Hogy ne kelljen az LCD programját minden alkalommal félig újraírni.

Mikrovezérlős alkalmazások gyakori kiegészítő hardver eszköze a karakteres LCD kijelző. Ezek kaphatóak 1, 2, 4 soros, soronként 8, 16, 20, 24, 40 karakteres változatban, háttérvilágítással és anélkül is. Közös jellemzőjük, hogy Hitachi HD44780 vagy azzal kompatíbilis vezérlőchipet tartalmaznak (én legalább is még csak ilyennel találkoztam). Ezekhez a kijelzőkhöz készült egy univerzálisan felhasználható program, amelynek felhasználásával mentesülhetünk a kijelző programjának elkészítésével járó vesződséggel. Hogy miért készült? Régebben gyakran előfordult, hogy a kijelző programját az előző munkámból átemelve az nem akart egyből működni. Ilyenkor mindig sok idő ment el azzal amíg megkerestem mi nincs jól beállítva, hol maradtak benne olyan függések amik a régi programhoz kapcsolódtak és már nem aktuálisak. Megunva az ezzel járó gyakori vesződséget megírtam olyanra, amit minden programba könnyedén be tudok illeszteni, és a lehető legkisebb a csatlakozási felület az aktuálisan készülő programunkkal. Mivel egy helyre van összegyűjtve az összes beállítási lehetőség, így nem kell a programban szanaszét található paramétereket és lábhozzárendeléseket keresgélni. A beállításokat is nagyon gyorsan és egyszerűen (majdnem mint egy ikszelős kérdőívet kitöltve) el lehet végezni. A kiírandó szöveg átadása is nagyon egyszerű, mindössze az **LcdText** tömbbe kell berakni azt a szöveget amit a kijelzőn szeretnénk látni. Innen majd a "driver" program (LcdChar) fogja azt a kijelzőre áttölteni. Van olyan üzemmódja a driver-nek, ami az inicializálást leszámítva sehol nem tartalmaz úgynevezett blokkoló típusú függvényt, ezért időkritikus (USB, ethernet stb.) alkalmazásokhoz is használható. Ne ijedjünk meg attól, hogy a forrás igen nagy lett, a kész programba ennek csak a töredéke fog belekerülni (csak a konfigurációs beállítások alapján felhasznált funkciókhoz tartozó kód) a fordító feltételes fordítási lehetőségeit felhasználva. Ezt le is tudjuk ellenőrizni, ha fordítás után megnézzük a disassambly listát pl. az MPLAB-ban. Ha pl. nem használjuk a kurzort (kikommentezzük az LCDCURSOR definíciót), akkor a kurzor megjelenítéséhez készült kód egyszerűen bele sem kerül a programmemóriába, továbbá az ahhoz tartozó változók sem fognak létezni, ezáltal nem foglalnak helyet RAM-ban.

A program képességei a következők:

- A kód néhány 8bites AVR, PIC16Fxxx, PIC18Fxxx, PIC24xxx, DSPICxxx mikrovezérlőkhöz készült. A következő fordítókkal lett tesztelve:
  AVR: AVR studio 4.xx/WinAVR
  PIC16Fxxx: Hi-tech PICC
  PIC18Fxxx: Microchip C18, Hi-tech PICC18
  PIC24xxx/DSPICxxx: Microchip C30
- 1, 2, 4 soros kijelző használata.
- legfeljebb 40 karakter széles kijelző használata.
- 2db kijelző használata (**LCDSTEREO**).
- Dupla vezérlős, pl. 4x40 karakteres, E1, E2 lábat is tartalmazó kijelző működtetése.
- 4 vagy 8 bites vezérlési mód.
- A kijelző vezérlő lábait a mikrovezérlő bármely kimenetként működtethető lábára köthetjük. Nincs semmi megkötés, a lábak összerendelése nagyon egyszerű.
- 5 féle üzemmód.

- Megszakításos, folyamatos frissítési üzemmódban (LCDMODECONTIRQ) a frissítési frekvencia (FPS) megadható **(LCDFRAMEPERSEC**).
- Megszakításos üzemmódban a felhasznált időzítő kiválasztásának lehetősége (olyat adjunk meg, ami létezik a chipben és nem használunk más célra).
- Ha nem BUSY flag figyeléses üzemmódot használunk, az LCD RW lábát GND-re kötve egy I/O láb megspórolható.
- 8 saját tervezésű karakter, inicializáláskor történő feltöltése az LCD-re (ezeket a #0...#7, #8..#15 karakterkódokkal jeleníthetjük meg).
- Futás közben a saját tervezésű karakterek cseréjének a lehetősége.
- Kurzor használata (csak az egyszeri frissítéses üzemmódokban).
- Beállíthatjuk, hogy a #0 kódú karaktereket szóközre cserélje.
- Villogó karakterek használata.
- Megszakításos, folyamatos frissítési üzemmódban (**LCDMODECONTIRQ**) automatikus villogás is használható, ilyenkor a villogási sebesség is megadható (**LCDBLINKSPEED**).

