Cuprins

1. Programarea in C folosind WinAVR

WinAVR (pronuntat "whenever") este o solutie gratuita, open-source, pentru programarea in C, sub Windows a microcontrollere-lor Atmel AVR. Pachetul contine:

- Compilatorul gcc/g++ portat pentru microcontroller-ele AVR
- Avr-glibc (biblioteca standard portata pentru AVR)
- Gdb (debugger)
- Pachetul binutils (linker, stripper, etc.)
- Simulator pentru AVR
- Tool-uri pentru programarea device-urilor

1.1. Despre formatele fisierelor obiect

Asamblorul folosit pana acum genera direct imaginea care trebuia incarcata pe procesor (fisierul .hex). In general compilatoarele nu se comporta asa: ele dumpeaza codul si datele programului intr-un fisier obiect. Deci, dupa compilarea programului, partile relevante trebuie extrase din fisierul obiect si trasformate in format .hex.

Fisierele obiect joaca rol de container pentru diversele "bucati" ale programului. Avr-gcc foloseste un format de fisier obiect independent de arhitectura numit ELF (Executable Linkable Format). "Bucatile" de program sunt organizate in interiorul fisierului elf sub foma de blocuri continue de adrese denumite "sectiuni". Iata cateva sectiuni importante:

- `".text"` – reprezinta zona codului propriu-zis (corespunde in mare directive de asambare `.cseg`)
- `".data"` – contine zona de memorie rezervata stivei variabilelor programului (analog directivei `.dseg`)
- `".bss"` – zona de memorie a variabilelor statice initializate cu zero
- `".eeprom"` – date pe care compilatorul a decis sa le plaseze in memoria eeprom
- `".symtab"` – tabela de simboli a programului

- `".debug"` – zona de informatie pentru debugger (corespondenta dintre adrese de cod si linii de program, intre adrese de date si nume de variabile, etc.)

Pentru obtinerea imaginii de incarcat pe microcontroller, trebuie extrase sectiunile relevante din fisierul .elf. Din acest motiv compilarea unui program C trebuie realizata in trei pasi:

1. Compilarea propriu-zisa folosind gcc. Rezultatul este un fisier .elf
2. Extragerea sectiunilor ".text", ".data" si ".bss" intr-un fisier .hex
3. Extragerea sectiunii ".eeprom" intr-un fisier .eep.

1.3. Compilarea din linia de comanda

Pasul 1. compilarea sursei si obtinerea fisierului .elf:

```
avr-gcc –mmcu=atmega16 –Os –g –Wall –o test.elf test.c
```

Pasul 2. extragerea sectiunilor pentru imaginea memorie flash (fisierul .hex)

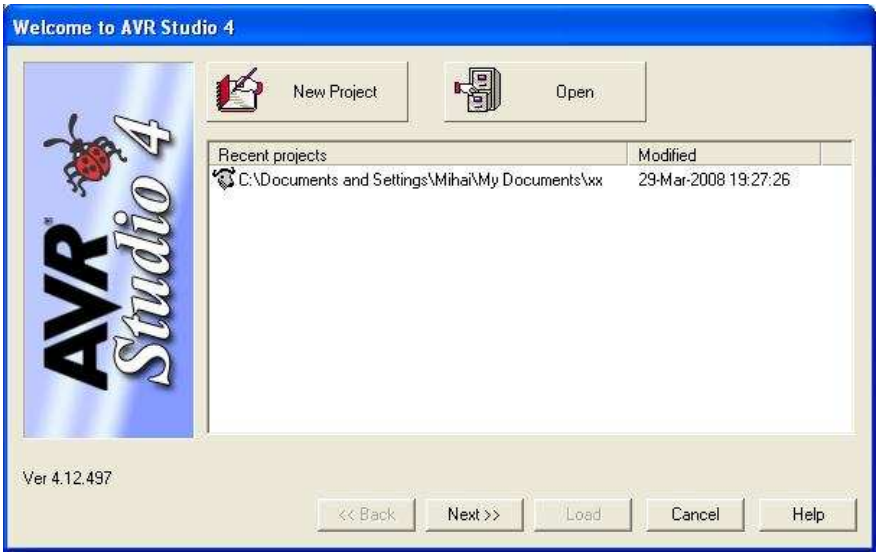
```
avr-objcopy -j .text -j .data -O ihex lab5.elf lab5.hex
```

Pasul 3. extragerea sectiunii eeprom si generarea fisierului .eep:

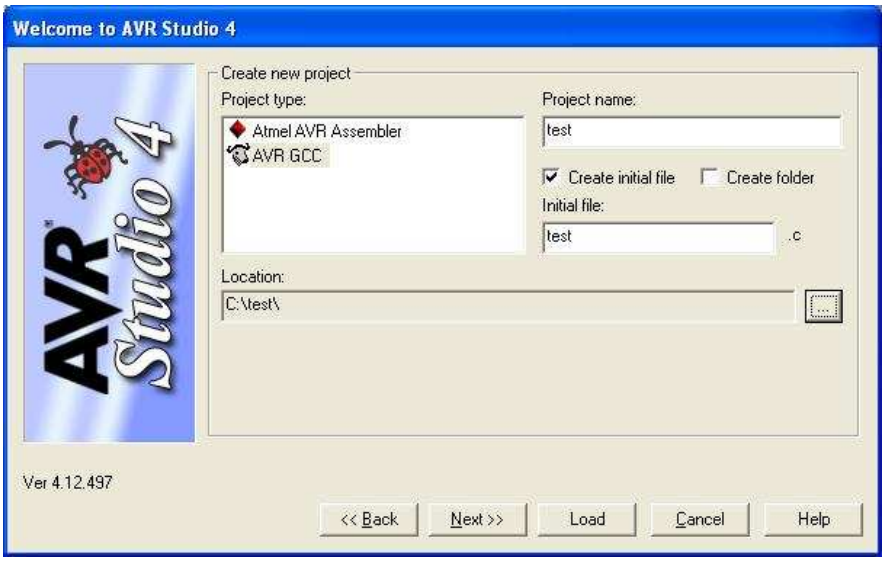
```
avr-objcopy -j .eeprom -O ihex lab5.elf lab5.eep
```

1.4. Compilarea din AVR Studio

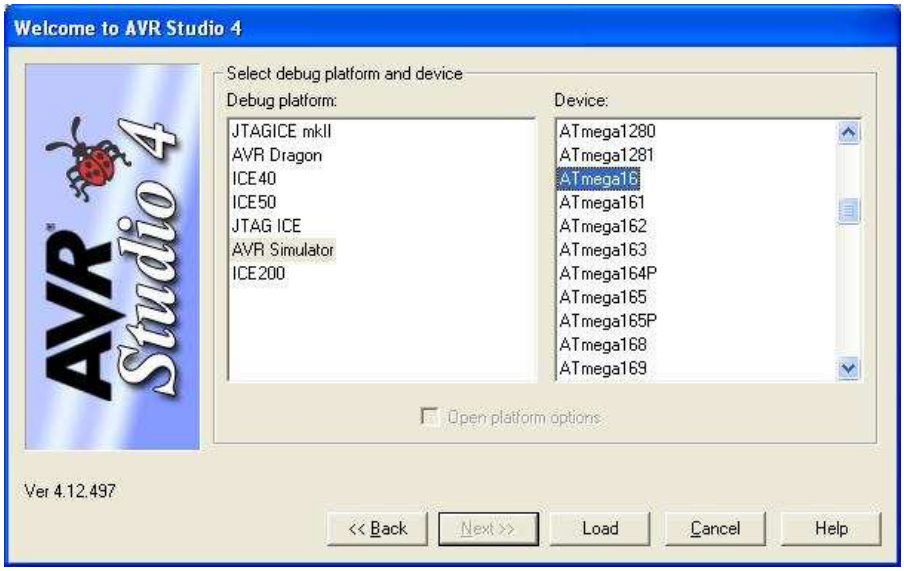
Pasul 1. Deschide AVR Studio si da click pe "New Project"



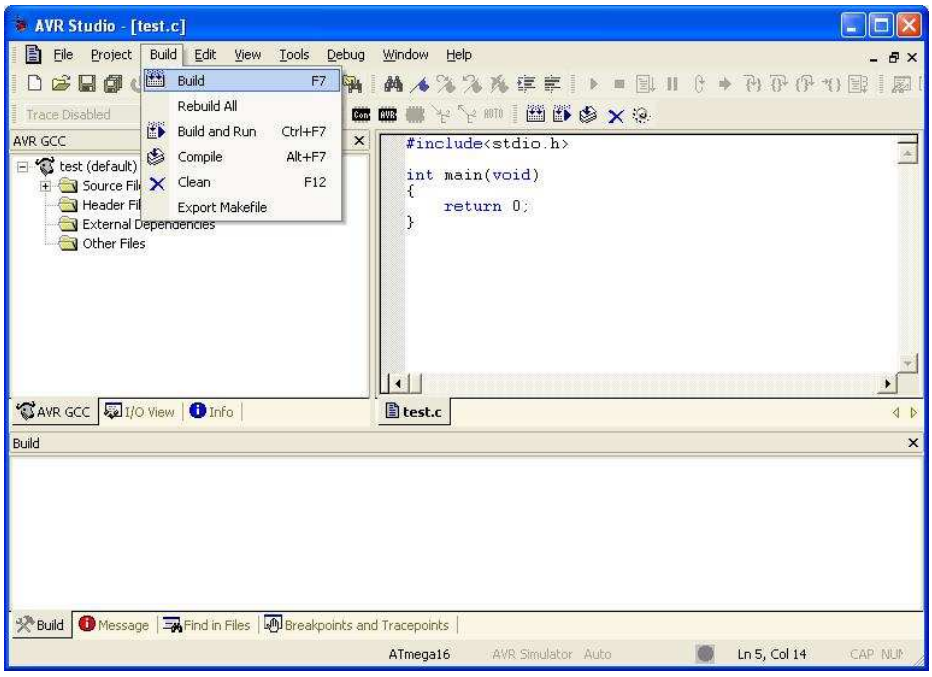
Pasul 2. Denumeste proiectul si alege-l un director. Alege ca tip al proiectului "AVR GCC".



