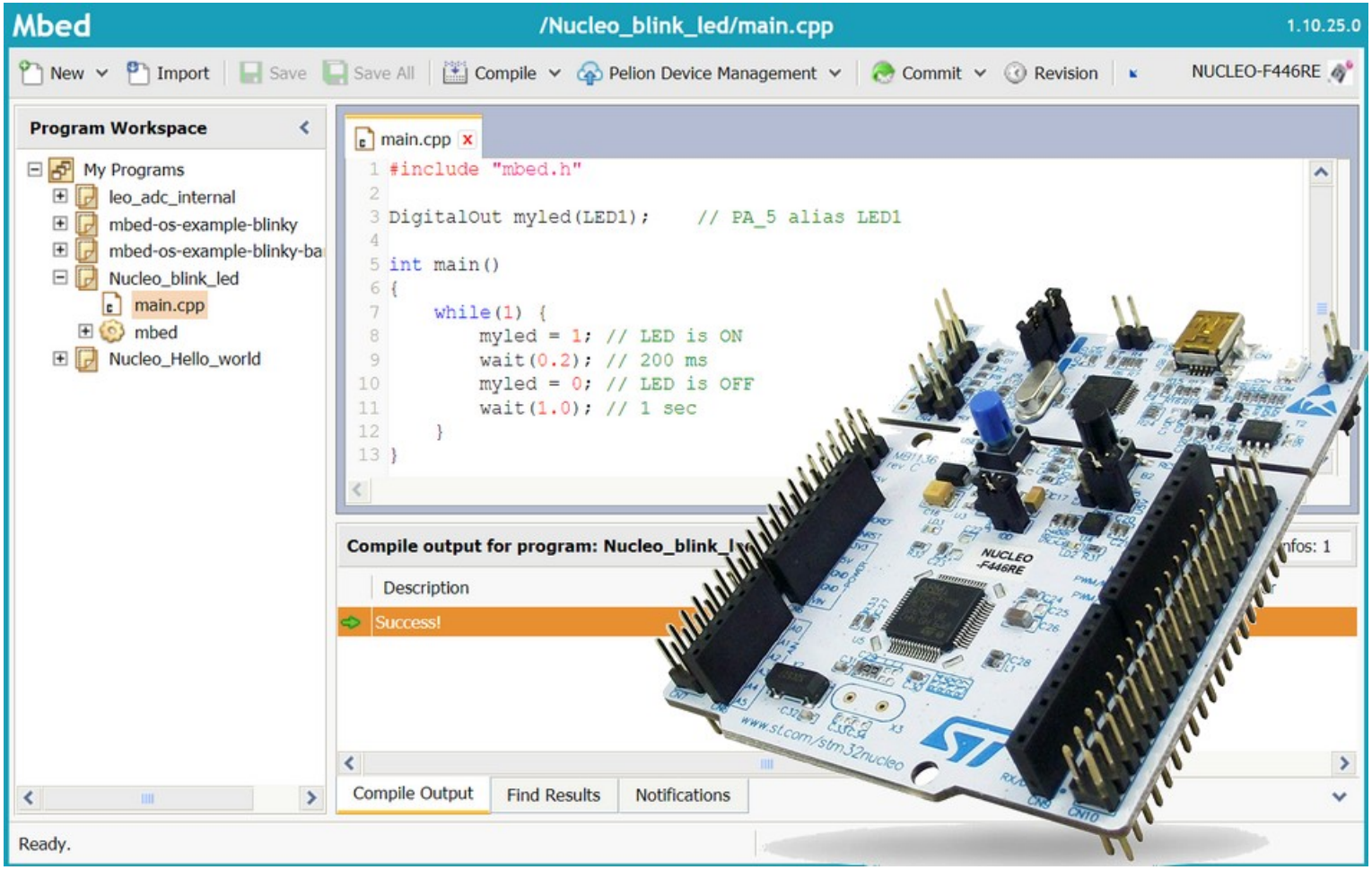
STM32 mikrovezérlők programozása ARM mbed környezetben



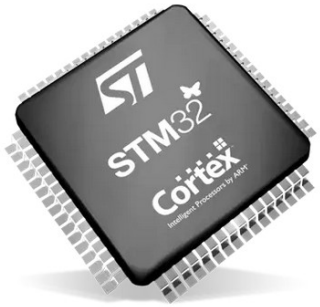
5. Soros kommunikáció (UART, SPI, I2C)

Felhasznált és ajánlott irodalom

- Cserny István: A FRDM-KL25Z kártya programozása mbed környezetben
- Rob Toulson and Tim Wilmhurst:
  Fast and Effective Embedded Systems Design: Applying the ARM mbed
- Perry Xiao: Designing Embedded Systems and the Internet of Things (IoT) with the ARM mbed
- Dogan Ibrahim: ARM-based Microcontroller Projects Using mbed
- **ARM mbed honlap: https://os.mbed.com/**
  - ARM mbed Compiler: https://ide.mbed.com/compiler/
  - ARM mbed 2 Handbook (elavult): https://os.mbed.com/handbook/Homepage
  - ARM mbed 2 Cookbook (elavult): https://os.mbed.com/cookbook/Homepage
  - ARM mbed forráskód: https://github.com/ARMmbed/mbed-os

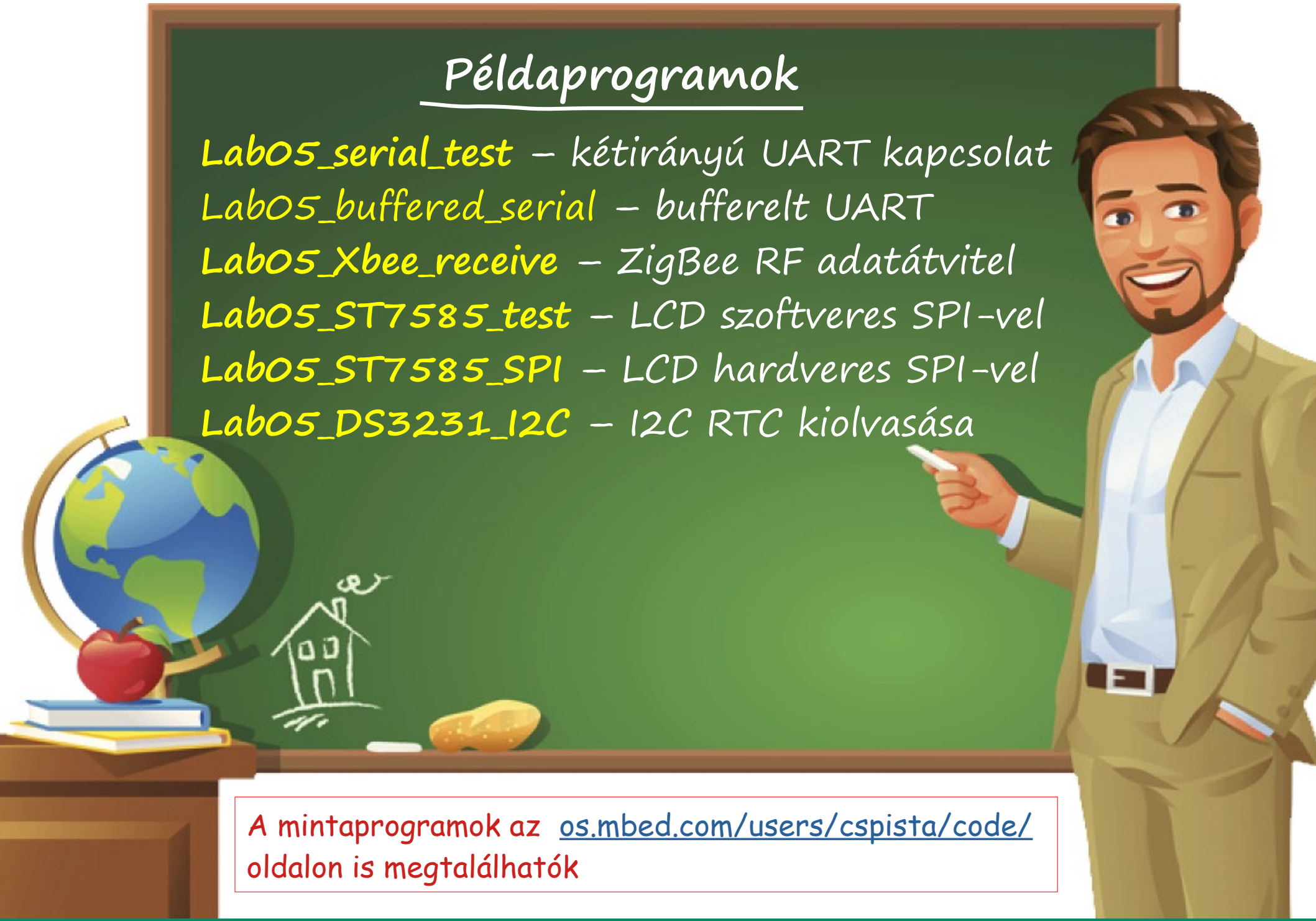
Adatlapok:

- **STM32F446RE** adatlap és termékinfo
- **STM32F446** Family Reference Manual



Példaprogramok

Lab05_serial_test – kétirányú UART kapcsolat
Lab05_buffered_serial – bufferelt UART
Lab05_Xbee_receive – ZigBee RF adatátvitel
Lab05_ST7585_test – LCD szoftveres SPI-vel
Lab05_ST7585_SPI – LCD hardveres SPI-vel
Lab05_DS3231_I2C – I2C RTC kiolvasása

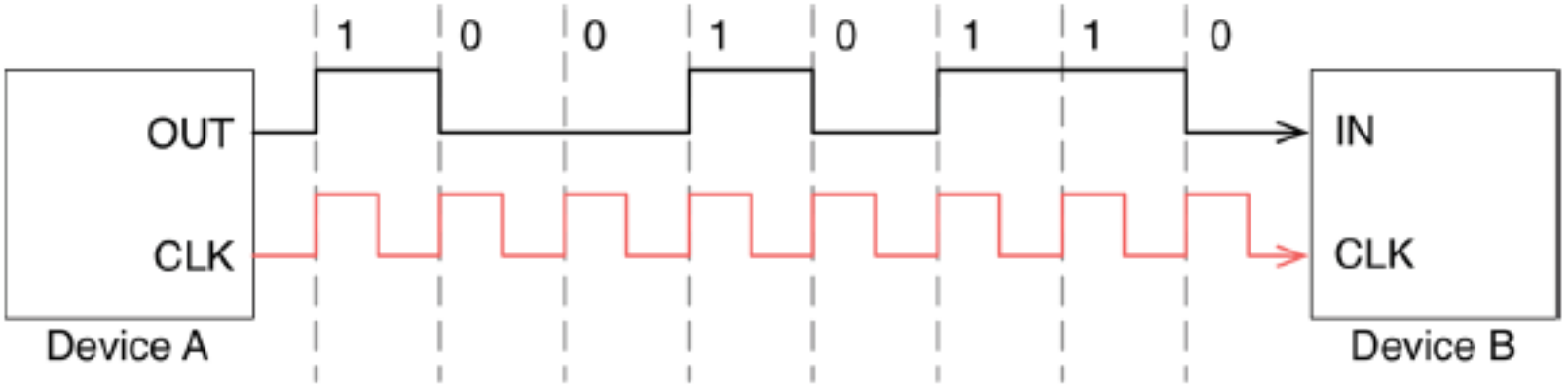


A mintaprogramok az os.mbed.com/users/cspista/code/ oldalon is megtalálhatók

Soros adatátvitel

- A soros kommunikáció olyan adatátvitel, amelyben az átküldendő üzenet minden egyes bitje időben egymás után kerül továbbításra
- Aszinkron soros átvitel: nincs közös órajel
- Szinkron soros átvitel: az adó és a vevő közös órajelet használ

