Lucrarea 1
Microcontrolerul ATmega32. Utilizarea porturilor

Scopul lucrării

a) Noţiuni generale despre ATmega32
b) Setul de instrucţiuni al ATmega32
c) Porturile ATmega32 şi exemple de utilizare
d) Programarea unor aplicaţii utilizând placa de dezvoltare EasyAVRv7

1. Noţiuni generale

ATmega32 este un microcontroler RISC pe 8 biţi realizat de firma Atmel. Caracteristicile principale ale acestuia sunt:

- 32KB de memorie Flash reinscriptibilă pentru stocarea codului
- 2KB de memorie RAM
- 1KB de memorie EEPROM
- două numărătoare/temporizatoare de 8 biţi
- un numărător/temporizator de 16 biţi
- conţine un convertor analog – digital cu intrări multiple
- conţine un comparator analogic
- conţine un modul USART pentru comunicaţie serială
- dispune de oscilator intern
- oferă 32 de linii I/O organizate în patru porturi.

Structura internă generală a controlerului este prezentată în Figura 1. Se poate observa că există o magistrală generală de date la care sunt conectate mai multe module:

- unitatea aritmetică şi logică (ALU)
- registrele generale
- memoria RAM şi memoria EEPROM
- liniile de intrare (porturile – I/O Lines) şi celelalte blocuri de intrare/ieşire. Aceste ultime module sunt controlate de un set special de registre, fiecare modul având asociat un număr de registre specifice.

Memoria Flash de program împreună cu întreg blocul de extragere a instrucţiunilor, decodare şi execuţie comunică printr-o magistrală proprie, separată de magistrala de date menţionată mai sus. Acest tip de organizare este conform principiilor unei arhitecturi Harvad şi permite controlerului să execute instrucţiunile foarte rapid.

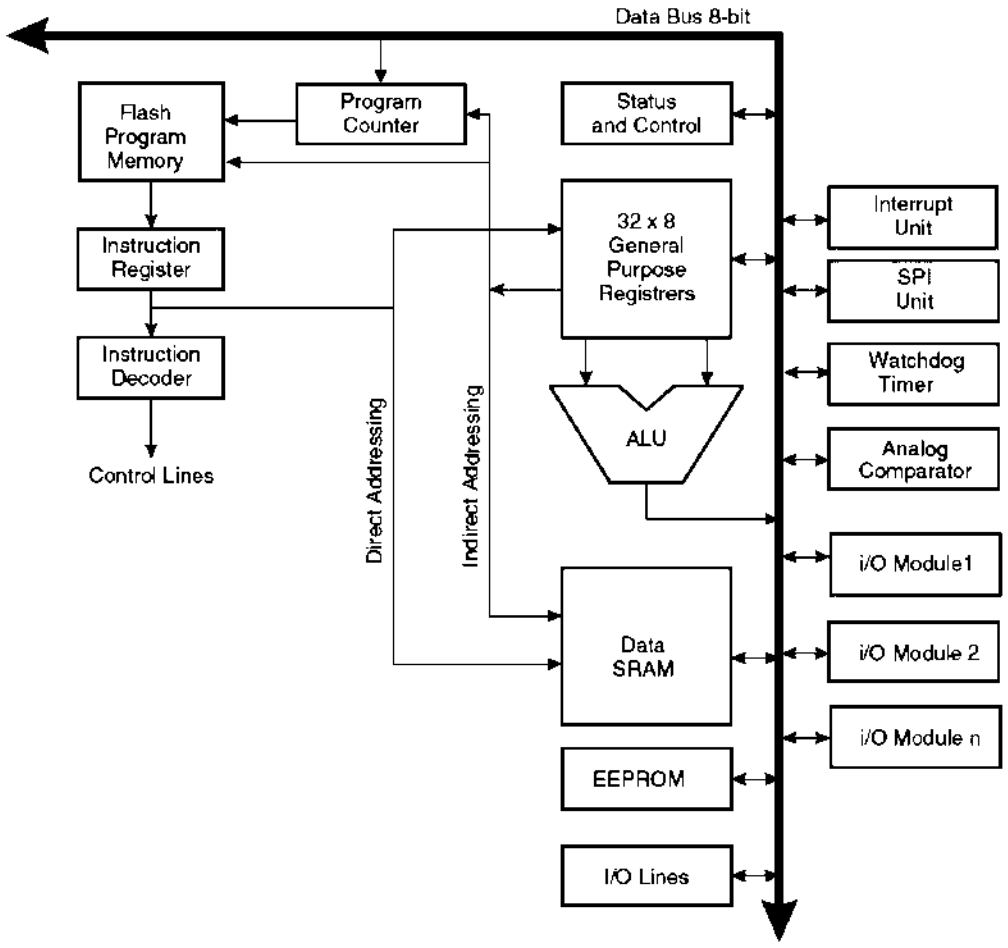


Figura 1. Structura bloc generală a microcontrolerului ATmega32

ATmega32 conţine 32 de registre de uz general şi 64 de registre speciale pentru modulele I/O. Aceste registre sunt mapate la adrese din memoria RAM cuprinse între 0000h şi 005Fh.

Una din caracteristicile foarte importante pentru orice procesor şi în particular pentru microcontrolere este sistemul de întreruperi. O întrerupere reprezintă un semnal generat de un modul extern unităţii centrale de procesare (CPU) pentru a anunţa apariţia unui eveniment care trebuie tratat. Utilizarea unui astfel de sistem permite implementarea de module specializate care să execute operaţii în paralel cu CPU şi să solicite intervenţia acestuia numai la terminarea operaţiilor sau în alte cazuri definite.

ATmega32 dispune de 21 surse de întrerupere. Atunci când una dintre ele devine activă se suspendă cursul normal de execuţie şi se face salt automat la o adresă prestabilită din memoria program. Astfel, structura tipică a unui program conţine la adresele respective instrucţiuni care apelează procedurile create special pentru fiecare întrerupere:

jmp RESET
jmp EXT_INT0
jmp EXT_INT1
jmp EXT_INT2
jmp TIM2_COMP
jmp TIM2_OVF
jmp TIM1_CAPT

```
jmp TIM1_COMPA
jmp TIM1_COMPB
jmp TIM1_OVF
jmp TIM0_COMP
jmp TIM0_OVF
jmp SPI_STC
jmp USART_RXC
jmp USART_UDRE
jmp USART_TXC
jmp ADC
jmp EE_RDY
jmp ANA_COMP
jmp TWI
jmp SPM_RDY
RESET:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
```

Dacă una din întreruperi nu este utilizată se recomandă ca pe poziţia acesteia să se introducă o instrucţiune de salt către eticheta RESET.

Primele patru instrucţiuni care apar după eticheta RESET sunt utilizate pentru a iniţializa pointer-ul de stivă definit de registrele SPH şi SPL.

2. Setul de instrucţiuni

ATmega32 dispune de un registru special de stare ai cărui biţi oferă informaţii despre rezultatul ultimei instrucţiuni aritmetice sau logice executate. Componenţa registrului de stare este prezentată în Figura 2.

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
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	

Figura 2. Registrul de stare

Registrul de stare SREG conţine următorii biţi:

- bitul 7 – activare/dezactivare globală întreruperi

Acest bit trebuie să aibă valoarea „1” pentru a permite execuţia oricărei întreruperi. Dacă are valoarea „0” orice sursă de întrerupere este dezactivată.

- bitul 6 – bit de copiere
- bitul 5 – indicator de transport la jumătate
- bitul 4 – indicator de semn
- bitul 3 – indicator de depăşire în cazul operaţiilor în complement faţă de doi
- bitul 2 – indicator de rezultat negativ
- bitul 1 – indicator de zero

Acest bit devine „1” dacă rezultatul unei operaţii aritmetice sau logice a fost zero.

- bitul 0 – indicator de transport

Acest bit devine „1” în cazul unei depăşiri la operaţii pe 8 biţi.

ATmega32 dispune de un set de 131 de instrucţiuni grupate în mai multe categorii ce vor fi prezentate în continuare.

2.1. Instrucţiuni aritmetice şi logice

Cele mai uzuale instrucţiuni aritmetice şi logice sunt:

- **ADD Rd,Rr**

Efectul: se adună conţinutul registrului Rd cu cel al registrului Rr şi rezultatul se pune în Rd.

- **ADC Rd,Rr**

Efectul: se adună conţinutul registrului Rd cu cel al registrului Rr şi cu indicatorul de transport şi rezultatul se pune în Rd.

- **SUB Rd,Rr**

Efectul: se scade conţinutul registrului Rr din cel al registrului Rd şi rezultatul se pune în Rd.

- **AND Rd,Rr**

Efectul: se face „ŞI logic” între conţinutul registrului Rd şi cel al registrului Rr iar rezultatul se pune în Rd.

- **INC Rd**

Efectul: incrementează conţinutul registrului Rd şi pune rezultatul în Rd.

- **DEC Rd**

Efectul: decrementează conţinutul registrului Rd şi pune rezultatul în Rd.

2.2. Instrucţiuni de salt

Cele mai uzuale instrucţiuni de salt sunt:

- **JMP k**

Efectul: se face salt necondiţionat cu „k” poziţii faţă de adresa curentă din memoria program. Instrucţiunea se poate utiliza şi în forma „JMP etichetă”.

- **RCALL subrutină**

Efectul: se apelează o subrutină. Pentru a reveni din aceasta se utilizează instrucţiunea RET.

- **RETI**

Efectul: se revenire dintr-o subrutină de tratare a unei întreruperi.

- **CPSE Rd,Rr**

Efectul: se compară valorile regiştrilor Rd şi Rr şi în caz de egalitate instrucţiunea imediat următoare nu se mai execută.

- **CP Rd,Rr**

Efectul: compară valorile regiştrilor Rd şi Rr fără să modifice conţinutul acestora. Ca urmare a execuţiei acestei instrucţiuni se vor schimba corespunzător biţii registrului de stare.

- **SBRC Rd,b**

Efectul: dacă bitul b din registrul Rd are valoarea „0” instrucţiunea imediat următoare nu se mai execută.

- **SBRS Rd,b**

Efectul: dacă bitul b din registrul Rd are valoarea „1” instrucţiunea imediat următoare nu se mai execută.

- **SBIC P,b**

Efectul: dacă bitul b din registrul de intrare/ieşire P are valoarea „0” instrucţiunea imediat următoare nu se mai execută.

- **SBIS P,b**

Efectul: dacă bitul b din registrul de intrare/ieşire P are valoarea „1” instrucţiunea imediat următoare nu se mai execută.

- **BREQ etichetă**

Efectul: dacă indicatorul Z are valoarea „1” se face salt la etichetă.

2.3. Instrucţiuni de transfer

Cele mai uzuale instrucţiuni de transfer sunt:

- **MOV Rd,Rr**

Efectul: copiază conţinutul registrului Rr în registrul Rd.

- **LDI Rd,k**

Efectul: copiază valoarea k în registrul Rd. Această instrucţiune lucrează numai cu registrele r16 – r31.

- **IN Rd,P**

Efectul: copiază conţinutul registrului de intrare/ieşire P în registrul Rd.

- **OUT P,Rr**

Efectul: copiază conţinutul registrului Rr în registrul de intrare/ieşire P.

2.4. Instrucţiuni care lucrează la nivel de bit

Cele mai uzuale instrucţiuni care lucrează la nivel de bit sunt:

- **SBI P,b**

Efectul: bitul b din registrul de intrare/ieşire P ia valoarea „1”. Instrucţiunea se poate utiliza numai pentru regiştri P situaţi la adresele de memorie 20h – 3Fh.

- **CBI P,b**

Efectul: bitul b din registrul de intrare/ieşire P ia valoarea „0”. Instrucţiunea se poate utiliza numai pentru regiştri P situaţi la adresele de memorie 20h – 3Fh.

- **LSL Rd**

Efectul: Registrul Rd este deplasat logic la stânga cu o poziţie.

- **ROR Rd**

Efectul: Registrul Rd este rotit la dreapta cu o poziţie prin bitul indicator de transport.

2.5. Instrucţiuni speciale

Instrucţiunile speciale sunt:

- **NOP**
- **SLEEP**
- **WDR**

3. Porturile ATmega32

3.1. Prezentare generală

ATmega32 dispune de 32 de linii de I/O grupate în patru porturi de 8 biţi. Porturile sunt denumite cu literele A, B, C şi D. Fiecare pin al oricărui port se poate seta individual ca intrare sau ieşire fără să afecteze ceilalţi pini. In plus, anumiţi pini

se pot utiliza pentru funcţii speciale ale microcontrolerului. In Figura 3 este prezentată structura generală a unui pin.

Toţi cei 32 de pini au fiecare câte o rezistenţă „pull-up” care poate fi activată sau dezactivată.

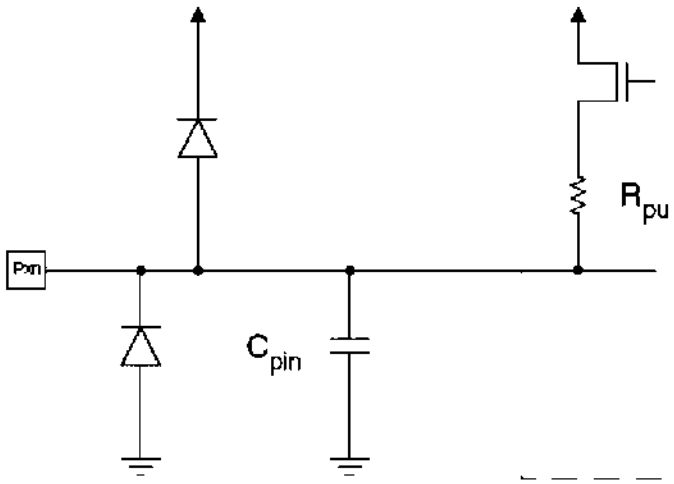


Figura 3. Structura generală a unui pin

Operaţiile cu porturile se fac prin intermediul unui set de patru regiştri alocaţi fiecărui port: PORTx, PINx şi DDRx; x poate fi A, B, C sau D. Aceşti regiştri fac parte din categoria regiştri de intrare/ieşire de aceea instrucţiunile care pot lucra direct cu ei sunt CBI, SBI, IN şi OUT.

Regiştrii DDRx stabilesc dacă un pin este intrare sau ieşire. Astfel, un bit cu valoarea „1” în registrul DDRx face ca pinul corespunzător să fie considerat ieşire; altfel pinul va fi intrare.

Regiştrii PORTx sunt utilizaţi pentru a scrie o valoare în portul corespunzător iar regiştrii PINx se folosesc pentru a citi valoarea prezentă pe pinii unui port.

Activarea sau dezactivarea rezistenţelor „pull-up” este determinată de bitul PUD din registrul SFIOR şi de valorile regiştrilor PORTx şi DDRx aşa cum se poate observa în tabelul următor.

Bitul din DDRx	**Bitul din PORTx**	**Bitul PUD**	**Tipul pinului**	**Rezistenţă pull-up**
0	0	nu contează	intrare	inactivă
0	1	0*	intrare	activă
0	1	1	intrare	inactivă
1	0	nu contează	ieşire	inactivă
1	1	nu contează	ieşire	inactivă

* Valoare implicită

3.2. *Prezentare generală a mediului software Atmel Studio*

Atmel Studio este un mediu software dezvoltat de Atmel pentru scrierea în limbaj de asamblare sau C/C++, compilarea şi simularea de programe destinate microcontrolerelor produse de această companie.

Orice program scris în Atmel Studio este conţinut într-o structură de tip proiect. Paşii care trebuie urmaţi pentru a crea un program nou sunt:

- după lansarea programului, apare o fereastră similară cu cea din Figura 4. Se va alege opţiunea „New Project";

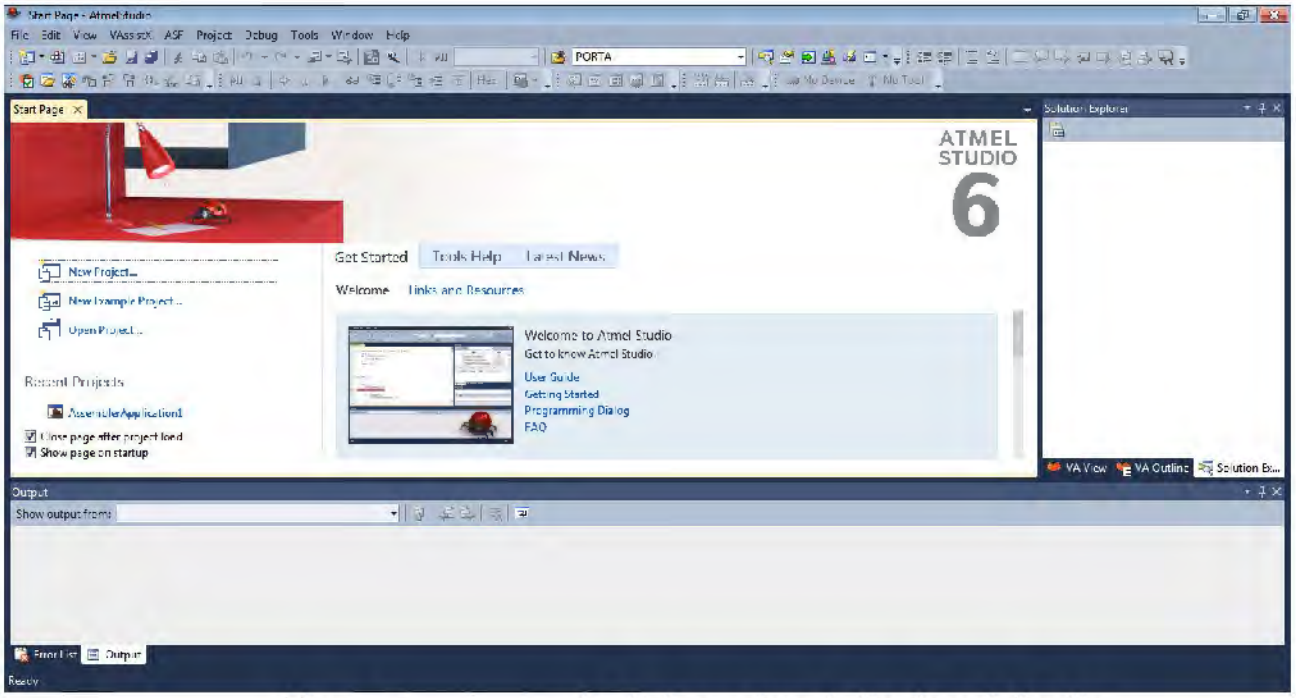


Figura 4. Fereastra de dialog primară din Atmel Studio

- în acest moment se va afişa o fereastră asemănătoare cu cea din Figura 5. Se selectează ca Template „Assembler" şi apoi „AVR Assembler Project" şi se bifează opţiunea „Create directory for solution". Apoi se completează câmpurile „Name" şi „Location" şi se apasă butonul „OK";

