

LABORATOR 6 - Interfață pentru afișor cu cristale lichide (LCD), partea II-a

Scopul lucrării

În laboratorul anterior s-au implementat funcțiile `rd_LCDreg(char vRS)`, `wr_LCDreg(char vRS, char data)`, `void ckbfb()` și `void initLCD()`, `sysinit()`.

În continuare se vor implementa și testa următoarele funcții:

<code>void putchLCD(char ch)</code>	scrie caracter
<code>void putsLCD(char *ch)</code>	scrie șir de caractere
<code>void clrLCD()</code>	șterge LCD și mută cursorul în prima poziție
<code>void gotoLC(unsigned char line, unsigned char col)</code>	mută cursorul

În secțiunea opțională se vor implementa și alte funcții. Aceste funcții se vor scrie în fișierul `IOfn.c`.

Pasul 1: `putchLCD(char ch)`

Scrieți funcția `putchLCD` în `IOfn.c`. Pentru ca un caracter să apară pe LCD la poziția cursorului scrieți respectivul caracter în registrul **DRegWr**. Nu uitați ca înainte de scriere BF să fie '0'.

Apoi modificați `LCDtest.c`. Pentru a scrie un caracter `LCDtest.c` va avea următoarea structură:

```
int main() {  
  
    sysinit();  
    putchLCD('H');  
  
    while(1) {  
    }  
}
```

Dacă adăugați mai multe apeluri `putchLCD` puteți scrie „Hello Micro”. Nu e obligatoriu.

Pasul 2: `putsLCD(...)`

Folosind `putchLCD` scrieți funcția `putsLCD` – scrie șir de caractere. Primul caracter al mesajului se va scrie la poziția cursorului. Cu un singur apel `putsLCD`, scrieți mesajul „Hello Micro”!

Informație utilă: în C un șir de caractere se termină `0x00 = '\0'`.

Pasul 3: `clrLCD()`

Scrieți funcția `clrLCD`. Folosiți instrucțiunea „Clear Display”. Pentru a testa funcționarea acestei comenzi, scriem un mesaj, așteptăm 1-2 secunde cu funcția `wait(...)` și apoi ștergem display-ul cu `clrLCD`. Structura programului de test este:

```
#define S2 250000UL  
...  
  
int main() {
```

```

    sysinit();

    while(1) {
        putsLCD("Hello Micro!");
        wait(S2);
        clrLCD();
        wait(S2);
    }
}

```

Pasul 4: gotoLC

Funcția *putsLCD* pe care ați scris-o anterior scrie mesajul începând de la poziția curentă a cursorului. Pentru a putea scrie mesaje pe orice linie, începând cu orice colană, este nevoie de o funcție care să mute cursorul. Această funcție se va numi *gotoLC* și va avea ca parametrii pe *line* și pe *col*:

```
void gotoLC(unsigned char line, unsigned char col)
```

unde *line* este linia pe care se va poziționa cursorul iar *col* este coloana. **Implementarea funcției se va baza pe comanda „Set DDRAM Address”.**

Pentru a afla adresa DDRAM corespunzătoare unei locații LCD **analizați** linia „Ce se vede” și linia „Conținut DDRAM” din figura 1 de la pagina următoare.

Parametrul *line* poate lua valorile 1 sau 2. Parametrul *col* ia valori în intervalul [1, 16]. Dacă vreunul din parametrii *line* și *col* sunt invalizi, comanda va fi ignorată. De exemplu *gotoLC(3,1)* este invalidă și nu va produce efecte.

Testați funcția cu secvența:

```

int main() {
    unsigned char k;

    sysinit();

    while(1) {
        putsLCD("0123456789abcdef");
        gotoLC(2,1);
        putsLCD("ghijklmnopqrstuv");

        for(k=16;k>0;k--) {
            gotoLC(2,k);
            wait(S2);
        }

        for(k=16;k>0;k--) {
            gotoLC(1,k);
            wait(S2);
        }

        clrLCD();
        wait(S2);
    }
}

```

Când funcționează, chemați profesorul pentru validare. Dacă ați ajuns aici aveți nota 5.

Secțiunea opțională

Pasul 5: Wrap

Funcția *putchLCD*, scrisă anterior, nu verifică dacă adresa din DDRAM la care se scrie caracterul este afișabilă sau nu. Dacă din prima coloană a primei linii scriem un mesaj mai lung de 16 caractere, vor apare numai primele 16 caractere. Încercați!

Reamintim că interfața LCD are memoria DDRAM alcătuită din 80 de caractere și poate comanda orice afișaj cu maxim două linii, fiecare linie cu 40 de coloane. Cum afișorul de care dispunem are numai 16 coloane, tot ce se va scrie dincolo de coloana 16 va fi memorat în DDRAM, dar nu se va afișa. De exemplu, dacă scriem pe linia 1, din coloana 16, mesajul „56”. Va apare numai caracterul „5” iar cursorul va dispărea:

	Coloana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16				
Linia 1	Adresa DDRAM	00h	01h	02h	03h	04h	05h	06h	07h	08h	09h	0Ah	0Bh	0Ch	0Dh	0Eh	0Fh	10h	11h	...	27h
	Conținut DDRAM	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	35h	36h	20h		20h
	Ce se vede																5	6	-		
Linia 2	Ce se vede																				
	Conținut DDRAM	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h	20h		20h
	Adresa DDRAM	40h	41h	42h	43h	44h	45h	46h	47h	48h	49h	4Ah	4Bh	4Ch	4Dh	4Eh	4Fh	50h	51h	...	67h

figura 1

1) **Creați un nou proiect, identic cu proiectul LCD. Fie numele acestui proiect LCD2.**

