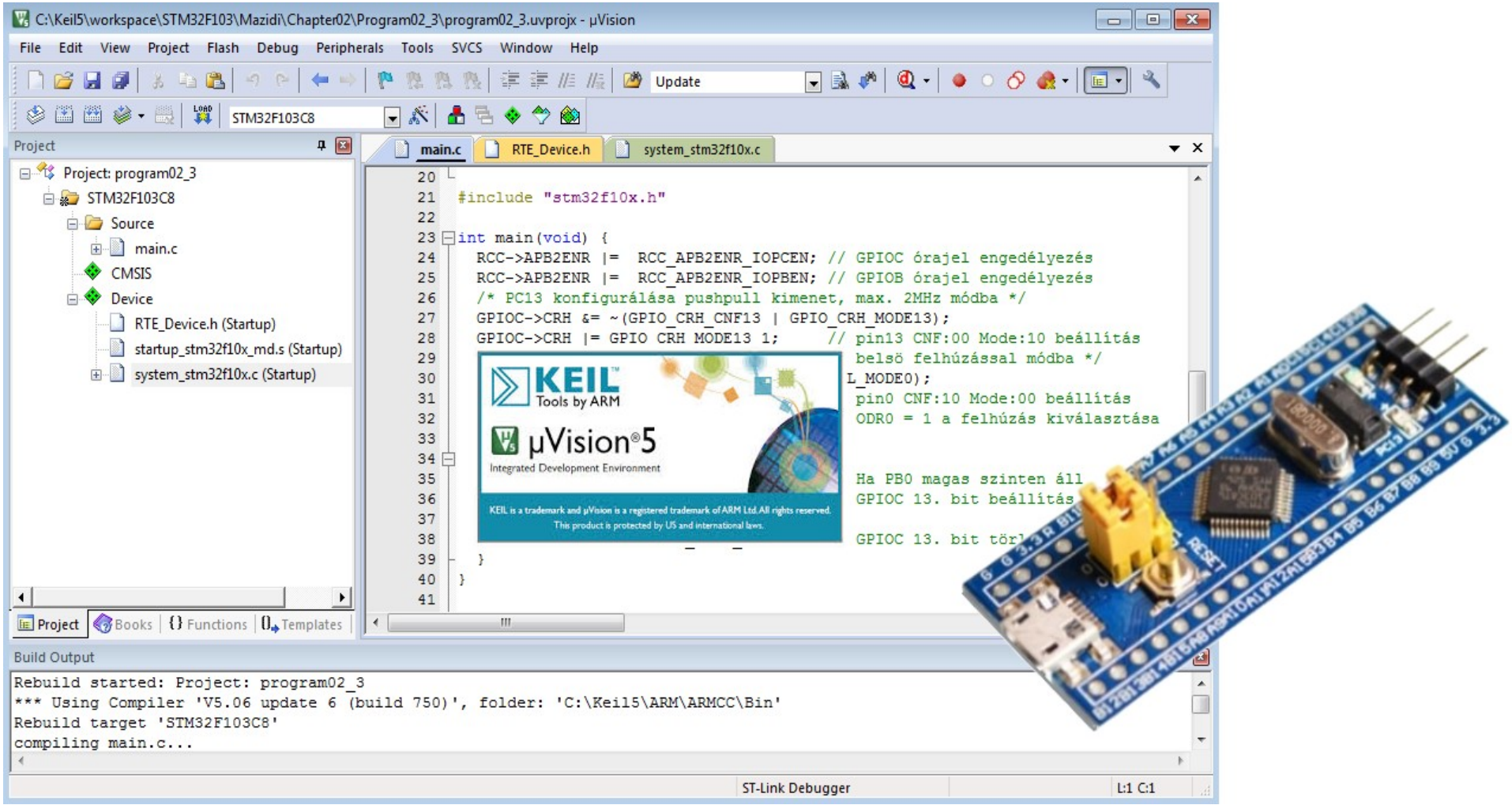
STM32 mikrovezérlők programozása ARM Keil környezetben



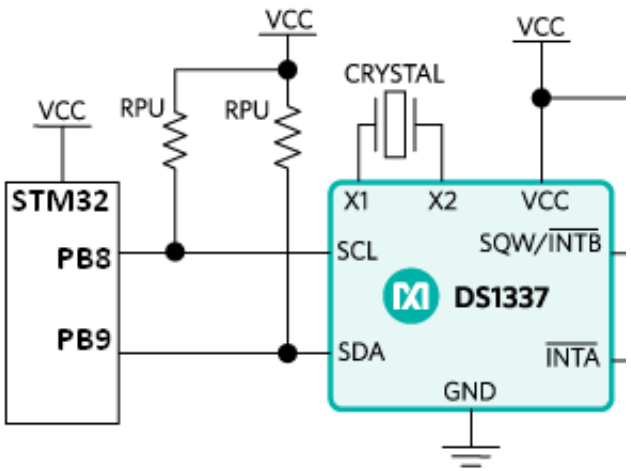
10. Az I2C kommunikációs csatorna – 2. rész

Felhasznált és ajánlott irodalom

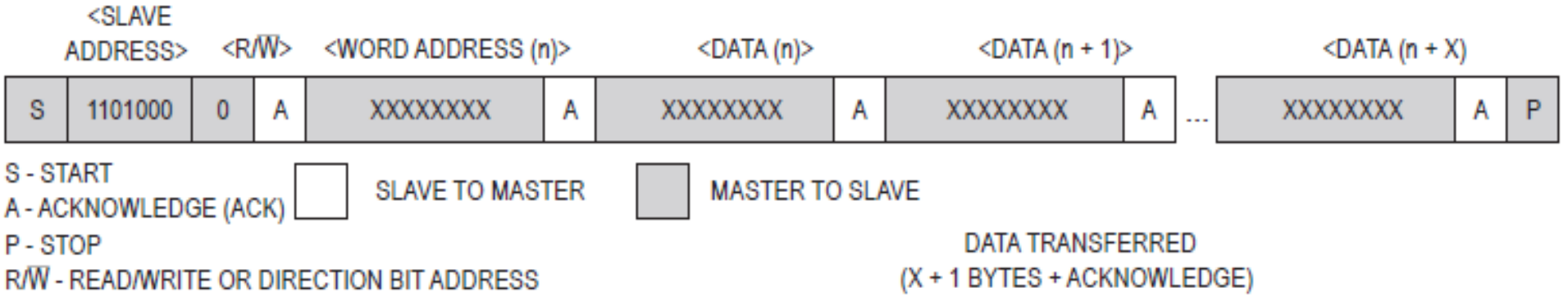
- Joseph Yiu: Cortex-M for Beginners
- Joseph Yiu: The Definitive Guide To The ARM CORTEX-M3
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: STM32 Arm Programming for Embedded Systems
- Alexander Tarasov: **Курс «Штурмуем STM32»**
- Warren Gay: Beginning STM32 - Developing with FreeRTOS, libopencm3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

Csoportos adatküldés

- Csoportos adatküldés: több adatbájtot küldünk egymás után, a fogadó *slave* eszköz pedig automatikusan lépteti a címet
- A *slave* eszköz dönti el, hogy hány adatot fogad egyszerre (pl. lapokba szervezett memória esetén a laphatár nem léphető át)



- Az előző előadásban is használt **DS3231** RTC esetén a *slave* címet követően egy egybájtos regisztercímet kell küldeni, majd következhetnek a regiszterekbe írandó adatok.
- Ilyen eset például az idő és a dátum beírása



DS3231 regisztertérkép

2020. jan. 28.
11:56:55 kedd

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	0x55
01h	0	10 Minutes			Minutes				Minutes	00–59	0x56
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23	0x11
03h	0	0	0	0	0	Day			Day	1–7	0x02
04h	0	0	10 Date		Date				Date	01–31	0x28
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century	0x01
06h	10 Year				Year				Year	00–99	0x20
07h	A1M1	10 Seconds			Seconds				Alarm 1 Seconds	00–59	0
08h	A1M2	10 Minutes			Minutes				Alarm 1 Minutes	00–59	0
09h	A1M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 1 Hours	1–12 + $\overline{AM}$/PM 00–23	0
0Ah	A1M4	DY/$\overline{DT}$	10 Date		Day				Alarm 1 Day	1–7	0
					Date				Alarm 1 Date	1–31	0
0Bh	A2M2	10 Minutes			Minutes				Alarm 2 Minutes	00–59	0
0Ch	A2M3	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Alarm 2 Hours	1–12 + $\overline{AM}$/PM 00–23	0
0Dh	A2M4	DY/$\overline{DT}$	10 Date		Day				Alarm 2 Day	1–7	0
					Date				Alarm 2 Date	1–31	0
0Eh	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE	Control	—	0
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	—	

