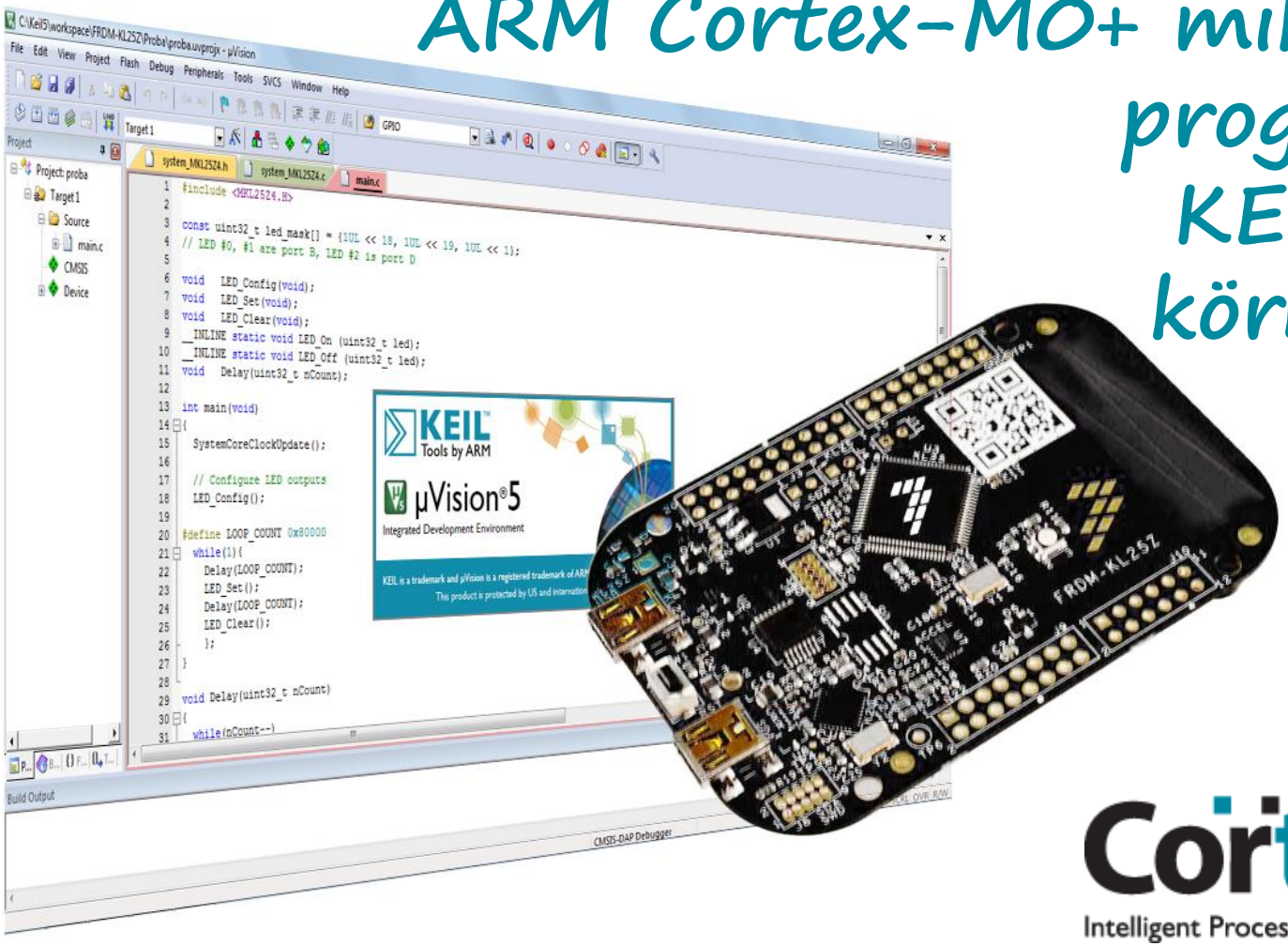
ARM Cortex-M0+ mikrovezérlő programozása KEIL MDK 5 környezetben



Cortex
Intelligent Processors by ARM

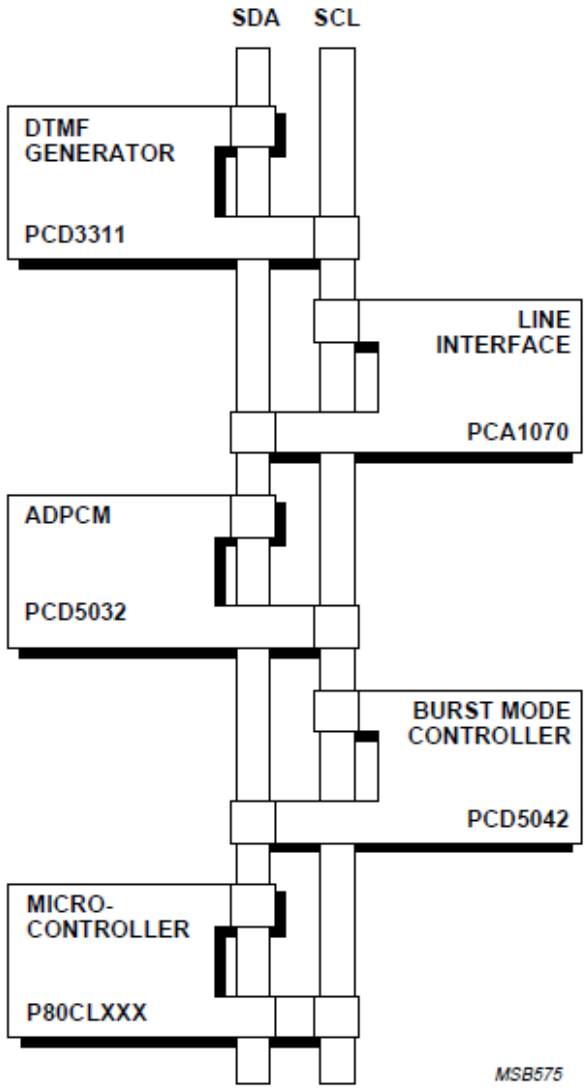
9. Az I2C kommunikációs csatorna

Felhasznált anyagok, ajánlott irodalom

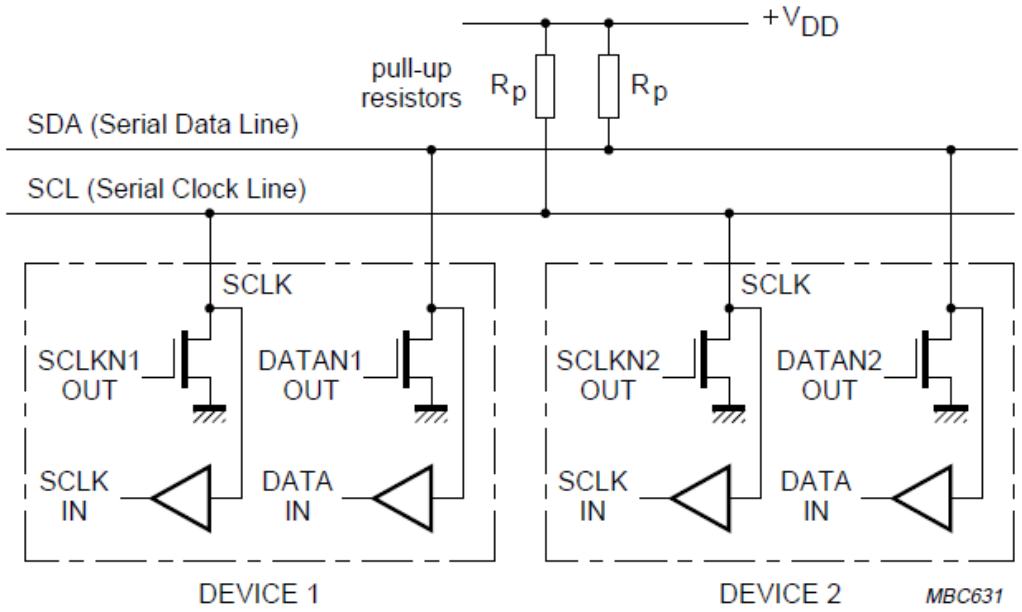
- Joseph Yiu: **The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors** (2nd Ed.)
- Muhammad Ali Mazidi, Shujen Chen, Sarmad Naimi, Sepehr Naimi: **Freescale ARM Cortex-M Embedded Programming**
- ARM University Program: **Course/Lab Material for Teaching Embedded Systems/MCUs** (for the **FRDM-KL25Z** board)
- ARM Information Center: **Cortex-M0+ Devices Generic User Guide**
- Freescale: **MKL25Z128VLK4 MCU datasheet**
- Freescale: **KL25 Sub-Family Reference Manual**
- Freescale: **FRDM-KL25Z User Manual**
- Freescale: **FRDM-KL25Z Pinouts**

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 I^2C busz két jelvezetéket használ**
  - **SCL:** szinkronizáló órajel
  - **SDA:** soros adat
- **A részletes leírás** az ”Az I^2C-busz specifikációja és használata” című dokumentumban olvasható

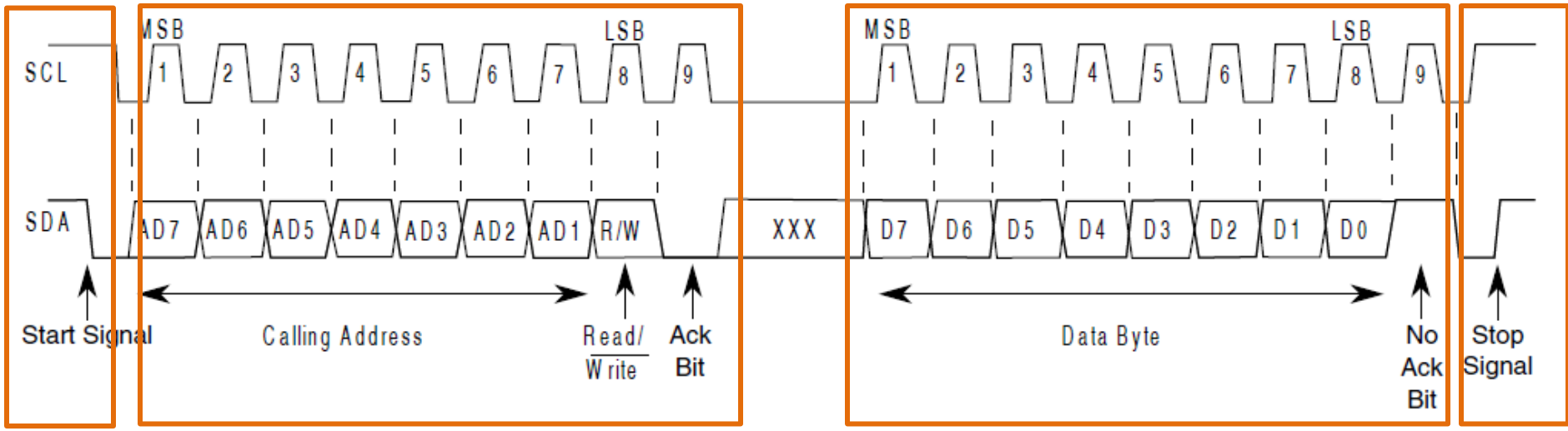


Az I2C busz vezérlése



- 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:

