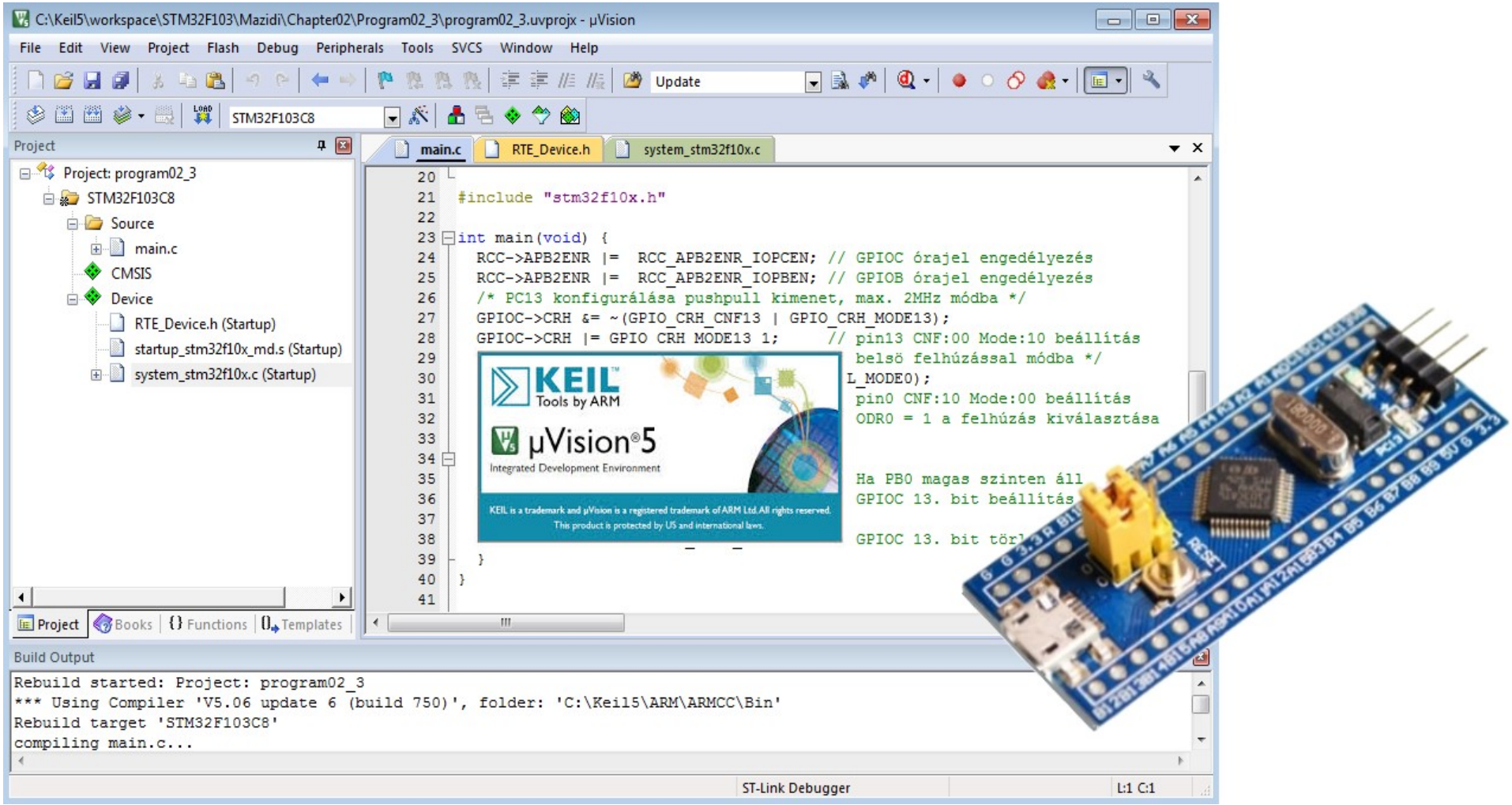
STM32 mikrovezérlők programozása ARM Keil környezetben