Program09_3/main.c

```
#include "stm32f10x.h"
#define SLAVE_ADDR 0x68                          /* 1101 000.    DS3231 */
void I2C1_init(void);
void I2C1_burstWrite(char saddr, char maddr, int size, char* data);

int main(void) {                                // 2020.01.28 11:56:55
  char timeDateToSet[15] = {0x55, 0x56, 0x11, 0x02, 0x28, 0x01, 0x20};
  I2C1_init();
  I2C1_burstWrite(SLAVE_ADDR, 0, 7, timeDateToSet);
  while(1);
}

void I2C1_init(void) {
  RCC->APB1ENR |=  RCC_APB1ENR_I2C1EN;          // I2C1 engedélyezése
  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;           // AFIO engedélyezése
  RCC->APB2ENR |=  RCC_APB2ENR_IOPBEN;          // GPIOB engedélyezése
  AFIO->MAPR |= AFIO_MAPR_I2C1_REMAP;           // I2C1 REMAP=1 (SCL/PB8, SDA/PB9)
  GPIOB->CRH |= GPIO_CRH_CNF8|GPIO_CRH_MODE8; // PB8 ALT/OD kimenet (SCL)
  GPIOB->CRH |= GPIO_CRH_CNF9|GPIO_CRH_MODE9; // PB9 ALT/OD kimenet (SDA)
  I2C1->CR1 =   I2C_CR1_SWRST;                  // Szoftver RESET
  I2C1->CR1 &= ~I2C_CR1_SWRST;                  // RESET vége
  I2C1->CR2 = 36;                               // PCLK1 = 36 MHz
  I2C1->CCR = 180;                              // Standard mód 100 kHz
  I2C1->TRISE = 37;                             // maximum felfutási idő + 1
  I2C1->CR1 |=  I2C_CR1_PE;                     // I2C1 modul indítása
}
```

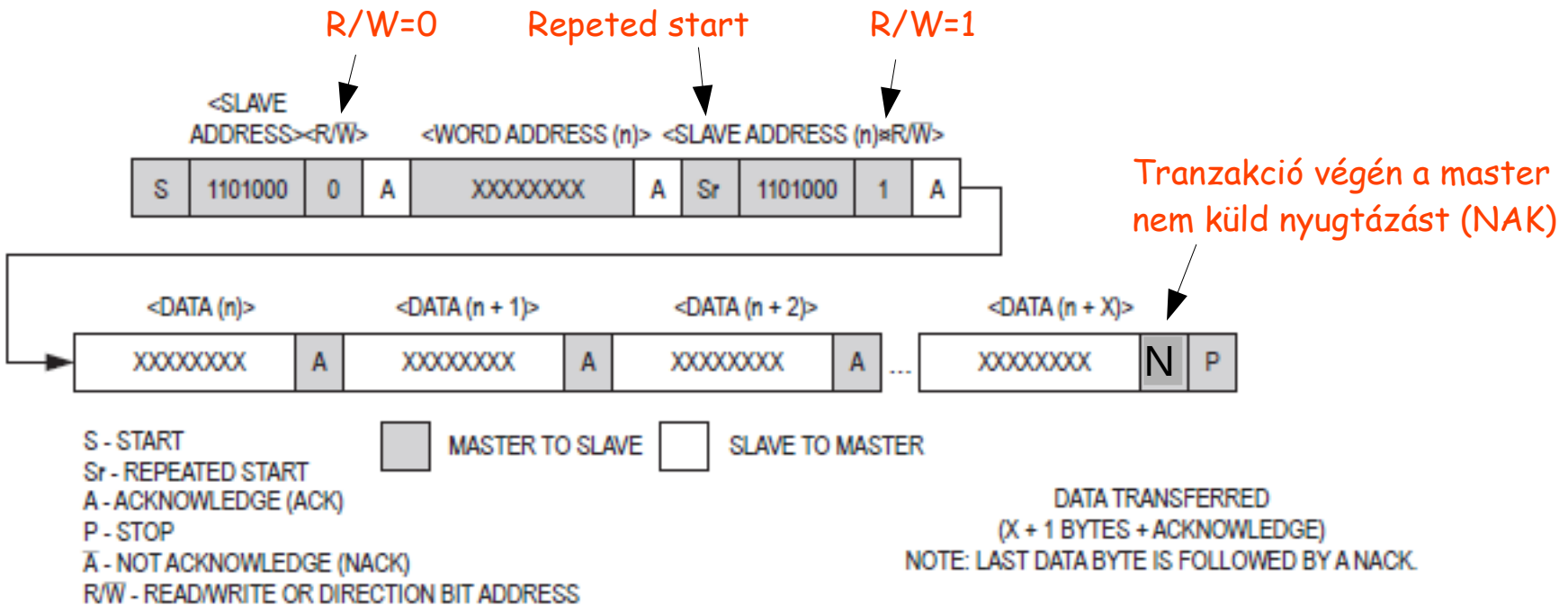
Program09_3/main.c (folytatás)

```
void I2C1_burstWrite(char saddr, char maddr, int size, char* data)  {
    volatile int i, tmp;
    while (I2C1->SR2 & 2);                          /* Vár, amíg a busz foglalt */
    I2C1->CR1 |=  0x100;                            /* START generálás */
    while (!(I2C1->SR1 & 1));                       /* Vár, amíg SR1.SB = 0 */
    I2C1->DR = saddr<<1;                            /* Slave cím kiküldése */
    while (!(I2C1->SR1 & 2));                       /* Vár, amíg SR1.ADDR = 0 */
    tmp = I2C1->SR2;                                /* SR1.ADDR törlése */
    while (!(I2C1->SR1 & 0x80));                    /* Vár, amíg SR1.TXE = 0 */
    I2C1->DR = maddr;                               /* regisztercím küldése */
    for(i = 0; i < size; i++) {
        while (!(I2C1->SR1 & 0x80));                /* Vár, amíg SR1.TXE = 0 */
        I2C1->DR = data[i];                         /* Adatbájt küldése */
    }
    while (!(I2C1->SR1 & 4));                       /* Vár, amíg SR1.BTF = 0  */
    I2C1->CR1 |= 0x200;                             /* STOP generálás */
}
```

- A program lefutásának nincs látható hatása, csupán az RTC-be beíródnak a megadott dátum és idő adatok
- Elem vagy akkumulátoros $\mathbf{V_{BAT}}$ táplálás esetén az RTC megőrzi és lépteti az időt és a dátumot

Csoportos adatlekérés az I2C buszon

- Csoportos adatfogadás: több adatbájtot fogadunk egymás után, a küldő *slave* eszköz pedig automatikusan lépteti a címet
- A korábbi **DS3231** RTC-nél maradva csoportos adatlekérésnél a ió írás (regisztercím) 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



Program09_4/main.c

```c
#include "stm32f10x.h"
#include <stdio.h>
#pragma import(__use_no_semihosting_swi)
struct __FILE { int handle; };
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE *f) { ITM_SendChar(ch); return (ch); }
void _sys_exit(int return_code) { label: goto label; }
#define SLAVE_ADDR 0x68           // 1101 000.    DS3231
void I2C1_init(void);
void I2C1_burstRead(char saddr, char maddr, int n, char* data);
void displayTime(char* data);

int main(void) {
    char timeDateReadback[15];  // Ide töltjük a fogadott adatokat
    char lasttime = 0;
    I2C1_init();

    while (1) {
        I2C1_burstRead(SLAVE_ADDR,0,7,timeDateReadback); // Adatbeolvasás
        if (timeDateReadback[0] != lasttime) {           // Ha eltelt egy másodperc
          displayTime(timeDateReadback);                 // Kiírjuk az időt
          lasttime = timeDateReadback[0];
        }
  }
}
```

printf() átirányítása az SWO kimenetre (debug módban)