Inicializálás

Az **LcdInit()** függvény inicializálja és bekapcsolja az LCD kijelzőt. Ha saját karaktereket is használunk, az inicializáló függvény azt is feltölti a kijelzőbe. A függvény futási ideje kb. 130msec ideig tart, ezért célszerű a programunk elejére tenni. Ennek természetesen csak egyetlen alkalommal kell lefutnia. Kötelező használni, e nélkül nem fog működni a kijelző. Hibás vagy rosszul bekötött kijelző esetén is le fog futni az inicializálás, nem olvas semmit a kijelzőről vissza, ezért nem tudja a driver sikeres volt-e az inicializálás.

Üzemmódok

A kijelző drivert 5 különböző üzemmódon működtethetjük. Az üzemmód kiválasztása a feladattól függ, mindegyik üzemmódnak vannak lehetőségei és korlátai is, ezek alapján döntsük el melyiket választjuk.

- **LCDMODEONCEBUSY**: Megszakítás nélküli egyszeri frissítési üzemmód BUSY flag figyeléssel. Ilyenkor nem fut állandóan a háttérben a kijelző driver, hanem az **LcdText** tömb feltöltése után szólni kell a drivernek, hogy frissítse a kijelző tartalmát. Ezt az **LcdRefreshAll()** függvényhívással tehetjük meg. A frissítő függvény akkor tér csak vissza, ha a frissítés befejeződött (blokkoló függvény!). Az írási műveletet az LCD BUSY olvasása alapján ütemezi. Figyelem! Ha a kijelző, vagy a bekötése meghibásodna, esetleg az rossz az I/O lábak beállítása, a program végtelen ciklusba kerülhet. Ezt az üzemmódot akkor célszerű választani, ha a programunk nem egy végtelen programhurokban kering, hanem sok kis különálló hurok található benne. A kijelző írási sebessége a legmagasabb lesz, amit a kijelző és a mikrovezérlő sebessége biztosítani tud. Lassabb kijelzőt csatlakoztatva sem maradhatnak ki karakterek, hiszen csak akkor ír a kijelzőre, amikor az képes azt fogadni.
  Lehetőségek: kurzor használata, saját karakterek feltöltése és cseréje, villogtatás (csak felhasználó által megoldott ütemezéssel), két kijelző vagy dupla vezérlős kijelző használata.
  Korlátozások: R/W láb kötelező használata, automatikus villogtatás nem lehetséges.

- **LCDMODEONCEDELAY**: Megszakítás nélküli egyszeri frissítési üzemmód időzítéses írással. Ilyenkor sem fut állandóan a háttérben a kijelző driver, hanem az **LcdText** tömb feltöltése után szólni kell a drivernek, hogy frissítse a kijelző tartalmát. Ezt az **LcdRefreshAll()** függvényhívással tehetjük meg. A frissítő függvény akkor tér csak vissza, ha a frissítés befejeződött (blokkoló függvény!). Ezt az üzemmódot akkor célszerű választani, ha a programunk nem egy végtelen programhurokban kering, hanem sok kis különálló hurok található benne. Az írási műveletet az **LCDEXECUTIONTIME** –ban beállított idő szerinti paramétere szerint ütemezi, a függvény futási ideje hozzávetőleg a karakterek száma * LCDEXECUTIONTIME mikroszekundum ideig tart. Arra ügyeljünk, hogy ne adjunk meg rövidebb időt, mint amit a kijelző tud, mert akkor hiányosan jelenhet meg a szöveg. Erre főleg akkor figyeljünk, ha a fejlesztési környezetben találhatótól eltérő típusú kijelzőkkel is üzembiztos működést szeretnénk biztosítani.
  Lehetőségek: kurzor használata, saját karakterek feltöltése és cseréje, villogtatás (csak felhasználó által megoldott ütemezéssel), két kijelző vagy dupla vezérlős kijelző használata, R/W láb GND-re kötésével egy I/O láb megspórolása.
  Korlátozások: automatikus villogtatás nem lehetséges.

- **LCDMODEONCEIRQ**: Megszakításos egyszeri frissítési üzemmód: Ilyenkor sem fut állandóan a háttérben a kijelző driver, hanem az **LcdText** tömb feltöltése után szólni kell a drivernek, hogy frissítse a kijelző tartalmát. Ezt az **LcdRefreshAll()** függvényhívással tehetjük meg. A függvény csak elindítja a frissítést és azonnal visszatér (nem blokkoló függvény!). A karakterek átvitele időzítő megszakításból fog majd megtörténni. A teljes tartalom átvitele kb. 1/ **LCDFRAMEPERSEC** másodperc idő alatt fog megtörténni. Ha ez megtörtént, leállítja az időzítőt. Ezt az üzemmódot akkor célszerű választani, ha a programunk nem egy végtelen programhurokban kering, hanem sok kis különálló hurok található benne, továbbá nem szeretnénk a programunkat megakasztani a kijelzőre történő szöveg kiírásának idejére. Ha nem fejeződött be a kijelző teljes tartalmának kiírása, és ismét meghívjuk **LcdRefreshAll()** függvényt, elölről fog kezdődni az átvitel. Ez önmagában még nem okoz gondot, de ha folyamatosan félbeszakítjuk előfordulhat, hogy csak az első néhány karakter kerül át a kijelzőre. Ilyenkor el kell gondolkoznunk azon, hogy más üzemmódot válasszunk.
  Lehetőségek: kurzor használata, saját karakterek feltöltése és cseréje, villogtatás (csak felhasználó által megoldott ütemezéssel), két kijelző vagy dupla vezérlős kijelző használata, R/W láb GND-re kötésével egy I/O láb megspórolása.
  Korlátozások: automatikus villogtatás nem lehetséges.