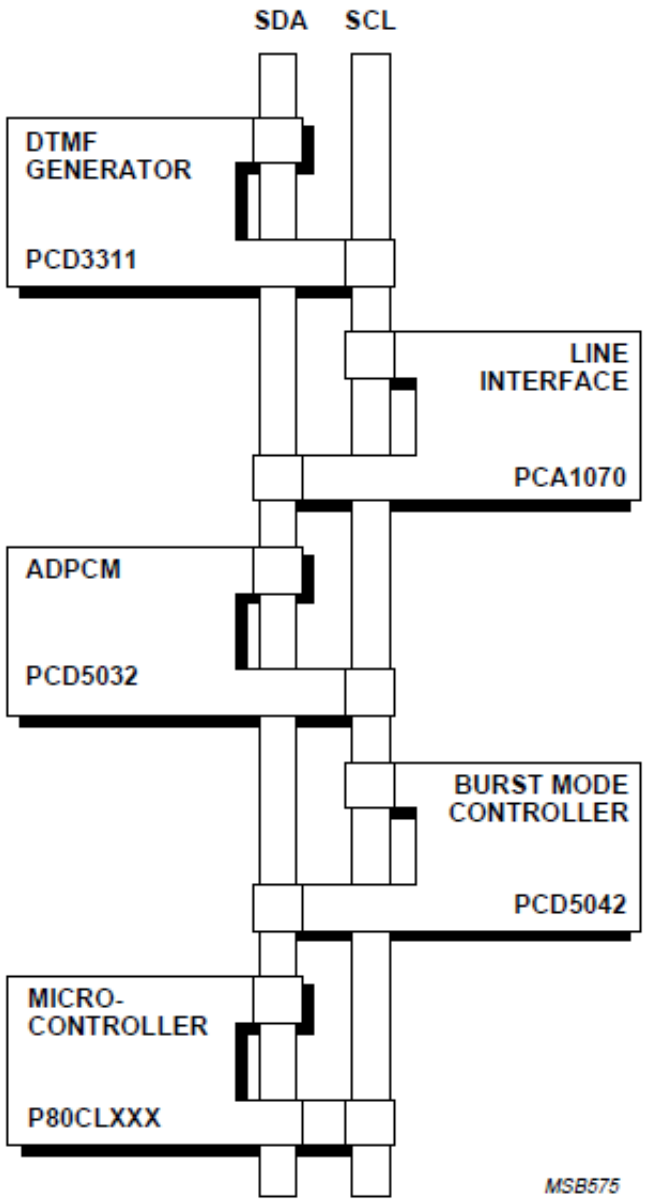
9. Az I2C kommunikációs csatorna

Felhasznált és ajánlott irodalom

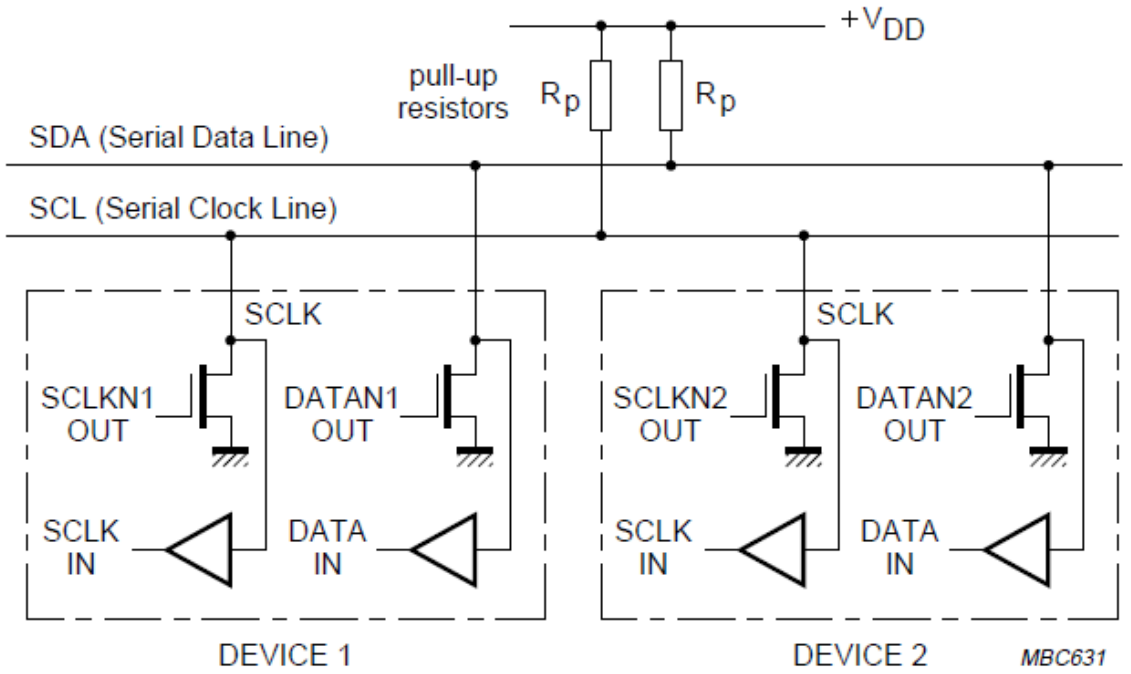
- Joseph Yiu: Cortex-M for Beginners
- Joseph Yiu: The Definitive Guide To The ARM CORTEX-M3
- Muhammad Ali Mazidi, Shujen Chen, Eshragh Ghaemi: STM32 Arm Programming for Embedded Systems
- Alexander Tarasov: **Курс «Штурмуем STM32»**
- Warren Gay: Beginning STM32 - Developing with FreeRTOS, libopencm3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

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

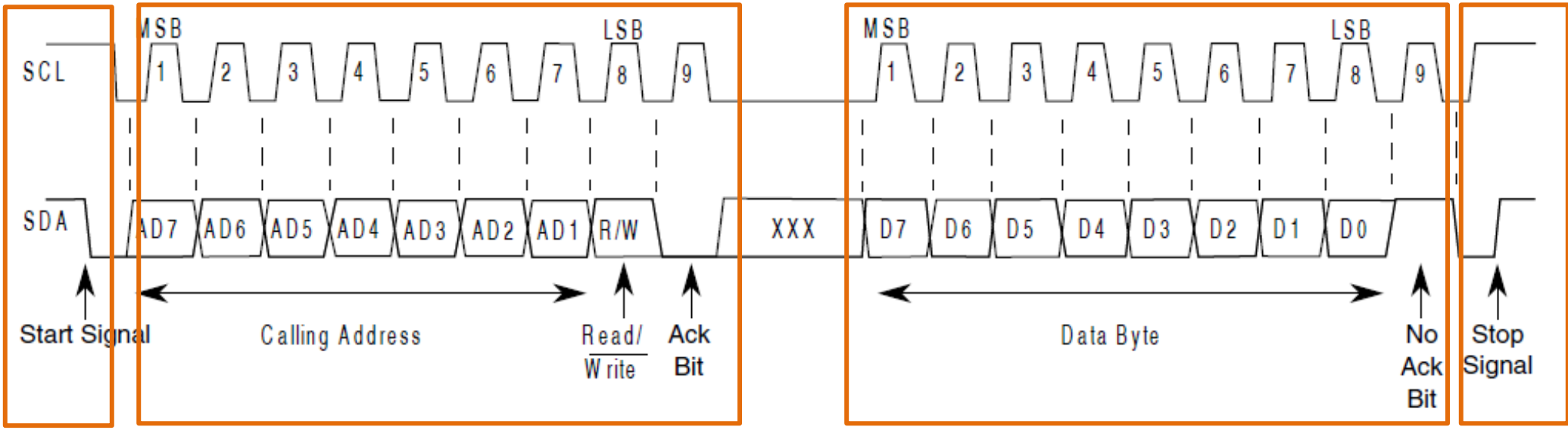


Az I2C busz vezérlése



- 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:

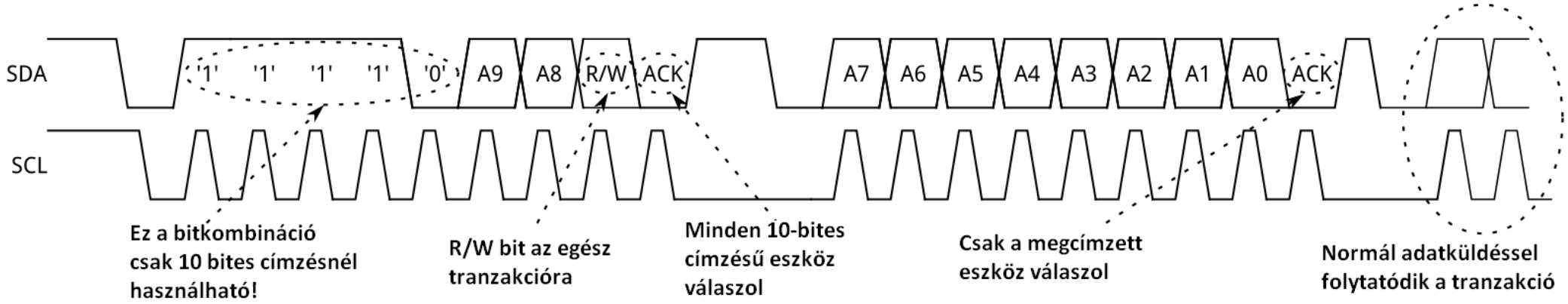
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

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.

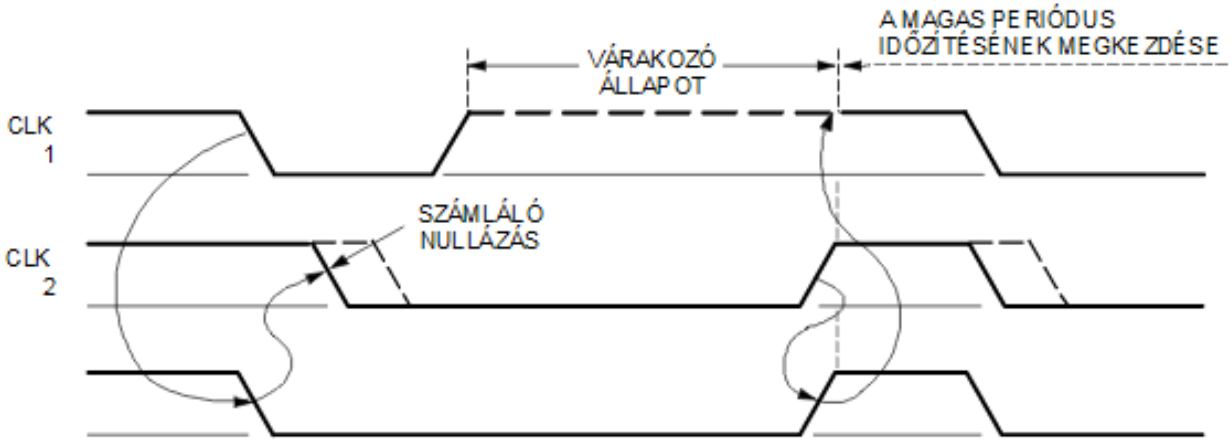


Órajel szinkronizálás

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

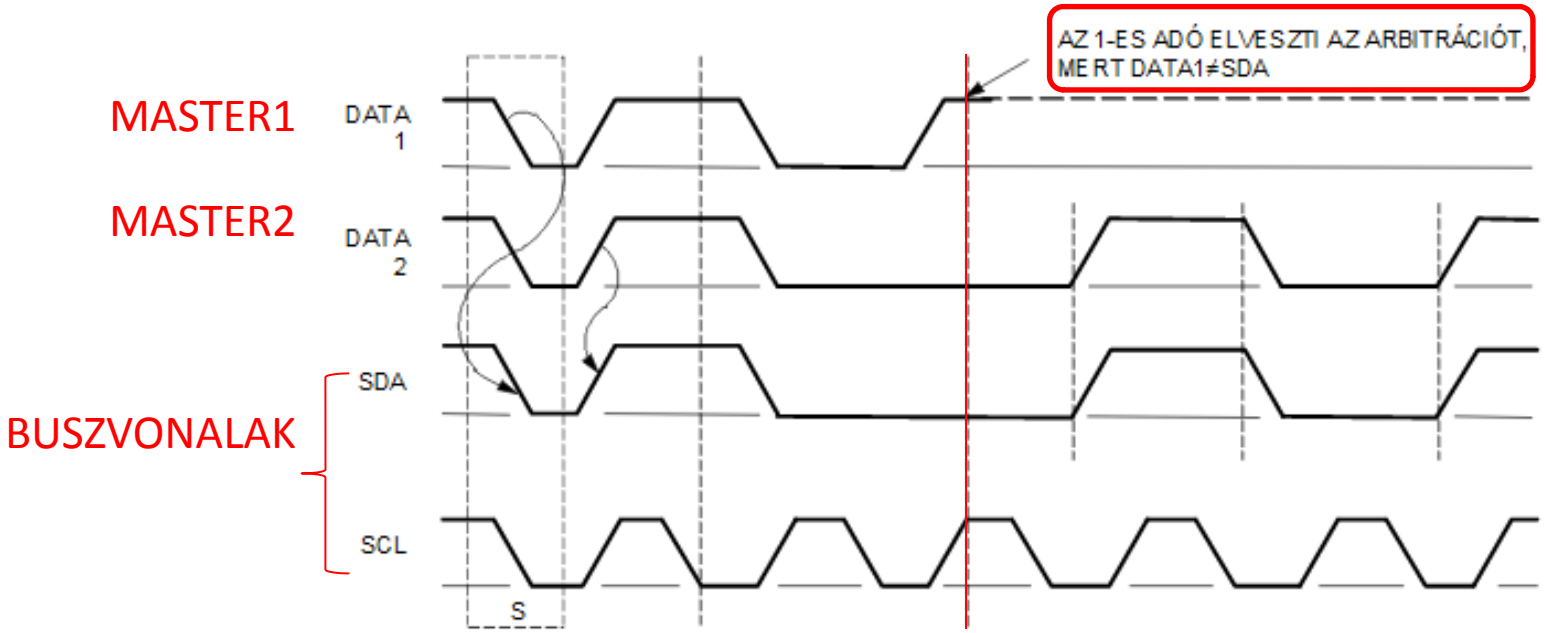
Az órajelszinkronizáció az I²C interfészek huzalozott **ÉS** kapcsolatával megy végbe (a vezetéken csak akkor lesz magas szint, ha minden eszköz magas szintre vált).

