STM32 mikrovezérlők programozása STM32CubeIDE környezetben

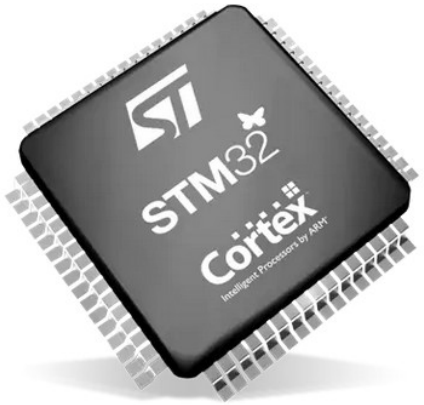
8. Az I2C kommunikációs csatorna – 1. rész

Felhasznált és ajánlott irodalom

- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi:
  STM32 Arm Programming for Embedded Systems
- Carmine Noviello: **Mastering STM32**

  A könyv mintapéldái: **github.com/cnoviello/mastering-stm32**

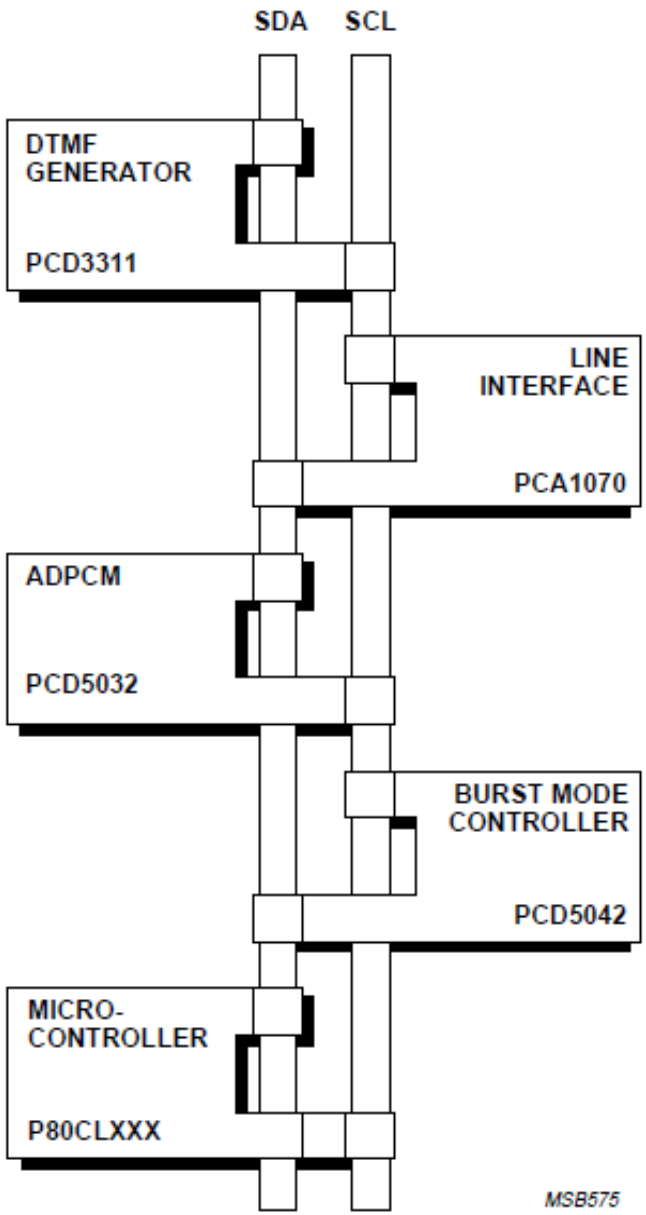
Adatlapok, kézikönyvek:



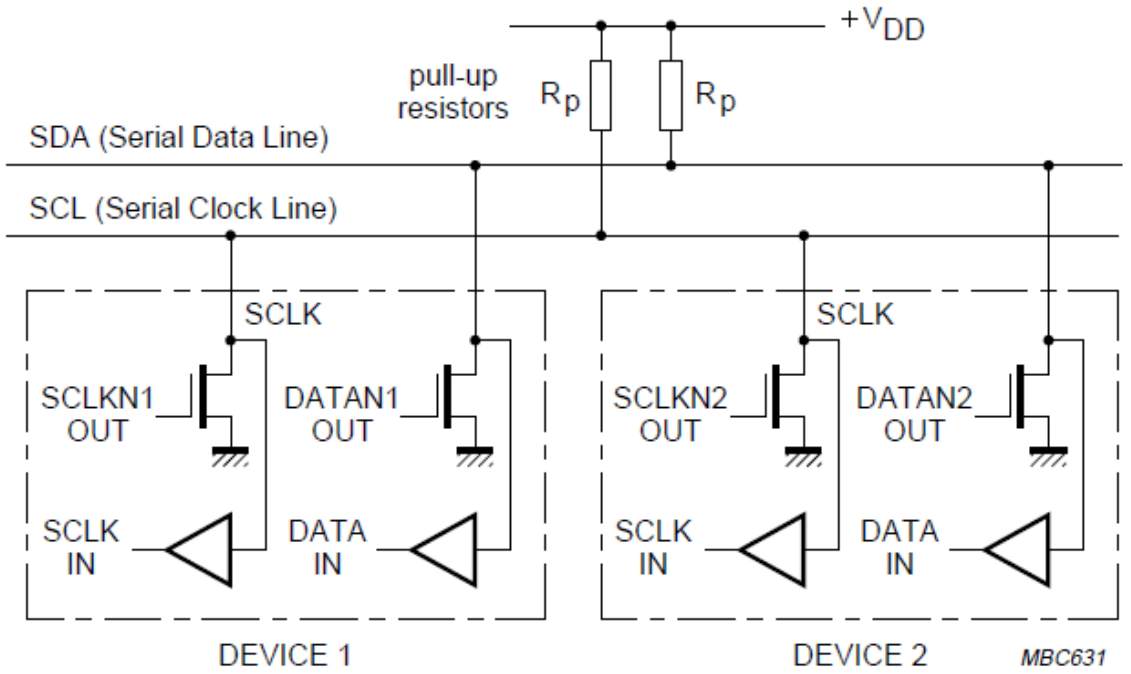
- **STM32F103C8 adatlap és termékinfo**
- **STM32F103 Family Reference Manual**
- **STM32F446RE adatlap és termékinfo**
- **STM32F446 Family Reference Manual**
- STmicro: **Description of STM32F4 HAL and low-layer drivers**
- Maxim Integrated: **DS3231 RTC adatlap**
- Bosch Sensortec: **BME280 adatlap**

Az I2C busz

- „Inter-Integrated Circuit” busz – vagy „kétvezetékes busz”, amelyet eredetileg a Philips cég dolgozott ki 1982-ben.
- **Több eszköz** (master és slave) **fűzhető fel a buszra**
- A buszt a **mester (master) eszközök** vezérlik és kezdeményezik az adatforgalmat. **A szolga (slave) eszköz** akkor válaszol, ha címzéssel megszólítják
- **Az I²C busz két jelvezetéket használ**
  - **SCL:** szinkronizáló órajel
  - **SDA:** soros adat
- **A részletes leírás** az ”Az I²C-busz specifikációja és használata” című dokumentumban olvasható

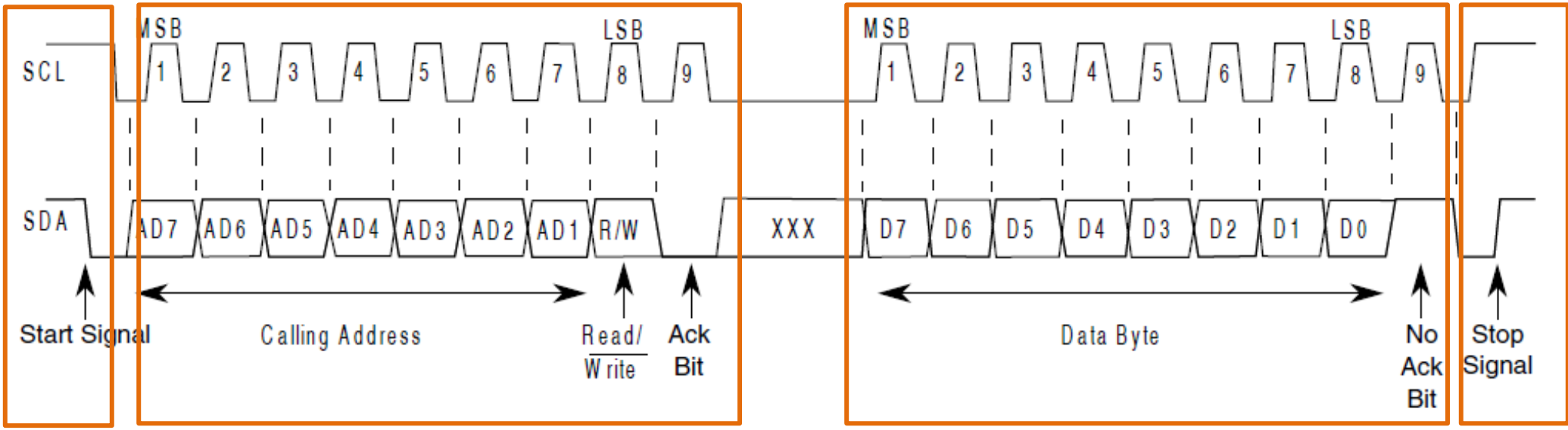


Az I2C busz vezérlése



- A jelvezetékeket ellenállások húzzák fel tápfeszültségre V_{DD}
- Nyitott nyelőelektródás FET-ek húzzák le a vonalakat alacsony szintre
- A buszt vezérlő mester eszköz állítja elő az SCL órajelet
  - normál mód: 100 kHz
  - gyors mód: 400 kHz
  - nagysebességű mód: 1 MHz, vagy több, ez esetben már aktív felhúzással.

I2C üzenetformátum



Üzenet-orientált adatátvitel négy felvonásban:

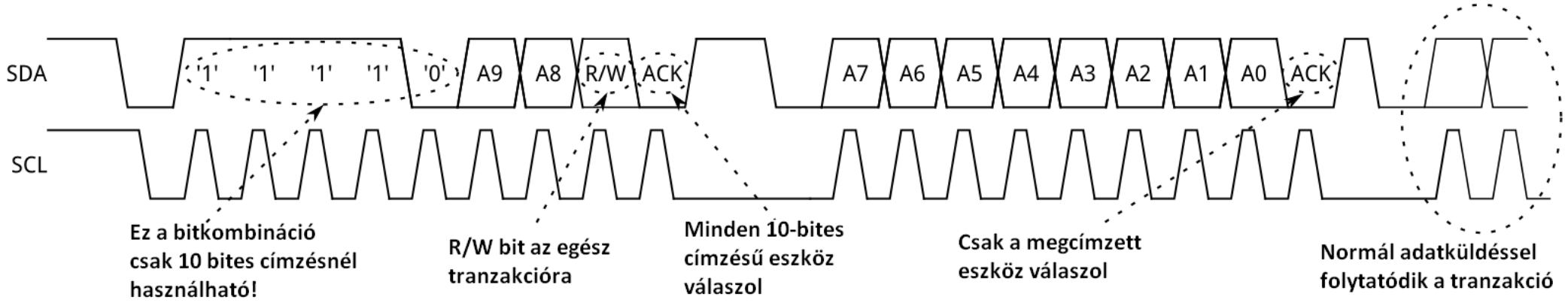
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

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.

10-bites címzés az I2C buszon

- A 10-bites címzés az eredetileg 7-bites címzést használó I2C protokoll bővítése.
- A 10 bites címet csak két részletben tudjuk kiküldeni.
- Az első rész egy speciális bitsorozattal (11110) kezdődik. Ez a bitkombináció a 10 bites címzés számára fenntartott, tehát 7 bites címzésnél ez a bitsorozat nem használható eszközcímként.
- Az első rész kiküldése után minden 10-bites címzést használó eszköz küld nyugtázást.
- A második bájtban a cím többi bitjeit (A0…A7) küldjük ki. Erre már csak az az eszköz válaszol, amelynek mind a 10 címbitje megegyezik az általunk kiküldöttekkel.
- Az adatküldés a továbbiakban ugyanúgy folyik tovább, mint a 7-bites címzésnél.

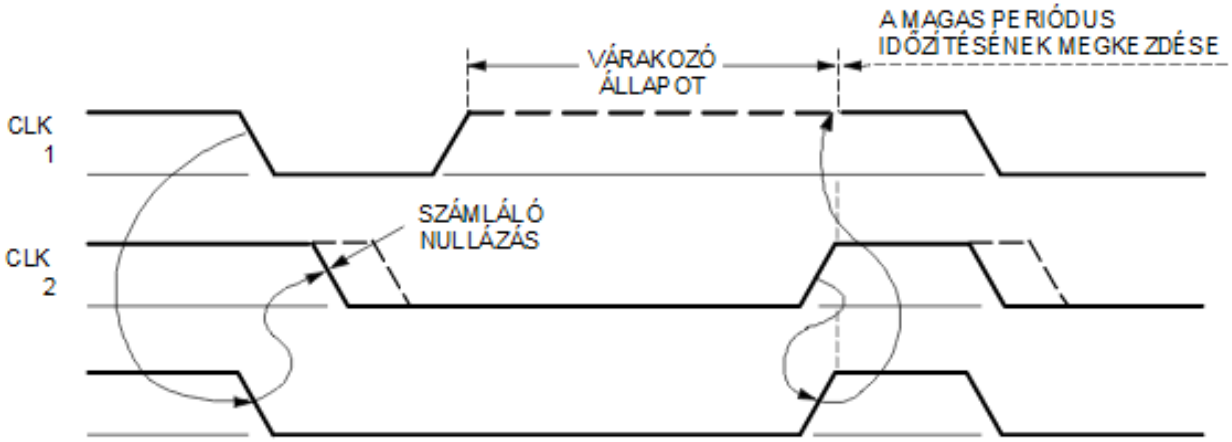


Órajel szinkronizálás

Minden master a saját órajelét generálja az **SCL** vezetéken ahhoz, hogy üzenetet vigyen át az I²C-buszon.

