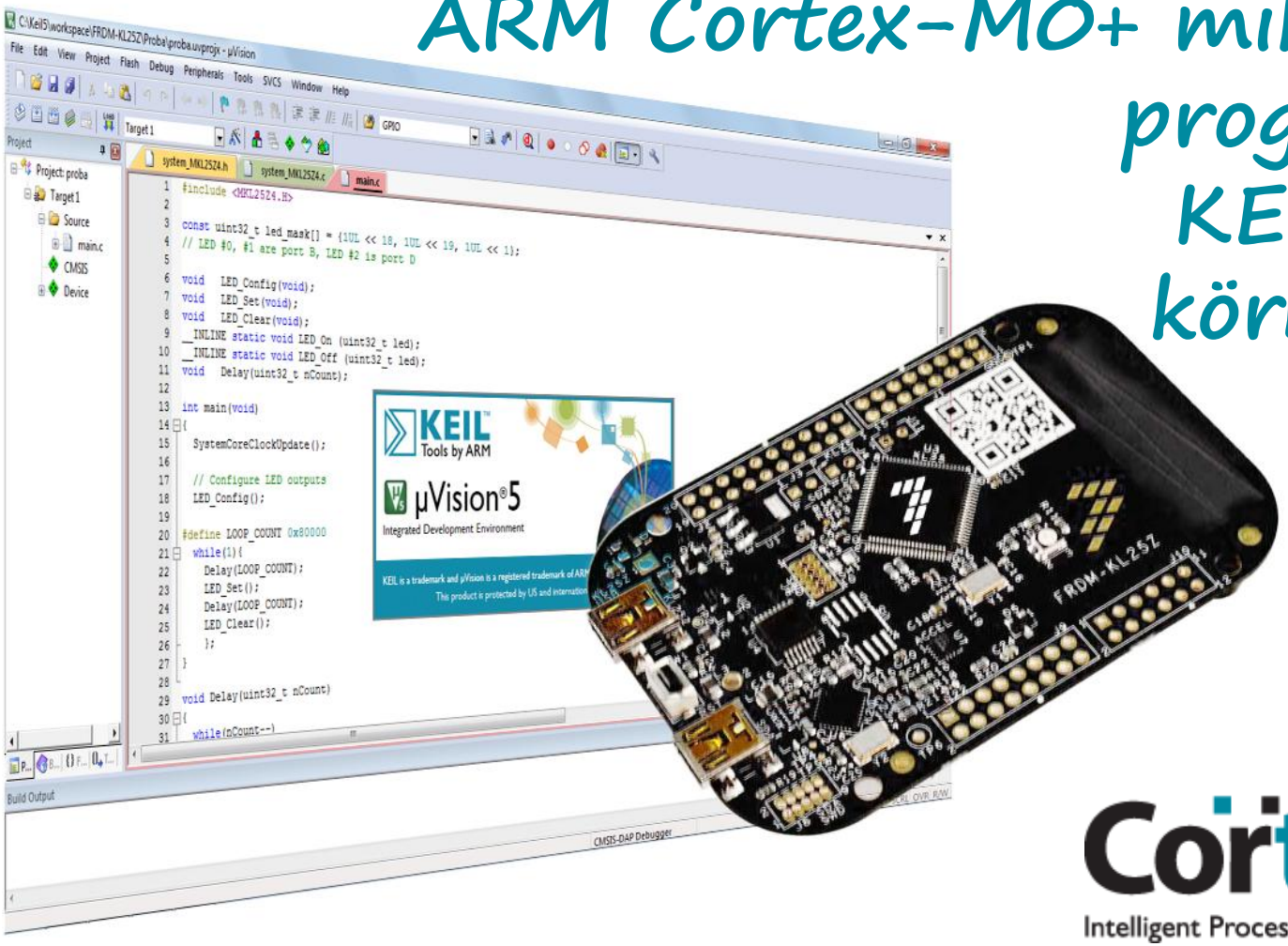
ARM Cortex-M0+ mikrovezérlő programozása KEIL MDK 5 környezetben



Cortex
Intelligent Processors by ARM®

4. Aszinkron soros kommunikáció (UART)

Felhasznált anyagok, ajánlott irodalom

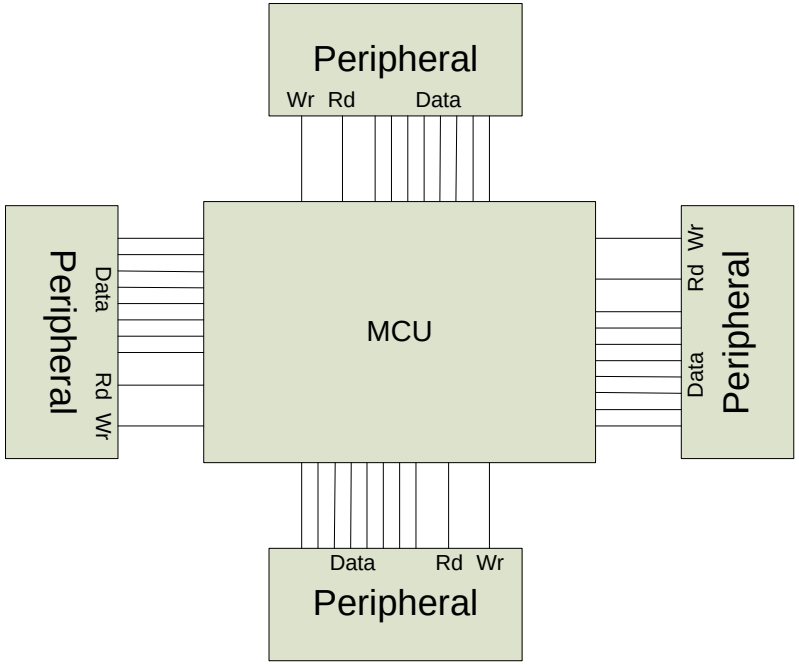
- Joseph Yiu: **The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors** (2nd Ed.)
- Muhammad Ali Mazidi, Shujen Chen, Sarmad Naimi, Sepehr Naimi: **Freescale ARM Cortex-M Embedded Programming**
- ARM University Program: **Course/Lab Material for Teaching Embedded Systems/MCUs** (for the FRDM-KL25Z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

Miért kell soros kommunikáció?

- Az adatok bájt vagy szó szervezésűek (8, 16, 32, stb.), ilyen „adagokban” küldjük vagy fogadjuk.

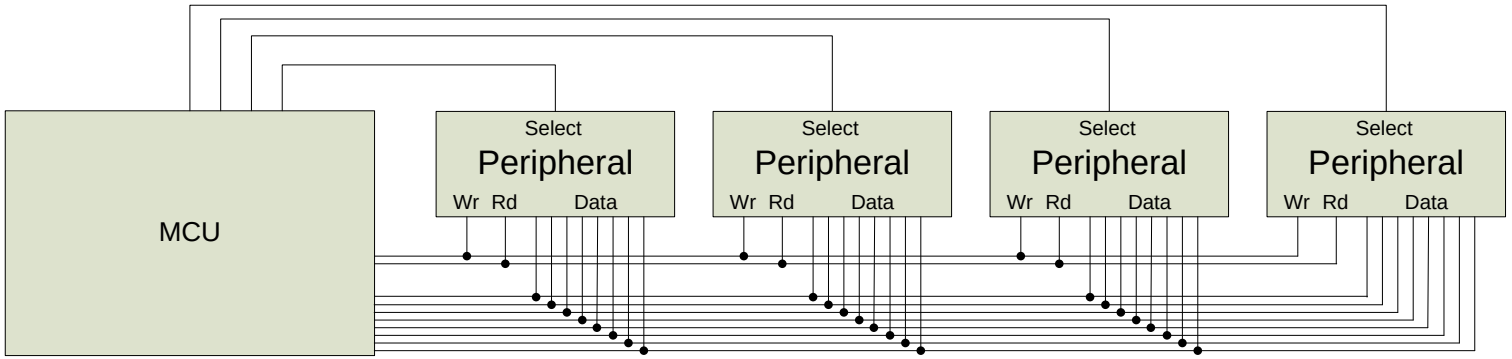
- Sok esetben nem célszerű az adatbiteket egyszerre, párhuzamosan küldeni
  - Költség és súly: több vezeték kell, nagyobb csatlakozókra van szükség
  - Mechanikus megbízhatóság: több vezeték=> több csatlakozási pont => több érintkezési meghibásodási lehetőség
  - Időzítési bonyodalmak: előfordulhat, hogy némelyik bit később érkezik meg, mint a többi, a vezetékek ellenállásának és szórt kapacitásának eltérései miatt
  - Áramkör bonyolultság és áramigény: nem biztos, hogy szeretnénk beépíteni 16 db rádió adóvevőt

Példa párhuzamos csatlakozásra



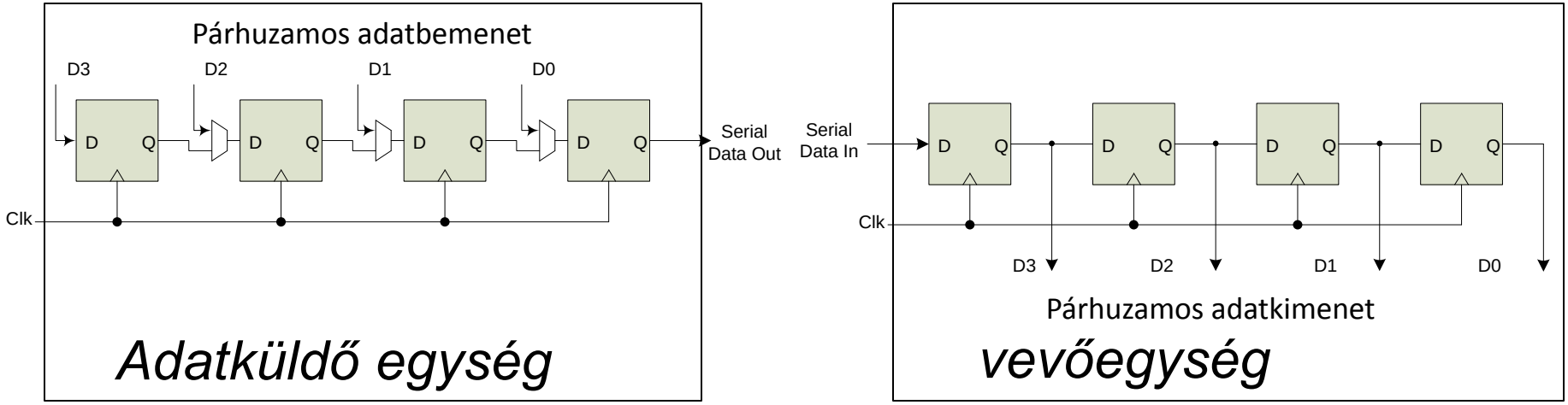
- Dedikált pont – pont kapcsolódás
  - Párhuzamos adatvonalak, küldő/fogadó vezetékek az MCU és a perifériák között
- Gyors, szimultán adatküldést is támogat
- Sok vezetéket, nyomtatott áramkörön sok helyet igényel, rosszul skálázható