- 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.



Arbitráció

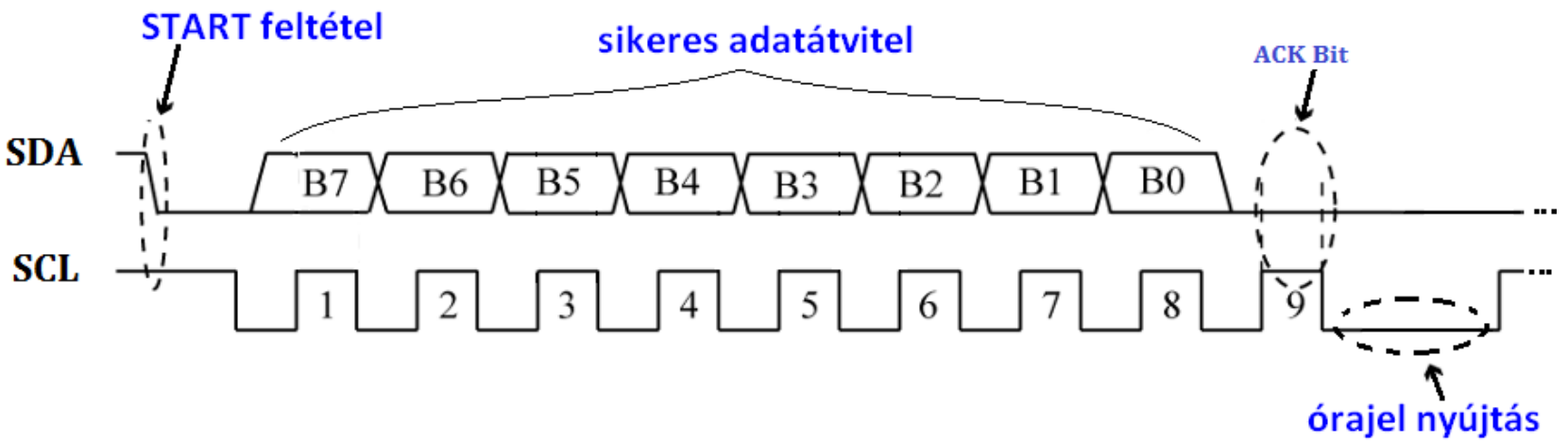
- Multi-master környezetben csak akkor kezdhetünk átvitelt, ha a busz szabad.
- Két, vagy több master generálhat egy start feltételt a START feltétel minimális tartási idején belül. Amig az SCL vezeték magas szinten van az SDA vezetéken végbemegy az arbitráció.
- Az a master veszít, amelyik magas szintet 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



Órajel megnyújtása

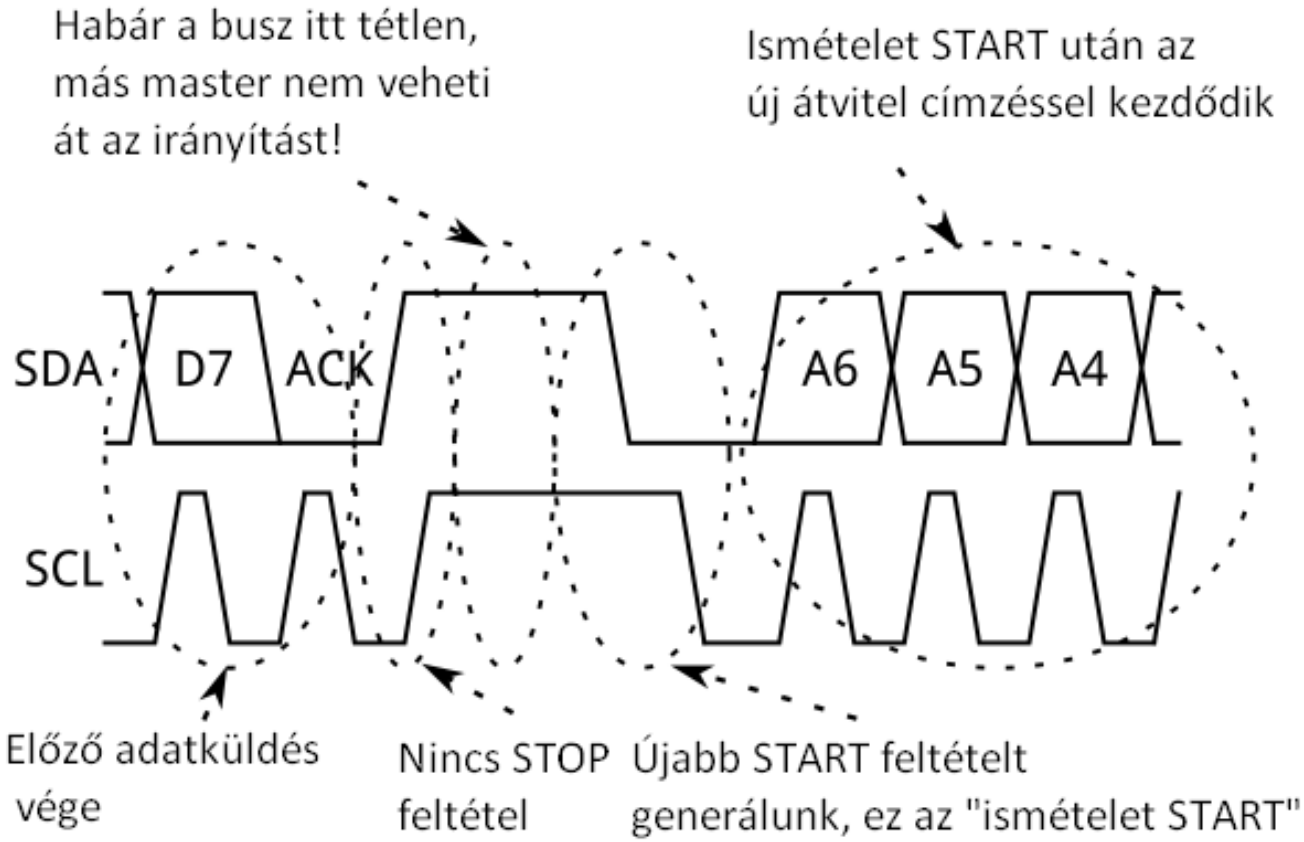
Az órajelet a mester generálja, de előfordulhat, hogy egy slave eszköz nem képes együttműködni a diktált tempóval, s „időt kér”. A már említett órajelszinkronizációs mechanizmus használható fel erre. Ez azt jelenti, hogy egy eszköz képes lehet az adatbájtok gyors fogadására, de az adat eltárolására vagy 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.