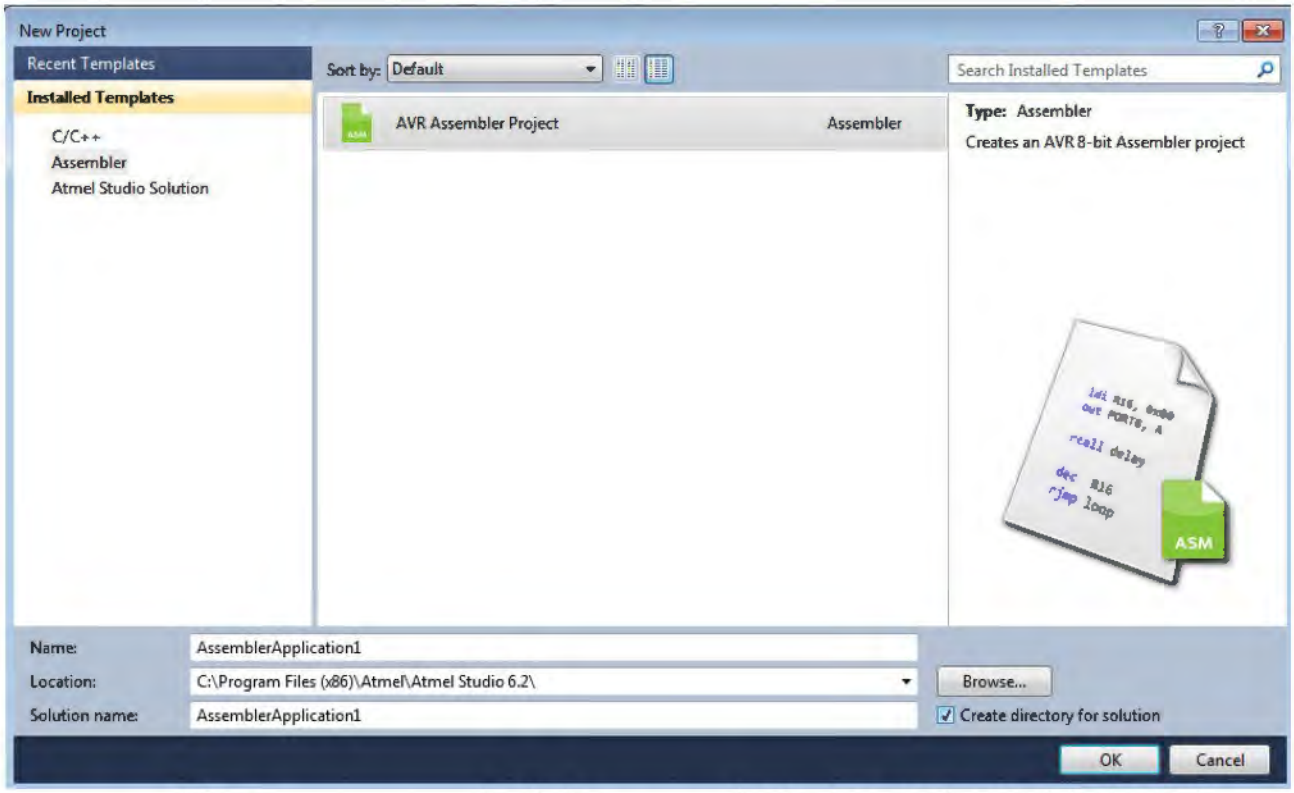


Figura 5. Fereastra de dialog pentru denumirea proiectului (solution)

- din fereastra care apare se selectează din listă modelul de microcontroler – în cazul acestui laborator „ATmega 32" şi în final se apasă butonul „OK";

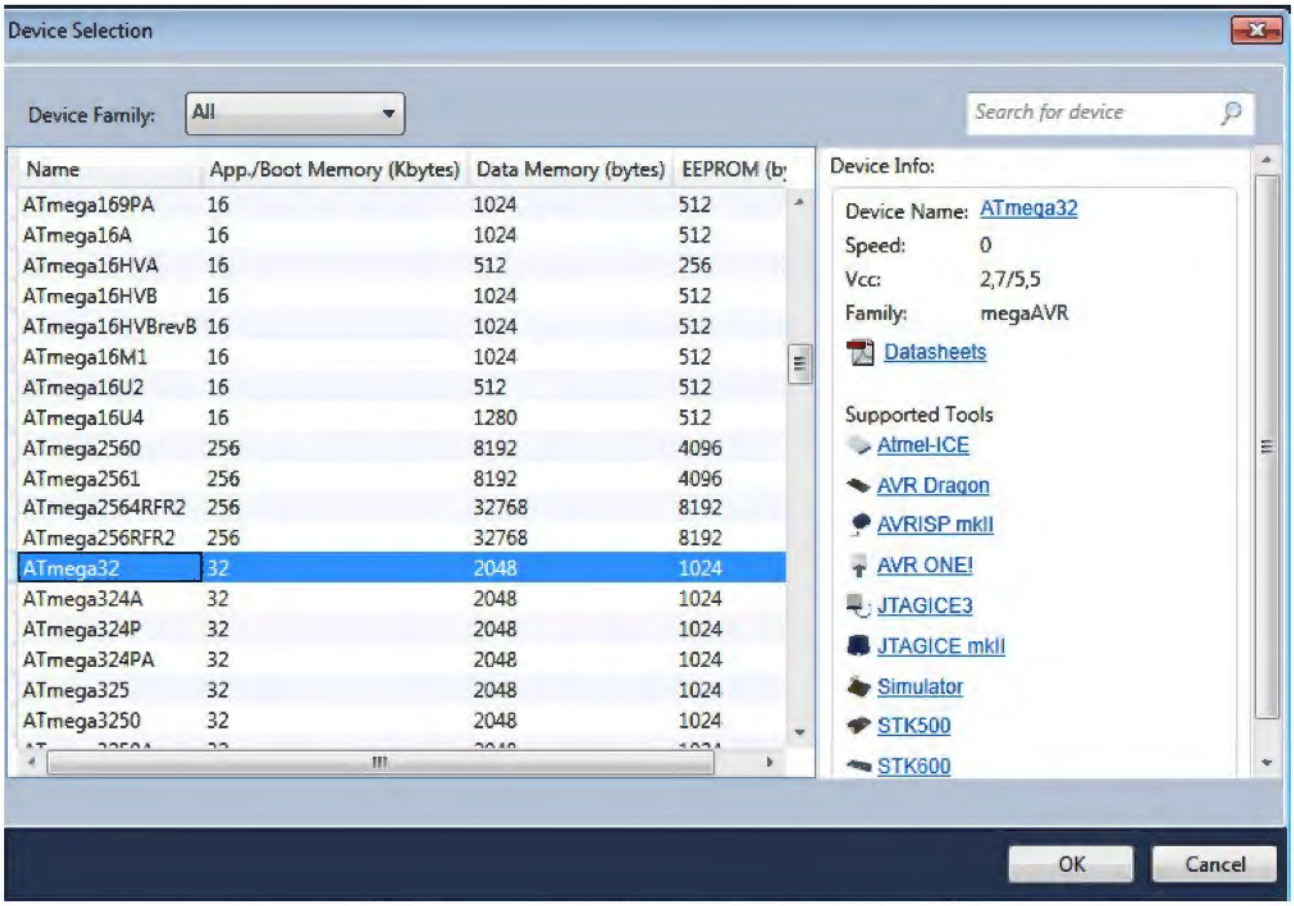


Figura 6. Alegerea tipului de controler utilizat într-un proiect

- înainte de a scrie programul propriu-zis, se introduce următoarea linie:
  .include „m32def.inc”
  Această linie este o directivă de compilare care indică numele fişierului ce conţine descrierea regiştrilor şi caracteristicilor microcontrolerului.
- după scrierea programului, se aleg opţiunile „Save All” din meniul „File” şi apoi „Build Solution” din meniul „Build”. Orice erori de sintaxă vor fi semnalate de compilator. Acestea trebuie corectate şi apoi se vor repeta paşii anteriori.
- pentru a scrie un nou program, din meniul „File” se va alege opţiunea „New Project” şi apoi se vor repeta paşii prezentaţi mai sus.

3.3. Programarea microcontrolerlor folosind software-ul „AVRFLASH”

AVRFLASH este un program oferit împreună cu placa de dezvoltare EasyAVRv7 care permite programarea unei game largi de microcontrolere utilizând această placă conectată la calculator prin intermediul interfeţei USB. Înainte de orice operaţie se va specifica prin intermediul opţiunii „Device” tipul de microcontroler programat.

Pentru orice microcontroler trebuie programate două elemente distincte: biţii de configurare şi programul propriu-zis.

Aşa cum se poate vedea în Figura 7, biţii de configurare sunt disponibili în zona centrală a ferestrei programului AVRFLASH. Valorile acestor biţi se stabilesc pe baza datelor de catalog. În cadrul laboratorului nu se vor modifica valorile acestor biţi.

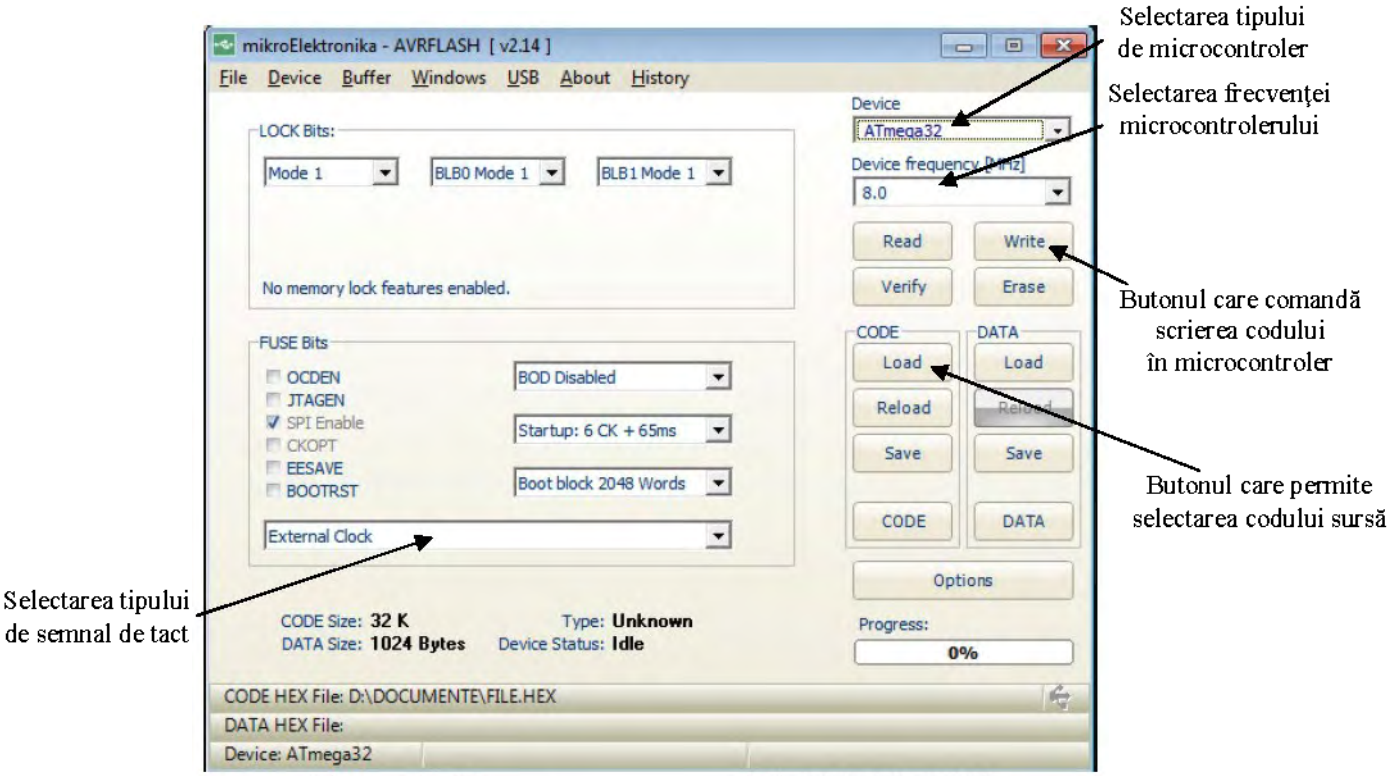


Figura 7. Fereastra programului AVRFLASH

Scrierea programului propriu-zis presupune următorii paşi:

- se asigură conectarea plăcii de dezvoltare la calculator – ledul Link trebuie să fie aprins;
- la opţiunea „Device" se alege „ATmega32" iar la opţiunea „Device frequency [MHz]" se alege valoarea 8;
- se selectează opţiunea „External Clock" pentru semnalul de tact al microcontrolerului;
- se deschide fişierul care conţine codul: se apasă butonul „Load" din cadrul grupului de butoane numit „Flash" şi se identifică fişierul „.hex" corespunzător. Acest tip de fişier este generat automat de AVRStudio după compilarea cu succes, fără erori, a unui program.
- se apasă butonul „Write" şi se aşteaptă finalizarea operaţiei urmărind indicatorul „Progress" din partea inferioară a programului.

3.4. Placa de dezvoltare EasyAVRv7

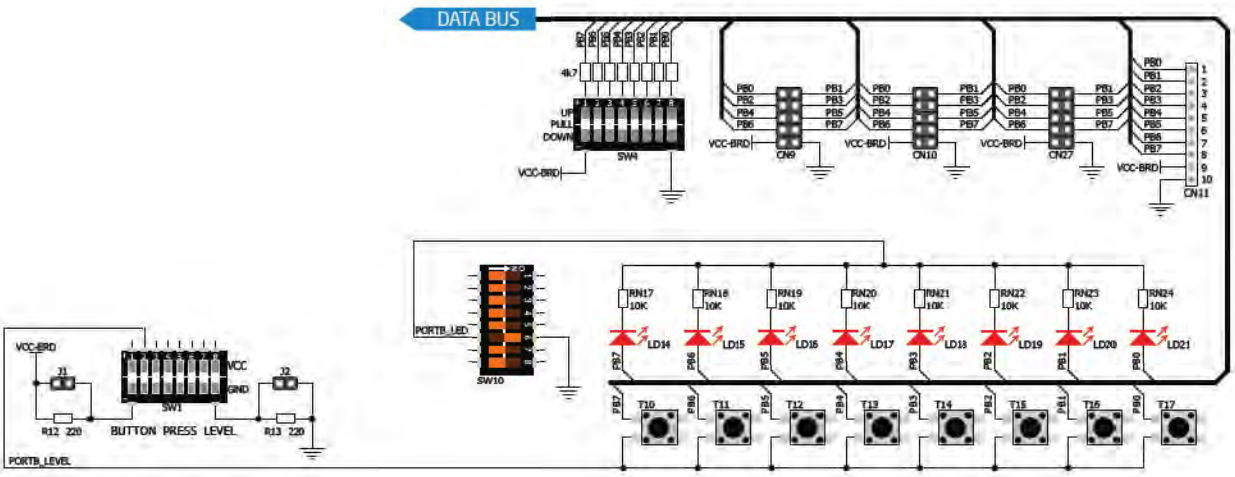
EasyAVRv7 este o placă de dezvoltare produsă de firma Mikroelectronika şi permite realizarea de aplicaţii cu microcontrolere Atmel.

Placa dispune de următoarele resurse hardware principale:

- programator cu interfaţă USB;
- bloc de alimentare a plăcii cu posibilitatea de a utiliza o sursă de tensiune externă sau tensiunea furnizată de interfaţa USB;
- socluri pentru conectarea microcontrolerelor Atmel în capsule DIP40, DIP28, DIP18, DIP20, DIP14 sau DIP8;
- SW pushbuton conectate la toate porturile microcontrolerului;
- LED-uri conectate la toate porturile microcontrolerului;
- rezistenţe externe de pull-up sau pull-down pentru fiecare pin al fiecărui port. Prin intermediul unor micro-comutatoare aceste rezistenţe pot fi conectate sau deconectate de pe pini, pentru fiecare pin individual.
- generator de tensiune pentru diferite canale ale modulului ADC din microcontroler
- generator de semnal de tact cu frecvenţa de 8MHz;
- modul de afişare cu LED-uri tip 7 segmente;
- interfaţă serială RS232 cu modul harware de conversie de la RS232 la USB;
- senzor de temperatură DS1820 cu interfaţă One-wire;
- modul de afişare LCD 2x16 caractere.

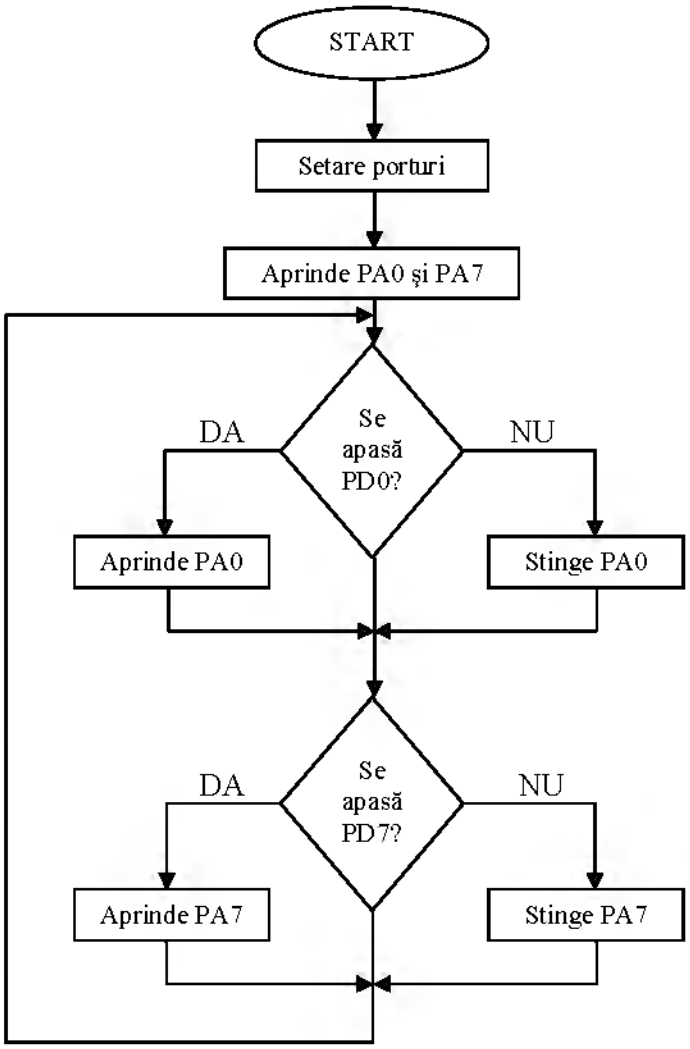
Sunt disponibile pentru interfaţarea cu alte blocuri suplimentare toate porturile microcontrolerului cu ajutorul unor conectori.