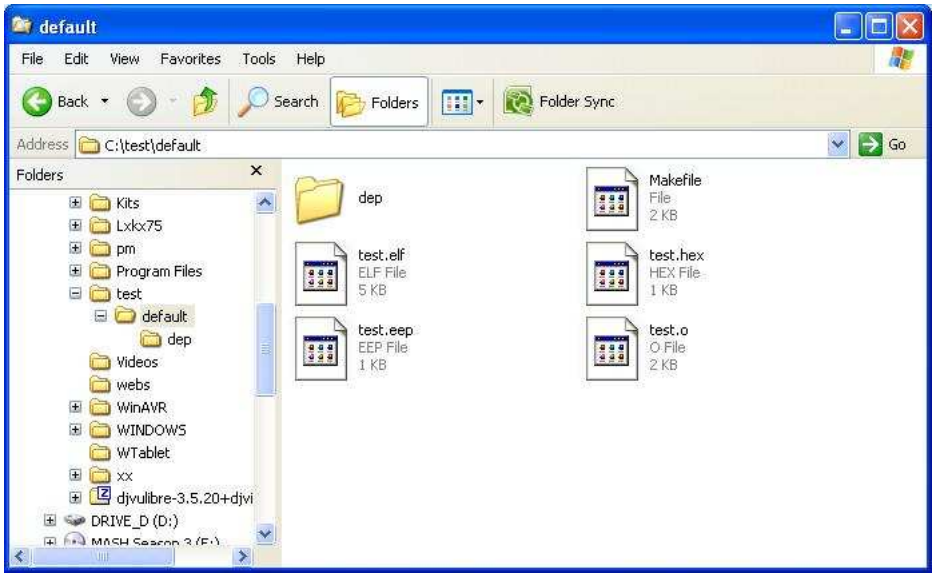
Pasul 3: Selecteaza simulatorul ca platform de debug si atmega16 ca device. Apoi click pe "Load".



Pasul 4: Scrie programul si alege din meniul "Build" optiunea "Build" (sau apasa F7)

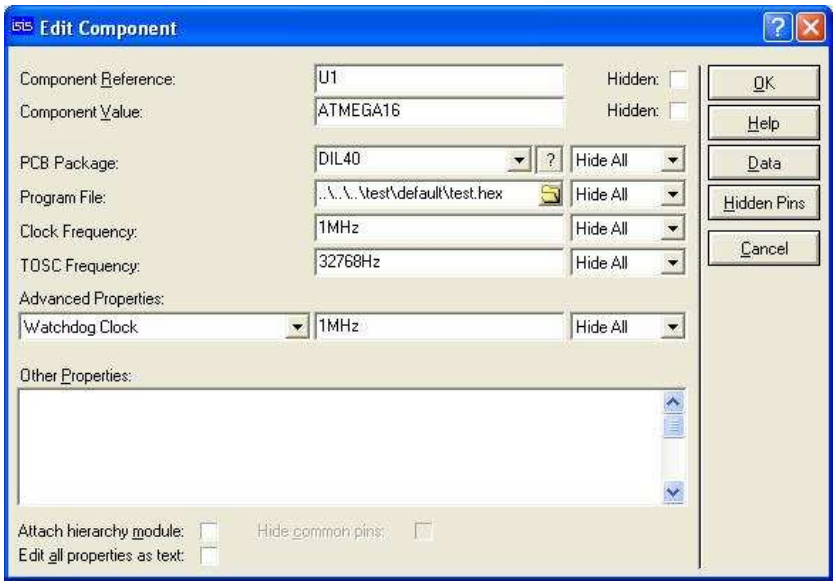


Pasul 5: In directorul creat pentru proiect, exista in folder numit "default" care contine fisierele .hex si .eep.



