Základy ovládania LCD displejov kompatibilných s radičom HD44780

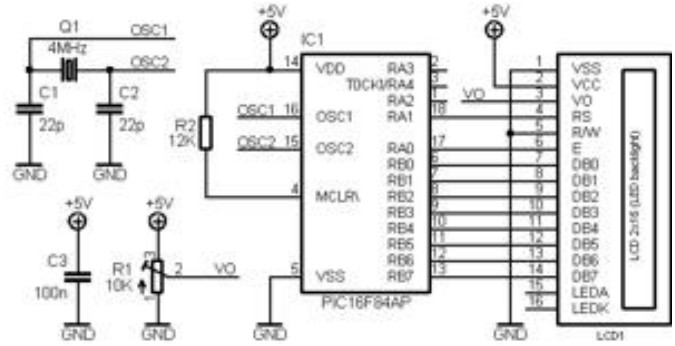
Máme tu druhú časť o ovládaní znakových displejov, v ktorej si ukážeme, ako tieto displeje ovládať v 8 - bitovom móde pomocou mikrokontrolérov.

Čo budeme k tejto ukážke potrebovať?

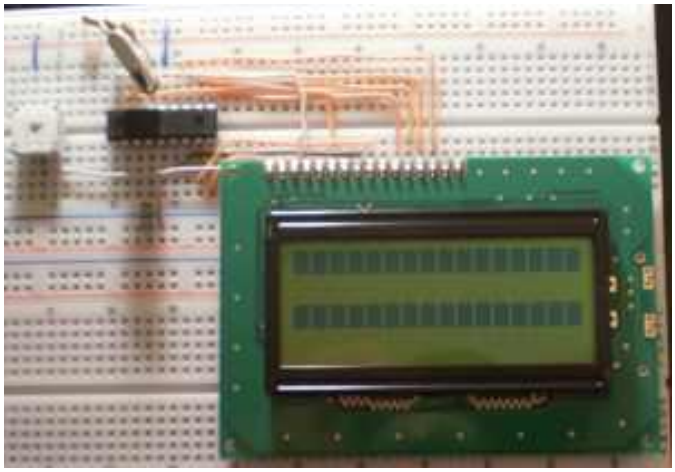
V prvom rade, keď už nič iné aspoň zhruba nahliadnuť do prvej časti tohto miniseriálu. Potom jedno menšie kontaktné pole, napájací zdroj, nejakú tú bižutériu (rezistory, kondenzátory, prepojovacie drôtiky, kryštál), samotné displeje (v ukážkach sú použité konkrétne typy BC1602HYPNEH a MC1604B-SYR) a samozrejme jeden kus mikrokontroléru (pre jednoduchosť som zvolil starú dámu PIC16F84A).

Príprava:

Recept je prostý, všetky súčiastky poukladáme na kontaktné pole a zapojíme podľa schémy na nasledujúcom obrázku. Schéma je nakreslená pre tzv. 8 - bitový mód displejov, kde sa na prenos dát využíva ako už samotný názov trochu naznačuje všetkých 8 dátových vodičov DB0 až DB7. Vidíme, že dátovými vodičmi sme zabrali celý jeden port mikrokontroléru, v našom prípade PORTB. Ak budete mať iný typ mikrokontroléru s viacerými portami môžete si displej kľudne pripojiť k portu A, B, C atď. Samozrejmé je, že ak na sa na danom porte budú nachádzať napríklad piny zdieľané s UART perifériou (alebo inými periférami) a my ju (ich) budeme chcieť použiť, na tento port displej zapojiť nemôžme (resp. môžme, ale bude si to potom vyžadovať prepínanie medzi perifériou a displejom podľa toho ktoré budeme aktuálne používať).