Program09_4/main.c (folytatás)

```
#define dec1(x)     (x &0x0f)+48
#define dec10(x)    ((x>>4)&0x03)+48

void displayTime(char* data) {
  char ts[9]; ts[8]=0;
  ts[7]=dec1(data[0]); ts[6]=dec10(data[0]); ts[5]=':';
  ts[4]=dec1(data[1]); ts[3]=dec10(data[1]); ts[2]=':';
  ts[1]=dec1(data[2]); ts[0]=dec10(data[2]);
  printf("%s\n",ts);                              // HH:MM:SS formátumú kiírás
}

void I2C1_init(void) {
  RCC->APB1ENR |=  RCC_APB1ENR_I2C1EN;          // I2C1 engedélyezése
  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;           // AFIO engedélyezése
  RCC->APB2ENR |=  RCC_APB2ENR_IOPBEN;          // GPIOB engedélyezése
  AFIO->MAPR |= AFIO_MAPR_I2C1_REMAP;           // I2C1 REMAP=1 (SCL/PB8, SDA/PB9)
  GPIOB->CRH |= GPIO_CRH_CNF8|GPIO_CRH_MODE8; // PB8 ALT/OD kimenet (SCL)
  GPIOB->CRH |= GPIO_CRH_CNF9|GPIO_CRH_MODE9; // PB9 ALT/OD kimenet (SDA)
  I2C1->CR1 =   I2C_CR1_SWRST;                  // Szoftver RESET
  I2C1->CR1 &= ~I2C_CR1_SWRST;                  // RESET vége
  I2C1->CR2 = 36;                               // PCLK1 = 36 MHz
  I2C1->CCR = 180;                              // Standard mód 100 kHz
  I2C1->TRISE = 37;                             // maximum felfutási idő
  I2C1->CR1 |=  I2C_CR1_PE;                     // I2C1 modul indítása
}
```

Program09_4/main.c (folytatás)

- Az eszköz- és a regisztercím kiküldése, az *Ismételt Start* , majd a *slave* cím újbóli kiküldése **Program09_2**-höz hasonlóan történik

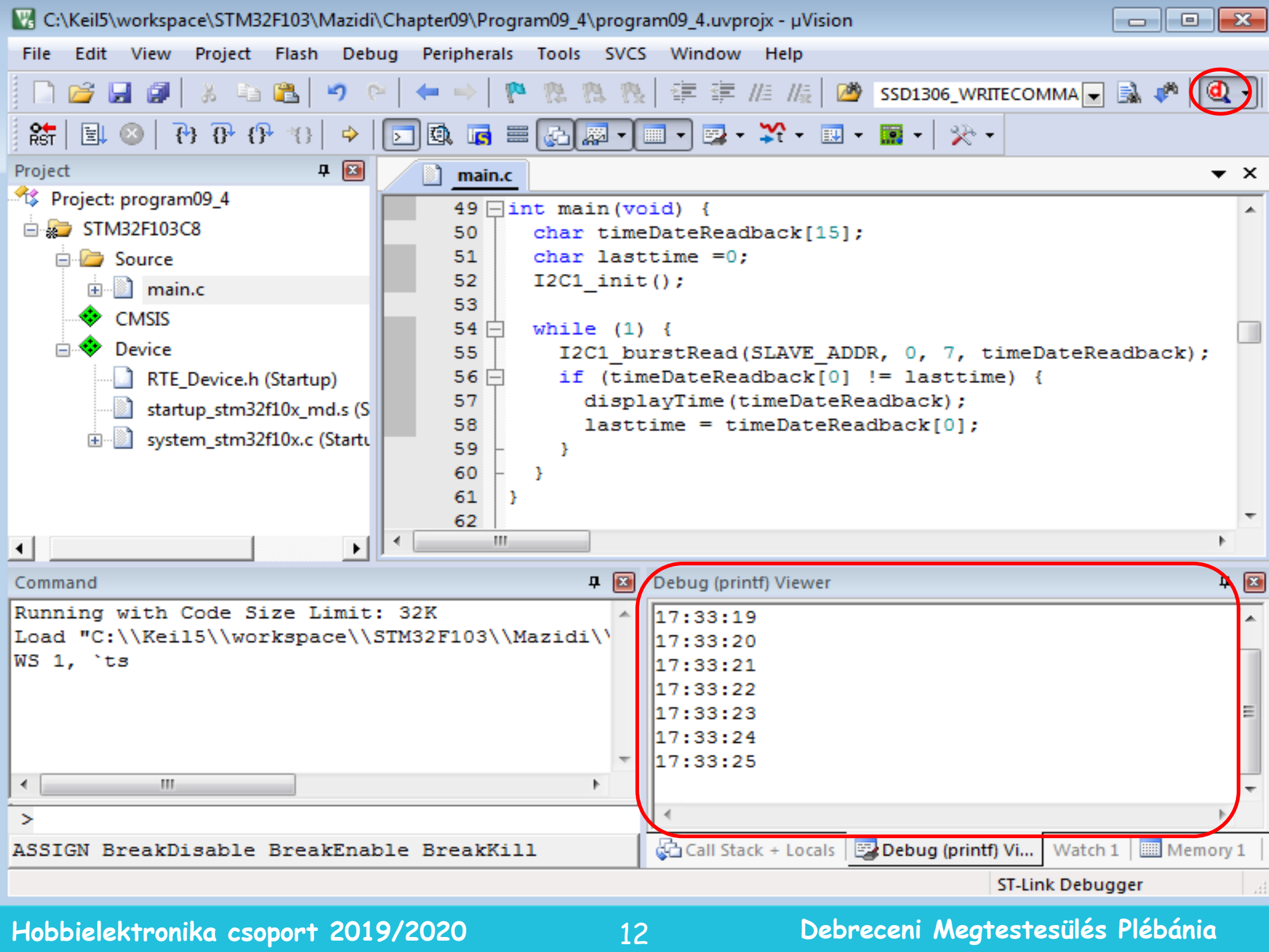
```
void I2C1_burstRead(char saddr, char maddr, int size, char* data)  {
  volatile int tmp;

  while (I2C1->SR2 & 2);                    /* wait until bus not busy */
  I2C1->CR1 &= ~0x800;                      /* disable POS */
//--- Start, majd Slave cím és regisztercím kiküldése R/W = 0 ------
  I2C1->CR1 |= 0x100;                       /* generate start */
  while (!(I2C1->SR1 & 1));                 /* wait until start flag is set */
  I2C1->DR = saddr << 1;                    /* transmit slave address + Write */
  while (!(I2C1->SR1 & 2));                 /* wait until addr flag is set */
  tmp = I2C1->SR2;                          /* clear addr flag */
  while (!(I2C1->SR1 & 0x80));              /* wait until transmitter empty */
  I2C1->DR = maddr;                         /* send memory address */
  while (!(I2C1->SR1 & 0x80));              /* wait until transmitter empty */
//--- Restart, Slave cím kiküldése R/W = 1, majd olvasás indul ------
  I2C1->CR1 |= 0x100;                       /* generate restart */
  while (!(I2C1->SR1 & 1));                 /* wait until start flag is set */
  I2C1->DR = saddr << 1 | 1;                /* transmit slave address + Read */
  while (!(I2C1->SR1 & 2));                 /* wait until addr flag is set */
  tmp = I2C1->SR2;                          /* clear addr flag */
  I2C1->CR1 |=  0x0400;                     /* Enable Acknowledge */
```

Program09_4/main.c (folytatás)

- Több-bájtos beolvasásnál meg kell különböztetni azt az esetet, amikor az utolsó bájt olvasása következik
- Az utolsó bájt fogadása előtt le kell tiltani a nyugtázás (**ACK**) küldését és elő kell jegyezni a **Stop** feltételt

```
    while(size > 0U) {
        /* One byte left */
        if(size == 1U) {
            I2C1->CR1 &= ~(0x400);          /* Disable Acknowledge */
            I2C1->CR1 |= 0x200;             /* Generate Stop */
            while (!(I2C1->SR1 & 0x40));    /* Wait for RXNE flag set */
            *data++ = I2C1->DR;             /* Read data from DR */
            break;
        }
        else {
            while (!(I2C1->SR1 & 0x40));    /* Wait until RXNE flag is set */
            (*data++) = I2C1->DR;           /* Read data from DR */
            size--;
        }
    }
}
```