1.4. Simularea in Proteus

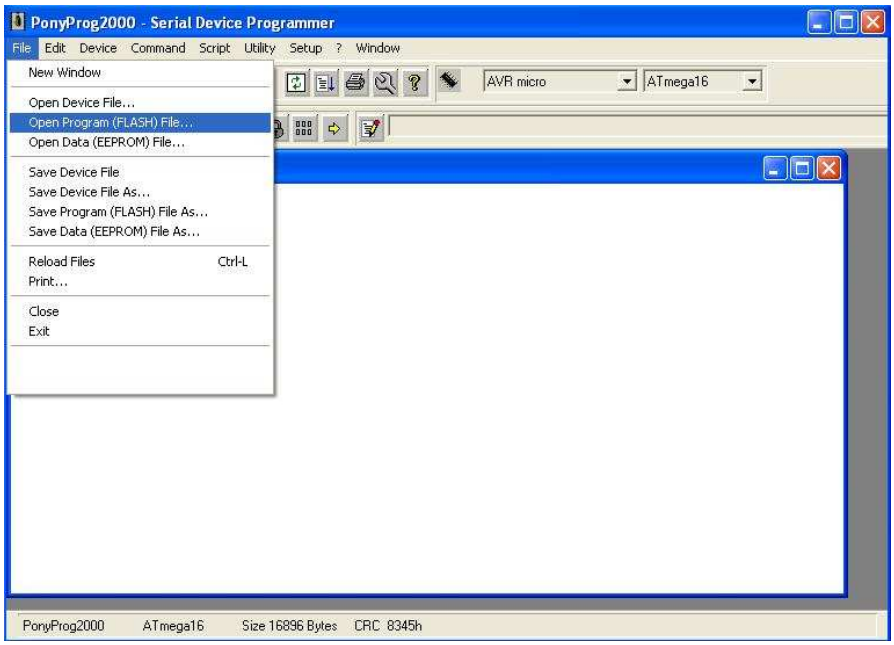
Pentru a simula aplicatia in Proteus, este suficient sa incarci hex-ul ca "Program file" in proprietatile procesorului. Nu adauga nimic in meniul "Source".



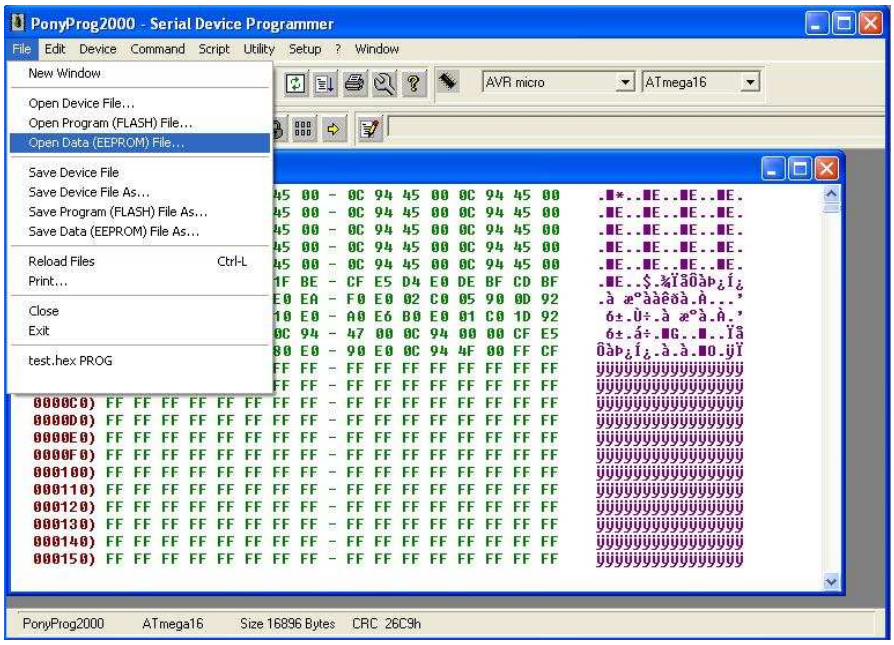
1.5. Incarcarea imaginii folosind Pony Prog

Programarea folosing Pony Prog se face la fel, atata doar ca trebuie incarcate si scrise ambele imagini (flash – fisierul .hex si eeprom – fisierul .eep).