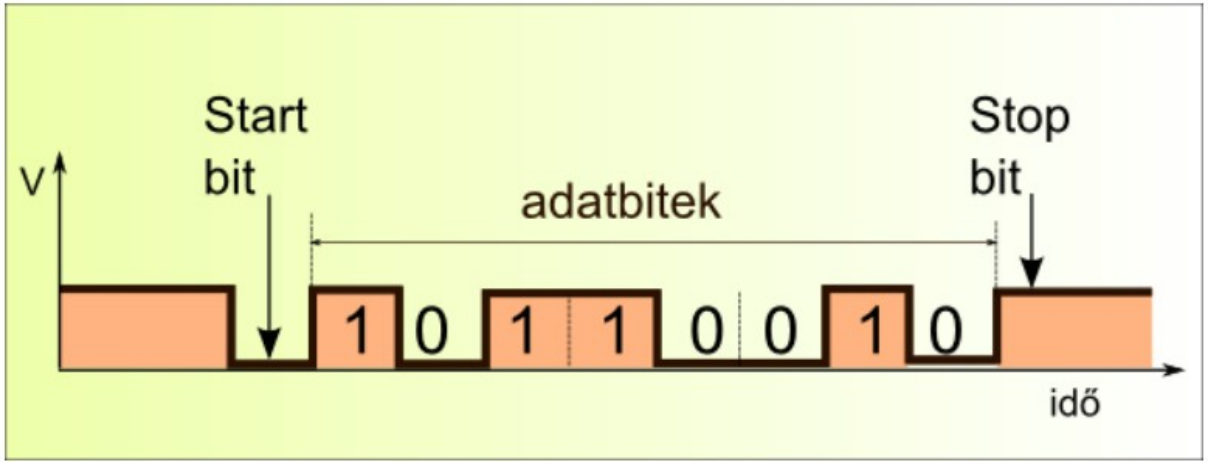
Szinkron soros kommunikáció



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

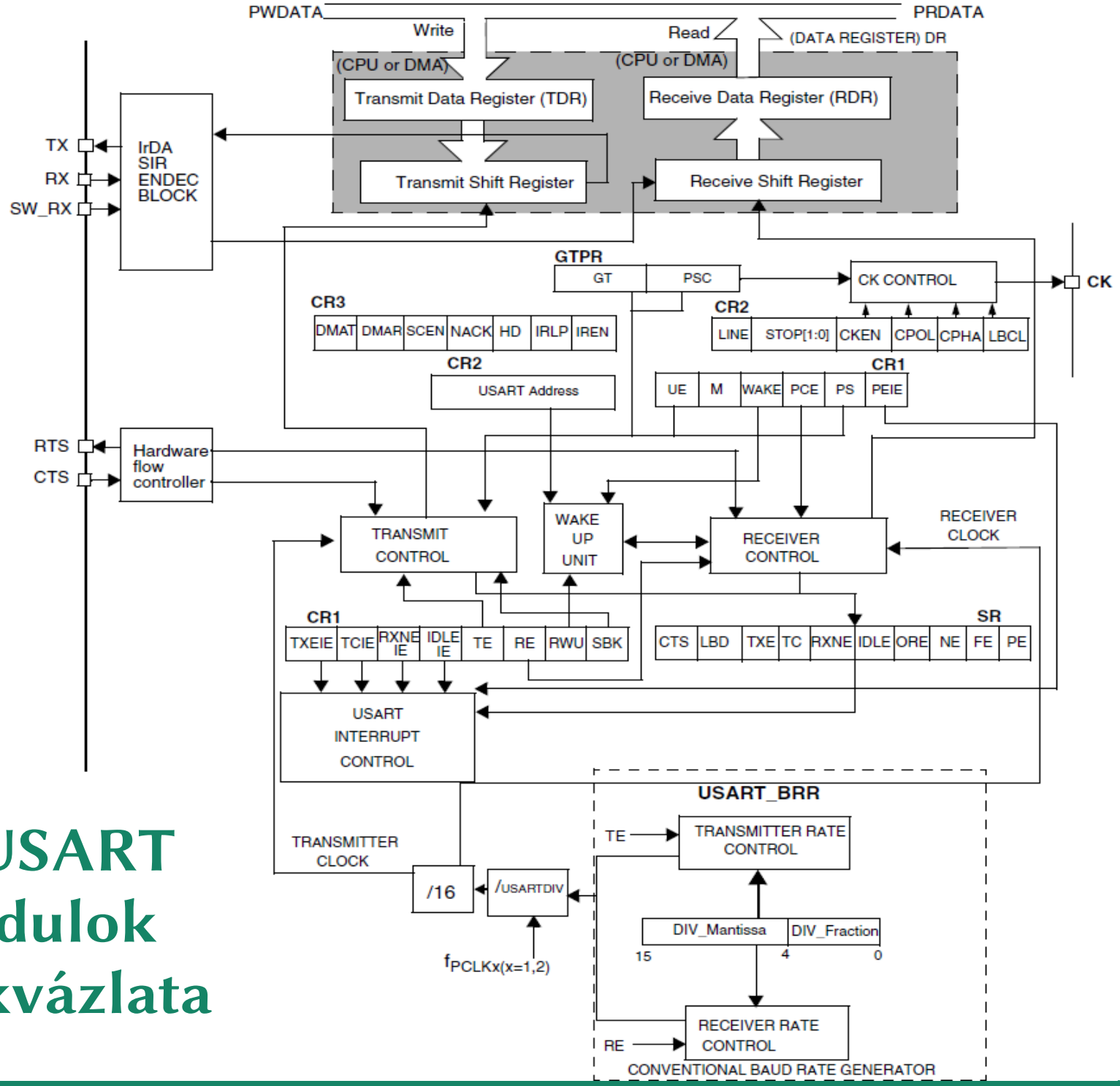


- Nincs szükség órajel továbbításra!
- Az adóban és a vevőben helyileg kell órajelet előállítani
- Az adó egy állandó értékű start bitet kell, hogy küldjön minden adategység előtt az adás kezdetének jelzésére
- A vevő detektálja a start bit homlokélét, és ehhez viszonyítva jelöli ki a mintavételezési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

UART kommunikáció jellemzői

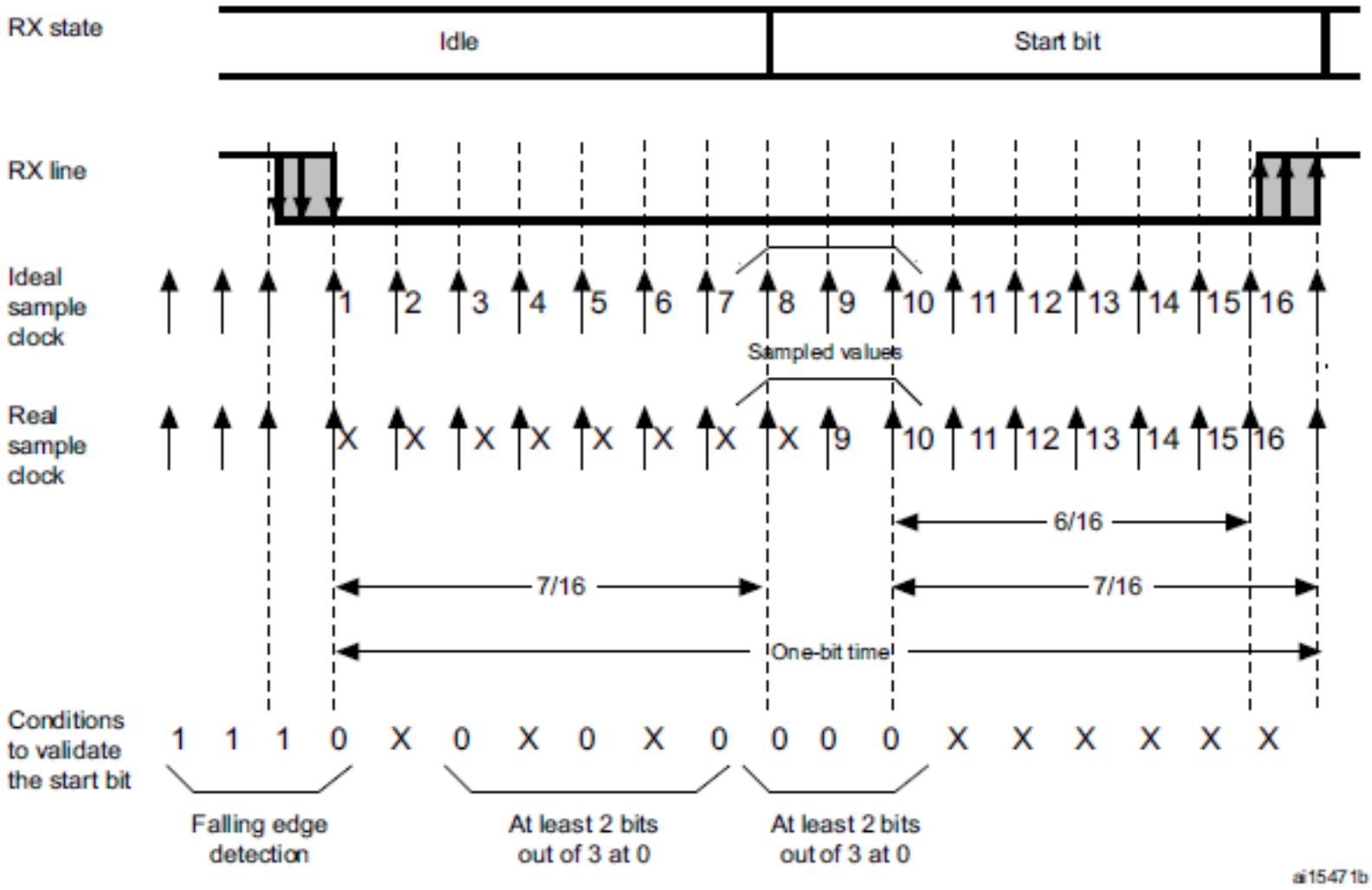
- **Adatkeret:**
  - Start bit (1 bit)
  - Adat (LSB vagy MSB sorrend, adatméret – 7, 8, 9 bit)
  - Opcionális paritásbit használható az adatban szereplő egyesek páros vagy páratlan számának jelzésére.
  - Stop bit (egy vagy két stop bit használható)
- **Az adó és a fogadó fél meg kell, hogy egyezzen:**
  - 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



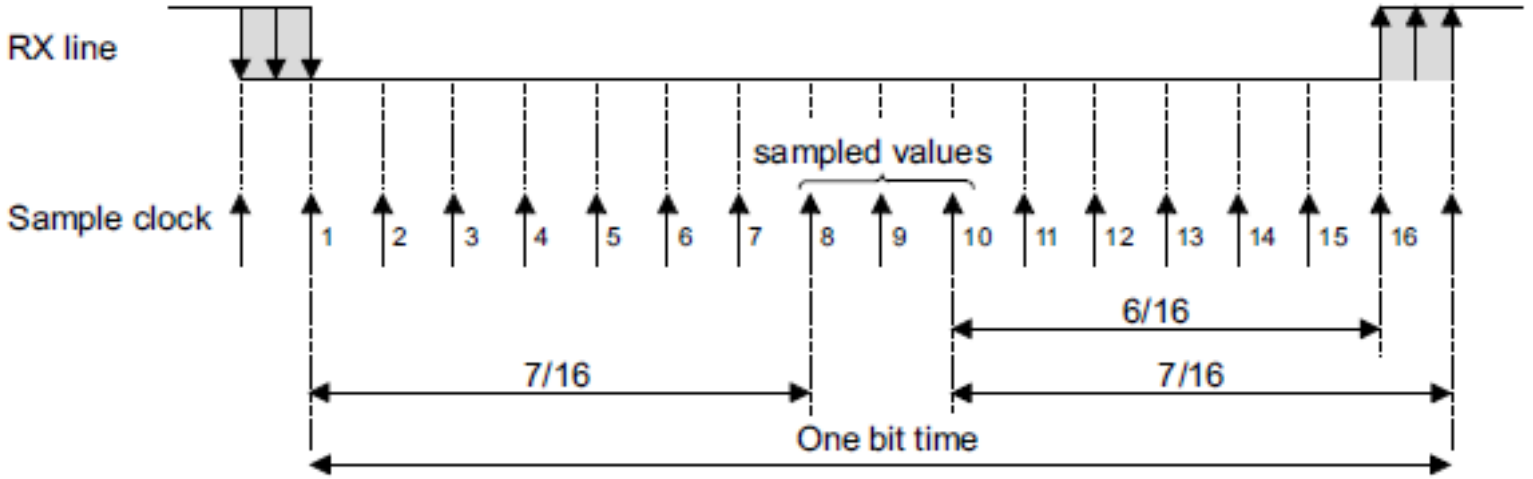
START bit detektálás túlmintavételezéssel

- 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)



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
USART3	PB13	PB14	PB10	PC5	PB12	AF7
			PC10	PC11	PC12	AF7
UART4	PB0	PA15	PA0	PA1	–	AF8
			PC10	PC11	–	AF8
UART5	PC9	PC8	PC12	PD2	–	AF7 (RTS,CTS)/AF8 (RX, TX)
USART6			PC6	PC7	PC8	AF8

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

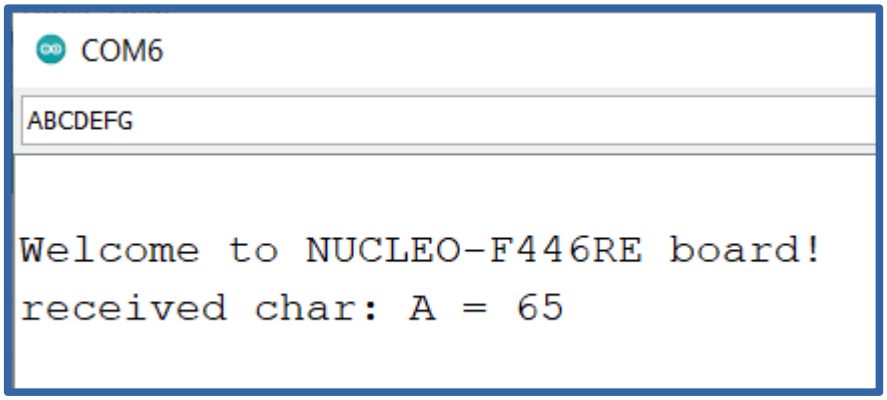
A Serial objektumosztály

- Az U(S)ART perifériákat a **Serial** osztállyal konfigurálhatjuk, amely az **mbed::SerialBase** és az **mbed::Stream** osztályok leszármazottja

Függvény	Használat
Serial név(txpin, rxpin**)**	Létrehoz egy "név" nevű **Serial** objektumot, a txpin, rxpin paraméterek szerint kiválasztja a hozzá rendelt UARTx periférát, és UART ki/bemenetként konfigurálja a megnevezetett kivezetések funkcióját (lásd 2. táblázat)
baud(baudrate**)**	Az adatküldési sebesség beállítása (alapértelmezetten 9600 bit/s)
format(bits, parity, stop**)**	Az adatküldés formátumának megadása (alapértelmezetten 8,N,1)
readable ()	Megvizsgálja, hogy van-e beérkezett karakter
writeable ()	Megvizsgálja, hogy van-e szabad hely a kimeneti tárban
putc(c**)**	Egy karakter küldése, blokkoló típusú várakozással.
getc()	Egy karakter fogadása, blokkoló típusú várakozással.
puts(s**)**	Szövegkonstans karakterfüzér kiíratása
gets(s,n**)**	*n* db karakter fogadása és eltárolása karakterfüzérként az *s* tömbbe
printf()	Formázott kiíratás
scanf()	Formázott szöveg beolvasása