C:\Keil5\workspace\STM32F103\Mazidi\Chapter09\Program09_4\program09_4.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
SSD1306_WRITECOMMA
Project
Project: program09_4
STM32F103C8
Source
main.c
CMSIS
Device
RTE_Device.h (Startup)
startup_stm32f10x_md.s (S
system_stm32f10x.c (Startu
main.c
49 int main(void) {
50 char timeDateReadback[15];
51 char lasttime =0;
52 I2C1_init();
53
54 while (1) {
55 I2C1_burstRead(SLAVE_ADDR, 0, 7, timeDateReadback);
56 if (timeDateReadback[0] != lasttime) {
57 displayTime(timeDateReadback);
58 lasttime = timeDateReadback[0];
59 }
60 }
61 }
62
Command
Running with Code Size Limit: 32K
Load "C:\\Keil5\\workspace\\STM32F103\\Mazidi\\
WS 1, `ts
>
ASSIGN BreakDisable BreakEnable BreakKill
Debug (printf) Viewer
17:33:19
17:33:20
17:33:21
17:33:22
17:33:23
17:33:24
17:33:25
Call Stack + Locals
Debug (printf) Vi...
Watch 1
Memory 1
ST-Link Debugger

I2C Fast mode (400 kHz)

- **UM10 204:** I2C-bus specification and user manual (NXP)

Symbol	Parameter	Conditions	Standard-mode		Fast-mode		Fast-mode Plus		Unit
			Min	Max	Min	Max	Min	Max	
f_{SCL}	SCL clock frequency		0	100	0	400	0	1000	kHz
$t_{HD;STA}$	hold time (repeated) START condition	After this period, the first clock pulse is generated.	4.0	-	0.6	-	0.26	-	µs
t_{LOW}	LOW period of the SCL clock		4.7	-	1.3	-	0.5	-	µs
t_{HIGH}	HIGH period of the SCL clock		4.0	-	0.6	-	0.26	-	µs
$t_{SU;STA}$	set-up time for a repeated START condition		4.7	-	0.6	-	0.26	-	µs
$t_{HD;DAT}$	data hold time[2]	CBUS compatible masters (see Remark in Section 4.1)	5.0	-	-	-	-	-	µs
		I²C-bus devices	0[3]	-[4]	0[3]	-[4]	0	-	µs
$t_{SU;DAT}$	data set-up time		250	-	100[5]	-	50	-	ns
t_r	rise time of both SDA and SCL signals		-	1000	20	300	-	120	ns
t_f	fall time of both SDA and SCL signals[3][6][7][8]		-	300	20 × (V_{DD} / 5.5 V)	300	20 × (V_{DD} / 5.5 V)[9]	120[8]	ns
$t_{SU;STO}$	set-up time for STOP condition		4.0	-	0.6	-	0.26	-	µs
t_{BUF}	bus free time between a STOP and START condition		4.7	-	1.3	-	0.5	-	µs
C_b	capacitive load for each bus line[10]		-	400	-	400	-	550	pF
$t_{VD;DAT}$	data valid time[11]		-	3.45[4]	-	0.9[4]	-	0.45[4]	µs
$t_{VD;ACK}$	data valid acknowledge time[12]		-	3.45[4]	-	0.9[4]	-	0.45[4]	µs
V_{nL}	noise margin at the LOW level	for each connected device (including hysteresis)	0.1V_{DD}	-	0.1V_{DD}	-	0.1V_{DD}	-	V
V_{nH}	noise margin at the HIGH level	for each connected device (including hysteresis)	0.2V_{DD}	-	0.2V_{DD}	-	0.2V_{DD}	-	V

Időzítést vezérlő regiszterek

- **I2C*n*_CCR** – clock control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
F/S	DUTY	Reserved		CCR[11:0]											
rw	rw			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

F/S – üzemmód váltó (**0:** standard, **1:** fast mód)
DUTY – t_{LOW}/t_{HIGH} arány Fast módban (**0:** 2, **1:** 16/9)
CCR – **SCL** órajel frekvencia megadása
Standard mód: $t_{LOW} = t_{HIGH} = CCR * T_{PCLK1}$)
Fast mód: $t_{HIGH} = CCR * T_{PCLK1}$ és $t_{LOW} = 2* CCR * T_{PCLK1}$ ha **DUTY** = 0,
illetve $t_{HIGH} = 9 * CCR * T_{PCLK1}$ és $t_{LOW} = 16 * CCR * T_{PCLK1}$ ha **DUTY** = 1

$CCR = f_{PCLK1}/f_{I2C}/3$

$CCR = f_{PCLK1}/f_{I2C}/25$

- **I2C*n*_TRISE** regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										TRISE[5:0]					
										rw	rw	rw	rw	rw	rw

TRISE – maximális felfutási idő + 1 (csak master módhoz kell)

Például **Fast** módban **SCL** max. felfutási ideje 300 ns. Ha pl PCLK1 = 36 MHz, azaz T_{PCLK1} = 27.8 ns, akkor TRISE = 300 ns / 27.8 ns + 1 = 10(.8) + 1 Lefelé kerekítünk!

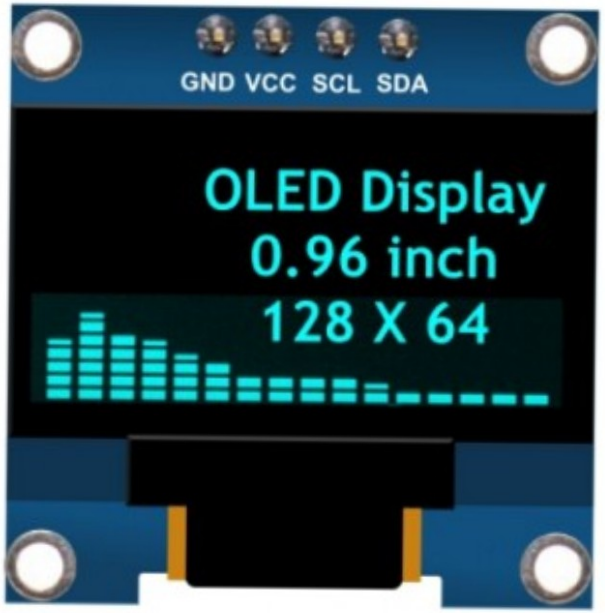
I2C1 inicializálása Fast módba

- A **Fast mód** inicializálása abban különbözik a **Standard módtól,** hogy újra kell számolni a **CCR** és **TRISE** regiszterekbe írandó számokat
- Itt F/S=1, DUTY=0, CCR = $f_{PCLK1}/f_{I2C}/3 = 36x10^6/4x1o^5/3 = 30$
  TRISE = 300 ns * 36 MHz +1 = $0.3x1o^{-6} * 36x1o^6 + 1 = 11$

```
void I2C1_init() {
    RCC->APB1ENR |=  RCC_APB1ENR_I2C1EN;          // I2C1 engedélyezése
    RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;           // AFIO engedélyezése
    RCC->APB2ENR |=  RCC_APB2ENR_IOPBEN;          // GPIOB engedélyezése
    AFIO->MAPR |= AFIO_MAPR_I2C1_REMAP;           // I2C1 REMAP=1 (SCL/PB8, SDA/PB9)
    GPIOB->CRH |= GPIO_CRH_CNF8|GPIO_CRH_MODE8; // PB8 ALT/OD kimenet (SCL)
    GPIOB->CRH |= GPIO_CRH_CNF9|GPIO_CRH_MODE9; // PB9 ALT/OD kimenet (SDA)
    I2C1->CR1 =   I2C_CR1_SWRST;                  // Szoftver RESET
    I2C1->CR1 &= ~I2C_CR1_SWRST;                  // RESET vége
    /* FREQ = PCLK1/1_000_000         */
    I2C1->CR2 = 36;                               // PCLK1 = 36 MHz
    /* CCR = PCLK1/ClkFastSpeed/3, FS =1, DUTY = 0  */
    I2C1->CCR = I2C_CCR_FS + 30;                  // Fast mode 400 kHz, 2:1 duty
    /* TRISE = tRISE/tPCLK1+1 = 300ns*36 MHz +1 */
    I2C1->TRISE = 11;                             // maximum felfutási ido
    I2C1->CR1 |=  I2C_CR1_PE;                     // I2C1 modul indítása
}
```