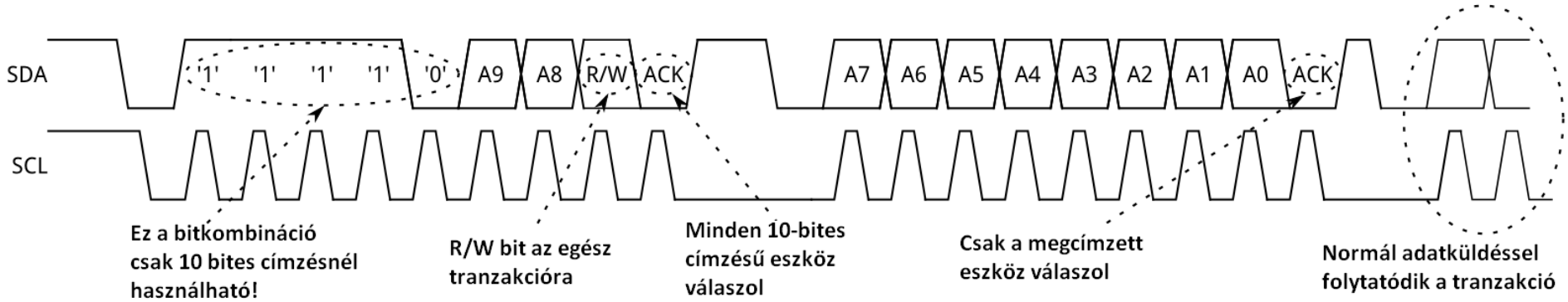
1. **Start** feltétel
2. **A szolga eszköz megcímzése**
   - 7-bites cím
   - Parancsbit (1: olvasás, 0: írás)
   - Nyugtázás (a vevő visszajelzése)
3. **Adatmező**
   - Adatbájt
   - Nyugtázás (a vevő visszajelzése)
4. **Stop** feltétel

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 minen 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.

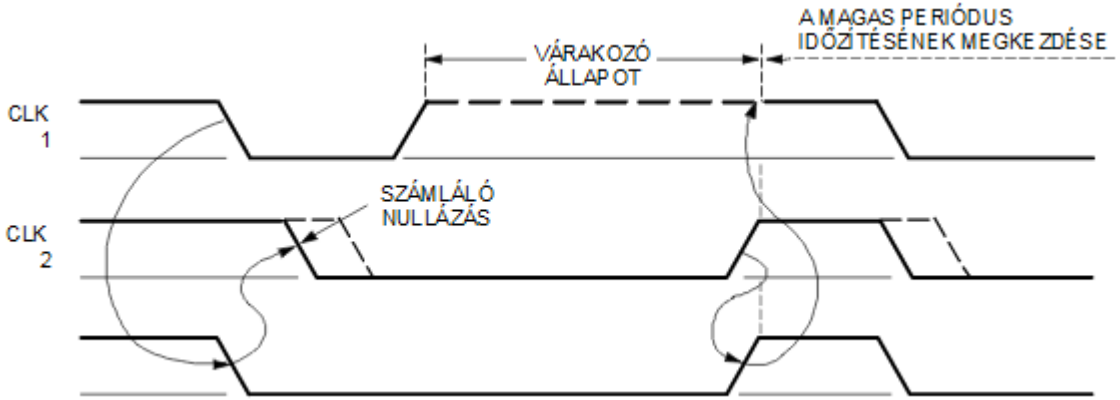


Órajel szinkronizálás

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

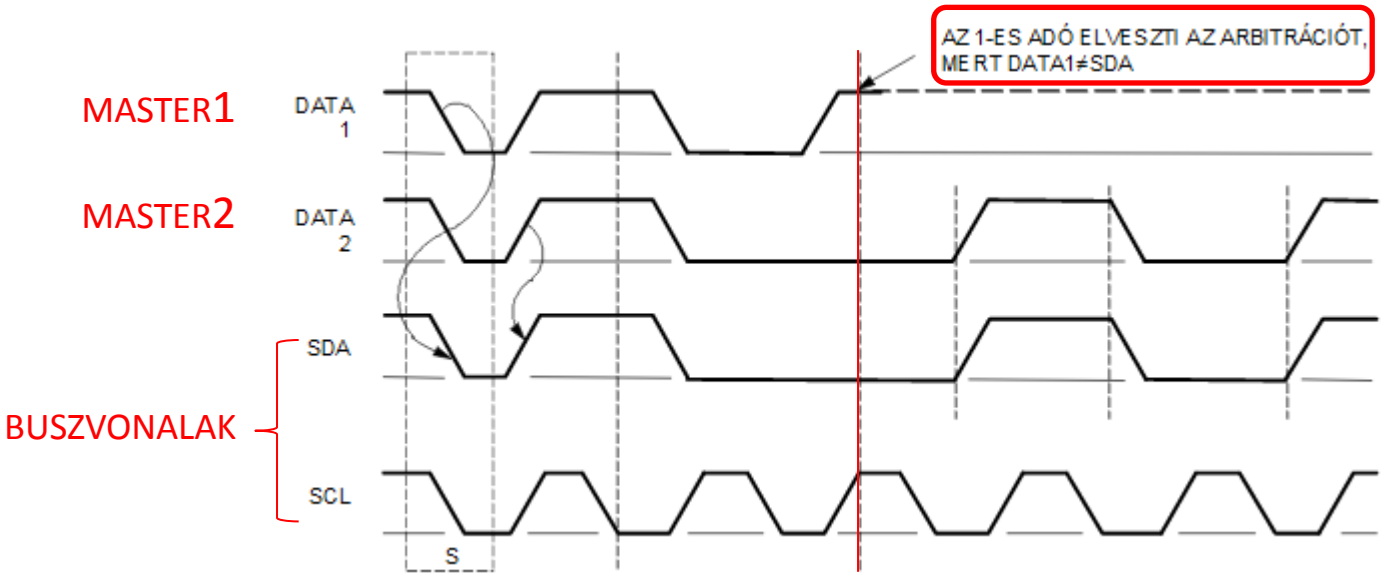
Az órajelszinkronizáció az I²C interfészek huzalozott **ÉS** kapcsolatával megy végbe (a vezetéken csak akkor lesz magas szint, ha minden eszköz magas szintre vált).

- 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 .

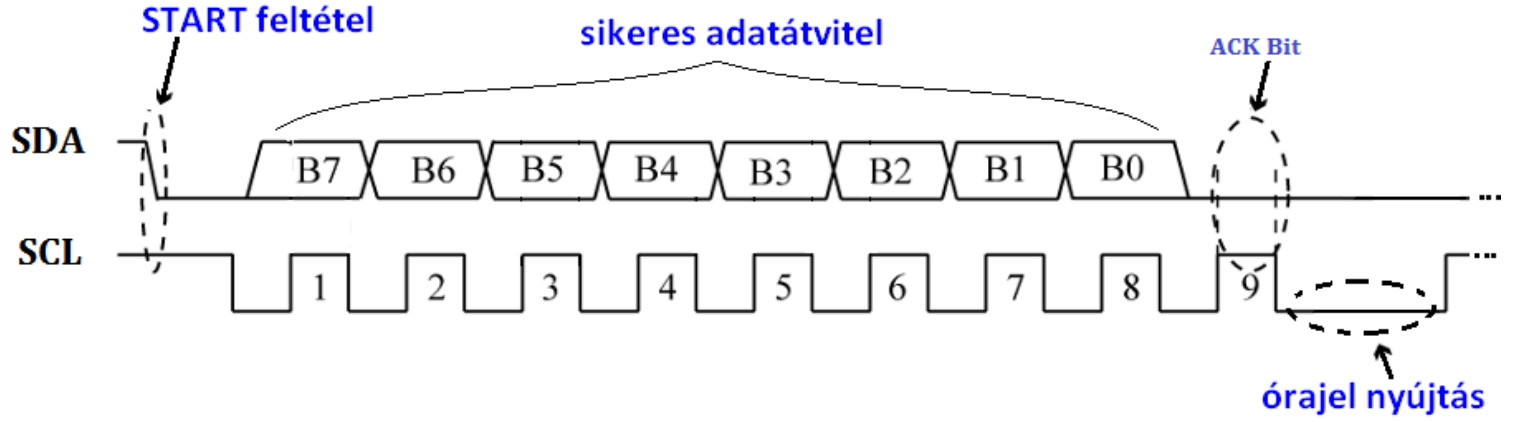


Órajel megnyújtása

Az órajelet a mester generálja, de előfordulhat, hogy egy slave eszköz nem képes együttműködni a master által 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 a fogadott byte eltárolására vagy egy másik byte átvitelhez való előkészítéséhez több időre van szüksége.

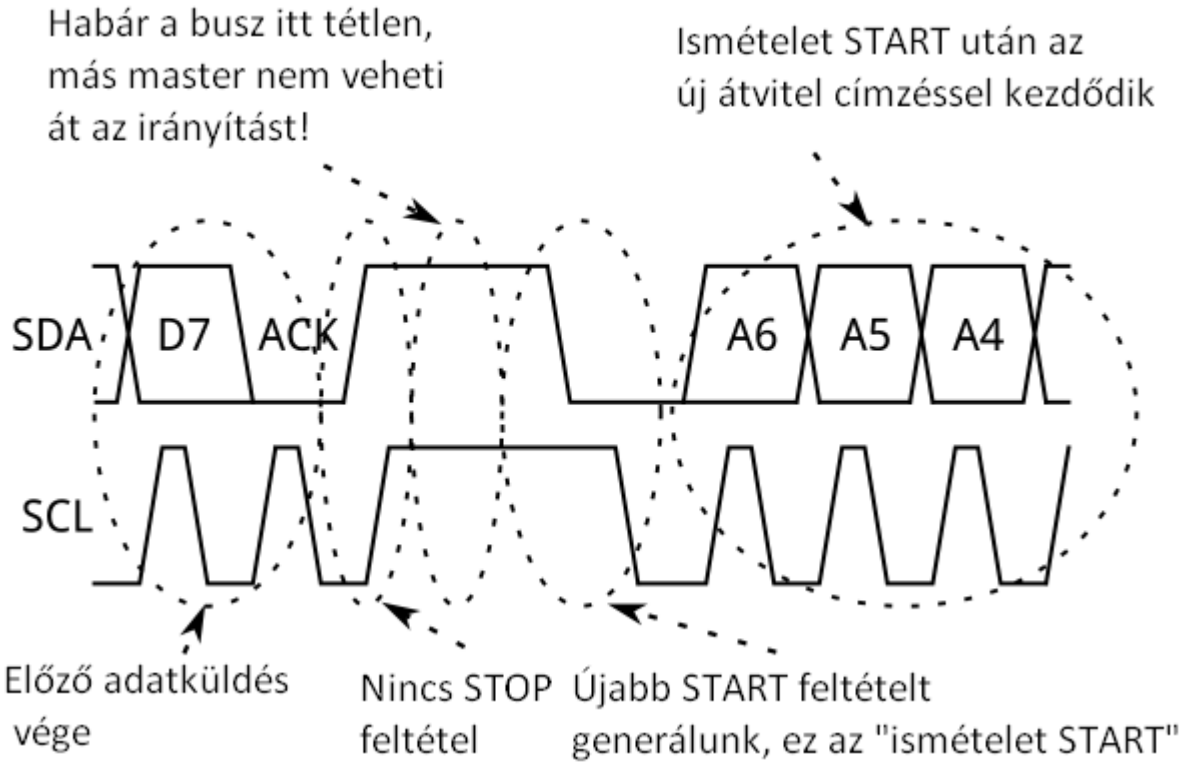
A slave ilyenkor a fogadás és a nyugtázás után alacsony szinten tarthatja az **SCL** vezetéket, hogy a mastert várakozási állapotba kényszerítse, amíg a slave kész nem lesz a következő bájt átvitelére, mint egy kézfogásos típusú eljárásban.