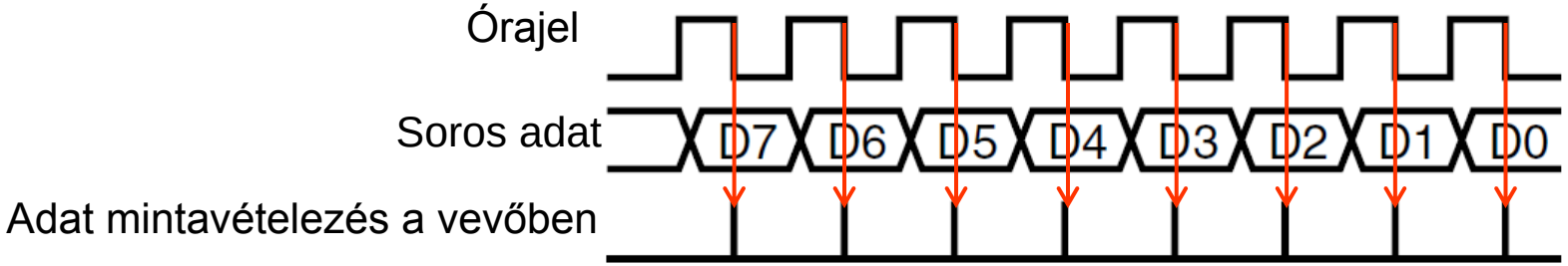
Párhuzamos buszrendszer



- Minden eszköz a buszvezetékekre csatlakozik
- Az MCU egyedi választóvonalakat használ a periféria kiválasztására
- Az MCU kevesebb vezetéket használ, de az adatbitenként külön vezeték kell
- Az MCU egyszerre csak egy perifériával kommunikálhat

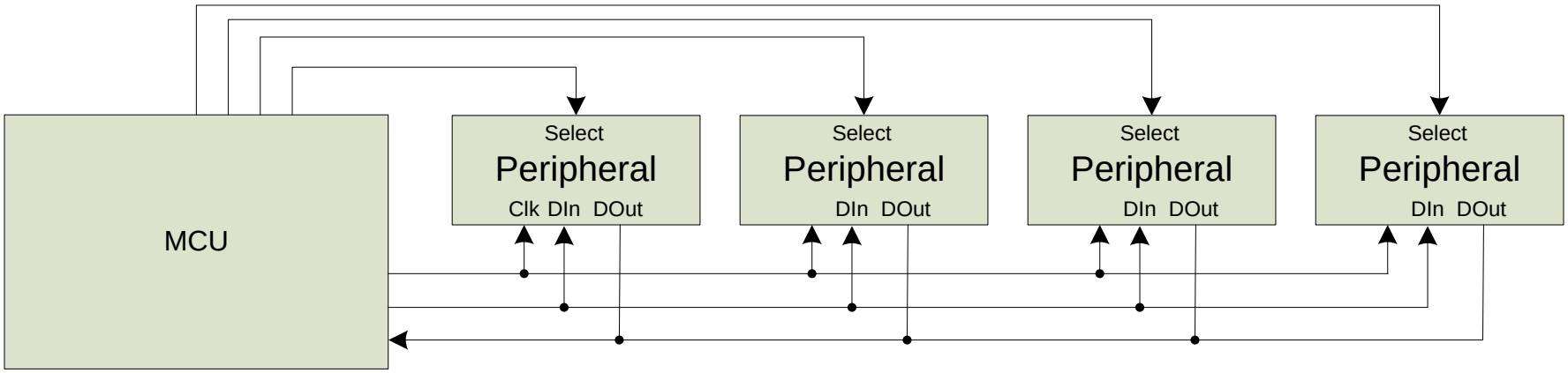
Szinkron soros adatküldés





- Shift regiszterek és órajel segítségével történik a soros és párhuzamos adatformátumok közötti átalakítás
- Szinkron: az adatjellel szinkronban egy ütemvezérlő órajelet is küldeni kell

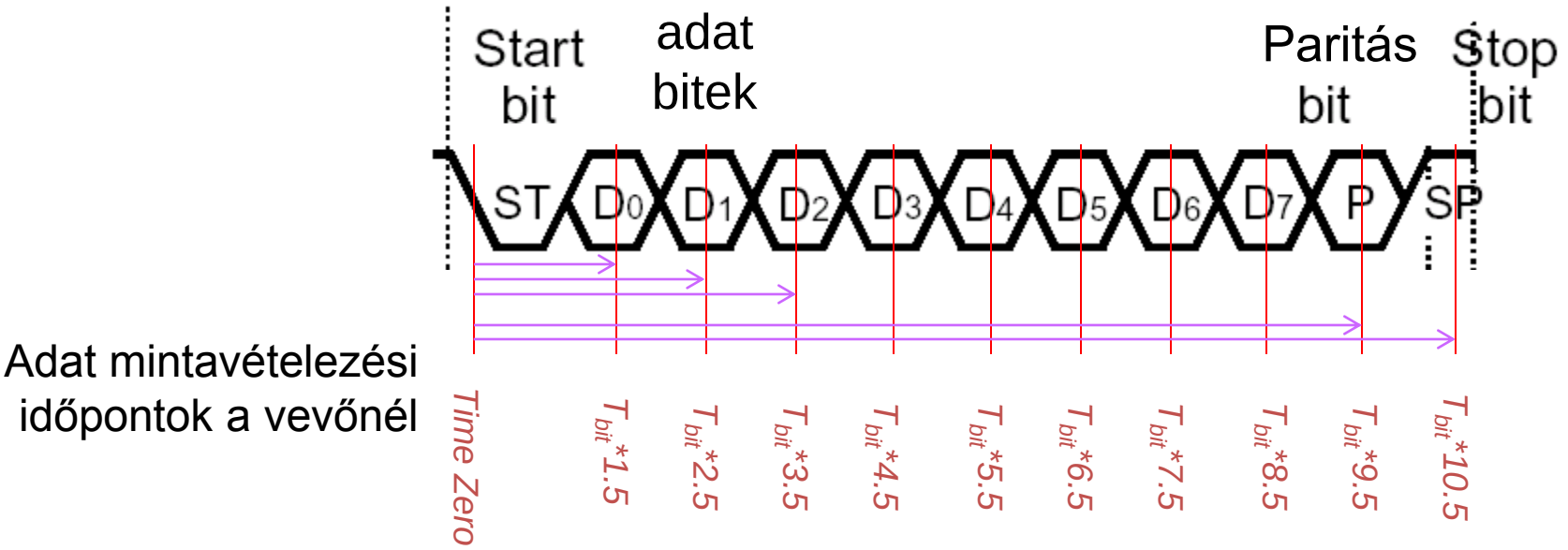
Szinkron kétirányú soros adatbusz



- Két adatvonalat használunk – egyet küldésre, egyet fogadásra.
  - Szimultán adatküldést és –fogadást tesz lehetővé (full-duplex kapcsolat) az MCU és egy kiválasztott periféria között.
  - A kapcsolat aszimmetrikus: az MCU a ***master*** eszköz, ez biztosítja az órajelet és a kiválasztó jelet. A megszólított eszköz a ***slave,*** amelynek alkalmazkodnia kell a masterhez.

Ilyen típusú kapcsolatot valósítanak meg meg a mikrovezérlő SPI moduljai (SPI = Soros Periféria Illesztő).

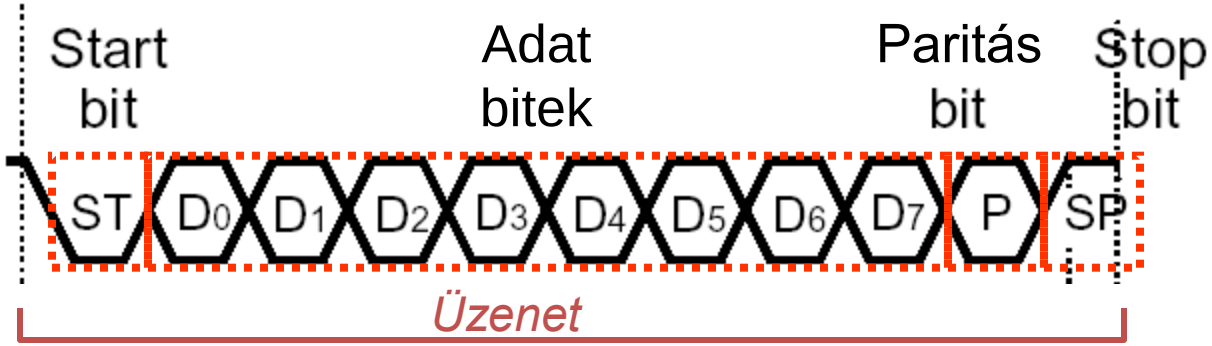
Aszinkron soros kommunikáció (UART)



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

UART kommunikáció jellemzői

- Adatkeret:
  - Start bit (1 bit)
  - Adat (LSB vagy MSB sorrend, adatméret – 7, 8, 9 bit)
  - Opcionális paritásbit használható az adatban szereplő egyesek páros vagy páratlan számának jelzésére.
  - Stop bit (egy vagy két stop bit használható)



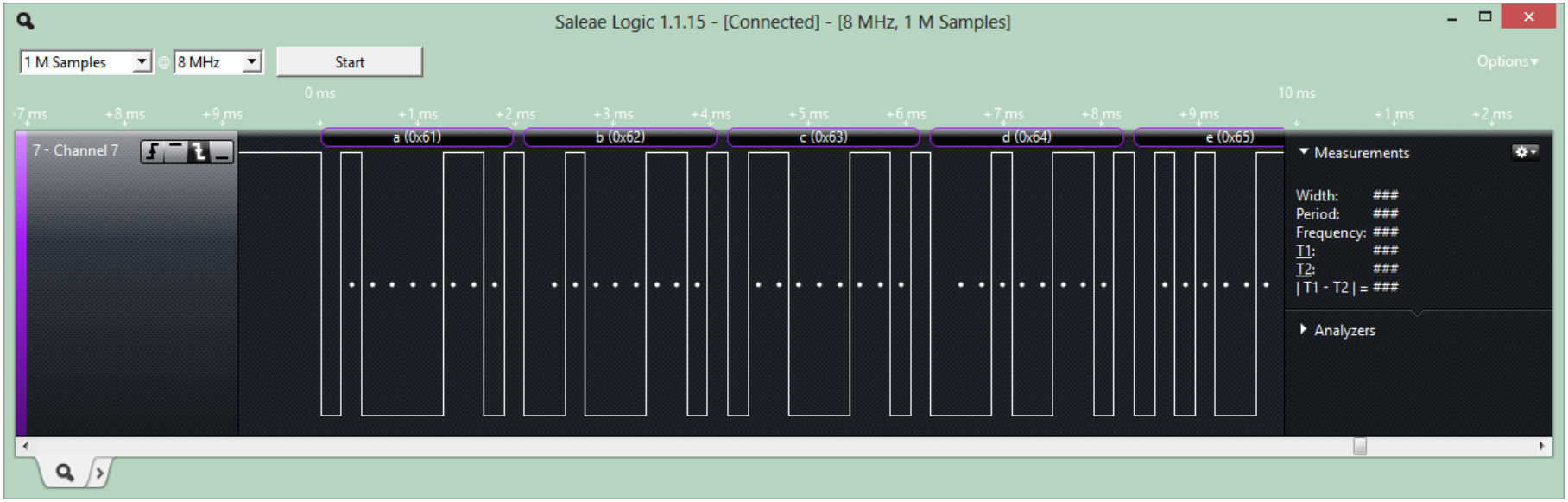
- Az adó és a fogadó fél meg kell, hogy egyezzen:
  - Kommunikációs sebesség (300 baud, 600, 1200, 2400, 9600, 14400, 19200, stb.)

- Szofisztikált hálózati protokollok üzenetenként további adatokat tartalmazhatnak:
  - Médium hozzáférés – ha több egység kapcsolódik a buszra, *arbitrációra* van szükség annak eldöntésére, hogy melyikük küldhet adatot
  - Címzési információ – melyik csomópontnak szól az üzenet?
  - Nagyobb méretű keretezett üzenetcsomag
  - Erősebb hibadetektáláshoz vagy hibajavításhoz szükséges információ (pl. CRC)
  - Kérelem azonnali válaszért

Hibadetektálás

- Járulékos adat küldhető az adattévesztések detektálásához
- Specifikálni kell, hogy milyen paritásjelzést várunk: páros, páratlan vagy nincs.
- A paritásbit páratlan számúra egészíti ki az adatban szereplő egyesek számát (páratlan paritás esetén) vagy páros számúra (páros paritás esetén)
  - 01110111 például 6 db. “1”-es bitet tartalmaz, tehát a paritásbit 1 lesz páratlan, illetve 0 lesz páros paritás esetén
  - 01100111 például db. “1”-es bitet tartalmaz, tehát a paritásbit 0 lesz páratlan, illetve 1 lesz páros paritás esetén
- Egyetlen paritásbit jelzi, ha 1, 3, 5, 7 vagy 9 bit hibás, de nem detektálja a páros számú bithibákat.
- Erősebb hibadetektáló kód (pl. Cyclic Redundancy Check – CRC) is létezik, mely több bitnyi (pl. 8, vagy 16) bites, és többféle hiba detektálását teszi lehetővé.
  - Használja a CAN, USB, Ethernet, Bluetooth stb.