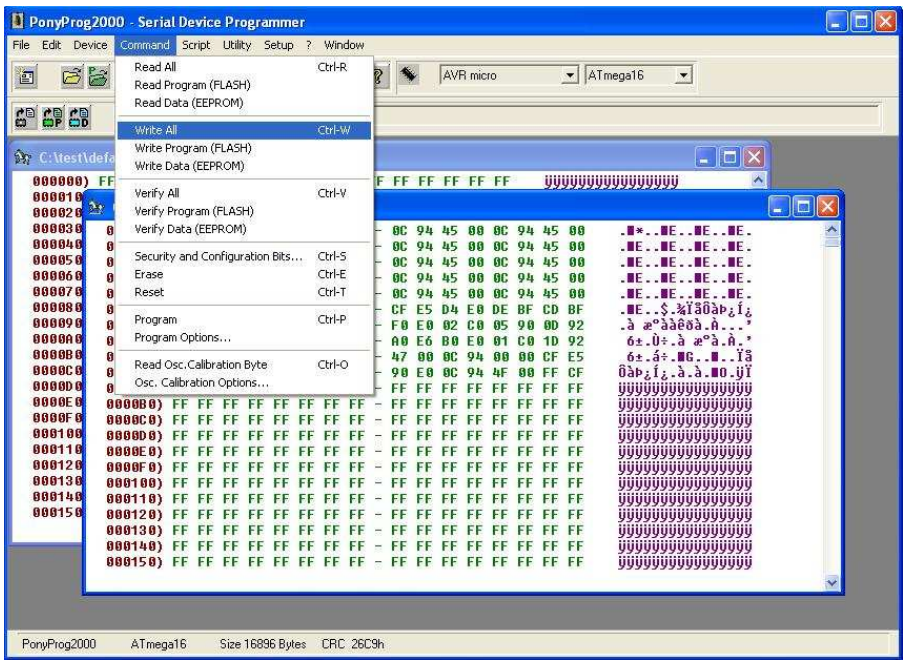
Pasul 1. Alege "File->Open Program (FLASH) File" si incarca fisierul .hex.



Pasul 2. Alege "File->New Window" apoi "File->Open Data (EEPROM) File" si selecteaza fisierul .eep.



Pasul 3. Alege “Command -> Write all”.



1.6. Porturi, Intreruperi, stdin si stdout

In C exista un mecanism foarte simplu pentru accesul la porturi, bazat pe faptul ca porturile microcontroller-ului sunt mapate in memorie (vedeti harta memoriei din datasheet). Intr-un fisierul antet “avr/io.h” exista constante definite ca pointeri la adresele de memorie in care sunt mapate porturi. Constanta este denumita precum portul corespunzator adresei, astfel ca se poate scrie:

```
PORTA = 0xFF; char x = DDRB; TIMSK |= 0x40;
```

Pentru a scrie, citi sau modifica valoarea aflata intr-un port.

In C nu exista un mechanism built-in pentru scrierea valorilor in binar. Pentru a compensa acest neajuns, biblioteca pune la dispozitie macro definitia `_BV`. De exemplu, pentru a incarca in portul UCSRB o valoare care are bitii 7, 5 si 3 setati, se poate scrie:

```
UCSRB = _BV(7) | _BV(5) | _BV(3);
```

Lucrul cu intreruperi este mai simplu decat in asamblare. Nu este necesara plasarea unui jump in tabela de intreruperi si nici salvarea explicita a registrilor. In C se poate scrie:

```
SIGNAL(SIG_OUTPUT_COMPARE2) { do_stuff(); i++; k--; }
```

pentru a defini, de exemplu, rutina de tratare a intreruperii generate de comparatorul timer-ului 2. SIGNAL este un macro cu ajutorul caruia se definesc functiile de tratare ale intreruperilor. SIG_OUTPUT_COMPARE2 este o constanta care identifica sursa intreruperii (comparatorul timer-ului 2). Atat SIGNAL cat si constantele sunt definite in fisierul antet "avr/interrupt.h".

Functiile printf si scanf au functionalitatea cunoscuta, dar pentru a putea fi folosite trebuie initializate stream-urile stdin si stdout. Initializarea se face cu ajutorul macro-ului _FDEV_SETUP_STREAM:

```
static my_stdout = FDEV_SETUP_STREAM(write_char, NULL,
                   _FEV_SETUP_WRITE);

static my_stdin = FDEV_SETUP_STREAM(NULL, read_char,
                  _FDEV_SETUP_READ);
```

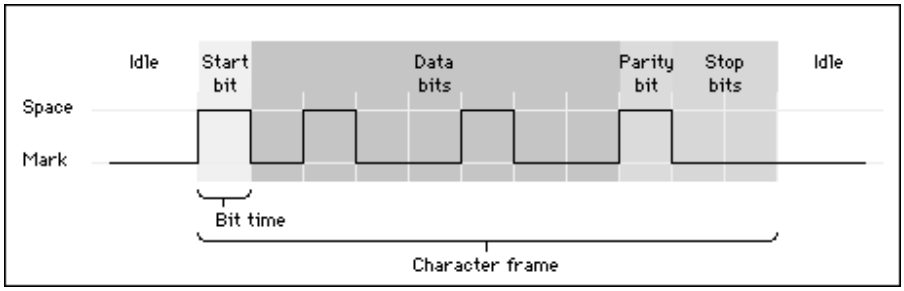
Cele doua declaratii defines my_stdout si my_stdin ca doua strimuri in/din care se scrie/citeste un caracter folosind write_char, respective read_char. Aceste doua functii trebuie implementate separat. In functia main se face apoi initializarea:

```
stdin = &my_stdin; stdout = &my_stdout;
```