- **LCDMODECONTBUSY:** Megszakítás nélküli folyamatos frissítési üzemmód. Olyan programban célszerű használni, ahol a program folyamatosan egy (vagy néhány) végtelen hurokban kering. A hur(k)okba be kell iktatni egy **LcdUpdateChar()** függvényhívást. Ez egy karaktert visz át a kijelzőbe, de csak akkor, ha a kijelző BUSY FLAG kiolvasása alapján ez lehetséges, foglaltság esetén azonnal visszalép a függvényből. A karakter kiírása után lépteti az **LcdText** tömb indexét, így a következő futáskor a következő karaktert írja majd ki. R/W lábat is kötelezően be kell kötni, hiszen a BUSY FLAG olvasása csak így lehetséges.
  Lehetőségek: saját karakterek feltöltése és cseréje, villogtatás (csak felhasználó által megoldott ütemezéssel), két kijelző vagy dupla vezérlős kijelző használata.
  Korlátozások: kurzor nem használható, R/W láb kötelező használata, automatikus villogtatás nem lehetséges.

- **LCDMODECONTIRQ**: Megszakításos folyamatos frissítési üzemmód. A kijelző frissítése időzítő megszakításból folyamatosan történik, semmi más dolgunk nincs, csak az **LcdText** -be a megfelelő szöveg elhelyezése. Időzítéskritikus feladat végrehajtásakor lehetőség van a frissítést átmenetileg szüneteltetni -

LcdRefreshStop(), majd folytatni - **LcdRefreshStart()** . Szüneteltetés alatt a villogás is szünetelni fog. Ezt az üzemmódot bármilyen felépítésű programnál használhatjuk, hiszen a kijelző működtetése teljesen függetlenül a programunktól, megszakításból történik. Automatikus villogtatást is csak ebben az üzemmódban van lehetőségünk használni. Hátránya, hogy néhány százalék processzoridőt igényel a folyamatos frissítés, bár ha nem változik a megjelenítendő szöveg, a fentebb leírt módon tudjuk szüneteltetni a kijelző folyamatos frissítését. A szüneteltetés alatt természetesen semmi processzoridőt nem használ a driver.
Lehetőségek: saját karakterek feltöltése és cseréje, automatikus villogtatás (beállítható sebességgel), két kijelző vagy dupla vezérlős kijelző használata.
Korlátozások: kurzor nem használható.

Használatának módja:

- Hozzunk létre egy új projectet, a projecthez adjuk hozzá a **charlcd.c**-t.
- Hozzuk létre a főprogramunkat a „main” függvénnyel (ne a charlcd.c-ben).
- Az imént létrehozott kódunk elejére illesszük be a következőt: **#include "charlcd.h"**
- A **charlcd.h**-ben állítsuk be a kívánt paramétereket (ennek részleteiről később lesz szó).
- A main függvényünk elejére tegyünk be egy **LcdInit();** függvényhívást. Ha valamelyik LCD-hez használt láb bekapcsolás után A/D bemenetként funkcionál, azt mindenképp az **LcdInit()** hívása előtt állítsuk át digitális I/O-ra!
- A megjelenítendő szöveget **LcdChar[]** tömb megfelelő pozíciójába tegyük bele.
A program 1 karakterrel nagyobb helyet foglal a szükségesnél, hogy az esetleges lezáró #0 is beleférjen. Arra nagyon vigyázzunk, hogy ezt a tömböt ne írjuk túl!
- Megszakításos üzemmódban PIC16, PIC18-at használva az interrupt kiszolgáló függvénybe be kell szúrni az **LcdIntProcess();** függvényt. PIC18 esetében ezt az **LCDTIMERPR18** által kiválasztott prioritáshoz illesszük be!. Ez azért szükséges, mert ezeknek csak 1, illetve 2 közös megszakításvektora van.
- Ha megszakítás nélküli folyamatos frissítési üzemmódot használunk **(LCDMODECONTBUSY**), a fő programhurokba szúrjunk be egy **LcdProcess();** függvényt. Az fontos, hogy a függvény rendszeresen (legalább néhány milliszekundum sűrűséggel) meghívásra kerüljön, ellenkező esetben a kijelző tartalma nem, vagy nagyon lassan fogja követni az **LcdChar** tömb tartalmának változásait. Ha ez nem biztosítható, válasszunk más üzemmódot.
- **Egyszeri frissítési üzemmódokban** szólni kell az Lcd driver programnak, hogy változott a megjelenítendő szöveg, ugyan másolja már át a szöveget a kijelzőbe. Ezt az **LcdRefreshAll()** függvény meghívásával tudjuk megtenni.

A konfiguráció beállítása a charlcd.h –ban (az üzemmódokból csak egyet válasszunk ki!**):**

#define LCDMODEONCEBUSY
Egyszeri frissítési üzemmód megszakítás nélkül, BUSY flag figyeléssel kiválasztása.

#define LCDMODEONCEDELAY

Egyszeri frissítési üzemmód megszakítás nélkül, LCD írási műveletek között várakozási ciklussal kiválasztása.

#define LCDMODEONCEIRQ

Egyszeri frissítési üzemmód megszakítással kiválasztása.