Ismételt START feltétel

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.



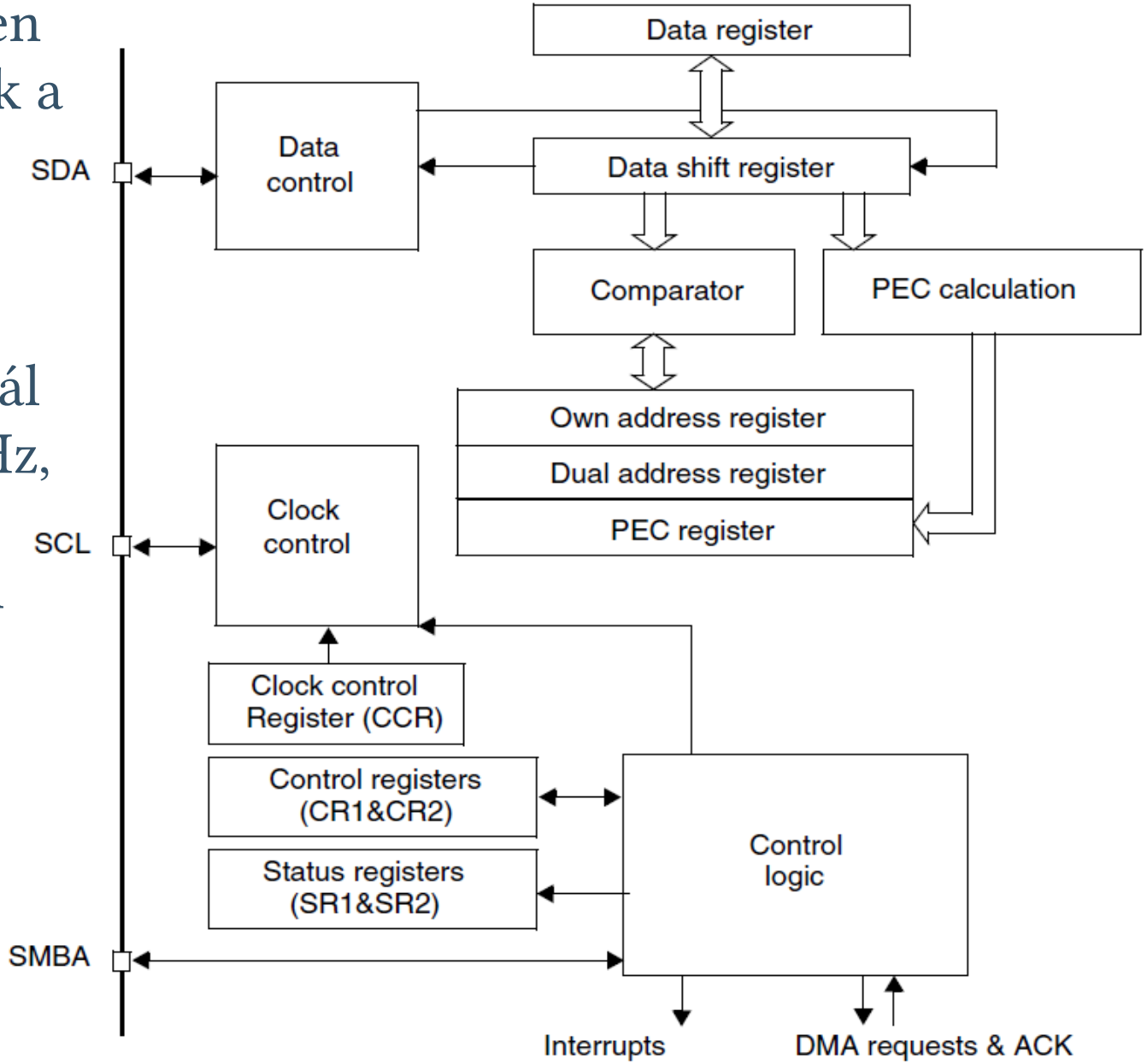
Az STM32F103C8 mikrovezérlő I2C perifériái

Az **STM32F103C8** mikrovezérlő két I2C modult tartalmaz (**I2C1** és **I2C2**) amelyek 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)
- Analóg zajszűrés, állapot- és hibajelző bitek, megszakítások
- 1 bájtos buffer, DMA lehetőséggel
- Konfigurálható PEC (packet error checking)
- SMBus 2.0/PMbus kompatibilitás

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** jel csak SMBus módban áll rendelkezésre



Az I2C modulok engedélyezése

- Az **I2C** modulok az **APB1** fejlett periféria buszra csatlakoznak, engedélyezésük az **RCC** modul **APB1ENR** regiszterében történik
- Az **RCC_APB1ENR** regiszter:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

Bit 22: I2C2 órajelének engedélyezése (**0:** tilt, **1:** enged)
Bit 21: I2C1 órajelének engedélyezése (**0:** tilt, **1:** enged)

