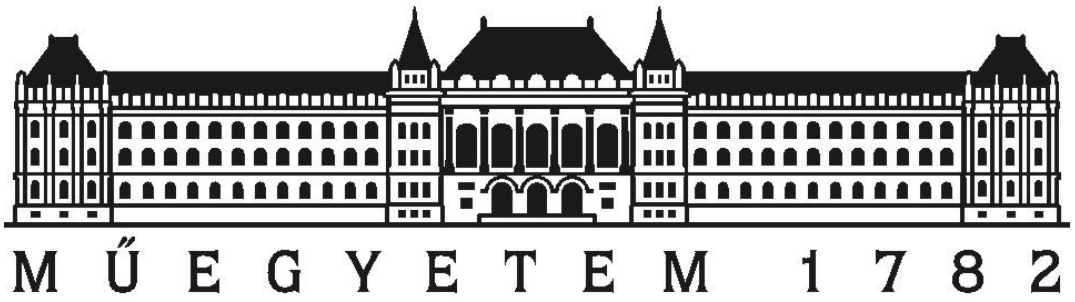


Gekko CLI

STK3700 fejlesztőkártyával megvalósított soros porti parancssori értelmező

Seres Zsófia

Sebők Bence

Beágyazott és irányító rendszerek specializáció

Beágyazott információs rendszerek ágazat (MIT)

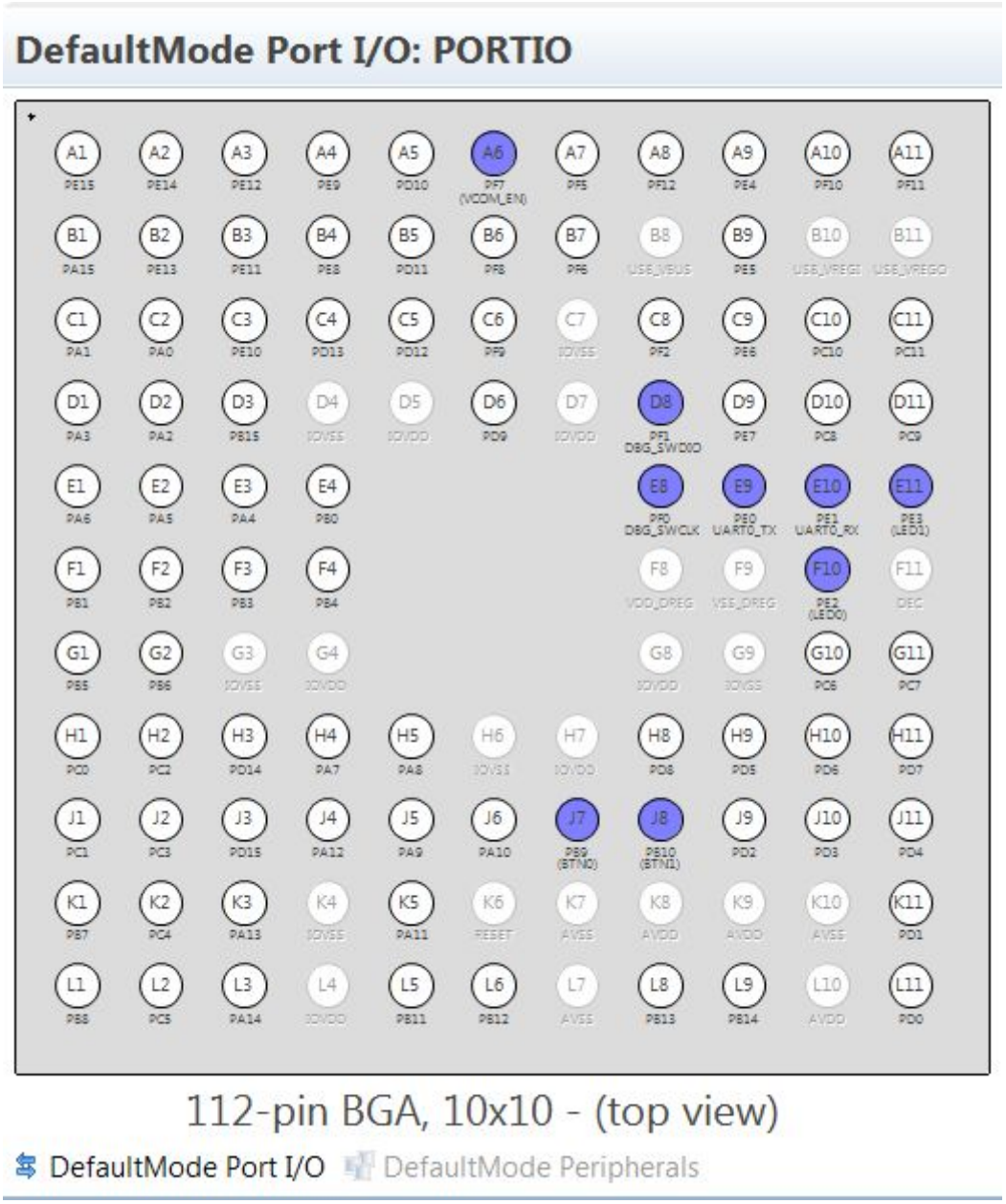
Beágyazott és ambiens rendszerek (VIMIAC06) házi feladat