Lab05_serial.test/main.cpp

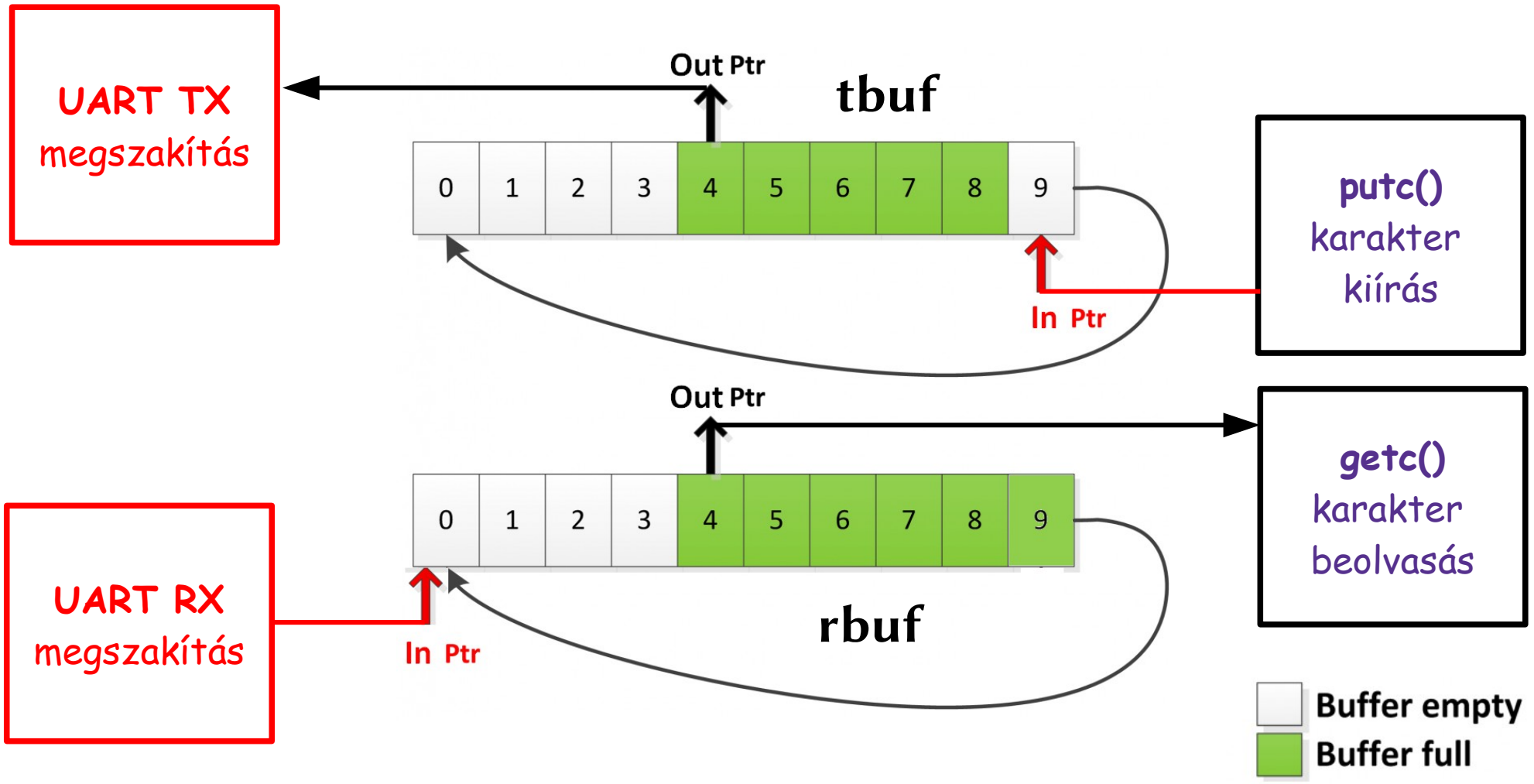
- Az alábbi mintaprogram az alapértelmezett UART2 csatornán és az ST-Link-1 eszközön keresztül kommunikál a számítógéppel, s minden beérkező karaktert visszatükröz és kiírja a kódját (tízes számrendszerben). Az alapértelmezett beállításokat használjuk: 9600 bit/s, 8-bit, nincs paritásbit, 1 stop bit
- **Probléma:** ha egyszerre több karaktert küldünk (pl. az Arduino IDE soros termináljáról), akkor adatvesztést tapasztalunk




```
#include "mbed.h"
Serial pc(USBTX,USBRX);      //UART via ST-Link

int main() {
  pc.printf("\r\nWelcome to NUCLEO-F446RE board!\r\n");
  while(1) {
    char c = pc.getc(); //Read one character
    pc.printf("received char: %c = %d\r\n",c,c);
  }
}
```

Megoldás: interruptos, bufferelt adatátvitel

- Kétirányú kommunikációhoz két bufferre lesz szükségünk:
  egy az UART Tx adó, egy pedig az UART Rx vevő számára
- Nem kell feltalálni a spanyolviaszt, mások már megírták (BufferedSerial)



Megoldás: interruptos, bufferelt adatátvitel

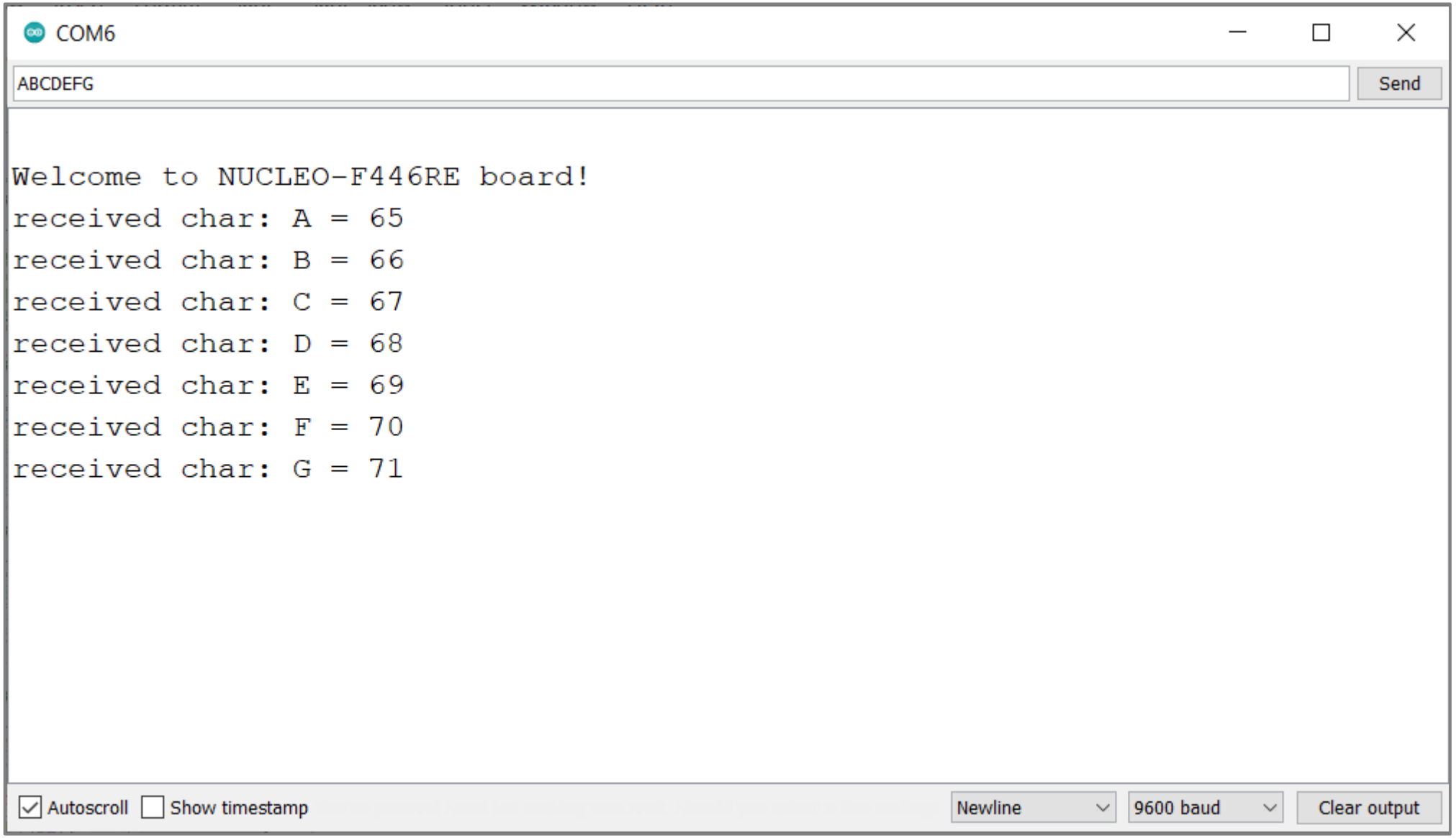
- Veysel Karadag gyűjteményében találtuk meg a **BufferedSerial** programkönyvtár STM32 F4 mikrovezérlőkre adaptált változatát

```
/* Use: BufferedSerial library
 * adopted for STM32F4 series by  Veysel KARADAG
 * https://os.mbed.com/users/veyselka/code/BufferedSerial/
 */
#include "mbed.h"                   // Bufferméret
#include "BufferedSerial.h"
BufferedSerial pc(USBTX, USBRX, 1024);

int main() {
    pc.printf("\r\nWelcome to NUCLEO-F446RE board!\r\n");
    while(1) {
        if(pc.readable() > 0) {       // Ha van olvasható adat
            char c = pc.getc();       // Read one character
            if((c>32) && (c<128)) {   // Ha nyomtatható a karakter
                pc.printf("received char: %c = %d\r\n",c,c);
            }
        }
    }
}
```

Lab05_buffered_serial/main.cpp

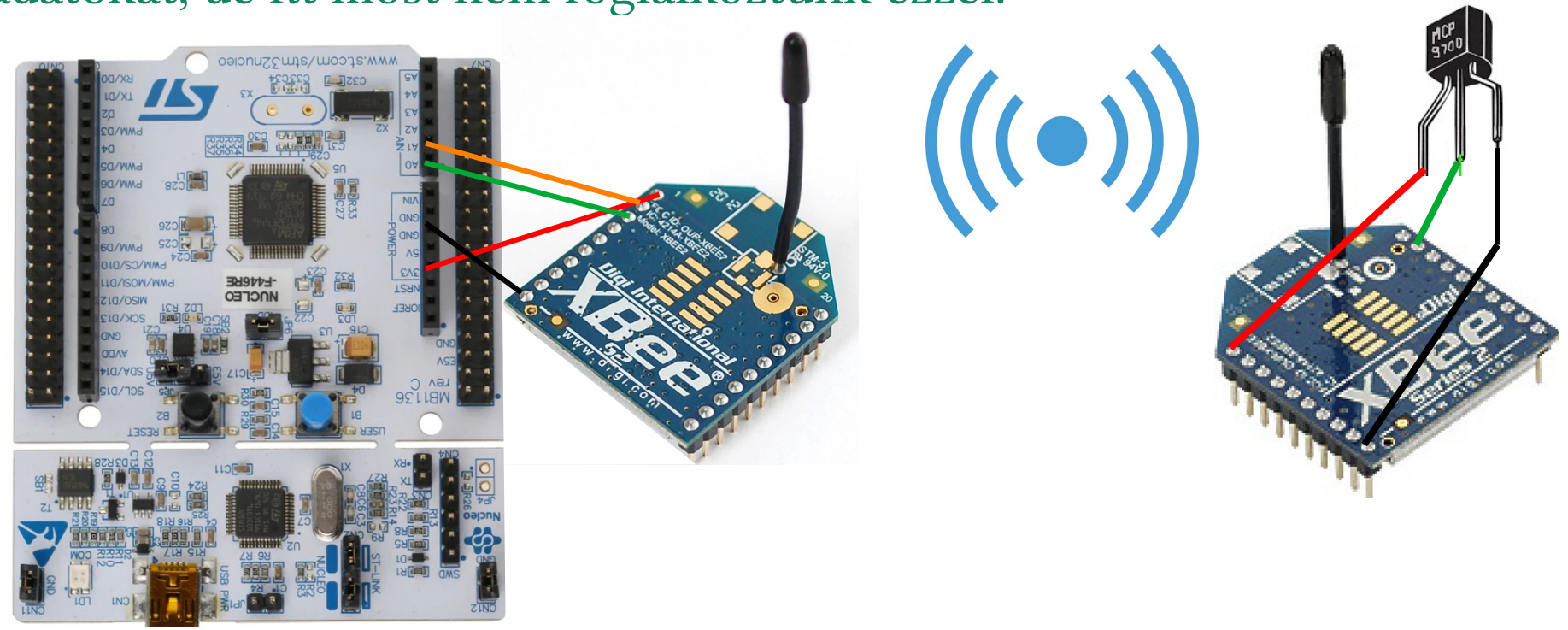
Lab05_buffered_serial: eredmény

- Hosszabb szövegnél sincs adatvesztés



Vezeték nélküli adatátvitel (XBee/ZigBee)

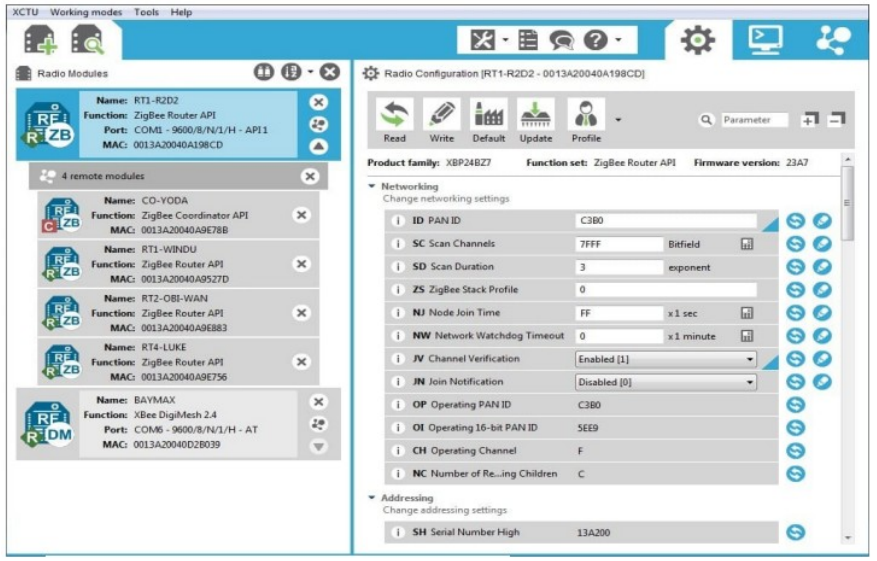
- A következő példa akár egy rádiós távhőmérő projekt alapja is lehet
- A vezeték nélküli jeltovábbítására Xbee/ZigBee modulokat használunk
- A hőmérő közvetlenül az **XBee** modulhoz kapcsolható, elemmel is táplálható. A másik **XBee** modul a **Nucleo** kártya **UART4** portjára csatlakozik, itt vesszük az adást és továbbítjuk a PC felé az UART2 porton keresztül. Az **STM32** mikrovezérlő értelmezhetné is a bejövő adatokat, de itt most nem foglalkoztunk ezzel.



