_RBUFLEN** makró segítségével: `(rbuf.in - rbuf.out)`

```
/*-------------------------------------------------------
  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)]);
}

/*-------------------------------------------------------
  PeekKey - Peek a character, but do nat remove from buffer
 *-------------------------------------------------------*/
int PeekKey (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)]);
}

/*-------------------------------------------------------
  IsAvailable - chech if there is any character recived
 *-------------------------------------------------------*/
int IsAvailable(void) {
  return SIO_RBUFLEN;     // Nullát ad, ha nincs beérkezett karakter
}
```

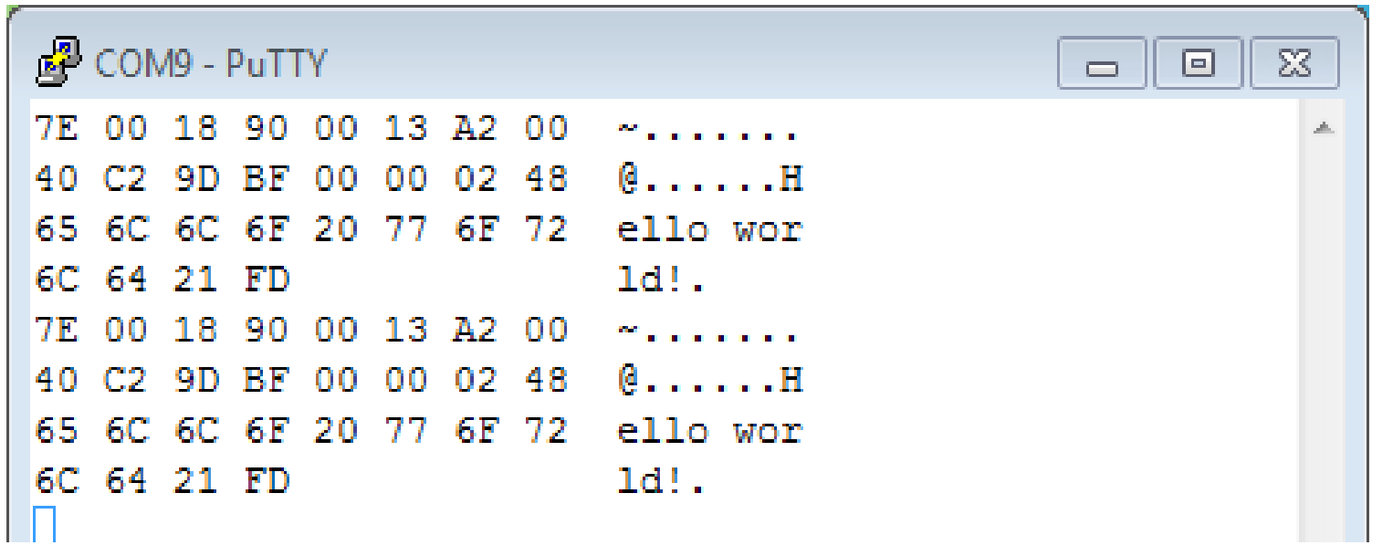
Uart2.c bővítése

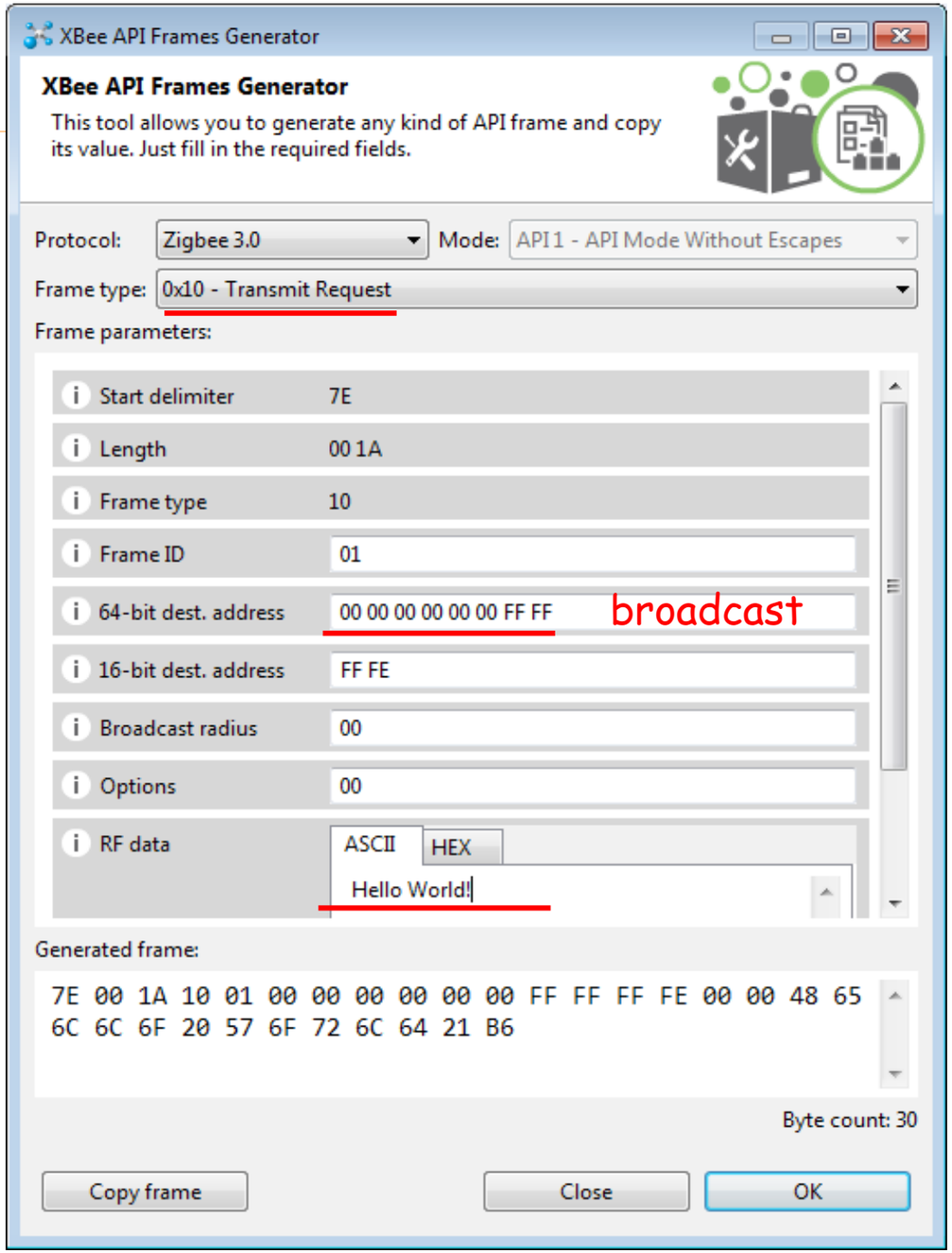
- **USART2** kezelését a korábbi programokból at Uart2.c állományba gyűjtöttük össze
- Az új függvényeket itt láthatjuk
- **USART2_putstring(char* s)** egy mutatót vár (vagy egy idézőjelek közé zárt szöveget). Addig ír, amíg nulla értékű bájtot nem talál