Az órajelszinkronizáció az I²C interfészek huzalozott **ÉS** kapcsolatával megy végbe (a vezetéken csak akkor lesz magas szint, ha minden eszköz magas szintre vált).

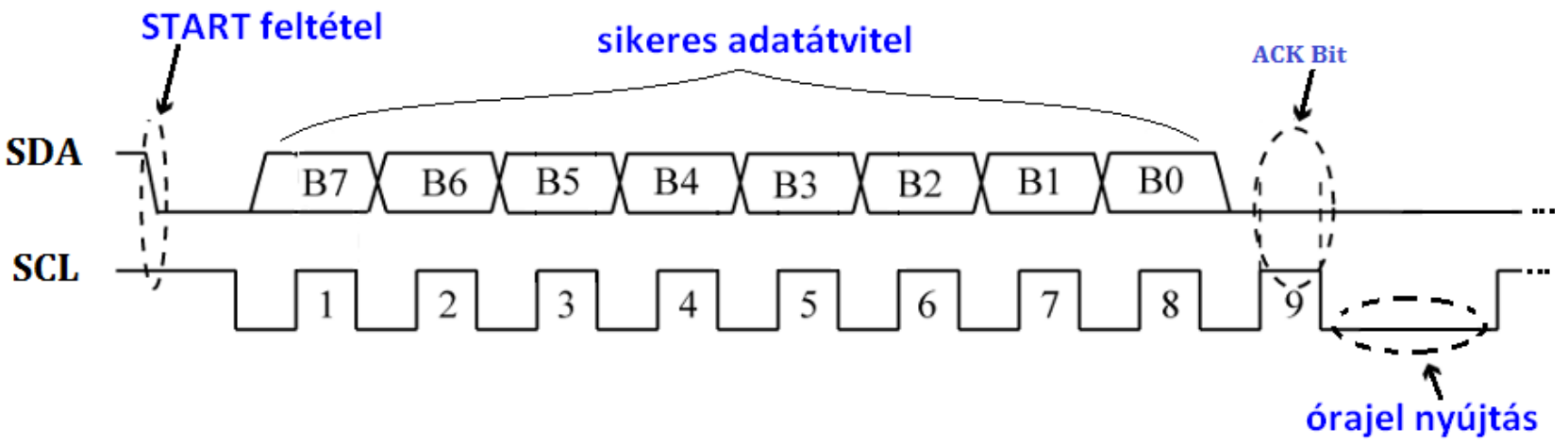
- Az **SCL** vezetéken egy **magas-alacsony átmenet** az érintett eszközöknél az alacsony periódusuk időzítésének megkezdését eredményezi. Ha egy eszköz órajele alacsonyra váltott egészen addig alacsony állapotban tartja az **SCL** vezetéket, amíg az órajele magas periódusához nem ér.
- Az **SCL** vezetéket a leghosszabb alacsony periódusú eszköz tartja alacsonyan. Erre az időre a rövidebb alacsony periódussal rendelkező eszközök **magas szintre várakozó állapotba** kerülnek.



Órajel megnyújtása (clock stretching)

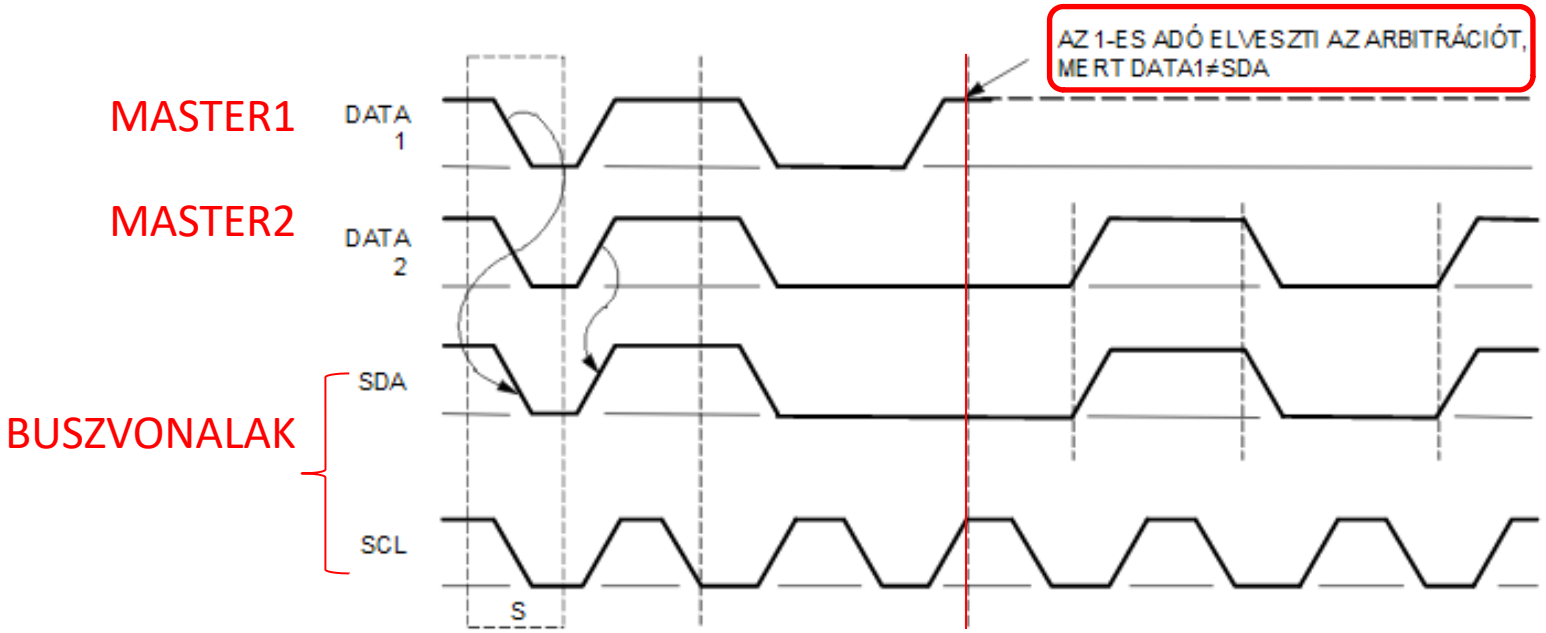
Az órajelet a mester generálja, de előfordulhat, hogy egy slave eszköz nem képes együttműködni a diktált tempóval, s „időt kér”. A már említett órajelszinkronizációs mechanizmus használható fel erre. Ez azt jelenti, hogy egy eszköz képes lehet az adatbájtok gyors fogadására, de az adat eltárolására vagy átvitelhez való előkészítéséhez több időre van szüksége. A slave ilyenkor a fogadás és a nyugtázás után **alacsony szinten tarthatja** az **SCL** vezetéket, hogy a mastert várakozási állapotba kényszerítse, amíg a slave kész nem lesz a következő bájt átvitelére, mint egy kézfogásos típusú eljárásban.

Nomál és gyors (fast) módban az adatátvitel bármelyik fázisában megnyújthatja a slave az órajelet. Nagysebességű (high-speed) módban azonban, ahol az órajelek fehúzásában aktív elemek is részt vesznek, csak a nyugtázó bit után és az azt követő adatbit előtt megengedett az SCL vonal lehúzása.



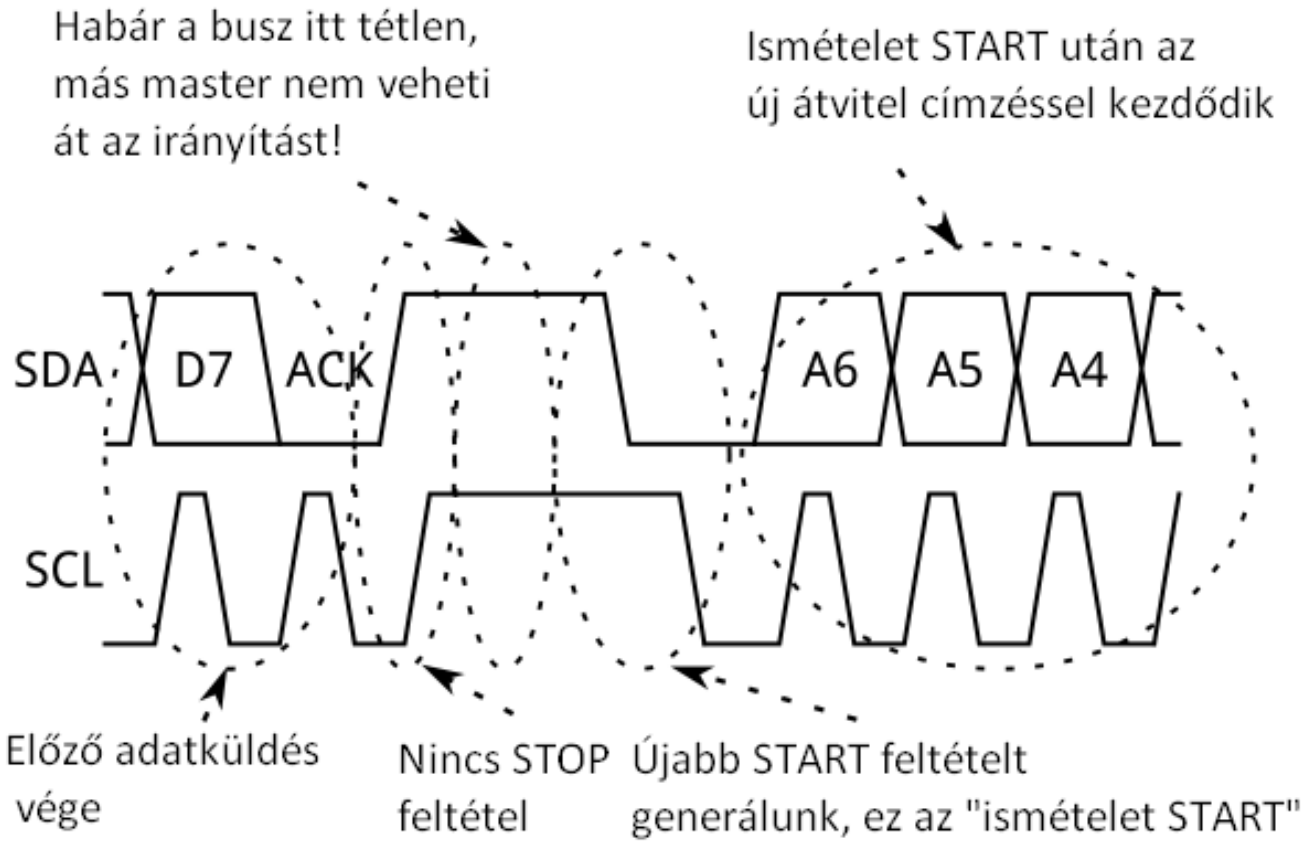
Arbitráció

- **Multi-master** környezetben csak akkor kezdhetünk átvitelt, ha a busz szabad.
- Két, vagy több master generálhat egy start feltételt a START feltétel minimális tartási idején belül. Amig az SCL vezeték magas szinten van az SDA vezetéken végbemegy az arbitráció.
- Az a master veszít, amelyik magas szintet küld, miközben egy másik master alacsonyat. A vesztes master lekapcsolja az adatkimeneti fokozatát, mert a busz jelszintje nem egyezik meg a saját jelszintjével.
- Az arbitráció több biten keresztül folytatódhat



Ismételt START feltétel (repeated Start)

Gyakran szükség van arra, hogy egy összetett tranzakciót egy master egy menetben végigvihessen. Ilyen eset lehet például egy eszköz valamely regiszterének vagy memóriaterületének megcímzése, majd onnan adatok beolvasása. Ha az adatküldés és adatfogadás között nem generálunk **STOP** feltételt, akkor a master magánál tudja tartani a busz feletti vezérlést. A **STOP** nélkül generált újabb **START** feltételt **ismételt START**-nak (repeated START) nevezzük. Az **ismételt START** feltételt újabb cím/parancs bájt küldése kell, hogy kövesse.