Schema blocului de LED-uri şi push-butoane este prezentată în figura de mai jos. Se observă că pentru fiecare port se pot dezactiva LED-urile corespunzătoare iar pentru aprinderea unui LED este nevoie ca pinul respectiv să aibă valoarea 1 logic. De asemenea se poate selecta, pentru fiecare port, nivelul de tensiune la care este conectat pinul în momentul când se apasă pe push-buton cu următoarele opţiuni: masă (GND), +5V sau liber/deconectat.



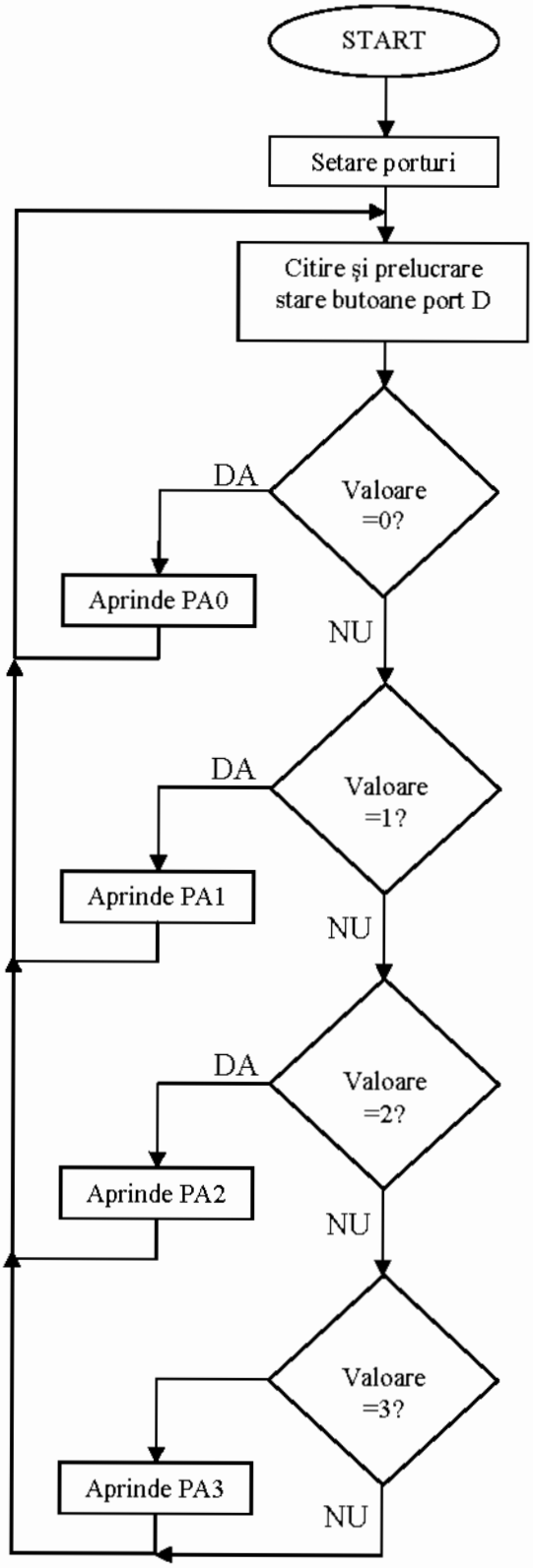
4. Exemple de programe

a) Să se scrie un program care menţine aprinse doar led-urile PA0 şi PA7 şi le stinge la fiecare apăsare a pushbuton-ului corespunzător de pe portul D. Se va asigura că apăsarea butonului determină conectarea pinului la potenţialul GND.



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b00000000
out DDRD,r16   ; portul D este setat ca intrare
ldi r16,0b11111111
out PORTD,r16 ; se activeaza rezistentele pull-up pentru toti pinii portului D
ldi r16,0b11111111
out DDRA,r16   ; portul A, unde sunt conectate led-urile, este setat ca iesire
main:
ldi r17,0b00000000
in r16,PIND       ; se citesc pushbutoanele
sbrc r16,PD0      ; s-a apasat butonul corespunzator led-ului PA0 ?
ldi r17,0b00000001        ; nu s-a apasat deci led-ul PA0 trebuie sa fie aprins
out PORTA,r17
ldi r17,0b00000000
in r16,PIND
sbrc r16,PD7      ; s-a apasat butonul corespunzator led-ului PA7 ?
ldi r17,0b10000000        ; nu s-a apasat deci led-ul PA7 trebuie sa fie aprins
out PORTA,r17
jmp main
```

b) Să se scrie un program care aprinde unul din ledurile portului A corespunzător valorii binare creată de primele 2 butoane PD0 – PD1 ai portului D, considerând PD0 LSB. Ledul 0 se consideră PA0. Se va asigura că apăsarea butonului determină conectarea pinului la potenţialul GND.

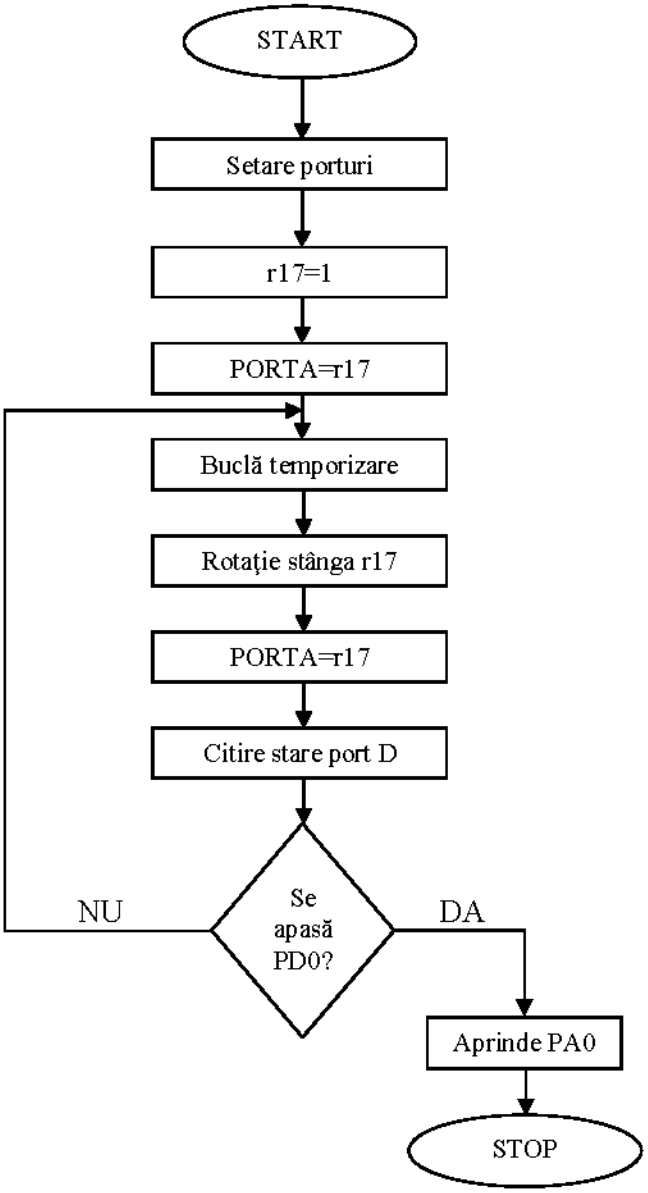



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
```

```
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b00000000
out DDRD,r16   ; portul D este setat ca intrare
ldi r16,0b11111111
out PORTD,r16 ; se activeaza rezistentele pull-up pentru toti pinii portului D
ldi r16,0b11111111
out DDRA,r16; portul A, unde sunt conectate led-urile, este setat ca iesire
main:
in r16,PIND       ; se citesc pushbutoanele
andi r16,0b00000011     ;se izoleaza doar valorile de pe primele 2 butoane
ldi r17,0x00
cp r16,r17          ;configuratia butoanelor corespunde valorii 0 ?
brne et1            ;nu, se merge mai departe
ldi r18,0b00000001       ;da, se va aprinde led-ul PA0
out PORTA,r18
jmp main
et1:
ldi r17,0x01
cp r16,r17
brne et2
ldi r18,0b00000010
out PORTA,r18
jmp main
et2:
ldi r17,0x02
cp r16,r17
brne et3
ldi r18,0b00000100
out PORTA,r18
et3:
ldi r17,0x03
cp r16,r17
brne et4
```

```
ldi r18,0b00001000
out PORTA,r18
et4:
jmp main
```

c) Să se scrie un program care deplasează în mod automat ledul aprins de pe portul A cu o poziţie către stânga şi se opreşte la apăsarea butonului PD0. Ledul 0 se consideră PA0. Se va asigura că apăsarea butonului determină conectarea pinului la potenţialul GND.




```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
```

```
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b00000000
out DDRD,r16
ldi r16,0b11111111
out PORTD,r16
ldi r16,0b11111111
out DDRA,r16
ldi r17,0b00000001
out PORTA,r17       ;initial se aprinde ledul PA0
clc                 ;idicatorul carry se face 0
main:
ldi r18,0xFF        ;registrii r18,r19 si r20 sunt utilizati pentru a crea o temporizare
ldi r19,0xFF
ldi r20,0x13
bucla:
dec r18
brne bucla
ldi r18,0xFF
dec r19
brne bucla
ldi r19,0xFF
dec r20
brne bucla
rol r17             ;dupa terminarea temporizarii se face o rotatie catre stanga prin carry
out PORTA,r17       ;se scrie noua valoare la leduri
in r16,PIND
sbrc r16,PD0        ;am apasat butonul PD0 ?
jmp main            ;nu, se cotinua deplasarea ledului aprins
ldi r17,0b00000001  ;da, aprind ledul PA0 si ma opresc
out PORTA,r17
end:
jmp end
```

5. Teme şi exerciţii

a) Să se scrie un program care aprinde toate LED-urile portului B, mai puţin cel corespunzător numărului butonului apăsat pe portul A (se va alege acest buton).
Observaţie: butoanele şi LED-urile se numerotează corespunzător valorii pinului pe care sunt conectate.

b) Să se scrie un program care adună două valori de 8 biţi şi aprinde ledurile portului D astfel încât acestea să reflecte reprezentarea binară a rezultatului.
Observaţie: butoanele şi LED-urile se numerotează corespunzător valorii pinului pe care sunt conectate.

c) Să se scrie un program care deplasează în mod automat ledul aprins al portului B cu o poziţie către dreapta şi se opreşte la apăsarea butonului PC3; în acest moment se va aprinde ledul 3 considerând că ledul 0 este PB0.

Lucrarea 2
Microcontrolerul ATmega32. Utilizarea modulelor timer

Scopul lucrării

a) Prezentarea modulului Timer/Counter0 pe 8 biţi
b) Prezentarea modulului timer pe 16 biţi
c) Programarea unor aplicaţii utilizând placa de dezvoltare EasyAVRv7

1. Noţiuni generale privind sistemul de întreruperi

Microcontrolerul ATmega32 dispune de mai multe tipuri de module hardware care pot funcţiona independent de unitatea centrală utilizând sistemul de întreruperi. Astfel, după ce un modul termină activitatea pe care trebuie să o realizeze, generează o întrerupere pentru a avertiza procesorul că este pregătit să execute o nouă sarcină. Ca exemple de module hardware se pot menţiona:

- două numărătoare (timere) de 8 biţi şi un numărător (timer) de 16 biţi
- modul de comunicaţie serială pe două fire TWI
- modul de comunicaţie serială USART
- convertor analog – digital
- comparator analogic.

Tabela de întreruperi se plasează de obicei la începutul zonei de memorie flash. **În cazul microcontrolerului ATmega32, fiecare linie din tabela de întreruperi ocupă două locaţii de memorie**. Această structură permite utilizarea instrucţiunii *jmp* pentru saltul către rutina de tratare a întreruperii. Instrucţiunea *jmp*, spre deosebire de *rjmp*, **ocupă un număr dublu de biţi (32)** şi oferă posibilitatea de a accesa orice zonă din cei 32KB de memorie flash ai microcontrolerului ATmega32. Cea mai uzuală amplasare a întreruperilor este descrisă în tabelul de mai jos.

Nr. crt.	Adresa în memoria Flash	Descriere
1	0x00	Generată la alimentare sau la un semnal pe pinul RESET
2	0x02	Întrerupere externă 0
3	0x04	Întrerupere externă 1
4	0x06	Întrerupere externă 2
5	0x08	Generată când Timer/Counter2 atinge valoarea de prag
6	0x0A	Generată când Timer/Counter2 atinge valoarea maximă
7	0x0C	Generată de unitatea de captură a timerului pe 16 biţi
8	0x0E	Generată când timerul pe 16 biţi atinge valoarea de prag A

9	0x10	Generată când timerul pe 16 biţi atinge valoarea de prag B
10	0x12	Generată când timerul pe 16 biţi atinge valoarea maximă
11	0x14	Generată când Timer/Counter0 atinge valoarea de prag
12	0x16	Generată când Timer/Counter0 atinge valoarea maximă
13	0x18	Generată de unitatea SPI
14	0x1A	Generată la recepţia completă a unor date de către modulul USART
15	0x1C	Generată când registrul de date al modulului USART este gol
16	0x1E	Generată la transmisia completă a unor date de către modulul USART
17	0x20	Generată de modulul ADC
18	0x22	Generată de modulul EEPROM
19	0x24	Generată de comparatorul analogic
20	0x26	Generată de modulul TWI
21	0x28	Generată de modulul de auto-scriere a memoriei Flash

Pentru orice sursă de întrerupere, există un bit specific de activare/dezactivare. În afară de biţii specifici, toate întreruperile sunt controlate de bitul I din registrul de stare. Astfel, dacă acest bit este 0, nu se va executa nici o întrerupere, indiferent de setarea biţilor individuali. Bitul I se poate face 0 cu instrucţiunea *cli* şi 1 cu instrucţiunea *sei*.

În rutina de tratare a întreruperii, utilizatorul trebuie să salveze de la început registrul de stare SREG şi apoi să îl restaureze la sfârşitul rutinei. De asemenea, ieşirea din rutina de tratare a întreruperii se face cu instrucţiunea *reti*.

Un exemplu tipic de rutină pentru întreruperi este prezentat mai jos:

```
intrerupere:
in r20, SREG   ; salvarea registrului de stare in r20
.......
.......
.......
out SREG, r20  ; restaurarea registrului de stare din r20
reti
```

2. Modulul Timer/Counter0 pe 8 biţi

Modulul Timer/Counter0 pe 8 biţi poate fi utilizat pentru realizarea de temporizări, numărarea evenimentelor externe sau generarea de forme de undă. În Figura 1 este prezentată schema bloc a modulului timer. Dintre cele mai importante caracteristici se pot menţiona:

- un prag independent de comparaţie
- poate număra impulsuri externe
- posibilitate de auto-iniţializare
- posibilitate de generare facilă a semnalelor Pulse Width Modulation (PWM)

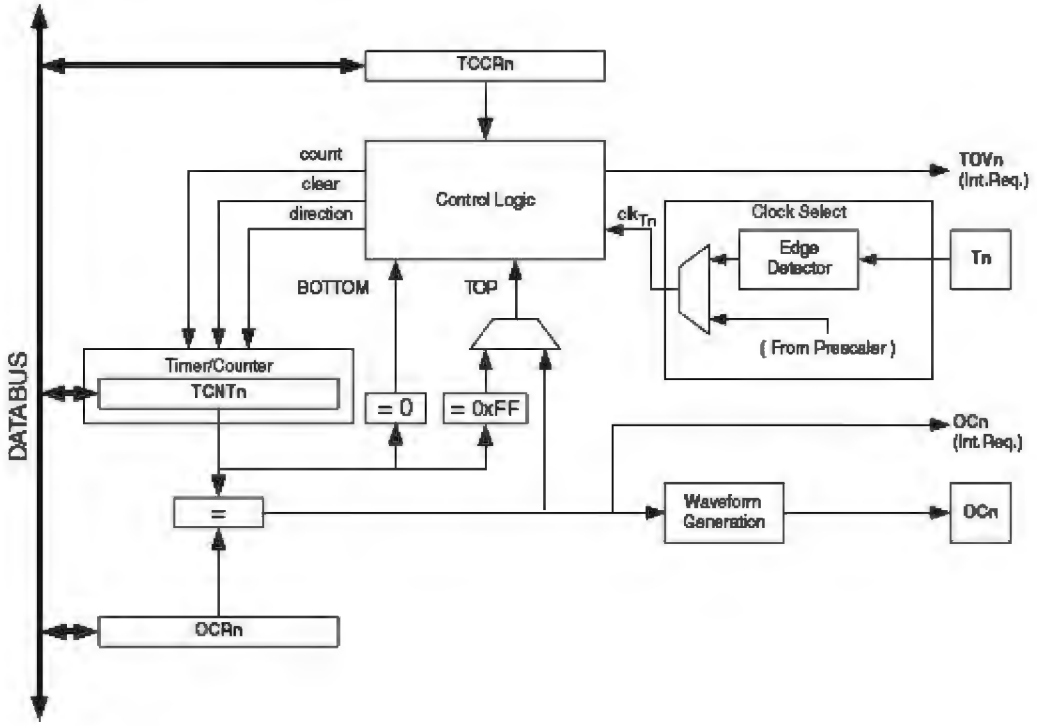


Figura 1. Schema bloc a modulului Timer/Counter0 pe 8 biţi

Regiştrii alocaţi acestui timer pot fi împărţiţi în trei mari categorii:
- registrul de numărare: TCNT0
- pragul de comparaţie: OCR0
- comandă şi control: TCCR0, TIMSK şi TIFR

Conţinutul regiştrilor de comandă şi control este prezentat în Figura 2.