gekko_cli dokumentáció

Tartalomjegyzék

Hardver

A főbb perifériákat és GPIO-kat a **Hardware Configurator** segítségével állítottuk be. A konfigurációt a **gekko_cli.hwconf** fájl tartalmazza.
A konfiguráció grafikusan:



LED-ek

Mindkét felhasználói LED (LED0, LED1) **push-pull** pin módban van. Default output értékük 0.
A LED0 a PE2-n van, a LED1 pedig a PE3-on.

Properties of PE2

Port Pin

Property	Value
Settings	
Pin mode	Push-pull
Data output	0
Custom pin name	LED0
Reserve	Not reserved

UARTo

A Communications részből az UART0-át használjuk.
Az UART0 feladata a fejlesztőkártya és a PC-s soros porti parancssor közötti adatcsere.

Properties of UART0

UART 0

Property	Value
Asynchronous Settings	
Baudrate	115200 (0x1C200)
Databits in frame	8
Parity bits	No parity
Stop bits	1 stopbit
Oversampling	16x
Majority vote	Enabled
PRS for USART Rx	Disabled
Selected channel state	PRS input not enabled
PRS triggering settings	
Enable RX triggering through PRS	Disabled
Enable TX triggering through PRS	Disabled
Triggering channel	Channel 0
Selected channel state	PRS peripheral is not en...

UART0 az 1-es útvonalon került engedélyezésre, vagyis az **RX pin a PE1 pin**, a **TX pin pedig a PE0 pin**.

A PE1 pin az RX soros porti bemeneti vonal, emiatt a pin mód: **Input**.

A PE0 pin a TX kimeneti vonal, ezért a Pin Mode ennél: **push-pull**. Mivel magas-aktív kommunikáció az USART, ezért a PE0-nál a Data Output default kimeneti érték: 1.

Timer

A Timers részből a TIMER0-át használjuk.
A TIMER0-át a képernyőn futó szöveg másodpercenkénti léptetésére használjuk.

Properties of TIMER0

Timer 0

Property	Value
Settings	
Enable	Run timer after initialization
Run while core is halted	Do not run timer when core is halted
Sync to other timers	Timer is not started/stopped/reloaded by other timers
Clear DMA request during transfe	Do not clear the DMA request when a transfer is active
Clock settings	
Clock selection	Prescaled HFPER clock
HFPER prescaler	No division
Timer behavior	
One shot	Continuous timer
Counting mode	Up-counting
Quadrature mode	X2 quadrature mode
Action on falling edge (CC0)	No action
Action on rising edge (CC0)	No action
Always track input polarity with C	Do not track polarity in CCMODE 0
x2 counting mode	x1 counting mode