DIGI XBee fejlesztői eszközök

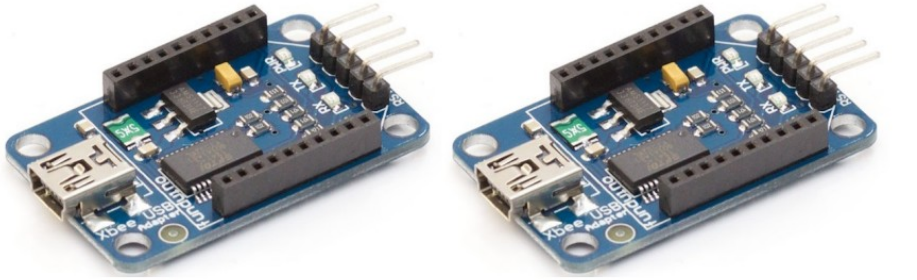
XCTU – Xbee Configuration & Test Utility

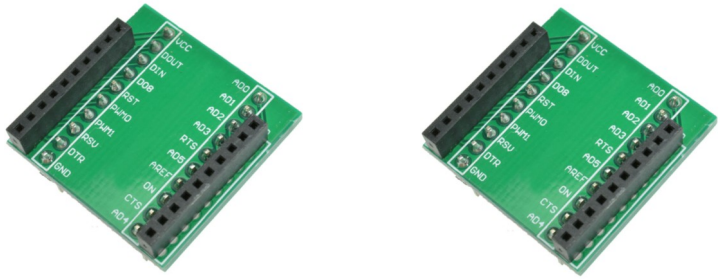
- ingyenes, multi-platformos alkalmazás
- Grafikus hálózat megjelenítés
- Firmware frissítés és konfigurálás
- API Frame Builder segíti az üzenetek összeállítását és dekódolását



Fejlesztői csomag:

- Xbee Pro S2 modulok
- Funduino XBee USB adapterek
- 2mm ⟶ 2,54 mm-es adapter kártya XBee modulokhoz



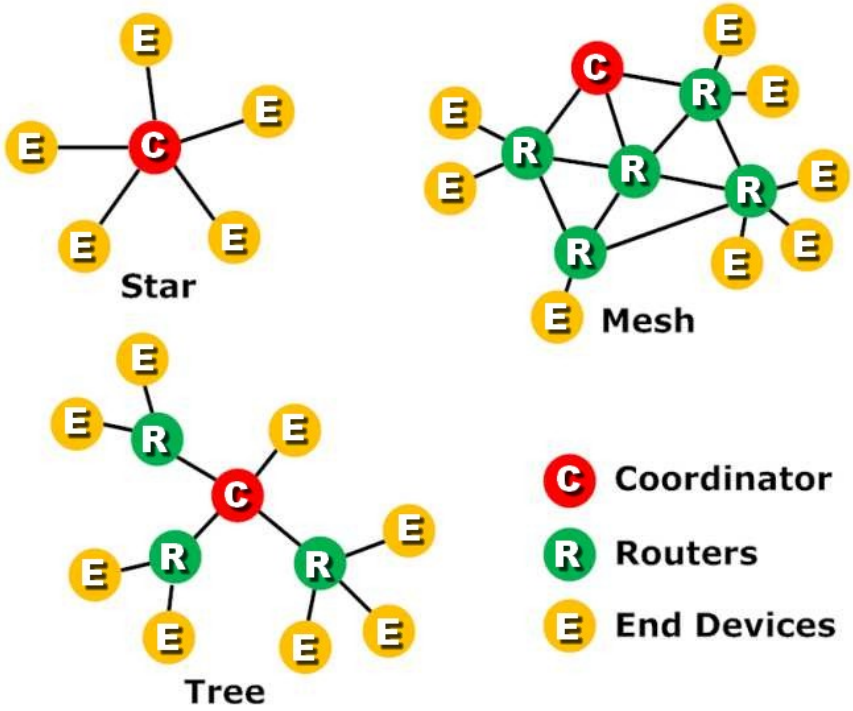


XBee eszközök

- A **DIGI International** sokféle vezetéknélküli kommunikációs eszközt gyárt **XBee** elnevezéssel, a ZigBee, Bluetooth, WiFi, 3G, 4G és 5G hálózatok eszközei számára
- Mi ZigBee kompatibilis **Xbee Pro S2** (**XBP24-ZB** termékcsalád) modulokkal végeztünk kísérleteket
- Korábban már részletesen ismertettük ezen modulok használatát, van videófelvétel is róla ezért itt nem térünk ki a részletekre
  - Vezeték nélküli kommunikáció (XBee/ZigBee) - 1. rész (2020. április 23.)
    Előadásvázlat Video Példaprogramok
  - Vezeték nélküli kommunikáció (XBee/ZigBee) - 2. rész (2020. május 7.)
    Előadásvázlat Video Példaprogramok
- Az általánosabb lehetőségek bemutatása céljából nem a pont-pont kapcsolathoz való transzparens módot, hanem a komplex hálózatok kialakítására is használható **API módot** fogjuk használni, a modulokat is ehhez fogjuk konfigurálni

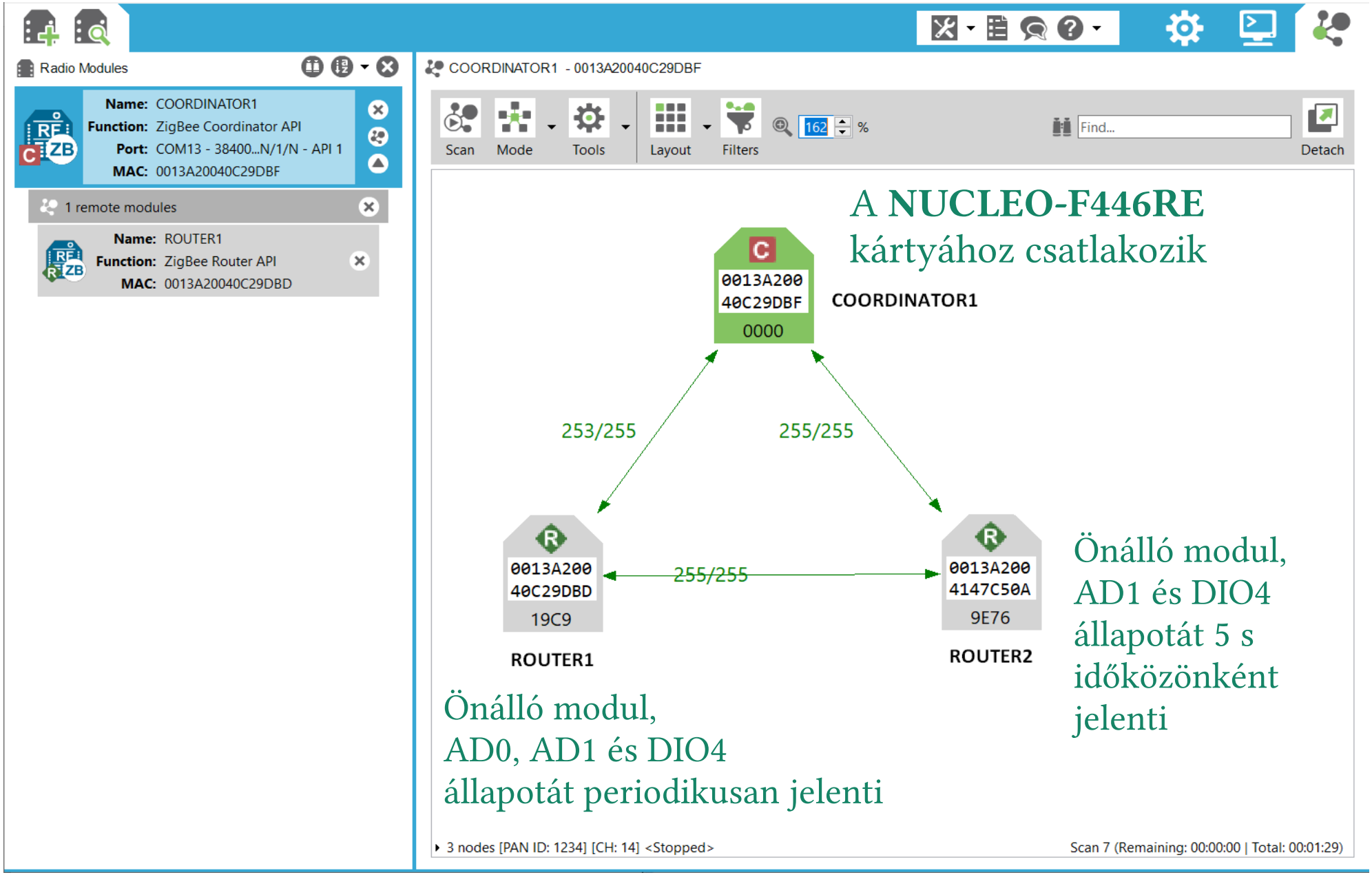
802.15.4 hálózati topológia modellek

- **Star (csillag) elrendezés jellemzői:**
  - Egyszerű felépítés
  - Korlátozott kiterjedésű
  - A koordinátor szűk keresztmetszet lehet
- **Tree (fa) elrendezés jellemzői:**
  - Kiterjeszti a hálózat elérhetőségét
  - Szűk keresztmetszet még mindig lehet
- **Mesh (háló) elrendezés jellemzői:**
  - Komplex felépítés
  - Nagy megbízhatóság
  - Tehermentesíti a szűk keresztmetszetet



- Minden hálózatban pontosan egy koordinátor modulnak kell lennie
- Az átjárást biztosító router moduloknak is állandóan elérhetőnek kell lenniük

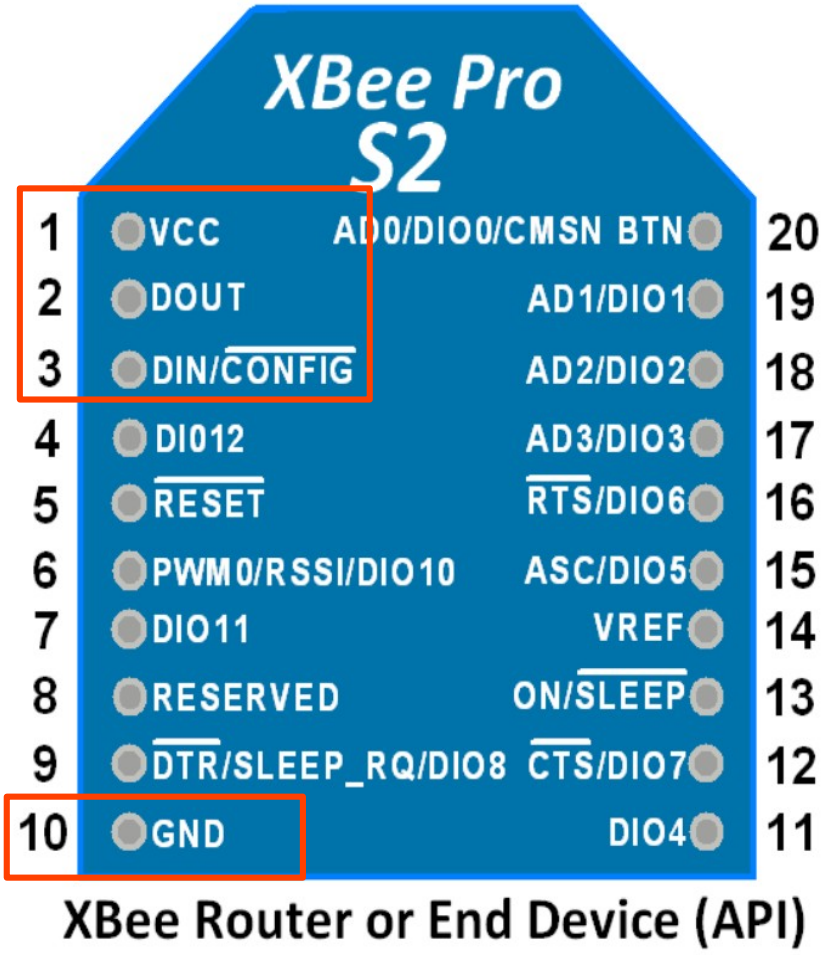
Hálózati topológia



Az Xbee modul kivezetései – 1. oldal

- A **Digi Xbee/ZigBee Pro S2** modulok önálló eszközként is használhatjuk, pl. az analóg/digitális lábak állapota lekérdezhető
- A Koordinátornál a VCC, DOUT, DIN és GND lábakat kötjük be

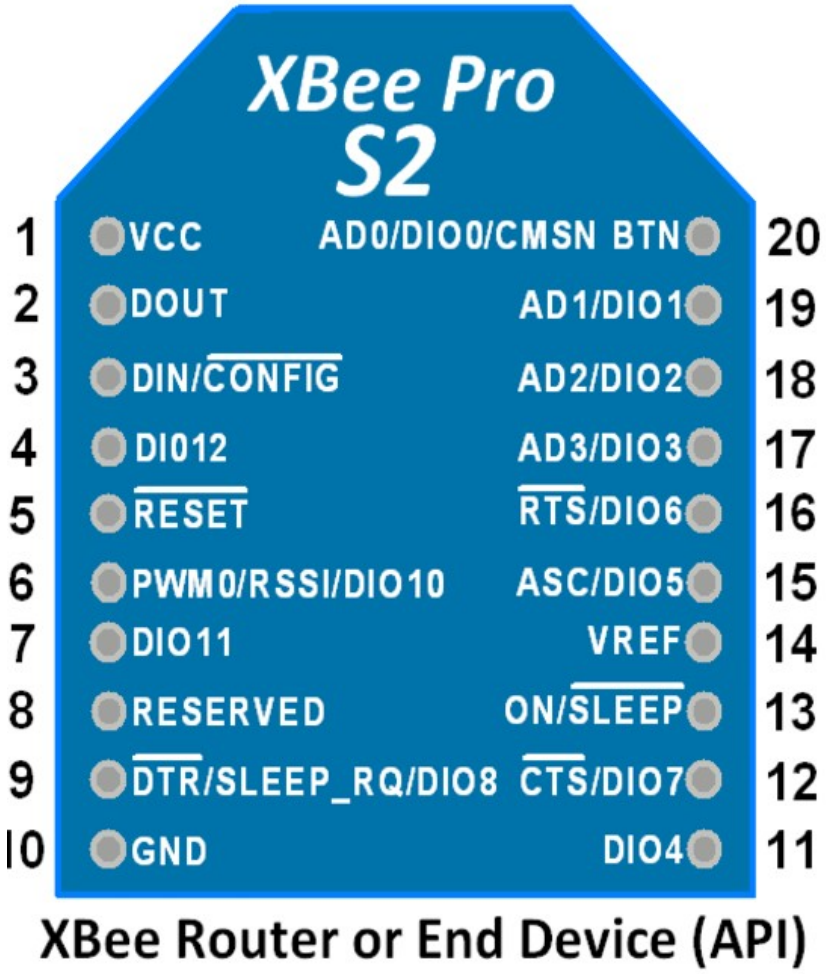
Pin	Name	dir	default	Fuction
1	**VCC**	PWR	–	Power supply
2	**DOUT**	out	out	UART data out
3	**DIN**/CONFIG	in	in	UART data in
4	DIO12	both	out	Digital I/O 12
5	RESET	both	OC w Pup	Module reset
6	RSSI PWM/ DIO10	both	out	RSSI indicator/ I/O
7	DIO11	both	in	Digital I/O 11
8	Reserved		disabled	Do not connect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