Din acest moment printf si scanf vor lucre pe cele doua stream-uri definite de noi. Implicatia este ca putem folosi orice dispozitiv de afisare pentru stdout si orice dispozitiv de intrare pentru stdin, atata timp cand stim cum sa scriem functii care scriu/citesc un singur caracter.

2. Interfata Seriala

Un frame transmis pe interfata serial are urmatorul format:



Se transmite un bit de start, apoi un cuvant de date. Urmeaza un bit optional de partitate si unul sau doi biti de stop. Microcontrollerul ATMEGA16 include un controller pentru interfata seriala, care este controlat de registrii descrisi in sectiunile urmatoare

2.1. UDR

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXB si TXB sunt buffere-le de receptie, respectiv transmisie. Ele folosesc aceeasi adresa de I/O. Deci RXB este accesat citind din UDR, TXB scriind in UDR. Bufferul de transmisie poate fi scris numai atunci cand bitul UDRE din portul UCSRA este 1. In caz contrar, scrierile vor fi ignorate.

2.2. UCSRA

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSRA este primul registru de stare al controllerului de comunicatie. Bitii cei mai importanti sunt:

- RXC – Receive Complete – devine 1 cand exista date primite si necitite. Cand bufferul de receptie este gol, bitul este resetat automat
- TXC – Transmit Complete – devine 1 cand bufferul de transmisie devine gol

- UDRE – Data Register Empty – devine 1 cand bufferul de transmisie poate accepta noi date

2.3. UCSRB

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UCSRB este al doilea registru de control. Bitii importanti:

- RXCIE – Receive Complete Interrupt Enable – cand este 1, controllerul de comunicatie va genera o intrerupere cand au fost primite date
- TXCIE – Transmit Complete Interrupt Enable – cand este 1, controllerul de comunicatie va genera o intrerupere cand bufferul de transmisie devine gol
- UDRIE – Data Register Empty Interrupt Enable – cand este 1, controllerul de comunicatie va genera o intrerupere cand bufferul de transmisie mai poate accepta date
- RXEN – Receiver Enable – daca este 0, nu se pot recepta date
- TXEN – Transmitter Enabler – daca este 0, nu se pot transmite date
- UCSZ2 – impreuna cu UCSZ1 si UCSZ0 din portul UCSRC, selecteaza dimensiunea unui cuvant de date

2.4. UCSRC

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Al treilea registru de control. Bitii importanti:

- URSEL – Register Select – trebuie sa fie 1 cand se scrie in UCSRC
- UMSEL – Mode Select – 0 pentru functionare asincrona, 1 pentru functionare sincrona

- UPM1, UPM0 – Parity Mode

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- USBS – Stop Bit Select – 0 pentru un bit de stop, 1 pentru doi biti de stop
- UCSZ1, UCSZ0 – impreuna cu UCSZ2 din portul UCSRB, decid dimensiunea cuvantului de date

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

2.5. UBRR

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRR este registrul care decide baud rate-ul. Are 12 biti. Primii 4 se afla in UBRRH, ceilalti 8 in UBRRL. Valoarea de scris in UBRR depinde de frecventa procesorului si de baud rate-ul dorit. In tabelul de dedesubt gasiti valorile pentru frecventa de 16 Mhz. Bitul

URSEL trebuie sa fie 0 cand se scrie in UBRRH (si este, pentru ca valoarea din UBRRH incape mereu 4 biti)

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max (1)	1 Mbps		2 Mbps	

3. Interfatarea unui LCD alfanumeric

3.1. Introducere

Display-urile, si in principal cele fabricate in tehnologie LCD, reprezinta una din cele mai folosite moduri pentru a efectua debugging asupra unui circuit, de a oferi informatii utilizatorului sau de a oferi un aspect profesional unui dispozitiv.

Unul din cele mai uzuale controllere LCD este Hitachi 44780 ce ofera o modalitate simpla de interfatare intre un microcontroller si ecranul LCD. Din punct de vedere al costului dispozitivele ce se bazeaza pe controllerul Hitachi 44780 sunt de obicei relative ieftine sau pot fi recuperate din dispozitivele mai vechi si astfel refolosite.