Eszközök soros kommunikációs fejlesztéshez



- Fárasztó és lassú a soros protokollok értelmezése oszcilloszkóppal
- Helyette használjunk olyan logikai analizátort, amely képes értelmezni az adatfolyamot

- Saelae 8-csatornásl Logikai Analizátor
  - $150 (www.saelae.com)
  - USB porton csatlakozik a PC-hez
  - Képes dekódolni az SPI, UART, I^2C, 1-Wire, CAN, stb. kommunikációt
- Építs magadnak!: a Logic Sniffer vagy más hasonló projekt alapján

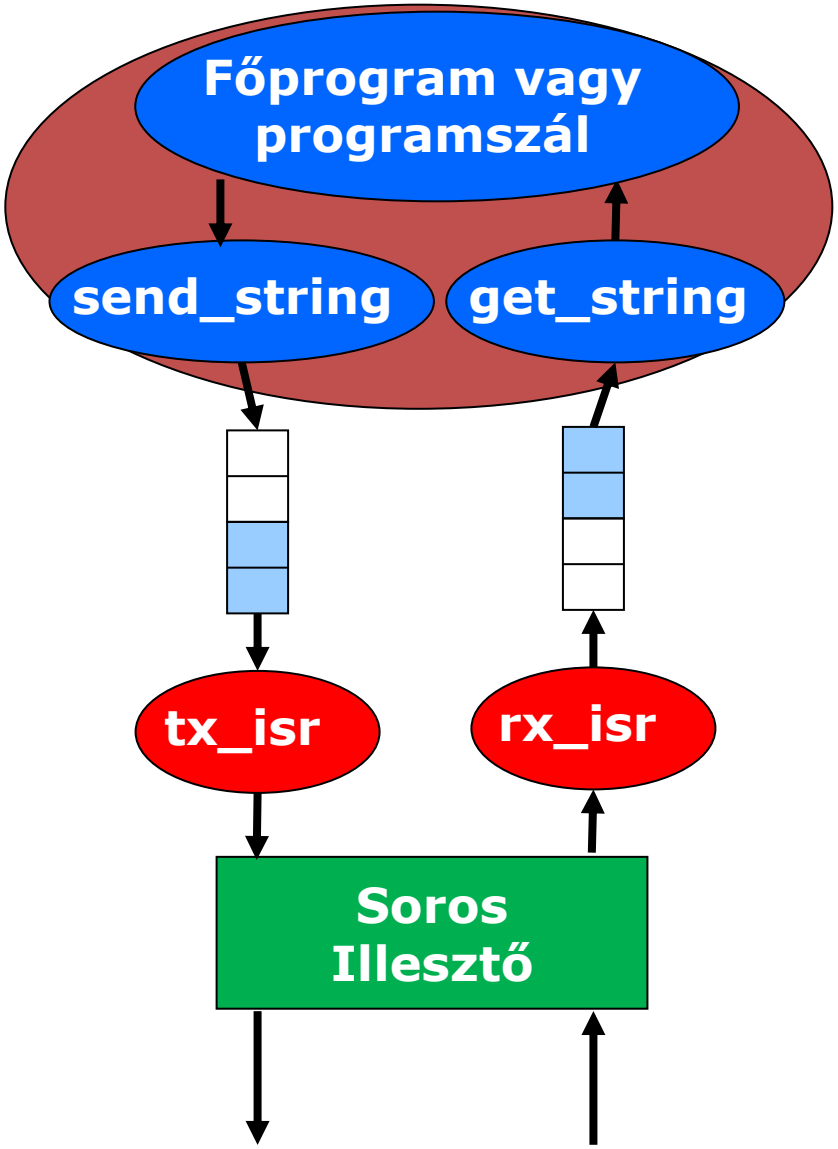
Szoftver megfontolások

- A kommunikáció aszinkron a programhoz képest is
  - Nem lehet előre tudni, hogy a program merre jár…
    - Amikor a következő adat beérkezik
    - Amikor a folyamatban levő adatátvitel befejeződik
    - Amikor valamilyen hiba lép fel az átvitelben
  - Valahogy szinkronizálni kell a program futását a kommunikációs interfésszel

- Lehetőségek
  - Lekérdezés
    - Várunk, amíg adat érkezik be
    - Egyszerű, de nem hatékonyan használja a processzor idejét
  - Megszakítás
    - Megszakítjuk a program futását, ha új adat érkezik
    - Hatékony, de bonyolultabb módszer

Soros kommunikáció és a megszakítások

- Több programszálat kell biztosítani a zavartalan kezeléshez
  - **Főprogram** (és az abból hívott függvények)
  - **Transmit ISR** – akkor aktiválódik, amikor az eszköz kész új adat küldésére
  - **Receive ISR** – akkor aktiválódik, amikor új adat érkezett
  - **Error ISR** – akkor aktiválódik, ha adatátvitel közben hiba történt
- Adatbufferelés a programszálak között
  - Megoldás: **gyűrűs tár,** beléptetési is kivételi mutatókkal
  - Egy az adáshoz, egy a vételhez



KL25 soros perifériák

Az MKL25Z128VLK4 MCU soros kommunikációs perifériái:

- **USB OTG** periféria, amely Full speed (12 Mbit/s) és Low speed (1.5 Mbit/s) kommunikációra alkalmas
- **2 db SPI** (Serial Peripheral Interface) modul
- **2 db I2C** (Inter Integrated Circuits) modul
- **3 db UART** (Universal Asynchron Transmitter Receiver) melyek közül az egyik kis fogyasztású
  - **UART0**
    - Energia-takarékos MCU módokban is működhet
    - Választható órajelforrás
    - Túlmintavételezés 4x – 32x
  - **UART1, UART2**
    - Fix órajelforrás (a busz órajel)
    - Túlmintavételezés 16x

UART modulok kivezetései

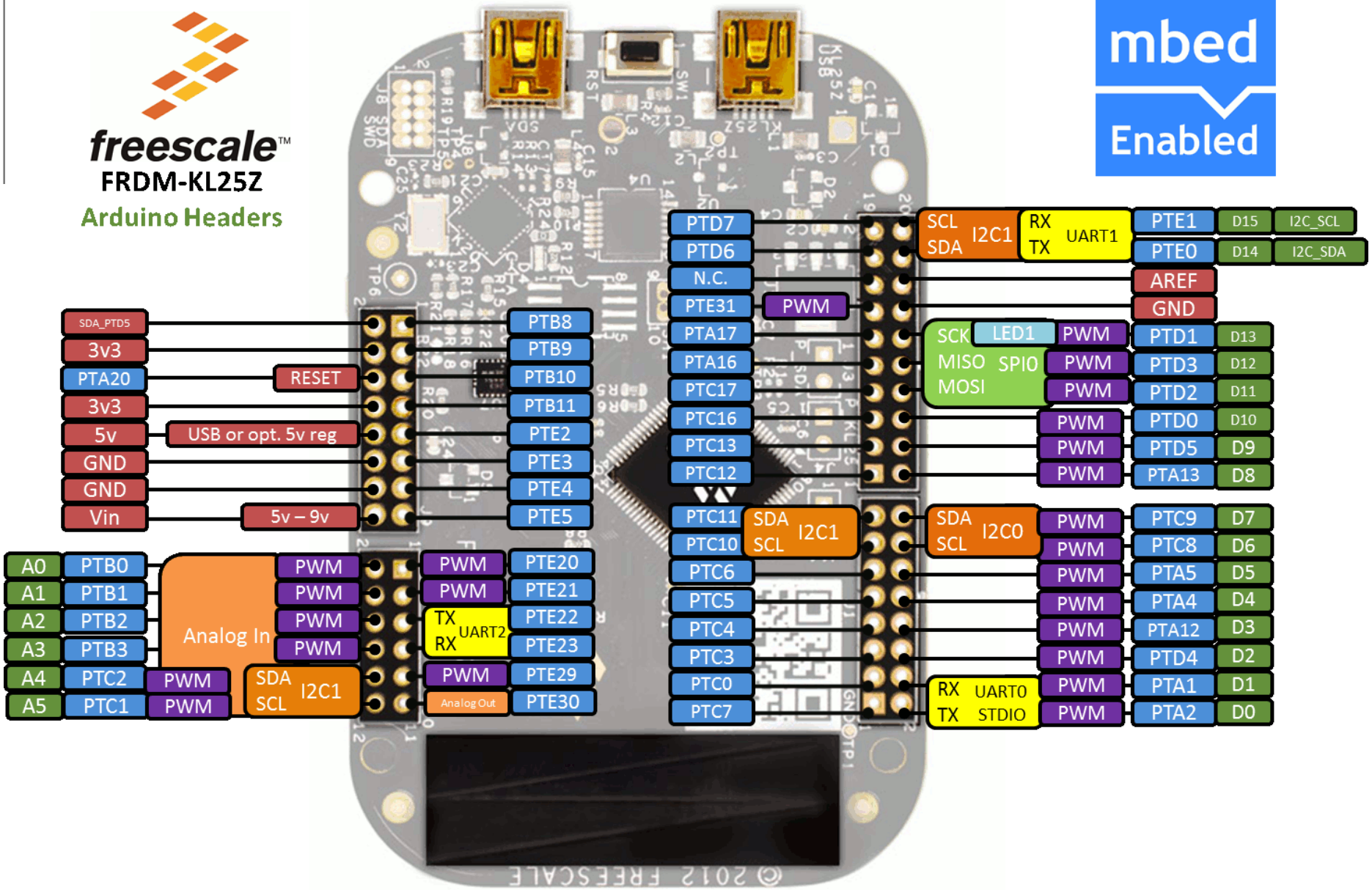
Az alábbi táblázatban felsoroltuk azokat az UART kivezetéseket, amelyek a **FRDM-KL25Z** kártyán elérhetők. Zárójelben megadtuk a kivezetés üzemmódjának beállításához szükséges számot is, amit a megfelelő **PORT***x*_**PCR***n* portvezérlő regiszter **MUX** bitcsoportjába kell beírni az UART mód kiválasztásához. Például **UART0 PTA1** és **PTA2** kivezetések engedélyezése:

```
SIM->SCGC5 |= 0x0200;    // PORTA engedélyezése
PORTA->PCR[1] = 0x0200; // PTA1 UART RX módba állítása
PORTA->PCR[2] = 0x0200; // PTA2 UART TX módba állítása
```

Modul	TX	RX	Megjegyzés
UART0	**PTA2 (2)**	**PTA1 (2)**	OpenSDA-hoz csatlakozik
	PTD7 (3)	PTD6 (3)	
	PTE20 (4)	PTE21 (4)	
UART1	**PTE0 (3)**	**PTE1 (3)**	Támogatott kivezetés
	PTC4 (3)	PTC3 (3)	
UART2	**PTE22 (4)**	**PTE23 (4)**	Támogatott kivezetés
	PTD3 (3)	PTD2 (3)	
	PTD5 (3)	PTD4 (3)	