- A második oldalon találjuk az analóg bemeneteket, illetve a **DIO0 – DIO7** digitális ki/bemeneteket
- Az adatáramlást vezérlő **$\overline{\text{RTS}}$/$\overline{\text{CTS}}$** kivezetéseket most nem használjuk



XBee Router or End Device (API)

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

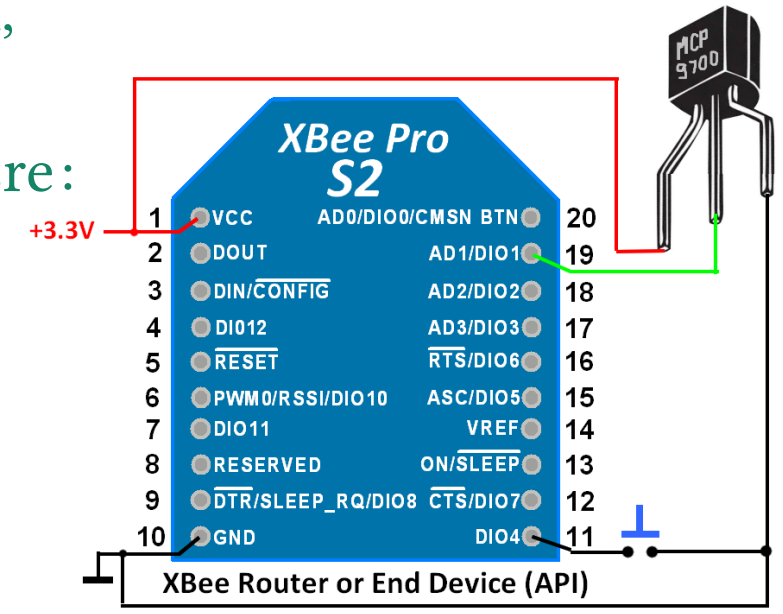
- Az Xbee moduloknak elég tápfeszültséget adni, hogy működőképesek legyenek
- Egy nagyon egyszerű példa az önálló működésre:
  - **AD1/DIO1** lábon **MCP9700** analóg hőmérő
  - **DIO4** lábon egy nyomógomb
- A 10 bites ADC 1,20 V-os belső referenciát használ, így a millivoltokban mért feszültség:

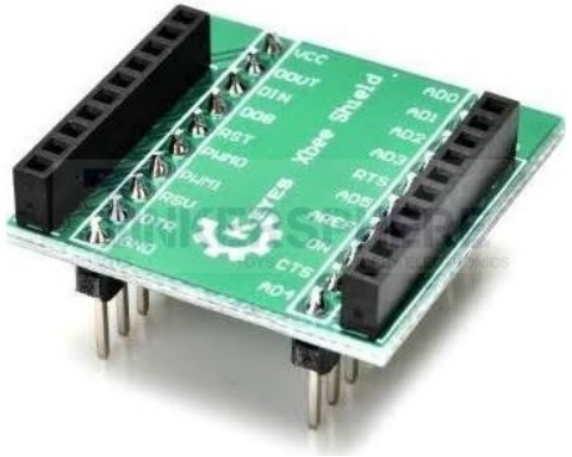
$$U[mV] = \frac{N_{ADC} \cdot 1200}{1024}$$

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

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

- Az XBee modul 2 mm-es osztású, a dugaszolós panelhoz csak adapterrel tudjuk csatlakoztatni!

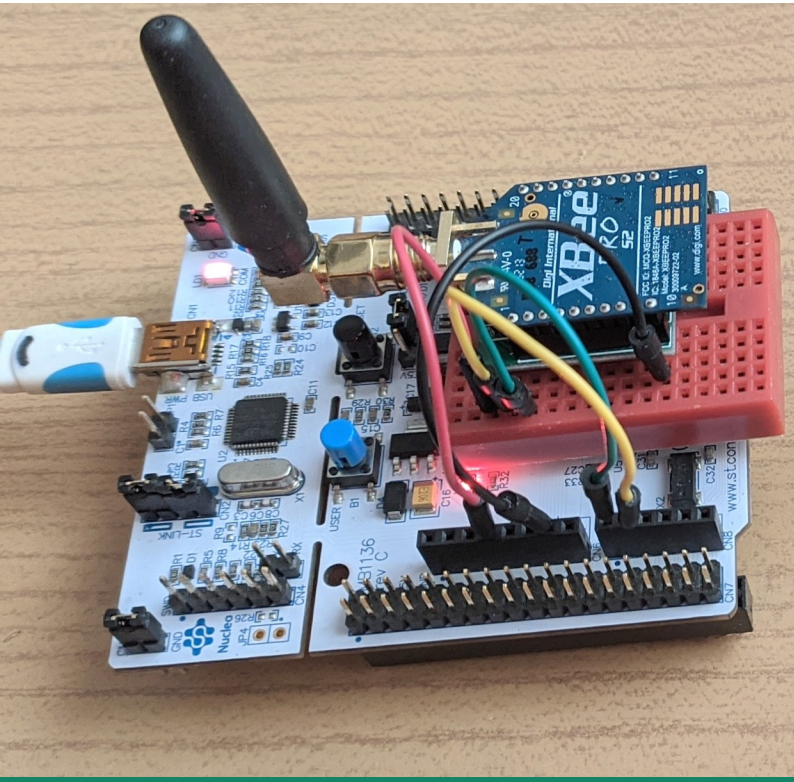


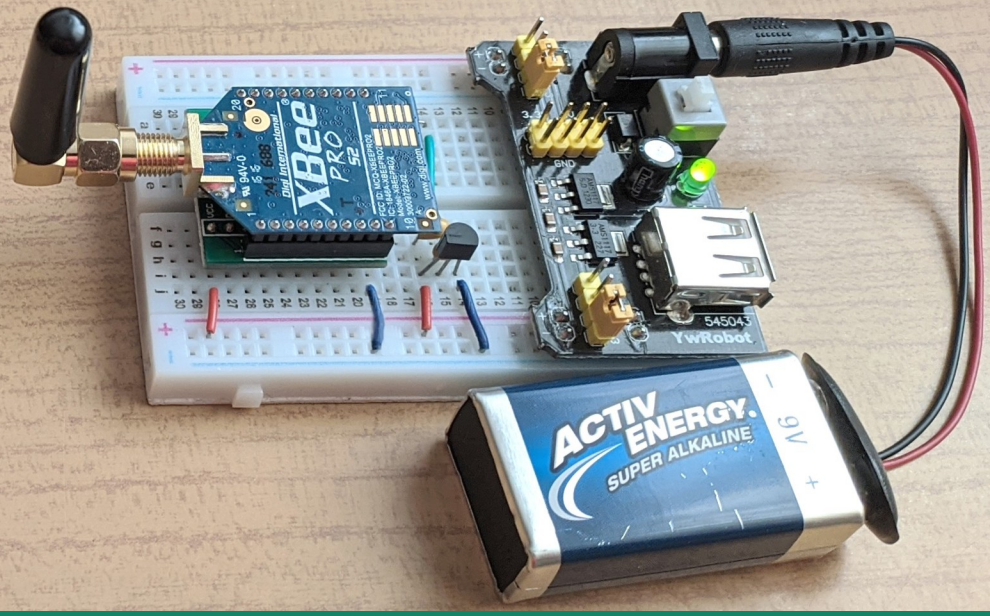


Az áramköri elrendezés

- **COORDINATOR1**
  VDD – 3,3 V
  DOUT – PA_1
  DIN – PA_0
  GND – GND

ROUTER2
VDD – 3,3 V (elem + tápegység)
GND – GND
AD1 – MCP9700 kimenőjele
DIO4 – nyomógomb (lehagytuk)



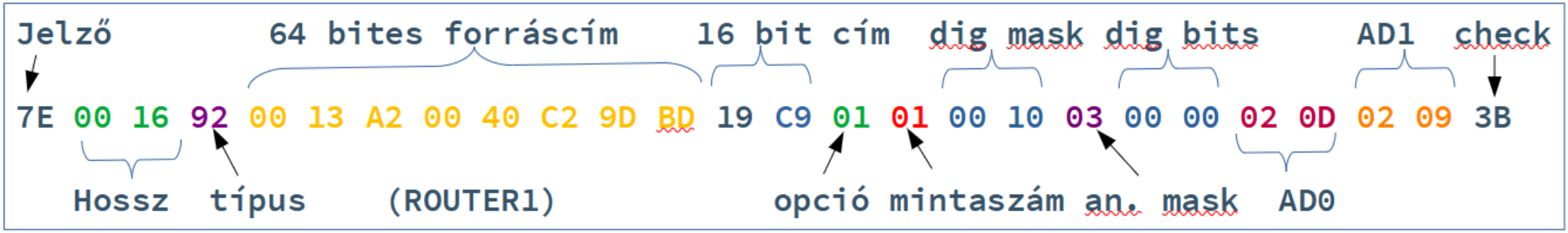


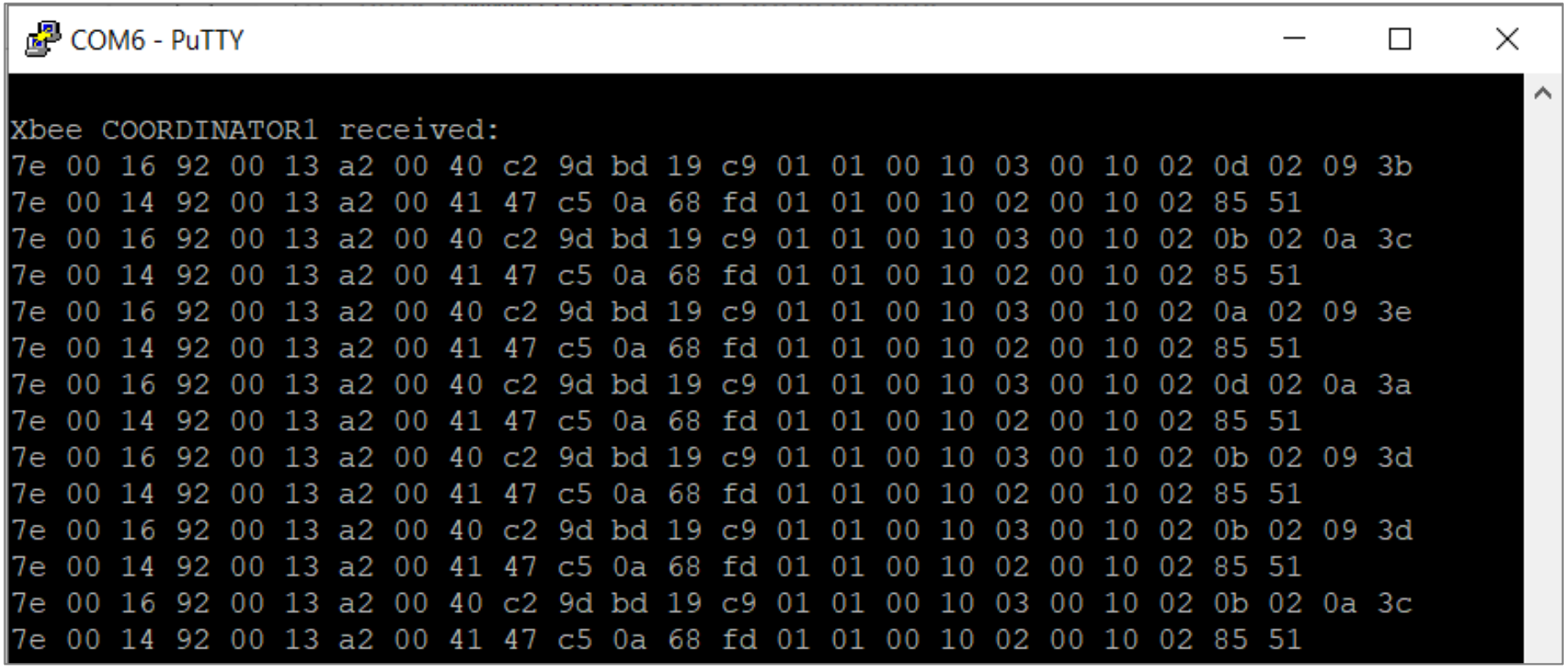
Lab05_Xbee_receive/main.cpp

```
#include "mbed.h"
#include "BufferedSerial.h"
BufferedSerial pc(USBTX,USBRX);
BufferedSerial xbee(PA_0, PA_1);
char fb[1024];                          // Frame buffer

int main() {
    pc.baud(115200);
    xbee.baud(38400);
    int i = 0;
    pc.printf("\r\nXbee COORDINATOR1 received:\r\n");
    while(1) {
        while(xbee.readable()) {        // Read all data
            fb[i++] = xbee.getc();
        }
        if(i<20) {                      // Wait for further data id frame incomplete
            wait(0.5);
        } else {                        // Write out data to PC
            for(int c=0; c<i; c++) {
                pc.printf("%02x ",fb[c]);
            }
            pc.printf("\r\n");
            i = 0;                      // Free up buffer if printed out
        }
    }
}
```