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
7	6	5	4	3	2	1	0	
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
7	6	5	4	3	2	1	0	
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

Figura 2. Conţinutul regiştrilor de comandă şi control ai Timer/Counter0

Modurile principale de funcţionare ale Timer/Counter0 sunt:
- funcţionare normală
- re-iniţializare la atingerea valorii de prag (CTC)
- semnal PWM rapid
- semnal PWM corect în fază

Biţii care selectează modul de funcţionare se regăsesc în registrul TCCR0. Rezultatul posibilelor combinaţii ale acestora este dat în Figura 3.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Figura 3. Setarea modului de funcţionare al Timer/Counter0

Toate modurile de funcţionare se bazează pe incrementarea sau decrementarea automată a registrului de numărare TCNT0. Frecvenţa de incrementare/decrementare este controlată de combinaţia biţilor CS02, CS01, CS00 din registrul TCCR0 conform tabelului de mai jos (cu f_{osc} s-a notat frecvenţa ceasului microcontrolerului).

CS02	CS01	CS00	Frecvenţa de inc/dec
0	0	0	timer oprit
0	0	1	$f_{osc}/1$
0	1	0	$f_{osc}/8$
0	1	1	$f_{osc}/64$
1	0	0	$f_{osc}/256$
1	0	1	$f_{osc}/1024$
1	1	0	frecvenţa semnalului de pe pinul T0, frontul descendent
1	1	1	frecvenţa semnalului de pe pinul T0, frontul ascendent

Unitatea timer compară **în permanenţă** valoarea TCNT0 cu OCR0. În modurile de funcţionare normală şi re-iniţializare la atingerea valorii de prag, valoarea 1 a bitul OCIE0 din registrul TIMSK determină generarea unei întreruperi când TCNT0 atinge valoarea registrului de prag OCR0.

În vederea utilizării timerului Timer/Counter0, programatorul trebuie să prevadă o secvenţă iniţială de cod pentru setarea parametrilor de funcţionare. Această secvenţă trebuie să cuprindă cel puţin următorii paşi:

- dezactivarea globală a întreruperilor (folosind *cli*);
- setarea porturilor ce urmează a fi folosite, dacă este cazul;
- alegerea modului de funcţionare dorit prin modificarea biţilor corespunzători din registrul de control. Se va avea în vedere să nu se pornească timerul.
- iniţializarea cu o valoarea a registrului de numărare şi/sau a pragului, dacă este cazul;
- activarea întreruperilor dorite prin modificarea biţilor corespunzători din registrul TIMSK. Se va avea în vedere să nu se modifice biţii corespunzători altui modul timer.
- pornirea timerului la frecvenţa dorită, având în vedere să nu se modifice valoarea altor biţi din registrul de control;

- activarea globală a întreruperilor folosind *sei*, dacă este cazul.

2.1. Modul de funcţionare normală

În acest mod valoarea TCNT0 este incrementată continuu la fiecare tact de numărare. Când se atinge valoarea maximă reprezentabilă pe 8 biţi, TCNT0 ia valoarea 0 şi apoi continuă să numere.

Dacă bitul TOIE0 din registrul TIMSK are valoarea 1, se va genera o întrerupere când se atinge valoarea maximă reprezentabilă pe 8 biţi.

2.2. Modul de funcţionare cu re-iniţializare la atingerea valorii de prag (CTC)

În acest mod valoarea TCNT0 este incrementată la fiecare tact de numărare până când devine egală cu OCR0. În acest moment, TCNT0 se re-iniţializează cu 0 şi numărătoarea reîncepe.

În funcţie de valorile biţilor COM01 şi COM00 din registrul TCCR0 (Figura 4), este posibil să se programeze modificarea nivelului logic al pinului OC0 când TCNT0 atinge valoarea registrului de prag OCR0. În acest caz pinul **trebuie setat ca ieşire**.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Figura 4. Efectul combinaţiilor posibile ale biţilor COM01 şi COM00

2.3. Modul de funcţionare semnal PWM rapid

În acest mod de funcţionare, TCNT0 este incrementat până atinge valoarea 0xFF. În acest moment se iniţializează TCNT0 cu 0 şi se reia numărătoarea.

Valoarea logică a pinului OC0 se modifică numai când TCNT0 este egal cu o valoare de prag şi când ajunge la valoarea 0, aşa cum se poate vedea în Figura 5.

În acest mod de funcţionare utilizatorul poate modifica oricând valoarea de prag deoarece ea nu va fi transferată în modulul timer decât după ce TCNT0 a realizat un ciclu complet de numărare.

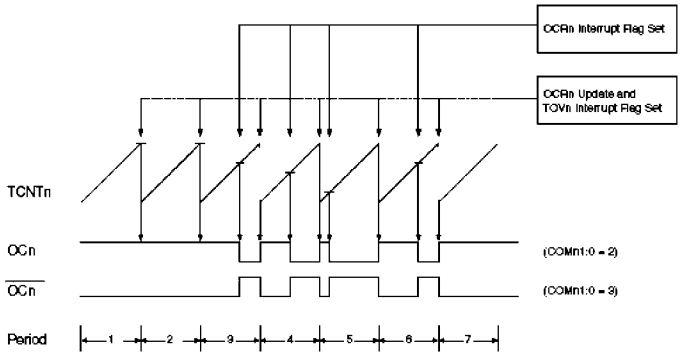


Figura 5. Modul de funcţionare semnal PWM rapid

2.4. *Modul de funcţionare semnal PWM corect în fază*

În acest mod de funcţionare TCNT0 este incrementat până atinge valoarea 0xFF. În acest moment TCNT0 începe să fie decrementat până ajunge la 0 după care se reia ciclul.

Valoarea logică a pinului OC0 se modifică numai când TCNT0 este egal cu valoarea de prag în timp ce creşte şi apoi în timp ce descreşte, aşa cum se poate vedea în Figura 6.

În acest mod de funcţionare utilizatorul poate modifica oricând valoarea de prag deoarece ea nu va fi transferată în modulul timer decât după ce TCNT0 a realizat un ciclu complet de numărare.

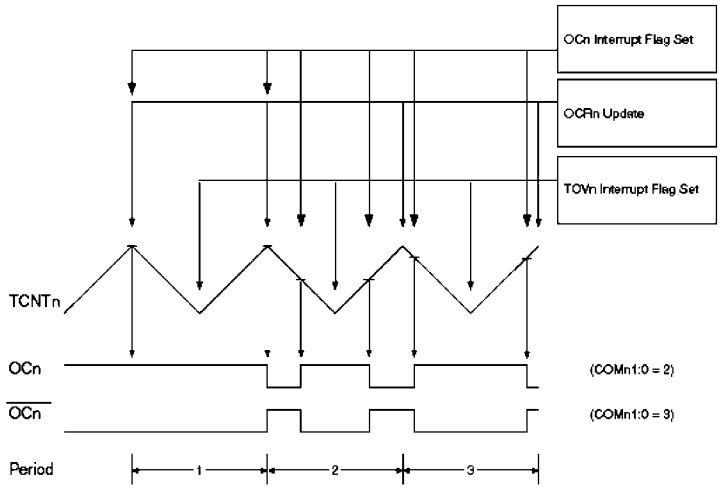


Figura 6. Modul de funcţionare semnal PWM corect în fază

3. Modulul timer pe 16 biţi

Modulul timer pe 16 biţi poate fi utilizat pentru realizarea de temporizări, numărarea evenimentelor externe sau generarea de forme de undă. În Figura 7 este prezentată schema bloc a modulului timer. Dintre cele mai importante caracteristici se pot menţiona:

- lucrează cu registre de 16 biţi
- două praguri independente de comparaţie
- unitate de captură evenimente
- posibilitate de auto-iniţializare
- posibilitate de generare facilă a semnalelor Pulse Width Modulation (PWM)

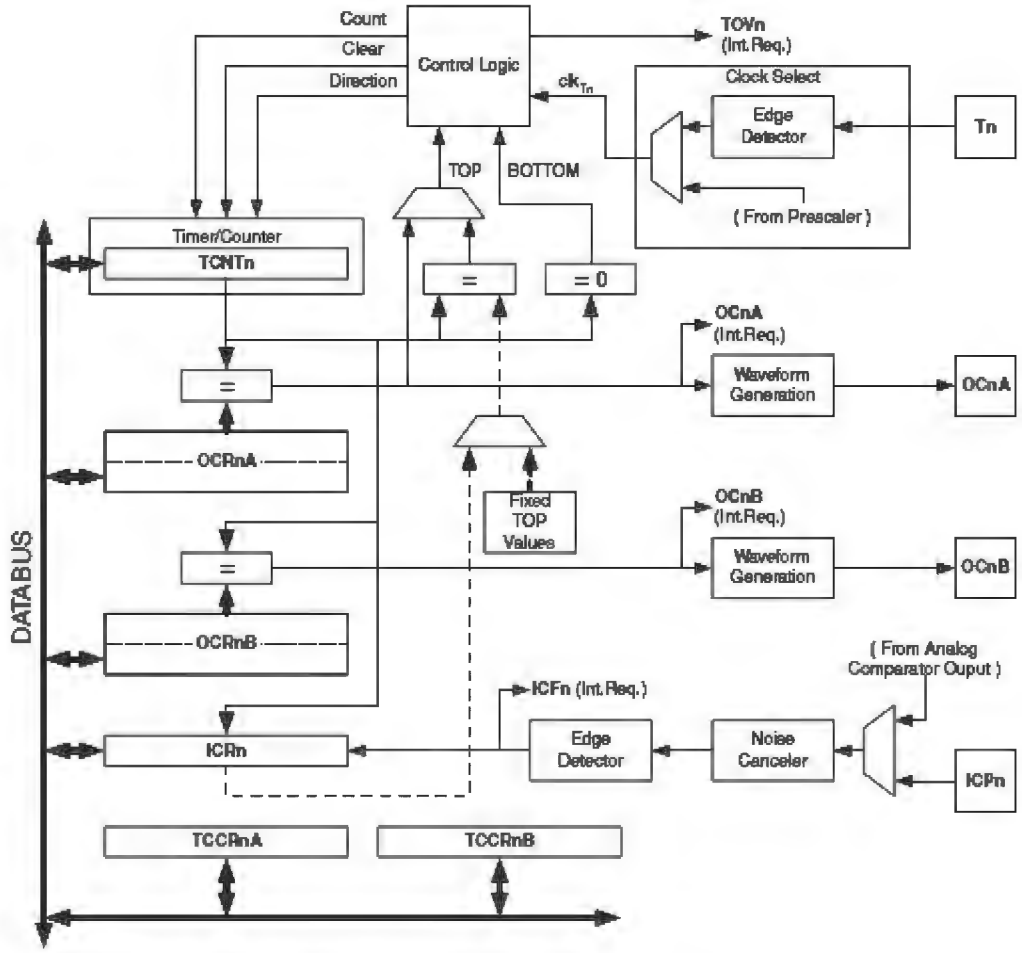


Figura 7. Schema bloc a modulului timer pe 16 biţi

Regiştrii alocaţi acestui timer pot fi împărţiţi în trei mari categorii:
- registrul de numărare: TCNT1H şi TCNT1L
- praguri de comparaţie: OCR1AL şi OCR1AH, OCR1BL şi OCR1BH, ICR1H şi ICR1L
- comandă şi control: TCCR1A, TCCR1B, TIMSK şi TIFR

Conţinutul regiştrilor de comandă şi control este prezentat în Figura 8.

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
7	6	5	4	3	2	1	0	
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
7	6	5	4	3	2	1	0	
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR

Figura 8. Conţinutul regiştrilor de comandă şi control ai timerului pe 16 biţi

Toţi regiştrii utilizaţi ca praguri de comparaţie şi registrul de numărare sunt de 16 biţi dar utilizatorul nu are acces direct la octeţii superiori şi inferiori ai acestora. În schimb, se utilizează un registru temporar de 8 biţi invizibil pentru programator şi o secvenţă specială de scriere şi citire.

Astfel, **pentru a scrie** o valoare în regiştrii de tip prag sau în cel de numărare, se va scrie **mai întâi partea superioară** (de exemplu OCR1AH) şi apoi partea inferioară (de exemplu OCR1AL). Când **se citeşte** o valoare din regiştrii de tip prag ai timerului sau din cel de numărare, se va citi **mai întâi partea inferioară** şi apoi partea superioară.

Modurile principale de funcţionare ale timerului de 16 biţi sunt:

- funcţionare normală
- re-iniţializare la atingerea valorii de prag (CTC)
- semnal PWM rapid
- semnal PWM corect în fază
- semnal PWM corect în fază şi frecvenţă

Biţii care selectează modul de funcţionare se regăsesc în regiştrii TCCR1A şi TCCR1B. Rezultatul posibilelor combinaţii ale acestora este dat în Figura 9.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Figura 9. Setarea modului de funcţionare al timerului de 16 biţi

Toate modurile de funcţionare se bazează pe incrementarea sau decrementarea automată a registrului de numărare TCNT1. Frecvenţa de incrementare/decrementare este controlată de combinaţia biţilor CS12, CS11, CS10 din registrul TCCR1B conform tabelului de mai jos (cu f_{osc} s-a notat frecvenţa ceasului microcontrolerului).

CS12	CS11	CS10	Frecvenţa de inc/dec
0	0	0	timer oprit
0	0	1	$f_{osc}/1$
0	1	0	$f_{osc}/8$
0	1	1	$f_{osc}/64$

1	0	0	$f_{osc}/256$
1	0	1	$f_{osc}/1024$
1	1	0	frecvenţa semnalului de pe pinul T1, frontul descendent
1	1	1	frecvenţa semnalului de pe pinul T1, frontul ascendent

În modurile de funcţionare normală şi re-iniţializare la atingerea valorii de prag se poate utiliza unitatea de captură evenimente. Aceasta este controlată de tranziţiile semnalului de pe pinul ICP1 sau cele ale ieşirii comparatorului analogic. Când se produce o anumită tranziţie selectată de utilizator, unitatea de captură copiază registrul TCNT1 (16 biţi) în registrul ICR1, obţinându-se astfel o amprentă temporală a producerii evenimentului.

Dacă bitul TICIE1 din registrul TIMSK are valoarea 1, se va genera o întrerupere când are loc copierea amprentei temporale.

Unitatea timer compară **în permanenţă** valoarea TCNT1 cu OCR1A şi OCR1B. În modurile de funcţionare normală şi re-iniţializare la atingerea valorii de prag, valoarea 1 a bitul OCIE1A sau OCIE1B din registrul TIMSK determină generarea unei întreruperi când TCNT1 atinge valoarea registrului de prag OCR1A respectiv OCR1B.

În vederea utilizării timerului pe 16 biţi, programatorul trebuie să prevadă o secvenţă iniţială de cod pentru setarea parametrilor de funcţionare. Această secvenţă trebuie să cuprindă cel puţin următorii paşi:

- dezactivarea globală a întreruperilor (folosind *cli*);
- setarea porturilor ce urmează a fi folosite, dacă este cazul;
- alegerea modului de funcţionare dorit prin modificarea biţilor corespunzători din regiştrii de control. Se va avea în vedere să nu se pornească timerul.
- iniţializarea cu o valoarea a registrului de numărare şi/sau a pragurilor, dacă este cazul;
- activarea întreruperilor dorite prin modificarea biţilor corespunzători din registrul TIMSK. Se va avea în vedere să nu se modifice biţii corespunzători altui modul timer.
- pornirea timerului la frecvenţa dorită, având în vedere să nu se modifice valoarea altor biţi din registrul de control;
- activarea globală a întreruperile folosind *sei*, dacă este cazul.

3.1. Modul de funcţionare normală

În acest mod valoarea TCNT1 este incrementată continuu la fiecare tact de numărare. Când se atinge valoarea maximă reprezentabilă pe 16 biţi, TCNT1 ia valoarea 0 şi apoi continuă să numere.

Dacă bitul TOIE1 din registrul TIMSK are valoarea 1, se va genera o întrerupere când se atinge valoarea maximă reprezentabilă pe 16 biţi.

3.2. Modul de funcţionare cu re-iniţializare la atingerea valorii de prag (CTC)

În acest mod valoarea TCNT1 este incrementată la fiecare tact de numărare până când devine egală cu OCR1A sau ICR1, în funcţie de setările făcute. În acest moment, TCNT1 se re-iniţializează cu 0 şi numărătoarea reîncepe.

În funcţie de valorile biţilor COM1A1/COM1B1 şi COM1A0/COM1B0 din registrul TCCRA, este posibil să se programeze modificarea nivelului logic al pinului OC1A când TCNT1 atinge valoarea registrului de prag OCRA1/OCRB1. În acest caz pinul respectiv **trebuie setat ca ieşire**.

3.3. Modul de funcţionare semnal PWM rapid

În acest mod de funcţionare, TCNT1 este incrementat, în funcţie de setări, până atinge una din valorile 0x00FF, 0x01FF, 0x03FF, valoarea din ICR1 sau cea din OCR1A. În acest moment se iniţializează TCNT1 cu 0 şi se reia numărătoarea.

Valoarea logică a pinilor OC1A sau OC1B se modifică numai când TCNT1 este egal cu o valoare de prag şi când ajunge la valoarea maximă, aşa cum se poate vedea în Figura 10.

În acest mod de funcţionare, când TCNT1 este setat să numere până la valoarea registrului OCR1A, utilizatorul o poate modifica oricând deoarece ea nu va fi transferată în modulul timer decât după ce TCNT1 a realizat un ciclu complet de numărare.