V schéme ďalej vidíme, že posledné dva vývody displeja sme nezapojili. Vývody patria podsvieteniu, ktoré používať nebudeme. Vy si ho však pripojiť môžete, vývod 16 stačí uzemniť a vývod 15 pripojiť ku kladnému pólu zdroja cez odpor, ktorého veľkosť si vyrátate z údajov pre podsvietenie uvedených v datasheete. Kontrast displeja sa nastavuje trimrom R1, tak ako sme to písali v predošlej časti, kde rovnako nájdete dôvod pripojenia pinu R/W na zem. Na ďalšom obrázku vidíme všetko zapojené na kontaktnom poli a pripravené pre nahranie programu do mikrokontroléru. Ešte malá poznámka: Pozor na zapojenie vývodov displejov! Nie vždy bývajú rovnaké, myslím tým skutočnosť, že priradenie pinov dátovým a riadiacim signálom sa zhoduje, ale u niektorých je pin 1 úplne vľavo u niektorých úplne vpravo a pod.



Program pre mikrokontrolér:

Keďže sme hardvérovú prípravu zvládli (na displeji by sme mali vidieť zobrazenie aké sa nachádza na obrázku vyššie), pristúpime k samotnému programu. Pred začiatkom písania tohto textu som si položil otázku: "Ako vlastne začať?" Nuž myslím, že najlepšie bude od začiatku. V minulej časti sme si opísali sekvenciu, ktorú musíme vykonať aby sme do radiča displeja zapísali príkaz alebo dáta. Následne sme ju manuálne pomocou DIP prepínačov a tlačítka vykonali. Teraz tu však máme mikrokontrolér a na túto sekvenciu si napíšeme funkciu, ktorá ju bude vykonávať.

```
1.
   void Lcd_write(unsigned char byte)
2.
   {
3.
   DelayUs(43);
4.
   LCD_DATA = byte;
5.
   Lcd_strobe();
6.
   }
```

Funkcia **Lcd_write()** zapíše jeden byte do radiča displeja. V tele funkcie vidíme najprv ďalšiu funkciu, ktorá vytvára 43us oneskorenie (o tom prečo sa tam toto oneskorenie nachádza a prečo práve 43us si povieme neskôr). Za oneskorením nasleduje nastavenie logických úrovní na dátových vodičoch DB0 - DB7, príkazom LCD_DATA = byte; sa obsah premennej byte objaví na porte B mikrokontoléru a tým aj na dátových vodičoch DB0 až DB7. K samotnému zapísaniu bajtu do radiča displeja však musí nastať ešte jedna udalosť, zmena z log.1 na log.0 na vodiči EN (príkaz alebo dáta sú do radiča zapísané pri zostupnej (dobežnej) hrane signálu na vodiči EN). Toto nám zabezpečuje funkcia **Lcd_strobe()** (dá sa samozrejme napísať ako makro, ale pre tento príklad ju budeme mať ako funkciu).

```
1.
   void Lcd_strobe(void)
2.
   {
3.
   LCD_EN=1;
4.
   //DelayUs(10);
5.
   LCD_EN=0;
6.
   }
```