- **setTimeout(int n)** elindítja a **SysTick** számlálót (fCPU/8 frekvencia választással)
- **int TimedOut()** lekérdezi a **COUNT** jelzőbit állapotát és leállítja a számlálót, ha letelt a beállított idő

```
//-- Nullával lezárt szöveg kiírása ------
void USART2_putstring(char* s) {
  char c;
  while(c=*s++) USART2_write(c);
}

//-- A SysTick számlálóval kialakított homokóra segítségével
//-- korlátozott idejű várakozásokat szervezhetünk. A várakozási
//-- ciklusban csak rápillantunk, hogy letelt-e már az idő...
//-- Homokóra indítása --------------------
void setTimeout(int n) {
  SysTick->LOAD = SystemCoreClock/8000*n-1; // Újratöltési érték
  SysTick->VAL  = 0;                        // Számláló törlése
  SysTick->CTRL = 0x1;                      // Engedélyezés, nincs
                                            // interrupt, fCPU/8
}

//-- Homokóra állapotának lekérdezése -----
int TimedOut(void) {
  int flag = 0;
  if((SysTick->CTRL & 0x10000) != 0)  {     // COUNT flag ellenörzés
    SysTick->CTRL = 0;                      // SysTick leállítása
    flag = 1;
  }
  return flag;
}
```