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.



A FRDM-KL25Z kártya I2C perifériái

Az **MKL25Z128VLK4** mikrovezérlő két **I2C** modult tartalmaz (I2C1 és I2C2) amelyek az alábbi tulajdonságokkal rendelkeznek:

- Multimaster környezetben is használhatók
- Az adatsebesség szoftveresen beállítható (64 fokozat választható)
- Szoftveresen választható nyugtázó bit küldés
- Programmegszakításos adatküldés/fogadás
- Arbitráció elvesztése esetén programmegszakítás és automatikus átváltás master módból slave módba
- Hívási cím egyezésekor programmegszakítás
- START és STOP feltétel generálás, illetve felismerés
- Ismételt START feltétel generálás, illetve felismerés
- Nyugtázó jel küldése, illetve felismerése
- Busz foglaltságának felismerése
- Általános hívás felismerése
- 10-bites kiterjesztett címzés lehetősége
- Kompatibilis a System Management Bus (SMBus) v2 specifikációjával
- Programozható bemeneti zavarszűrés
- Kisfogyasztású módból ébresztés slave címegyezés detektálásakor
- Slave címtartomány korlátozás támogatása
- DMA adatátvitel támogatása

A kivezetések konfigurálása

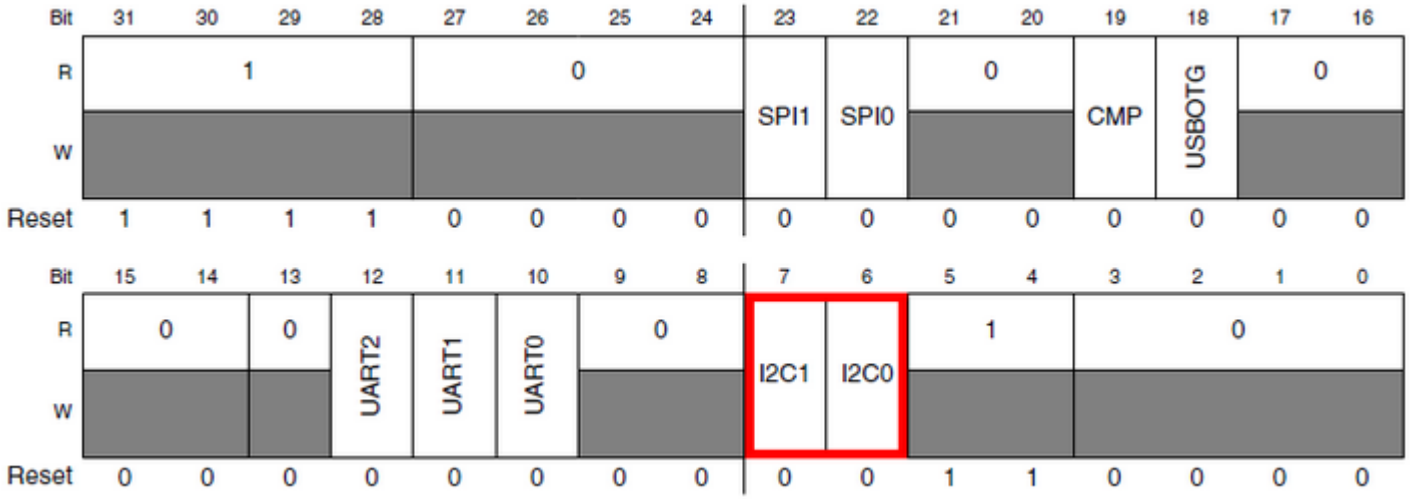
A konfigurálás egyik fontos eleme, hogy kivezetéseket rendelünk az adott perifériához. Az alábbi táblázatban összefoglaltuk az **I2C** modulokhoz rendelhető kivezetéseket. Zárójelben a megfelelő portvezérlő regiszter **MUX** bitcsoportjába írandó módválasztó kódot is megadtuk. Itt jegyezzük meg, hogy a Mazidi könyv 9-8. táblázatában óriási a kavarodás, hibás adatok szerepelnek benne!

I2C modul	I2Cx_SDA	I2Cx_SDA
I2C0	PTE25(5), PTB1(2), PTB3(2), PTC9(2)	PTE24(5), PTB0(2), PTB2(2), PTC8(2)
I2C1	**PTE0(6),** PTA4(2), PTC2(2), PTC11(2)	**PTE1(6),** PTC1(2), PTC10(2)

Megjegyzések:

- Az Arduino kompatibilis bekötéshez a vastagon szedett kivezetések tartoznak: **I2C1_SDA**=**PTE0, I2C1_SCL=PTE1**. A másik Arduino kompatibilis kivezetés, az **A4** (SDA) és **A5** (SCL) analóg bemenetek használata, **PTC2, PTC1** kivezetéspárnak felel meg.
- A **PTE25, PTE24** kivezetéspár csak a **FRDM-KL25Z** kártyára szerelt **MMA8451Q** gyorsulásmérő IC számára elérhető, a csatlakozókra nincsenek kivezetve.
- Az **I2C1_SCL** jel elvileg a **PTA3** kivezetéshez is hozzárendelhető lenne, de ez hardverütközéshez vezetne, mivel a **PTA3** láb egyúttal a programozó és nyomkövető **SWDIO** kivezetése, s a kártyára épített **Open_SDA** hibavadász eszközhöz csatlakozik..

Az I2C modulok engedélyezése



Az **I2C** modulok működését a többi soros perifériához hasonlóan a Rendszer-Integrációs Modul **SIM_SCGC4** regiszterében lehet engedélyezni, a megfelelő bit '1'-be állításával.

Például:

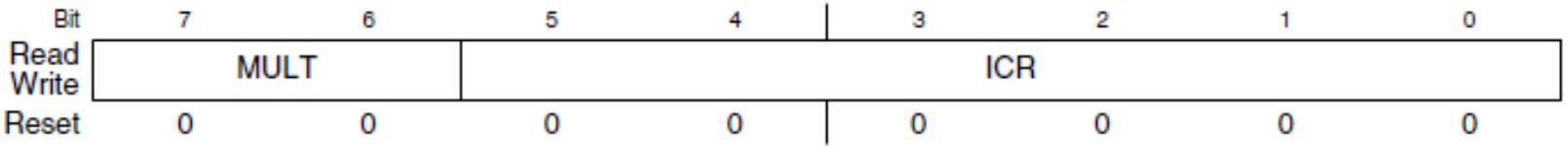
```
SIM->SCGC4 |= 0x40;                    // I2C0 engedélyezése
SIM->SCGC4 |= 0x80;                    // I2C1 engedélyezése
```

Az I2C modulok regiszterkészlete

Az **I2C*x*** modulok azonos regiszterkészlettel rendelkeznek.
Az **I2C0** modul báziscíme **0x4006_6000,** az **I2C1** modulé **0x4006 7000.**
Az alábbi táblázatban a báziscímekhez viszonyított eltolási címét (ofszet cím) adtuk meg.

Ofszet cím	Regiszter neve, funkciója	Méret	Elérés	Reset
0x0000	**I2C*x*_A1** address register 1. (1. cím regiszter)	8 bit	R/W	0x00
0x0001	**I2C*x*_F** frequency divider register(frekvencia osztó reg.)	8 bit	R/W	0x00
0x0002	**I2C*x*_C1** control register 1. (1. vezérlő regiszter)	8 bit	R/W	0x00
0x0003	**I2C*x*_S** status register (állapotjelző regiszter)	8 bit	R/W	0x80
0x0004	**I2C*x*_D** data register (adatregiszter)	8 bit	R/W	0x00
0x0005	**I2C*x*_C2** control register 2. (2. vezérlő regiszter)	8 bit	R/W	0x00
0x0006	**I2C*x*_FLT** Input Glicth filter (bemeneti zavarszűrő reg.)	8 bit	R/W	0x00
0x0007	**I2C*x*_RA** range address register (tartomány cím regiszter)	8 bit	R/W	0x00
0x0008	**I2C*x*_SMB** SMB bus control and status register (SMB busz vezérlő és állapotjelző regiszter)	8 bit	R/W	0x00
0x0009	**I2C*x*_A2** address register 2. (SMB busz cím regiszter)	8 bit	R/W	0xC2
0x000A	**I2C*x*_SLTH** SCL Low timeout high register (SCL lehúzás időtúllépés időzítés magas helyiértékű bitjei)	8 bit	R/W	0x00
0x000B	**I2C*x*_SLTL** SCL Low timeout low register (SCL lehúzás időtúllépés időzítés alacsony helyiértékű bitjei)	8 bit	R/W	0x00

Az adatsebesség beállítása



Az **I2C** kommunikáció adatátviteli sebességét a master határozza meg (az **SCL** órajelet a master állítja elő). Az **MKL25Z128VLK4** mikrovezérlő **I2C** moduljai órajelének forrása a buszfrekvencia, amelyből leosztással állíthatjuk elő a kívánt frekvenciájú **SCL** órajelet.

A leosztási arányt az **I2Cx_F** regiszterben adhatjuk meg, egy előosztási és egy osztási arány kiválasztásával, az alábbi képlet szerint:

$$\text{I2C baud rate} = f_{bus} / (2^{MULT} * \text{SCL osztó})$$

I2Cx_F: Frequencia osztó regiszter

- **MULT:** ez a bitcsoport az előosztási arányt szabja meg (**00:** 1/1, **01:** 1/2, **10:** /1/4, **11:** fenntartott kód)
- **ICR:** leosztási arány

Az **SCL** osztó és az **ICR** bitcsort értéke közötti (bonyolult) összefüggést a következő oldalon egy táblázatban mutatatjuk meg.