#define LCDMODECONTBUSY

Folyamatos frissítési üzemmód megszakítás nélkül kiválasztása.

#define LCDMODECONTIRQ

Folyamatos frissítési üzemmód megszakítással kiválasztása.

#include "hardwareprofile.h"

Ha közös hardverkonfigurációs állományt használunk, akkor azt itt megadhatjuk. Ebben az esetben az ott megadott beállításokat ebben a file-ban töröljük komment jellel.

#define LCD4BITMODE

Ha definiálva van 4 bites módban lesz működtetve a kijelzőt. Ekkor az adatlábak közül csak D4..D7 lábak aktívak. 8 bites módot használva komment jellel töröljük.

#define LCDRWUSED

Ha nincs szükség BUSY FLAG olvasására, az **R/W láb** megspórolható. Ekkor az kijelző R/W lábát GND-re kell kötni, a definíció pedig komment jellel törölhető. Ilyenkor **LCDRWPORT** és **LCDRWPINNUM** értékei nem lesz felhasználva.

#define LCDEPORT 'A..G'

Itt adjuk meg, hogy az LCD E lábat melyik porthoz tartozó lábra kötjük.

#define LCDEPINNUM n

Itt adjuk meg, hogy az LCD E lábat a kiválasztott port melyik lábára kötjük.

#define LCDE2PORT 'A..G'

Itt adjuk meg, hogy az LCD E2 lábat melyik porthoz tartozó lábra kötjük (csak 2db, vagy dupla vezérlős LCD esetén).

#define LCDE2PINNUM n

Itt adjuk meg, hogy az LCD E2 lábat a kiválasztott port melyik lábára kötjük (csak 2db, vagy dupla vezérlős LCD esetén).

#define LCDRWPORT 'A..G'

Itt adjuk meg, hogy az LCD R/W lábat melyik porthoz tartozó lábra kötjük.

#define LCDRWPINNUM n

Itt adjuk meg, hogy az LCD R/W lábat a kiválasztott port melyik lábára kötjük.

#define LCDRSPORT 'A..G'

Itt adjuk meg, hogy az LCD RS lábat melyik porthoz tartozó lábra kötjük.

#define LCDRSPINNUM n

Itt adjuk meg, hogy az LCD RS lábat a kiválasztott port melyik lábára kötjük.

#define LCDDT0PORT 'A..G'

Itt adjuk meg, hogy az LCD D0 lábat melyik porthoz tartozó lábra kötjük.

#define LCDDT0PINNUM n

Itt adjuk meg, hogy az LCD D0 lábat a kiválasztott port melyik lábára kötjük.

#define LCDDT1PORT 'A..G'

Itt adjuk meg, hogy az LCD D1 lábat melyik porthoz tartozó lábra kötjük.

#define LCDDT1PINNUM n

Itt adjuk meg, hogy az LCD D1 lábat a kiválasztott port melyik lábára kötjük.

D2..D7 lábakat is a fentiek szerint kell megadni. 4 bites üzemmódban elég D4..D7 lábak megadása, ilyenkor a D0..D3 definícióit nem használja.

#define SystemClock x,

#define CpuClock y

Itt adhatjuk meg a rendszer és a CPU órajel frekvenciáját. A **CpuClock** alapján számítja ki az inicializáláshoz szükséges várakozási ciklusszámokat és ez határozza meg az E lábra adott impulzus megfelelő hosszához szükséges Nop utasítások számát is. Megszakításos üzemmódban a **SystemClock** alapján számítja ki a Timer beállításait. Ha ez a két órajel megegyezik, elegendő csak a **SystemClock**-ot megadni.

#define LCDLINES n

Az LCD kijelző sorainak száma (n értéke 1, 2 vagy 4 lehet)

#define LCDWIDTH n

Az LCD kijelző oszlopainak száma (n értéke maximum 40 lehet)

#define LCDSTEREO

2 db azonos sorszámú, 80 karakteresnél kisebb kijelzőt is használhatunk, ekkor a szöveg első fele az első, a második fele a második kijelzőn fog megjelenni. A kijelzők E lábait E, E2 lábak definíciói szerint kell bekötni, a többi lábat pedig párhuzamosan.

#define LCDTIMERNUM n

Megszakításos üzemmódban itt adhatjuk meg, melyik időzítőt használjuk. Az adott mikrovezérlőn létező és másra nem használt időzítőt válasszunk.

#define LCDTIMERPR18 -1

PIC18 csak megszakítás módban: melyik prioritású megszakítást használja az időzítőhöz? Lehetséges értékei:

- -1 : nincs a többszintű prioritás használva (ilyenkor minden megszakítás a HIGH interruptra kerül)
- 0 : alacsony prioritás
- 1 : magas prioritás

#define LCDUSERTIMER

Csak megszakítás módban: lehetőség van saját időzítésből hívogatni a frissítő rutint. Ilyenkor nem történik meg a TIMER inicializálása, azt nekünk kell megoldanunk. Akkor érdemes ezt használni, ha más célra is használunk időzítőt, ami a kijelző frissítési ütemezése (néhány milliszekundum) szerinti tempóban jár. Ebben az esetben az időzítőhöz tartozó megszakításkiszolgáló függvényből hívjuk meg az **LcdIntProcess()** függvényt.