serial_dump – 1. kísérlet

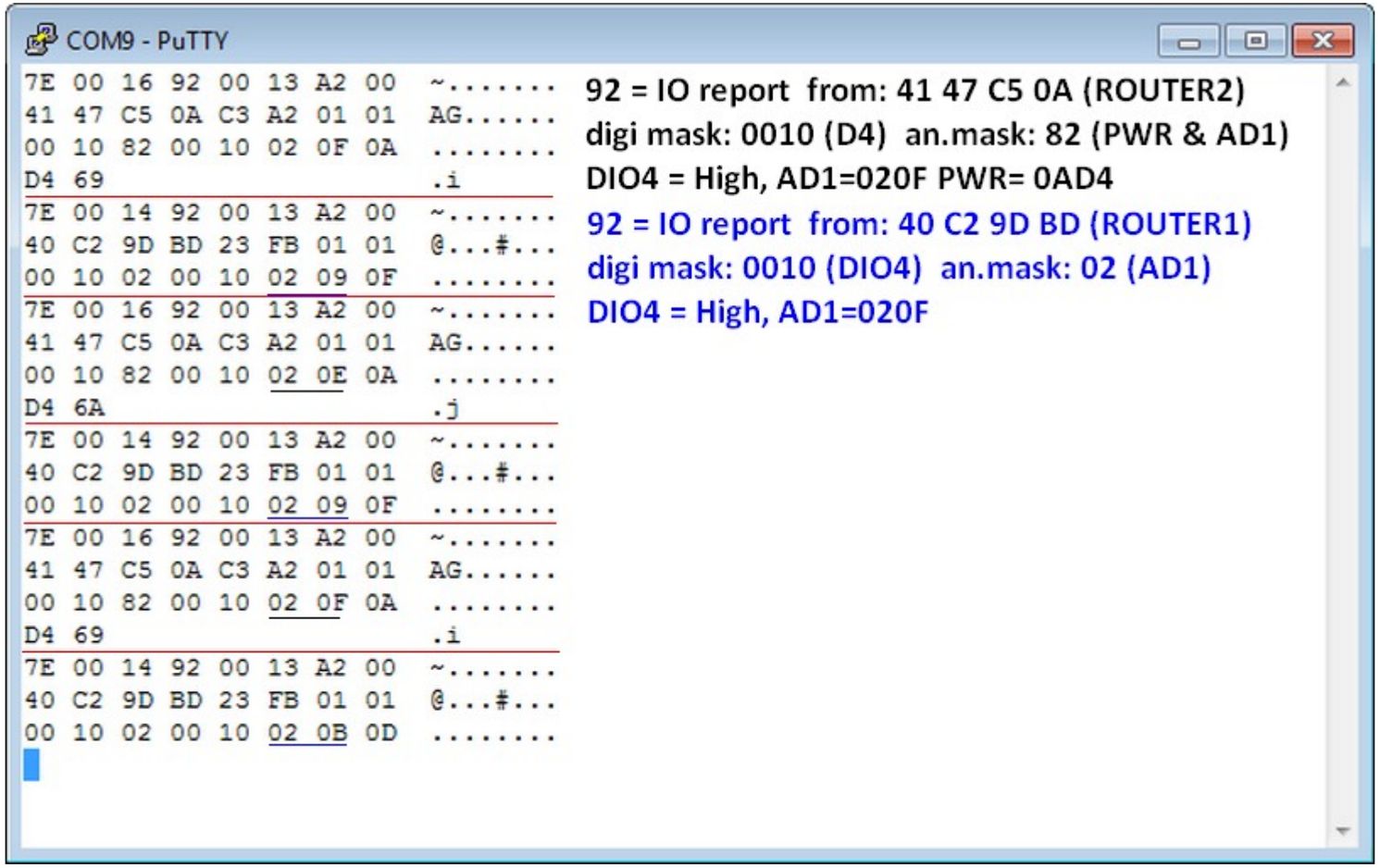
- Az alábbi példában a **ROUTER1** modult kötöttük az **STM32** mikrovezérlőhöz, tehát azokat az üzeneteket látjuk, amelyeket ez a modul kap
- A **COORDINATOR1** modult egy **XBee adapter** kártya segítségével a PC-hez kötöttük és az **XCTU** program használatával küldtünk ki 0x10 típusú (Transmit Request) üzeneteket *Hello world!* tartalommal
- Itt láthatjuk a **serial_dump** futási eredményt:





serial_dump – 2. kísérlet

- A második kísérletnél a **COORDINATOR1** modult kötöttük az **STM32**-re, a **ROUTER1** és **ROUTER2** modulok pedig csak tápfeszültséget kaptak és az automatikusan és periodikusan küldött I/O jelentéseiket láthatjuk
- **ROUTER1** a **DIO4** digitális bemenet állapotáról, valamint az **AD1** analóg bemenet állapotáról küldött jelentéseket
- **ROUTER2** a fentieken kívül a tápfeszültségről is küldött jelentéseket