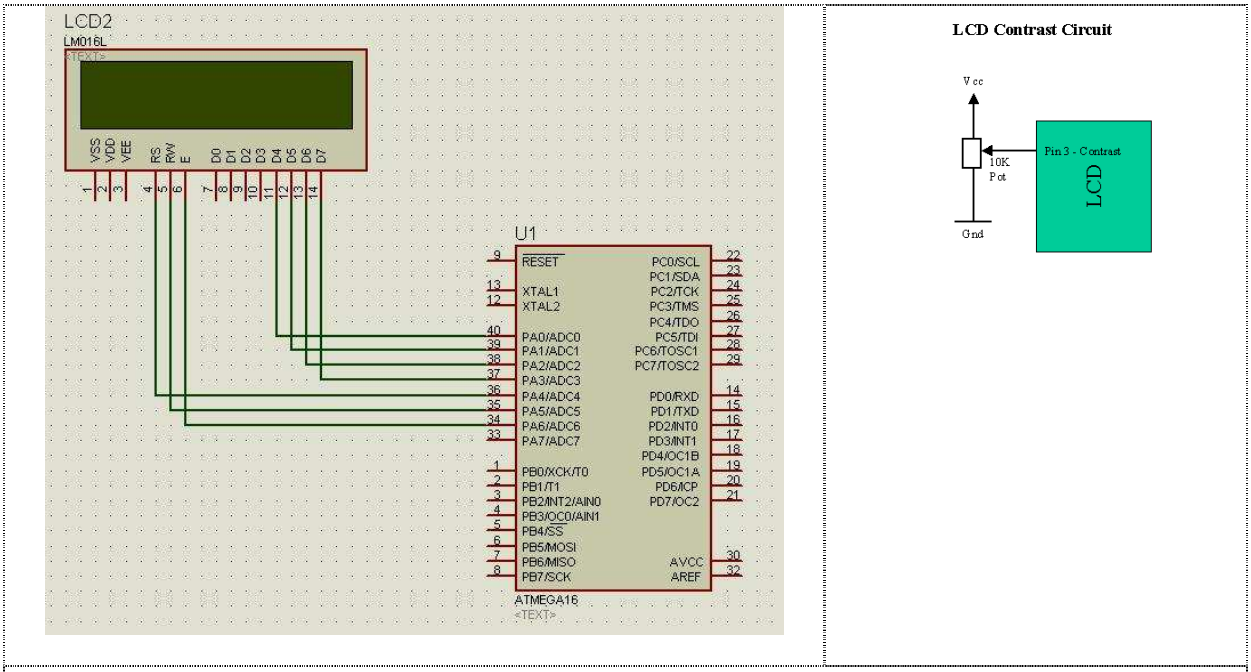
3.2. Interfata de conectare

Cel mai usual conector pentru controller tip 44780 este format din 14 pini dispuse pe o linie, cu distant dintre pini de 0.1'' (inch). Semnificatia pinilor este urmatoarea:

PIN	Descriere
1	Impamantare (Ground)
2	Alimentare +5V (Vcc)
3	Contrast (0V=contrast maxim, 5V=contrast minim)
4	Pin R/S (se seteaza daca se transmit instructiuni sau date)
5	Pin R/W (se seteaza daca se efectueaza o citire sau o scriere catre LCD)
6	Pin E (Enable, se semnalizeaza cand se pot prelua datele de pe celelalte fire spre interpretare si executie)
7-14	Pinii de date, I/O

Precum se poate observa si din tabelul de mai sus interfata de communicatie este una paralela, permitand astfel sa se efectueaza scrieri sau citiri de date intr-un mod simplu si rapid. Pe cele 8 fire de date se transmit octeti (grupuri de 8 biti), acest octeti reprezentand coduri interne ale LCD-ului (in modul de instructiuni) sau coduri ASCII ale caracterelor ce se doresc a fi afisate (in modul de date).

Din punct de vedere hardware interfatarea se efectueaza pin la pin cu microcontrollerul, precum in schema de mai jos. Exemplul presupune o interfatare pe 4 fire. In cazul in care se doreste interfatare pe 8 fire este sufficient sa se contecteze toti pinii D0-D7 la acelasi port si cei de control DR, RW si E pe un alt port sis a se scrie programul tinand cont de aceasta structura. Pinii VSS si VDD se conecteaza la masa si alimentare. Pinul 3, contrast, se contecteaza direct la masa, pentru contrast maxim, sau int-run potentiometru pentru a putea regal contrastul, precum in scehama de mai jos.