2) În noul proiect modificați funcția *putchLCD* astfel încât:

- După ce s-a scris caracterul din poziția 16 a liniei 1 cursorul să apară pe linia 2 coloana 1.
- După ce s-a scris caracterul din poziția 16 a liniei 2 cursorul să apară pe linia 1 coloana 1

Atenție:

- Nu aveți voie să folosiți variabile auxiliare pentru a memora poziția cursorului. Poziția cursorului se află citind SReg.
- Imediat ce BF a devenit „0” poziția cursorului nu este cea corectă. Poziția se va actualiza după cel puțin 6 us. Informațiile despre acest comportament se găsesc în figura 10 de la pagina 25 din documentul HD44780U.pdf. Pentru a fi siguri că valoarea AC este corectă, așteptați 8 microsecunde.
- Pentru a aștepta perioade mici de timp, de ordinul microsecundelor, vom utiliza funcția:

```
void _delay_us(double __us)
```

Această funcție poate fi folosită dacă adăugăm următoarele două linii de cod **exact** în ordinea de mai jos:

```
#define F_CPU 8000000UL  
#include <util/delay.h>
```

Prototipul funcției este definit în *delay.h*. Pentru ca *_delay_us* să funcționeze corect trebuie specificată frecvența ceasului procesor. Specificarea se face prin intermediul constantei *F_CPU*.

- Cursorul nu trebuie să dispară în nici o situație! După ce scrieți un mesaj care are exact lungimea 16, cursorul trebuie să apară în pe linia 2, coloana 1.

Pentru testare, folosiți următoarea secvență de cod :

```

while(1){
    clrLCD(); // Ce se vede pe LCD
    wait(S2);
    putsLCD("0123456789abcdef"); //0123456789abcdef
    wait(S2); // _

    putcharLCD('g'); //0123456789abcdef
    wait(S2); //g_

    clrLCD(); // _
    wait(S2); //

    putsLCD("0123456789abcdefg"); //0123456789abcdef
    wait(S2); //g_

    putsLCD("hijklmnopqrstuv"); //0123456789abcdef
    wait(S2); //ghijklmnopqrstuv

    putcharLCD('w'); //w123456789abcdef
    wait(S2); //ghijklmnopqrstuv
}

```

Când funcționează, chemați profesorul pentru validare. Dacă ați ajuns aici între 1 și 2 puncte, în funcție de calitatea implementării.

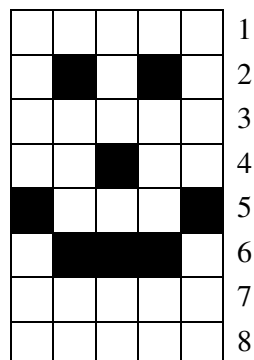
Pasul 6: Smiley

Pentru fiecare caracter se afișează o matrice de puncte. În tabelul 4 de la pagina 18 din documentul HD44780U.pdf se prezintă aceste matrice.

Este posibil ca utilizatorul să definească propriile matrice. Dacă dimensiunea matricei de afișare are dimensiunea 5x10, utilizatorul poate defini 4 matrice proprii. Aceste matrice se scriu în memoria grafică de caractere – CGRAM. Procesul de definire a acestor matrice este prezentat la pagina 19 din documentul HD44780U.pdf.

Correspondentul binar al liniilor care formează o configurație se scriu în **CGRAM** la adresa specificată de AC. Exact același mecanism este folosit și pentru scrierea în DDRAM. Destinația datelor scrise în DReg (Write), DDRAM sau CGRAM, este decisă de ultima instrucțiune **Set address** executată. După executarea unei instrucțiuni **Set DDRAM address** până la executarea altei instrucțiuni **Set address** toate datele scrise în DR (Write) vor ajunge în DDRAM. Similar, după executarea unei instrucțiuni **Set CGRAM address** toate datele scrise după aceea în DR (Write) vor ajunge în CGRAM.

Se cere ca pentru caracterul cu **codul ASCII 0x02** să apară modelul din figura următoare:



Verificați apelând *putchLCD(0x02)*. Trebuie să apară Smiley face din figura de mai sus.

```
while(1){
    clrLCD();
    wait(S2);

    putsLCD("This ");
    putchLCD(2);
    putsLCD(" is Smiley!");
    wait(S2);
}
```

Atenție!

- Scrierea în CGRAM se face **o singura data**, după inițializare (initLCD).
- Toate funcțiile scrise până acum trebuie să funcționeze și în continuare.

Când funcționează, chemați profesorul pentru validare.

Dacă ați ajuns aici între 1 și 3 puncte, în funcție de calitatea implementării.

Pasul 7: Deconectare

La terminare executați următoarele operații:

1. Se oprește execuția programului cu **Break**,
2. Se oprește depanatorul cu **Stop**
3. Se apasă **Con** și apoi se șterge memoria de program din ATmega16 cu **Erase**
4. Se închide AVR Studio.
5. Se oprește alimentarea JTAG ICE.
6. Se oprește sursa Hameg
7. Se desface montajul, numai componentele adăugate în acest laborator.