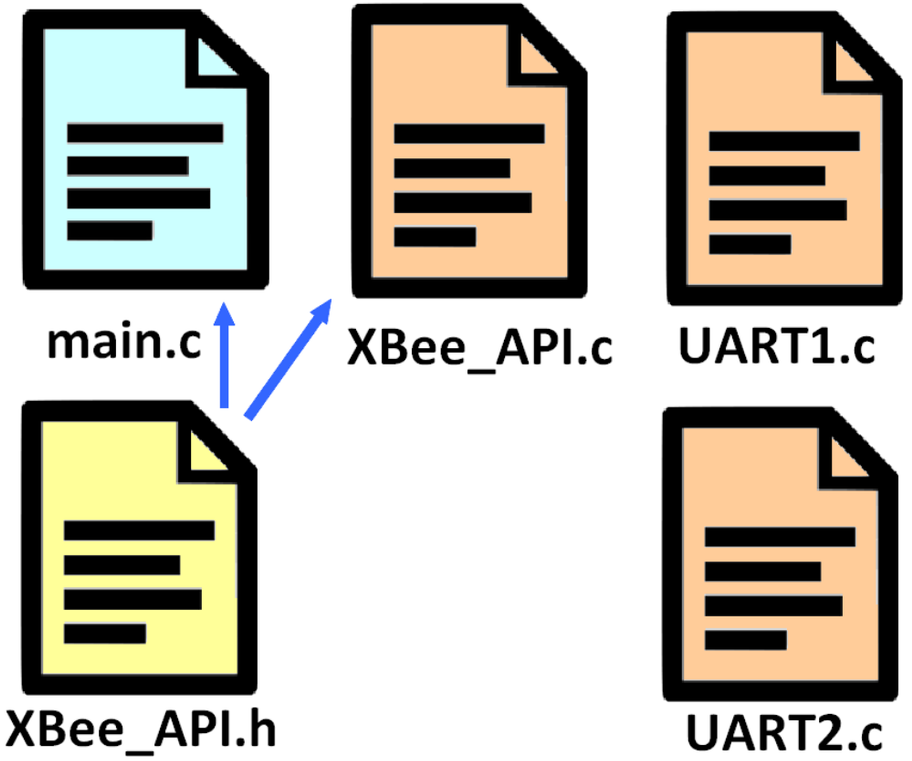


Távoli ADC-k adatainak lekérése

- A legutóbbi kísérlethez hasonlóan most is önálló eszközként használt XBee modulok analóg bemenetét vizsgáljuk, de nem előre ütemezett mintavételezés adataira várunk, hanem "távoli AT parancs" üzenetekkel kérjük le az adatokat
- Program eredetileg **PSOC 4** mikrovezérlőre készült és a **DigiKey** oldalán bukkantunk rá (Scott Schmit: XBee API Mode - Read Remote ADC Example), ezt írtuk át STM32-re, Keil MDK5 alá
- A kapcsolási elrendezés ugyanaz, mint a korábbi kísérletnél:
  - Kapcsoljuk össze az **STM32** kártyát a koordinátor **XBee** modullal!
  - UART1 Tx (PB6) ⟶ XBee DIN (3)
  - UART1 Rx (PB7) ⟵ XBee DOUT (2)
  - UART2 Tx (PA2) ⟶ USB-UART konverteren a PC-be
  - UART2 Rx (PA3) ⟵ USB-UART konverteren a PC-ről (nem használjuk)
- A másik két **XBee** modul kapjon valamilyen mérendő jelet az **AD0** bemenetre (pin 20) és kapjanak tápfeszültséget

A program felépítése

- Az **XBee** modulok API módú kezelését (üzenetcsomagok összerakása, checksum számítás, csomagok kiküldése, a válasz figyelése, a szükség szerinti újraküldések és a timeout figyelése) az **XBee_API.c** állomány tartalmazza. Egyelőre hiányos, csak néhány csomagtípust kezel
- A main függvény a hardver inicializálások után:
  - **Lokális AT** paranccsal **API1** módba kapcsolja a koordinátor modult (**0x08/ AP 1**)
  - **Távoli AT** paranccsal analóg módba állítja a router modulok **AD0** bemenetét (**0x17/ D0 2**)
  - **Távoli AT** paranccsal periodikusan lekérdezi az ADC konverziók eredményét (**0x17/ IS**), majd kiírja az **UART2** porton
- A soros portok kezelése az **UART1.c** és **UART2.c** állományokban található, ugyanúgy, mint az előző programban



read2adc/main.c

```
#include <XBee_API.h>

const char dash_line[] = "\n\r-----------------------------\n\r";
const char startup_header[] = "\n\rXBee API Example\r\n";
const char adc_header[] = ""\n\rNode 1 Node 2\n\r"";
const uint8 router_1_id[] = {0x00,0x13,0xA2,0x00,0x40,0xC2,0x9D,0xBD};
const uint8 router_2_id[] = {0x00,0x13,0xA2,0x00,0x41,0x47,0xC5,0x0A};
char adc_string[] = {0,0,0,0,109,86}; // 0000mV
uint16 ms_count = 0;
uint8 timeout_flag = 0;

struct API_Frame_t API_Tx_Frame;
struct API_Frame_t API_Rx_Frame;

void sys_start (void);              // Koordinátort API1 módba állítja
void config_adc (uint8 node_id);    // Távoli modulok AD0 konfigurálása
void read_adc (uint8 node_id);      // Távoli ADC-k lekérdezése
void Set_Decade (uint16 val);       // Az eredmények kiíratása
uint8 Convert_to_ASCII (uint8 val);
```

```
int main() {
  sys_start();
  config_adc(ROUTER_1);
  config_adc(ROUTER_2);
  PC_UartPutString(adc_header);

  for(;;) {
    PC_UartPutChar('\r');
    read_adc(ROUTER_1);
    PC_UartPutChar(' ');
    read_adc(ROUTER_2);
    setTimeout(1000);
    while(!TimedOut()) {}
  }
}
```