Leosztási arányok az ICR érték függvényében

ICR (Hex)	SCL osztó	ICR (Hex)	SCL osztó	ICR (Hex)	SCL osztó	ICR (Hex)	SCL osztó
0	20	10	48	20	160	30	640
1	22	11	56	21	192	31	768
2	24	12	64	22	224	32	896
3	26	13	72	23	256	33	1024
4	28	14	80	24	288	34	1152
5	30	15	88	25	320	35	1280
6	34	16	104	26	384	36	1536
7	40	17	128	27	480	37	1920
8	28	18	80	28	320	38	1280
9	32	19	96	29	384	39	1536
A	36	1A	112	2A	448	3A	1792
B	40	1B	128	2B	512	3B	2048
C	44	1C	144	2C	576	3C	2304
D	48	1D	160	2D	640	3D	2560
E	56	1E	192	2E	768	3E	3072
F	68	1F	240	2F	960	3F	3840

Példa: 24 MHz-es buszfrekvencia esetén a 400 kHz-es I2C bitráta beállításához 60-szoros leosztás kell. Ez a fenti táblázat alapján **MULT = 1** és **SCR = 0x05** értékekkel, végeredményben tehát az **I2Cx_F = 0x45;** értékadással állítható be.

I2Cx_C1 vezérlő regiszter

Az **I2Cx_C1** vezérlő regiszter központi szerepet játszik az **I2C** modul kezelésében, hiszen ebben tilthatjuk le, vagy engedélyezhetjük a modul működését, választhatjuk ki a master/slave módot, illetve állíthatjuk be az adatáramlás irányát (adatküldés vagy -fogadás).

Bit	7	6	5	4	3	2	1	0
Read	IICEN	IICIE	MST	TX	TXAK	0	WUEN	DMAEN
Write						RSTA		
Reset	0	0	0	0	0	0	0	0

Az egyes bitek szerepe:

IICEN - Az I2C modul engedélyezése (**0:** letiltás, **1:** engedélyezés)

IICIE - Az I2C modul megszakításainak engedélyezése (**0:** tiltás, **1:** engedélyezés)

MST - Master mód kiválasztása (**0:** slave mód, **1:** master mód). A 0➔1 átmenet automatikusan Start feltételt, a 1➔0 átmenet pedig Stop feltételt generál.

TX - Adatküldés mód választás (**0:** adatfogadás mód, **1:** adatküldés mód)

TXAK - Nyugtázó bit választása (**0:** ACK pozitív nyugtázás küldése, **1:** NAK negatív nyugtásá küldése)

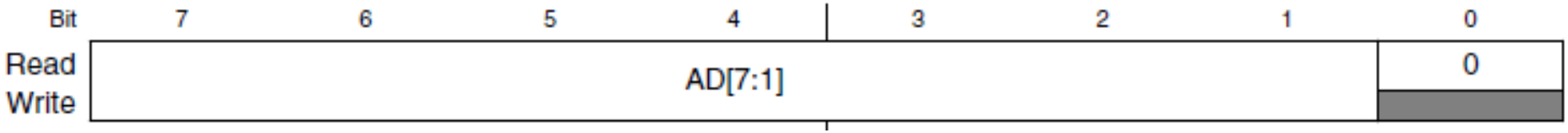
RSTA - Újraindítás (ismételt START) küldése (1: írása új START feltételt indít)

WUEN - Felébresztés engedélyezése energiatakarékos módban (**0:** tiltás, **1:** engedélyezés)

DMAEN - DMA adatátvitel engedélyezése (**0:** tiltás, **1:** engedélyezés)

Az I2Cx_A1 címregiszter

Slave módban ez a regiszter tartalmazza a 7-bites címzéshez az eszköz címét (10-bites címzéshez a kiegészítő címbiteket az **I2Cx_C2** regiszter alsó három bitje tartalmazza).



Az I2Cx_D adatregiszter

8-bites adatregiszter

Mester adatküldés módban:

Az **I2Cx_D** regiszterbe írás elindítja az adatküldést

Mester adatfogadás módban

Az **I2Cx_D** regiszter olvasása elindítja a következő adatbájt vételét

Az I2Cx_S állapotjelző regiszter

Az **I2Cx_S** állapotjelző regiszter bitjei az I2C modul, mint állapotgép pillanatnyi állapotáról értesít bennünket.

Bit	7	6	5	4	3	2	1	0
Read	TCF	IAAS	BUSY	ARBL	RAM	SRW	IICIF	RXAK
Write				w1c			w1c	
Reset	1	0	0	0	0	0	0	0

TCF - Adatküldés vége jelzőbit (**0:** adatküldés folyamatban, **1:** adatküldés befejeződött)
A jelzőbit az adatregiszter írásával (vételi módban az olvasásával) törlődik.

IAAS - "Slave eszközként megszólítva" állapotjelző (**0:** nincs megszólítva a mikrovezérlő, **1:** egy külső eszköz slave módban megszólította a mikrovezérlőt)

BUSY - Busz foglaltság jelzése (**0:** az I2C busz tétlen, **1:** az I2C busz foglalt)

ARBL - Arbitráció vesztés (**0:** normál busz művelet, **1:** arbitráció vesztés történt).
Törlése: '1' beírásával.

RAM - Címtartomány egyezés (**0:** nincs megszólítva, **1:** slave-ként megszólítva)

SRW - Slave írás/olvasás vezérlés (**0:** a mikrovezérlő adatot kell, hogy fogadjon, **1:** a mikrovezérlőtől adatot kérnek)

IICIF - megszakításjelző bit (**0:** nincs megszakítási kérelem, **1:** függőben levő megszakítás)

RXAK - Nyugtázás vétele (**0:** nyugtázás (ACK) érkezett, **1:** negatív nyugtázás (NAK) érkezett)

Az adatküldés folyamata

- Egy **START** feltétel után az **I2C** busz foglalt (**BUSY**) állapotba kerül, amíg egy **STOP** feltétel nem érkezik.

- A busz foglalt állapotában csak akkor küldhetünk adatokat, ha mi generáltuk a **START** feltételt, és a slave címének kiküldése az arbitráció elvesztése nélkül sikerült. Ezért fontos a **BUSY** jelzőbit vizsgálata, mielőtt **START** feltétel generálásával új tranzakciót indítanánk.

- Hasonlóan fontos az **ARBL** bit vizsgálata a slave címének kiküldése után, mielőtt további adatokat küldenénk.

  Az adatregiszter feltöltése után az adatátvitel elindul és az **I2Cx_S** regiszter **TCF** jelzőbitje '0'-ra vált és ebben az állapotban marad, amíg az adatbájt küldése be nem fejeződik.

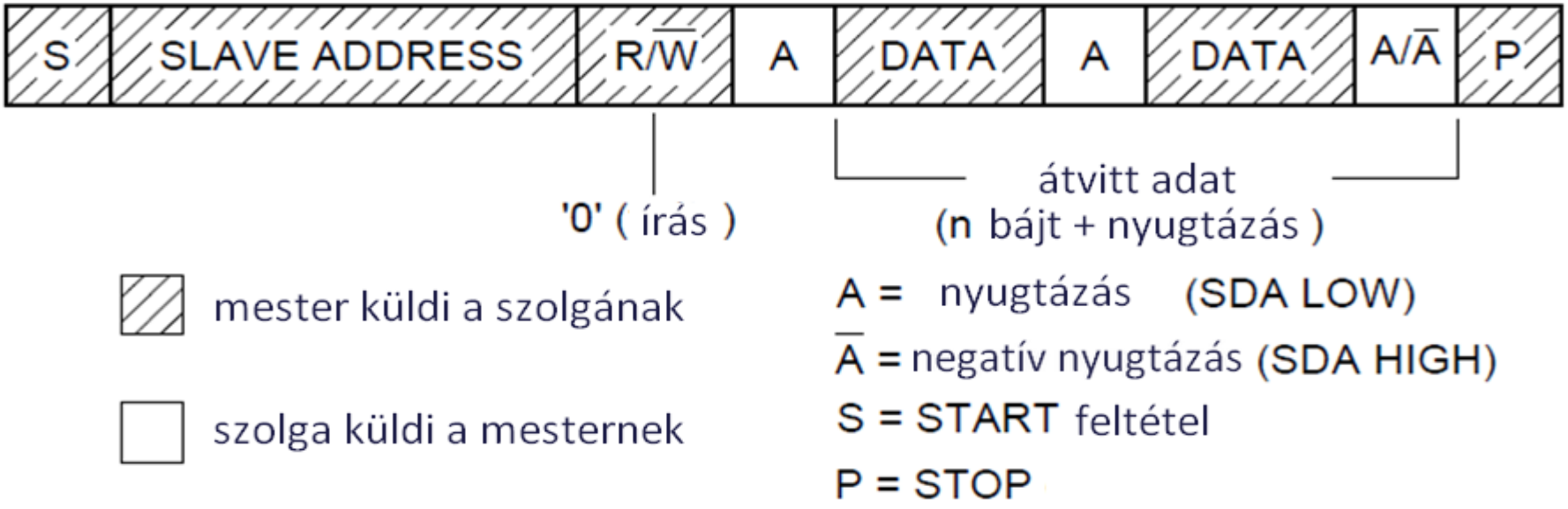
- A program az **I2Cx_S** regiszter **IICIF** jelzőbitjének vizsgálatával értesül róla, hogy az adatküldés folyamata mikor ért véget. Ekkor az **I2Cx_S** regiszter **RXAK** jelzőbitjének vizsgálatával értesülhetünk róla, hogy a slave eszköz nyugtázta-e az adat vételét.

- Ha az adatküldés végén az **RXAK** bit '1'-be áll, akkor hiba történt (vagy negatív nyugtázás érkezett).

Az I2C modul konfigurálása

A GPIO kivezetések **I2C** módba történő konfigurálása után az alábbi lépéseket követve konfigurálhatjuk az **I2C** modult, hogy adatot küldjön egy slave eszköznek:

1. Engedélyezzük az **I2C** modul órajelét a **SIM_SCGC4** regiszter tartalmának módosításával!
2. Tiltsuk le az **I2C** modult a konfigurálás tartamára az **I2Cx_C1** regiszter nullázásával!
3. Állítsuk be a kívánt adatátviteli sebességhez tartozó osztási arányt az **I2Cx_F** regiszterben!
4. Engedélyezzük az **I2C** modult az **I2Cx_C1** regiszter **IICEN** bitjének '1'-be állításával!