Pri tejto funkcii sa chvíľku pozastavíme a odpovieme si otázky, ktoré možno niektorým z vás vŕtajú hlavou (priznám sa, že ak by som bol čitateľ problematiky neznalý aj mne by napadlo pár otázok). Keď sa pozrieme na funkciu Lcd_strobe() v jej strede uvidíme funkciu oneskorenia DelayUs(10), ktorá je kvôli dvom lomítkam pred sebou prekladačom braná ako poznámka (prekladač ju neberie do úvahy pri preklade programu). Mohlo by sa zdať, že len to len zabudnutý kúsok kódu, ale nie je tomu tak. Keď sa pozrieme do datasheetu rodiny displejov BC1602A (uvedený v prílohe), v časti 12-2.4.1 write operation sa nachádza časový diagram riadiacich a dátových signálov displeja spolu s tabuľkou, v ktorej sú definované jednotlivé minimálne alebo maximálne časy trvania rôznych parametrov jednotlivých signálov. Z tejto tabuľky nás zaujíma parameter **Enable pulse width (high level)** (čiže dĺžka trvania log.1 na vodiči EN), ktorého minimálna hodnota je 230ns (táto hodnota nie je u všetkých displejov rovnaká, napr. pre druhý nami použitý displej MC1604B-SYR je táto hodnota minimálne 450ns). Prečo nás táto hodnota zaujíma? Z jednoduchého dôvodu, frekvencia použitého kryštálu je 4MHz, čo pre daný mikrokontrolér znamená dĺžku inštrukčného cyklu 1us a tým pádom aj dobu trvania log.1 na vodiči EN. Myslím, že je to dosť jasne vidieť zo samotnej funkcie. Najprv nastavíme na vodič EN úroveň log.1 a následne úroveň log.0. Vykonanie inštrukcie ale zaberá istý čas, a tým pádom úroveň signálu na vodiči EN neklesne na log.0 okamžite, ale práve za čas vykonania jednej inštrukcie, čiže za 1us. Vidíme, že minimálna hodnota parametra Enable pulse width pre korektný zápis údajov do radiča displeja je dodržaná. Čo sa stane ak bude mať frekvencia kryštálu hodnotu napr. 20MHz? Inštrukčný cyklus sa skráti na 200ns a minimálna hodnota trvania log.1 na vodiči EN nie je dodržaná ani pre jeden z našich displejov. Čo z toho plynie si iste každý z vás domyslí, áno znamená to, že do radiča displeja sa nezapíše nič (a my sa budeme čudovať prečo nám náš displej nič nezobrazuje). Presne z tohto dôvodu sa vo funkcii Lcd_strobe() nachádza funkcia oneskorenia DelayUs(10), ktorá zabezpečí korektné trvanie úrovne log.1 na vodiči EN (hodnota 10 je použitá len pre ilustráciu).

Keďže sme sa už začali vŕtať v oneskoreniach, myslím že je na čase objasniť 43us oneskorenie spomínané vo funkcii Lcd_write(). Tak isto ako človeku trvá urobiť určitý úkon (zdvihnúť ruku, napísať písmenko na papier), aj radičom v našich displejoch trvá istý čas, kým si dáta zapíšu do internej RAM, alebo vykreslia písmenko, atď. Tento čas je neporovnateľne menší ako u človeka, avšak nie je nulový. Keďže mikrokontrolér, ktorým displej ovládame je vo väčšine prípadov omnoho rýchlejší, nemôžme radiču podávať bezprostredne jednu informáciu za druhou ako sa nám zachce. Aj tu musia platiť určité pravidlá, ktoré môžete nájsť v datasheete (v prílohe), konkrétne v časti 12-2.3 Instruction table. Uvedieme si pár príkladov, zapísanie údaja do internej dátovej pamäte RAM trvá radiču 43us. Ak by sme posielali radiču údaje na zápis do jeho internej RAM kontinuálne, toto 43us oneskorenie nám zabezpečí, že každý poslaný bajt bude korektne prijatý a zapísaný do RAM radiča. Vymazanie displeja napríklad trvá 1,53ms, posun na určitú adresu v RAM pamäti 39us atď. (viď. spomínaná tabuľka).

Aby sme neskákali z miesta na miesto ostaneme ešte nejakú chvíľku pri vyššie spomínanej tabuľke inštrukcií. Okrem doby trvania jednotlivých inštrukcií si môžeme všimnúť, že každá z nich má svoj kód, skladajúci sa z bitov DB0 - DB7 (naše dátové vodiče), bitu R/W (ak by bol pin R/W na displeji zapojený k mikrokontroléru hodnoty zo stĺpca R/W tabuľky by sa posielali práve naň) a bitu RS, ktorý zodpovedá pinu RS na displeji a riadime ním, či si má radič informáciu na dátových vodičoch interpretovať ako príkaz alebo dáta. Keďže teraz poznáme kódy jednotlivých inštrukcií, môžeme si napísať nasledujúce funkcie na ich vykonanie. Funkciu **Lcd_clear()**, ktorá vymaže obsah dátovej pamäte RAM (a tým celé zobrazenie na displeji) a počítadlo adries RAM pamäte nastaví na jeho východziu pozíciu 0x00.

```
1.
   void Lcd_clear(void)
2.
   {
3.
    LCD_RS = 0;
4.
    Lcd_write(0x01);
5.
    DelayMs(2);
```

```
6.
}
```