#define LCDFRAMEPERSEC n

Csak megszakításos módban: a kijelző frissítési sebessége (FPS). A javasolt értéke 20. A megadott értékkel azt adjuk meg, hogy másodpercenként hányszor vigye át a kijelzőre az **LcdChar** tömb teljes tartalmát. Az időzítő osztóit az itt megadott értékből, a kijelző karaktereinek számából és a megadott rendszerórajelből fogja a fordítóprogram kikalkulálni.

#define LCDEXECUTIONTIME 50

Csak **LCDMODEONCEDELAY** módban felhasználva: a kijelző parancsvégrehajtási ideje (usec)
(A kijelző adatlapján EXECUTION TIME találjuk. 40usec a tipikus ideje, legalább ekkora időt válasszunk!)

#define LCDBLINKCHAR

Ha definiálva van, lehetőségünk van bármelyik és bármennyi karakter villogtatására (akár az összes is villoghat).
Villogás be: **LcdBlinkChar(5)** függvény kiadása után az 5. karakter villogni fog (változót is adhatunk paraméternek).
Villogás ki: **LcdUnBlinkChar(5)** függvény kiadása után az 5. karakter nem fog villogni (változót is adhatunk paraméternek). Arra ügyeljünk, hogy a tömböt ne írjuk túl!

#define LCDBLINKSPEED n

Csak **LCDMODECONTIRQ** üzemmódban, az automatikus villogás sebességének megadása. Ha pl. 5-öt adunk meg, az a következőt jelenti: 5 frame ideig látszik a szöveg, majd 5 frame ideig sötét (ez **LCDFRAMEPERSEC 20** esetén fél másodperces periódusidőt jelent). Ha nem szeretnél automatikus villogtatást, akkor n = 0 értéket adj meg. Ekkor **BlinkPhase** változóba 0 értéket írva látszanak a villogó karakterek, egyébként nem. Azokban az üzemmódokban, ahol nem lehetséges az automatikus villogás, ott is ezzel a módszerrel lehet villogást létrehozni. Egyszeri frissítési üzemmód esetén **LcdRefreshAll()** függvényt is meg kell hívni ahhoz, hogy az aktuális villogási fázis szerint megjelenítendő tartalom átkerüljön a kijelzőre.

#define LCDCURSOR

Kurzorhasználat engedélyezése (csak egyszeri frissítési üzemmódban)

kurzor pozíció beállítása: **LcdCursorPos = n** (0 < n < kijelző(k) karaktereinek száma - 1)

- kurzor bekapcsolása: **LcdCursorOn()**
- kurzor kikapcsolása: **LcdCursorOff()**
- villogó kurzor bekapcsolása: **LcdCursorBlink()**
- villogó kurzor kikapcsolása: **LcdCursorUnBlink()**

A kurzor pozíció és kurzor állapot beállítása után **LcdRefreshAll()** függvényt is meg kell hívni, hogy a beállított állapot átkerüljön a kijelzőre.

#define LCDZEROCHANGE

Ha definiálva van, az **LcdText**-ben levő #0 kódú karakterek helyett #32 (SPACE) karaktert ír ki a kijelzőre. Ez akkor hasznos, ha könyvtári számátalakító függvényeket használunk. Ezek ugyanis #0 lezáró karaktert tesznek a szám végére, így az LcdText tömbünk belsejébe #0 karakterek kerülhetnek, ami az LCD-nek az első felhasználói karaktert jelenti. Mivel ez a karakter a #8 karakterkóddal is megjeleníthető, célszerű szóközre cserélni.

#define USERCHARSET

Ha definiáljuk, a kijelző inicializálásakor a kijelzőbe feltölti az általunk megtervezett 8 saját karaktert is. A karaktereket a **#define USERxCHARy** definíciók értékeinek módosításával állíthatjuk elő. A 0 érékű bit világos, az 1 értékű bit sötét pontot fog eredményezni világos hátterű kijelző esetén. Az általam létrehozott példákban 0..7-ig inverz számok vannak, hogy könnyen be lehessen azonosítani.