Soros porti parancssor

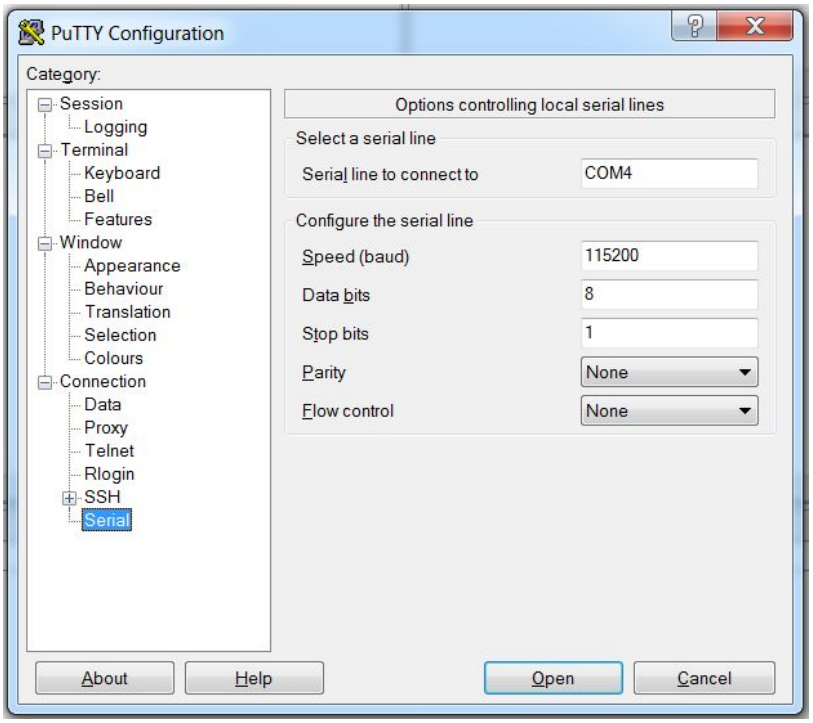
A soros porti kommunikációra a **Putty** nevű programot használtuk a PC-n. Ez egy ingyenesen elérhető program, amit az alábbi weboldalról lehet letölteni:

https://www.putty.org

Putty beállítása

UART kommunikáció

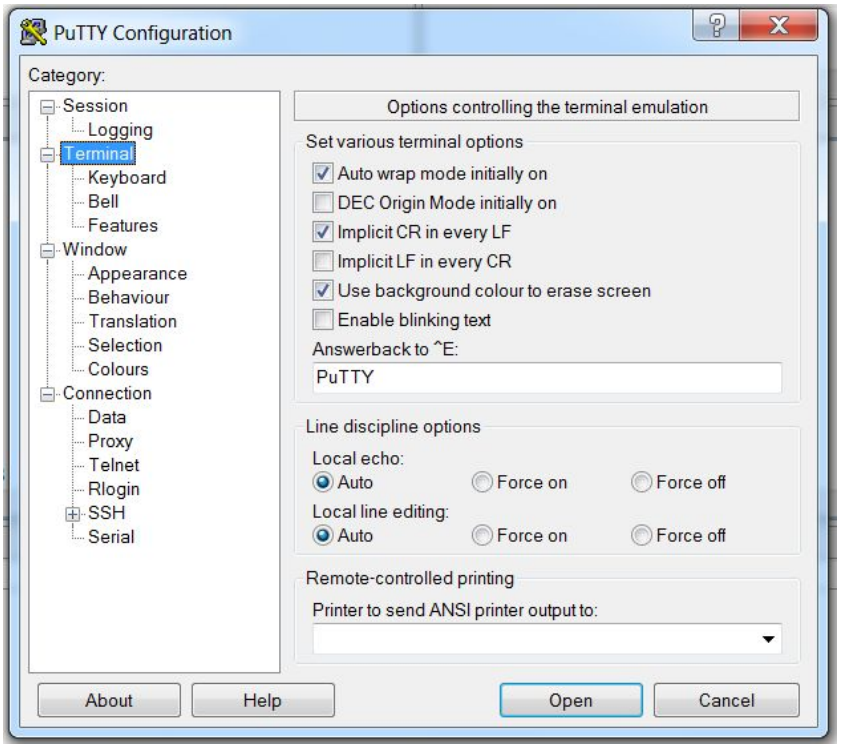
Az UART kommunikáció megfelelő paramétereit beírtuk a Putty **SSH - Serial** menüpontjában: **8N1 @ 115200 baud**
A Gekkó kártya a COM4-es virtuális soros porton érhető el.