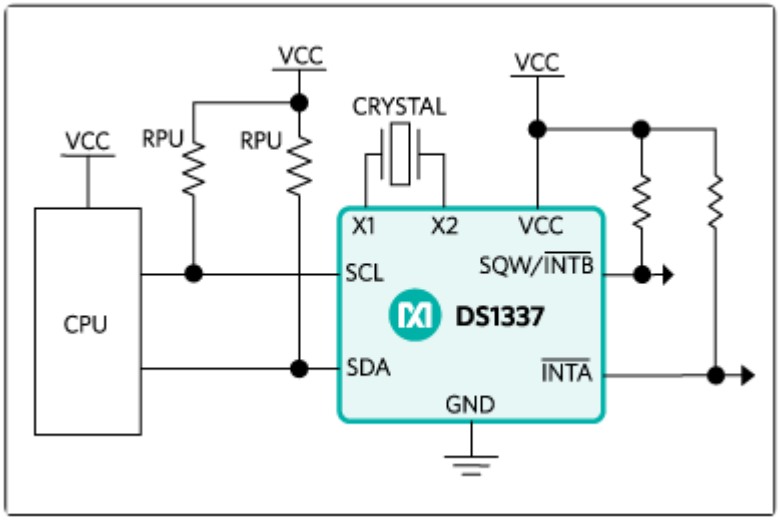


Egy adatbájt küldése a slave eszköznek

1. Az **I2Cx_S** regiszter **BUSY** bitjének lekérdezésével várakozzunk, amíg az **I2C** busz szabaddá válik!
2. Az **I2Cx_C1** regiszter **TX** bitjét állítsuk '1'-be az adatküldéshez!
3. Az **I2Cx_C1** regiszter **MST** bitjét állítsuk '1'-be a master mód beállításához és a **START** feltétel generálásához!
4. Írjuk a slave 7-bites címét és az adatküldést jelző 0 bitet az **I2Cx_D** adatregiszterbe!
5. Az **I2Cx_S** regiszter **IICIF** bitjének lekérdezésével várakozzunk, amíg az átvitel le nem zajlik.
6. Töröljük az **IICIF** bitet '1' beírásával!
7. Vizsgáljuk meg az **I2Cx_S** regiszter **ARBL** bitjét, hogy nem vesztettük-e el az arbitrációt! Ha igen, akkor töröljük az **ARBL** bitet '1' beírásával és szakítsuk félbe a tranzakciót!
8. Ellenőrizzük az **I2Cx_S** regiszter **RXAK** bitjét, hogy a slave küldött-e pozitív nyugtázást! Ha nem, akkor szakítsuk félbe a tranzakciót!
9. Írjuk a kiküldeni kívánt adatot az **I2Cx_D** adatregiszterbe!
10. Az **I2Cx_S** regiszter **IICIF** bitjének lekérdezésével várakozzunk, amíg az átvitel le nem zajlik.
11. Töröljük az **IICIF** bitet '1' beírásával!
12. Ellenőrizzük az **I2Cx_S** regiszter **RXAK** bitjét, hogy a slave küldött-e pozitív nyugtázást! Ha nem, akkor szakítsuk félbe a tranzakciót!
13. Töröljük az **I2Cx_C1** regiszter **TX** és **MST** bitjeit a STOP feltétel generálásához!

Program9_1: Adatküldés az I2C buszon

A programban egy **DS1337** (vagy **DS3231)** Real-time órában az időt és a dátumot (az RTC elsö hét regiszterét) bájtonként külön-külön tranzakcióval töltjük fel, előre definiált értékkel.



RTC modul	FRDM-KL25Z kártya
GND	GND
VCC	3,3 V
SDA	PTE0 (I2C1_SDA)
SCL	PTE1 (I2C1_SCL)

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

```
#include "MKL25Z4.h"

#define SLAVE_ADDR 0x68                 // 1101 000
#define ERR_NONE 0                      // Hibátlan, nyugtázott átvitel
#define ERR_NO_ACK 0x01                 // Negatív nyugtázás
#define ERR_ARB_LOST 0x02               // Arbitráció vesztés
#define ERR_BUS_BUSY 0x03               // Busz foglaltság

void I2C1_init(void);
int I2C1_byteWrite(unsigned char slaveAddr, unsigned char memAddr,
unsigned char data);
void delayUs(int n);

int main(void) {
    unsigned char timeDateToSet[] = {0x55, 0x58, 0x16, 0x01, 0x19, 0x10, 0x09};
    int rv;
    int i;

    I2C1_init();
    for (i = 0; i < 7; i++) {
        rv = I2C1_byteWrite(SLAVE_ADDR, i, timeDateToSet[i]);
        if (rv != ERR_NONE)
            for(;;) ;                   // Hiba kezelés jöhet ide!
    }
    for (;;)      {
    }
}
```

2009. Október 19. 16:58:55

```
//---------------------------------------
// I2C1 és a kivezetések inicializálása
//---------------------------------------
void I2C1_init(void) {
    SIM->SCGC4 |= 0x80;                 // I2C1 engedélyezése
    SIM->SCGC5 |= 0x2000;               // PORT E engedélyezése
    PORTE->PCR[1] = 0x0600;             // PTE1 legyen I2C1 SCL kivezetés
    PORTE->PCR[0] = 0x0600;             // PTE0 legyen I2C1 SDA kivezetés

    I2C1->C1 = 0;                       // I2C1 letiltása konfiguráláshoz
    I2C1->S = 2;                        // IICIF törlése
    I2C1->F = 0x1F;                     // 100 kHz bitráta választása
    I2C1->C1 = 0x80;                    // I2C1 engedélyezése
}


//---------------------------------------
// Késleltető függvény 48 MHz órajelhez
//---------------------------------------
void delayUs(int n) {
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 8; j++);
}
```

```
int I2C1_byteWrite(unsigned char slaveAddr, unsigned char memAddr, unsigned char data) {
    int retry = 1000;
    while (I2C1->S & 0x20) {            // Várunk, míg a busz szabaddá válik
      if (--retry <= 0) return ERR_BUS_BUSY; // Legfeljebb 1000-szer próbálkozunk
      delayUs(100);
    }
    I2C1->C1 |= 0x10;                   // TX beírása
    I2C1->C1 |= 0x20;                   // MST beírása, START generálása
    I2C1->D = slaveAddr << 1;           // Slave cím és W bit küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x10) {               // Arbitráció elvesztése esetén
        I2C1->S |= 0x10;                // ARBL bit törlése
        return ERR_ARB_LOST;            // Kilépés hibakóddal
    }
    if (I2C1->S & 0x01) return ERR_NO_ACK;  // Negatív nyugtázás
    I2C1->D = memAddr;                  // Regisztercím küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    I2C1->D = data;                     // Adat küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    I2C1->C1 &= ~0x30;                  // TX és MST törlése, STOP generálás
    return ERR_NONE;
}
```

Program9_2: Tömbösített adatküldés

Az előzőhez képest csak annyi a különbség, hogy a 7 db adatot tömbösítve küldjük ki.

```
int main(void) {
    unsigned char timeDateToSet[] = {0x00, 0x50, 0x18, 0x07, 0x18, 0x02, 0x17};
    int rv;
    I2C1_init();
    rv = I2C1_burstWrite(SLAVE_ADDR,0,7,timeDateToSet);
    if(rv != ERR_NONE) {
      for(;;);                          // Hibakezelés jöhet ide!
    }
    for (;;) {
    }
}

//---------------------------------------
// Tömbösített adatcsomag kiküldése a slave eszköznek
// Tranzakció: S-(scím+w)-ACK-rcím-ACK-adat-ACK-...-adat-ACK-P
//---------------------------------------
int I2C1_burstWrite(unsigned char slaveAddr, unsigned char memAddr,
                    int byteCount, unsigned char* data)  {
    int retry = 1000;
    while (I2C1->S & 0x20) {            // Várunk, míg a busz szabaddá válik
        if (--retry <= 0)               // Legfeljebb 1000-szer próbálkozunk
            return ERR_BUS_BUSY;
        delayUs(100);
    }
```

Itt van változás

```
    I2C1->C1 |= 0x10;                   // TX beírása
    I2C1->C1 |= 0x20;                   // MST beírása, START generálása
    I2C1->D = slaveAddr << 1;           // Slave cím és W bit küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x10) {               // Arbitráció elvesztése esetén
        I2C1->S |= 0x10;                // ARBL bit törlése
        return ERR_ARB_LOST;            // Kilépés hibakóddal
    }
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    I2C1->D = memAddr;                  // Regisztercím küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    while(byteCount-- > 0) {
        I2C1->D = *data++;              // Adat küldése
        while(!(I2C1->S & 0x02));       // Átvitel végére várunk
        I2C1->S |= 0x02;                // IICIF törlése
        if (I2C1->S & 0x01)             // Negatív nyugtázás esetén
            return ERR_NO_ACK;          // Kilépés hibakóddal
    }
    I2C1->C1 &= ~0x30;                  // TX és MST törlése, STOP generálás
    return ERR_NONE;
}
```

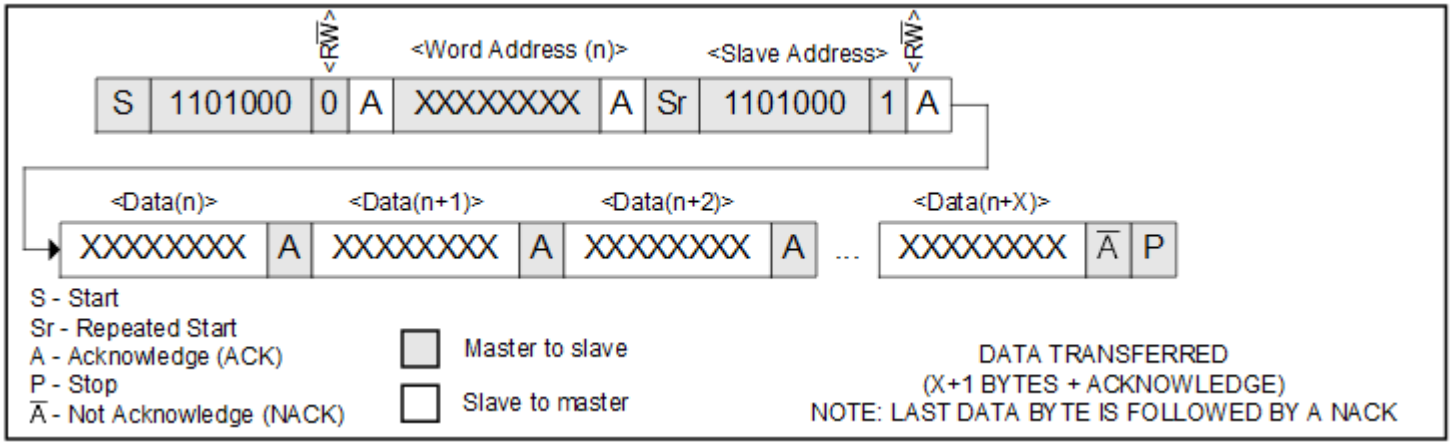
Itt van változás

Program9_3: tömbösített beolvasás

Az előző program fordítottja, amelyben egyetlen tranzakcióban olvassuk ki az RTC első hét regiszterét (a futó időt). Az ehhez definiált **I2C1_burstRead()** függvény paraméterei:

slaveAddr - a slave eszköz címe (0x68)
memAddr - a legelső adathoz tartozó regiszter sorszáma (esetünkben 0)
byteCount - a beolvasandó adatok darabszáma (esetünkben 7)
data - mutató az adatok eltárolására kijelölt adattáblázathoz (esetünkben a **timeDateToSet[]** tömb kezdőcíme)

Az adatbeolvasás komplikáltabb, mint az adatküldés. Az olvasást is adatküldéssel kell kezdeni: (slave címét, s meg kell adni azt a címet, ahonnan a kiolvasást kezdeni akarjuk). Ezután adatáramlási irányt kell váltani - **RESTART** feltétel generálásával, majd újra ki kell küldeni a slave eszköz címét, de az R/W biten most az olvasásnak megfelelő '1' áll. Ezután következhet az adatbájtok beolvasása, melyeknek beérkezését az utolsó bájt kivételével pozitív nyugtázással jeleznünk kell.