Az STM32 mikrovezérlők I2C perifériái

Az **STM32** mikrovezérlők több I2C modult is tartalmaznak amelyek általban az alábbi tulajdonságokkal rendelkeznek:

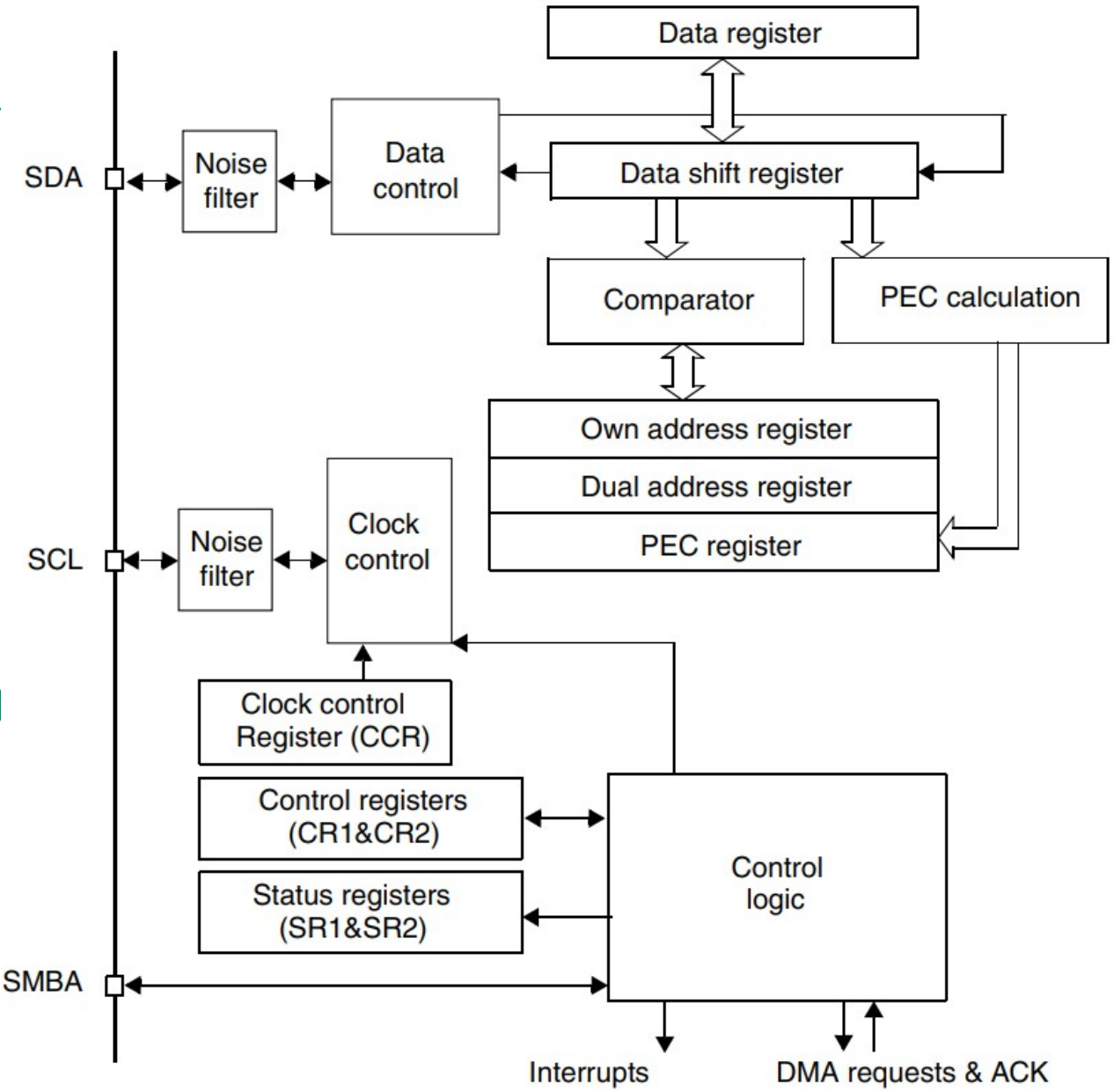
- Multimaster környezetben is használhatók, Master/Slave mód
- I2C Master jellemzők: órajel generálás, Start és Stop feltétel generálás
- I2C Slave jellemzők: két programozható I2C cím felismerése, 7/10 bites cím, ill. általános hívás felismerése, Stop detektálás
- Kommunikációs sebesség: normál (100 kHz-ig), gyors (400 kHz-ig) (**STM32F446RE** Fast-mode plus I2C csatornája 1 MHz-ig)
- Analóg zajszűrés, állapot- és hibajelző bitek, megszakítások
- 1 bájtos buffer, DMA lehetőséggel
- Konfigurálható PEC (packet error checking)
- SMBus 2.0/PMbus kompatibilitás (SMB = System Management Bus)

I2C csatornák elérhetőségei

STMF446RE	SCL	SDA
I2C1	PB6	PB7
	PB8	PB9
I2C2	PB10	PC12
	–	PB3
I2C3	PA8	PC9
	–	PB4
FMP-I2C1	PC6	PC7
STMF103C8	**SCL**	**SDA**
I2C1	PB6	PB7
	PB8	PB9
I2C2	PB10	PB11
	–	–

Az I2C modulok blokkvázlata

- A modul alaphelyzetben slave módban van, csak a START feltétel generálásakor kerül master módba
- A bemenő órajel normál módban legalább 2 MHz, gyors módban pedig legalább 4 MHz legyen
- Az **SMBA** (SM bus Alert) jel csak SMBus módban áll rendelkezésre



A HAL_I2C fontosabb függvényei

- HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout) - ellenőrzi, hogy a megcímzett eszköz válaszol-e (pl. a flash memória befejezte-e az írás műveletet…)
- HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout) - master módú írás a megcímzett slave eszközbe
- HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint8_t * pData, uint16_t Size, uint32_t Timeout) - master módú olvasás a megcímzett eszközből
- HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout) - master módú írás: az adatok kiküldése előtt egy címet is kiküldve
- HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef * hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t * pData, uint16_t Size, uint32_t Timeout) - master módú olvasás: az adatok olvasása előtt egy címet is kiküldve, majd repeat start feltétel után az olvasással folytatva

Példaprogramok

F446RE_I2Cscan – az I2C buszon található eszközök címének felderítése

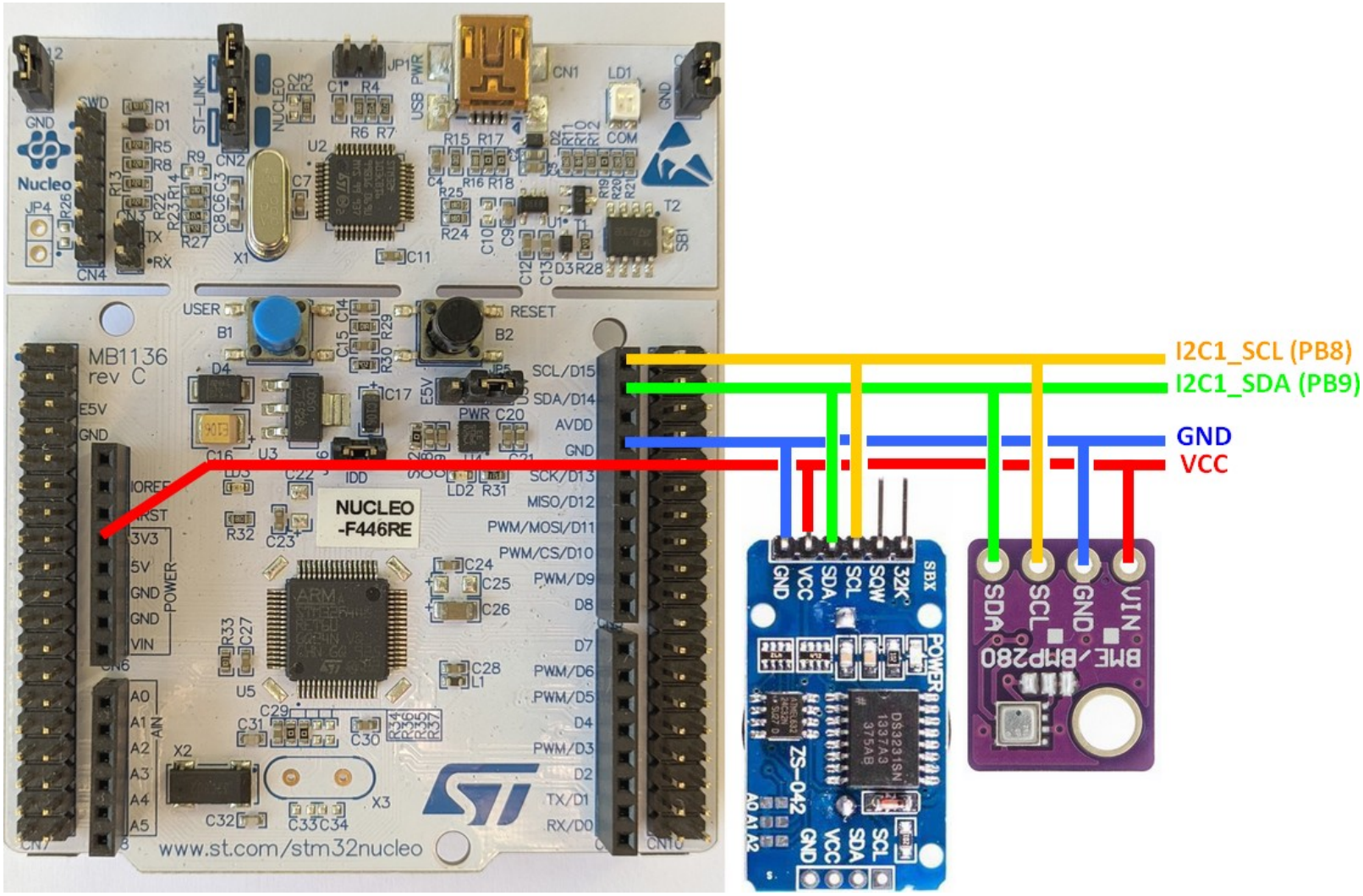
F446RE_DS3231RTC – Real-Time óra kezelése (írás, olvasás)

F446RE_BME280 – nyomásmérő, hőmérő és relatív páratartalom mérő szenzor használata

F103 projektek – a fenti példákat STM32F103C8-ra is adaptáltuk!

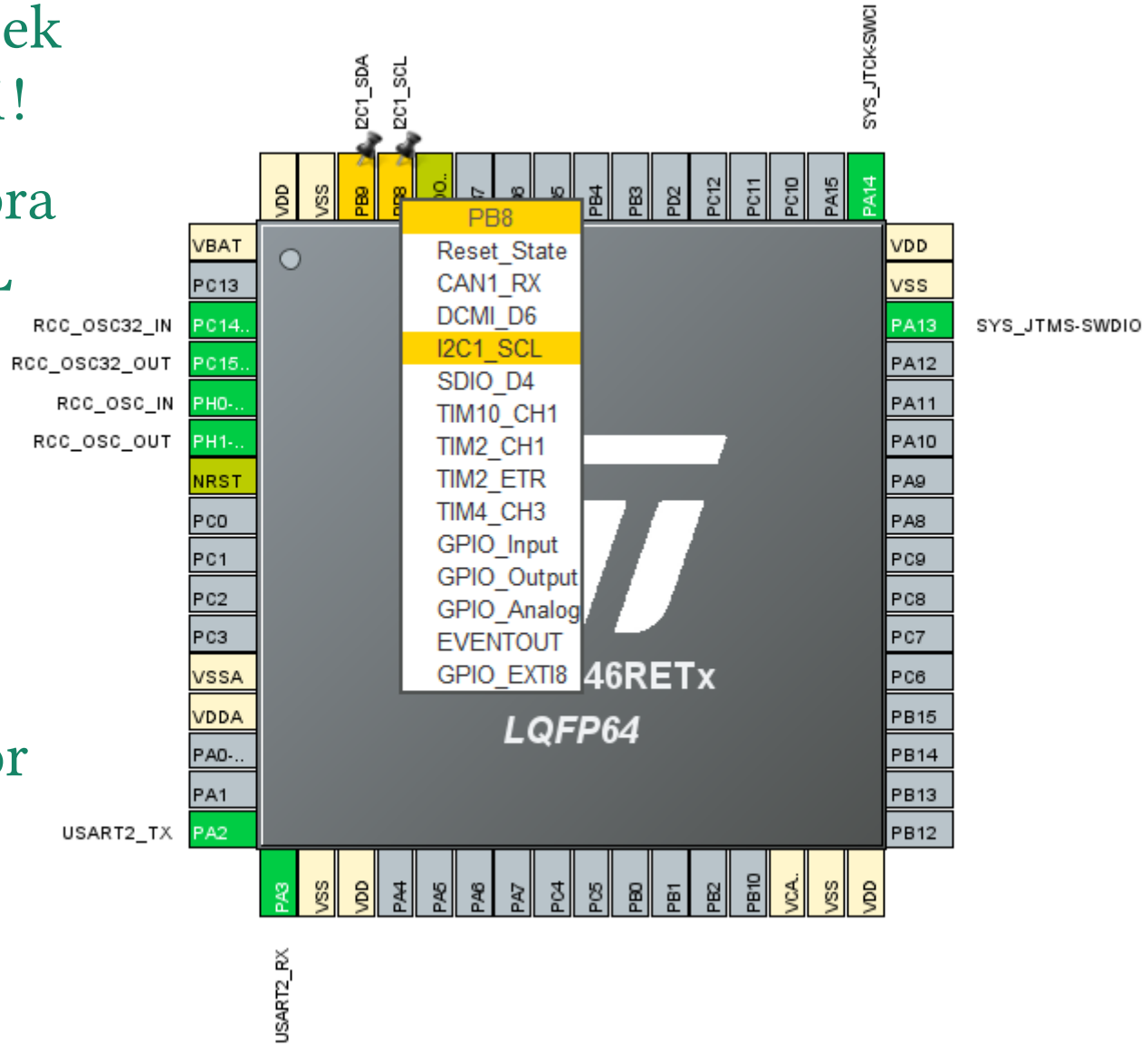
F446RE_I2Cscan – az I2C busz felderítése

- Az első példaprogram felderíti és kiírja az aktív I2C eszközök címeit
- Felhasznált forrás: github.com/ProjectsByJRP/stm32_hal_i2c_bus_scan