CR, LF helyes megjelenítése

A Putty felületén a megfelelő újsor karakter kezelés beállítását a **Terminal** menüpontban állítottuk be:

- **Implicit CR in every LF. pipa**
- **Implicit LF in every CR: nincs pipa**



Inicializáció

Gekko_Init()

- initDevice.c fájlban van kifejtve.
- Hardver és szoftver inicializáció
    - Könyvtári függvényekkel hardver inicializációja
    - UART0 interrupt beállítása
    - TIMER0 és interruptjának beállítása
    - LED-ek ne világítsanak induláskor (LED0, LED1)
    - LCD kijelző bekapcsolása és üdvözlő üzenet kiírása
    - Parancssori felület előkészítése (kezdő karakterek kiküldése UART-on)

Perifériák

LED-ek

LED0 és LED1 felhasználói LED-ek használata a specifikáció szerint.

LCD kijelző

A fejlesztői kártyán található LCD kijelzőn különféle információkat jelenít meg a program, többek között:

- üdvözlő üzenet induláskor (*CLI*)
- LED-ek aktuális értéke lekérdező parancs esetén (GETLED0, GETLED1)
- Futó szöveg megjelenítése a megfelelő parancs és üzenet esetén (WRITETEXT *szöveg*)

UART0

Paraméterek

- UART0 beállítása röviden: 8N1
- baud rate: 115200
- adatbitek száma: 8 bit
- paritás: nincs paritás
- stop bitek száma: 1 bit
- túlmintavételezés: 16-szoros

Interrupt

USART_IF_RXDATAV

IT Handler függvény
Minden egyes **RXDATAV megszakítás** esetén a **globális ch karakter**be bemásoljuk az UART0-án érkező karaktert.
Beállítja a **new_char** flaget **true** értékre, amivel jelezzük az állapotgépnek, hogy új karakter érkezett.
Ha az újonnan érkezett karakter az újsor karakter (**END_CHAR**), akkor lezárjuk az üzenetet (lezáró nullával) és továbbítjuk feldolgozásra a kapott parancsot.

A **receivedMessage** flag **true** értékre állításával jelezzük az állapotgépnek, hogy lezárult egy üzenet és elkezdődhet a feldolgozása.
Ha még nem az újsor karakter érkezett, akkor eltároljuk az új karaktert is az eddigi üzenetet tartalmazó tömbbe.

```
void UART0_RX_IRQHandler(void)
{
    ch = USART_RxDataGet(UART0);
    new_char = true;

    // Ha az érkezett karakter az üzenet vége jel, akkor vége egy üzenetnek és dolgozzuk fel (feldolgozás flag set)
    if(ch == END_CHAR)
    {
        message[messageSize++] = '\0';
        receivedMessage = true; // Flag beállítása, hogy a main-ben feldolgozzuk az üzenetet.
    }
    // Ha még nem jött az üzenet vége jel, akkor tároljuk el az új karaktert az üzenetben.
    else
    {
        // A messageSize változó mindig az új karakter indexelésére alkalmas.
        message[messageSize] = ch;
        messageSize++; // Jött egy új karakter, növeljük az üzenet hosszát jelentő változót.
    }
    //USART_IntClear(UART0, USART_IF_RXDATAV);
}
```

TIMER0

Paraméterek

- Top érték: 1000

Interrupt

TIMER0 Overflow interruptot használ a program másodperc számlálás céljából.
Az eltelt milliszekundomokat számoljuk egy **uint16_t ms_counter** változóval.

IT Handler függvény
Minden egyes megszakítás esetén növeljük a milliszekundom számlálá értékét eggyel, majd töröljük az érvényre jutott megszakításhoz tartozó flaget.

```
void TIMER0_IRQHandler(void)
{
  ms_counter++;                          // Increment counter
  TIMER_IntClear(TIMER0, TIMER_IF_OF);   // Clear overflow flag
}
```

Üzenetkezelés

Paraméterek

A constants.h fájlban:

- Üzenet maximális hossza: MESSAGE_MAX_SIZE
- Begépelt parancs maximális hossza: COMMAND_MAX_SIZE
- Kijelzőn egyszerre megjeleníthető karakterek száma: KIJELZO_MERET

- Értelmezett parancsok:

```
#define HELP "Help" // Elérhető parancsok lekérdezéséhez tartozó üzenet.
#define LED0BE "Set LED 0 1" // LED0 bekapcsolásához tartozó üzenet.
#define LED0KI "Set LED 0 0" // LED0 kikapcsolásához tartozó üzenet.
#define LED1BE "Set LED 1 1" // LED1 kikapcsolásához tartozó üzenet.
#define LED1KI "Set LED 1 0" // LED1 kikapcsolásához tartozó üzenet.
#define GETLED0 "Get LED 0" // LED0 lekérdezéséhez tartozó üzenet.
#define GETLED1 "Get LED 1" // LED1 lekérdezéséhez tartozó üzenet.
#define WRITETEXT "Write Text" // Kijelzőn futó szöveghez tartozó üzenet.
```