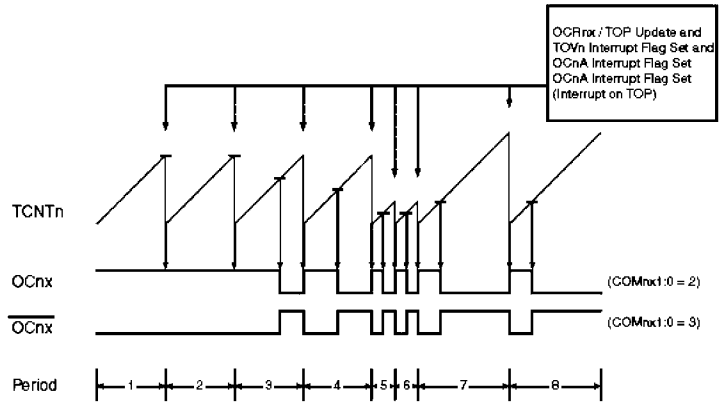


Figura 10. Modul de funcţionare semnal PWM rapid

3.4. Modul de funcţionare semnal PWM corect în fază

În acest mod de funcţionare, TCNT1 este incrementat, în funcţie de setări, până atinge una din valorile 0x00FF, 0x01FF, 0x03FF, valoarea din ICR1 sau cea din OCR1A. În acest moment TCNT1 începe să fie decrementat până ajunge la 0 după care se reia ciclul.

Valoarea logică a pinilor OC1A sau OC1B se modifică numai când TCNT1 este egal cu o valoare de prag în timp ce creşte şi apoi în timp ce descreşte, aşa cum se poate vedea în Figura 11.

În acest mod de funcţionare, când TCNT1 este setat să numere până la valoarea registrului OCR1A, utilizatorul o poate modifica oricând deoarece ea nu va fi transferată în modulul timer decât după ce TCNT1 a ajuns la valoarea maximă.

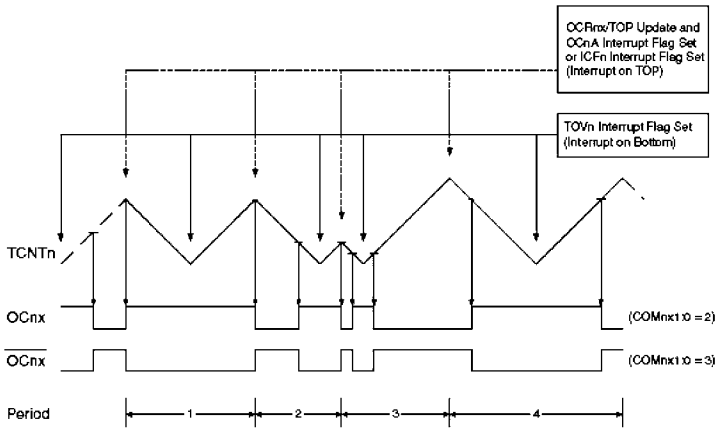


Figura 11. Modul de funcţionare semnal PWM corect în fază

3.5. *Modul de funcţionare semnal PWM corect în fază şi frecvenţă*

În acest mod de funcţionare, TCNT1 este incrementat, în funcţie de setări, până atinge valoarea din ICR1 sau cea din OCR1A. În acest moment TCNT1 începe să fie decrementat până ajunge la 0 după care se reia ciclul.

Valoarea logică a pinilor OC1A sau OC1B se modifică numai când TCNT1 este egal cu o valoare de prag în timp ce creşte şi apoi în timp ce descreşte, aşa cum se poate vedea în Figura 12.

În acest mod de funcţionare, când TCNT1 este setat să numere până la valoarea registrului OCR1A, utilizatorul o poate modifica oricând deoarece ea nu va fi transferată în modulul timer decât după ce TCNT1 a terminat un ciclu.

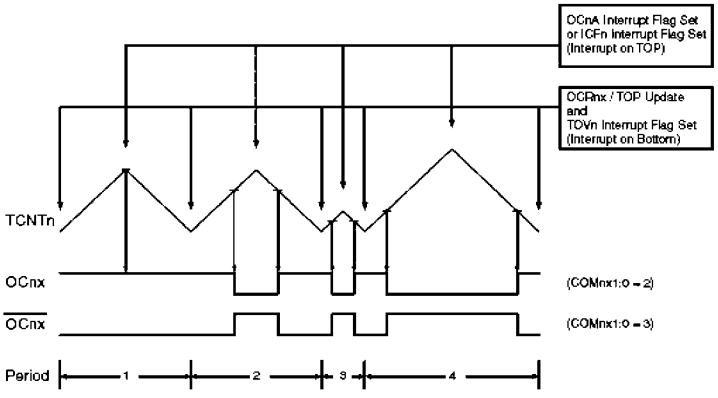
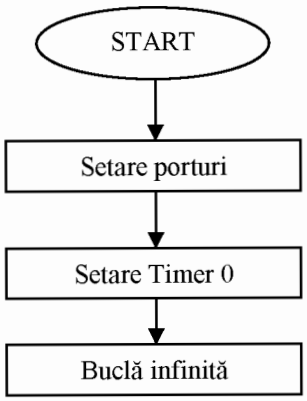


Figura 12. Modul de funcţionare semnal PWM corect în fază şi frecvenţă

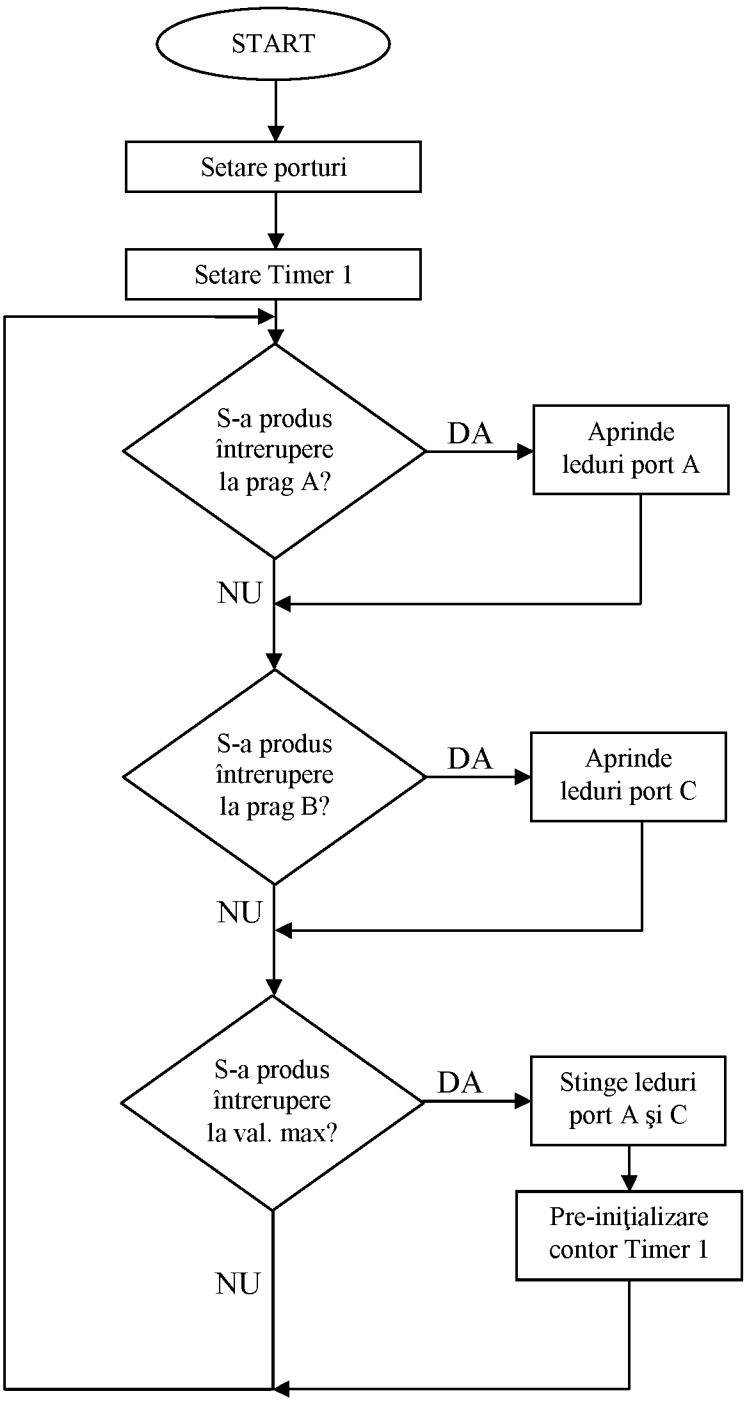
4. Exemple de programe

a) Să se scrie un program care generează pe pinul OC0 (PB3) un semnal dreptunghiular cu frecvenţa de aproximativ 15 Hz şi factor de umplere 50%.




```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0xFF
out DDRB,r16 ; OC0 este pe pinul PB3
main:
cli                 ;dezactivare intreruperi
ldi r16,0b00011000  ;setare timer: se utilizeaza pinul OC0, timerul este oprit deocamdata
out TCCR0,r16       ;modul va fi CTC cu prag dat de OCR0
in r16, TIMSK
andi r16,0b11111100 ;nu se utiliz. nici o intrer., fara a modifica alti biti din TIMSK
out TIMSK, r16
ldi r16, 0xFA       ;se incarca valoarea de prag: 0xFA=250
out OCR0,r16        ;250 * 1/(8MHz/1024) =~ 32ms
in r16,TCCR0
andi r16,0b11111101 ;se porneste timerul si este setat sa numere
ori r16,0b00000101  ;la fiecare 1024 perioade de ceas, fara a modifica alti biti
out TCCR0,r16
bucla:
rjmp bucla
```

b) Să se scrie un program care într-un interval de aproximativ 8s efectuează următoarele operaţii: după 2s aprinde ledurile portului A, după 6,5s aprinde ledurile portului C iar la finalul intervalului stinge toate ledurile.

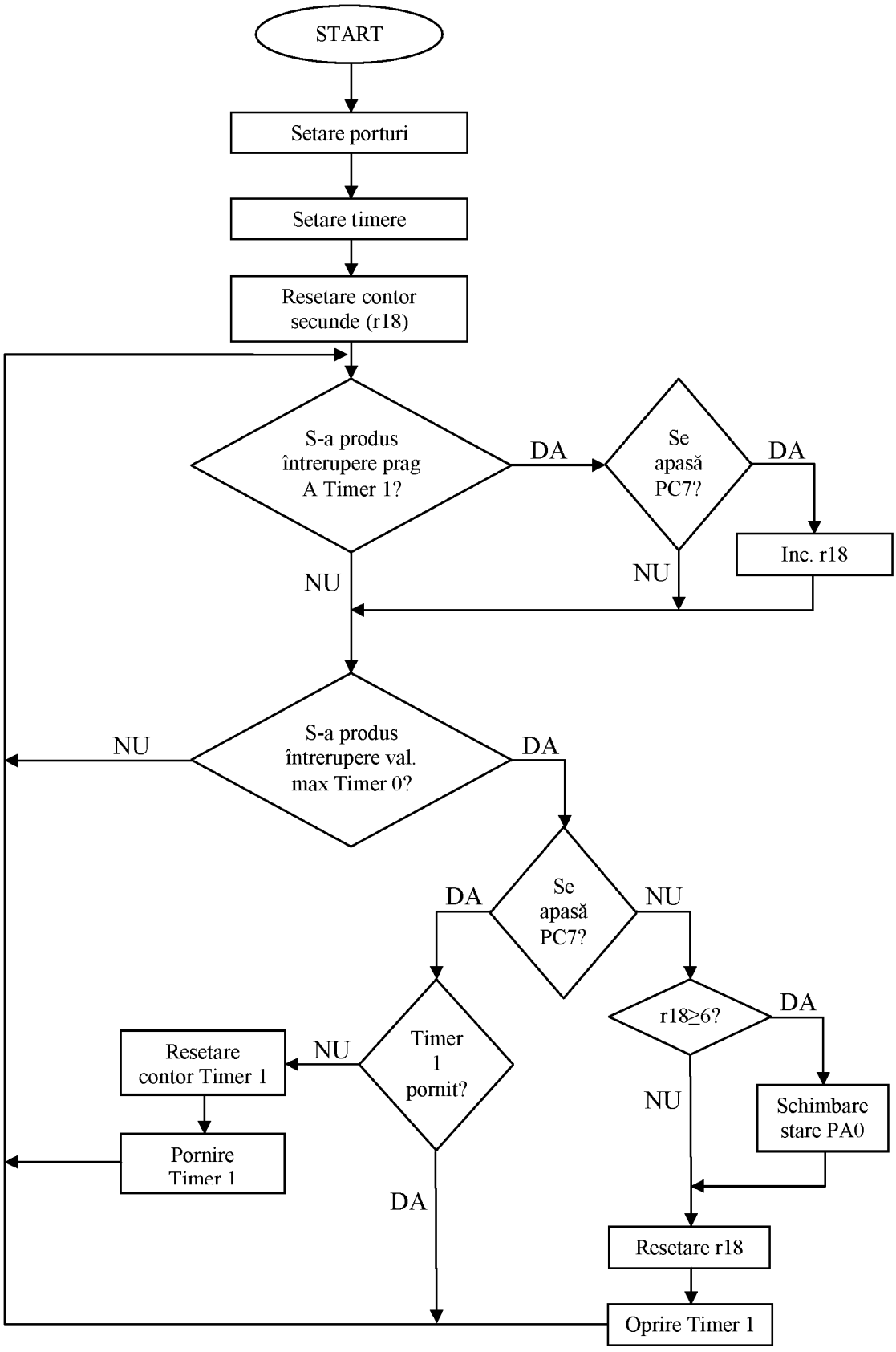



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp prag_2sec
jmp prag_6_5sec
jmp prag_8sec
jmp reset
```

```
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0xFF
out DDRA,r16
out DDRC,r16
main:
cli                 ;dezactivare globala intreruperi
ldi r16,0b00000000  ;setare timer: nu se utiliz. OC1A si OC1B, oprit deocamdata
out TCCR1A,r16      ;modul va fi normal
ldi r16,0b00000000
out TCCR1B,r16
in r16, TIMSK
andi r16,0b11000011
ori r16,0b00011100  ;se utiliz. toate intreruperile, fara a modifica alti biti din TIMSK
out TIMSK, r16
ldi r16, 0x3D       ;se incarca valoarea de prag A: 0x3D09=15625
out OCR1AH,r16            ;15625 * 1/(8MHz/1024) =~ 2s
ldi r16, 0x09
out OCR1AL,r16
ldi r16, 0xC6       ;se incarca valoarea de prag: 0xC65D=50781
out OCR1BH,r16            ;50781 * 1/(8MHz/1024) =~ 6,5s
ldi r16, 0x5D
out OCR1BL,r16
ldi r16, 0x0B       ;se incarca o valoarea initiala: 0x0BDB=3035 si 65535-3035=62500
out TCNT1H,r16            ;62500 * 1/(8MHz/1024) = 8s
ldi r16, 0xDB
out TCNT1L,r16

in r16,TCCR1B
andi r16,0b11111101       ;se porneste timerul care va numara la 1024 impulsuri de ceas
ori r16,0b00000101
out TCCR1B,r16
sei                             ;activare globala a intreruperilor
bucla:
rjmp bucla


prag_2sec:
in r20, SREG
ldi r18,0xFF
out PORTA,r18
out SREG,r20
reti

prag_6_5sec:
in r20, SREG
ldi r18,0xFF
out PORTC,r18
out SREG,r20
reti

prag_8sec:
in r20, SREG
ldi r18,0x00
out PORTA,r18
out PORTC,r18
ldi r16, 0x0B       ;se incarca o valoarea initiala: 0x0BDB=3035 si 65535-3035=62500
out TCNT1H,r16            ;62500 * 1/(8MHz/1024) = 8s
```

```
ldi r16, 0xDB
out TCNT1L,r16
out SREG,r20
reti
```

c) Să se scrie un program care schimbă starea ledului conectat pe pinul PA7 dacă s-a ţinut apăsat cel puţin 6s pe butonul conectat pe PC7.



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp prag_1sec
jmp reset
jmp reset
jmp reset
jmp scanare_butoane        ;la fiecare 32ms se verifica starea butonului
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0x00
out DDRC,r16
ldi r16,0xFF
out DDRA,r16
out PORTC,r16
ldi r18,0x00         ;r18 este contorul de secunde
main:
cli                  ;dezactivare globala intreruperi
ldi r16,0b00000000   ;setare timer: nu se utiliz. OC1A si OC1B, oprit deocamdata
out TCCR1A,r16       ;modul va fi CTC
ldi r16,0b00001000
out TCCR1B,r16
in r16, TIMSK
andi r16,0b11000011
ori r16,0b00010000   ;se utiliz. doar intrer. prag A fara a modifica alti biti din TIMSK
out TIMSK, r16
ldi r16, 0x7A        ;se incarca valoarea de prag A: 0x7A12=31250
out OCR1AH,r16             ;31250 * 1/(8MHz/256) = 1s
ldi r16, 0x12
out OCR1AL,r16

ldi r16,0b00000000   ;setare timer0: nu se utiliz OC0, oprit deocamdata
out TCCR0,r16        ;modul va fi normal
in r16, TIMSK
andi r16,0b11111100 ;se utiliz. intrer. la depasire,
ori r16,0b00000001   ;fara a modifica alti biti din TIMSK
out TIMSK, r16
in r16,TCCR0
andi r16,0b11111101 ;se porneste timerul si este setat sa numere
ori r16,0b00000101   ;la fiecare 1024 perioade de ceas, fara a modifica alti biti
out TCCR0,r16        ; va ajunge la val max. in aprox 32ms
sei                  ;activare globala a intreruperilor
bucla:
rjmp bucla


scanare_butoane:
in r20, SREG
in r16,PINC
sbrc r16,PC7               ;verific daca se apasa pe buton
rjmp turn_off
in r16,TCCR1B
mov r17,r16
andi r17,0b00000111
```

```
cpi r17,0b00000100        ;daca timerul e deja pornit nu il mai pornesc
breq end
ldi r17,0x00
out TCNT1H,r17
out TCNT1L,r17            ;inainte de o noua pornire se reseteaza
andi r16,0b11111000 ;se porneste timerul cu numarare la fiecare 256 impulsuri de ceas
ori r16,0b00000100
out TCCR1B,r16
rjmp end
turn_off:
cpi r18,6           ;s-a tinut apasat cel putin 6s ?
brlo end1           ;nu, se opreste timer si se iese din intrerupere
in r16,PINA         ;am tinut apasat cel putin 6s, citesc starea ledului
ldi r17,0b10000000  ;   ca sa o pot schimba
eor r16,r17
out PORTA, r16      ;trimit noua stare a ledului
end1:
ldi r18,0x00        ;pregatesc contorul pt. o noua numarare
in r16,TCCR1B
andi r16,0xF8
out TCCR1B,r16
end:
out SREG,r20
reti

prag_1sec:
in r20, SREG
in r16,PINC
sbrs r16,PC7        ;verific daca inca se mai apasa pe buton
inc r18
out SREG,r20
reti
```

5. Teme şi exerciţii

a) Să se scrie un program care generează pe pinul OC1A (PD5) un semnal dreptunghiular cu frecvenţa de 0,5 Hz şi factor de umplere 50%.

b) Să se scrie un program care numără câte secunde s-a ţinut apăsat pe butonul conectat pe PB6 sau cel conectat pe PC5 şi reflectă reprezentarea binară a acestui număr pe portul A respectiv portul D.