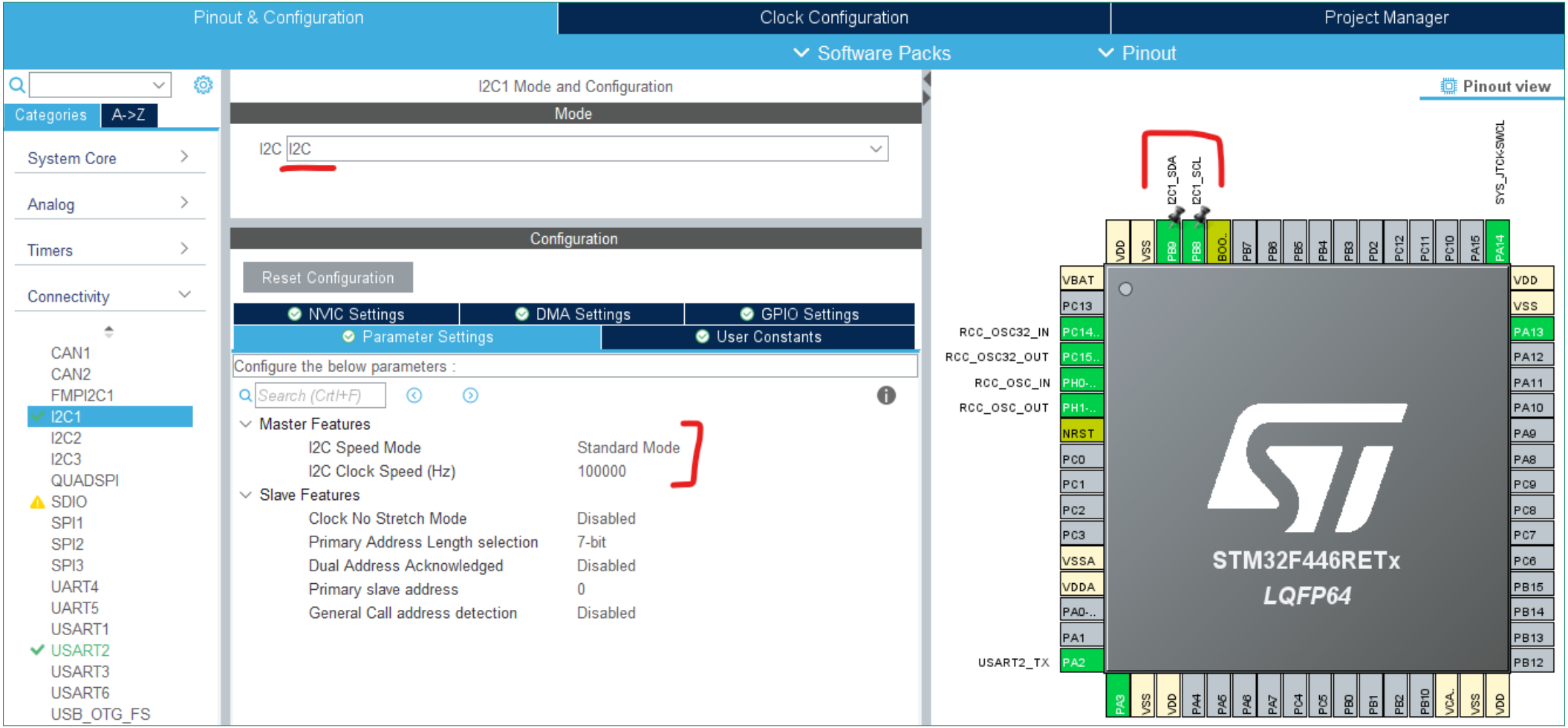
Az I2C1 kivezetések konfigurálása

- Ha nem az alapértelmezett lábkiosztást akarjuk használni, akkor a konfigurálást a kivezetések beállításával kell kezdeni!
- Kattintsunk rá a **PB8** lábra és válasszuk az **I2C_SCL** alternatív funkciót!
- Kattintsunk rá a **PB9** lábra és válasszuk az **I2C_SDA** funkciót!
- Ekkor **PB8** és **PB9** még narancssárga (majd akkor zöldülnek ki, ha engedélyezzük az **I2C1** csatornát)



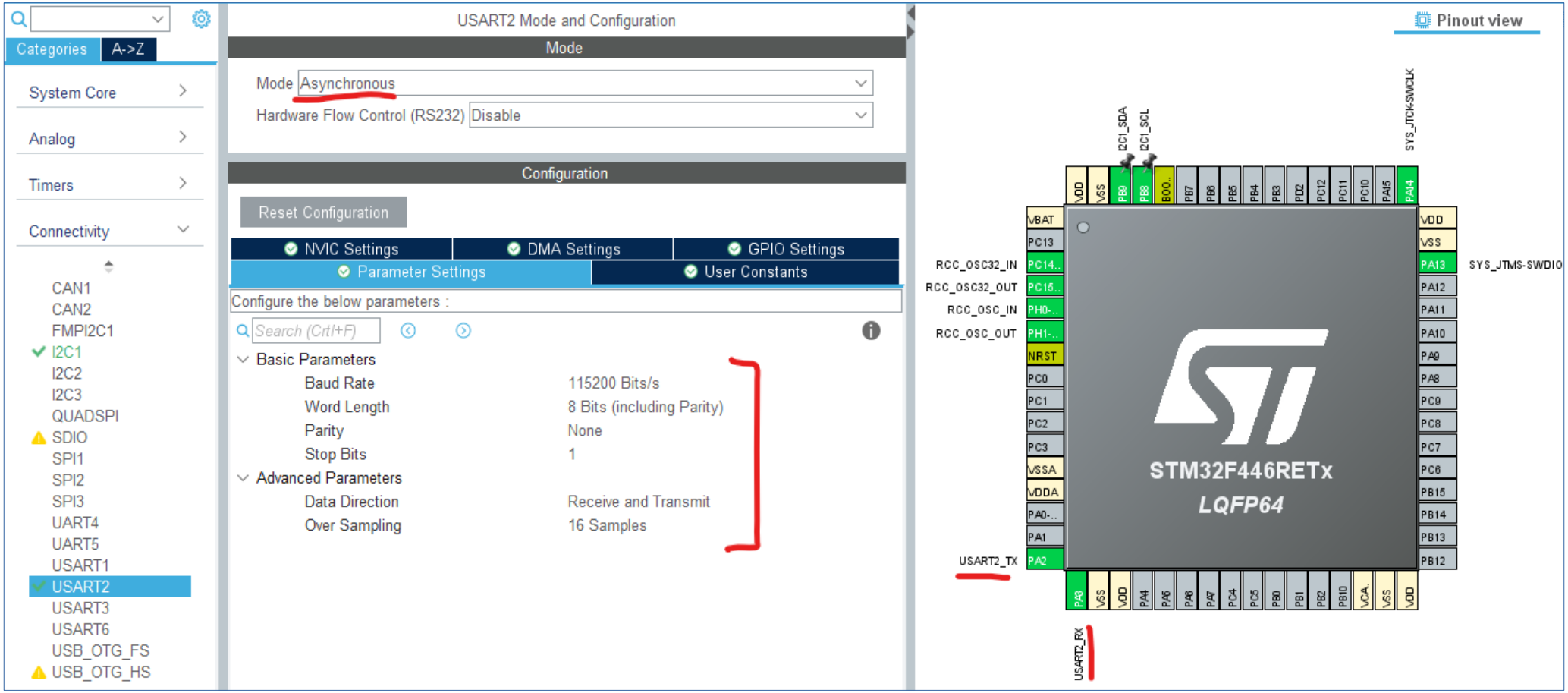
Az I2C1 periféria konfigurálása

- Engedélyezzük az **I2C** módot
- Az alapértelmezett **Standard mód**-ot és a **100 kbit/s** sebességet választjuk



Az USART2 periféria konfigurálása

- Engedélyezzük az Aszinkron módot
- Az alapértelmezett paramétereket (115 200 bps) hagyjuk



F446RE_I2Cscan: main.c (részlet 2/1.)

```c
#include "main.h"
#include "stdio.h"
I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART2_UART_Init(void);

/* Retarget printf output to UART2 Tx */
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
  HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
#ifndef __UUID_H
#define __UUID_H
//#define STM32_UUID ((uint32_t *)0x1FF0F420)
#define STM32_UUID ((uint32_t *)UID_BASE)
#endif //__UUID_H
```

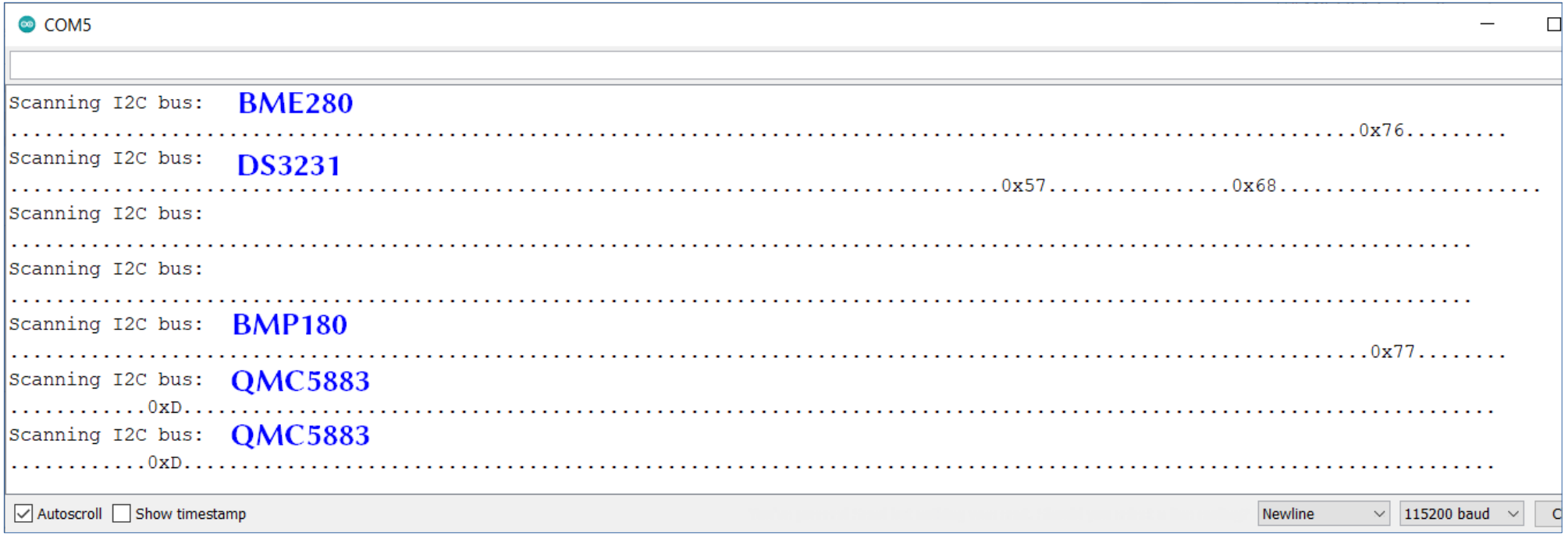
Printf kimenet (a standard output) átirányítása az UART2 kimenetre

F446RE_I2Cscan: main.c (részlet 2/2.)

```c
int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();
  HAL_Delay(2000);
  printf("Scanning I2C bus:\r\n");
  HAL_StatusTypeDef result;
  for (uint8_t i=1; i<128; i++) {
    /* HAL requires left aligned i2c addresses
     * &hi2c1 is the handle
     * (uint16_t)(i<<1) is the i2c address left aligned
     * retries 2
     * timeout 2
     */
    result = HAL_I2C_IsDeviceReady(&hi2c1, (uint16_t)(i<<1), 2, 2);
    if (result == HAL_OK) {
      printf("0x%X", i);    // Received an ACK at that address
    }
    else {
      printf(".");          // No ACK received at that address
    }
  }
  printf("\r\n");
  while (1) { }
}
```

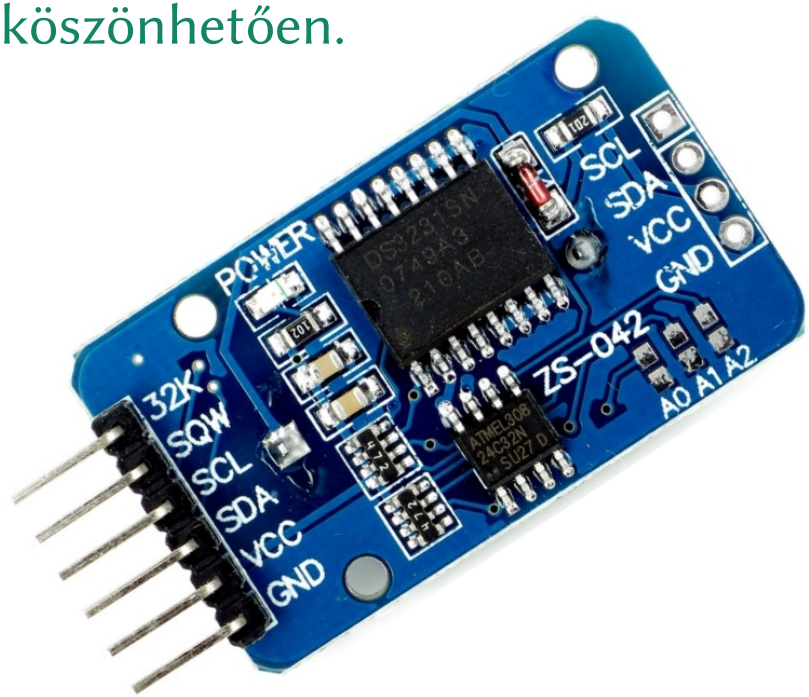
F446RE_I2Cscan: futási eredmény

- Az **I2C** buszra különböző eszközöket csatlakoztatva, megjelenik a felderített eszköz címe
- A **DS3231** real-time óra (0x68) kártyán egy flash memória is található (0x57), ezért jelenik meg két cím is a buszon
- A **BME280/BMP280** modulok címe attól függ, hogy az **SDI** láb alacsony, vagy magas szintre van húzva