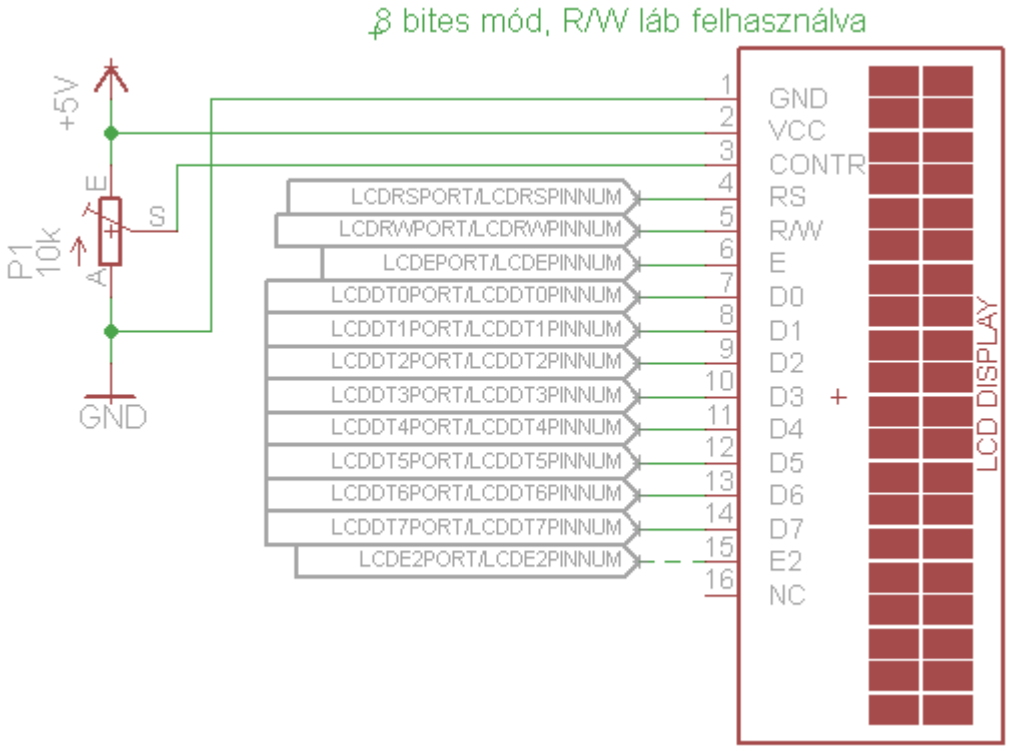
#define USERCHARSETCHANGE

Karakterkészlet futás alatti változtatásának lehetősége. **LcdChangeCharset(char* pch)** függvénnyel lehet feltölteni az aktuális karakterkészletet. a karakterkészletet **USERCHARSETARRAY** típusú változóval lehet létrehozni a RAM-ban.

A kijelzőt négy módon köthetjük be a beállított üzemmódtól függően. A rajzon a dupla kijelző E2 lábát szaggatott vonallal jelöltem, ez 80 karakteresnél kisebb kijelző esetén természetesen nem is létezik. Ilyenkor teljesen mindegy, hogy a konfigurációban mit definiálunk hozzá, nem használja azt a portlábat (az adatirányt sem állítja be hozzá, bármire szabadon felhasználnható). Ez természetesen más fel nem használt lábra is vonatkozik (R/W-re, ha GND-re kötjük, D0..D3 4 bites módban). Két kijelzőt **(LCDSTEREO)** működtetve az első kijelző E lábát kell **LCDEPORT/LCDEPINNUM** által meghatározott lábra, a második kijelző E lábát pedig **LCDE2PORT/LCDE2PINNUM** által meghatározott lábra kell kötni. Az összes többi lábat párhuzamosan mindkét kijelzőre be kell kötni.

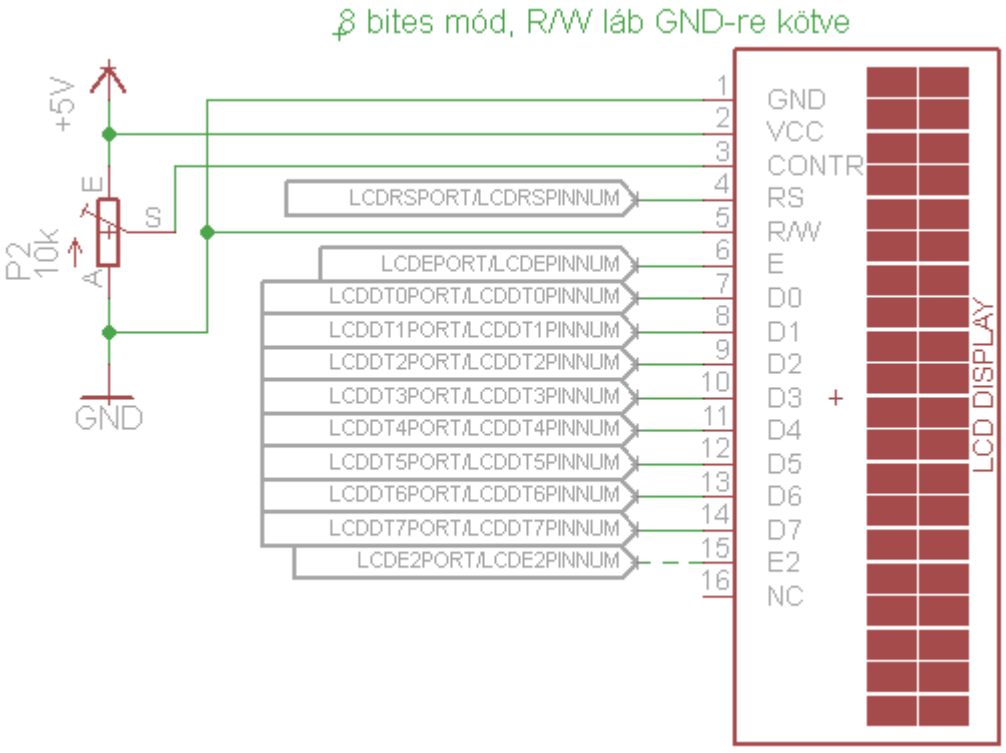
8 bites üzemmód, R/W lábat is felhasználva

```
1. // #define LCD4BITMODE
2. #define LCDRWUSED
```