Lucrarea 3
Microcontrolerul ATmega32. Utilizarea modulului USART

Scopul lucrării

a) Prezentarea modulului USART
b) Programarea unor aplicaţii utilizând placa de dezvoltare EasyAVRv7

1. Tabela de întreruperi

Cea mai uzuală amplasare a întreruperilor este descrisă în tabelul de mai jos.

Nr. crt.	Adresa în memoria Flash	Descriere
1	0x00	Generată la alimentare sau la un semnal pe pinul RESET
2	0x02	Întrerupere externă 0
3	0x04	Întrerupere externă 1
4	0x06	Întrerupere externă 2
5	0x08	Generată când Timer/Counter2 atinge valoarea de prag
6	0x0A	Generată când Timer/Counter2 atinge valoarea maximă
7	0x0C	Generată de unitatea de captură a timerului pe 16 biţi
8	0x0E	Generată când timerul pe 16 biţi atinge valoarea de prag A
9	0x10	Generată când timerul pe 16 biţi atinge valoarea de prag B
10	0x12	Generată când timerul pe 16 biţi atinge valoarea maximă
11	0x14	Generată când Timer/Counter0 atinge valoarea de prag
12	0x16	Generată când Timer/Counter0 atinge valoarea maximă
13	0x18	Generată de unitatea SPI
14	0x1A	Generată la recepţia completă a unor date de către modulul USART
15	0x1C	Generată când registrul de date al modulului USART este gol
16	0x1E	Generată la transmisia completă a unor date de către modulul USART
17	0x20	Generată de modulul ADC
18	0x22	Generată de modulul EEPROM
19	0x24	Generată de comparatorul analogic
20	0x26	Generată de modulul TWI
21	0x28	Generată de modulul de auto-scriere a memoriei Flash

2. Prezentarea modulului USART

Modulul USART implementează o unitate de emisie şi o unitate de recepţie care funcţionează conform standardului RS232. În Figura 1 este prezentată schema bloc a modulului USART. Principalele lui caracteristici sunt:

- poate opera full duplex (unităţile de emisie şi recepţie sunt independente)
- lucrează în mod sincron sau asincron
- cadrele de date pot avea între 5 şi 9 biţi
- calculează şi verifică paritatea independent, fără intervenţie software
- poate lucra pe bază de întreruperi

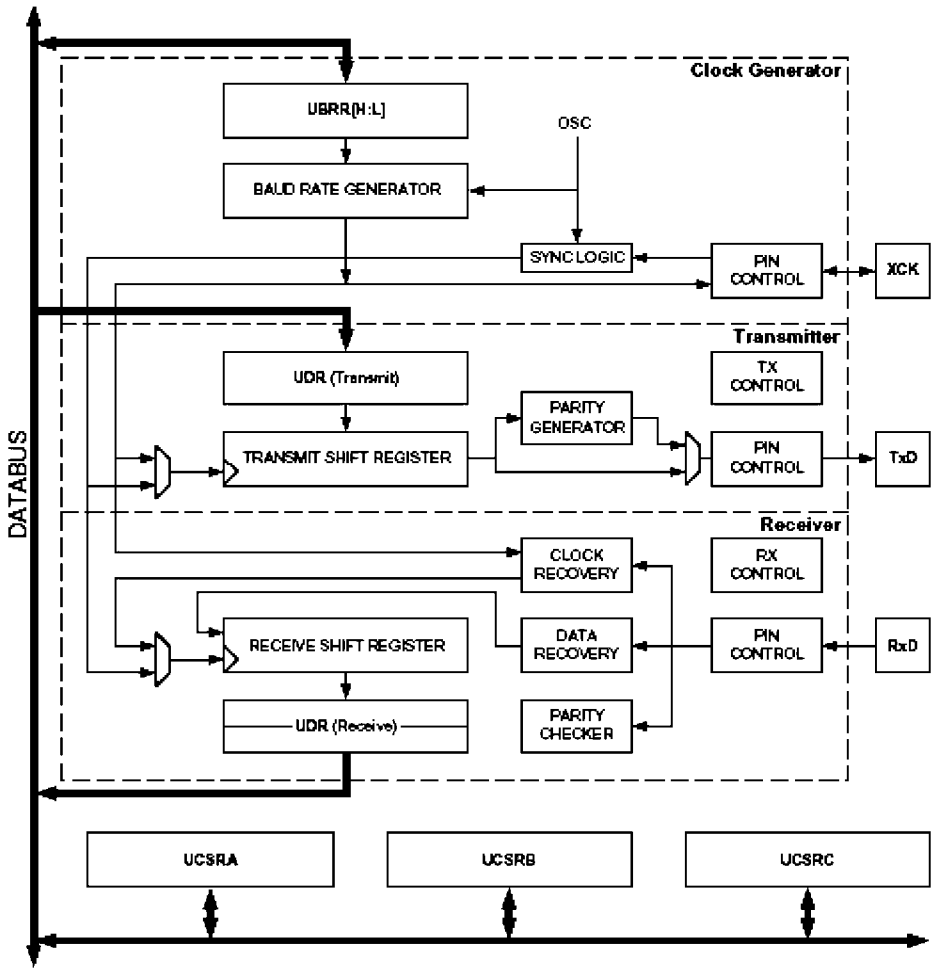


Figura 1. Schema bloc a modulului USART

Pentru a comanda şi a utiliza modulul USART, programatorul are la dispoziţie un număr de regiştrii care pot fi grupaţi în următoarele categorii:

- registrul de date pentru emisie/recepţie UDR
- registrul de 12 biţi UBRR pentru calculul vitezei de comunicaţie. UBRR este implementat sub forma a doi regiştrii de 8 biţi UBRRH şi UBRRL care se iniţializează separat
- regiştrii generali de configurare: UCSRA, UCSRB, UCSRC

2.1. *Moduri de operare*

Există patru moduri distincte de operare a modulului USART: asincron normal, asincron cu viteză dublă, sincron master şi sincron slave.

Modul asincron normal este cel mai utilizat. În acest mod dispozitivul emiţător şi cel care recepţionează au fiecare un ceas intern iar sincronizarea se face prin intermediul protocolului de comunicare. Modul asincron cu viteză dublă operează la fel ca cel normal dar viteza de comunicaţie a microcontrolerului poate fi de două ori mai mare.

În modul sincron se utilizează un singur ceas pentru emiţător şi receptor. În acest caz microcontrolerul poate fi master (generează el ceasul pentru comunicaţie) sau slave (primeşte semnalul de ceas de la dispozitivul cu care comunică).

În tabelul de mai jos este prezentat modul cum se calculează viteza de comunicaţie în modurile asincron normal, asincron cu viteză dublă şi sincron master în funcţie de valoarea din registrul UBRR şi frecvenţa ceasului microcontrolerului f_{osc}.

Modul de operare	**Viteza de comunicaţie (bps)**
Asincron normal	$\frac{f_{osc}}{16(UBRR+1)}$
Asincron viteză dublă	$\frac{f_{osc}}{8(UBRR+1)}$
Sincron master	$\frac{f_{osc}}{2(UBRR+1)}$

2.2. *Recepţia şi transmisia datelor*

Modulul USART din Atmega32 foloseşte un singur registru de date pentru emisie/recepţie şi anume UDR. Diferenţierea între cele două situaţii se face astfel: octeţii care se scriu în registrul UDR sunt automat trimişi la unitatea de emisie; atunci când programatorul solicită citirea registrului UDR, datele din unitatea de recepţie sunt încărcate automat în registru şi astfel se poate avea acces la datele recepţionate.

Modulul USART poate genera întreruperi pentru următoarele evenimente:

- recepţia corectă a unui cadru de date;
- golirea registrului de date de emisie;
- terminarea completă a emisiei unui cadru de date.

2.3. *Configurarea modulului USART*

Regiştrii utilizaţi pentru configurare sunt UCSRA, UCSRB şi UCSRC.

O descriere generală a biţilor registrului UCSRA este dată în tabelul de mai jos.

Poziţia bitului	**Denumirea bitului**	**Descriere**
7	RXC	Este 1 dacă s-a terminat recepţia; altfel este 0

6	TXC	Este 1 dacă s-a terminat emisia; altfel este 0
5	UDRE	Este 1 dacă registrul de emisie este gol; altfel este 0
4	FE	Valoarea 1 indică o eroare de format al cadrului la recepţie
3	DOR	Valoarea 1 indică situaţia în care buffer-ele de recepţie sunt pline şi apare un nou bit de start
2	PE	Valoarea 1 indică eroare de paritate la recepţie
1	U2X	Valoarea 1 activează modul asincron cu viteză dublă
0	MPCM	Valoarea 1 activează sistemul de recepţie multiprocesor

O descriere generală a biţilor registrului UCSRB este dată în tabelul de mai jos.

Poziţia bitului	**Denumirea bitului**	**Descriere**
7	RXCIE	Valoarea 1 activează întreruperea la recepţie
6	TXCIE	Valoarea 1 activează întreruperea la emisie
5	UDRIE	Valoarea 1 activează întreruperea pentru registru de emisie gol
4	RXEN	Valoarea 1 activează unitatea de recepţie
3	TXEN	Valoarea 1 activează unitatea de emisie
2	UCSZ2	Împreună cu biţii UCSZ1:0 din UCSRC determină formatul cadrului de emisie şi a celui de recepţie
1	RXB8	Reprezintă al 9-lea bit când se recepţionează cadre de 9 biţi
0	TXB8	Reprezintă al 9-lea bit când se emit cadre de 9 biţi

O descriere generală a biţilor registrului UCSRC este dată în tabelul de mai jos.

<table>
<tr><th>Poziţia bitului</th><th>Denumirea bitului</th><th colspan="3">Descriere</th></tr>
<tr><td>7</td><td>URSEL</td><td colspan="3">Trebuie să fie 1 când se scriu date în registrul UCSRC</td></tr>
<tr><td>6</td><td>UMSEL</td><td colspan="3">Valoarea 1 activează modul sincron iar valoarea 0 activează modul asincron</td></tr>
<tr><td rowspan="5">5</td><td rowspan="5">UPM1</td><td>UPM1</td><td>UPM0</td><td>Paritate</td></tr>
<tr><td>0</td><td>0</td><td>fără</td></tr>
<tr><td rowspan="3">4</td><td rowspan="3">UPM0</td><td>0</td><td>1</td><td>comb. neutiliz.</td></tr>
<tr><td>1</td><td>0</td><td>pară</td></tr>
<tr><td>1</td><td>1</td><td>impară</td></tr>
<tr><td>3</td><td>USBS</td><td colspan="3">Valoarea 0 configurează utilizarea unui bit de</td></tr>
</table>

<table>
<tr><td></td><td></td><td colspan="4">stop iar valoarea 1 utilizarea a doi biţi de stop la emisie</td></tr>
<tr><td rowspan="5">2</td><td rowspan="5">UCSZ1</td><td>UCSZ2</td><td>UCSZ1</td><td>UCSZ0</td><td>Dimen. cadru</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>5 biţi</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>6 biţi</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>7 biţi</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>8 biţi</td></tr>
<tr><td rowspan="4">1</td><td rowspan="4">UCSZ0</td><td>1</td><td>0</td><td>0</td><td>comb. neutiliz.</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>comb. neutiliz.</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>comb. neutiliz.</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>9 biţi</td></tr>
<tr><td>0</td><td>UCPOL</td><td colspan="4">Este utilizat doar pentru modul sincron</td></tr>
</table>

3. Interfaţa RS232 pe placa de dezvoltare EasyAVRv7

Placa de dezvoltare EasyAVRv7 oferă două posibilităţi de conectare la modulul USART al microcontrolerului:

- un montaj realizat cu ajutorul circuitului integrat MAX232 care asigură legătura şi interfaţa între pinii USART ai microcontrolerului (PD0 – recepţie şi PD1 – transmisie) şi alte echipamente prevăzute cu o conexiune RS232, prin intermediul unei mufe tip DB9. Circuitul integrat MAX232 realizează conversia de tensiune între nivelele TTL folosite de microcontroler şi nivelele uzuale pentru alte echipamente, mai ales PC-uri
- un montaj (USB UART) care converteşte interfaţa USART a microcontrolerului (PD0 – recepţie şi PD1 – transmisie) într-o interfaţă USB folosind circuitul integrat FTDI FT232RL

Deoarece PC-urile actuale nu mai sunt echipate cu porturi RS232, în cadrul laboratorului se va utiliza montajul cu conexiune USB (reprezentat în figura de mai jos) pentru a comunica cu interfaţa USART a microcontrolerului. Pentru aceasta sunt necesare următoarele:

- un cablu USB (altul decât cel folosit pentru programarea microcontrolerului) conectat între mufa USB UART şi un port USB al calculatorului
- instalarea pe calculator a driverelor pentru circuitul FT232RL. Astfel se va crea un port de tip COM virtual care poate fi utilizat în aplicaţiile de comunicaţie serială
- comutarea SW10 de pe placa EasyAVRv7 pe poziţia „ON” pentru pinii PD0 şi PD1
- comutarea J12 şi J23 de pe placa EasyAVRv7 pe poziţia corespunzătoare USB UART.

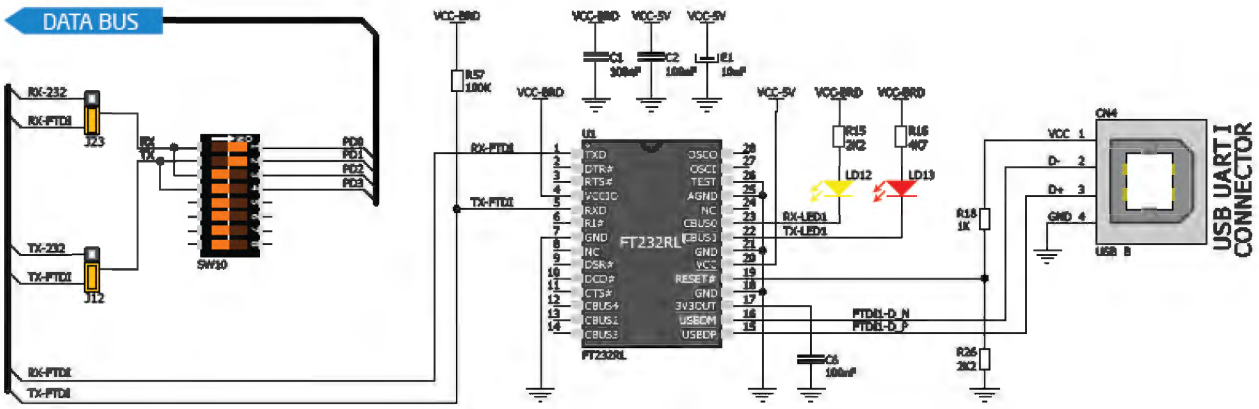


Figura 2. Montajul USB UART al plăcii EasyAVRv7

4. Software-ul Simple Serial Monitor

Simple Serial Monitor este un software dedicat pentru a recepţiona şi transmite date prin intermediul unei interfeţe COM a calculatorului. Acesta este preinstalat pe calculatoarele din laborator şi poate fi accesat din directorul My Documents. Pe sistemele de operare Windows 7/8/8.1 fişierul executabil „sermon.exe" trebuie lansat cu opţiunea „Run as administrator" pentru a funcţiona corect.

La execuţie programul afişează fereastra din Figura 3. În partea stângă sunt grupate setările pentru interfaţa serială. Astfel, câmpul „VISA resource name" trebuie să conţină denumirea portului COM creat la instalarea driverelor pentru circuitul FT232RL. Celelate câmpuri trebuie să fie configurate cu aceleaşi valori ca cele specificate în programul implementat pe microcontroler.

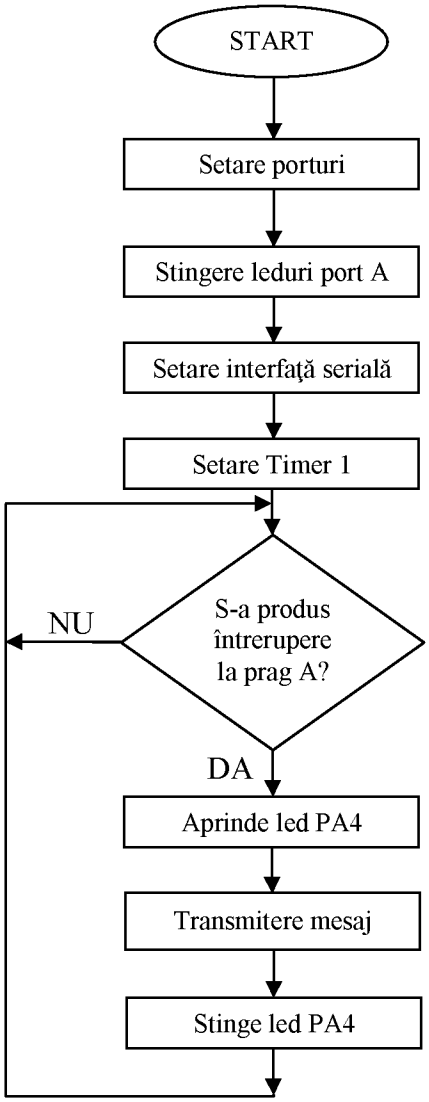
După configurarea setărilor pentru interfaţă serială, utilizatorul trebuie să apese butonul „ON/OFF" pentru a porni programul. Din acest moment se pot recepţiona date – dacă butonul „read" este în poziţia ON (culoare verde) şi se pot transmite date dacă butonul „write" este în poziţia ON (culoare verde). Datele vor fi transmise caracter cu caract pe măsură ce utilizatorul le introduce în câmpul „string to write".