- Engedélyezési példa:

```
#include "stm32f10x.h"
RCC->APB1ENR |=  RCC_APB1ENR_I2C1EN;      // I2C1 engedélyezése
RCC->APB1ENR |=  RCC_APB1ENR_I2C2EN;      // I2C2 engedélyezése
```

A kivezetések konfigurálása

- Az **AFIO** modul **AFIO_MAPR** regiszter **I2C1_REMAP** bitjének '1'-be állításával választhatjuk az alternatív kivezetéseket
- A kivezetéseket alternatív funkció, Open Drain módba kell állítani

```
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)
```

I2C jel	Alapértelmezett kivezetés	Alternatív kivezetés
I2C1 SCL	PB6	PB8
I2C1 SDA	PB7	PB9
I2C1 SMBA	PB5	–
I2C2 SCL	PB10	–
I2C2 SDA	PB11	–
I2C2 SMBA	PB12	–

Vezérlő regiszterek

- **I2C*n*_CR1** regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

SWRST – szoftver reset, **STOP, START** – feltétel generálás, **PE** – periféria engedélyezés

- **I2C*n*_CR2** regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			LAST	DMA EN	ITBUF EN	ITEVTEN	ITERR EN	Reserved		FREQ[5:0]					
			rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

FREQ[5:0] – az **APB1** busz órajel frekvenciája (***PCLK1***) MHz-ben
minimális értéke: 2, maximális értéke: 50
Esetünkben FCPU = 72 MHz, PCLK1 = 36 MHz

Állapotjelző regiszterek

- **I2C*n*_SR1** regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

TxE – adatregiszter üres, **RxNE** – vevő regiszter nem üres, **BTF** – bájt átvitel vége
ADDR – cím kiküldve és elfogadva, **SB** – START feltétel generálva

- **I2C*n*_SR2** regiszter

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMBDE FAULT	GEN CALL	Res.	TRA	BUSY	MSL
r	r	r	r	r	r	r	r	r	r	r	r		r	r	r

BUSY – az I2C busz foglaltságát jelzi (ha SCL vagy SDA alacsony)

Adatregiszter

- **I2C*n*_DR** regiszter

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

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

- **I2C*n*_TRISE** regiszter

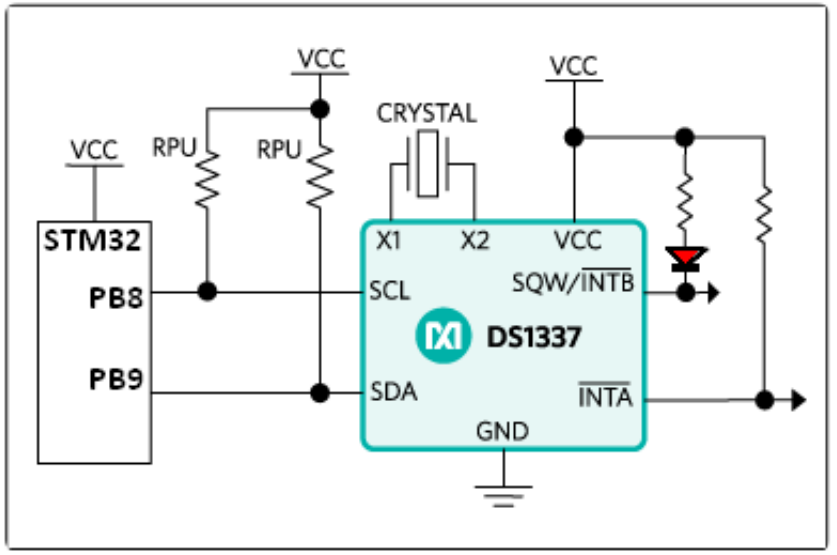
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										TRISE[5:0]					
										rw	rw	rw	rw	rw	rw

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

Például standard módban **SCL** max. felfutási ideje 1000 ns. Ha pl PCLK1 = 8 MHz, azaz T_{PCLK1} = 125 ns, akkor TRISE = 1000 ns / 125 ns + 1 = 8 + 1

Program09_1: adatküldés az I2C buszon

- Egy **DS1337** (vagy **DS3231**) RTC modul vezérlő regiszterébe (**0x0E**) nullát írunk, hogy bekapcsoljuk az 1 Hz-es négyszögjel kimenetet
- Az **SQW** kivezetésre egy LED katódját kötve szabad szemmel is ellenőrizhetjük a működést



Control Register (0Eh)

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NAME:	$\overline{EOSC}$	BBSQW	CONV	RS2	RS1	INTCN	A2IE	A1IE
POR:	0	0	0	1	1	1	0	0