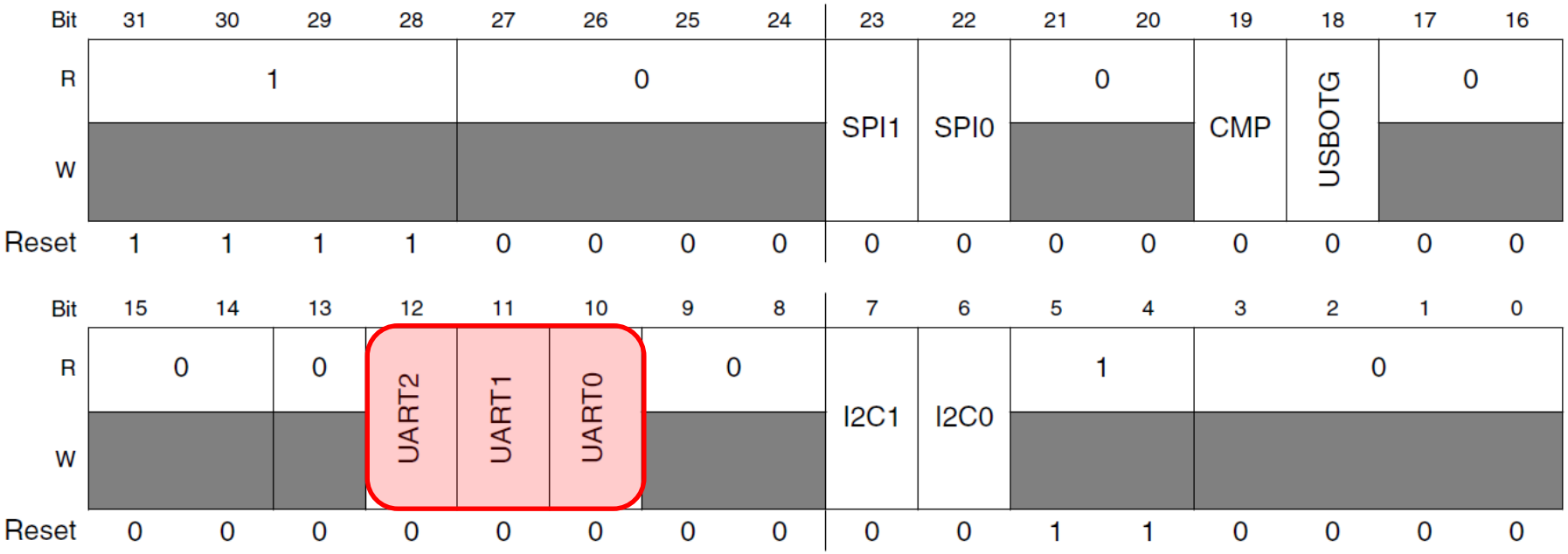
freescale™
FRDM-KL25Z
Arduino Headers

mbed Enabled



KL25Z soros perifériák engedélyezése

SIM_SCGC4 regiszter



- A használni kívánt perifériát engedélyezni kell a **SIM_SCGC4** regiszter megfelelő bitjének 1-be állításával

Az UARTx modulok regiszterei

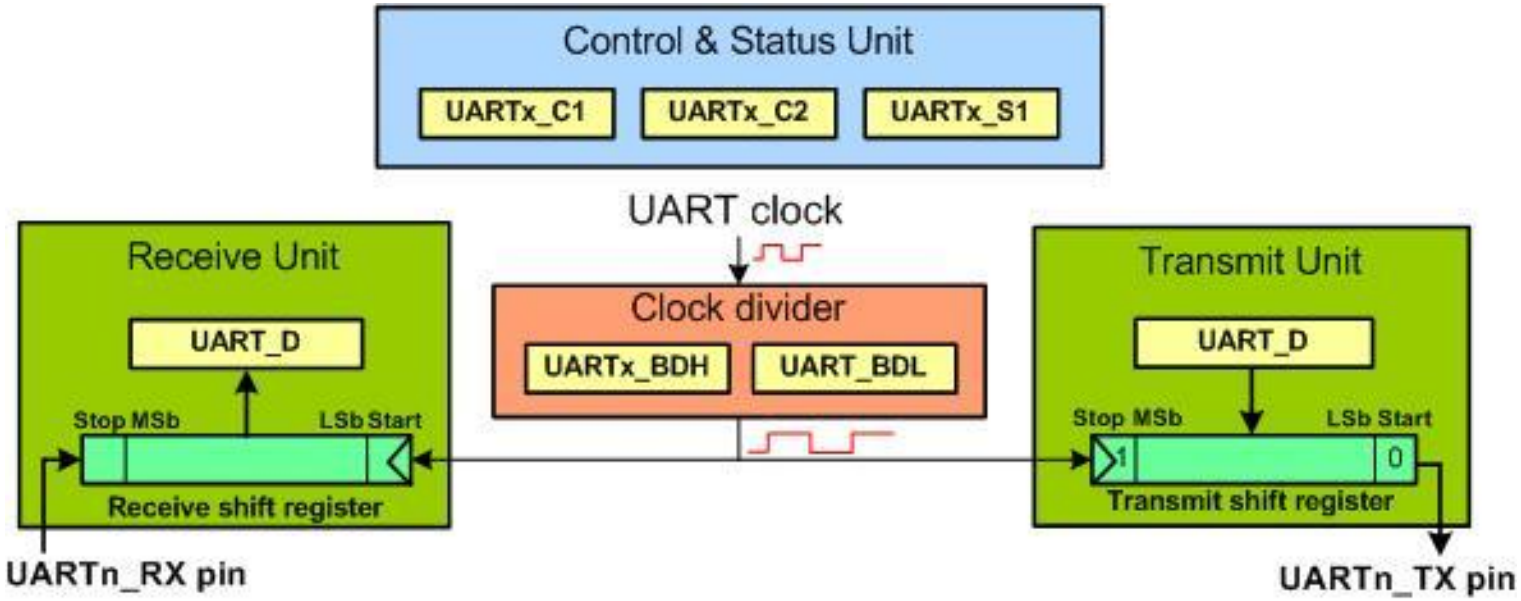
UART0 báziscím: 0x4006 A000, **UART1** báziscím: 0x4006 B000, **UART2** báziscím: 0x4006 C000

Az **UART0** modul regiszterkészlete (8-bites regiszterek):

(Freescale: **KL25 Sub-Family Reference Manual**, **39. fejezet**)

Regiszternév	A regiszter funkciója	Cím
UART0_BDH	Baud Rate Register High (adatsebesség - magas h.é.)	4006_A000
UART0_BDL	Baud Rate Register Low (adatsebesség - alacsony h.é.)	4006_A001
UART0_C1	Control Register 1 (1. vezérlő regiszter)	4006_A002
UART0_C2	Control Register 2 (2. vezérlő regiszter)	4006_A003
UART0_S1	Status Register 1 (1. állapotjelző regiszter)	4006_A004
UART0_S2	Status Register 2 (2. állapotjelző regiszter)	4006_A005
UART0_C3	Control Register 3 (3. vezérlő regiszter)	4006_A006
UART0_D	Data Register (Adat regiszter)	4006_A007
UART0_MA1	Match Address Register 1 (1. címegyezés regiszter)	4006_A008
UART0_MA2	Match Address Register 2 (2. címegyezés regiszter)	4006_A009
UART0_C4	Control Register 4 (4. vezérlő regiszter)	4006_A00A
UART0_C5	Control Register 5 (5. vezérlő regiszter)	4006_A00B

Egyszerűsített blokkvázlat



- **Konfigurációs regiszterek:** Mielőtt használnánk az UART modult, konfigurálni kell azt. Be kell állítani az adatsebességet, a szóhosszt, a stop bitek számát, a megszakítást (ha kell) stb. A legfontosabb regiszterek: **UARTx_BDH, UARTx_BDL, UARTx_C1,** és **UARTx_C2.**
- **Adatküldő és vételi regiszter:** az adatküldés vagy –fogadás esetünkben egyszerűen az UART_D regiszter írásval/olvasásával történik.
- **Státusz regiszter:** a státusz regiszter bitjei az UART modul állapotát, különféle események bekövetkeztét jelzik, például új adat beérkezését, a beérkezett adat érvénytelenségét, az adatküldés végét stb. A számunkra legfontosabb állapotjelző biteket az UARTx_S1 regiszter tartalmazza.

UART órajel források

UART0 számára többféle forrásból választhatunk. Ha STOP módban is működtetni akarjuk UART0-t, a választott órajel generátor is működjön abban a módban!

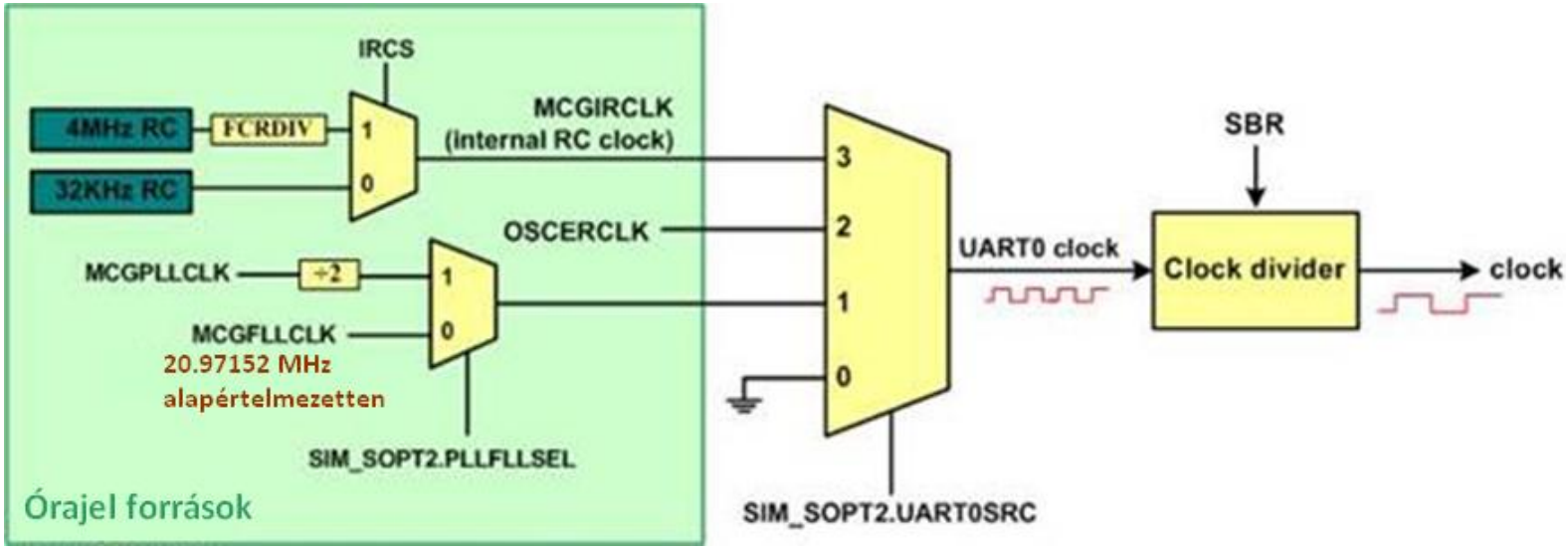
Az órajel kiválasztása a **SIM_SOPT2** regiszter segítségével történik.