OLED I2C kijelző SSD1306 vezérlővel

Jellemzők:



- OLED technológia
- 128x64 képpont
- 0,96” (2,4 cm) képátló
- I2C illesztő
- 2.5V-5.5V tápfeszültség
- SSD1306 vezérlő
- Monokróm (egyes változatoknál a felső harmad más színű)
- Grafikus megjelenítés
- Inverz mód
- Görgetés több irányba
- Dokumentáció: Solomon Systech SSD1306 adatlap

SSD1306 programkönyvtárak

- Az általunk használt programkönyvtár eredete ide nyúlik vissza: Tilen Majerle: Library 61- SSD1306 OLED I2C LCD for STM32F4xx
- A fenti könyvtárat **STM32F103C8** mikrovezérlőre átdolgozta Alexander Lutsai: Блиблиотека OLED дисплея SSD1306 для STM32 микроконтроллеров
- Alexander Lutsai programkönyvtárát valaki átdolgozta az **STM32 CubeMx**-hez és két demó programmal együtt közzétette a **Controllerstech.com** honlapon: Oled display using I2C and STM32
- A **Conrollerstech** honlapjáról letölthető programkönyvtárat adaptáltuk a **Keil MDK5 Lite** környezethez és a korábbiakban bemutatott I2C kezelő függvényekhez
- A látványos demók, sajnos, meghaladják az ingyenes **Keil MDK5 Lite** fejlesztői környezet korlátait, ezért itt csak egy egyszerű programokat mutatunk be

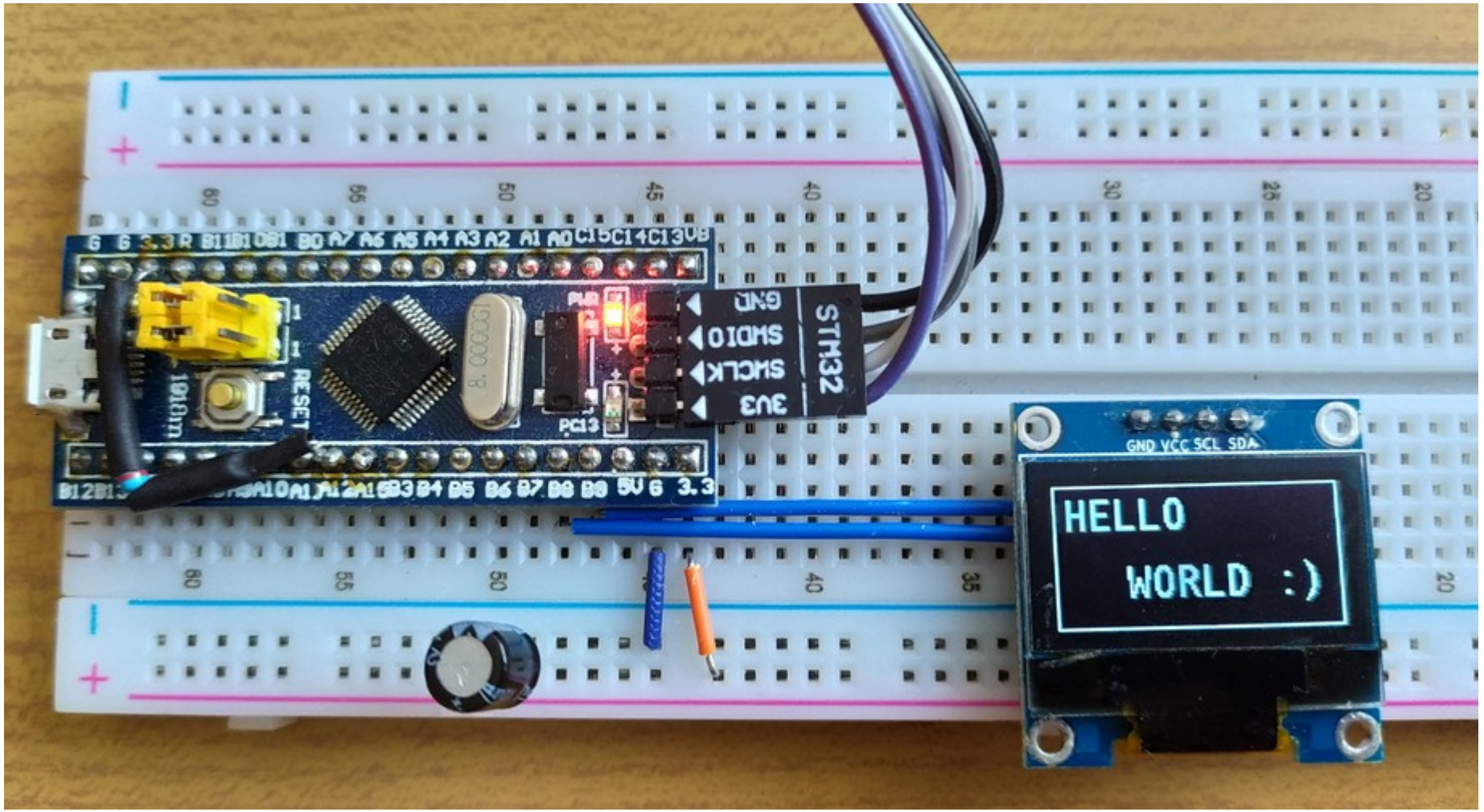
Program09_05: SSD1306 I2C demo

- A projekthez az alábbi forrásfájlokat kell hozzáadni:
  - **main.c** – a főprogram
  - **ssd1306.h** – az SSD1306 programkönyvtár fejléc állománya
  - **ssd1306.c** – az SSD1306 programkönyvtár (az I2C kezeléssel)
  - **fonts.h** – a fontleíró adattömbök típusdefiníciós fejléc állománya
  - **fonts.c** – a fontleíró adattömbök

```
#include "stm32f10x.h"
#include "ssd1306.h"
int main(void) {
    SSD1306_Init();             // inicializálás
    //--- Írjunk ki valamit! -------
    SSD1306_GotoXY(5, 5);
    SSD1306_Puts("HELLO", &Font_11x18, SSD1306_COLOR_WHITE);
    SSD1306_GotoXY(10, 35);
    SSD1306_Puts("  WORLD :)", &Font_11x18, SSD1306_COLOR_WHITE);
    //--- Keretező téglalap kirajzolása ------
    SSD1306_DrawRectangle(0,0,127,63,SSD1306_COLOR_WHITE);
    SSD1306_UpdateScreen();     // Megjelenítés
    while(1) { }
}
```

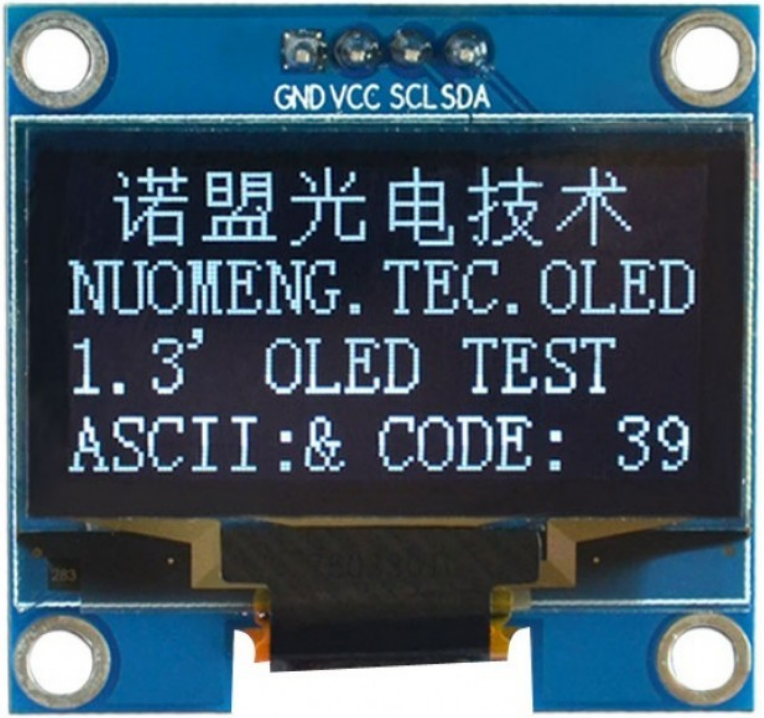
Kapcsolás és programfutttatás

- Bekötés: 3.3V → VDD, GND → GND, PB8 → SCL, PB9 → SDA
- A programfutás eredménye az alábbi ábrán látható