read2adc/main.c – inicializálás

- **USART1** és **USART2** inicializálása után engedélyezzük a megszakításokat **USART1** számára
- Kiíratjuk a fejléceket. **PC_UartPutString** valójában **USART2_putstring**-et hívja meg – **XBee_API.h** tartalmazza az átnevezést biztosító makrókat
- A **lokális AT parancs** kiadásához csak az **AP 1** adatokat kell megadni

```
void sys_start(void) {
  uint8 payload[MAX_DATA_BYTES];
  uint8 payload_size = 0;
  USART1_init();
  USART2_init();
  __enable_irq();
  // Print start headers in PC terminal
  PC_UartPutString(dash_line);
  PC_UartPutString(startup_header);
  // Enable API mode operation on local module
  // send local "AP1" command
  payload[payload_size++] = 'A';
  payload[payload_size++] = 'P';
  payload[payload_size++] = 0x1;

  Build_API_Tx_Frame(FRAME_TYPE_AT_COM, payload, payload_size);
  Tx_Request();  // Parancs kiküldése és a válasz fogadása
}
```

Lokális AT parancs:
API 1 mód beállítása

read2adc/main.c – ADC-k konfigurálása

```
void config_adc (uint8_t node_id) {
  uint8_t i;
  uint8_t payload[MAX_DATA_BYTES];
  uint8_t payload_size = 0;
  for(i=0; i<8; i++) {              // load the 64-bit destination address
    if(node_id == ROUTER_1) {
      payload[i] = router_1_id[i];
    }
    if(node_id == ROUTER_2) {
      payload[i] = router_2_id[i];
    }
    payload_size++;
  }
  payload[payload_size++] = 0xFF; // set 16-bit network address to "unknown"
  payload[payload_size++] = 0xFE;
  payload[payload_size++] = 0x02; // set Activate Changes bit
  payload[payload_size++] = 'D';  // set ADC0 as analog input
  payload[payload_size++] = '0';
  payload[payload_size++] = 0x2;
  Build_API_Tx_Frame(FRAME_TYPE_REMOTE_AT_COM, payload, payload_size);
  Tx_Request();                     // Parancs kiküldése és a válasz fogadása
}
```

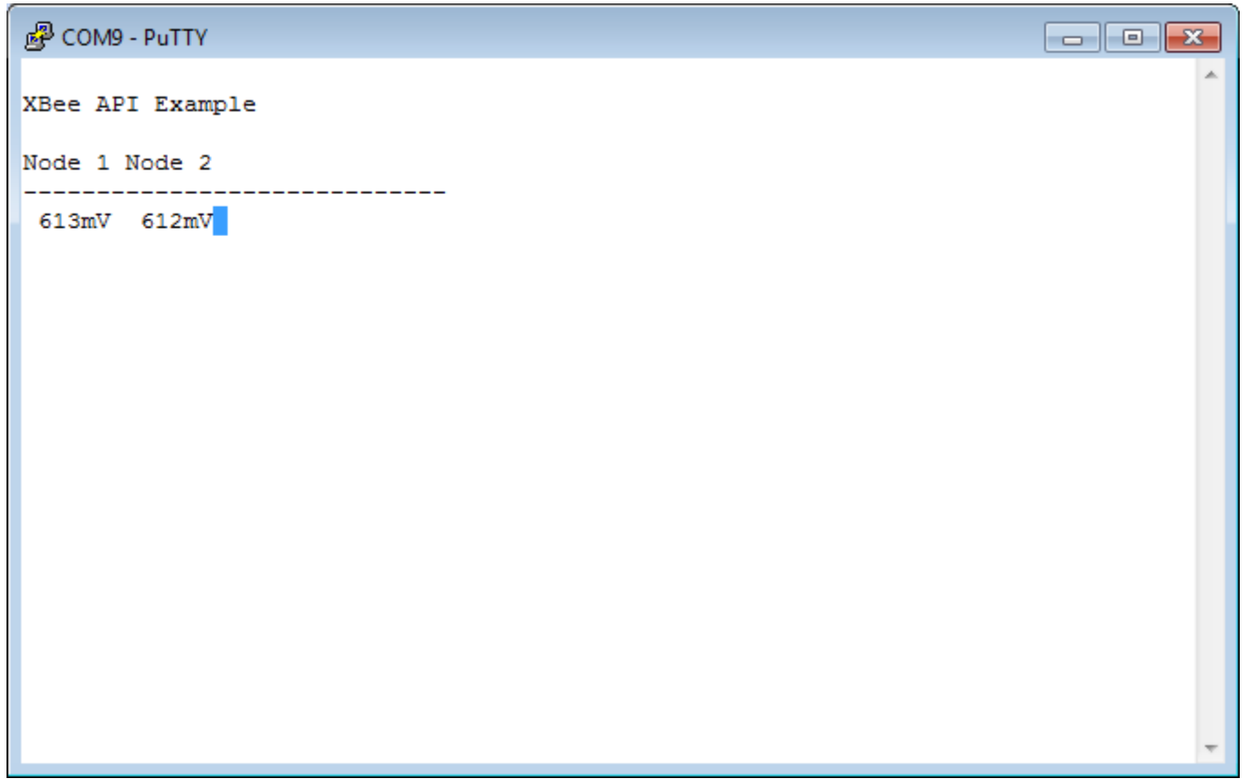
Távoli AT parancs: D0 2 kiküldése
(az AD0 analóg bemenet engedélyezése)
Az opció bájt AC bitjének 1-be állítása
külön parancs nélkül aktiválja a módosítást