Üzenet feldolgozása

processCommand(char * message)

- A message.c fájlban van kifejtve.
- Egy elküldött üzenet feldolgozása és a megfelelő tevékenység elvégzése.
- Feldolgozás: ha az üzenet tartalma...
    - **HELP** sztringgel megegyezik, akkor a parancsok() függvénnyel az elérhető parancsok listázása a parancssorban.
    - **LED0BE** sztringgel megegyezik, akkor a LED0 bekapcsolása a GPIO_PinOutSet(port, pin) függvénnyel.
    - **LED1BE** sztringgel megegyezik, akkor a LED1 bekapcsolása a GPIO_PinOutSet(port, pin) függvénnyel.
    - **LED0KI** sztringgel megegyezik, akkor a LED0 kikapcsolása a GPIO_PinOutClear(port, pin) függvénnyel.
    - **LED1KI** sztringgel megegyezik, akkor a LED1 kikapcsolása a GPIO_PinOutSet(port, pin) függvénnyel.
    - **GETLED0** sztringgel megegyezik, akkor a LED0 aktuális értékének lekérdezése a GPIO_PinOutGet(port, pin) függvénnyel és az érték kiírása az LCD kijelzőre, valamint üzenet küldése a parancssorra.

```
else if(strcmp(message, GETLED0) == 0) // Ha az üzenet a LED0 lekérdezése, akkor ...
{
    int value = GPIO_PinOutGet(LED0_PORT, LED0_PIN); // kérdezzük le a LED0-et.
    USART_Tx(UART0, '\n');
    if(value == 1)
    {
        string2USART("LED0 vilagit");
        SegmentLCD_Write("LED0 1");
    }
    else
    {
        string2USART("LED0 nem vilagit");
        SegmentLCD_Write("LED0 0");
    }
    SegmentLCD_Number(value);
}
```

    - **GETLED1** sztringgel megegyezik, akkor a LED1 aktuális értékének lekérdezése a GPIO_PinOutGet(port, pin) függvénnyel és az érték kiírása az LCD kijelzőre, valamint üzenet küldése a parancssorra.

GETLED0 és GETLED1 értékének kijelzése:

- Ha a LED0 vagy LED1 éppen világít, akkor az LCD kijelzőre kiírja a program, hogy *LED0 1* vagy *LED1 1*, valamint kiküldi a parancssorra, hogy *LED0 vilagit* vagy *LED1 vilagit.*

- Ha a LED0 vagy LED1 éppen nem világít, akkor az LCD kijelzőre kiírja a program, hogy *LED0 0* vagy *LED1 0*, valamint kiküldi a parancssorra, hogy *LED0 nem vilagit* vagy *LED1 nem vilagit.*
- Aktuális érték kijelzése az LCD kijelző Number részén:
    - 1: világít
    - 0: nem világít

- Ha az üzenet a fentiek egyikével sem egyezik meg, akkor két lehetőség maradt:
    - WRITETEXT üzenet érkezett vagy
    - érvénytelen, nem értelmezhető üzenet érkezett: ha a kapott üzenet hossza rövidebb, mint a WRITETEXT parancs hossza, akkor a kapott üzenet nem értelmezett és kiküldi a parancssorra az INVALID üzenetet, illetve kiír egy 8-(szmájlit az LCD kijelzőre.
        - A :-(nem jelenik meg szépen a kijelzőn, de a napszemüveges verzió (8-( ) már jól néz ki.