SIM_SOPT2: (SIM->SOPT2)	D31 … D28	D27 D28	D25 D24	D23 … D19	D18	D17	D16	D15 ….. D8	D7 … D5	D4 D3	… D0
	0	UART0SRC	TPMSRC	0	USBSRC	0	PLLFLLSEL	0	CLKOUTSEL	RTCCLKOUTSEL	0

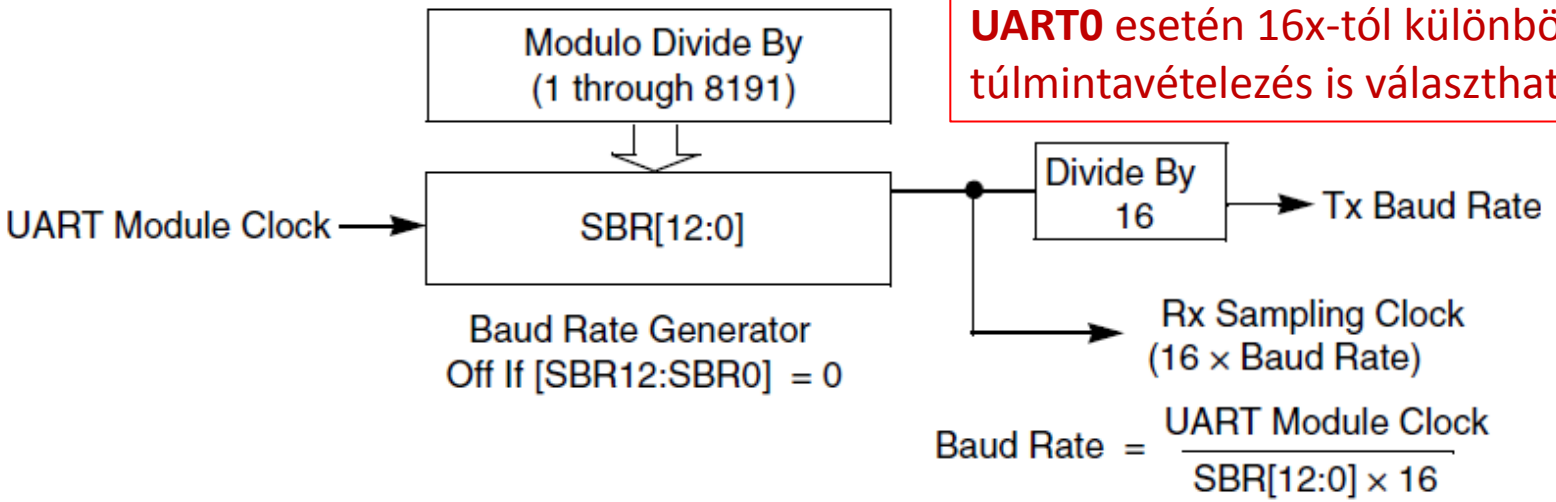
0x1004

bit	Name	Description
27-26	UART0SRC	***UART0SRC (UART0 clock source select)*** selects the clock source for UART0. 00 Clock disabled 01 MCGFLLCLK clock or MCGFLLCLK/2 10 OSCERCLK clock 11 MCGIRCLK clock
16	PLLFLLSEL	***It selects between MCGPLLCLK/2 and MCGFLLCLK for various peripherals:*** 0: MCGFLLCLK 1: MCGPLLCLK/2

(Lásd: [KL25 Sub-Family Reference Manual](), **5. és 12. fejezet**)



Baud Rate Generator



UART0 esetén 16x-tól különböző túlmintavételezés is választható!

- Az UART modul órajelének frekvenciáját le kell osztani a kívánt bitsebesség és a túlmintavételezési arány szorzatának eléréséhez
- **Példa:** 24 MHz buszfrekvencia -> 4800 baud, 16-szoros túlmintavételezéssel

  Osztó = 24 000 000/(4800*16) = 312.5. Kerekíteni kell a legközelebbi egészre (312 vagy 313), végeredményben lesz egy kis eltérés a névleges értéktől.

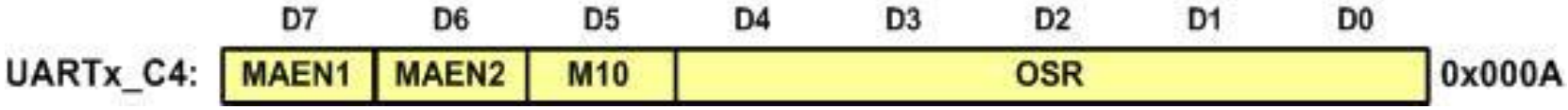
UART1, illetve **UART2** órajele csak a periféria busz órajele lehet:

- Alapértelmezett órajel esetén (CLOCK_SETUP = 0) a busz órajel 10,48576 MHz-es
- 48 MHz CPU frekvencia (CLOCK_SETUP = 1, vagy 4) esetén a busz órajel 24 MHz-es

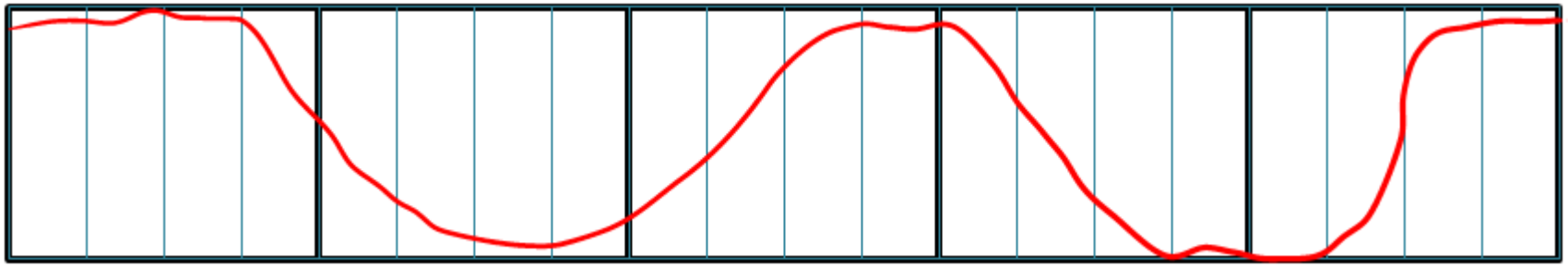
SBR és OSR konfigurálása

A 13 bites **SBR** értéke 1 – 8191 közötti érték lehet, s bitjeit a **BDH:BDL** regiszterekbe kell írni

UART0 esetén meg kell adni a túlmintavételezési arányt is (OSR), melynek 1-gyel csökkentett értékét az **UART0_C4** regiszter **OSR** bitcsoportjába kell írni.



Bemenő adat túlmintavételezés



- Vételnél az UART túlmintavételezi (oversampling) a bejövő adatvonalat
  - Az extra minta lehetővé teszi a többségi szavazást, javítja a zajérzéketlenséget
  - Jobb szinkronizálás a bejövő adathoz

- **UART0** konfigurálható túlmintavételezés: 4-szerestől 32-szeresig
  - A kívánt túlmintavételezési arány – 1 írandó az **UART0_C4** vezérlőregiszter **OSR** bitcsoportjába.

- **UART1, UART2** fix 16-szoros túlmintavételezés

UART vezérlő regiszter 1 (UART0_C1)

Bit	7	6	5	4	3	2	1	0
Read Write	LOOPS	DOZEEN	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

LOOPS: Visszahurkolás/egyvezetékes (TX/RX) mód

DOZEEN: Energiatakarékos mód engedélyezés – UART tiltása alvás módban

RSRC: Visszahurkolás vagy egyvezetékes mód közötti választás

M: 9-bites adatmód (8-bites mód helyett)

WAKE: Felébresztési mód

ILT: Tétlen vonal típusa

PE: Paritás engedélyezés (ha 1)

PT: Paritás (0: páros, 1: páratlan)

Egyszerű esetekben megfelel számunkra az alapértelmezett UART0_C1 = 0; beállítás

UART vezérlő regiszter 2 (UART0_C2)

Bit	7	6	5	4	3	2	1	0
Read Write	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

- Megszakítás engedélyezés
  - **TIE:** Megszakítás ha a küldő adatregiszter üres
  - **TCIE:** Megszakítás az átvitel végén
  - **RIE:** Megszakítás ha van beérkezett adat
- **Modul engedélyezés**
  - **TE:** Adó (transmitter) engedélyezés
  - **RE:** Vevő (receiver) engedélyezés
- Továbbiak
  - **RWU:** a vevőt standby módba teszi, felébresztés az előírt feltételek esetén
  - **SBK:** Break karakter (csupa nulla) küldése

Az UARTx modulok konfigurálása előtt le kell tiltani az adó és vevőegységet!

UART státusz regiszter 1 (UART_S1)

Bit	7	6	5	4	3	2	1	0
Read	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Write				w1c	w1c	w1c	w1c	w1c
Reset	1	1	0	0	0	0	0	0

- **TDRE:** A küldő regiszter felszabadult, újabb adatot írhatunk bele
- **TC:** Adatátvitel kész.
- **RDRF:** Adat érkezett a vételi adatregiszterbe
- **IDLE:** az UART vevő bemenete tétlen már egy karakternyi ideje
- **OR:** Ráfutási hiba. A beérkezett adat felülírta a korábbi, még nem kiolvasottat.
- **NF:** Zaj jelző. A vett bitminták ellentmondásosak (túl gyors változás)
- **FE:** Keretezési hiba. A stop bit helyén 0-t érzékeltünk 1 helyett.
- **PF:** Paritáshiba a vett adatban.

Az UART használata

- Mikor küldhetünk adatot?
  - A küldő buffer üres legyen!
  - Vizsgáljuk az **UARTx->S1 TDRE** jelzőt!
  - Vagy használjunk megszakítást a küldéshez, ehhez azonban várakozási sorba kell pakolni az adatokat!
- Írjuk az adatot az UARTx_D (CMSIS-ben **UARTx->D**) regiszterbe!

- Mikor tudunk adatot fogadni?
  - A vevőbuffer megtelt legyen
  - Vizsgáljuk a **UARTx->S1 RDRF** jelzőt
  - Vagy használjunk megszakítást a fogadáshoz, majd pakoljuk várakozási sorba az adatokat
- Az adatot az UARTx_D (CMSIS esetén *UARTx->D*) regiszterből vehetjük elő

Program4_1: UART0 egyszerű adatküldés

Az első programban "Yes" szöveget küldünk ki a **FRDM-KL25Z** kártya **UART0 Tx** kimenetén (PTA2) keresztül a számítógép felé, az OpenSDA által biztosított virtuális soros portra.

A konfigurálás lépései:

1. Engedélyezzük az **UART0** modult a **SIM_SCGC4** regiszter 10. bitjének 1-be állításával.
2. Legyen **FLL** az **UART0** órajel forrása: **SIM_SOPT2** bit27-26 = 01 és bit16 = 0.
3. Az **UART0** Tx/Rx letiltása konfigurálás előtt: **UART0_C2** regiszter TE=0, RE = 0.
4. Az **UART0** bitsebesség konfigurálása **UART0_BDH** és **UART0_BDL** beírásával.
5. Túlmintavételezés konfigurálása: OSR = 16x-hoz **UART0_C4**-ba írjunk 0x0F-et!
6. Adatformátum konfigurálása: 1 stop bit, nincs paritás vizsgálat, 8-bites adat megadásához írjunk 0x00-t az **UART0_C1** vezérlő regiszterbe!
7. Engedélyezzük **UART0** adóját: írjunk 0x08-at az **UART0_C2** regiszterbe (**TE** bit = 1)!
8. Engedélyezzük az **A** portot: a **SIM_SCGC5** regiszter 9. bitje legyen 1!
9. Válasszuk ki a **PTA2** kivezetés Alt2 funkcióját (UART0_Tx) **PORTA_PCR2** MUX=2.

Az adatküldés lépései:

10. Vizsgáljuk a Status Register 1 (**UART0_S1**) **TDRE** bitjét és várjunk addig, amíg a küldő buffer üressé válik (TDRE = 1)!
11. Írjuk bele a kiküldendő bájtot az **UART0_D** adatregiszterbe!
12. Újabb karakter kiküldéséhez menjünk a 10. ponthoz!

Program4_1/1.