Lab05_Xbee_receive futási eredménye

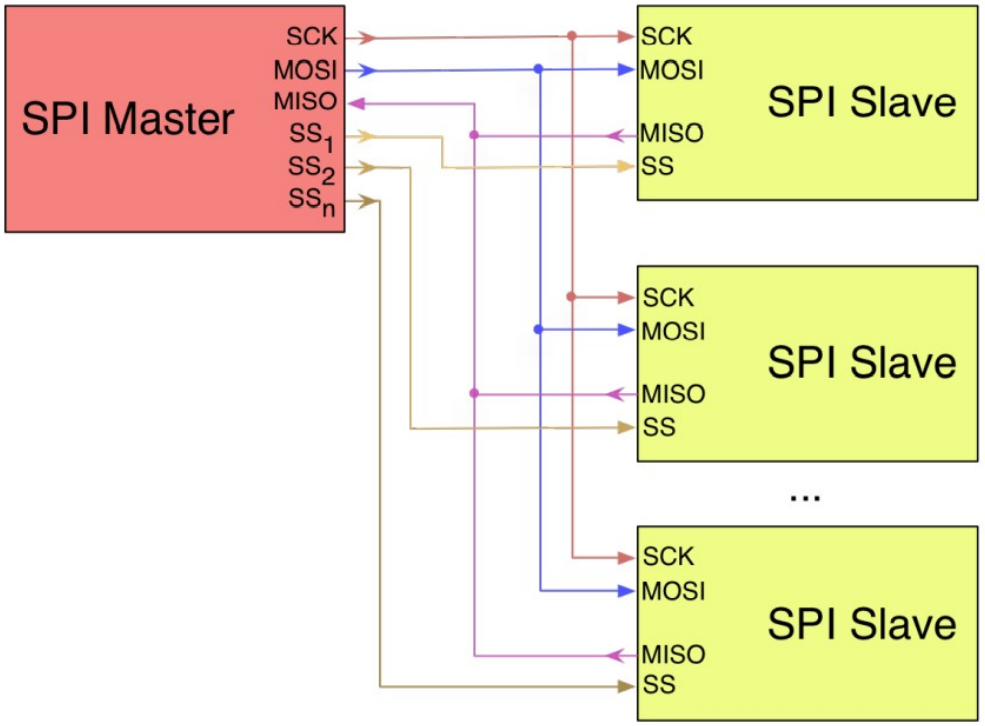
- Az adatok a két routertől keretezett csomagokban érkeznek, értelmezésükben az alábbi ábra segít



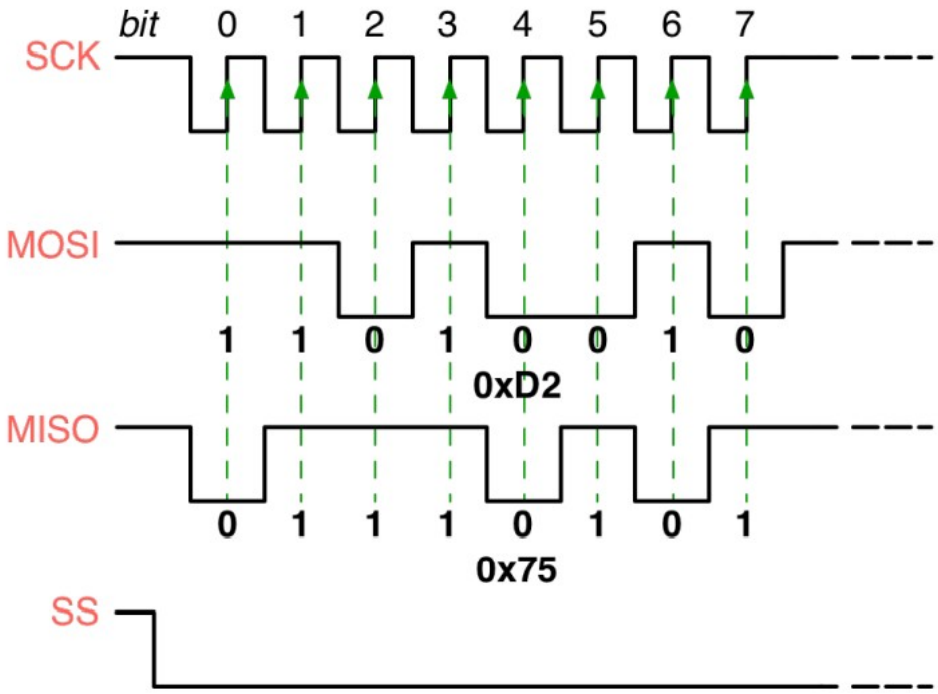


Az SPI kommunikációs csatorna

- Az **SPI** (soros periféria illesztő = Serial Peripheral Interface) busz kétirányú szinkron soros kommunikációt valósít meg
- A kommunikációban résztvevő eszközök között master/slave (mester/szolga) viszony áll fenn
- Több eszköz kezelése esetén egyedi választóvonalakkal (SS) oldható meg a „címzés”



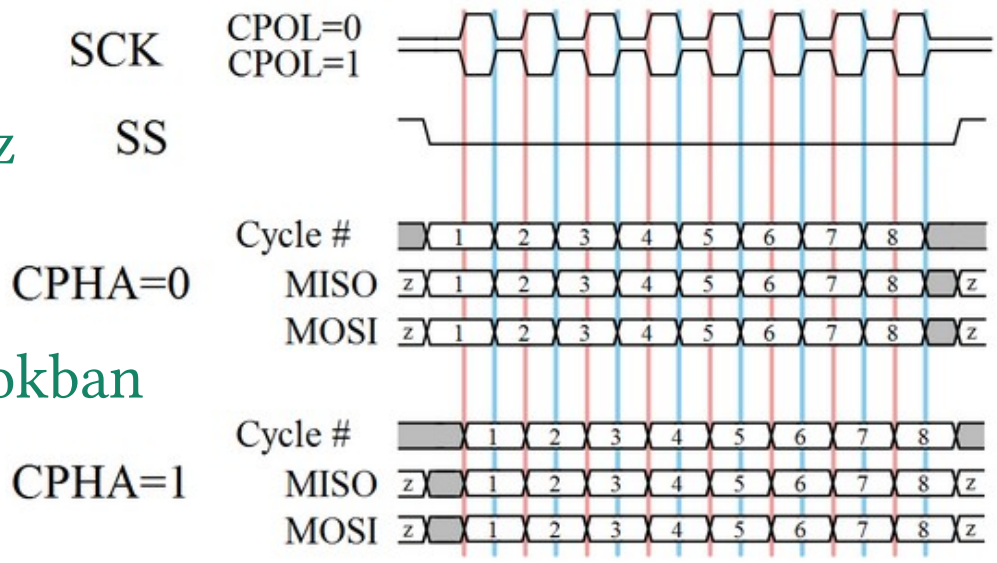
Kétirányú adatforgalom



Az ábrák forrása: Carmine Noviello: Mastering STM32

Az SPI órajel polaritása és fázisa

- Az **SPI** adatátvitel szinkronizálását a master által keltett szinkron órajel (SCK) biztosítja. Az órajel frekvenciáján kívül biztosítani kell a slave eszközhöz illeszkedő polaritását és fázisát is
- **SCK** - a szinkronizálást biztosító órajel, a master eszköz állítja elő
- **SS** - Slave select, azaz a slave eszköz kiválasztására szolgáló jel, melynek '0' állapota aktivizál. A master eszköz állítja elő.
- **MOSI** - a master kimeneti adatvonala (Master out, Slave in). A jelet a master eszköz állítja elő.
- **MISO** - A slave eszköz kimeneti adatvonala, melyet a master olvas (Master in, Slave out). A slave eszköz állítja elő
- **CPHA = 0** esetén az adatvonalak bekapuzása a rózsaszín jelű időpontokban
- **CPHA = 1** esetén az adatvonalak bekapuzása a kék jelű időpontokban



Az SPI objektumosztály

- Az **SPI** osztállyal a mikrovezérlőt ***master*** módban használhatjuk, az SPI perifériákat (pl. EEPROM memória, TFT kijelző, digitális potenciométer, külső SPI DAC, LED mátrix kijelző, SD kártya) vezéreljünk.
- Van **SPIslave** osztály is ***slave*** módhoz, de ezzel most nem foglalkozunk
- Bármilyen meglepő, nincs read() tagfüggvény, azaz a master mindig kell, hogy küldjön valamilyen adatot, s a slave által küldött adat a write() tagfüggvény visszatérési értéke lesz, pl.:

```
int response = device.write(0xFF);
```

- Használhatunk hardveres **Chip Select** kezelést is (4. paraméter), de így nem lesz hordozható a kód, mert nem mindegyik mikrovezérlő támogatja

Függvény	Használat
SPI név(mosipin, misopin, sclkpin**)**	Létrehoz egy "név" nevű **SPI** objektumot, a *mosipin, misopin* és *sclkpin* paraméterek szerint kiválasztja a hozzá rendelt SPIx periférát, és SPI buszként konfigurálja a megnevezetett kivezetések funkcióját.
frequency(hz**)**	Az adatküldési sebesség beállítása (alapértelmezetten 1 MHz)
format(bits, mode**)**	Az üzemmód beállítása (alapértelmezetten 8-bit, 0 mód)
write(data**)**	Adat kiküldés és olvasás (a beolvasott értékkel tér vissza)

ST7585 monokróm grafikus LCD vezérlése

- A **Sitronix ST7585** vezérlővel ellátott, Shanyan TFT module v1.0 kijelzőt már bemutattuk egy korábbi előadáson:
  ST7585 monokróm grafikus LCD vezérlése (2018. december 13.)
  Előadásvázlat Mintaprogramok
- A monokróm grafikus LCD 96*64 képpont felbontású és egy fix ikonsor megjelenítésére is alkalmas
- Most a korábbi előadáson bemutatott **lcd_IO.ino** Arduino programot írjuk át **mbed** környezetre – két változatban:
- **Lab05_ST7585_test:** szoftveres SPI kezeléssel
- **Lab05_ST7585_SPI:** hardveres SPI kezeléssel

Lab05_ST7585_test/main.cpp 5/1.

```
#include "mbed.h"
#include "bitmap.h"

DigitalOut SCK_pin(D13);    // SPI clock
DigitalOut SDI_pin(D11);    // SPI MOSI
DigitalOut DC_pin(D10);     // Data/command selector
DigitalOut RST_pin(D9);     // HW Reset
DigitalOut CS_pin(D8);      // SPI chip select

void LCDShiftWrite(unsigned char dat) {
    unsigned char i;
    unsigned char Series,Temp;
    SCK_pin = false;                // SCK LOW
    Series = dat;
    for(i=0; i<8; i++) {            // For each bit
        SCK_pin = false;            // SCK LOW
        Temp=Series & 0x80;         // MSB first
        if(Temp) {                  // Set MOSI line
            SDI_pin = true;
        } else {
            SDI_pin = false;
        }
        SCK_pin = true;             // SCK HIGH
        Series = Series << 1;       // Shifts data bits to left
    }
}
```

Az **LCDShiftWrite()** függvény az **SPI.write()** szoftveres megvalósítása (bitbanging)

Egy bájtot küldünk ki, elsőként az MSB bittel kezdve

Lab05_ST7585_test/main.cpp 5/2.

```
void send_cmd(unsigned char cmd, unsigned char dat) {
    DC_pin = false; //digitalWrite(RSX,LOW);
    CS_pin = false; //digitalWrite(CSX,LOW);
    LCDShiftWrite(cmd|dat);
    CS_pin = true;  //digitalWrite(CSX,HIGH);
    DC_pin = true;  //digitalWrite(RSX,HIGH);
}

void send_dat(unsigned char dat) {
    DC_pin = true;  //digitalWrite(RSX,HIGH);
    CS_pin = false; //digitalWrite(CSX,LOW);
    LCDShiftWrite(dat);
    CS_pin = true;  //digitalWrite(CSX,HIGH);
}

void LCDInit() {
    RST_pin = false; //digitalWrite(RESX,LOW);
    wait_ms(10);
    RST_pin = true;  //digitalWrite(RESX,HIGH);
    wait_ms(100);
    send_cmd(Function_Set,0x01);
    send_cmd(Set_V0,0x20);
    send_cmd(Set_Test_Mode,0x02);
    send_cmd(Function_Set,0x00);
    send_cmd(Display_Control,0x04);
}
```

A parancs és az adatküldés között csak a D/C vonal állapotában van eltérés
D/C = 0 parancs
D/C = 1 adat

HW Reset

Itt nem ártana még egy képernyőtörlés!

Lab05_ST7585_test/main.cpp 5/3.

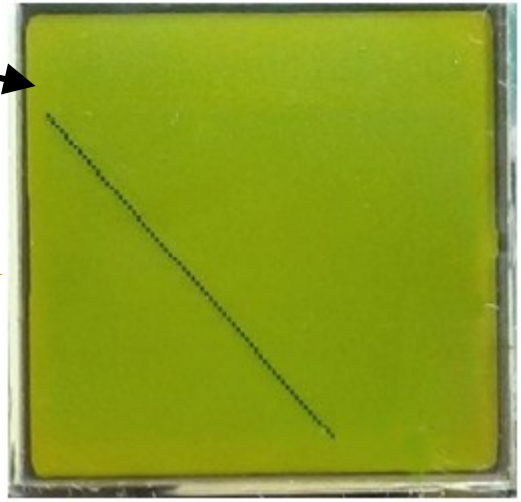
```
void LCD_set_XY(unsigned char x,unsigned char y) {      // Set cursor position
    send_cmd(Set_X_Address,x);
    send_cmd(Set_Y_Address,y);
}
void cls(void) {                                        // Clear screen
    int i;
    send_cmd(Set_X_Address,0);
    send_cmd(Set_Y_Address,0);
    for(i=0; i<960; i++)
        send_dat(0x00);
    send_cmd(Set_X_Address,0);
    send_cmd(Set_Y_Address,0);
}
void putch(unsigned char x, unsigned char y, unsigned int ch) { // write character
    unsigned char i;
    send_cmd(Set_X_Address,x);
    send_cmd(Set_Y_Address,y);
    for(i=0; i<5; i++)
        send_dat(FONT[(ch-0x20)*5+i]);
}
void putstr(unsigned char x, unsigned char y, char *str) {      // Write string
    while(*str!=0) {
        putch(x,y,*str++);
        x=x+6;
    }
}
```

Lab05_ST7585_test/main.cpp 5/4.