Funkciu **Lcd_ret_home()**, ktorá zachová obsah dátovej pamäte RAM a iba nastaví počítadlo adries RAM na jeho východziu pozíciu.

```
1.
void Lcd_ret_home(void)
2.
{
3.
 LCD_RS = 0;
4.
 Lcd_write(0x02);
5.
 DelayMs(2);
6.
}
```

Funckiu **Lcd_goto_pos(row, position)**, ktorá umožňuje nastaviť riadok a pozíciu v dátovej RAM kam chceme zapisovať (údaje zapísané do tejto pamäte sa zobrazia na displeji).

```
1.
void Lcd_goto_pos(unsigned char row, unsigned char position)
2.
{
3.
 unsigned char pos;
4.
5.
 LCD_RS = 0;
6.
 if(row == 1)
7.
        pos = 0x00 | position;
```

```
8.
    if(row == 2)
9.
        pos = 0x40 | position;
10.
    if(row == 3)
11.
        pos = 0x10 | position;
12.
    if(row == 4)
13.
        pos = 0x50 | position;
14.
    Lcd_write(0x80|pos);
15.
    }
```

Čo sa týka riadkov a pozícií, oba použité displeje majú maximálny počet zobraziteľných znakov v riadku 16, z čoho vyplýva, že rozsah čísel pre premennú position je 0 až 15. Adresa prvého riadku dvojriadkového displeja je 0x00, adresa druhého riadku je 0x40. Pri štvorriadkovom displeji navyše k prvej a druhej adrese pribudnú adresy začiatkov tretieho a štvrtého riadku, ktoré sú 0x10 a 0x50.

V predošlej časti je napísané, že k zobrazeniu textu na displej, je potrebné urobiť nasledujúce kroky:

- 1. inicializovať displej

- 2. nastaviť displej do požadovaného módu

- 3. poslať na displej postupnosť znakov, ktoré chceme zobraziť

Po chvíľkovej úvahe sa javí ako dosť rozumné spojiť body 1 a 2 do jedného a vytvoriť funkciu **Lcd_init()**, ktorá inicializuje displej a zároveň ho nastaví do módu podľa našich požiadaviek. Na tomto mieste budeme musieť skombinovať informácie z tabuľky 12-2.3 Instruction table, ktorú už poznáme s informáciami z vývojového diagramu 12-5.5.1 8-bit interface, ktorý ukazuje krok za krokom ako inicializovať displej a nastaviť užívateľom požadovaný mód. Nuž pustíme sa do toho. V prvom rade nastavíme náš dátový port LCD_DATA (PORTB) celý ako výstupný, ďalej nastavíme ako výstupy aj piny RS (RA1) a EN (RA0), ktorým priradíme úroveň log.0. Prečo priraďujeme úroveň log.0 na vodič EN je myslím každému jasné (stačí si spätne pozrieť funkciu Lcd_strobe). Odpoveď na to prečo priraďujeme log.0 na vodič RS je nasledujúca, pri inicializácii displeja sa posielajú radiču displeja len príkazy a preto je vhodné mu priradiť túto log. úroveň hneď pri inicializácii príslušných pinov mikrokontroléra.

Od tohto momentu sa pri písaní inicializačnej funkcie pre displej budeme riadiť podľa spomínaného vývojového diagramu. Po pripojení napájania k displeju musíme počkať minimálne 15ms. Do do inicializačnej rutiny teda zaradíme 15ms oneskorenie pomocou funkcie DelayMs(15). 15ms sme počkali a teraz vo vývojom diagrame vidíme, že radiču displeja musíme poslať po