A program forrása: Mazidi et al., Freescale ARM Cortex-M Embedded Programming
http://www.microdigitaled.com/ARM/Freescale_ARM/Code/Ver1/Chapter4/Program4_1.txt
Változtatás: BDH:BDL újraszámolása az eltérő frekvencia miatt,

```
#include <MKL25Z4.h>
void UART0_init(void);
void delayMs(int n);

int main (void) {
  UART0_init();                          // UART0 kimenet inicializálása
  while (1) {
    while(!(UART0->S1 & 0x80));          // Várunk, míg üres lesz a kimeneti buffer!
    UART0->D = 'Y';                      // karakter küldés (TDRE = 1 után)
    while(!(UART0->S1 & 0x80));          // Várunk, míg üres lesz a kimeneti buffer!
    UART0->D = 'e';                      // karakter küldés
    while(!(UART0->S1 & 0x80));          // Várunk, míg üres lesz a kimeneti buffer!
    UART0->D = 's';                      // karakter küldés
    delayMs(2000);                       // Várunk egy kicsit...
  }
}

void delayMs(int n) {                    // Késleltető függvény
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 3500; j++); // Alapértelmezett órajelhez (20.97152 MHz)
}
```

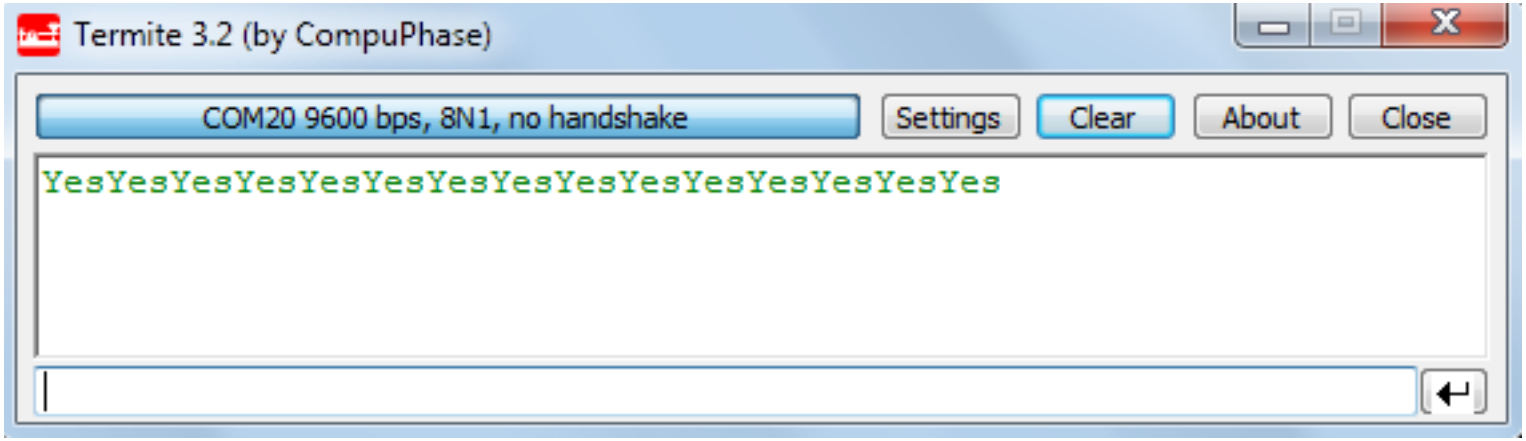
Folytatás a következő oldalon ...

Program4_1/2.

```
/* UART0 Tx inicializálása 9600 Baud átviteli sebességre
 * Baudrate = UART0 clock freq / [OSR+1] / BDH:BDL
 * Esetünkben 20.97 MHz / [15 + 1] / 0x00:0x88 =  9637 bps,
 * ami elfogadható eltérés a névleges értéktől.
 */
void UART0_init(void) {
    SIM->SCGC4 |= 0x0400;          // UART0 periféria engedélyezése
    SIM->SOPT2 |= 0x04000000;      // MCGFLLCLK választása UART0 baudrate órajelnek
    UART0->C2 = 0;                 // UART0 letiltása a konfigurálás időtartamára
    UART0->BDH = 0x00;             // Baudrate = 9600 bps (bit8 - bit12)
    UART0->BDL = 0x88;             // Baudrate = 9600 bps (bit0 - bit7)
           UART0->C4 = 0x0F;               // Túlmintavételezési arány = 16
    UART0->C1 = 0x00;              // 8-bites adatformátum
    UART0->C2 = 0x08;              // Adatküldés engedélyezése

    SIM->SCGC5 |= 0x0200;          // PORTA engedélyezése
    PORTA->PCR[2] = 0x0200;        // PTA2 legyen UART0_Tx üzemmódban
}
```

Program4_1 futási eredménye



Lekérdezéses küldés/fogadás

Hexadecimális bitvarázslás helyett használhatjuk az előre definiált elnevezéseket is!

```
void UART2_Transmit_Poll(uint8_t data) {
      // wait until transmit data register is empty
      while (!(UART2->S1 & UART_S1_TDRE_MASK));
      UART2->D = data;
}

uint8_t UART2_Receive_Poll(void) {
      // wait until receive data register is full
      while (!(UART2->S1 & UART_S1_RDRF_MASK));
      return UART2->D;
}
```

Program4_2: UART0 egyszerű adatfogadás

Karakter fogadása a **FRDM-KL25Z** kártya **UART0 Rx** bemenetén. A vett karakter legalsó bitjeivel az RGB LED állapotát vezéreljük. A LED kezelése **Program2_7**-hez hasonlóan történik.

A konfigurálás lépései:

1. Engedélyezzük az **UART0** modult a **SIM_SCGC4** regiszter 10. bitjének 1-be állításával.
2. Legyen **FLL** az **UART0** órajel forrása: **SIM_SOPT2** bit27-26 = 01 és bit16 = 0.
3. Az **UART0** Tx/Rx letiltása konfigurálás előtt: **UART0_C2** regiszter TE=0, RE = 0.
4. Az **UART0** bitsebesség konfigurálása **UART0_BDH** és **UART0_BDL** beírásával.
5. Túlmintavételezés konfigurálása: OSR = 16x-hoz **UART0_C4**-ba írjunk 0x0F-et!
6. Adatformátum konfigurálása: 1 stop bit, nincs paritás vizsgálat, 8-bites adat megadásához írjunk 0x00-t az **UART0_C1** vezérlő regiszterbe!
7. Engedélyezzük **UART0** vevőjét: írjunk 0x04-et az **UART0_C2** regiszterbe (**RE** bit = 1)!
8. Engedélyezzük az **A** portot: a **SIM_SCGC5** regiszter 9. bitje legyen 1!
9. Válasszuk ki a **PTA1** kivezetés Alt2 funkcióját (UART0_Rx) **PORTA_PCR1** MUX=2.

Az adatfogadás lépései:

10. Vizsgáljuk a Status Register 1 (**UART0_S1**) **RDRF** bitjét és várjunk addig, amíg a vevő bufferbe karakter érkezik (RDRF = 1)!
11. Olvassuk ki a vett bájtot az **UART0_D** adatregiszterből!
12. Újabb karakter fogadásához menjünk a 10. ponthoz!

Program4_2/1.

A program forrása: Mazidi et al., Freescale ARM Cortex-M Embedded Programming
http://www.microdigitaled.com/ARM/Freescale_ARM/Code/Ver1/Chapter4/Program4_2.txt

```
#include <MKL25Z4.h>

void UART0_init(void);                          // UART0 Rx bemenet inicializálása
void LED_init(void);                            // RGB LED inicializálása

int main (void) {
          char c;
    UART0_init();                               // UART0 Rx bemenet inicializálása
          LED_init();                                 // RGB LED inicializálása
    while (1) {
        while(!(UART0->S1 & 0x20));             // Adatbeérkezésre várunk
        c = UART0->D;                           // Kiolvassuk a vett karaktert
        if (c & 1) PTB->PCOR = 0x40000; // Piros LED: BE (bit18 = 0)
            else PTB->PSOR = 0x40000;   // Piros LED: KI           (bit18 = 1)
        if (c & 2) PTB->PCOR = 0x80000; // Zöld LED: BE  (bit19 = 0)
            else PTB->PSOR = 0x80000;   // Zöld LED: KI            (bit19 = 1)
        if (c & 4) PTD->PCOR = 0x02;    // Kék LED: BE   (bit1  = 0)
            else PTD->PSOR = 0x02;      // Kék LED: KI             (bit1  = 0)
    }
}
```

Folytatás a következő oldalon ...

Megjegyzés: az adatküldő terminál emulátor programot úgy konfiguráljuk, hogy a begépelt karakterhez elküldéskor ne fűzzön hozzá semmilyen sorvége jelet!

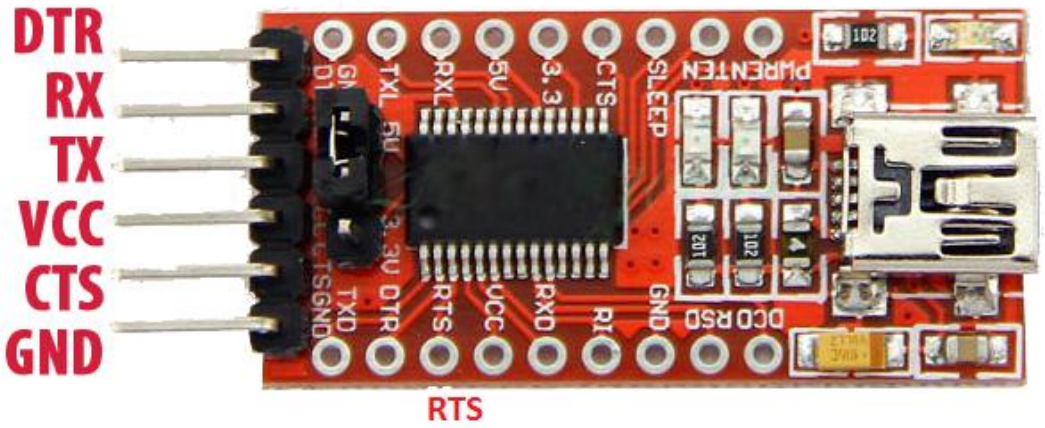
Program4_2/2.

```
/* UART0 Rx inicializálása 9600 Baud átviteli sebességre */
void UART0_init(void) {
    SIM->SCGC4 |= 0x0400;          // UART0 periféria engedélyezése
    SIM->SOPT2 |= 0x04000000;      // MCGFLLCLK választása UART0 baudrate órajelnek
    UART0->C2 = 0;                 // UART0 letiltása a konfigurálás idotartamára
    UART0->BDH = 0x00;             // Baudrate = 9600 bps (bit8 - bit12)
    UART0->BDL = 0x88;             // Baudrate = 9600 bps (bit0 - bit7)
    UART0->C4 = 0x0F;              // Túlmintavételezési arány = 16
    UART0->C1 = 0x00;              // 8-bites adatformátum
    UART0->C2 = 0x04;              // Adatfogadás engedélyezése
    SIM->SCGC5 |= 0x0200;          // PORTA engedélyezése
    PORTA->PCR[1] = 0x0200;        // PTA1 legyen UART0_Rx üzemmódban
}

void LED_init(void) {
    SIM->SCGC5 |= 0x400;           // Port B engedélyezése
    SIM->SCGC5 |= 0x1000;          // Port D engedélyezése
    PORTB->PCR[18] = 0x100;        // PTB18 legyen GPIO (MUX = 1)
    PTB->PDDR |= 0x40000;          // PTB18 legyen kimenet (bit18)
    PTB->PSOR = 0x40000;           // Piros LED lekapcsolása (negatív logika!)
    PORTB->PCR[19] = 0x100;        // PTB19 legyen GPIO (MUX = 1)
    PTB->PDDR |= 0x80000;          // PTB19 legyen kimenet (bit19=1)
    PTB->PSOR = 0x80000;           // Zöld LED lekapcsolása (negatív logika!)
    PORTD->PCR[1] = 0x100;         // PTD1 legyen GPIO  (MUX = 1)
    PTD->PDDR |= 0x02;             // PTD1 legyen GPIO (MUX = 1)
    PTD->PSOR = 0x02;              // Kék LED lekapcsolása (negatív logika!)
}
```

USB – UART átalakító

A mai laptopokon már nem találunk RS-232 portot. Akkor hogyan kommunikáljunk az UART eszközökkel?