DS3231 Real-time óra modul

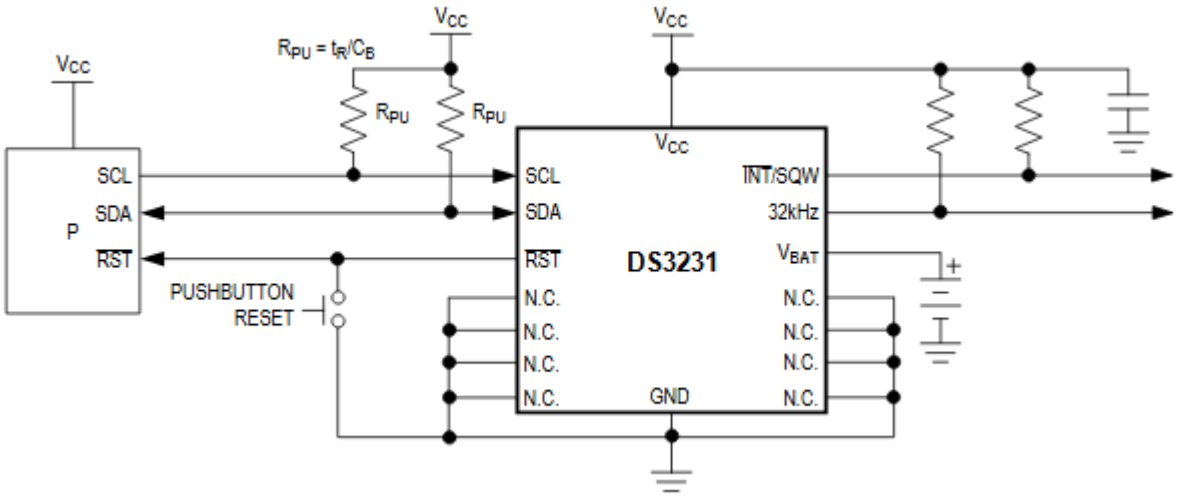
- Oszcillátor, óra és naptár egy tokban. A tápfeszültség megszűnésekor a hátoldalán elhelyezett telepről üzemel tovább.
- A modul többé-kevésbé cserekompatibilis a **DS1307**-tel, egy 4 kB EEPROM is tartalmaz, de van néhány eltérés:
  - pontosabb óra (±2 ppm a –40 – 80 °C tartományban) a beépített hőkompenzált oszcillátornak köszönhetően.
  - két riasztási időpont is megadható
  - van beépített hőmérője
  - nincs belső RAM
  - Vbat 4.2 V-os Li akkuval is táplálható!
- EEPROM I2C címe: **0x57** (állítható)
- DS3231 I2C címe: **0x68**



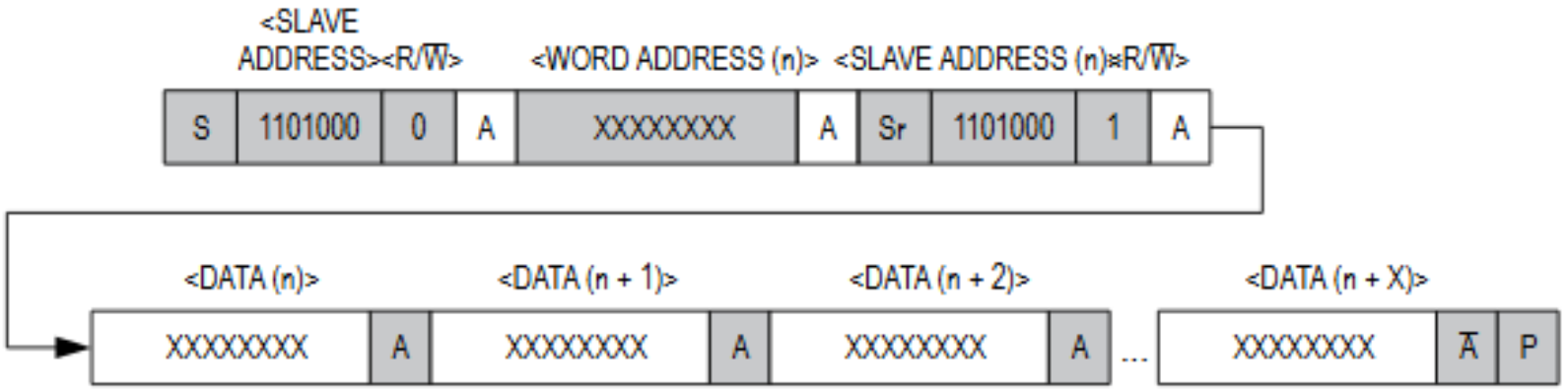
- Adatlap: datasheets.maximintegrated.com/en/ds/DS3231.pdf

A DS3231 bekötése, használata

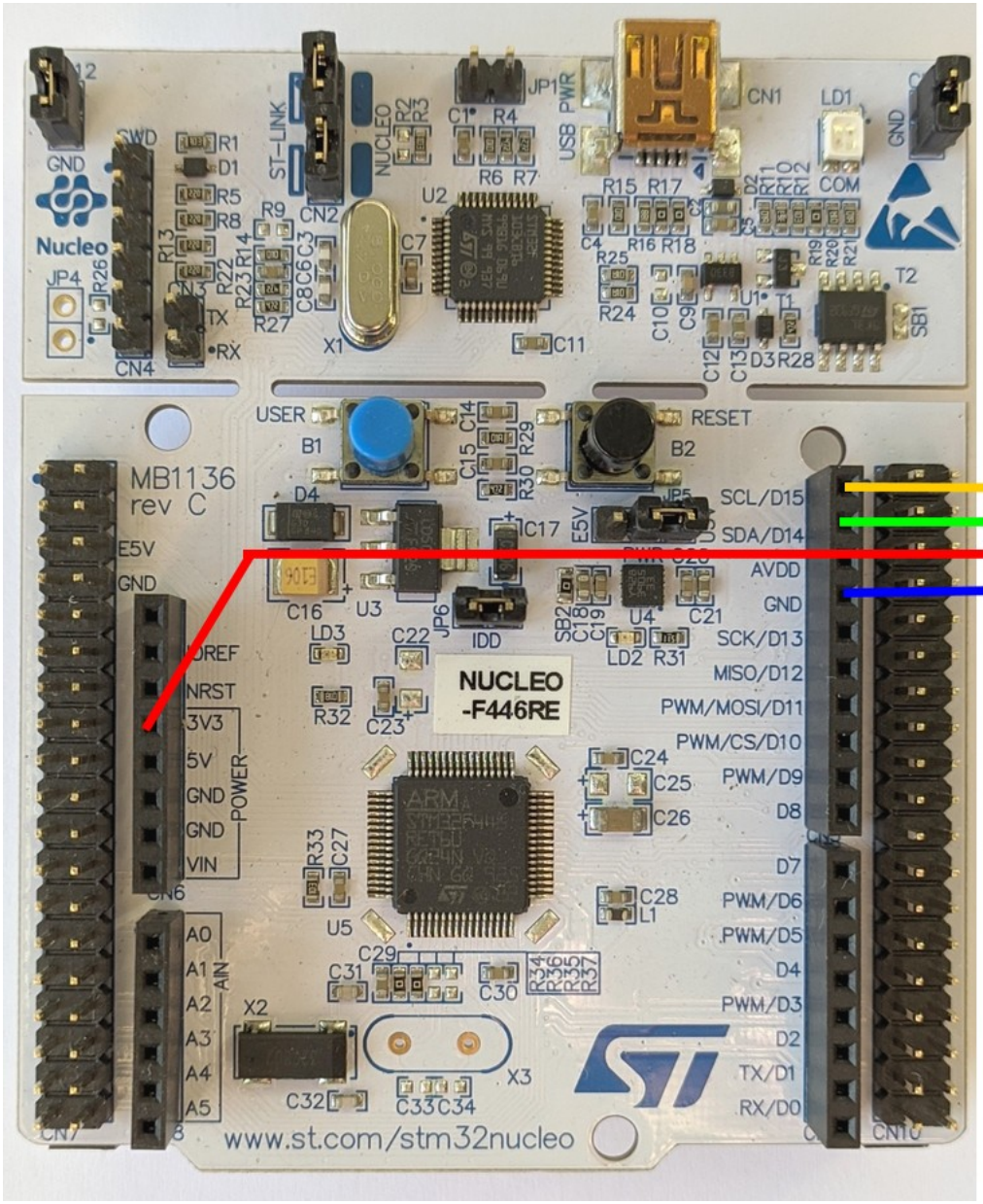
- Egy tipikus áramköri elrendezés:



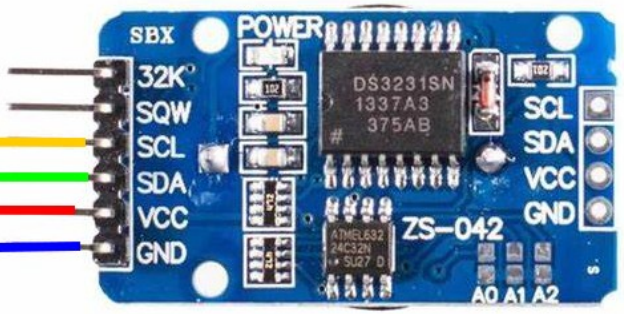
- Adatforgalom az **I2C** buszon:



Bekötési vázlat



DS3231 modul



SDA – PB9
SCL – PB8

UART2 Tx – PA2
UART2 Rx - PA3

DS3231 regiszterek

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	—

A DS3231 programkönyvtár

- Mintapéldánkban *Jihoon Lee (Eziya)* **DS3231** programkönyvtárát és projektjét adaptáltuk (github.com/eziya/STM32_HAL_DS3231)
- `void DS3231_Init(I2C_HandleTypeDef *handle);`
- `bool DS3231_GetTime(_RTC *rtc);`
- `bool DS3231_SetTime(_RTC *rtc);`
- `bool DS3231_ReadTemperature(float *temp);`
- `bool DS3231_SetAlarm1(AlarmMode mode, uint8_t date, uint8_t hour, uint8_t min, uint8_t sec);`
- `bool DS3231_ClearAlarm1();`
- `bool ReadRegister(uint8_t regAddr, uint8_t *value);`
- `bool WriteRegister(uint8_t regAddr, uint8_t value);`
- Az **F446RE_DS3231RTC** projektben inicializáljuk, illetve periodikusan kiolvassuk az óra modult (idő, hőmérséklet), majd az **UART2** soros porton keresztül (**PA2**, **PA3**) kiíratjuk az eredményt
- A projekt létrehozása és konfigurálása ugyanúgy történik, mint az előző példaprogramban

Forrásfájlok hozzáadása a projekthez

- A grafikus konfigurálás után a létrehozott projekthez még hozzá kell adnunk a github.com/eziya/STM32_HAL_DS3231 archívumból letöltött `stm32_ds3231.h` és `stm32_ds3231.c` forrásfájlokat, amire több út is kínálkozik:
  - Bemásoljuk a fájlokat egy üres mappába, a mappát importáljuk az **STM32CubeIDE** környezetbe, majd egérrel áthúzzuk a fájlokat a projektünkbe
  - File→New paranccsal új header ill .c forrásállományt hozunk létre, majd ezekbe Copy/Paste (Ctrl-C/Ctrl-V) módszerrel bemásoljuk a letöltött `stm32_ds3231.h` és `stm32_ds3231.c` forrásfájlok tartalmát
- A forrásfájlok hozzáadása után nyissuk meg a `main.c` állományt, majd egészítsük ki a következő oldalakon bemutatott programlista szerint!

F446RE_DS3231RTC: main.c (részlet 2/1.)