OLED I2C kijelző SH1106 vezérlővel

Jellemzők:



- OLED technológia
- 128x64 képpont (az IC 132x64-at tudna)
- 1.3” (3,3 cm) képátló
- I2C illesztő
- 3,3V-5.0V tápfeszültség
- SH1106 vezérlő
- Monokróm (egyes változatoknál a felső harmad más színű)
- Grafikus megjelenítés
- Inverz mód
- Eltérések SSD1306-tól: nincs görgetés, és 2 pixellel el van tolva a kép
- Dokumentáció: Sino Wealth SH1106 datasheet

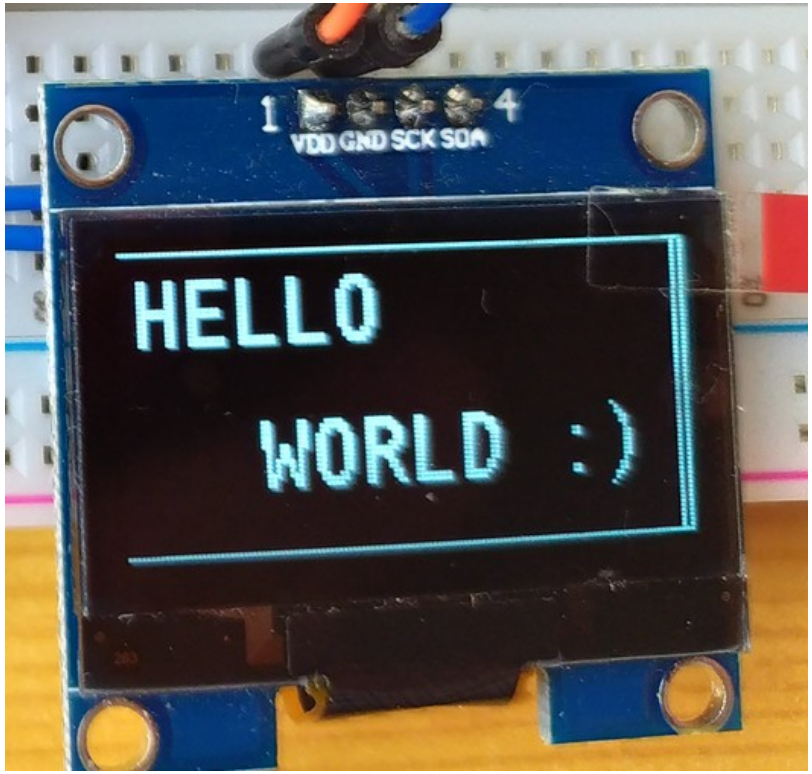
Hogyan használjuk SH1106 kijelzőt?

- Ha az **SH1106** vezérlővel ellátott kijelzőt kipróbáljuk az előző **Program09_5** projekttel, akkor azt látjuk, hogy a kép bal széle hiányzik, jobb szélen pedig extra, random pixelek láthatóak
- A jelenség oka, hogy a RAM – kijelző hozzárendelés 2 pixellel el van tolva, amit korrigálni kell az **ssd1306.c** állományban, így

SSD1306.c részlet

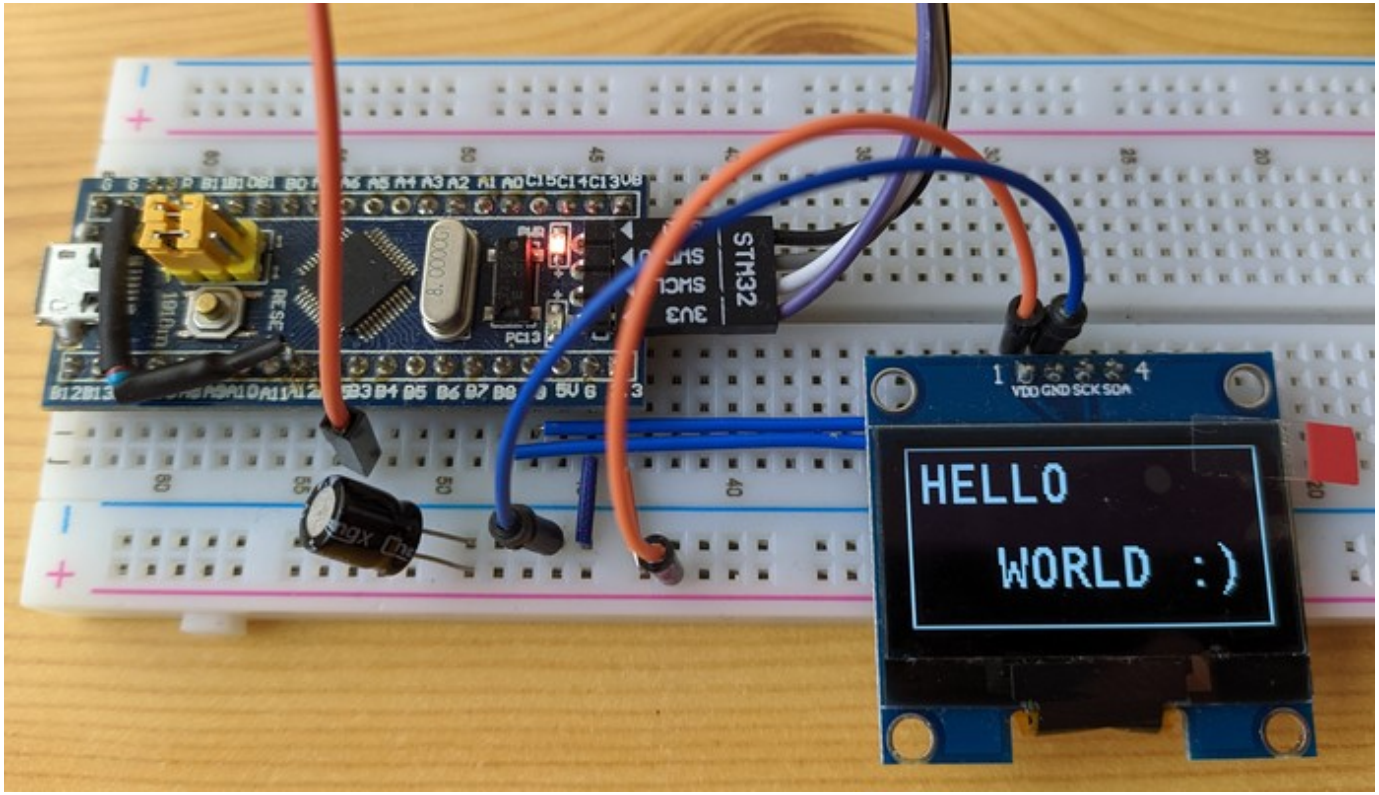
```
void SSD1306_UpdateScreen(void) {
  uint8_t m;

  for (m = 0; m < 8; m++) {
    SSD1306_WRITECOMMAND(0xB0 + m);
    SSD1306_WRITECOMMAND(0x02);   // 0: SSD1306,
    SSD1306_WRITECOMMAND(0x10);   // 2: SH1106
    /* Write multi data */
    ssd1306_I2C_WriteMulti(SSD1306_I2C_ADDR,
    0x40, &SSD1306_Buffer[SSD1306_WIDTH * m],
    SSD1306_WIDTH);
  }
}
```