Program9_3 (részletek)

```
#include "MKL25Z4.h"

#define SLAVE_ADDR 0x68                 // 1101 000
#define ERR_NONE 0                      // Hibátlan, nyugtázott átvitel
#define ERR_NO_ACK 0x01                 // Negatív nyugtázás
#define ERR_ARB_LOST 0x02               // Arbitráció vesztés
#define ERR_BUS_BUSY 0x03               // Busz foglaltság

void I2C1_init(void);
int I2C1_burstRead(unsigned char slaveAddr, unsigned char memAddr,
                   int byteCount, unsigned char* data);
void delayUs(int n);

int main(void) {
    unsigned char timeDateReadBack[7];
    int rv;

    I2C1_init();
    rv = I2C1_burstRead(SLAVE_ADDR,0,7,timeDateReadBack);
    if(rv != ERR_NONE) {
      for(;;);                          // Hibakezelés jöhet ide!
    }
    for (;;) {
    }
}
```

```
int I2C1_burstRead(unsigned char slaveAddr, unsigned char memAddr,
                   int byteCount, unsigned char* data)  {
    int retry = 1000;
    volatile unsigned char dummy;
    while (I2C1->S & 0x20) {            // Várunk, míg a busz szabaddá válik
        if (--retry <= 0)               // Legfeljebb 1000-szer próbálkozunk
            return ERR_BUS_BUSY;
        delayUs(100);
    }

    I2C1->C1 |= 0x10;                   // TX beírása
    I2C1->C1 |= 0x20;                   // MST beírása, START generálása
    I2C1->D = slaveAddr << 1;           // Slave cím és W bit küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x10) {               // Arbitráció elvesztése esetén
        I2C1->S |= 0x10;                // ARBL bit törlése
        return ERR_ARB_LOST;            // Kilépés hibakóddal
    }
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal

    I2C1->D = memAddr;                  // Regisztercím küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
```

Folytatás az előző oldalról:

```
    I2C1->C1 |= 0x04;                   // RESTART feltétel generálás
    I2C1->D = (slaveAddr<<1) | 1;       // Slave cím és R bit küldése
    while(!(I2C1->S & 0x02));           // Átvitel végére várunk
    I2C1->S |= 0x02;                    // IICIF törlése
    if (I2C1->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal

    I2C1->C1 &= ~0x18;                  // TX bit törlése, ACK küldés előkészítése
    if (byteCount == 1)
        I2C1->C1 |= 0x08;               // NAK küldés előkészítése
    dummy = I2C1->D;                    // kamu olvasás, és új ciklus indítása

    while(byteCount > 0) {
        if (byteCount == 1)
            I2C1->C1 |= 0x08;           // NAK küldés előkészítése
        while(!(I2C1->S & 0x02));       // Átvitel végére várunk
        I2C1->S |= 0x02;                // IICIF törlése
        if (byteCount == 1) {
            I2C1->C1 &= ~0x20;          // STOP feltétel generálása
        }
        *data++ = I2C1->D;              // Vett adat kiolvasása
        byteCount--;
    }
    return ERR_NONE;
}
```

Program9_3 futási eredmény

Ez a program sem tartalmaz kiíratást, így csak kiegészítő eszközökkel (pl. **Bus Pirate, Arduino** ArduPirate firmware-rel, esetleg a **program9_3** mintapélda futtatása **Keil MDK5** környezetben, nyomkövető módban vizsgálva) tudjuk ellenőrizni a programfutás eredményét.

Watch 1

Name	Value	Type	
timeDateReadBack	0x1FFFF15...	unsigned char[7]	
[0]	0x02	unsigned char	másodperc: 2
[1]	0x19	unsigned char	perc: 19
[2]	0x20	unsigned char	óra: 20
[3]	0x01	unsigned char	hét napja: 1
[4]	0x19	unsigned char	nap: 19
[5]	0x10	unsigned char	hónap: Október
[6]	0x09	unsigned char	év: 2009
<Enter expression>			

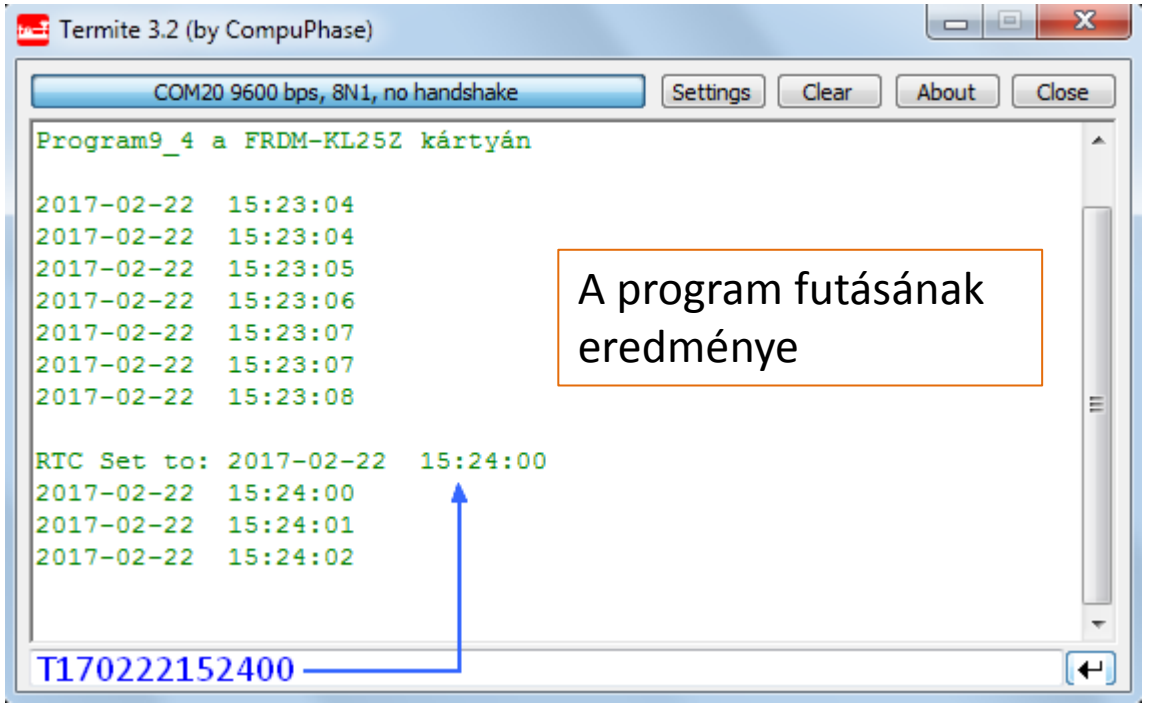
Program9_4: idő kiírása UART-on

Az előző programban bemutatott **I2C1_burstRead()** függvény segítségével kiolvassuk egy **DS1337** (vagy vele kompatibilis) Real Time órát, s az Aszinkron soros kommunikáció (UART) fejezetben bemutatott **stdio** átirányítással az **UART0** porton formázottan kiíratjuk az aktuális idő és dátumot. A programot egy órabeállítási lehetőséggel is kiegészítettük: a soros porton érkező, TYYMMDDHHMMSS formátumú paranccsal végezhetjük el a beállítást.

A kiíratásokhoz a kiolvasott adatokat binárisan kódolt decimális (BCD) ábrázolásból vissza kell alakítanunk "normális" számmá. Ehhez a program elején definiáltunk egy **bcdToDec()** függvényt.

Megjegyzés:

Az Aszinkron soros kommunikáció (UART) fejezetben bemutatott soros port kezelést átalakítottuk, programmegszakításásos, bufferelt adatküldést használunk mindkét irányba.



```c
#include <MKL25Z4.H>
#include <stdio.h>
#include "uart.h"
int main(void) {
  int hours=0,minutes=0,seconds=0;
  int rv, year=0, month=0, dayOfMonth=0;
  uint8_t tbuf[7];
  SystemCoreClockUpdate();
  UART_config(9600);
  __enable_irq(); // Megszakítások engedélyezése
  printf("\r\nProgram9_4 a FRDM-KL25Z kártyán\r\n");
  I2C1_init();
  while(1){
    if(UART_available()) processSerialCommand();
    rv = I2C1_burstRead(SLAVE_ADDR,0,7,tbuf);
    if(rv != ERR_NONE) {
      for(;;);                          // Hibakezelés jöhet ide!
    }
    seconds = bcdToDec(tbuf[0]) &0x7f;
    minutes = bcdToDec(tbuf[1]);
    hours = bcdToDec(tbuf[2]) & 0x3f;
    dayOfMonth = bcdToDec(tbuf[4]);
    month = bcdToDec(tbuf[5]);
    year = bcdToDec(tbuf[6]);
    printf("%d-%02d-%02d ",year+2000,month,dayOfMonth);
    printf(" %02d:%02d:%02d\n",hours,minutes,seconds);
    delayUs(1000000);
  }
}
```