USB - UART átalakító

- A PC felé USB eszköz (virtuális soros port)
- Logikai szint (3.3V, vagy 5V) a mikrovezérlő UART portja felől nézve (nem RS232 jelszinteket használ!)

FT232 USB - soros átalakító

- Választható jelszint (3,3V vagy 5V)
- „Modemvezérlő” jelek (pl. DTR, RTS)
- Tápfeszültséget is ad 5 V, 3.3 V az USB buszról
- RX vonalon küldhetünk adatot a PC felé
- TX vonalon érkezik, amit a PC küld
- Az adatsebesség a PC alkalmazásban állítható (PUTTy, Termite, TeraTerm, HyperTerminal stb.)

Program4_3: UART2 egyszerű adatküldés

"Hello" szöveget küldünk folyamatosan a **FRDM-KL25Z** kártya **UART2 Tx (PTE22)** kimenetén keresztül. Használjunk egy 3,3 V jelszintű USB-soros átalakítót az üzenetek PC-re küldéséhez!

A konfigurálás lépései:

1. Engedélyezzük az **UART2** modult a **SIM_SCGC4** regiszter 12. bitjének 1-be állításával.
2. Az **UART2** Tx/Rx letiltása konfigurálás előtt: **UART2_C2** regiszter TE=0, RE = 0.
3. Az **UART2** bitsebesség konfigurálása **UART2_BDH** és **UART2_BDL** beírásával.
4. Adatformátum konfigurálása: 1 stop bit, nincs paritás vizsgálat, 8-bites adat megadásához írjunk 0x00-t az **UART2_C1** vezérlő regiszterbe!
5. Engedélyezzük **UART2** adóját: írjunk 0x08-at az **UART2_C2** regiszterbe (**TE** bit = 1)!
6. Engedélyezzük az **E** portot: a **SIM_SCGC5** regiszter 13. bitje legyen 1!
7. Válasszuk ki a **PTE22** kivezetés Alt4 funkcióját (UART2_Tx) **PORTE_PCR22** MUX=4.

Az adatküldés lépései:

8. Vizsgáljuk a Status Register 1 (**UART2_S1**) **TDRE** bitjét és várjunk addig, amíg a küldő buffer üressé válik (TDRE = 1)!
9. Írjuk bele a kiküldendő bájtot az **UART0_D** adatregiszterbe!
10. Újabb karakter kiküldéséhez menjünk a 8. ponthoz!

Program4_3/1.

A program forrása: Mazidi et al., Freescale ARM Cortex-M Embedded Programming
http://www.microdigitaled.com/ARM/Freescale_ARM/Code/Ver1/Chapter4/Program4_3.txt

Változtatások:

- **BDH:BDL** újraszámolása az eltérő frekvencia miatt,
- **PTE22** kimenet használata

```
#include <MKL25Z4.h>

void UART2_init(void);                      // UART2 kimenet inicializálása
void delayMs(int n);                        // Késlelteto függvény

int main (void) {
    char message[] = "Hello\r\n";           // Ez lesz a kiírandó szöveg
    int i;                                  // Ciklusváltozó
    UART2_init();
    while (1) {
        for (i = 0; i < 7; i++) {
            while(!(UART2->S1 & 0x80)); // Várunk, míg kiürül a kimeneti buffer!
            UART2->D = message[i];          // Következo karakter küldése

        }
        delayMs(2000);                      // Várunk egy kicsit...
    }
}
```

Folytatás a következő oldalon ...

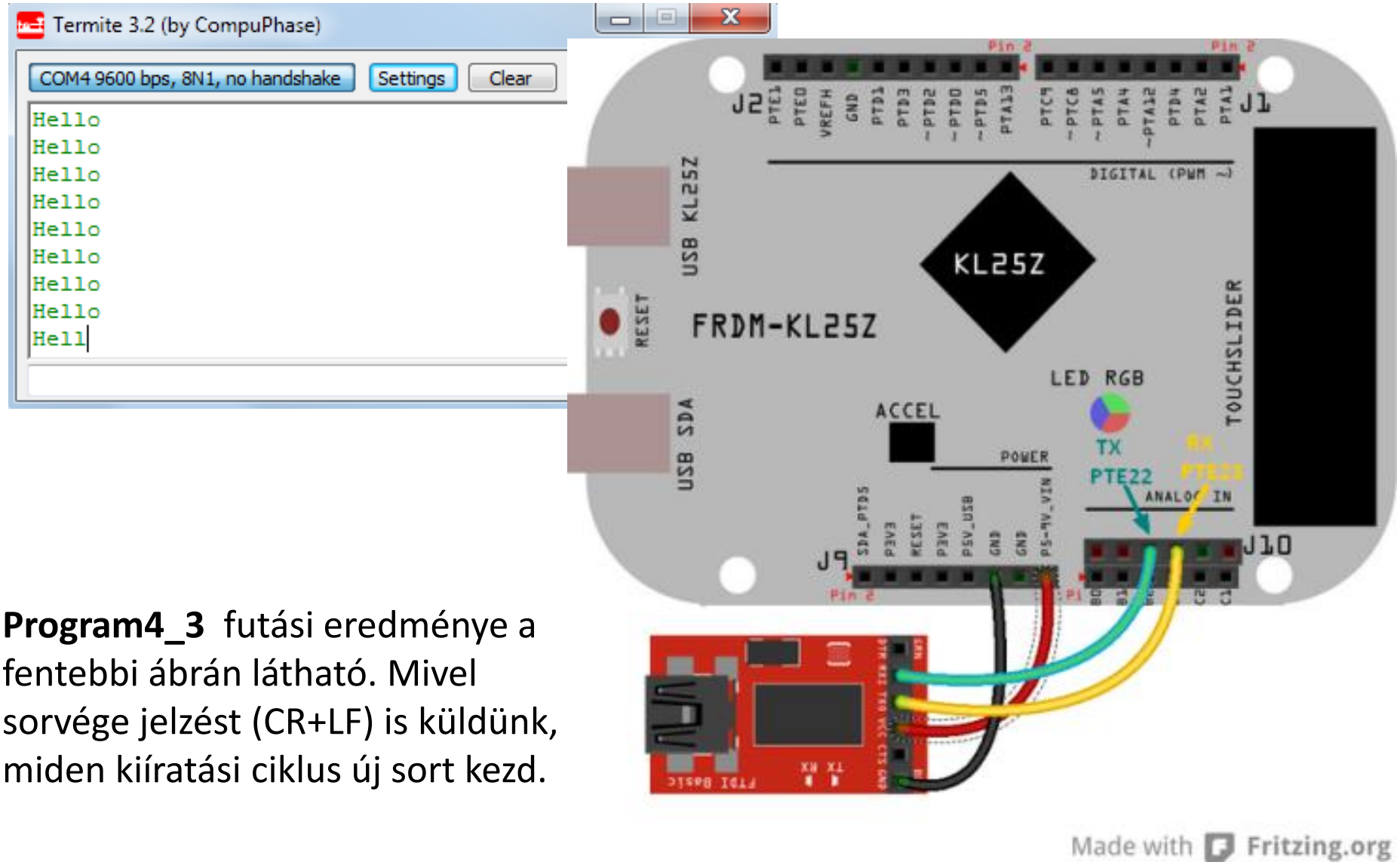
Program4_3/2.

```
/* UART2 Tx (PTE22) inicializálása 9600 Baud átviteli sebességre
 * Baudrate = bus freq/16/BDH:BDL = 10.49 MHz/16/0x00:0x44 =  9637 bps,
 */
void UART2_init(void) {
    SIM->SCGC4 |= 0x1000;          // UART2 periféria engedélyezése
    UART2->C2 = 0;                 // UART2 letiltása a konfigurálás időtartamára
    UART2->BDH = 0x00;             // Baudrate = 9600 bps (bit8 - bit12)
    UART2->BDL = 0x44;             // Baudrate = 9600 bps (bit0 - bit7)
    UART2->C1 = 0x00;              // 8-bites adatformátum
    UART2->C3 = 0x00;              // Nincs hibainterrupt
    UART2->C2 = 0x08;              // Adatküldés engedélyezése

    SIM->SCGC5 |= 0x2000;          // PORTE engedélyezése (bit 13)
    PORTE->PCR[22] = 0x0400;       // PTE22 legyen UART2_Tx üzemmódban (Alt4)
}
//Színezéssel az UART2 TX specifikus részeket jelöltük meg


void delayMs(int n) {              // Késleltető függvény
    int i, j;
    for(i = 0 ; i < n; i++)
        for (j = 0; j < 3500; j++); // Alapértelmezett órajelhez (20.97152 MHz)
}
```

Bekötési vázlat, futási eredmény



Program4_3 futási eredménye a fentebbi ábrán látható. Mivel sorvége jelzést (CR+LF) is küldünk, miden kiíratási ciklus új sort kezd.

Program4_4/1.

```
/* Ez a program egy karaktert fogad az UART2 port bemenetén, majd eggyel
 * megnövelve, visszaírja azt az UART2 port kimenetére.
 * Használjunk egy USB-soros átalakítót a kapcsolódáshoz és egy terminál emulátor
 * programot az üzenetek megjelenítéséhez! Beállítás: 9600 bps, 8N1, no handshake.
 * Kivezetések: UART2 Tx a PTE22, UART Rx pedig a PTE23 lábra konfigurálva
 */

#include <MKL25Z4.h>

void UART2_init(void);                          // UART2 kimenet inicializálása

int main (void) {
    char c;
    UART2_init();
    while (1) {
        while(!(UART2->S1 & 0x20));             // Karakter beérkezésre várunk
        c = UART2->D ;                          // Beolvassuk a vett karaktert
        while(!(UART2->S1 & 0x80));             // Várunk míg kiürül a küldo buffer
        UART2->D = c+1;                         // Kiküldjük az eggyel megnövelt kódot
    }
}
```

Folytatás a következő oldalon ...

Program4_4/2.

```
/* UART2 Tx (PTE22) inicializálása 9600 Baud átviteli sebességre
 * Baudrate = bus freq/16/BDH:BDL, Esetünkben 10.49 MHz/16/0x00:0x44 = 9637 bps,
 */
void UART2_init(void) {
    SIM->SCGC4 |= 0x1000;          // UART2 periféria engedélyezése
    UART2->C2 = 0;                 // UART2 letiltása a konfigurálás idotartamára
    UART2->BDH = 0x00;             // Baudrate = 9600 bps (bit8 - bit12)
    UART2->BDL = 0x44;             // Baudrate = 9600 bps (bit0 - bit7)
    UART2->C1 = 0x00;              // 8-bites adatformátum
    UART2->C3 = 0x00;              // Nincs hibainterrupt
    UART2->C2 = 0x0C;              // Adatküldés és fogadás engedélyezése

    SIM->SCGC5 |= 0x2000;          // PORTE engedélyezése (bit 13)
    PORTE->PCR[22] = 0x0400;       // PTE22 legyen UART2_Tx üzemmódban (Alt4)
    PORTE->PCR[23] = 0x0400;       // PTE23 legyen UART2_Rx üzemmódban (Alt4)
}
    // Színezéssel jelölve: változások a Program4_3-hoz képest
```

Megjegyzések:

- Az adatküldő terminál emulátor programot úgy konfiguráljuk, hogy a begépelt karakter(ek)hez elküldéskor ne fűzzön hozzá semmilyen sorvége jelet!
- Ha egyszerre több karaktert küldünk, a vevő lefulladhat (ráfutási hiba?). Mivel a programunkban a hiba nincs lekezelve, az első hiba után az UART modul újraindulásig nem tud fogadni további karaktereket.