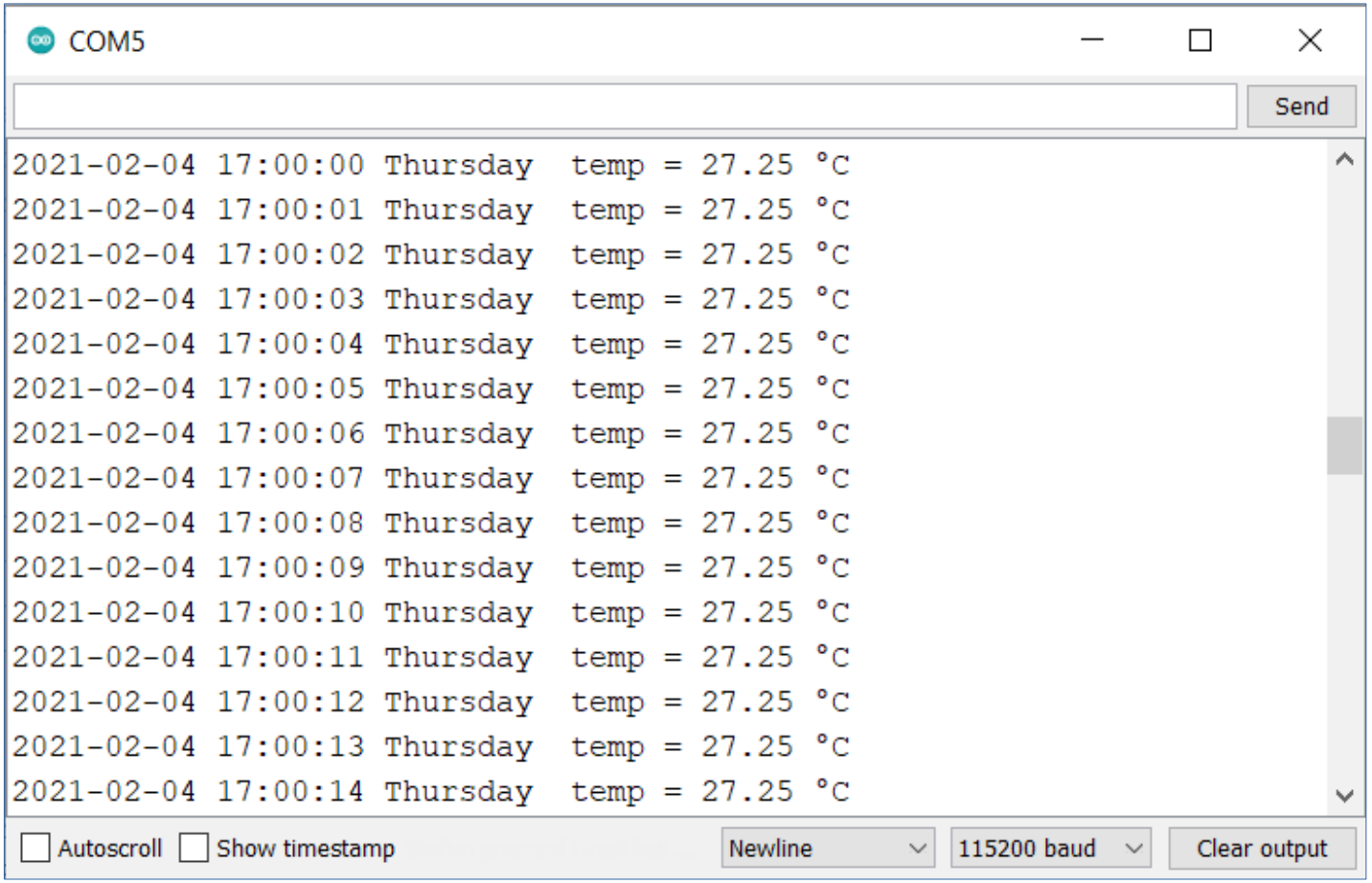
```c
#include "main.h"
#include "stm32_ds3231.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART2_UART_Init(void);

_RTC rtc = {
    .Year = 21, .Month = 2, .Date = 4,
    .DaysOfWeek = THURSDAY,
    .Hour = 17, .Min = 0, .Sec = 0
};

uint8_t regVal;
float rtcTemp;
uint8_t status;
char msg[80];
char dow[8][12] = {"Error", "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
```

F446RE_DS3231RTC: main.c (részlet 2/2.)

```
int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();

  DS3231_Init(&hi2c1);
  status = DS3231_SetTime(&rtc);
  DS3231_ClearAlarm1();
  // DS3231_SetAlarm1(ALARM_MODE_ONCE_PER_SECOND, 0, 0, 0, 0);
  DS3231_SetAlarm1(ALARM_MODE_SEC_MATCHED, 0, 0, 0, 30);

  while (1) {
    status = DS3231_GetTime(&rtc);
    status = DS3231_ReadTemperature(&rtcTemp);
    ReadRegister(DS3231_REG_STATUS, &regVal);
    if(regVal & DS3231_STA_A1F) { // Ha bebillent az Alarm jelzőbit...
      regVal &= ~DS3231_STA_A1F;
      WriteRegister(DS3231_REG_STATUS, regVal);
    }
    sprintf(msg, "20%02d-%02d-%02d %02d:%02d:%02d %s  temp = %5.2f °C\r\n",
    rtc.Year,rtc.Month,rtc.Date,rtc.Hour,rtc.Min,rtc.Sec, dow[rtc.DaysOfWeek],rtcTemp);
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
    HAL_Delay(1000);
  }
}
```

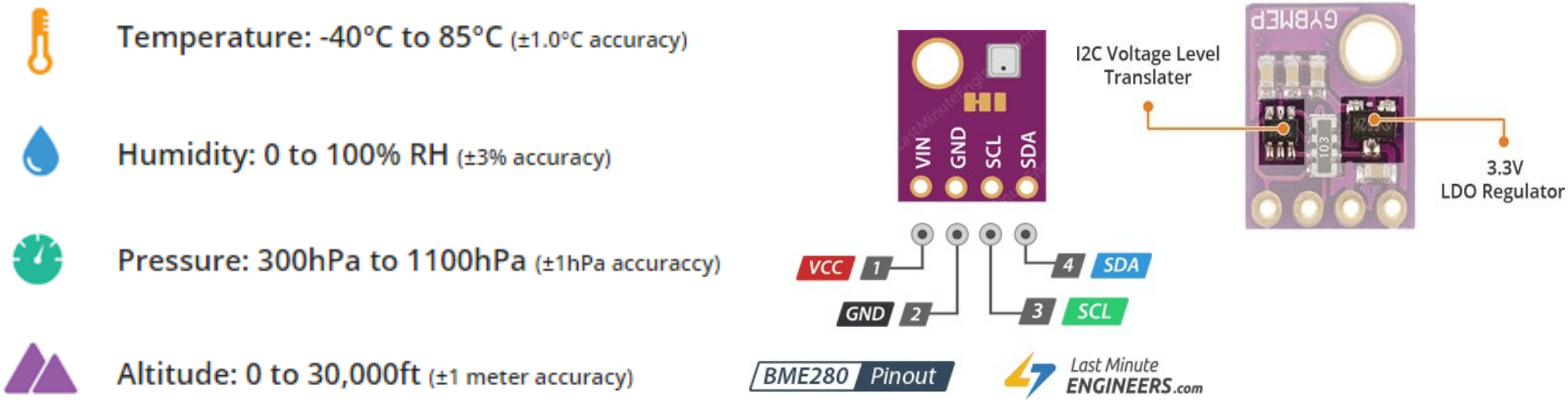
F446RE_DS3231RTC: futási eredmény

- A program futási eredménye az alábbi ábrán látható



A BME280/BMP280 I2C szenzor használata

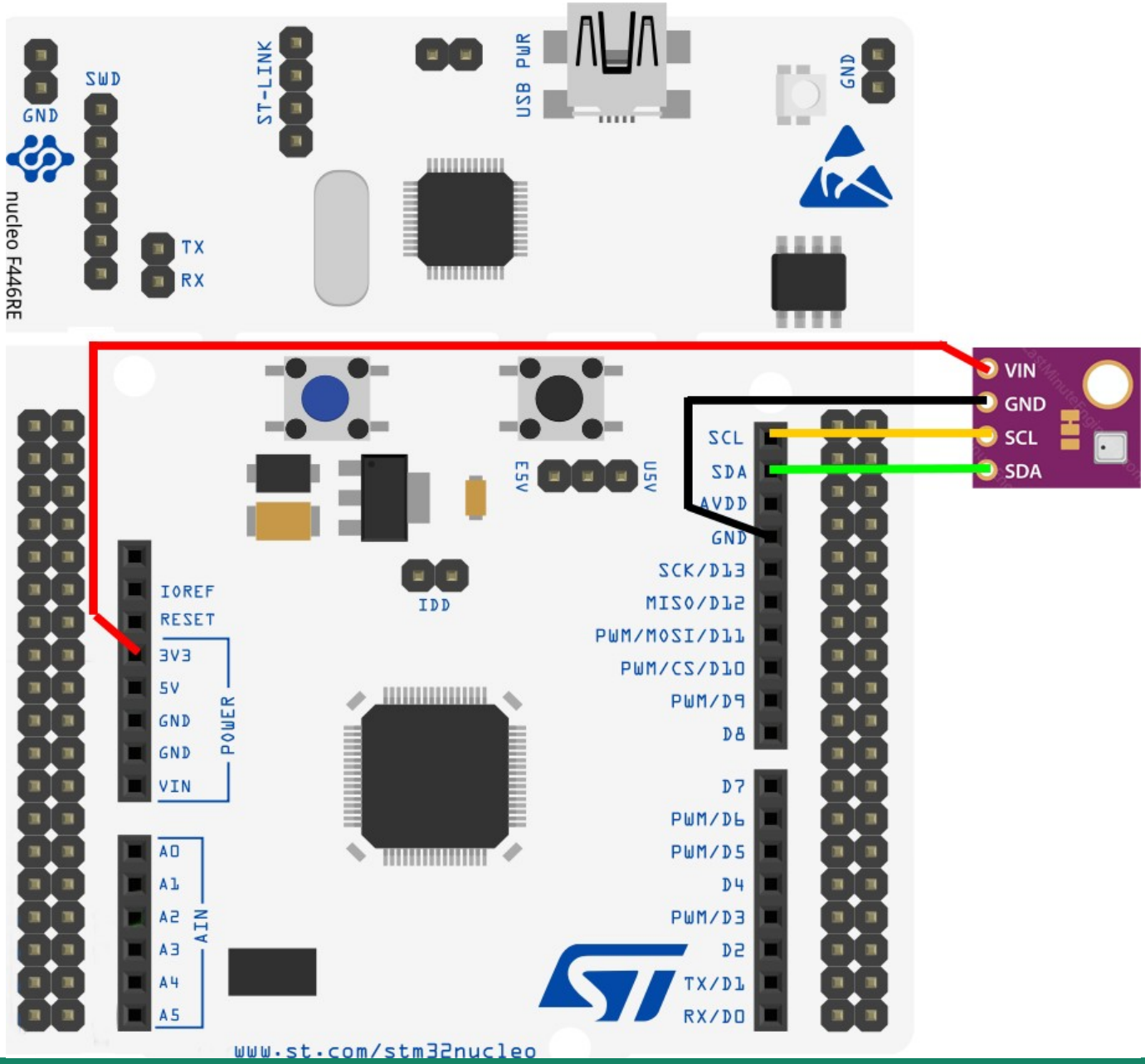
- A Bosch gyártmányú **BME280** szenzor nyomást, hőmérsékletet és relatív páratartalmat mér, a **BMP085, BMP180, BMP183** utódja
- A **BMP280** hasonló, de páratartalmat nem mér
- A szenzor **SPI** és **I2C** interfésszel rendelkezik, mi egy 5 V-tal és 3,3 V-tal egyaránt használható, **I2C** bekötésű modult használtunk
- Az **I2C** cím átforrasztással változtatható: **0x76** vagy **0x77**



Az ábrák forrása: lastminuteengineers.com/bme280-arduino_tutorial

A BME280/BMP280 I2C szenzor használata

- A **NUCLEO** kártyákon az Arduino kompatibilis **I2C** kivezetések a mikrovezérlő **PB8** (SCL) és **PB9** (SDA) lábaihoz csatlakoznak
- Az ábra szerinti bekötést használtuk



BME/BMP280 programkönyvtár

- Az Interneten számos programkönyvtár és mintaprogram található, mi Ciaskolog **BMP_STM32** (github.com/ciastkolog/BMP280_STM32) projektjét adaptáltuk. A legfontosabb API függvények:

- `bool bmp280_read_float(BMP280_HandleTypedef *dev, float *temperature, float *pressure, float *humidity) - a szenzor kompenzált adatainak lebegőpontos formátumú kiolvasása`
- `bool bmp280_read_fixed(BMP280_HandleTypedef *dev, int32_t *temperature, uint32_t *pressure, uint32_t *humidity) - a szenzor kompenzált adatainak lebegőpontos formátumú kiolvasása`
- `void bmp280_init_default_params(bmp280_params_t *params) - az alapértelmezett paraméterek beállítása (mode: NORMAL, filter: OFF, oversampling: x4, standby time: 250ms)`
- `bool bmp280_init(BMP280_HandleTypedef *dev, bmp280_params_t *params) - az eszköz inicializálása, kalibrációs konstansok kiolvasása, ID kiolvasása)`

F446RE_BME280: main.c (részlet 2/1.)