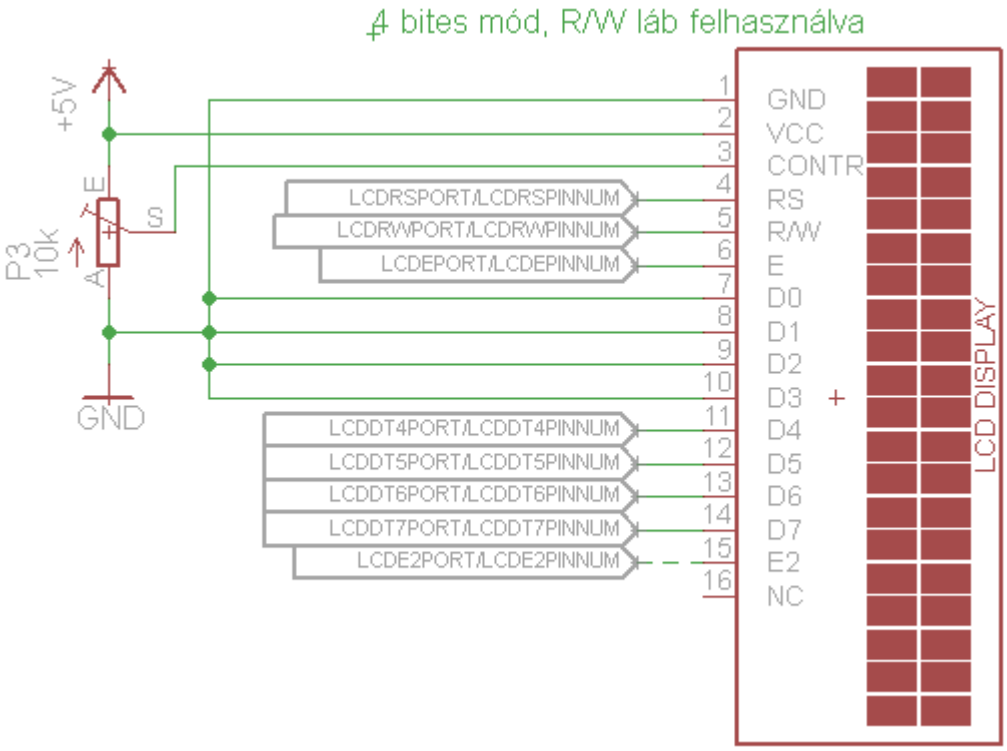
8 bites üzemmód, R/W láb GND-re kötve

```
1. // #define LCD4BITMODE
2. // #define LCDRWUSED
```



4 bites üzemmód, R/W lábat is felhasználva

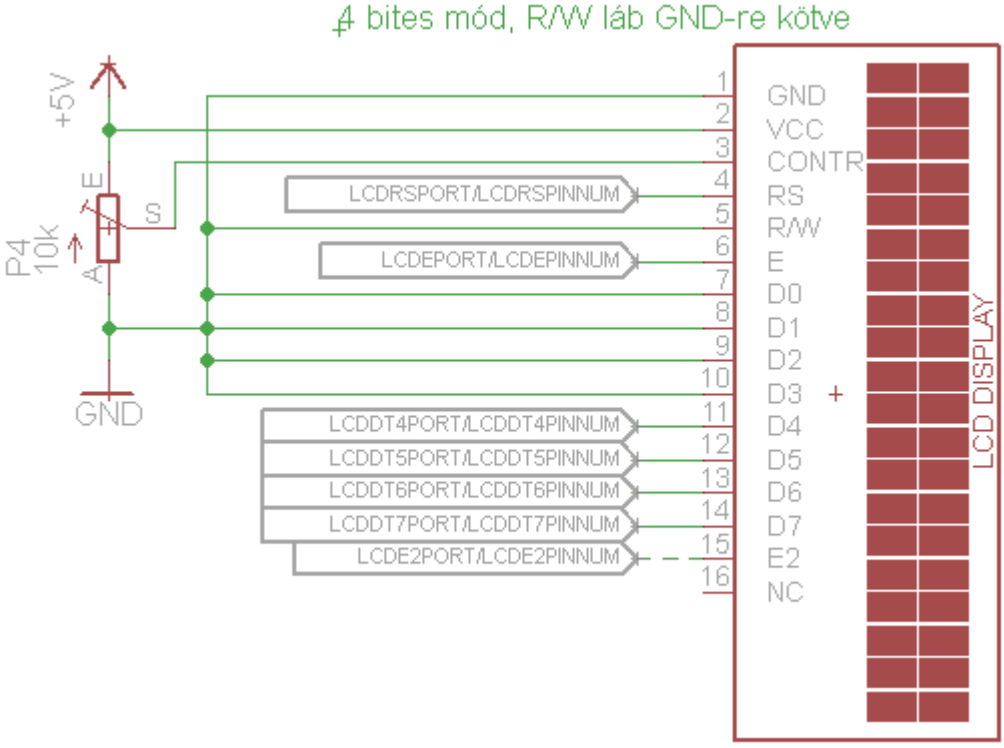
```
#define LCD4BITMODE
#define LCDRWUSED
```



---

4 bites üzemmód, R/W láb GND-re kötve

```
#define LCD4BITMODE
// #define LCDRWUSED
```



A ’Helloworld’ program

Kiindulásnak írjunk egy programot, ami a kijelzőn megjeleníti a "Hello world" szöveget (azért maradtam az angol verziónál, mert az ékezetes betűket sajnos nem tartalmazza a standard karakterkészlet). Miután a **charlcd.h**-ban elvégeztük a konfigurációs beállításokat, rátérhetünk a főprogramunkra (a cikkben a könyvtári és a mikrovezérlőcsaládtól függő inlude fájlok, valamint a main függvény különböző típusú visszatérési értéke miatti eltérések ismertetésétől eltekintek, a letölthető mintaprogramokban ezek megtekinthetőek).