Printf() kimenet átirányítása

Joseph Yiu: **„The Definitive Guide to ARM® Cortex®-M0 and Cortex-M0+ Processors”** c. könyvének egyik mintapéldája (18_2_uart_retarget) alapján bemutatjuk, hogy a stdio printf()/scanf() ki- és bemenetet hogyan irányíthatjuk át az UART0 portra. Az eredeti programot leegyszerűsítettük (kiszedve a LED vezérlést) és a kimenetet némiképp átalakítottuk (kiíratjuk a vett karakter betűképét és ANSI kódját is).

Az átirányításra általában két lehetőség kínálkozik:

Retargeting: néhány alacsony szintű I/O függvény felüldefiniálásával a kívánt kimenetre (UART vagy LCD) irányithatjuk a ki/bemenetet. Mi most ezt próbáljuk ki.

Semihosting: Némelyik fejlesztői eszköz és környezet lehetővé teszi, hogy a stdio kimenet a hardveres nyomkövető egység legyen, s azon keresztül történhet adat ki- és bevitel. Ez a lehetőség azonban nem minden fejlesztőeszköznél áll rendelkezésre, viszonylag lassú, és csak debug módban használható.

Az első módszerhez a **Keil MDK5** esetén a **fputc(), fgetc()** és **ferror()** függvényeket kell felüldefiniálni. A szükséges kódrészeket a mintaprojekt **retarget.c** állomány tartalmazza.

Az **UART0** kezeléséhez szükséges függvényeket (**UART_config(), UART_putc(), UART_getc(), UART_echo(), UART_puts()**) az **uart_funcs.c** állomány tartalmazza. Ennek ismertetésére itt nem térünk ki, hasonlóan lekérdezéssel működik, mint az előbb bemutatott programok.

Retarget.c

```
#include <stdio.h>
#include <time.h>
#include <rt_misc.h>
#pragma import(__use_no_semihosting_swi)
extern char UART_putc(char ch);
extern char UART_getc(void);
struct __FILE { int handle; /* Add whatever you need here */ };
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE *f) {
  if (ch == 10) UART_putc(13);
  return (UART_putc(ch));
}
int fgetc(FILE *f) {
  return (UART_putc(UART_getc()));
}
int ferror(FILE *f) {
  /* Your implementation of ferror */
  return EOF;
}
void _ttywrch(int ch) {
  UART_putc(ch);
}
void _sys_exit(int return_code) {
label:  goto label;  /* endless loop */
}
```

Az UART_Retarget program

```
#include <MKL25Z4.H>
#include "stdio.h"
// System runs at 48MHz
// Baud rate 38400, 8 bit data, no parity, 1 stop bit

// UART functions
extern void UART_config(void);
extern char UART_putc(char ch);
extern char UART_getc(void);
extern void UART_echo(void);
extern void UART_puts(char * mytext);

int main(void) {
  char txt_buf[100];
        char c;
  SystemCoreClockUpdate();
  UART_config();
  printf("\r\nWelcome to FRDM-KL25Z board!\r\n");
  printf("Please enter your name:");
  scanf("%99s", txt_buf);
  printf("\nName entered :[%s]\n", txt_buf);
  while(1){
      c = UART_getc(); //Read one character
      printf("received char: %c = %d\r\n",c,c);
  }
}
```

A 48 MHz-es órajel beállításához az 1. sorszámú előre definált beállítást kell kiválasztanunk. Ehhez a projekt lefordítása előtt a **system_MKL25Z4.h** állományba be kell szúrnunk az alábbi sort:

```
#define CLOCK_SETUP 1
```

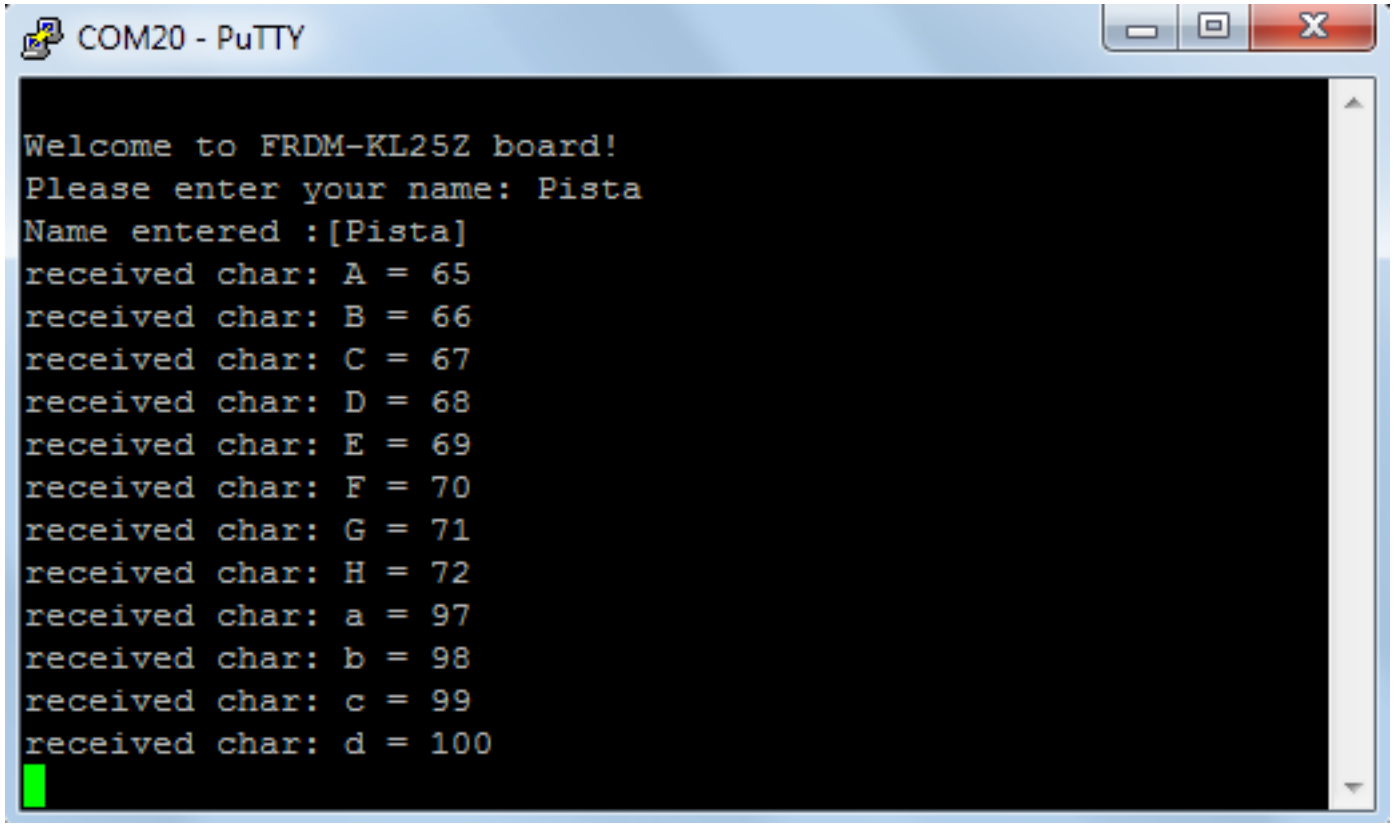
Ekkor a CPU órajele 48 MHz lesz (külső kvarccal), a buszfrekvencia pedig 24 MHz lesz

Futási eredmény

A program először egy nevet vár (a beolvasást a scanf() függvény kezeli), ennek lezárására egy szóközt kell nyomni. Ezután minden begépelt karakternek kiíratjuk a betűképét és az ANSI kódját.

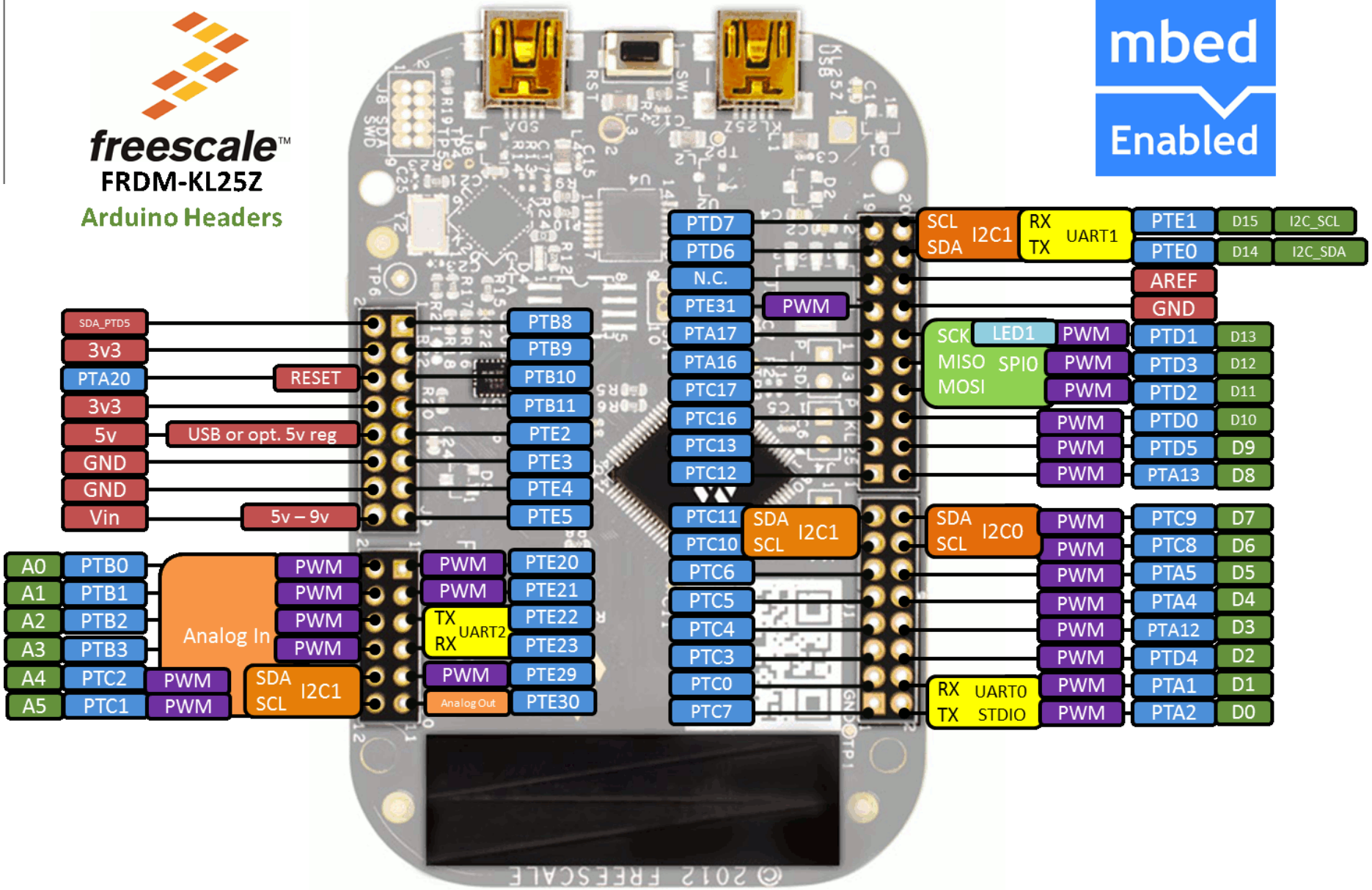
Ha egyszerre 2-3 karakternél többet küldünk, a vevő lefullad (ráfutási hiba?). Mivel a programunkban a hiba nincs lekezelve, az UART modul újraindulásig nem fogad újabb karaktert.

Az adatsebességet ennél a programnál 38400 bps-re kell állítani!



freescale™
FRDM-KL25Z
Arduino Headers

mbed Enabled



freescale™

FRDM-KL25Z

Additional Peripherals