Trebuie avut în vedere că datele transmise şi recepţionate de programul Simple Serial Monitor sunt interpretate sub formă de coduri ASCII.

Figura 3. Interfaţa programului Simple Serial Monitor

5. Exemple de programe

a) Să se scrie un program care trimite prin interfaţa serială textul „lab micro3” la fiecare 3s. Se utilizează mod asincron cu o viteză de 300bps, cadru de 8 biţi, paritate pară şi doi biţi de stop. În timpul emisiei se va aprinde ledul conectat pe pinul PA4.

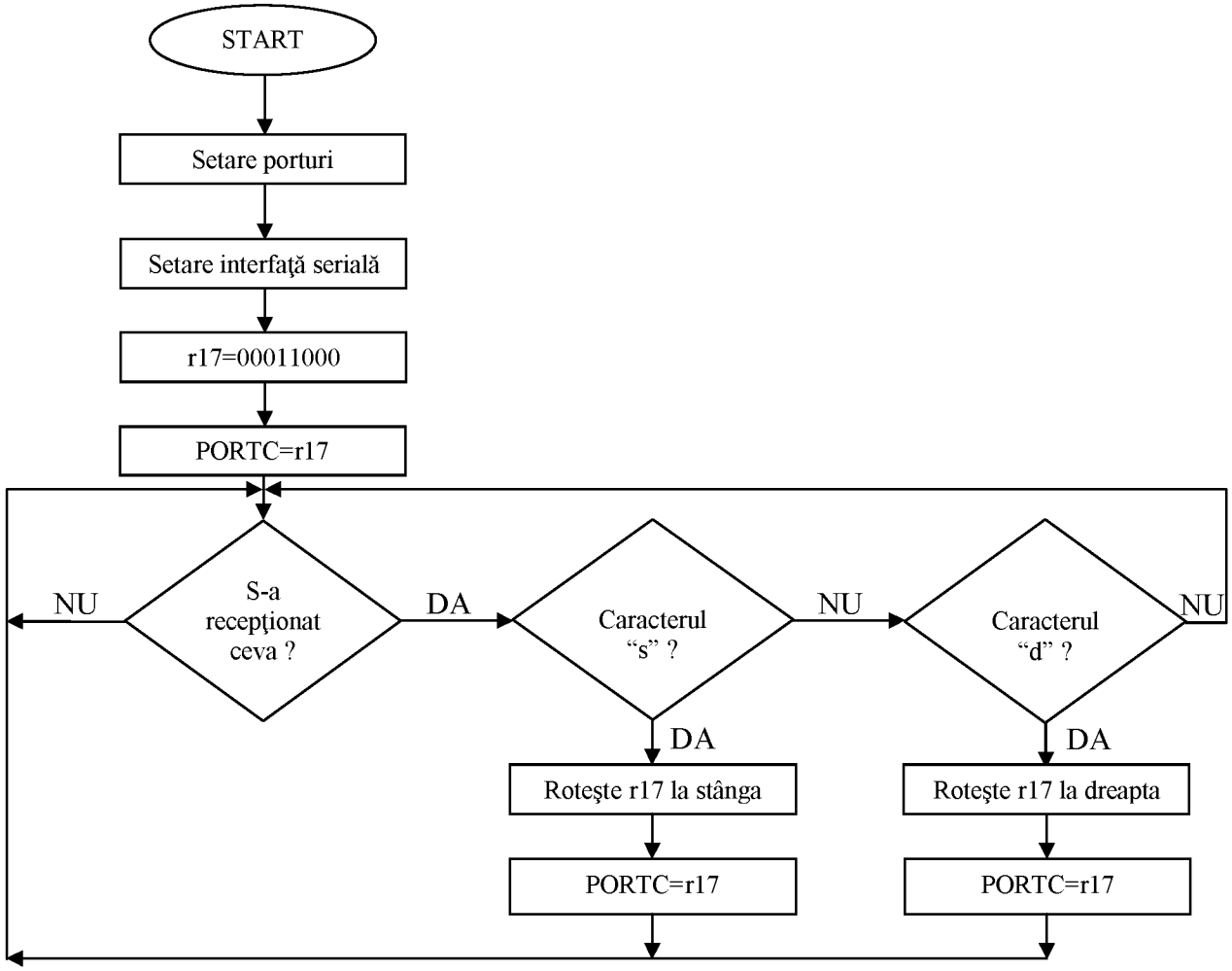



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp au_trecut_3s
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
```

```
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b11111111
out DDRA,r16 ; portul A, unde sunt conectate led-urile, este setat ca iesire
ldi r17,0x00
out porta,r17       ;sting toate ledurile
main:
cli                 ;dezactivare intreruperi
ldi r16,0b00001000  ;in randurile de mai jos se configureaza
out UCSRB,r16       ;modulul USART prin registrii UCSRB si UCSRC
ldi r16,0b10101110
out UCSRC,r16
ldi r16,0x06        ;se seteaza viteza de comunicatie
out UBRRH,r16       ;avand in vedere ca frecventa ceasului
ldi r16,0x81        ;este 8MHz
out UBRRL,r16
ldi r16,0b00000000  ;setare timer: nu se utilizeaz pinii OC1A si OC1B
out TCCR1A,r16      ;modul va fi CTC cu prag dat de OCR1A
in r16, TIMSK
andi r16,0b11010011 ;nu se utilizeaza intr. unitatii de captura
ori r16,0b00010000  ;se utilizeaza intr. de egalitate cu prag OCR1A
out TIMSK, r16      ;nu se utilizeaza intr. de egalitate cu prag OCR1B
                    ;nu se utilizeaza intr. la atingerea valorii max pe 16biti

ldi r16, 0x5B       ;se incarca valoarea de prag: 0x5B8E=23438
out OCR1AH,r16      ; 23438 * 1/(8MHz/1024) =~ 3s
ldi r16, 0x8E
out OCR1AL,r16
ldi r16,0b00001101  ;se porneste timerul si este setat sa numere
out TCCR1B,r16      ;la fiecare 1024 perioade de ceas
sei                 ;activare intreruperi
bucla:
rjmp bucla


au_trecut_3s:
in r20,SREG         ;in r20 salvez registrul de stare
ldi r16,0b00010000
out PORTA, r16      ;aprind ledul
ldi r16,0x6C        ;codul ascii pentru "l"
out UDR,r16
call wait           ;apelez o functie de asteptare

ldi r16,0x61        ;codul ascii pentru "a"
out UDR,r16
call wait

ldi r16,0x62        ;codul ascii pentru "b"
out UDR,r16
call wait

ldi r16,0x20        ;codul ascii pentru " "(spatiu)
out UDR,r16
call wait

ldi r16,0x6D        ;codul ascii pentru "m"
out UDR,r16
call wait

ldi r16,0x69        ;codul ascii pentru "i"
out UDR,r16
call wait

ldi r16,0x63        ;codul ascii pentru "c"
out UDR,r16
call wait

ldi r16,0x72        ;codul ascii pentru "r"
```

```
out UDR,r16
call wait

ldi r16,0x6F        ;codul ascii pentru "0"
out UDR,r16
call wait

ldi r16,0x33        ;codul ascii pentru "3"
out UDR,r16
call wait

ldi r16,0x20        ;codul ascii pentru " "(spatiu)
out UDR,r16
call wait

ldi r16,0x00
out PORTA,r16       ;sting ledul
out SREG,r20        ;restaurez registrul de stare
reti                ;revin din intrerupere


wait:
in r16,UCSRA        ;citesc starea modulului USART
sbrs r16,UDRE       ;verific daca registrul de emisie/receptie e gol
rjmp wait           ;registrul nu este gol, mai astept
ret                 ;registrul este gol, revin din asteptare
```

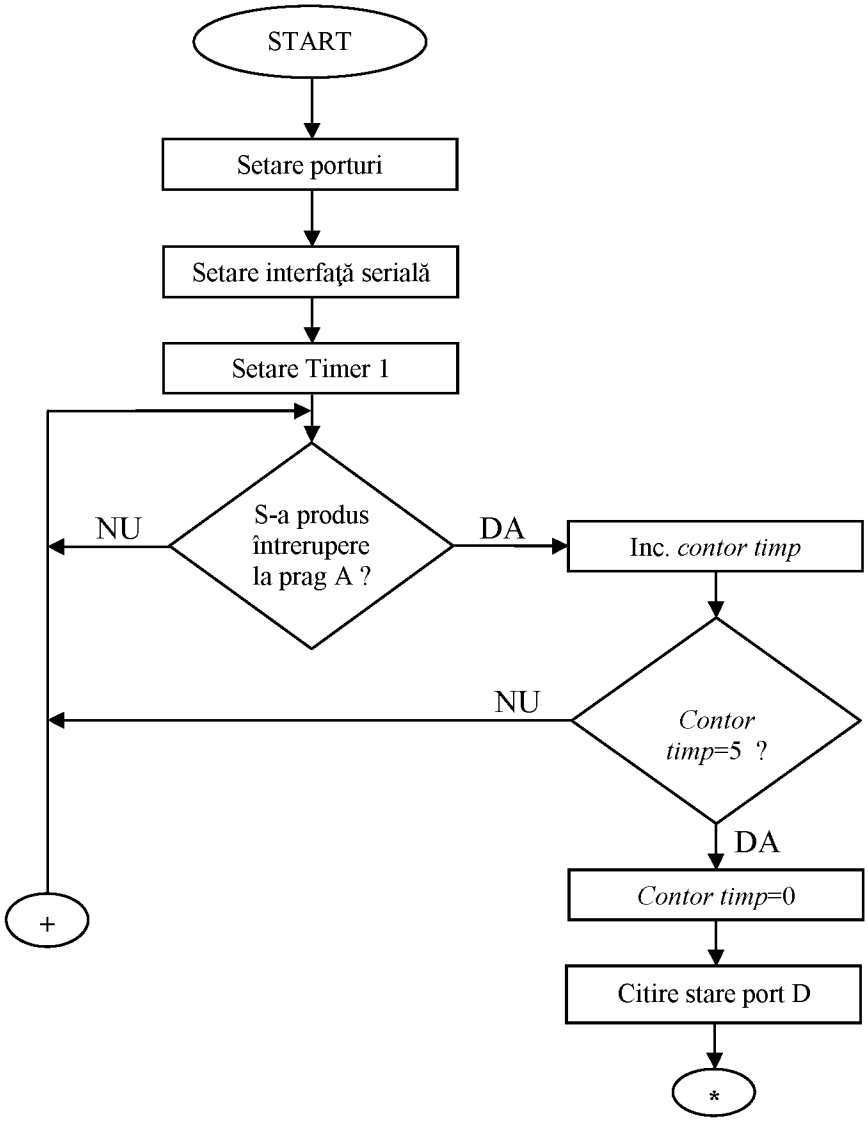
b) Să se scrie un program care aprinde iniţial ledurile conectate pe pinii PC3 şi PC4 şi primeşte următoarele comenzi pe interfaţa RS232: „s” – determină deplasarea celor două leduri cu o poziţie la stânga (către PC7), „d” – determină deplasarea celor două leduri cu o poziţie la dreapta (către PC0); orice alte caractere primite sunt ignorate. Se utilizează mod asincron cu o viteză de 9600bps, cadru de 8 biţi, paritate pară şi un bit de stop.

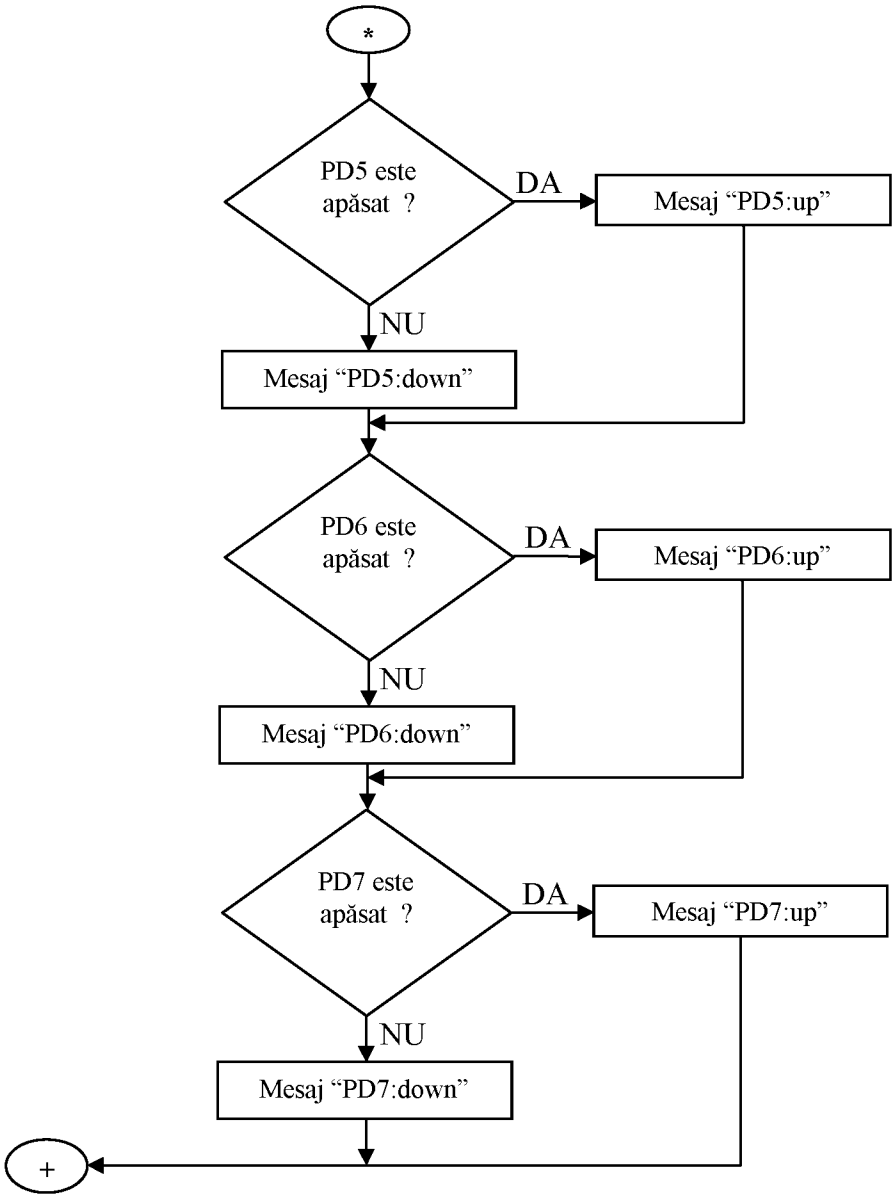


```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp receptie
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0xFF
out DDRC,r16        ; portul C este setat ca iesire
ldi r17,0b00011000  ;se aprind ledurile initiale
out portc,r17
main:
cli
ldi r16,0b10000000
out UCSRB,r16
ldi r16,0b10100110
out UCSRC,r16
ldi r16,0x00
out UBRRH,r16
ldi r16,0x33
out UBRRL,r16
sbi UCSRB,RXEN      ;activez unitatea de receptie
ldi r18,0x00        ;r18 va fi utilizat pentru a memora ultima valoare
                    ;a indicatorului carry
sei
bucla:
rjmp bucla


receptie:
in r20,SREG         ;salvez registrul de stare in r20
in r16,UDR
cpi r16,0x73        ;am receptionat "s" ?
breq PC+2           ;da, trebuie sa deplasez ledurile
                    ;(PC = Program Counter)
rjmp et2            ;nu am recptionat "s" si trec la urmatoarea verificare
sbrc r18,0          ;verfic daca anterior carry a fost 0 (aici este PC+2)
sec                 ;anterior carry a fost 1 deci acum va lua aceasta valoare
sbrs r18,0
clc                 ;anterior carry a fost 0 deci acum va lua aceasta valoare
rol r17             ;rotesc ledurile la stanga prin carry
brcc PC+3           ;salt daca in urma rotatiei carry = 0
ldi r18,0x01        ;salvez noul carry
brcs PC+2
ldi r18,0x00        ;aici este PC+3, corespunzator carry = 0
out portc,r17       ;scriu noua stare a ledurilor
rjmp end
et2:
cpi r16,0x64        ;am receptionat "d" ?
breq PC+2
rjmp end            ;nu am receptionat „d”, ignor orice alt caracter primit
```

```
sbrc r18,0
sec
sbrs r18,0
clc
ror r17
brcc PC+3
ldi r18,0x01
brcs PC+2
ldi r18,0x00
out portc,r17
end:
out SREG,r20        ;restaurez registrul de stare
reti                ;revin din intrerupere
```

c) Să se scrie un program care la fiecare 10s transmite pe interfaţa RS232 starea butoanelor PD5, PD6 şi PD7 sub forma unui mesaj text de tipul „PD*nr_buton*: up” sau „PD*nr_buton*: down”. Se presupune că apăsarea unui buton aduce pinul respectiv la nivelul 0 logic. Se utilizează mod asincron cu o viteză de 9600bps, cadru de 8 biţi, paritate pară şi un bit de stop.