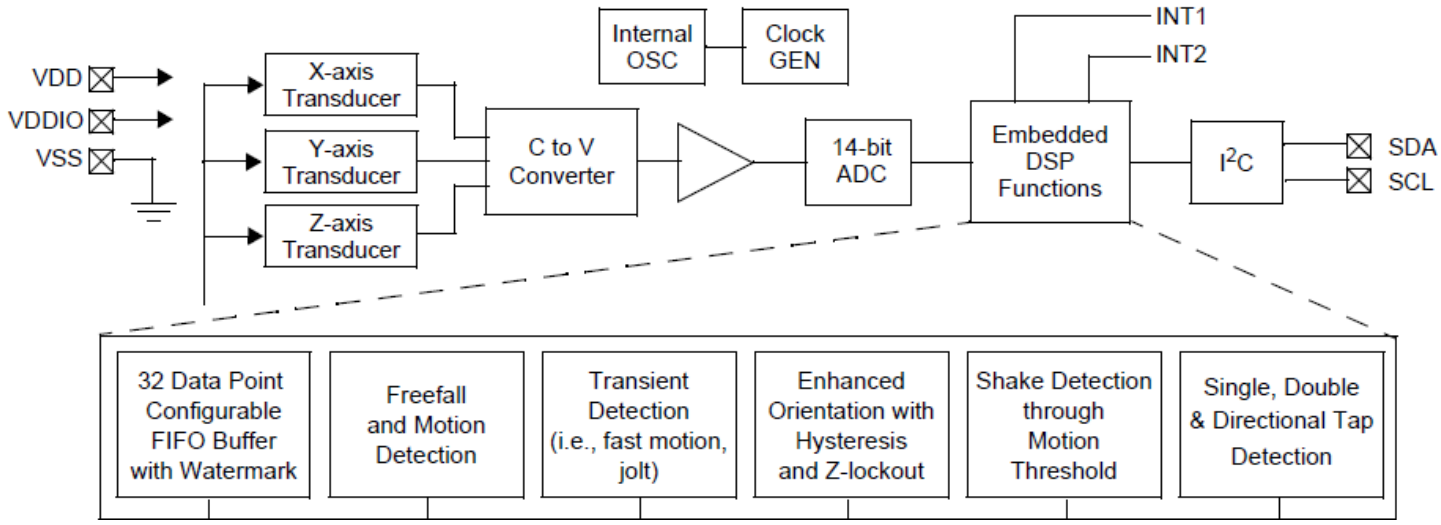
```c
//-- Convert decimal numbers to BCD --
uint8_t decToBcd(uint8_t val) {
  return( (val/10*16) + (val%10) );
}

//-- Convert BCD numbers to decimal  --
int bcdToDec(uint8_t val) {
  return (int)((val/16*10) + (val%16));
}
```

Soros porti parancsok feldolgozása
RTC beállítás parancs formátuma: TYYMMDDHHMMSS
Például: 2012 Oct 21 1:23pm → T121021132300

```
void processSerialCommand() {
  uint8_t y, m, d, hh, mm, ss, rv;
  uint8_t sbuf[7];
  char c = UART_getc();
  switch(c){
  case 'T':
    delayUs(100000);                    // Wait for all data to arrive
    if( UART_available() >= 12 ){       // process only if all expected data is available
    // Parse incomming 12 ASCII charaters into y,m,d,hh,mm,ss
    // no error checking for numeric values in YYMDDHHMMSS fields, so be carefull!
      c = UART_getc();        y = c - '0';
      c = UART_getc();        y = 10*y; y += c-'0';
      c = UART_getc();        m = c - '0';
      c = UART_getc();        m = 10*m; m += c-'0';
      c = UART_getc();        d = c - '0';
      c = UART_getc();        d = 10*d; d += c-'0';
      c = UART_getc();        hh = c - '0';
      c = UART_getc();        hh = 10*hh; hh += c-'0';
      c = UART_getc();        mm = c - '0';
      c = UART_getc();        mm = 10*mm; mm += c-'0';
      c = UART_getc();        ss = c - '0';
      c = UART_getc();        ss = 10*ss; ss += c-'0';
      printf("\nRTC Set to: ");
      sbuf[0] = decToBcd(ss); sbuf[1] = decToBcd(mm); sbuf[2] = decToBcd(hh);
      sbuf[3] = 0;  sbuf[4] = decToBcd(d); sbuf[5] = decToBcd(m); sbuf[6] = decToBcd(y);
      rv = I2C1_burstWrite(SLAVE_ADDR,0,7,sbuf);
      if(rv != ERR_NONE) {
        for(;;);                        // Hibakezelés jöhet ide!
      }
      while(UART_available()) UART_getc(); // clear serial buffer
    }
    break;
  }
}
```

Program9_5: 3-tengelyű gyorsulásmérő

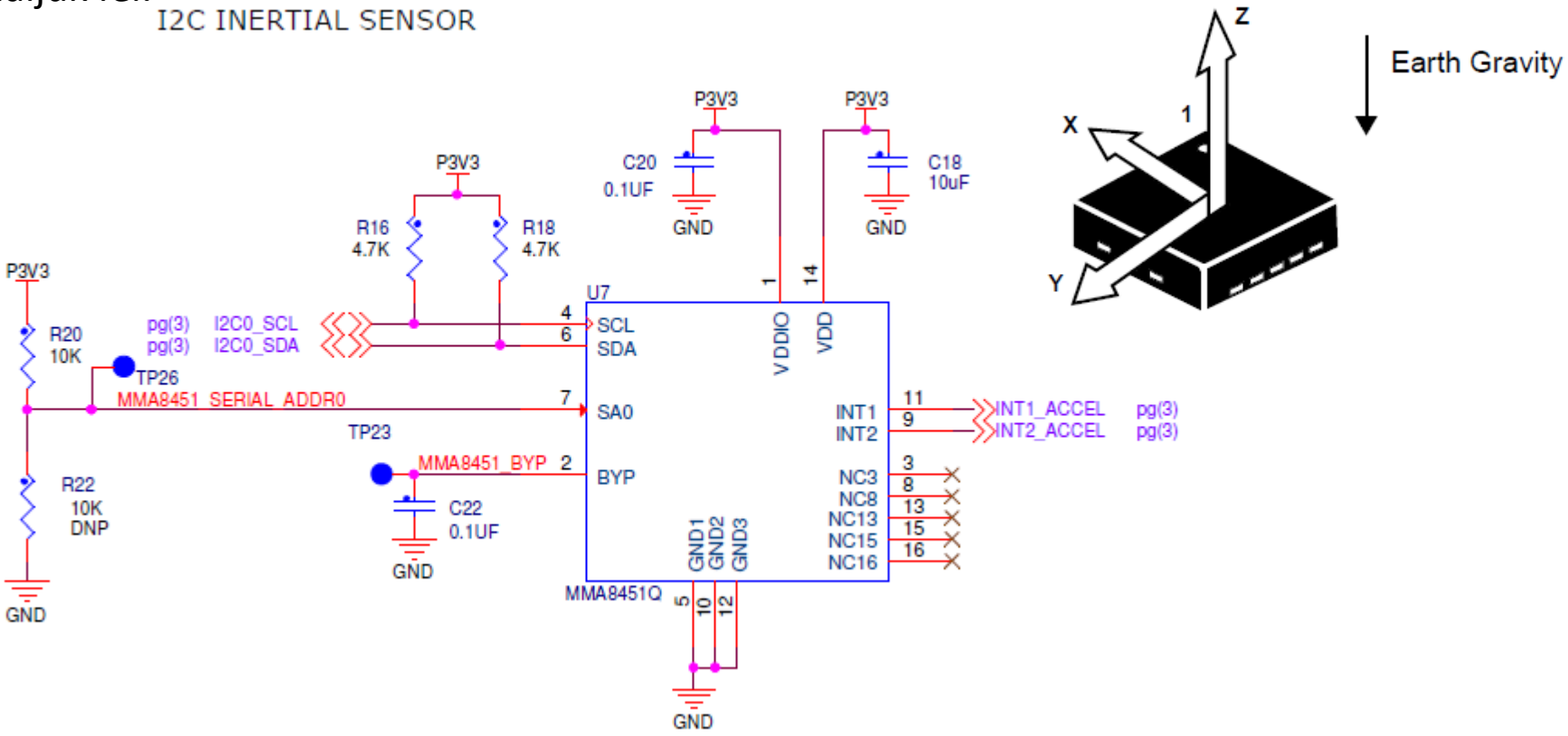


- Freescale **MMA8451** van ráépítve a **FRDM-KL25Z** kártyára
- A gyorsulást méri x/y/z irányba, ±2g, ± 4g, vagy ± 8g méréshatárral
- Ki tudja számítani az x/y tengelyek körüli elfordulást (pitch, roll)
- I2C cím: 0x1D

Program9_5: 3-tengelyű gyorsulásmérő

Az **MMA8451Q** gyorsulásmérő az **I2C0** modulhoz kapcsolódik a **PTE24** (SCL) és a **PTE25** (SDA) kivezetéseken keresztül. Mindkét kivezetésnél a megfelelő portvezérlő regiszter **MUX** bitcsoportjában az **Alt5** módot kell választani az **I2C0** funkció eléréséhez.

Ebben a programban a 14-bites X, Y, Z adatokból csak a legfelső 8 bitet olvassuk ki és használjuk fel.



Program9_5: 3-tengelyű gyorsulásmérő

Az **MMA8451Q** gyorsulásmérő bonyolult működésének és felépítésének részleteire itt nem kívánunk kitérni, ehelyett az MMA8451Q adatlap önálló tanulmányozására buzdítjuk az olvasót. Az **MMA8451Q** gyorsulásmérő IC számtalan regisztere közül mi most csak az alábbiakkal foglalkozunk:

```
#define REG_WHO_AM_I 0x0D    // Eszköz azonosító, gyárilag 0x1A-ra állítva
#define REG_CTRL_REG_1 0x2A  // Vezérlő regiszter. Legalsó bitjét 1-re kell állítanunk
#define REG_OUT_X_MSB 0x01   // A mért gyorsulás X komponense (legfelső 8 bit)
#define REG_OUT_Y_MSB 0x03   // A mért gyorsulás Y komponense (legfelső 8 bit)
#define REG_OUT_Z_MSB 0x05   // A mért gyorsulás Z komponense (legfelső 8 bit)
```

- A "Who am I" regiszter kiolvasásával az eszköz jelenlétét és típusát ellenőrizhetjük (0x1A olvasható ki)
- A vezérlő regiszter legalsó bitjét 1-be kell állítani a működéshez (felébresztés standby módból).
- Az Out_X_MSB, Out_X_MSB és Out_Z_MSB regiszterekből a gyorsulás három dimenziós vektorának komponenseit (pontosabban azok legfelső 8 bitjét) olvashatjuk ki, előjeles formában.

```
#include <MKL25Z4.H>
#include <stdio.h>
#include "uart.h„
// System runs at 48MHz
// Baud rate 9600, 8 bit data, no parity, 1 stop bit
#define SLAVE_ADDR 0x1D                 // 0011101
#define ERR_NONE 0                      // Hibátlan, nyugtázott átvitel
#define ERR_NO_ACK 0x01                 // Negatív nyugtázás
#define ERR_ARB_LOST 0x02               // Arbitráció vesztés
#define ERR_BUS_BUSY 0x03               // Busz foglaltság

#define REG_WHO_AM_I      0x0D
#define REG_CTRL_REG_1    0x2A
#define REG_OUT_X_MSB     0x01
#define REG_OUT_Y_MSB     0x03
#define REG_OUT_Z_MSB     0x05

void acc_init(void);
int acc_write(uint8_t regAddr, uint8_t data);
int acc_read(uint8_t regAddr, uint8_t* data);

//---------------------------------------
// Késlelteto függvény 48 MHz órajelhez
//---------------------------------------
void delayUs(int n) {
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 8; j++);
}
```

```
int main(void) {
  int rv,xx,yy,zz;
  uint8_t id,x,y,z;

  SystemCoreClockUpdate();
  UART_config(9600);
  acc_init();
  rv = acc_read(REG_WHO_AM_I, &id);
  printf("Device ID: %02x\n",id);
  printf("\r\nWelcome to FRDM-KL25Z board!\r\n");
  while(1){
    rv = acc_read(REG_OUT_X_MSB, &x);
    rv += acc_read(REG_OUT_Y_MSB, &y);
    rv += acc_read(REG_OUT_Z_MSB, &z);
    if(rv) {
      printf("*** Hiba az I2C0 buszon!\n");
    }
    else {
      xx = (x>127)? x - 256 : x;
      yy = (y>127)? y - 256 : y;
      zz = (z>127)? z - 256 : z;
      printf("position = %4d %4d %4d\n",xx,yy,zz);
    }
    delayUs(2000000);
  }
}
```