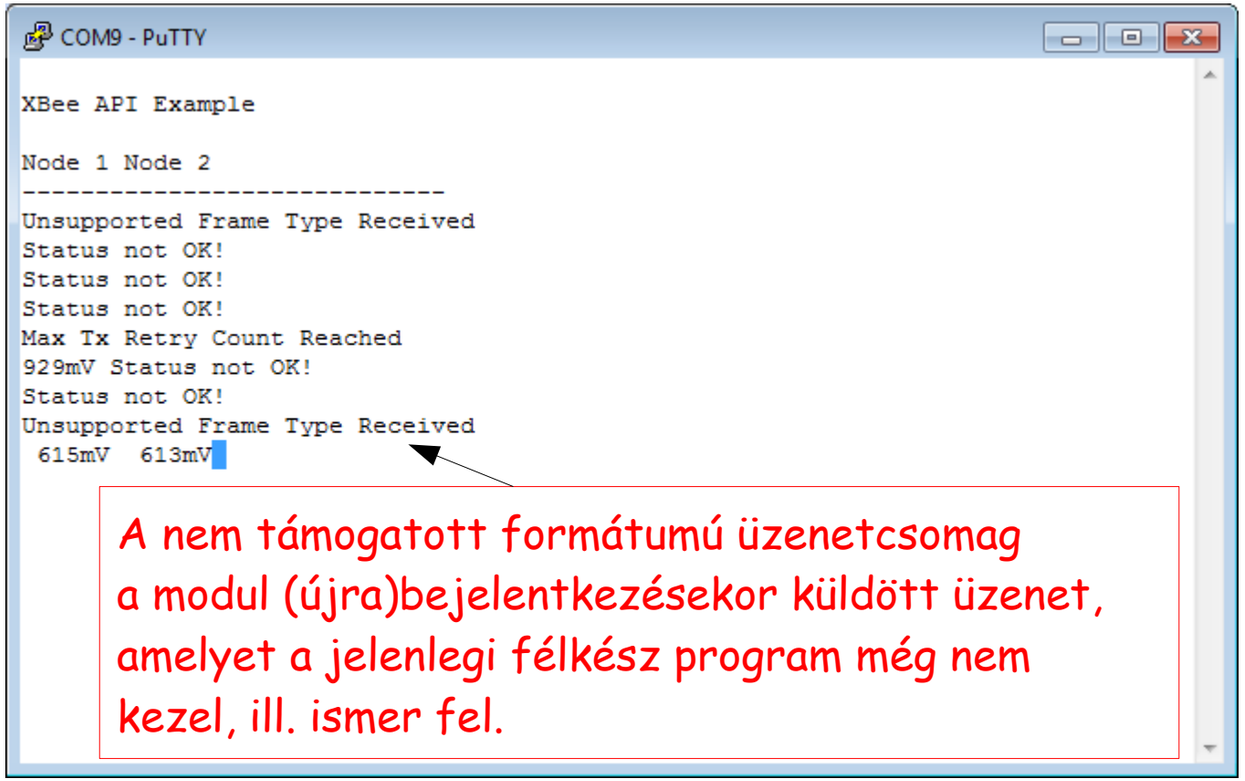
read2adc/main.c – ADC-k lekérdezése

```
void read_adc (uint8_t node_id) {
  uint8_t i, adc_result, payload[MAX_DATA_BYTES], payload_size = 0;
  for(i=0; i<8; i++) { // load the 64-bit destination address
    if(node_id == 1) payload[i] = router_1_id[i];
    if(node_id == 2) payload[i] = router_2_id[i];
    payload_size++;
  }
  payload[payload_size++] = 0xFF; // set 16-bit network address to "unknown"
  payload[payload_size++] = 0xFE;
  payload[payload_size++] = 0x02; // set AC bit
  payload[payload_size++] = 'I';  // set ADC0 as analog input
  payload[payload_size++] = 'S';
  Build_API_Tx_Frame(FRAME_TYPE_REMOTE_AT_COM, payload, payload_size);
  if(!Tx_Request()) {
    adc_result = (API_Rx_Frame.frame_data[API_Rx_Frame.length-4] << 8) & 0xFF00;
    adc_result |= (API_Rx_Frame.frame_data[API_Rx_Frame.length-3]) & 0x00FF;
    adc_result = adc_result*1200/1023;
    Set_Decade(adc_result);
    for(i=0; i<6; i++) {
      PC_UartPutChar(adc_string[i]);
    }
  }
}
```