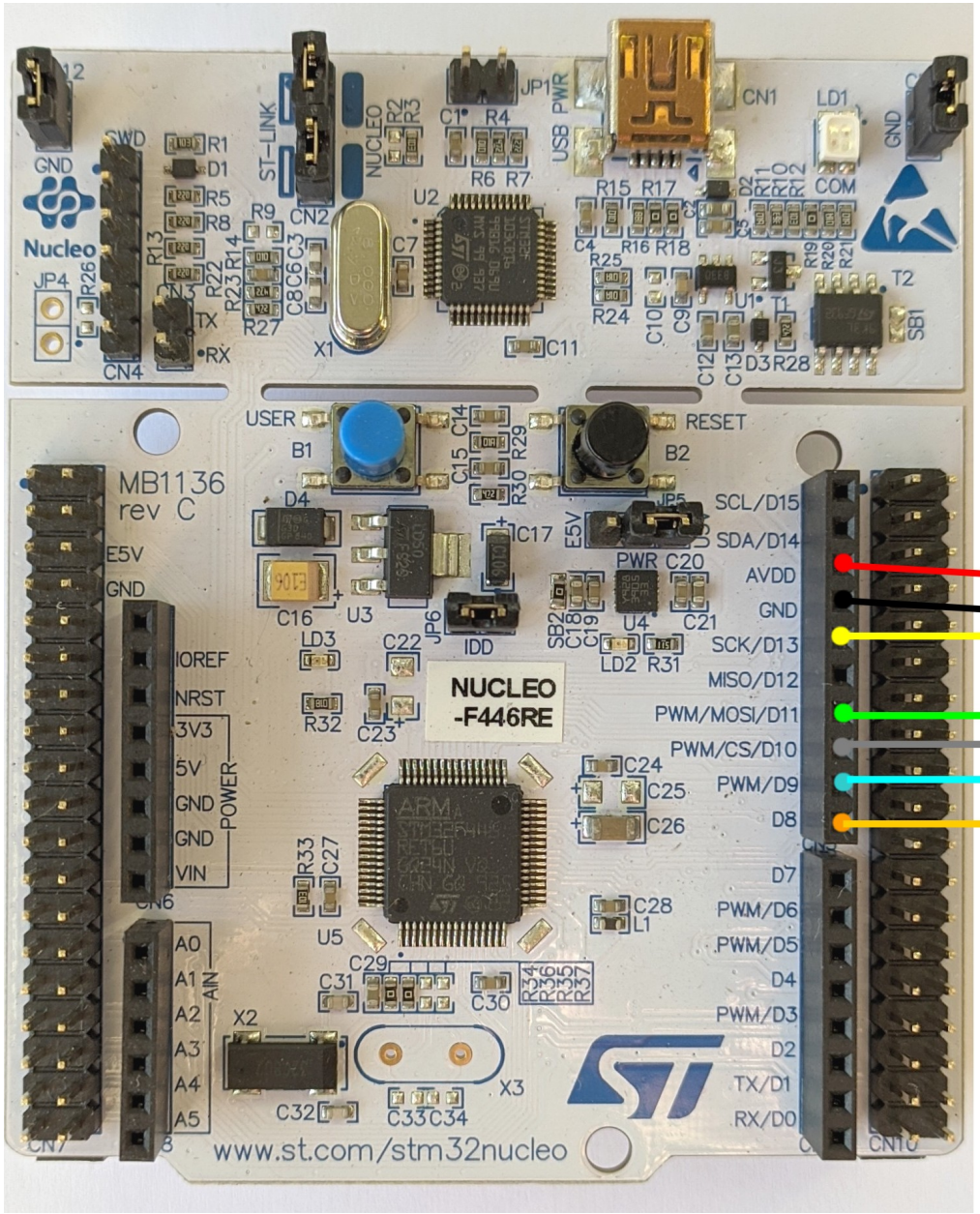
```
void SHOW_BMP() {                                   // Draw a bitmap from bitmap.h
    unsigned int i,j,n=0;
    for(i=0; i<8; i++) {
        for(j=0; j<96; j++) {
            send_cmd(Set_Y_Address,7-i);
            send_cmd(Set_X_Address,j);
            send_dat(BMP[n++]);
}   }   }
void SHOW_ICO() {                                   // Show icons
    unsigned char i;
    for(i=0; i<96; i++) {
        send_cmd(Set_Y_Address,8);
        send_cmd(Set_X_Address,i);
        send_dat(0xFF);
    }
}
void SHOW_LINE() {                                  // Draw a single line
    unsigned int i,j;
    const unsigned char line[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
    for(i=0; i<8; i++) {
        send_cmd(Set_Y_Address,7-i);
        for(j=0; j<8; j++) {
            send_cmd(Set_X_Address,j+8*i);
            send_dat(line[j]);
        }
    }
}
```

Lab05_ST7585_test/main.cpp 5/5.

```
int main() {
    LCDInit();
    while(1) {
        SHOW_ICO();

        putstr(30,5,"ST7585");
        putstr(15,3,"96*64 GLCD");
        putstr(10,1,"ARM Mbed demo");
        wait_ms(5000);

        cls();
        SHOW_LINE();
        wait_ms(5000);
        cls();

        SHOW_BMP();
        wait_ms(5000);
        cls();
    }
}
```

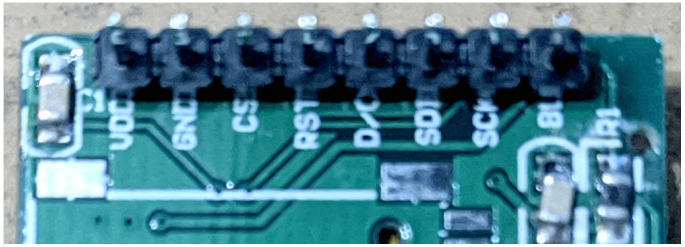


Bekötési vázlat



100R

Display	NUCLEO
VCC	3,3 V
GND	GND
CS	D8
RST	D9
D/C	D10
SDI	MOSI
SCK	SCK
BL	100R VCC-re



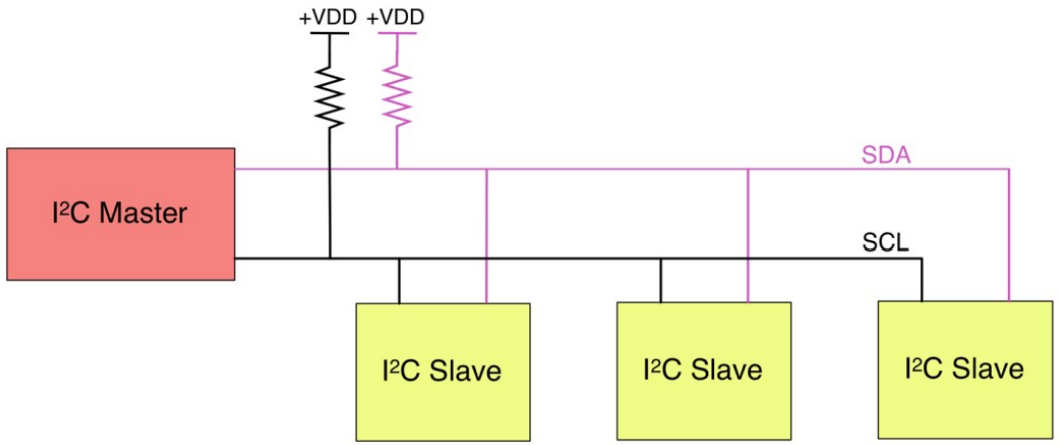
Lab05_ST7585_SPI/main.cpp (részlet)

- A második változatban az **LCDShiftWrite()** függvény helyett az adatküldés az **SPI** osztály **write()** tagfüggvényével történik

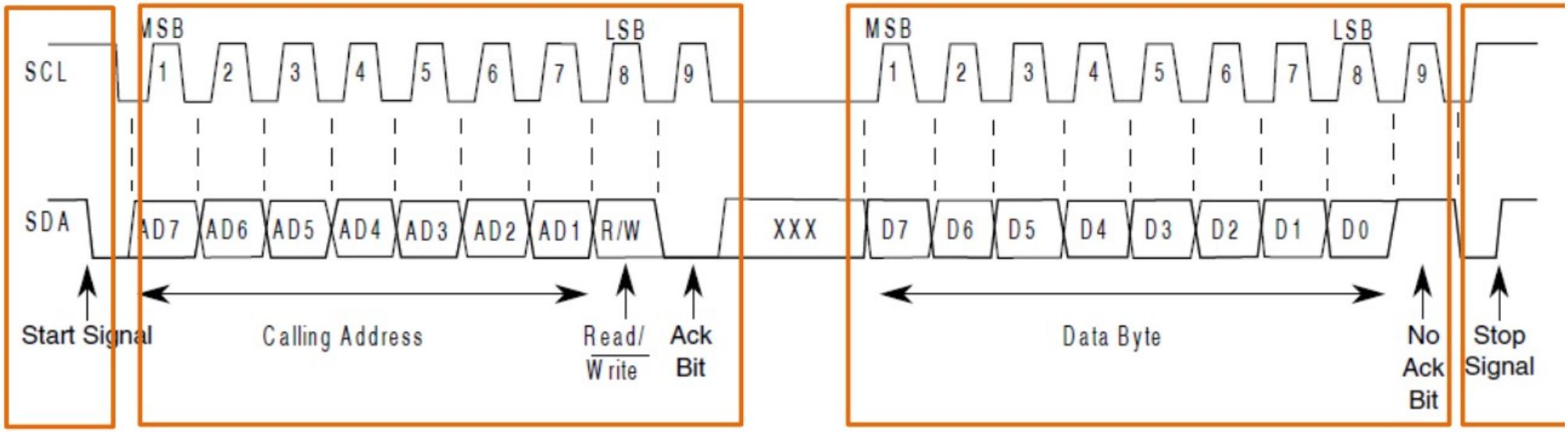
```
#include "mbed.h"
#include "bitmap.h"

// The default settings of the SPI interface are 1MHz, 8-bit, Mode 0.
SPI spi(D11, D12, D13);     // Arduino compatible MOSI, MISO, SCLK
DigitalOut DC_pin(D10);     // Data/command selector
DigitalOut RST_pin(D9);     // HW Reset
DigitalOut CS_pin(D8);      // SPI chip select

void send_cmd(unsigned char cmd, unsigned char dat) {
    DC_pin = false; //digitalWrite(RSX,LOW);
    CS_pin = false; //digitalWrite(CSX,LOW);
    spi.write(cmd|dat);
    CS_pin = true;  //digitalWrite(CSX,HIGH);
    DC_pin = true;  //digitalWrite(RSX,HIGH);
}

void send_dat(unsigned char dat) {
    DC_pin = true;  //digitalWrite(RSX,HIGH);
    CS_pin = false; //digitalWrite(CSX,LOW);
    spi.write(dat);
    CS_pin = true;  //digitalWrite(CSX,HIGH);
}
```

Az I2C busz vezérlése

- **Az I2C busz legfontosabb jellemzői:**
  - Csak két vonalat használ: az **SCL** vonal a szinkronjel (órajel), az **SDA** vonal pedig az adat jel, amelyeket open drain módban vezérlünk (csak passzív felhúzás van, egy-egy ellenállással)
  - Adatátvitel csak a busz tétlen (idle) állapotában indítható. A tétlen állapot jellemzője, hogy egy STOP feltételt követően mind az **SDA**, mind az **SCL** vonal magas szinten van (nincs aktív kimenet).
  - Soros, 8-bit-es, kétirányú adatforgalom, melynek maximális sebessége normál üzemmódban (standard mode) 100 kbit/s, gyors üzemmódban (fast mode) pedig 400 kbit/s
  - Mindegyik csatlakoztatott eszköz címezhető egy egyedi címmel.



I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

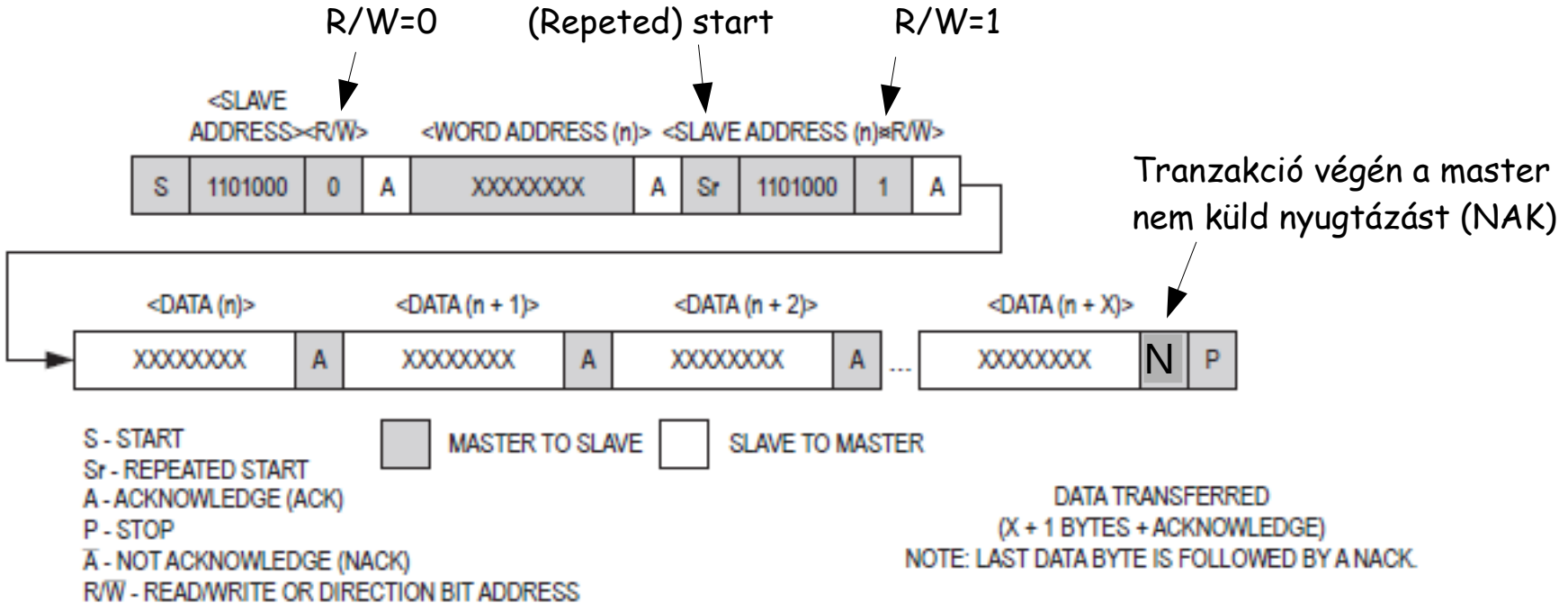
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

Nyugtázás (ACK): a 9. órajel impulzus tartamára a vevő alacsony szinten tartja az **SDA** jelvezetéket.