- INTCN = 0 esetén működik az SQW négyszögjel kimenet
- RS2 = RS1 =0 esetén 1 Hz

SQUARE-WAVE OUTPUT FREQUENCY

RS2	RS1	SQUARE-WAVE OUTPUT FREQUENCY
0	0	1Hz
0	1	1.024kHz
1	0	4.096kHz
1	1	8.192kHz

Program09_1/main.c

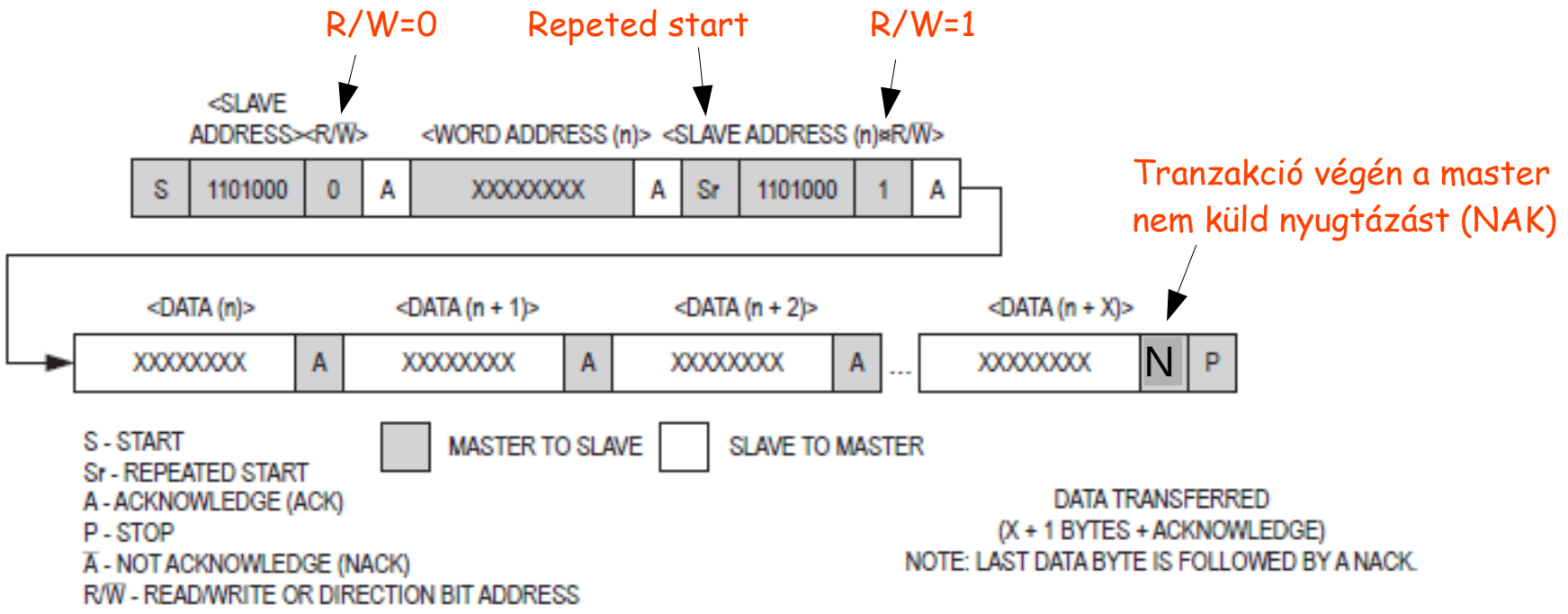
```
#include "stm32f10x.h"
void I2C1_init(void);
int I2C1_byteWrite(char saddr, char maddr, char data);

#define SLAVE_ADDR 0x68    /* 1101 000, a DS3231 eszköz I2C címe */

int main(void) {
    I2C1_init();
    I2C1_byteWrite(SLAVE_ADDR, 0x0E, 0);
    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
  I2C1->CR1 |=  I2C_CR1_PE;                    // I2C1 modul indítása
}
```

Program09_1/main.c (folytatás)

```
/* Ez a függvény kiír egy adatbájtot (data) a saddr című
 * I2C eszköz maddr című regiszterébe. Az egyszerűség kedvéért
 * nincs ellenőrzés, sem hibajelzés.
 */
int I2C1_byteWrite(char saddr, char maddr, char data) {
    volatile int tmp;

    while (I2C1->SR2 & 2);          BUSY    /* Vár, amíg a busz foglalt */

    I2C1->CR1 |=  0x100;            SB      /* START generálás */
    while (!(I2C1->SR1 & 1));               /* Vár, amíg SR1.SB = 0 */

    I2C1->DR = saddr << 1;                  /* Slave cím kiküldése */
    while (!(I2C1->SR1 & 2));       ADDR    /* Vár, amíg SR1.ADDR = 0 */
    tmp = I2C1->SR2;                        /* SR1.ADDR törlése */

    while (!(I2C1->SR1 & 0x80));    TXE     /* Vár, amíg SR1.TXE = 0 */
    I2C1->DR = maddr;                       /* Memóriacím küldése */

    while (!(I2C1->SR1 & 0x80));    TXE     /* Vár, amíg SR1.TXE = 0 */
    I2C1->DR = data;                        /* Adat küldése */

    while (!(I2C1->SR1 & 4));       BTF     /* Vár, amíg SR1.BTF = 0  */
    I2C1->CR1 |= 0x200;                     /* DTOP generálás */
    return 0;
}
```

Program09_2: adatlekérés az I2C buszon

- Most is egy **DS1337** (vagy **DS3231**) RTC modullal kommunikálunk, kiolvassuk a **0x00** című regisztert (másodpercek), s a legalsó bit értékét kijelezzük a beépített LED (**PC13** GPIO) segítségével
- A tranzakció írás (regisztercím) és olvasást (regiszter tartalom) is tartalmaz, s a kettő között a ***repeated start*** feltétel generálása biztosítja a busz folyamatos elérhetőségét, mely után újra ki kell küldeni a slave I2C címét