Futó szöveg (WRITETEXT)

Ha a kapott üzenet hossza hosszabb, mint a WRITETEXT parancs hossza, akkor eldönti a program, hogy érvényes futó szöveg parancsot kapott-e:

1. Az üzenet elejét bemásolja egy segéd tömbbe, hogy eldöntse, a kapott üzenet eleje megegyezik-e a WRITETEXT parancs szövegével
2. Ha nem egyezik meg, akkor érvénytelen üzenet érkezett. Ha megegyezik, akkor folytatja a 3. lépéssel
3. A WRITETEXT parancs szövege után a megjelenítendő szöveg található az újsor karakterig. Ezt a szöveget bemásolja egy segéd tömbbe, hogy kitegye az LCD kijelzőre.
4. UART0-án kiküldi a teljes üzenetet (parancs + szöveg) a parancssornak.
5. writingText flaget true értékre állítja, hogy az állapotgép megfelelő állapotába lépjünk bele ezután a ciklus további részében.
6. step = 0 értékadással a megjelenítendő szöveg elejét teszi majd ki a kijelzőre

Szöveg léptetése másodpercenként

A rendszer órajele: 14 MHz, vagyis a periódusidő: $\frac{1}{14 \cdot 10^6}$ másodperc, vagyis 71.42 ns. Egy másodperchez közelítőleg $\frac{1}{71.42}$ periódusnak kell eltelnie, ami számérték szerint $1.4 \cdot 10^7$.
Az ms_counter értéke milliszekundumonként nő eggyel. Ha értéke eléri a 14000-et, akkor eltelt egy másodperc és léptetjük a szöveget a step változóval, valamint előről kezdjük a milliszekundumok számolását.
Ha a step változó elérte a maximális értéket, amit felvehet, akkor nullára állítjuk és ezzel újból a szöveg elejét jelenítjük meg. **A step változó maximális értéke**: a megjelenítendő szöveg hosszának és a kijelzőn elférő karakterek számának a különbsége.

```
// 1 másodperc letelt már?
// f = 14 MHz, vagyis T = 71.42 ns
// 1 másodperc = 1.4 * 10^7 ciklus várakozás.
if(ms_counter >= 14000)
{
    // Elértük a szöveg végét?
    if (step >= (strlen(command) - KIJELZO_MERET))
    {
        step = 0; // Kezdjük előről.
    }
    else
    {
        step++; // Folytassuk a következő 7 darab karakterrel.
    }
    ms_counter = 0; // Kezdjük előről a számlálást.
}
```

Felhasználói segédfüggvények

void string2USART(char * string)

A string paramétert kiküldi karakterenként az UART0 periférián a soros porti parancssornak.

```
void string2USART(char * string)
{
    int i;
    for(i = 0; string[i] != '\0'; i++)
    {
            USART_Tx(UART0, string[i]);
    }
}
```

void echoMessage()

Teljes üzenet kiírása a PC-s terminálra UART-on keresztül.

```
void echoMessage()
{
    // Formátum: ÚJ SOR<ÜZENET>
    USART_Tx(UART0, '\n');
    string2USART("Echo:");
    USART_Tx(UART0, '<');
    string2USART(message);
    USART_Tx(UART0, '>');
    USART_Tx(UART0, '\n');
}
```

void parancsok(void)

A HELP parancs esetén ezzel a függvénnyel írja ki a program a parancssorba az elérhető parancsok listtáját.

```
void parancsok(void)
{
    USART_Tx(UART0, '\n');
    string2USART("Parancsok:"); USART_Tx(UART0, '\n');
    string2USART(WRITETEXT); USART_Tx(UART0, '\n');
    string2USART(HELP); USART_Tx(UART0, '\n');
    string2USART(LED0BE); USART_Tx(UART0, '\n');
    string2USART(LED0KI); USART_Tx(UART0, '\n');
    string2USART(LED1BE); USART_Tx(UART0, '\n');
    string2USART(LED1KI); USART_Tx(UART0, '\n');
    string2USART(GETLED0); USART_Tx(UART0, '\n');
    string2USART(GETLED1);
}
```

void updateScreen(char * screen, char * string)

A screen globális változó tartalmazza az aktuálisan a kijelzőn szereplő szöveg(részletet).
A string paraméter a teljes szöveget tartalmazza, amit futtat a kijelzőn a program.
A step egy ciklusváltozó, ami az egész szövegen való végigfutást valósítja meg.
A screen tömbbe bemásolja a szövegben következő részletet a memcpy függvénnyel, majd ezt kiírja az LCD kijelzőre.

```
void updateScreen(char * screen, char * string)
{
    memcpy(screen, string + step, KIJELZO_MERET); // step-edik 7 darab karakter másolása az LCD-re.
    screen[KIJELZO_MERET] = '\0'; // Lezárás a sztringgé alakítás miatt.
    SegmentLCD_Write(screen);
}
```