Program09_6: SH1106 I2C demo

- Lényegében ugyanaz, mint **Program09_5,** de az **ssd1306.c** állományt módosítottuk az **SH1106** kijelzőhöz (2 pixel eltolás, és az úgysem használható görgető függvények eltávolítása)
- Bekötés: 3.3V ⟶ VDD, GND ⟶ GND, PB8 ⟶ SCL, PB9 ⟶ SDA
- A programfutás eredménye az alábbi ábrán látható

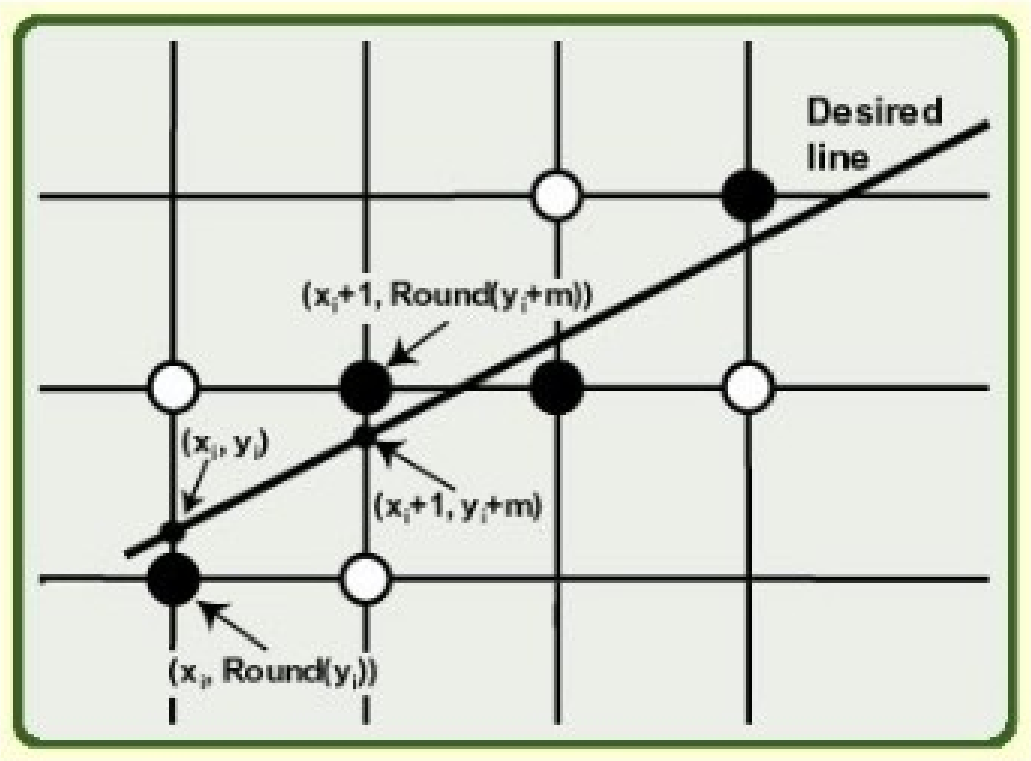


Függvények az SSD1306 kijező használatához

- SSD1306_Init() - a képernyő inicializálása
- SSD1306_UpdateScreen() - képkirajzolás
- SSD1306_ToggleInvert(void) – invertálás ki/bekapcsolása
- SSD1306_Fill(Color) – adott színnel tölti ki a képet
- SSD1306_DrawPixel(x, y, color) – egy pixel kirajzolása
- SSD1306_GotoXY(x, y) – kurzor pozicionálás
- SSD1306_Putc(ch, Font, color) – egy karakter kirajzolása
- SSD1306_Puts(srt, Font, color) – egy string kiírása
- SSD1306_DrawLine(x0, y0, x1, y1, color) – egyenesszakasz kirajzolása
- SSD1306_DrawRectangle(x, y, w, h, color) – téglalap rajzolása
- SSD1306_DrawFilledRectangle(x,y,w,h,color) – kitöltött téglalap rajzolása
- SSD1306_DrawTriangle(x1,y1,x2,y2,x3,y3,color) – háromszög rajzolása
- SSD1306_DrawCircle(x0, y0, r, color) – kör rajzolása
- SSD1306_DrawFilledCircle(x0, y0, r, color) – kitöltött kör rajzolása

Számítógépes grafika: szakasz rajzolása

- A kijelzőn csak diszkrét pontokat tudunk kivilágítani, a ferde szakaszok emiatt" lépcsősek" lesznek
- A koordinátákat emiatt kerekíteni kell, vagy pedig eleve csak a megjeleníthető képpontok koordináta-rendszerében dolgozunk, a koordinátákat egész számokkal kifejezve



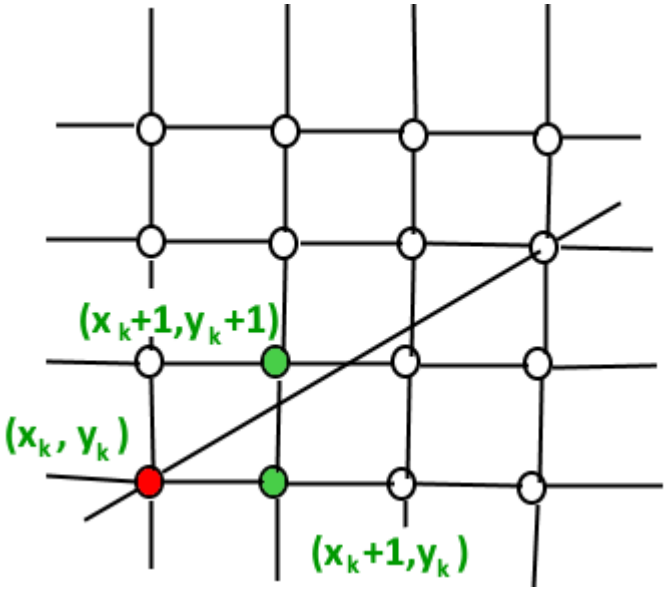
Szakaszrajzoló algoritmusok

- Rajzoljunk szakasz az *(x1,y1)* pontból az *(x2,y2)* pontba!
  (a koordináták egész számokkal adottak!)
- Néhány feltételezéssel élünk, hogy az algoritmus egyszerű legyen:
  - x1 < x2 és y1 < y2
  - A meredekség < 1, azaz (x2-x1) > (y2-y1)
  - Balról jobbra rajzolunk
- A naiv módszer az $y = mx + b$ egyenlet alapján például így néz ki (a változó deklarációkat elhagytuk). Működik, de lassú!

```
// A naive way of drawing line
void naiveDrawLine(x1, x2, y1, y2) {
   m = (y2 - y1)/(x2 - x1)
   for (x  = x1; x <= x2; x++) {
      // Assuming that the round function finds
      // closest integer to a given float.
      y = round(m*(x-x1) + y1);
      plot(x, y);
   }
}
```

A Breshenham algoritmus

- **Jack Elton Bresenham** 1962-ben az IBM-nél dolgozta ki az egészaritmetikájú vonalrajzoló algoritmusát
- Egy (x_k,y_k) pontban arról kell dönteni, hogy az (x_k+1,y_k), vagy az (x_k+1,y_k+1) pontok közül melyik esik közelebb az egyeneshez



- Elképzelés: **slope_error**-ban halmozódik az eltérés, s ha nagyobb 0.5-nél, ***y*** lép egyet, **slope_error** értékét pedig csökkentjük 1-gyel
- Az egészaritmetika érdekében az m meredekség és slope_error értékét felszorozzuk 2*(x2-x1)-gyel. Így m = (y2-y1)/(x2-x1) helyett m = 2*(y2-y1), slope_error

Forrás: Bresenhams line generation algorithm (tutorialspont.dev)

Python3 kód az algoritmus vizsgálatához

- Az alábbi programmal kipróbálhatjuk az algoritmus működését
  Forrás: Bresenhams line generation algorithm (tutorialspont.dev)

```
def bresenham(x1,y1,x2, y2):
    m_new = 2 * (y2 - y1)
    slope_error_new = m_new - (x2 - x1)
    y=y1
    for x in range(x1,x2+1):
        print("(",x ,",",y ,")")
        # Add slope to increment angle formed
        slope_error_new =slope_error_new + m_new
        # Slope error reached limit, time to
        # increment y and update slope error.
        if (slope_error_new >= 0):
            y=y+1
            slope_error_new =slope_error_new - 2 * (x2 - x1)

if __name__=='__main__':
    x1 = 3
    y1 = 2
    x2 = 15
    y2 = 5
    bresenham(x1, y1, x2, y2)

#This code is contributed by ash264
```