dátových vodičoch 3 krát tú istú kombináciu log. úrovní (0011xxxx, kde za x môžeme dosadiť log.0 alebo log.1) s určitými oneskoreniami. Poslané máme a teraz prichádza na rad špecifikovanie počtu riadkov displeja a fontu. Pozrieme sa naspäť do tabuľky kde sa nachádzajú kódy jednotlivých inštrukcií, konkrétne na riadok s názvom Function set. Môžeme vidieť nasledujúci kód 0 0 1 DL N F - -. Keďže sme v 8 - bitovom móde DL = 1, máme dvojriadkový displej N = 1 (môžeme si ho ale nastaviť aj ako jednoriadkový a potom N = 0), font nastavíme na 5*8 bodov, čiže F = 0, za pomlčky si môžeme doplniť čo chceme a teda na posledné dve miesta v bajte dáme nuly. Výsledná kombinácia log. úrovní poslaná na dátové vodiče bude vyzerať 00111000 = 0x38. Ďalší riadok tabuľky, ktorý nás zaujíma je Display ON/OFF control, vidíme v ňom nasledujúci kód 0 0 0 0 1 D C B. Displej chceme zapnúť takže D = 1, kurzor aj blikanie vypnúť, čo značí C = B = 0 (vy si kľudne kurzor aj blikanie môžete zapnúť nastavením bitov C a B na log.1). Kombinácia (byte) poslaná na dátové vodiče bude 00001100 = 0x0C. Nasledujúci krok je vymazanie displeja funkciou Lcd_clear(). Pri poslednom kroku uprieme svoj zrak do tabuľky na riadok s názvom Entry Mode Set, v ktorom vidíme kód 0 0 0 0 0 1 I/D SH. Bitom SH sa nastavuje posun displeja, ktorý nechceme a preto bit SH bude nulový. Bitom I/D nastavuje smer posunu kurzora (a zároveň a čo je hlavnejšie smer pohybu zápisu do dátovej RAM). My sa chceme posúvať smerom doprava a tak nastavíme inkrementáciu, čiže I/D = 1. Výsledná kombinácia poslaná na dátové vodiče bude 00000110 = 0x06. Týmto považujeme funkciu pre inicializáciu displeja za hotovú.

```
1.
   void Lcd_init(void)
2.
   {
3.
    LCD_DATA_DIR = output;
4.
    LCD_RS = LCD_EN = 0;
5.
    LCD_RS_DIR = LCD_EN_DIR = output;
6.
7.
    DelayMs(15);
8.
    Lcd_write(init_value);
9.
    DelayMs(5);
10.
    Lcd_write(init_value);
11.
    DelayUs(200);
12.
    Lcd_write(init_value);
13.
    Lcd_write(function_set);
14.
    Lcd_write(display_control);
15.
```

```
    Lcd_clear();
16.
    Lcd_write(entry_mode_set);
17.
  }
```

Teraz nám ešte zostáva napísať si funkcie k posielaniu samotných znakov na displej. Prvou z nich je funkcia **Lcd_putch(ch)**, ktorá zobrazí na displeji jeden znak.

```
1.
  void Lcd_putch(unsigned char ch)
2.
  {
3.
    LCD_RS = 1;
4.
    Lcd_write(ch);
5.
  }
```

Druhou je funkcia **Lcd_puts(*s)**, ktorá zobrazí na displeji nami zadaný reťazec nachádzajúci sa v programovej pamäti mikroradiča (viď. hlavná funkcia main ku koncu článku).

```
1.
  void Lcd_puts(const char *s)
2.
  {
3.
    LCD_RS = 1;
4.
    while(*s)
5.
    Lcd_write(*s++);
6.
```

) sú definované v hlavičkovom súbore LCD.h (taktiež v prílohe), rovnako ako aj symbolické názvy pre dátový port B a piny RA0, RA1.

```
}
```

```
#ifndef _LCD_h_
#define _LCD_h_
```