Megjegyzés:

Az Aszinkron soros kommunikáció (UART) fejezetben bemutatott soros port kezelést (uart_retarget) átalakítottuk, FIFO bufferelt és programmegszakításásos, adatküldést használunk mindkét irányba.

```
//-----------------------------------------------------
// I2C0, PTE24, PTE25 és az MMA8451 inicializálása
//-----------------------------------------------------
void acc_init(void) {
    SIM->SCGC4 |= 0x40;                 // I2C0 engedélyezése
    SIM->SCGC5 |= 0x2000;               // PORT E engedélyezése
    PORTE->PCR[24] = 0x0500;            // PTE1 legyen I2C0 SCL kivezetés
    PORTE->PCR[25] = 0x0500;            // PTE0 legyen I2C0 SDA kivezetés

    I2C0->C1 = 0;                       // I2C0 letiltása konfiguráláshoz
    I2C0->S = 2;                        // IICIF törlése
    I2C0->F = 0x1F;                     // 100 kHz bitráta választása
    I2C0->C1 = 0x80;                    // I2C0 engedélyezése
    acc_write(REG_CTRL_REG_1,0x01);     // MMA aktív módba kapcsolása
}
```

```
int acc_write(uint8_t regAddr, uint8_t data)  {
    int retry = 1000;
    volatile uint8_t dummy;
    while (I2C0->S & 0x20) {            // Várunk, míg a busz szabaddá válik
        if (--retry <= 0)               // Legfeljebb 1000-szer próbálkozunk
            return ERR_BUS_BUSY;
        delayUs(100);
    }
    I2C0->C1 |= 0x10;                   // TX beírása
    I2C0->C1 |= 0x20;                   // MST beírása, START generálása
    I2C0->D = SLAVE_ADDR << 1;          // Slave cím és W bit küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x10) {               // Arbitráció elvesztése esetén
        I2C0->S |= 0x10;                // ARBL bit törlése
        return ERR_ARB_LOST;            // Kilépés hibakóddal
    }
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    I2C0->D = regAddr;                  // Regisztercím küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal

    I2C0->D = data;                     // Adatbájt küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
    I2C0->C1 &= ~0x30;                  // TX és MST törlése, STOP generálás
    return ERR_NONE;
}
```

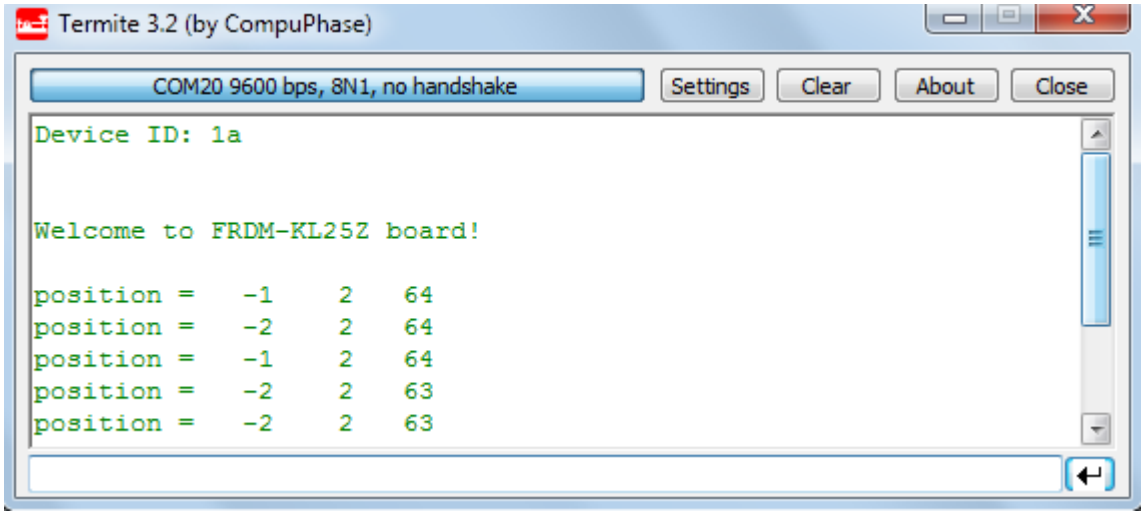
```
int acc_read(uint8_t regAddr, uint8_t* data)  {
    int retry = 1000;
    volatile uint8_t dummy;
    while (I2C0->S & 0x20) {            // Várunk, míg a busz szabaddá válik
        if (--retry <= 0)               // Legfeljebb 1000-szer próbálkozunk
            return ERR_BUS_BUSY;
        delayUs(100);
    }
    I2C0->C1 |= 0x10;                   // TX beírása
    I2C0->C1 |= 0x20;                   // MST beírása, START generálása
    I2C0->D = SLAVE_ADDR << 1;          // Slave cím és W bit küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x10) {               // Arbitráció elvesztése esetén
        I2C0->S |= 0x10;                // ARBL bit törlése
        return ERR_ARB_LOST;            // Kilépés hibakóddal
    }
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal

    I2C0->D = regAddr;                  // Regisztercím küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal

    I2C0->C1 |= 0x04;                   // RESTART feltétel generálás
    I2C0->D = (SLAVE_ADDR<<1) | 1;      // Slave cím és R bit küldése
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    if (I2C0->S & 0x01)                 // Negatív nyugtázás esetén
        return ERR_NO_ACK;              // Kilépés hibakóddal
```

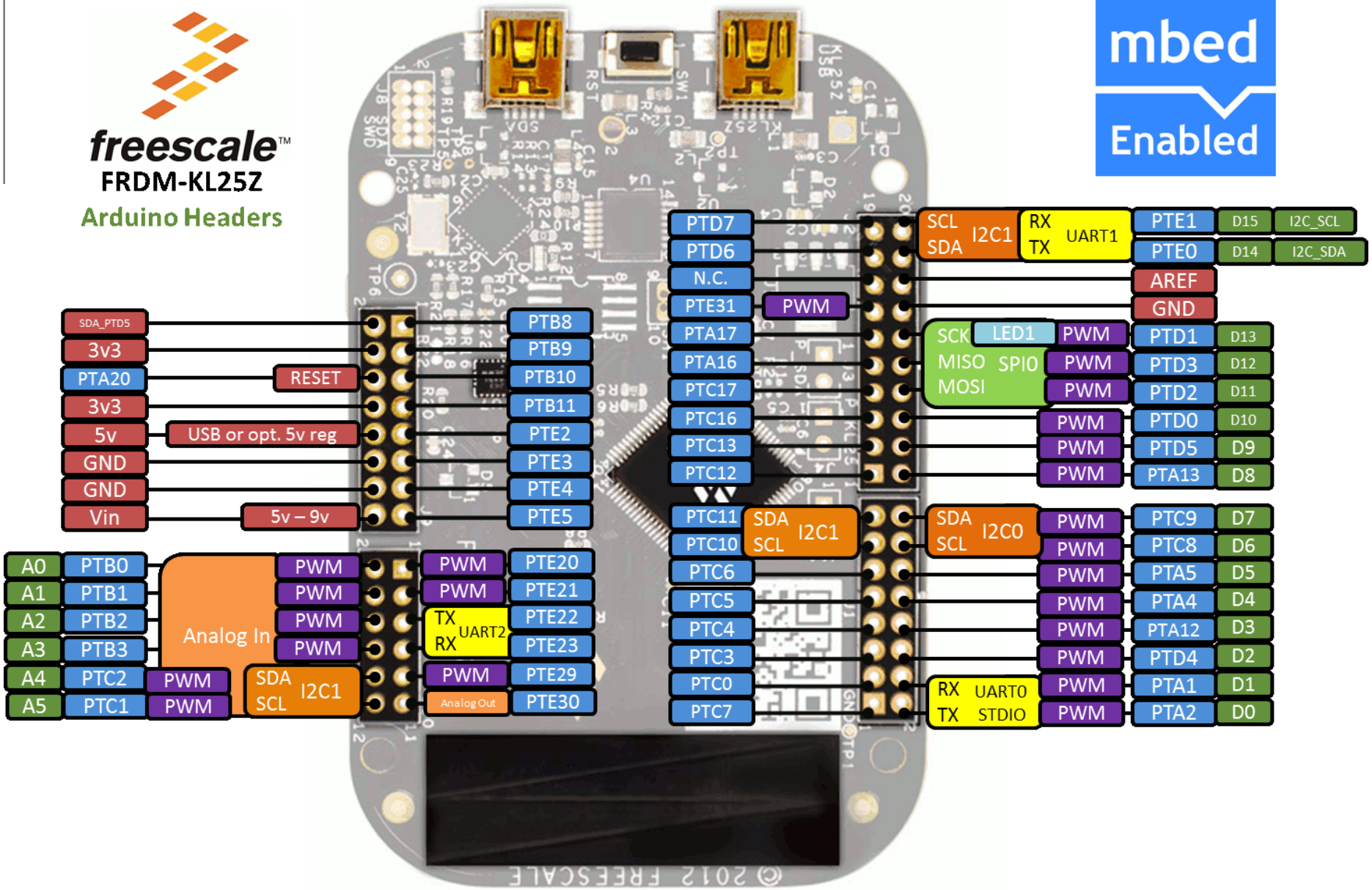
```
I2C0->C1 &= ~0x18;                  // TX bit törlése, ACK küldés elokészítése
    I2C0->C1 |= 0x08;                   // NAK küldés elokészítése
    dummy = I2C0->D;                    // kamu olvasás, és új ciklus indítása
    while(!(I2C0->S & 0x02));           // Átvitel végére várunk
    I2C0->S |= 0x02;                    // IICIF törlése
    I2C0->C1 &= ~0x20;                  // STOP feltétel generálása
    *data++ = I2C0->D;                  // Vett adat kiolvasása
    return ERR_NONE;
}
```

A program futási eredménye (közel vízszintes helyzetben)



freescale™
FRDM-KL25Z
Arduino Headers

mbed Enabled



freescale™

FRDM-KL25Z

Additional Peripherals