Az eredmény:

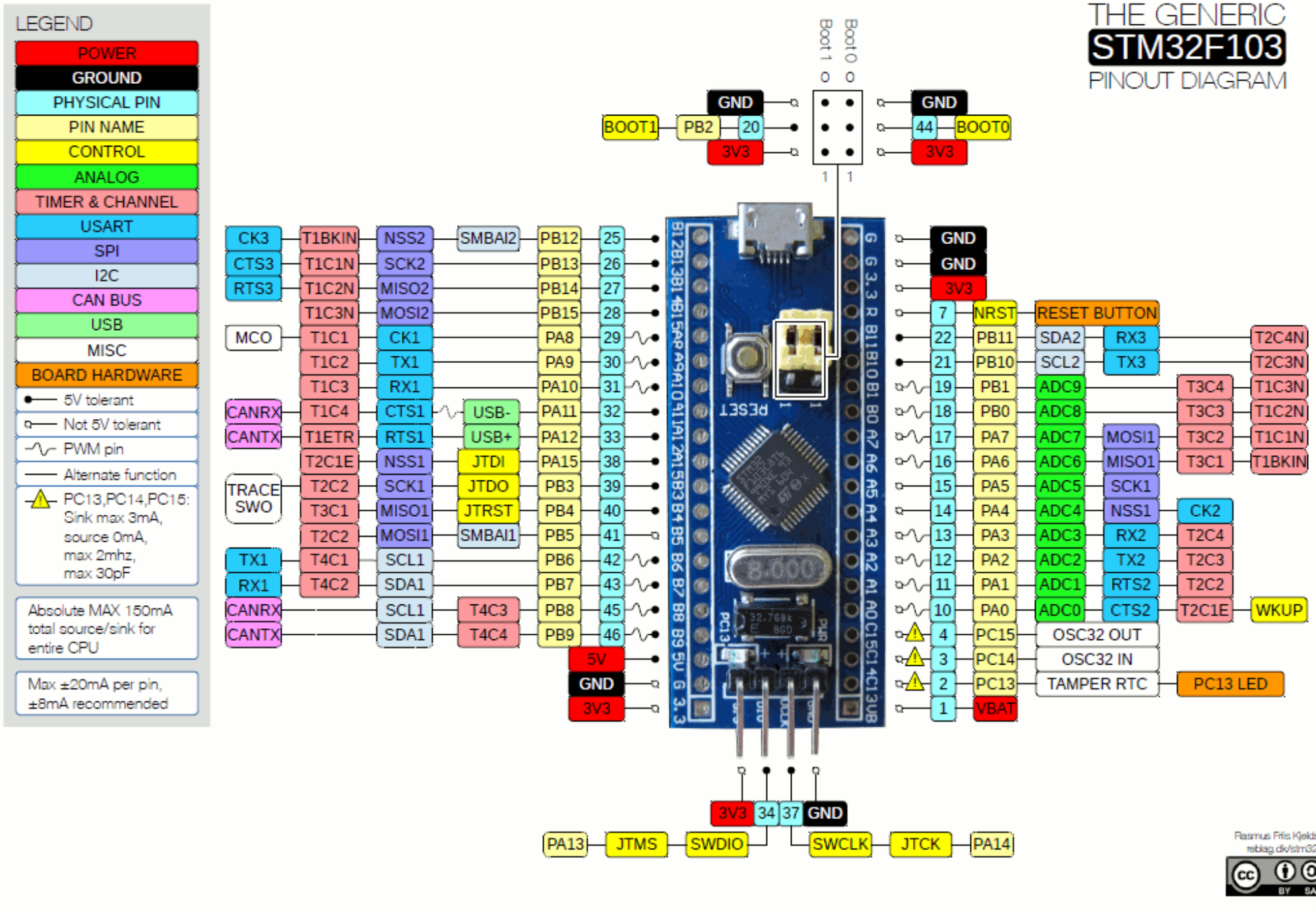
(3 , 2)
(4 , 3)
(5 , 3)
(6 , 3)
(7 , 3)
(8 , 4)
(9 , 4)
(10 , 4)
(11 , 4)
(12 , 5)
(13 , 5)
(14 , 5)
(15 , 5)

A Breshenham algoritmus általánosítása

- Általános esetre tükrözésekkel, illetve a pozitív és negatív hibák kiegyensúlyozásával terjeszthetjük ki a Bresenham algoritmust
- Az **ssd1306.c** forrásállományban az alábbi algoritmussal, ami a Wikipédián is megtalálható (a szócikk legvégén):

```
plotLine(int x0, int y0, int x1, int y1)
    dx =  abs(x1-x0);
    sx = x0<x1 ? 1 : -1;
    dy = -abs(y1-y0);
    sy = y0<y1 ? 1 : -1;
    err = dx+dy;              /* error value e_xy */
    while (true)              /* loop */
        plot(x0, y0);
        if (x0==x1 && y0==y1) break;
        e2 = 2*err;
        if (e2 >= dy)
            err += dy;        /* e_xy+e_x > 0 */
            x0 += sx;
        end if
        if (e2 <= dx)         /* e_xy+e_y < 0 */
            err += dx;
            y0 += sy;
        end if
    end while
```

Közben megváltozott a konvenció: itt most az (x0,y0) pontból húzunk szakaszt az (x1,y1) pontba!


LEGEND
POWER
GROUND
PHYSICAL PIN
PIN NAME
CONTROL
ANALOG
TIMER & CHANNEL
USART
SPI
I2C
CAN BUS
USB
MISC
BOARD HARDWARE
5V tolerant
Not 5V tolerant
PWM pin
Alternate function
PC13,PC14,PC15: Sink max 3mA, source 0mA, max 2mhz, max 30pF
Absolute MAX 150mA total source/sink for entire CPU
Max ±20mA per pin, ±8mA recommended
THE GENERIC
STM32F103
PINOUT DIAGRAM
Boot1
Boot0
GND
BOOT1
PB2
20
3V3
GND
44
BOOT0
3V3
CK3
CTS3
RTS3
MCO
CANRX
CANTX
TRACE SWO
TX1
RX1
CANRX
CANTX
T1BKIN
T1C1N
T1C2N
T1C3N
T1C1
T1C2
T1C3
T1C4
T1ETR
T2C1E
T2C2
T3C1
T2C2
T4C1
T4C2
NSS2
SCK2
MISO2
MOSI2
CK1
TX1
RX1
CTS1
RTS1
NSS1
SCK1
MISO1
MOSI1
SCL1
SDA1
SCL1
SDA1
SMBAI2
USB-
USB+
JTDI
JTDO
JTRST
SMBAI1
T4C3
T4C4
PB12
PB13
PB14
PB15
PA8
PA9
PA10
PA11
PA12
PA15
PB3
PB4
PB5
PB6
PB7
PB8
PB9
25
26
27
28
29
30
31
32
33
38
39
40
41
42
43
45
46
5V
GND
3V3
GND
GND
3V3
7
22
21
19
18
17
16
15
14
13
12
11
10
4
3
2
1
NRST
PB11
PB10
PB1
PB0
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC15
PC14
PC13
VBAT
RESET BUTTON
SDA2
SCL2
ADC9
ADC8
ADC7
ADC6
ADC5
ADC4
ADC3
ADC2
ADC1
ADC0
OSC32 OUT
OSC32 IN
TAMPER RTC
RX3
TX3
MOSI1
MISO1
SCK1
NSS1
RX2
TX2
RTS2
CTS2
T3C4
T3C3
T3C2
T3C1
CK2
T2C4
T2C3
T2C2
T2C1E
PC13 LED
T2C4N
T2C3N
T1C3N
T1C2N
T1C1N
T1BKIN
WKUP
3V3
34
37
GND
PA13
JTMS
SWDIO
SWCLK
JTCK
PA14
Rasmus Friis Kjeldsen
reblag.dk/stm32
V1.0