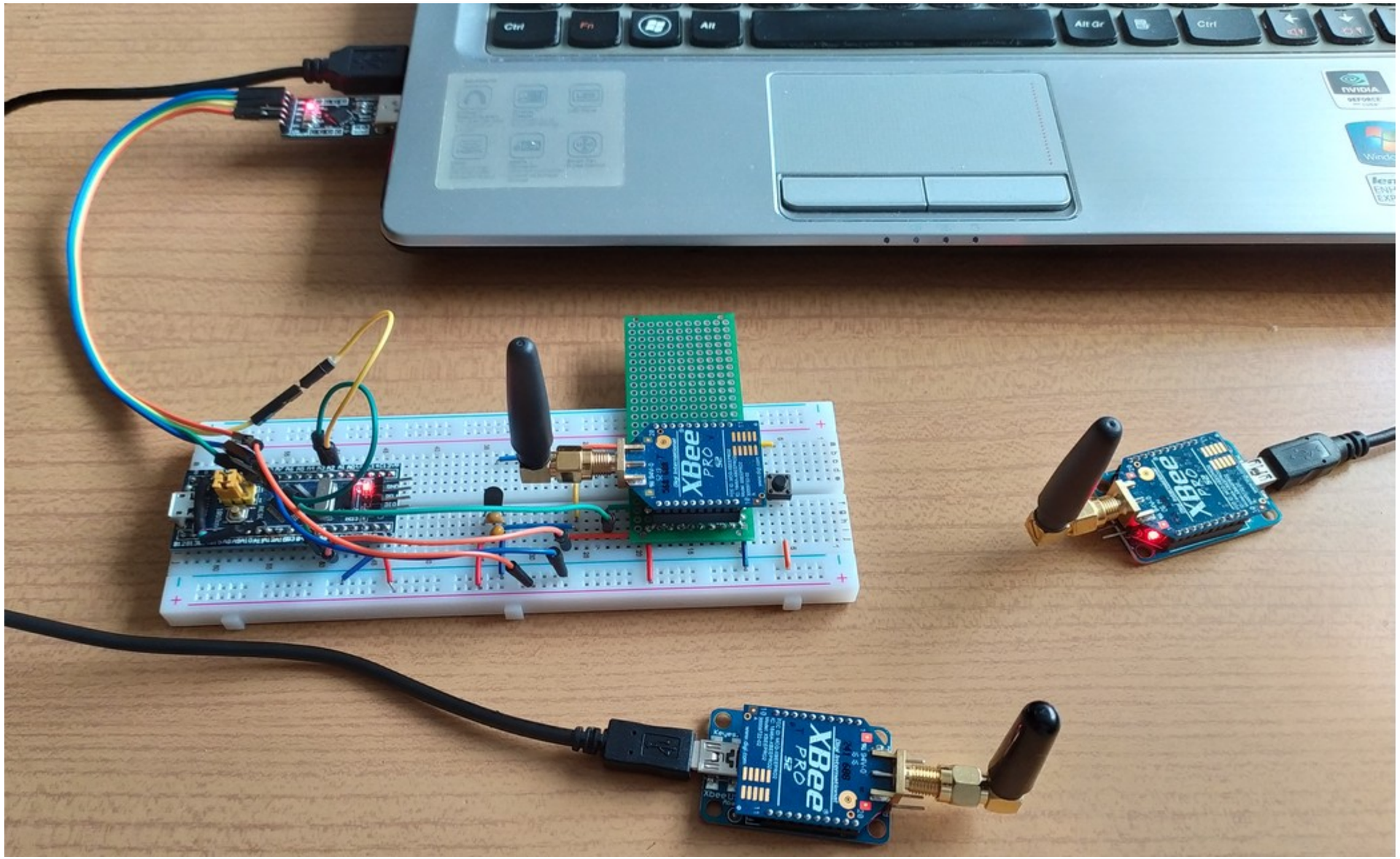
A read2adc program futási eredménye

- A program futási eredménye az alábbi ábrán látható
- A baloldali ábrán a hibátlan futás eredményét látjuk, az adat másodpercenként felülíródik
- A jobboldali ábra azt az esetet mutatja, amikor a ROUTER1 modult ideiglenesen eltávolítottuk, majd újra csatlakoztattuk (Status not OK, ill. Max Tx retry Count Reached)