Egyszeri frissítési üzemmódban **(LCDMODEONCE...)** az **LcdText** feltöltése után meg kell hívni az **LcdRefreshAll()** függvényhívást. (C18 fordító esetében a memcpy helyett a memcpypgm2ram függvényt használjuk!)

```
1. #include "charlcd.h"
2. int main(void)
3. {
4.   LcdInit();
5.   memcpy((char *)LcdText, "   Hello world                ", 32);
6.   LcdRefreshAll();
7.   while(1)
8.   {
9.   }
10.}
```

Megszakítás nélküli folyamatos frissítési üzemmódban **(LCDMODECONTBUSY)** a végtelen ciklusba be kell iktatni egy **LcdProcess()** függvényhívást.

```
1. #include "charlcd.h"
2. int main(void)
3. {
4.   LcdInit();
5.   memcpy((char *)LcdText, "   Hello world                ", 32);
6.   while(1)
7.   {
```

```
8.     LcdProcess();
9.   }
10. }
```

Megszakításos folyamatos frissítési üzemmódban **(LCDMODECONTIRQ)** üzemmódban csak az **LcdText** tömböt kell feltölteni, a többit elintézi a driver.

```
1. #include "charlcd.h"
2. int main(void)
3. {
4.   LcdInit();
5.   memcpy((char *)LcdText, "   Hello world                  ", 32);
6.   while(1)
7.   {
8.   }
9. }
```

Mivel a PIC16 és PIC18 vezérlők nem használnak a megszakításhoz ugrótáblázatot, manuálisan kell beilleszteni a közös megszakításkiszolgáló függvényünkbe. Ezt egyszerűen egy **LcdIntProcess();** függvényhívással tehetjük meg. PIC18 esetén az inicializáló függvény beállítástól függően engedélyezheti a kétszintű megszakítás használatát, a függvényhívást a beállított prioritási szint megszakításkiszolgáló függvényébe kell beilleszteni (ha nem használjuk a többszintű megszakítást, akkor a magas prioritásúba). Nem szükséges a megszakítás forrásának szelektálásával bajlódnunk, ezt az **LcdIntProcess()** megteszi helyettünk (**charlcd.h**-ban megnézhető hogyan). Ugyanazt a közös megszakításkiszolgáló függvényt természetesen a többi megszakítás kiszolgálására is felhasználhatjuk.

Ezt a következőképpen tehetjük meg PIC16 esetén:

```
1. static void interrupt isr(void)
2. {
3.   LcdIntProcess();
4. }
```

PIC18 esetén:

```
#pragma code high_vector_section = 0x8
void InterruptVectorHigh(void)
{
  _asm goto YourHighPriorityISRCode _endasm
}

#pragma code low_vector_section = 0x18
void InterruptVectorLow(void)
{
  _asm goto YourLowPriorityISRCode _endasm
}

#pragma code

#pragma interrupt YourHighPriorityISRCode
void YourHighPriorityISRCode(void)
{
   // ha LCDTIMERPR18 == -1 vagy 1 akkor ide illesszük be
   LcdIntProcess();
}

#pragma interruptlow YourLowPriorityISRCode
void YourLowPriorityISRCode(void)
{
  // ha LCDTIMERPR18 == 0 akkor ide illesszük be
  LcdIntProcess();
}
```

A mellékletben megtalálható példaprogramok:

Helloworld:

Hello world program a különböző üzemmódokhoz.

Cpu-usedmeter:

Megméri, hogy megszakításos folyamatos frissítési üzemmódban (LCDMODECONTIRQ)
mennyi processzoridőt vesz igénybe az LCD frissítése. A mérés úgy történik, hogy megméri egy adott idő alatt (1. Timerrel megvalósítva), egy 32 bites számlálót meddig tud egyesével megnövelni. Mindezt kikapcsolt és bekapcsolt Lcd frissítéssel is. Kiírja a két számláló értékét, valamint azt is, hogy a bekapcsolt frissítés alatt hány százalékkal lett alacsonyabb ez az érték.

Demo-cursor

A kijelzőn a karakterkészletet visszafelé gördíti, miközben a kurzort előre viszi. A kurzort e közben ki-be kapcsolgatja, és a formáját is váltogatja.

Lcd-speedmeter

TIMER1 segítségével megméri mennyi ideig tart az LCD frissítése BUSY flag figyeléses üzemmódban (LCDMODEONCEBUSY). Kiírja az ebből adódó képkocka/másodperc (FPS) és a karakter/másodperc (CPS) értékét.

Demo-ora

Óra program mind a 4 mikrovezérlő családhoz. A program bemutatja a villogtatási lehetőséget, a kurzort, a felhasználó által definiált karaktereket, és azok animálását is. Hogy mit mutasson be, az pusztán az **lcdchar.h**-ban beállított konfigurációtól fog függni. Sikeres konfigurálás és felprogramozás után a kijelzőn meg kell jelennie a működő órának.

A karakter LCD kezelő "driver" v0.5 letölthető innen: charlcd-v05.zip

Sok sikert a felhasználáshoz.

Roberto Benjami