Negatív nyugtázás (NAK): a 9. órajel impulzus idején senki sem húzza le az **SDA** jelvezetéket – az magas szinten marad.

Csoportos adatlekérés az I2C buszon

- **Csoportos adatfogadás:** több adatbájtot fogadunk egymás után, a küldő *slave* eszköz pedig automatikusan lépteti a címet
- Csoportos adatlekérésnél írás (regiszter- vagy memória cím beállítása) és olvasás (adatbájtok) is történik, s a kettő között a ***repeated start*** feltétel generálása biztosítja a busz folyamatos elérhetőségét, majd újra ki kell küldeni a *slave* I2C címét R/W bit = 1 mellett, s kezdődhet a beolvasás



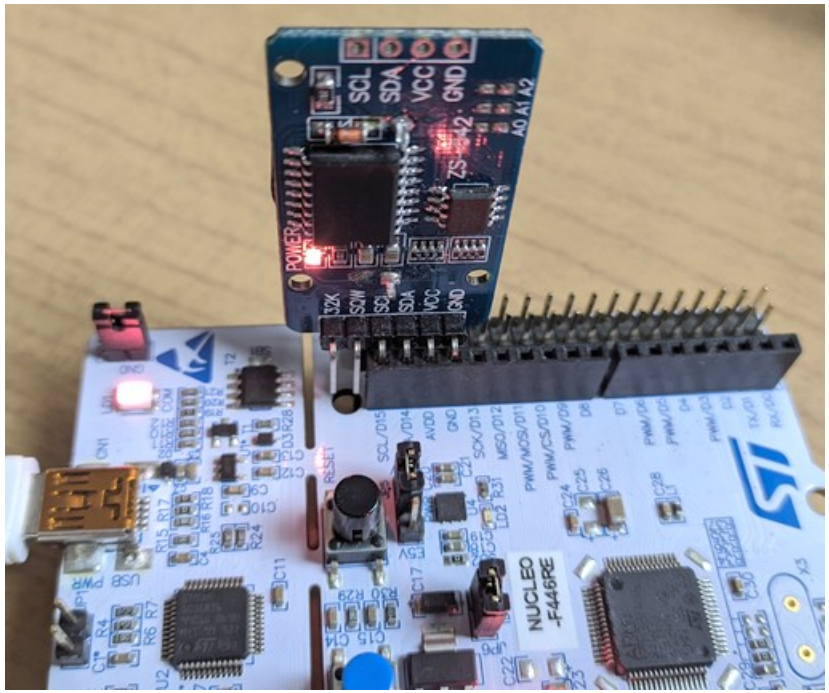
Az I2C objektumosztály

- Az **I2C** objektumosztály segítségével a a mikrovezérlőnket **master** módban használhatjuk, hogy a hozzákapcsolt **I2C** perifériákat (pl. EEPROM memória, LM75 vagy TCN75 hőmérő, BMP085/BMP180 nyomásmérő, DS1307 RTC óra, TLS2951 fénymérő, HMC5883 mágneses térmérő) vezéreljünk
- Az mbed API **I2Cslave** programkönyvtárat is biztosít, melynek segítségével a mikrovezérlőt **I2C slave** módban használhatjuk

Függvény	Használat
I2C név(sdapin, sclpin**)**	Létrehoz egy "név" nevű **I2C** objektumot, az *sdapin* és *sclpin* paraméterek szerint kiválasztja a hozzá rendelt **I2Cx** periférát, és I2C buszként konfigurálja a megnevezetett kivezetések funkcióját.
frequency(hz**)**	Az adatküldési sebesség beállítása (alapértelmezetten 100 kHz)
read(ack**)**	Egyetlen bájt olvasása nyugtázással vagy negatív nyugtázással
read(addr,*data,length**)**	Több bájt olvasása az *addr* című eszközről
write(data**)**	Egyetlen bájt küldése
write(addr,*data,length**)**	Több bájt írása az *addr* című eszközre
start()	START vagy RESTART feltétel generálása
stop()	STOP feltétel generálása

Lab05_DS3231_test

- Ebben a mintapéldában az **I2C** osztály tagfüggvényei segítségével írjuk és olvassuk a **DS3231** RTC modul regisztereit
- A kapcsolással nem kell bajlódni, mert az Arduino-kompatibilis **CN5** csatlakozó végére **(SCL, SDA, AVDD, GND)** egyszerűen odatűzhetjük az RTC modult (a **32K** és az **SQW** tüskék kimaradnak, de ezeket most úgysem használjuk)



- A programban először nullát írunk a **DS3231** vezérlő regiszterébe, majd kiolvassuk és kiíratjuk az összes regiszter tartalmát. A csoportos kiolvasáshoz először be kell állítani a kezdőcímet (itt most 0), utána indíthatjuk az adatok beolvasását (csoportos adatbeolvasás)
- A kiolvasott adatok értelmezésével most nem foglalkozunk, csak kiíratjuk azokat a soros porton keresztül

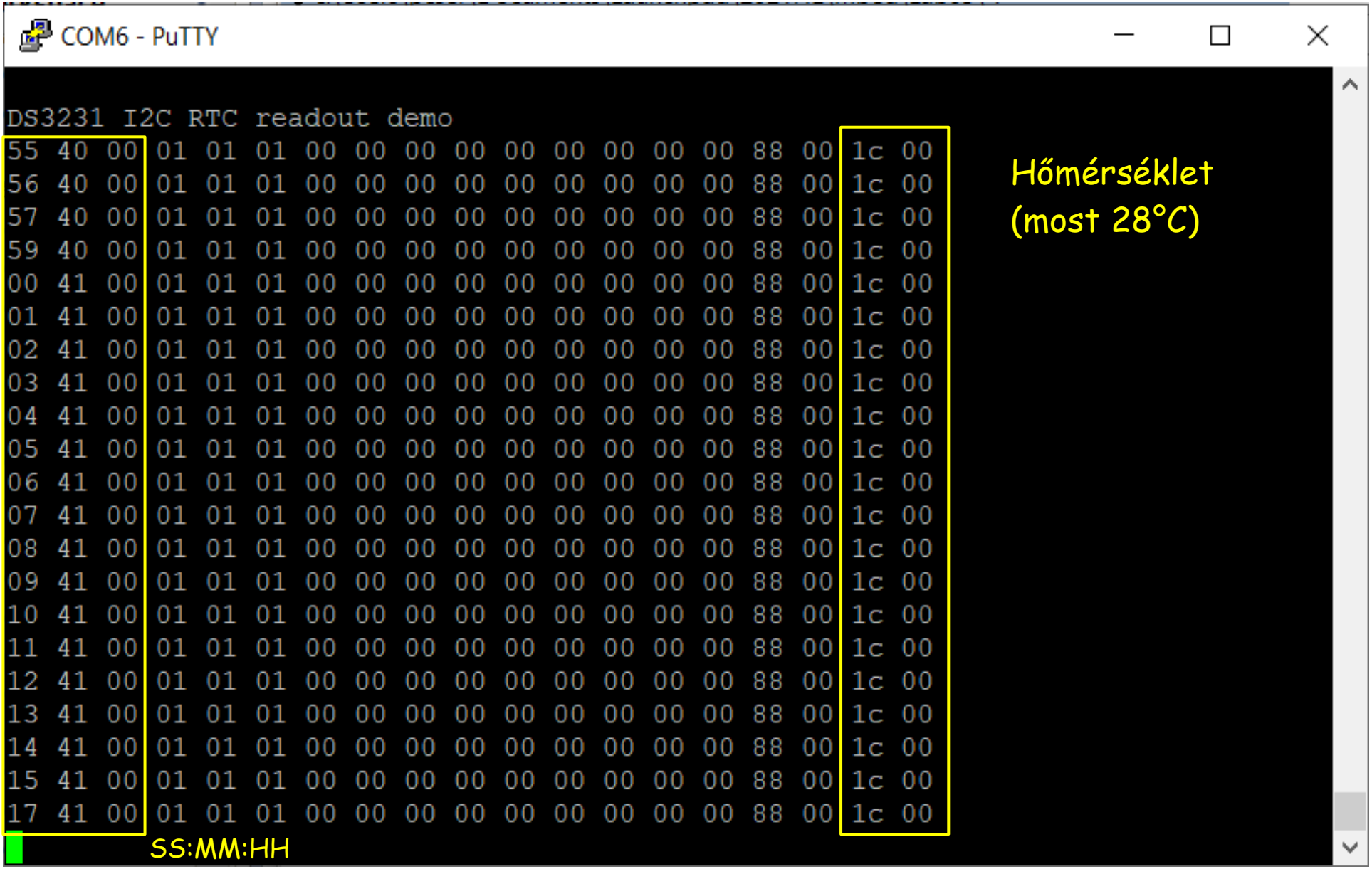
Lab05_DS3231_test/main.cpp

```
#include "mbed.h"
I2C i2c(D14, D15);                  // Arduino compatible I2C pins
const int addr = 0xD0;              // Left aligned DS3231 address: 0x68<<1

int main()
{
    char cmd[20];                   // data buffer
    printf("\r\nDS3231 I2C RTC readout demo\r\n");
    cmd[0] = 0x0E;                  // Pointer to CONFIG register
    cmd[1] = 0x00;                  // Data for CONFIG (enable clock)
    i2c.write(addr, cmd, 2);        // Send Address/command and two bytes
    while (1) {
        wait(1);
        cmd[0] = 0x00;              // Pointer to first register
        i2c.write(addr, cmd, 1);    // Write register address
        i2c.read(addr, cmd, 19);    // Read 19 bytes from the DS3231 chip
        for(int i=0; i<19; i++) { // Print data in hexadecimal format
            printf("%02x ",cmd[i]);
        }
        printf("\r\n");
    }
}
```

DS3231 regisztertérkép

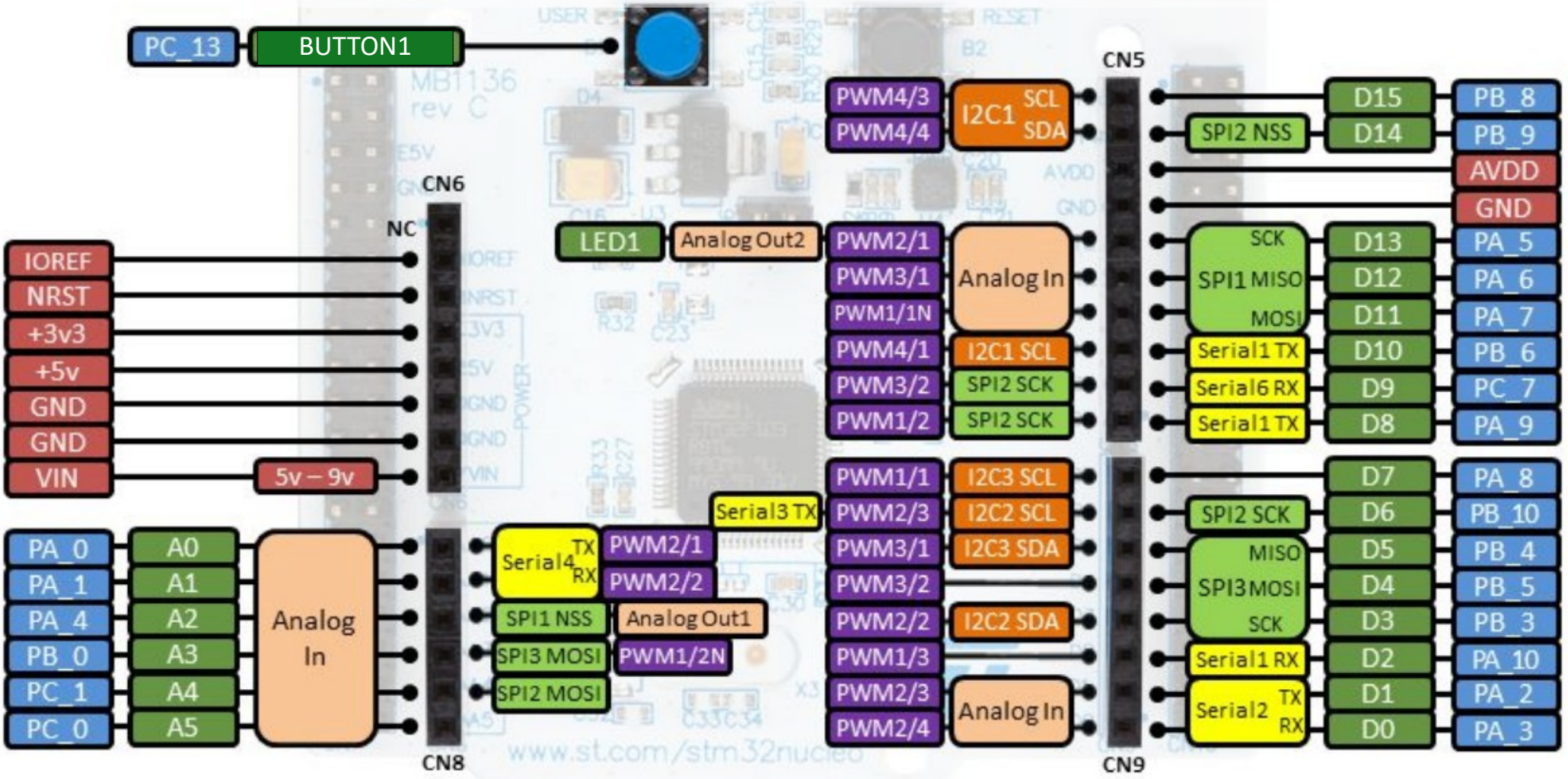
ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM / 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM / 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7
					Date				Alarm 1 Date	1–31
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM / 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7
					Date				Alarm 2 Date	1–31
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—
0Fh	OSF	0	0	0	EN32kHz	BSY	A2F	A1F	Control/Status	—
10h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	Aging Offset	—
11h	SIGN	DATA	DATA	DATA	DATA	DATA	DATA	DATA	MSB of Temp	—
12h	DATA	DATA	0	0	0	0	0	0	LSB of Temp	—

Lab05_DS3231_test futási eredmény



Arduino kompatibilis kivezetések

- Kivezetés azonosításra a kék, illetve a sötétzöld címkék használhatók (pl. PA_5, D13, vagy LED1)



Morpho kivezetések

Nucleo F446RE

Morpho Headers