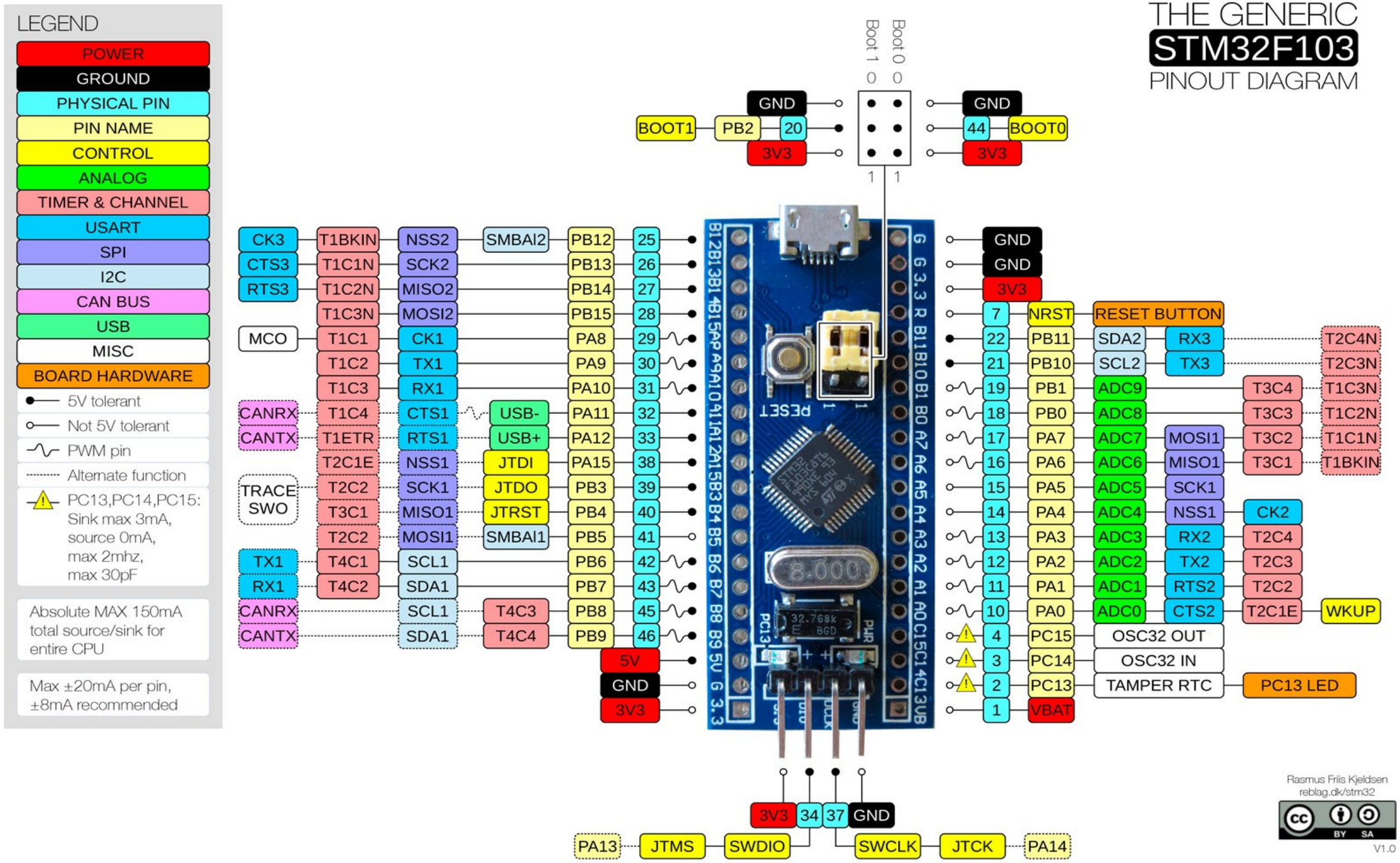
A kísérleti elrendezés



XBee_API.h

```
#include "stm32f10x.h"
extern void setTimeout(int n);
extern int TimedOut(void);
extern buffer_Init(void);
extern void USART1_init (void);
extern int GetKey (void);
extern int SendChar (int c);
#define XBee_UartWrite(x)   SendChar(x)
#define XBee_UartGetByte()  GetKey()
extern void USART2_init (void);
extern void USART2_write (int ch);
extern void USART2_putstring(char* s);
#define PC_UartPutString(x) USART2_putstring(x)
#define PC_UartPutChar(x)   USART2_write(x)
#define ROUTER_1 1
#define ROUTER_2 2
#define MAX_DATA_BYTES 100
#define MAX_TX_RETRY_COUNT 25
#define TIMEOUT_PERIOD 500 // ms
```

```
// Supported API Frame Types
#define FRAME_TYPE_AT_COM 0x08
#define FRAME_TYPE_REMOTE_AT_COM 0x17
#define FRAME_TYPE_AT_COM_RESPONSE 0x88
#define FRAME_TYPE_REMOTE_COM_RESPONSE 0x97
struct API_Frame_t {
  uint16_t length;
  uint8_t frame_type;
  uint8_t frame_id;
  uint8_t frame_data[MAX_DATA_BYTES];
  uint8_t check_sum;
};
extern struct API_Frame_t API_Tx_Frame;
extern struct API_Frame_t API_Rx_Frame;
// Public functions
void Build_API_Tx_Frame (uint8_t frame_type, uint8_t
*payload, uint8_t payload_size);
uint8_t Tx_Request (void);
uint8_t Calc_Checksum (struct API_Frame_t frame);
void Send_API_Frame (void);
uint8_t Wait_Response (void);
```


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
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