Program09_2/main.c

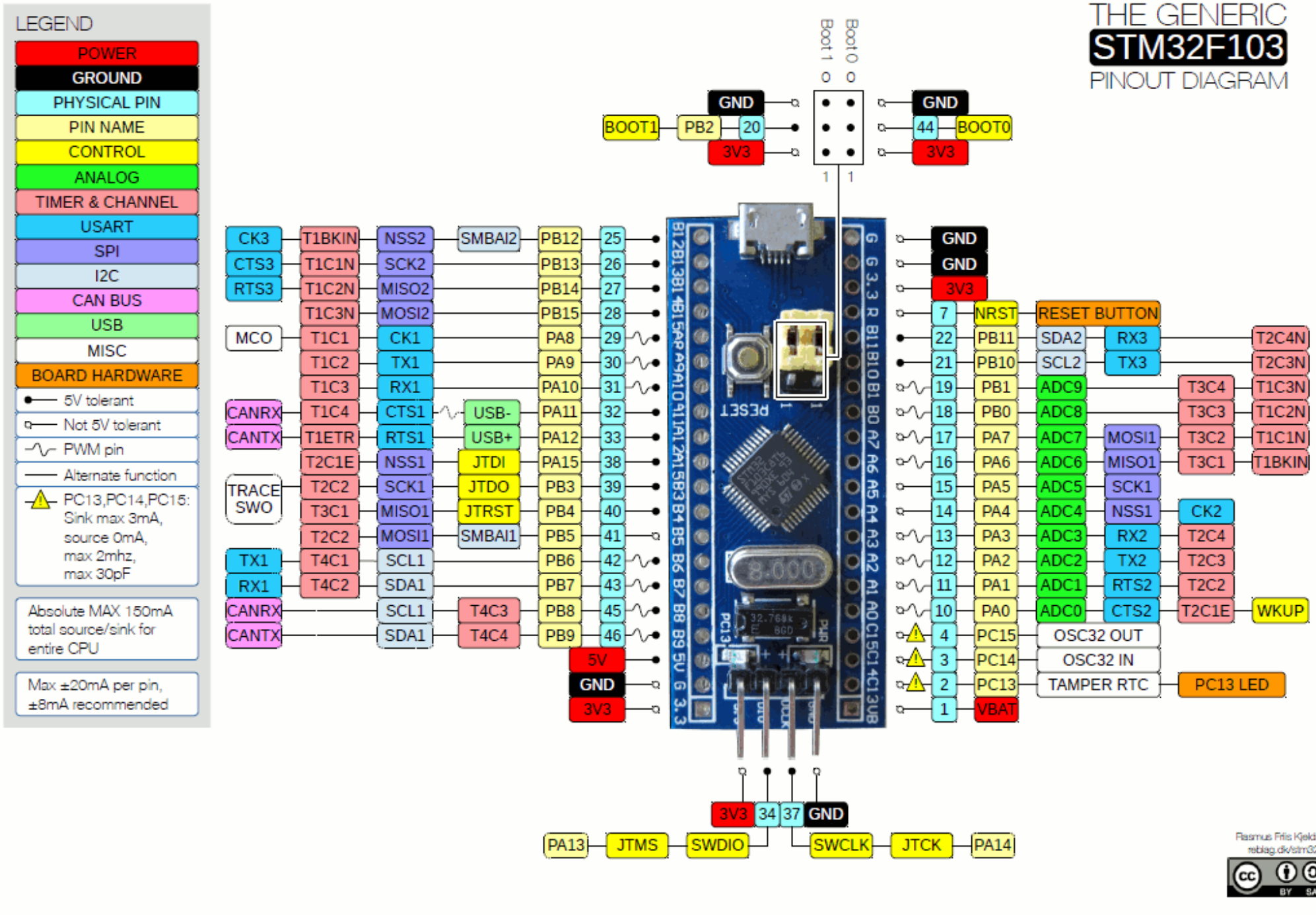
```
#include "stm32f10x.h"
#define SLAVE_ADDR 0x68    /* 1101 000.    DS3231 */

void I2C1_init(void);
int I2C1_byteRead(char saddr, char maddr, char* data);
void delayMs(int n);

int main(void) {
    char data;
  //--- PC13, a beépített LED konfigurálása --------
  RCC->APB2ENR |=  RCC_APB2ENR_IOPCEN;   // GPIOC órajel engedélyezés
  GPIOC->CRH &= ~(GPIO_CRH_CNF13|GPIO_CRH_MODE13); // pin13 CNF/Mode bitek törlése
  GPIOC->CRH |= GPIO_CRH_MODE13_1;       // CNF:00, Mode:10 PPout, 2MHz

  I2C1_init();

  while (1) {
    I2C1_byteRead(SLAVE_ADDR, 0, &data);
    if (data & 1)
      GPIOC->ODR &= ~GPIO_ODR_ODR13;     // LED bekapcsolás
    else
      GPIOC->ODR |=  GPIO_ODR_ODR13;     // LED kikapcsolás

    delayMs(10);
  }
}
```

Program09_2/main.c (folytatás)

```
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
    /* FREQ = PCLK1/1_000_000          */
  I2C1->CR2 = 36;                              // PCLK1 = 36 MHz
    /* CCR = max(PCLK1/ClkSpeed/2,4)   */
  I2C1->CCR = 180;                             // Standard mód 100 kHz
    /* TRISE = PCLK1/1000000+1         */
  I2C1->TRISE = 37;                            // maximum felfutási idő
  I2C1->CR1 |=  I2C_CR1_PE;                    // I2C1 modul indítása
}

void delayMs(int n) {
  int i;
  SysTick->LOAD = SystemCoreClock/1000-1;// Újratöltési érték 1 ms késleltetéshez
  SysTick->VAL  = 0;                     // Számláló törlése
  SysTick->CTRL = 0x5;                   // Engedélyezés, no int, rendszer órajel
  for(i = 0; i < n; i++)
    while((SysTick->CTRL & 0x10000)==0); // A COUNT jelzöre várunk
  SysTick->CTRL = 0;                     // SysTick leállítása
}
```

Program09_2/main.c (folytatás)

```
int I2C1_byteRead(char saddr, char maddr, char* data) {
    volatile int tmp;
    while (I2C1->SR2 & 2);              BUSY    /* wait until bus not busy */
    I2C1->CR1 |=  0x100;                        /* generate start */
    while (!(I2C1->SR1 & 1));           SB      /* wait until start flag is set */
    I2C1->DR = saddr << 1;                      /* transmit slave address + Write */
    while (!(I2C1->SR1 & 2));           ADDR    /* wait until addr flag is set */
    tmp = I2C1->SR2;                            /* clear addr flag */
    while (!(I2C1->SR1 & 0x80));        TXE     /* wait until data register empty */
    I2C1->DR = maddr;                           /* send memory address */
    while (!(I2C1->SR1 & 0x80));        TXE     /* wait until data register empty */

    I2C1->CR1 |= 0x100;                 START   /* generate restart */
    while (!(I2C1->SR1 & 1));           SB      /* wait until start flag is set */
    I2C1->DR = saddr << 1 | 1;                  /* transmit slave address + Read */
    while (!(I2C1->SR1 & 2));           ADDR    /* wait until addr flag is set */
    I2C1->CR1 &= ~0x400;                ACK     /* Disable Acknowledge */
    tmp = I2C1->SR2;                            /* clear addr flag */
    I2C1->CR1 |= 0x200;                 STOP    /* generate stop after data received */
    while (!(I2C1->SR1 & 0x40));        RXNE    /* Wait until RXNE flag is set */
    *data++ = I2C1->DR;                         /* Read data from DR */

    return 0;
}
```


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