Možno sa pozriete na funkciu Lcd_init() a na popis ku nej a zdá sa vám (samozrejme že nie všetkým) prinajmenšom čudné, že v popise som písal hotové čísla, ktoré sa majú poslať na dátový port a v programe je namiesto čísel napísané init_value, function_set atď. Je to ale iba prvý klamlivý pohľad, pretože všetky tieto symbolické konštanty (ktorých názvy by mali byť podľa štábnej kultúry jazyka C písané veľkými písmenami no ja som ich napísal písmenami malými, snáď nebude nikto pohoršený a ak aj bude tak čo už nemôže byť v živote všetko podľa zásad

```
// define LCD interface
#define LCD_DATA PORTB              // DB0-DB7
#define LCD_DATA_DIR TRISB


#define LCD_RS RA1                  // RS pin
#define LCD_RS_DIR TRISA1


//#define LCD_RW xx                 //if R/W not connect to GND
//#define LCD_RW_DIR xx


#define LCD_EN RA0                  // EN pin
#define LCD_EN_DIR TRISA0


//#define LCD_BL xx                 //backlight on/off
//#define LCD_BL_DIR xx


// define display settings
```

```
21.
    #define function_set 0x38       /* 0 0 1 DL N F 0 0 */
22.
                        // Function set
23.
                        // bit2 set 0 - dot             format 5x8
24.
                        //                  1 - dot format 5x11
25.
                        // bit3 set 0 - 1 line              mode
26.
                        //                  1 - 2 line mode
27.
                        // bit4 set 0 - 4 bit               mode
28.
                        //                  1 - 8 bit mode
29.
30.
    //#define cur_disp_shift 0x??   /* 0 0 0 1 S/C R/L 0 0 */
31.
                        // Cursor or display                shift
32.
                        // bit2 set 0 - shift               left
33.
                        //                  1 - shift right
34.
                        // bit3 set 0 - cursor              shift
35.
                        //                  1 - display shift
36.
37.
    #define display_control 0x0C    /* 0 0 0 0 1 D C B */
38.
                        // Display              control
39.
                        // bit0 set 0 - blink               OFF
40.
                        //                  1 - blink ON
```

```
41.                     // bit1 set 0 - cursor             OFF
42.                     //                 1 - cursor ON
43.                     // bit2 set 0 - display            OFF
44.                     //                 1 - display ON
45.
46. #define entry_mode_set 0x06     /* 0 0 0 0 0 1 I/D SH */
47.                     // Entry mode             set
48.                     // bit0 set 0 - entire            shift OFF
49.                     //                 1 - entire shift ON
50.                     // bit1 set 0 -            decrement mode
51.                     //                 1 - increment mode
52.
53. // define other constants
54. #define output 0
55. #define init_value 0x30
56.
57. // function prototypes
58. void Lcd_strobe(void);
59. void Lcd_write(unsigned char           byte);
60. void Lcd_putch(unsigned char            ch);
61.
```

```
    void Lcd_puts(const char *s);
62.
    void Lcd_clear(void);
63.
    void Lcd_ret_home(void);
64.
    void Lcd_goto_pos(unsigned char             row, unsigned char pos);
65.
    void Lcd_init(void);
66.


67.
    #endif
```

Tu môžeme vidieť, že ak použijeme iný mikrokontrolér alebo budeme potrebovať zmeniť dátový port, či niektorý z riadiacich pinov, dosiahneme to prostým prepísaním napr. RA0 na RA2 a podobne. U mikrokontrolérov PIC sa nastavuje pin ako výstupný poslaním log.0 na príslušné miesto v tzv. direction registri s názvom TRISx (za x sa dosadzuje A, B, C .... podľa používaného portu). Napríklad ale u mikrokontrolérov MSP430 je to presne naopak, jednotlivé piny portu nastavíme ako výstupy tak, že na príslušné miesto v direction registri, ktorý sa vzťahuje na daný port zapíšeme log.1. Preto je tu zadefinovaná symbolická konštanta output (momentálne má hodnotu 0), ktorá sa v samotnom programe použije namiesto nuly alebo jednotky. Ak budeme chcieť program použiť napríklad už na spomínaných mikrokontroléroch MSP430, nemusíme zasahovať už do napísaného programu, ale v hlavičkovom súbore nám stačí zmeniť nulu za symbolickou konštantou output na jednotku a je to.