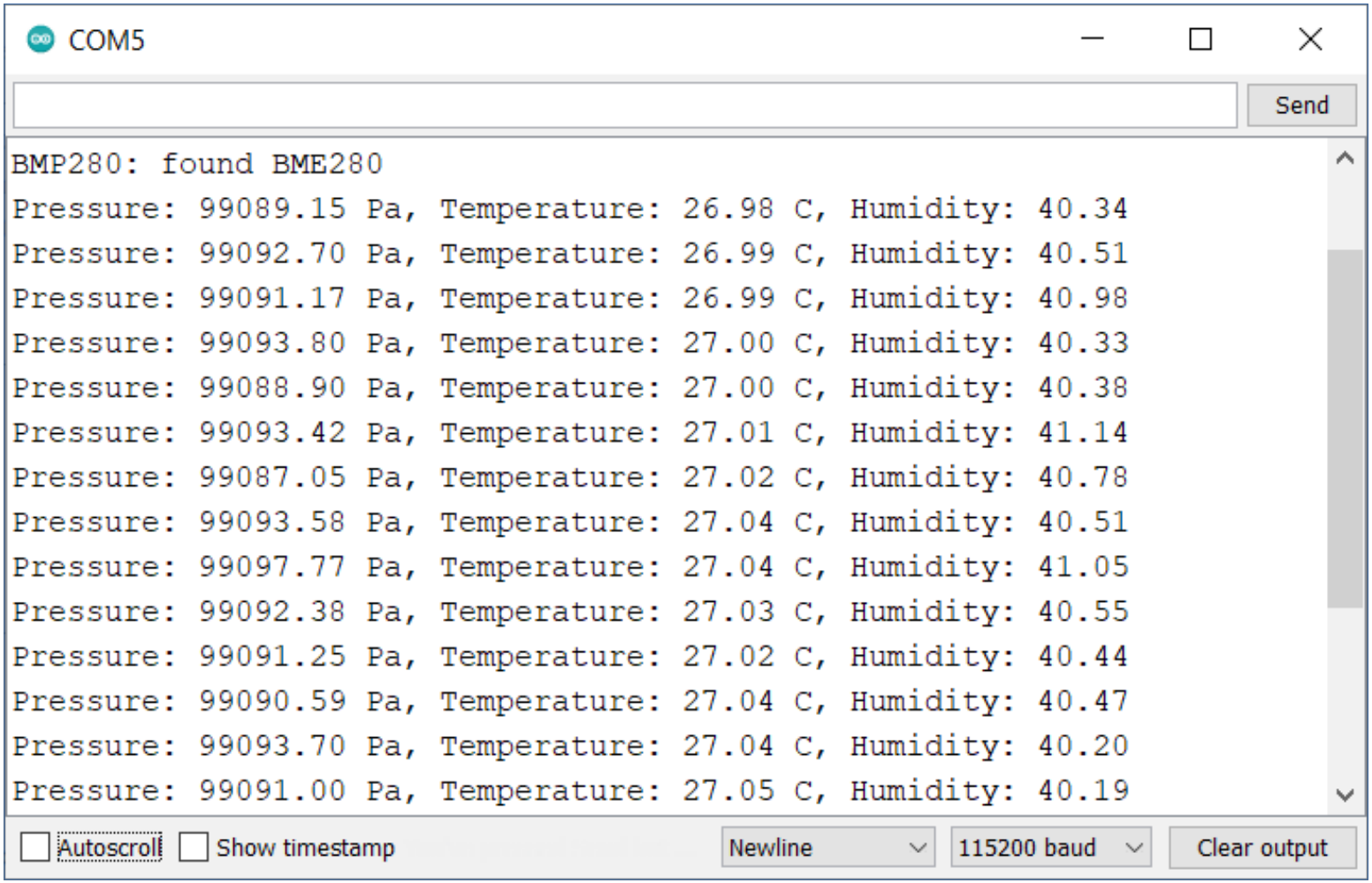
```c
#include "main.h"
#include "bmp280.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_USART2_UART_Init(void);

BMP280_HandleTypedef bmp280;
float pressure, temperature, humidity;
uint16_t size;
char msg[80];

int main(void) {
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_USART2_UART_Init();
  bmp280_init_default_params(&bmp280.params);
  bmp280.addr = BMP280_I2C_ADDRESS_0;              // 0x76
  bmp280.i2c = &hi2c1;
```

F446RE_BME280: main.c (részlet 2/2.)

```
  while (!bmp280_init(&bmp280, &bmp280.params)) {
    size = sprintf(msg, "BMP280 initialization failed\n"); // Hibajelzés, ha nem sikerült
    HAL_UART_Transmit(&huart2, (uint8_t*)msg, size, 1000);
    HAL_Delay(2000);
  }
  bool bme280p = bmp280.id == BME280_CHIP_ID;                // Milyen eszközt találtunk?
  size = sprintf(msg, "BMP280: found %s\n", bme280p ? "BME280" : "BMP280");
  HAL_UART_Transmit(&huart2, (uint8_t*)msg, size, 1000);
  while (1) {
    HAL_Delay(100);
    while (!bmp280_read_float(&bmp280, &temperature, &pressure, &humidity)) {
        size = sprintf(msg,"Temperature/pressure reading failed\n");
        HAL_UART_Transmit(&huart2,(uint8_t*)msg, size, 1000);
        HAL_Delay(2000);
    }
    size = sprintf(msg,"Pressure: %.2f Pa, Temperature: %.2f C", pressure, temperature);
    HAL_UART_Transmit(&huart2,(uint8_t*)msg, size, 1000);
    if (bme280p) {
        size = sprintf(msg,", Humidity: %.2f\n", humidity);
        HAL_UART_Transmit(&huart2,(uint8_t*)msg, size, 1000);
    }
    else { HAL_UART_Transmit(&huart2,(uint8_t *)"\n", 1, 1000); }
    HAL_Delay(2000);
  }
}
```

F446RE_BME280: futási eredmény

- A légnyomást még korrigálni kellene a tengerszintre, ami Debrecen esetén (kb. 120 m) leegyszerűsítve egy fix érték (1440 Pa) hozzáadásával megoldható. A mért érték tehát kb. **1005 hPa**



Helyi légnyomás átszámítása tengerszintre

- Tudnunk kell a helyi tengerszint feletti magasságot (altitude) és a szenzorral meg kell mérnünk az abszolút helyi nyomás értékét (p).
- A tengerszintre átszámított légnyomás (p_0) a következő képlettel számítható ki:

$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

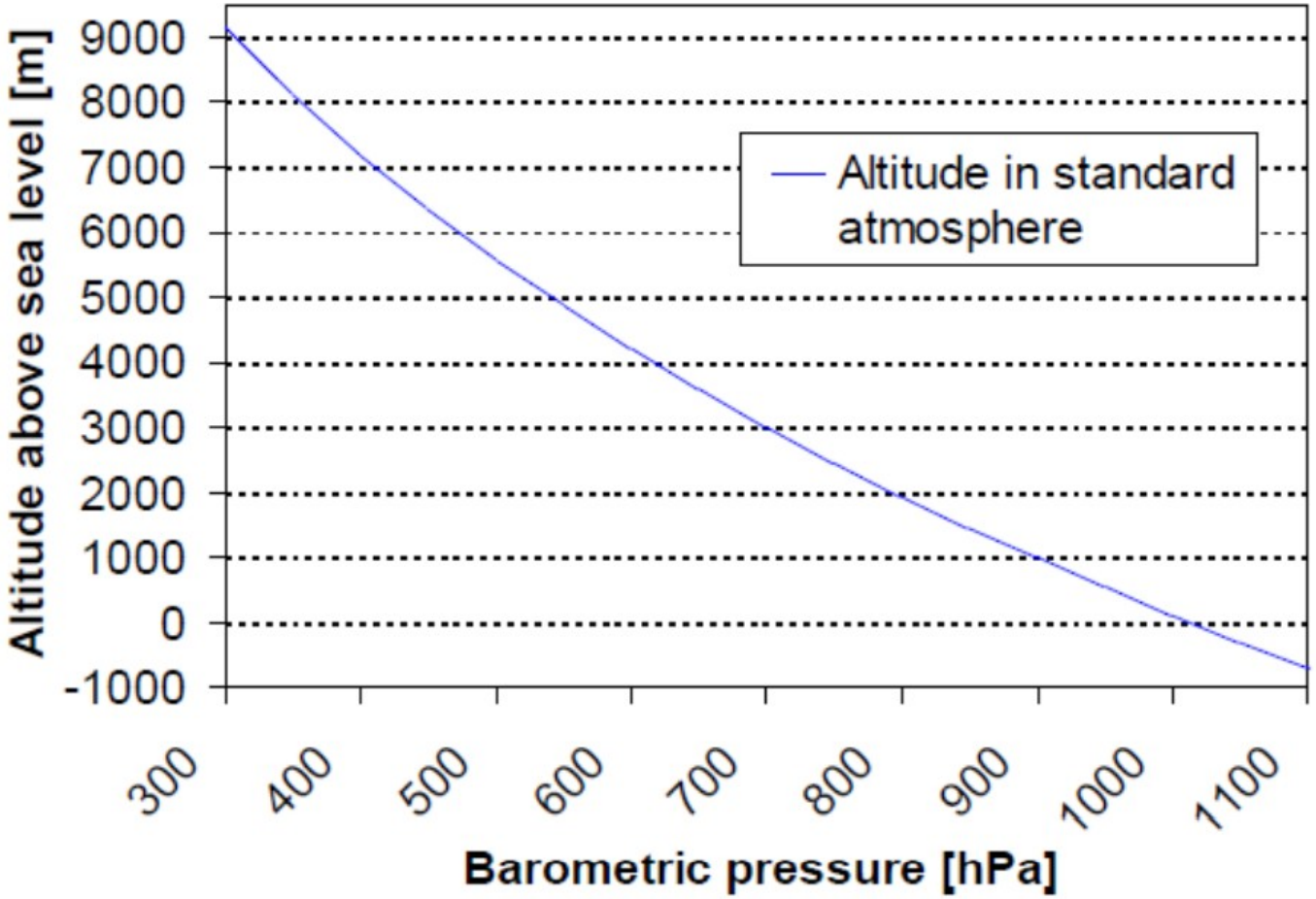
- **Egyszerű(bb) közelítés:**

  Debrecenben (kb. 120 m) 1440 Pa-t hozzáadunk a mért légnyomáshoz

Magasság meghatározása

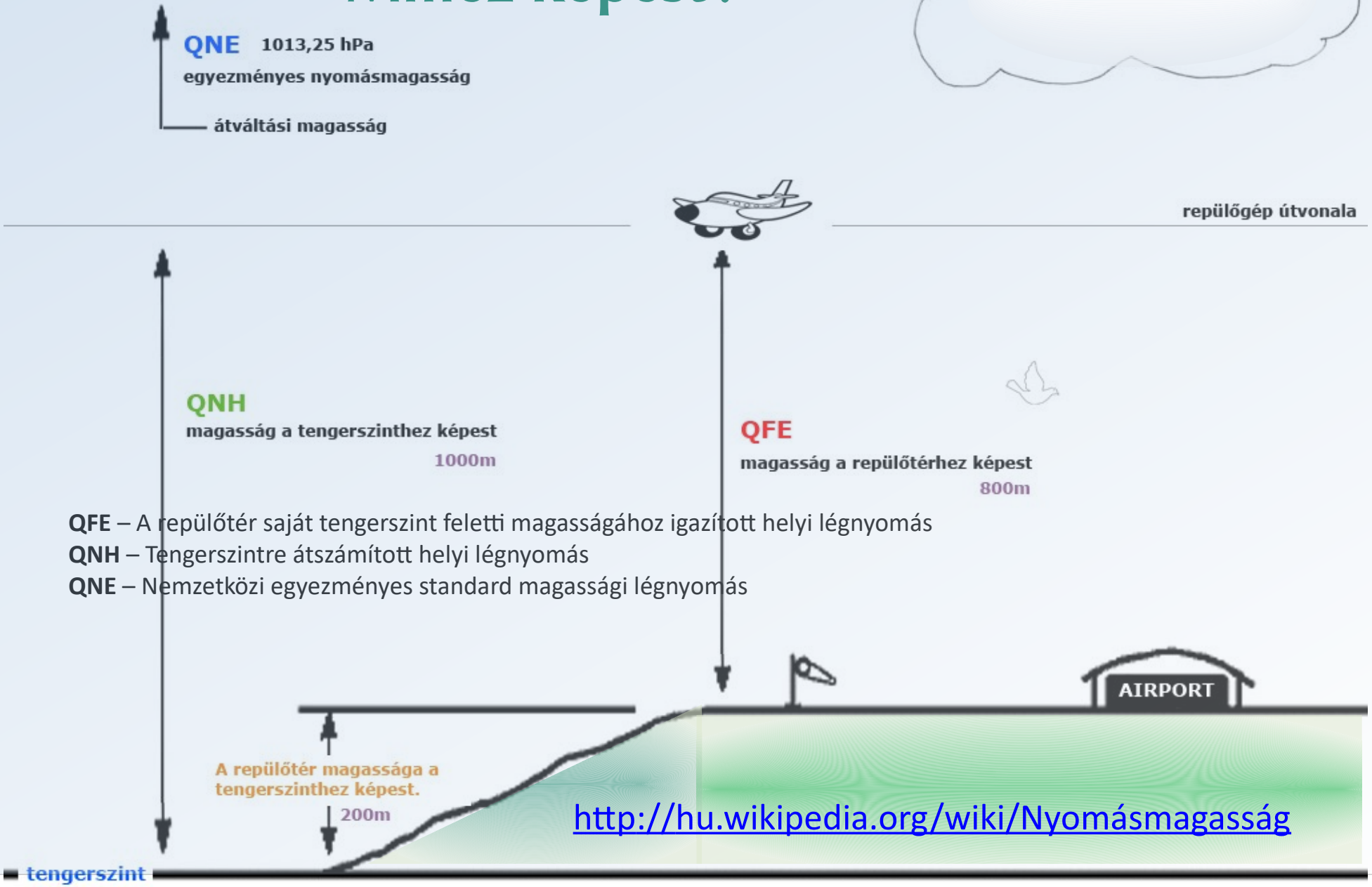
$$\text{magasság} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Ahol p az általunk mért nyomás, $\mathbf{p_0}$ pedig a tengerszinti nyomás