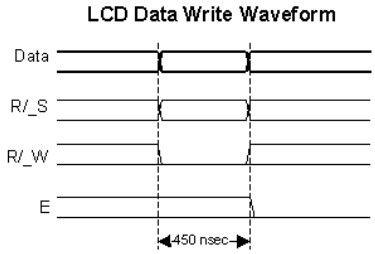


Exemplu de schema pt interfatare LCD Hitachi 44780 – AT Mega 16

3.3. Procedura de comunicatie

Din punct de vedere al modului de comunicatie trebuie urmati urmatorii pasi:

1. Se seteaza daca se doreste a se trimite o instructiune sau un character de date (pinul R,_S)
2. Se seteaza daca se doreste sa se citeasca informatii de la LCD sau sa se trimita informatii (date sau instructiuni dupa cum s-a selectat mai sus) (pinul R,_W).
3. Se pun datele pe busul Data (pinii D0-D7).
4. Se activeaza o perioada de timp pinul E (ENABLE).



In cazul in care se allege o interfatare pe 4 fire de date atunci se modifica procedura de mai sus prin faptul ca intai se pun pe busul de 4 fire cei mai importanti 4 biti, se activeaza o perioada de timp pinul E, dupa care se pun cei mai putin semnificativi 4 biti pe bus si se activeaza idn nou o perioada de timp pinul E (modificarea pasilor 3 si 4).

3.4. Instructiuni suportate

Lcd-urile ce se bazeaza pe controllerul Hitachi 44780 suporta un anumit set de instriuctiuni ce se trimit catre LCD atunci cand se doreste trecerea intr-un alt mod de operare, stergerea ecrantului, avansarea pe urmatoarea linie etc. Asa cum sunt ele descries in datasheet-ul Hitachi 44780, instructiunile principale sunt urmatoarele:

Instruction	Code RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f_{cp} or f_{osc} is 270 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function set	0	0	0	0	1	DL	N	F	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	AC	AC	AC	AC	AC	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs

Descrierile bitilor din tabelul de mai sus sunt:

Setarea directiei de deplasare a cursorului:

ID – incrementeaza cursorul dupa fiecare octet ce este scris (daca este setat)

S – shifteaza displayul atunci cand este scris un caracter pe acesta

Activeaza display-ul/ cursorul

D – Activeaza displayul On(1)/Off(0)

C – Activeaza cursorul On(1)/Off(0)

B – Seteaza cursorul pe blink On(1)/Off(0)

Mutarea cursorului/ Shiftarea displayului

SC – Activeaza shiftarea displayului On(1)/Off(0)

RL – Seteaza directia de shiftare la dreapta Right(1)/Left(0)

Setarea dimensiunii interfetei

DL – Latimea interfetei de conectarea 8 fire(1)/4 fire(0)

N – Numarul de linii afisare de display 1 linie(0)/2 linii(1)

F – Fontul caracterelor afisate 5x10(1)/5x7(0(

Activarea flag-ului "Busy Flag"

BF – Acest bit se seteaza atunci cand LCD-ul proceseaza informatiile primate si nu poate procesa altele in acelasi timp

Mutarea cursorului catre CGRAM/Display

A - Adresa

Citire/Scriere cod ASCII catre Display

D - Date

3.5. API Lcd

Pentru a va fi mai usoara interfatarea unui modul LCD am scris pentru voi o mica librarie de functii (API) pe care le puteti folosi atat pentru realizarea laboratorului cat si in dezvoltarea proiectului vostru. Functiile pot fi folosite ca atare sau modificate ca sa indeplineasca necesitatile proiectului.

Functiile principale implementate sunt:

void LCD_init();
Initializare modul LCD.. trebuie apelata inainte de a se face orice operatie cu LCD-ul.
Initializarea este facuta considerand o interfatare pe 4 fire !!!.
void LCD_writeInstruction(unsigned char _instruction);
Trimite o instructiune catre lcd (vezi datasheet)
void LCD_writeData(unsigned char _data);
Trimite date catre LCD pentru afisare
void LCD_write(unsigned char _byte);
Trimite un byte catre LCD pe pinii D4-7 in 2 timpi, intai bitii mai semnificativi si apoi cei mai putin semnidifcativi.

void LCD_waitNotBusy();

Functia asteapta pana cand lcd-ul devine disponibil pt o noua comanda.

void LCD_print(char* _msg);

Afiseaza imformatia pe LCD (doar 1 linie, primele 16 caractere din msg)

void LCD_print2(char* _msg1, char* _msg2);

Afisare pe 2 lini pe LCD. Pe prima linie afiseaza _msg1 si pe a 2-a _msg2.

void LCD_printDecimal2u(unsigned int _n);

Afisare numar in baza 10 pe LCD

void LCD_printHexa(unsigned int _n);

Afisare numar in baza 16 pe LCD

void LCD_waitInstructions(unsigned char _instructions);

Asteapta un numar de cicli de tact.. loop

Pentru mai multe detalii consultati direct codul sursa lab3.c.