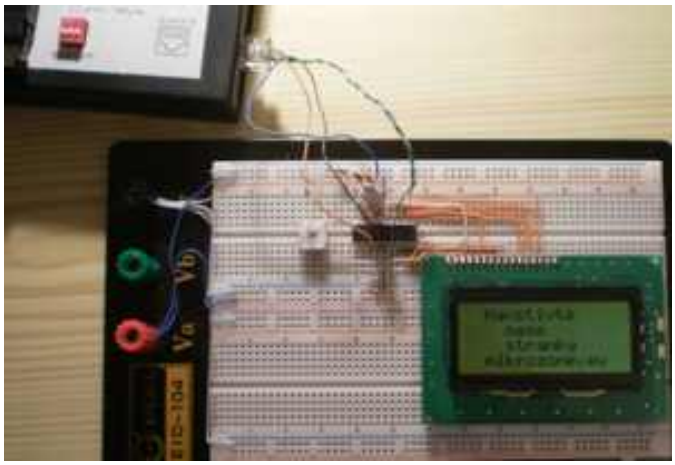
Keď sa tak pozerám vyššie, článok začína byť dlhý ako týždeň pred výplatou a preto by sa patrilo pomaličky to celé zakončiť, čo si koniec koncov aj môžem dovoliť, keďže všetko potrebné na ovládanie mojich dvoch displejov som napísal. Aby ste mi ale naozaj verili, že to celé funguje, napíšeme si ešte jednoduchý hlavný program v ktorom si na dvojriadkovom displeji zobrazíme text "Hello mikrozone" a na štvorriadkovom text "Navstivte nase stranky mikrozone.eu". A tu je sľúbený hlavný program:

```
1.
    #include // ...
2.
    #include "delay.h"
3.
    #include "lcd.h"
4.


5.
    __CONFIG(UNPROTECT&PWRTEN&WDTDIS&XT);
6.


7.
```

```
const unsigned char *text1          = "Hello";
const unsigned char *text2          = "mikrozone";
const unsigned char *text3          = "Navstivte";
const unsigned char *text4          = "nase";
const unsigned char *text5          = "stranky";
const unsigned char *text6          = "mikrozone.eu";


void main(void)
{
 Lcd_init();


// 2 line display
 Lcd_goto(1, 5);
 Lcd_puts(text1);
 Lcd_goto(2, 3);
 Lcd_puts(text2);


// 4 line display
/*
 Lcd_goto(1, 3);
 Lcd_puts(text3);
```

```
28.
    Lcd_goto(2, 5);
29.
    Lcd_puts(text4);
30.
    Lcd_goto(3, 4);
31.
    Lcd_puts(text5);
32.
    Lcd_goto(4, 2);
33.
    Lcd_puts(text6);
34.
   */
35.


36.
    while(1);
37.
  }
```

Firmvér mikrokontroléru bol skompilovaný s pomocou HI-TECH C Compiler for PIC10/12/16 MCU's ver. 9.70 pod MPLAB IDE ver. 8.50. Kompletné zdrojové súbory ako aj .hex súbory vhodné priamo na nahratie do mikrokontroléru môžete nájsť v prílohách. Aby to celé nebolo len o suchých opisoch funkcií pripájam dva obrázky, na ktorých sa môžeme konečne pokochať aj nejakými tými zobrazenými písmenkami a slovami.



Na záver mi nezostáva nič iné len vám zaželať pekné zobrazovanie a v budúcej (poslednej) časti si ukážeme ako ovládať tieto znakové displeje pomocou 4 - bitového módu.

Použitá literatúra a zdroje:

[1] Datasheet displeja BC1602HYPNEH
[2] Datasheet rodiny displejov BC1602A
[3] Datasheet displeja MC1604B-SYR časť.1
[4] Datasheet displeja MC1604B-SYR časť.2
[5] Datasheet mikrokontroléra PIC16F84A
[6] MPLAB IDE
[7] HI-TECH C for the PIC10/12/16 MCU Family

<- 1. časť

3. časť ->