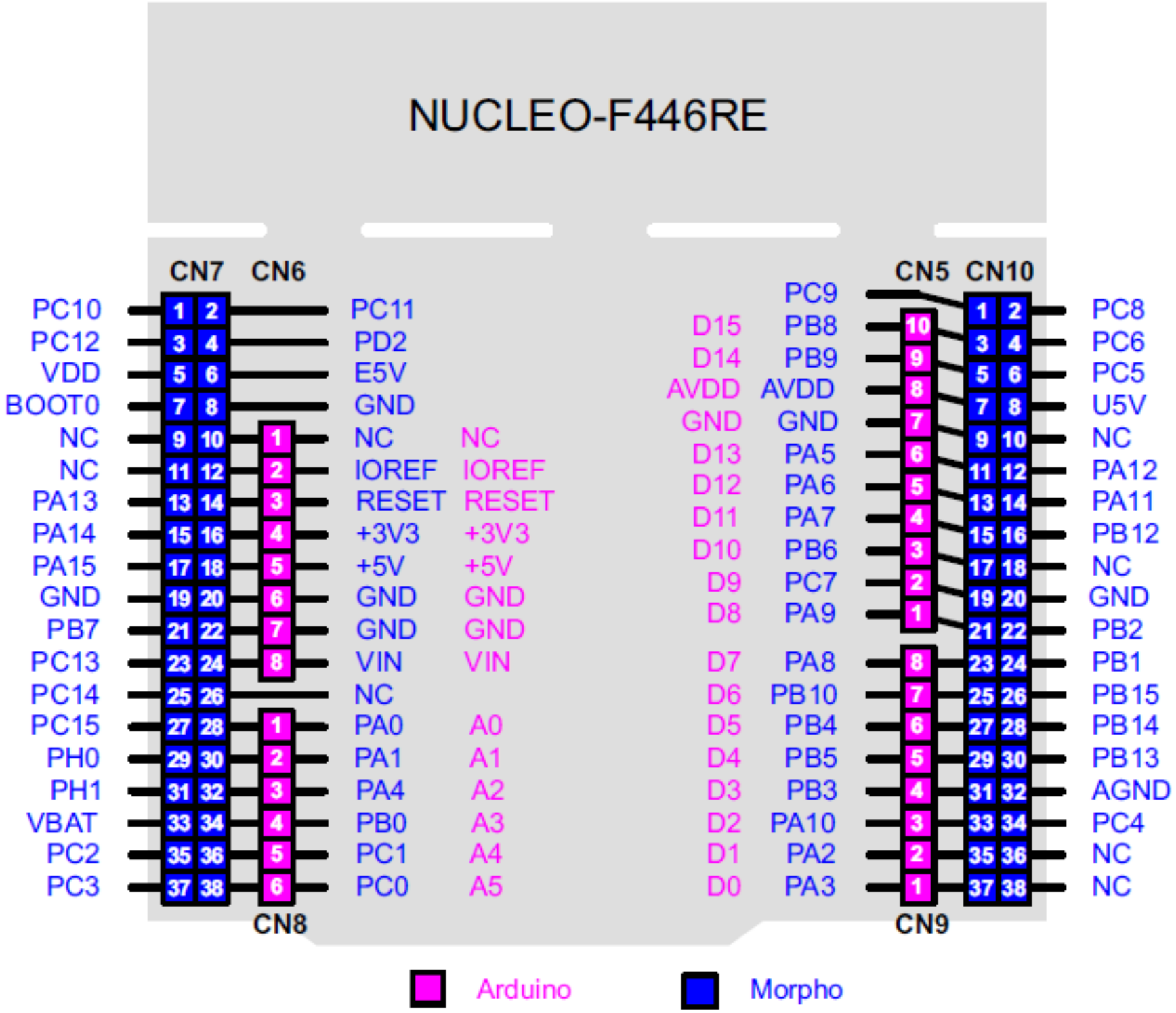
A gyakorlatban a nyomás nemcsak a magasságtól, hanem a meteorológiai viszonyoktól is függ (hőmérséklet, páratartalom, stb.)

- Milyen magasan repül a repülő?
- Mihez képest?



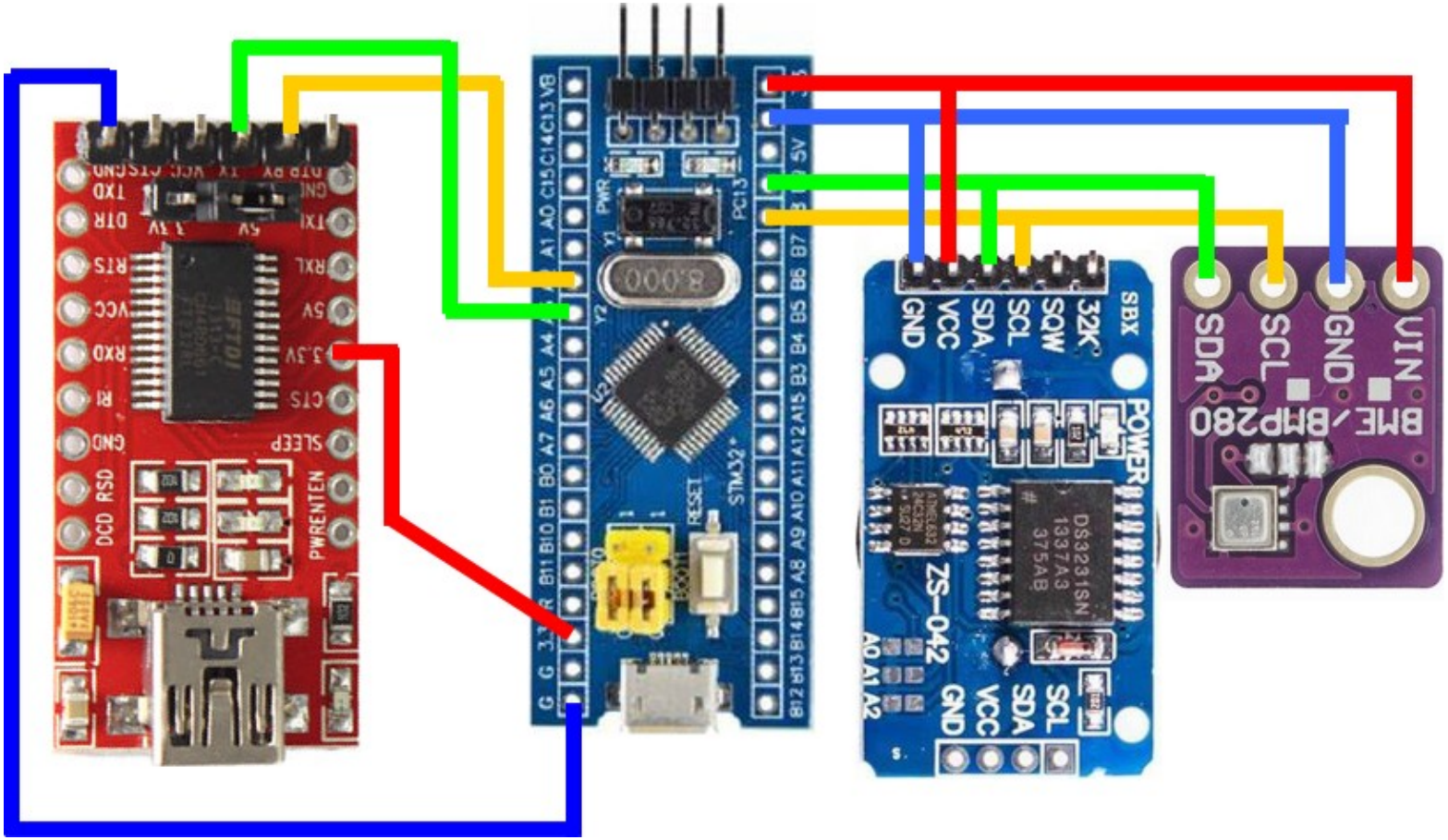
QFE – A repülőtér saját tengerszint feletti magasságához igazított helyi légnyomás
QNH – Tengerszintre átszámított helyi légnyomás
QNE – Nemzetközi egyezményes standard magassági légnyomás

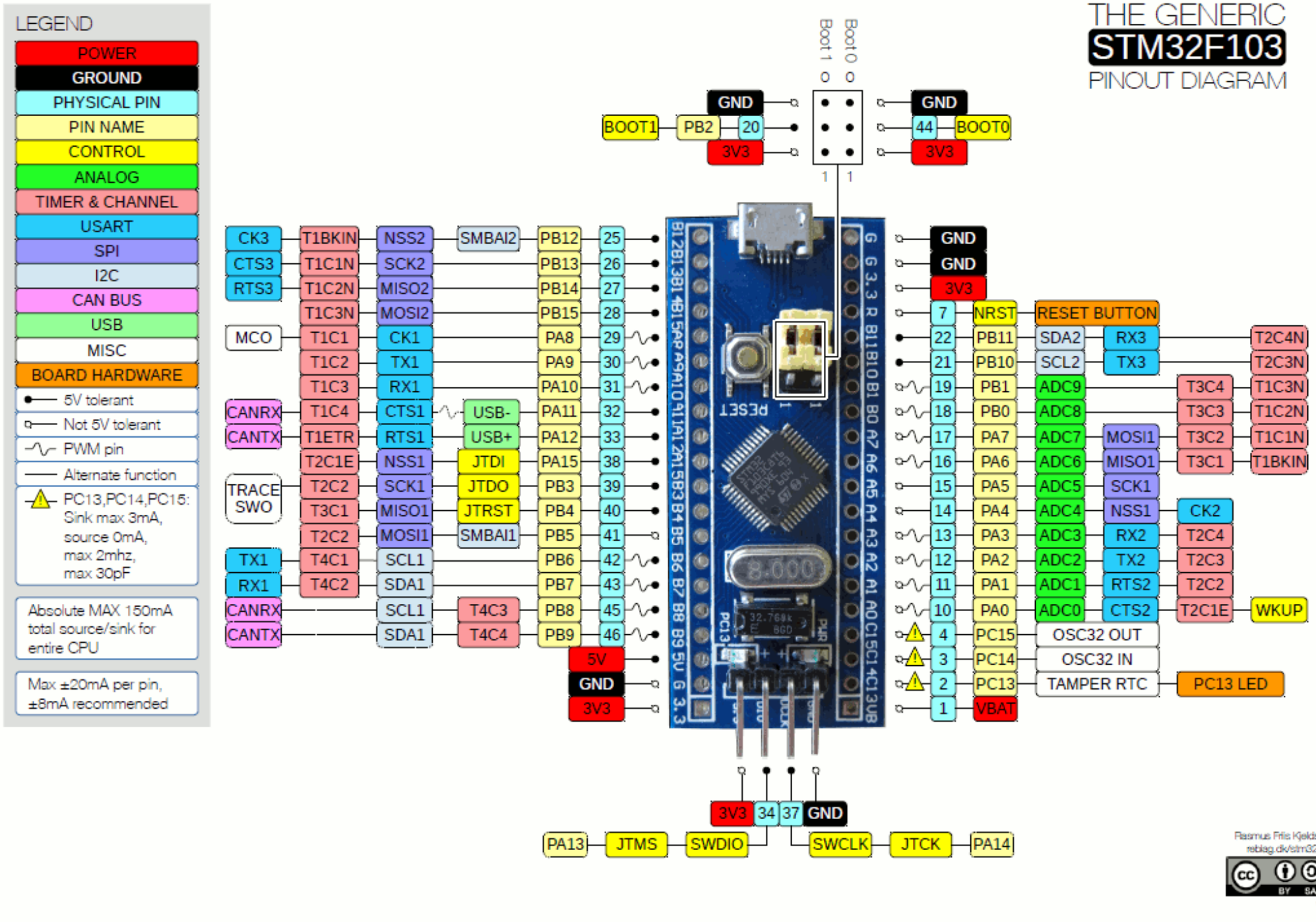
http://hu.wikipedia.org/wiki/Nyomásmagasság


NUCLEO-F446RE
CN7
CN6
PC10 1 2 PC11
PC12 3 4 PD2
VDD 5 6 E5V
BOOT0 7 8 GND
NC 9 10 NC
NC 11 12 IOREF
PA13 13 14 RESET
PA14 15 16 +3V3
PA15 17 18 +5V
GND 19 20 GND
PB7 21 22 GND
PC13 23 24 VIN
PC14 25 26 NC
PC15 27 28 PA0
PH0 29 30 PA1
PH1 31 32 PA4
VBAT 33 34 PB0
PC2 35 36 PC1
PC3 37 38 PC0
1 NC NC
2 IOREF IOREF
3 RESET RESET
4 +3V3 +3V3
5 +5V +5V
6 GND GND
7 GND GND
8 VIN VIN
1 A0
2 A1
3 A2
4 A3
5 A4
6 A5
CN8
CN5
CN10
PC9
D15 PB8
D14 PB9
AVDD AVDD
GND GND
D13 PA5
D12 PA6
D11 PA7
D10 PB6
D9 PC7
D8 PA9
D7 PA8
D6 PB10
D5 PB4
D4 PB5
D3 PB3
D2 PA10
D1 PA2
D0 PA3
10 9 8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
1 2 PC8
3 4 PC6
5 6 PC5
7 8 U5V
9 10 NC
11 12 PA12
13 14 PA11
15 16 PB12
17 18 NC
19 20 GND
21 22 PB2
23 24 PB1
25 26 PB15
27 28 PB14
29 30 PB13
31 32 AGND
33 34 PC4
35 36 NC
37 38 NC
CN9
Arduino
Morpho

Lab08 projektek STM32F103C8-ra

- Az **STM32F103C8** mikrovezérlőre (Blue pill) is adaptáltuk az előadás mintapéldáit, a felhasznált perifériák és kivezetések ugyanazok maradtak (**I2C1 – PB8, PB9, USART2 – PA2, PA3**)




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
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
PC13 LED
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Fris Kjeldsen
reblag.dk/stm32
V1.0