```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp au_trecut_2s
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
```

```
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b00000000
out DDRD,r16              ;portul D este setat ca intrare
ldi r16,0b11100000
out PORTD,r16             ;se activeaza rezistentele pull-up pentru PD5, PD6 si PD7
ldi r18,0x00              ;r18 este utilizat pt numararea secundelor
main:
cli                       ;dezactivare intreruperi
ldi r16,0b00001000        ;in randurile de mai jos se configureaza
out UCSRB,r16             ;modulul USART prin registrii UCSRB si UCSRC
ldi r16,0b10100110
out UCSRC,r16
ldi r16,0x00              ;se seteaza viteza de comunicatie
out UBRRH,r16             ;avand in vedere ca frecventa ceasului
ldi r16,0x33              ;este 8MHz
out UBRRL,r16
ldi r16,0b00000000        ;setare timer: nu se utilizeaz pinii OC1A si OC1B
out TCCR1A,r16            ;modul va fi CTC cu prag dat de OCR1A
in r16, TIMSK
andi r16,0b11010011       ;nu se utilizeaza intr. unitatii de captura
ori r16,0b00010000        ;se utilizeaza intr. de egalitate cu prag OCR1A
out TIMSK, r16            ;nu se utilizeaza intr. de egalitate cu prag OCR1B
                          ;nu se utilizeaza intr. la atingerea valorii max pe 16biti

ldi r16, 0xF4             ;se incarca valoarea de prag: 0xF424=62500
out OCR1AH,r16            ;62500 * 1/(8MHz/256) = 2s
ldi r16, 0x24
out OCR1AL,r16
ldi r16,0b00001100        ;se porneste timerul si este setat sa numere
out TCCR1B,r16            ;la fiecare 256 perioade de ceas
sei                       ;activare intreruperi
bucla:
rjmp bucla


au_trecut_2s:
in r20,SREG               ;in r20 salvez registrul de stare
inc r18
cpi r18,0x05              ;au trecut 2 x 5 = 10s ?
brlo end
ldi r18,0x00

in r16, PIND
ldi r17,0x35              ;codul ascii pentru 5
call msg_generic
sbrc r16, PD5             ;se verifica PD5
call msg_up
sbrs r16, PD5
call msg_down

ldi r17,0x36              ;codul ascii pentru 6
call msg_generic
sbrc r16, PD6             ;se verifica PD6
call msg_up
sbrs r16,PD6
call msg_down

ldi r17,0x37              ;codul ascii pentru 7
call msg_generic
sbrc r16, PD7             ;se verifica PD7
call msg_up
sbrs r16,PD7
call msg_down

ldi r19,0x0d              ;codul ascii pentru enter (CR)
out UDR,r19
call wait                 ;apelez o functie de asteptare
ldi r19,0x0a              ;codul ascii pentru enter (LF)
out UDR,r19
call wait                 ;apelez o functie de asteptare
```

```
end:
out SREG,r20              ;restaurez registrul de stare
reti                      ;revin din intrerupere



msg_generic:
ldi r19,0x50        ;codul ascii pentru "P"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x44        ;codul ascii pentru "D"
out UDR,r19
call wait           ;apelez o functie de asteptare

out UDR,r17         ;codul ascii corespunzator butonului
call wait           ;apelez o functie de asteptare
ldi r19,0x3A        ;codul ascii pentru ":"
out UDR,r19
call wait           ;apelez o functie de asteptare
ret

msg_up:
ldi r19,0x75        ;codul ascii pentru "u"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x70        ;codul ascii pentru "p"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x20        ;codul ascii pentru " "(spatiu)
out UDR,r19
call wait           ;apelez o functie de asteptare
ret

msg_down:
ldi r19,0x64        ;codul ascii pentru "d"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x6f        ;codul ascii pentru "o"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x77        ;codul ascii pentru "w"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x6e        ;codul ascii pentru "n"
out UDR,r19
call wait           ;apelez o functie de asteptare
ldi r19,0x20        ;codul ascii pentru " "(spatiu)
out UDR,r19
call wait           ;apelez o functie de asteptare
ret

wait:
in r21,UCSRA        ;citesc starea modulului USART
sbrs r21,UDRE       ;verific daca registrul de emisie/receptie e gol
rjmp wait           ;registrul nu este gol, mai astept
ret                 ;registrul este gol, revin din asteptare
```

6. Teme şi exerciţii

a) Să se scrie un program care primeşte pe interfaţa RS232 un număr cuprins între 0 şi 7 şi răspunde cu starea butonului PA corespunzător. Orice alte caractere primite se vor ignora. Mesajul de răspuns va fi un text de forma „sus" (coduri ASCII 115, 117 şi 115) sau „jos" (coduri ASCII 106, 111 şi 115). Imediat după ce se recepţionează un caracter valid, se va aprinde ledul PB corespunzător, se va menţine aprins pe toată perioada transmisiei mesajului de răspuns şi apoi se va stinge. Se utilizează mod asincron cu o viteză de 2400bps, cadru de 8 biţi, paritate pară şi un bit de stop.

Lucrarea 4
Microcontrolerul ATmega32. Utilizarea modulului de conversie A/D
Utilizarea unui afişor LCD alfanumeric

Scopul lucrării

a) Prezentarea modulului de conversie A/D al microcontrolerului ATmega32
b) Descrierea modului de utilizare a unui modul de afişare LCD alfanumeric
b) Programarea unor aplicaţii utilizând placa de dezvoltare EasyAVRv7

1. Tabela de întreruperi

Cea mai uzuală amplasare a întreruperilor este descrisă în tabelul de mai jos.

Nr. crt.	Adresa în memoria Flash	Descriere
1	0x00	Generată la alimentare sau la un semnal pe pinul RESET
2	0x02	Întrerupere externă 0
3	0x04	Întrerupere externă 1
4	0x06	Întrerupere externă 2
5	0x08	Generată când Timer/Counter2 atinge valoarea de prag
6	0x0A	Generată când Timer/Counter2 atinge valoarea maximă
7	0x0C	Generată de unitatea de captură a timerului pe 16 biţi
8	0x0E	Generată când timerul pe 16 biţi atinge valoarea de prag A
9	0x10	Generată când timerul pe 16 biţi atinge valoarea de prag B
10	0x12	Generată când timerul pe 16 biţi atinge valoarea maximă
11	0x14	Generată când Timer/Counter0 atinge valoarea de prag
12	0x16	Generată când Timer/Counter0 atinge valoarea maximă
13	0x18	Generată de unitatea SPI
14	0x1A	Generată la recepţia completă a unor date de către modulul USART
15	0x1C	Generată când registrul de date al modulului USART este gol
16	0x1E	Generată la transmisia completă a unor date de către modulul USART
17	0x20	Generată de modulul ADC
18	0x22	Generată de modulul EEPROM
19	0x24	Generată de comparatorul analogic
20	0x26	Generată de modulul TWI
21	0x28	Generată de modulul de auto-scriere a memoriei Flash

2. Prezentarea modulului de conversie A/D

Modulul de conversie A/D (ADC – Analog to Digital Converter) permite transformarea semnalelor analogice aplicate pe anumiţi pini ai microcontrolerului în valori numerice. În Figura 1 este prezentată schema bloc a modulului ADC. Principalele lui caracteristici sunt:

- mod de lucru pe bază de aproximaţii succesive
- alimentare separată pe pinul AVCC
- viteză maximă de 15kSPS
- rezoluţie de 10 biţi
- 8 canale multiplexate, ADC0...ADC7 corespunzătoare pinilor PA0...PA7
- rezultatul conversiei poate fi aliniat la stânga, simplificând calculele cu preţul reducerii rezoluţiei la 8 biţi
- poate genera întrerupere la finalizarea conversiei
- se pot selecta trei tipuri de nivele de referinţă
- se pot selecta mai multe surse pentru iniţierea unei conversii, incluzând un mod de funcţionare permanentă

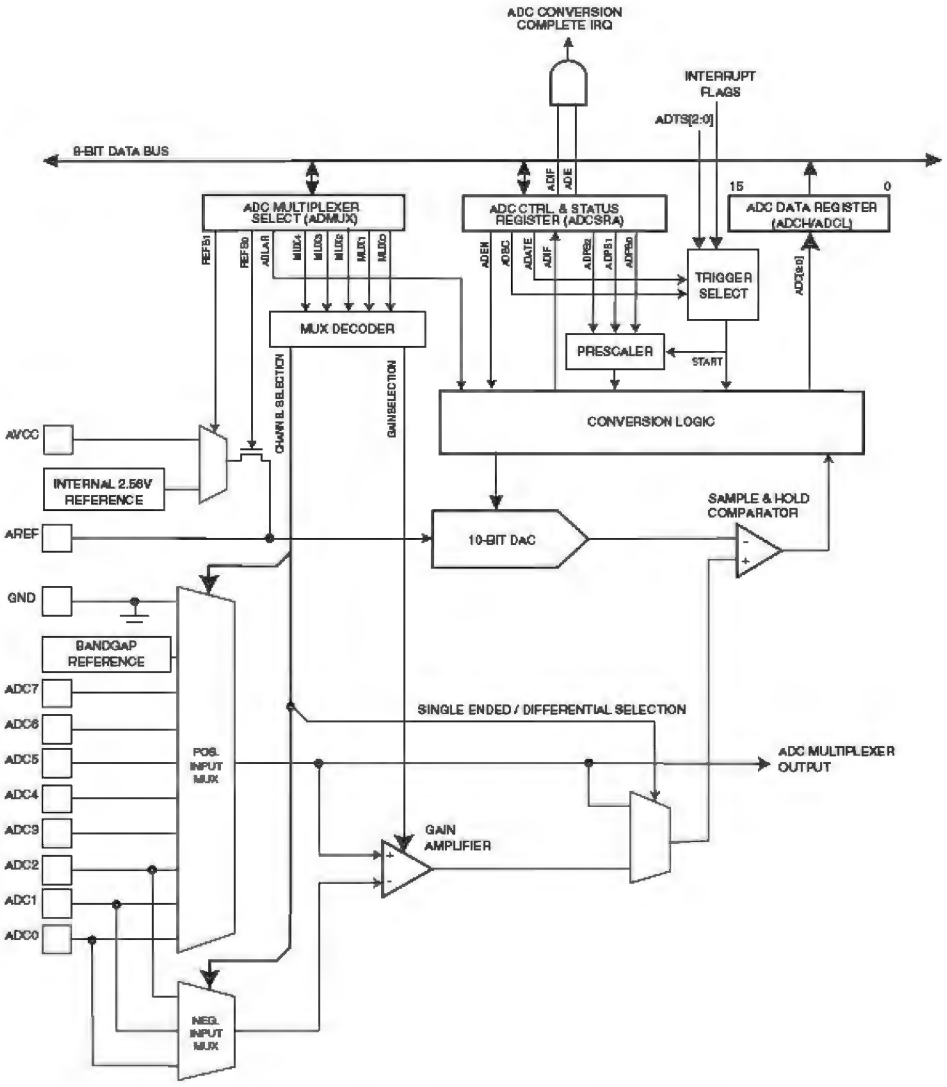


Figura 1. Schema bloc a modulului ADC

Rolul oricărui modul de conversie A/D este de a transforma o tensiune analogică într-o valoare numerică. Cele mai importante elemente care definesc această transformare sunt valoarea tensiunii de referinţă şi rezoluţia convertorului. Astfel, rezoluţia stabileşte numărul maxim de trepte în care se poate cuantiza tensiunea de intrare iar tensiunea de referinţă stabileşte valoarea maximă a tensiunii de intrare.

Dacă notăm cu N rezoluţia şi V_{ref} tensiunea de referinţă, atunci treapta unui converter A/D sau 1 LSB se defineşte ca:

$$1\ LSB = V_{ref} / 2^N \ (V)$$

Valoarea treptei se exprimă în volţi şi în funcţie de aceasta şi rezultatul conversiei C, se poate afla tensiunea de intrare în convertor V_{in} pe baza relaţiei:

$$V_{in} = C*1\ LSB\ (V)$$

Valoarea maximă a tensiunii de intrare care poate fi măsurată este întotdeauna $V_{ref} - 1$ LSB.

Modulul ADC din ATmega16 poate avea trei surse ca tensiune de referinţă:

- valoarea de pe pinul AVCC
- valoarea de pe pinul AREF
- o tensiune generată intern cu valoarea de 2,56V

În cadrul acestui laborator se va utiliza ca tensiune de referinţă valoarea tensiunii de alimentare de 5V pe care placa de dezvoltare EasyAVRv7 o conectează pe pinul AREF. Folosind formulele prezentate anterior, pentru rezoluţia de 10 biţi rezultă că 1 LSB=4,88mV.

Pentru a comanda şi a utiliza modulul ADC, programatorul are la dispoziţie un număr de regiştrii care pot fi grupaţi în următoarele categorii:

- regiştrii ADCL şi ADCH pentru stocarea rezultatului unei conversii
- regiştrii generali de configurare: ADMUX şi ADCSRA
- registrul special de configurare SFIOR, utilizat şi de alte module ale microcontrolerului.

2.1. Configurarea şi utilizarea modulului de conversie A/D

La finalul unei conversii, rezultatul acesteia se va regăsi în regiştrii ADCH şi ADCL. În mod normal, se citeşte mai întâi ADCL şi apoi ADCH pentru a utiliza precizia completă de 10 biţi. Cu toate acestea, dacă s-a configurat alinierea rezultatului la stânga (Figura 3), utilizatorul poate citi numai registrul ADCH obţinând astfel o valoare mai uşor de prelucrat dar care corespunde unei precizii a convertorului de numai 8 biţi.

Convertorul poate măsura tensiunea aplicată pe oricare din cele opt canale reprezentate de pinii ADC0...ADC7, dar nu simultan. De aceea, numai unul dintre acestea va fi activ la un moment dat, selecţia realizându-se pe baza unor biţi din registrul ADMUX. Trebuie avut în vedere că aceşti biţi pot fi oricând modificaţi dar, la nivelul hardware, selecţia unui nou canal se face imediat numai dacă nici o conversie nu este în desfăşurare. Altfel, selecţia se va realiza după ce se termină conversia curentă.

Regiştrii utilizaţi pentru configurare sunt ADMUX, ADCSRA şi SFIOR.

O descriere generală a biţilor registrului ADMUX este dată în tabelul de mai jos.

<table>
<tr><th>Poziţia bitului</th><th>Denumirea bitului</th><th colspan="3">Descriere</th></tr>
<tr><td>7</td><td>REFS1</td><td>REFS1</td><td>REFS0</td><td>Sursa tensiunii de referinţă</td></tr>
<tr><td rowspan="4">6</td><td rowspan="4">REFS0</td><td>0</td><td>0</td><td>AREF</td></tr>
<tr><td>0</td><td>1</td><td>AVCC</td></tr>
<tr><td>1</td><td>0</td><td>Neutilizat</td></tr>
<tr><td>1</td><td>1</td><td>2,56V intern</td></tr>
<tr><td>5</td><td>ADLAR</td><td colspan="3">Valoarea 0 determină alinierea la dreapta a rezultatului conversiei, aşa cum se poate vedea în Figura 2. Valoarea 1 determină alinierea la stânga a rezultatului conversiei, ca în Figura 3.</td></tr>
<tr><td>4</td><td>MUX4</td><td colspan="3" rowspan="5">Combinaţia acestor biţi selectează canalul de intrare pentru ADC. În cadrul laboratorului se va utiliza doar setarea 00000 care corespunde canalului ADC0, adică pinul PA0 al microcontrolerului.</td></tr>
<tr><td>3</td><td>MUX3</td></tr>
<tr><td>2</td><td>MUX2</td></tr>
<tr><td>1</td><td>MUX1</td></tr>
<tr><td>0</td><td>MUX0</td></tr>
</table>

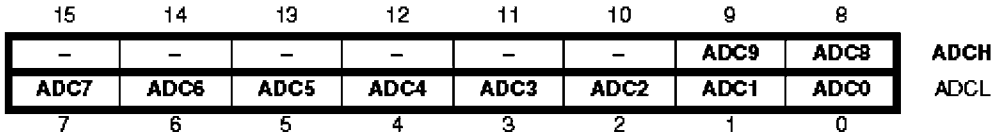


Figura 2. ADCH şi ADCL în cazul alinierii la dreapta a rezultatului conversiei

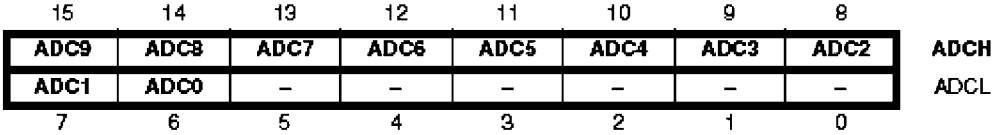


Figura 3. ADCH şi ADCL în cazul alinierii la stânga a rezultatului conversiei

O descriere generală a biţilor registrului ADCSRA este dată în tabelul de mai jos. Utilizatorul trebuie să acorde atenţie mai ales biţilor ADPS0...ADPS2 care stabilesc frecvenţa ceasului convertorului funcţie de frecvenţa de tact a procesorului, f_{osc}. Astfel, convertorul are nevoie de un semnal de tact cuprins între 50kHz şi 200kHz pentru a funcţiona corect.

Poziţia bitului	Denumirea bitului	Descriere
7	ADEN	Acest bit trebuie să fie 1 pentru ca modulul ADC să funcţioneze. Altfel, acesta este oprit
6	ADSC	Când acest bit devine 1 se iniţiază începerea unei conversii

5	ADATE	Când acest bit este 1, după o primă conversie iniţiată de utilizator, convertorul va realiza automat alte conversii în funcţie de evenimentele selectate în registrul SFIOR			
4	ADIF	Acest bit devine 1 la terminarea unei conversii			
3	ADIE	Valoarea 1 activează întreruperea la finalizarea conversiei			
2	ADPS2	**ADPS2**	**ADPS1**	**ADPS0**	**Ceasul ADC**
1	ADPS1	0	0	0	f_{osc} / 2
		0	0	1	f_{osc} / 2
		0	1	0	f_{osc} / 4
		0	1	1	f_{osc} / 8
0	ADPS0	1	0	0	f_{osc} / 16
		1	0	1	f_{osc} / 32
		1	1	0	f_{osc} / 64
		1	1	1	f_{osc} / 128

Biţii 5 (ADTS0), 6 (ADTS1) şi 7 (ADTS2) din registrul SFIOR selectează evenimentul care declanşează o nouă conversie în modul de funcţionare automat (ADATE=1). Aceste evenimente sunt de fapt semnale de întrerupere şi utilizarea lor va determina activarea flagului corespunzător. Acesta trebuie să fie dezactivat după ce s-a realizat o conversie pentru a permite o nouă autodeclanşare la respectivul eveniment. Deoarece acest registru este utilizat şi de alte module ale microcontrolerului, utilizatorul va avea grijă să nu modifice decât valoarea biţilor 5..7 ale căror combinaţii posibile sunt descrise în tabelul de mai jos.

ADTS2	**ADTS1**	**ADTS0**	**Eveniment de autodeclanşare**
0	0	0	Funcţionare permanentă: după terminarea unei conversii se începe imediat alta
0	0	1	Semnalul de întrerupere generat de comparatorul analogic
0	1	0	Semnalul de întrerupere generat de întreruperea externă 0
0	1	1	Semnalul de întrerupere generat când Timer/Counter0 atinge valoarea de prag
1	0	0	Semnalul de întrerupere generat când Timer/Counter0 atinge valoarea maximă
1	0	1	Semnalul de întrerupere generat când timerul pe 16 biţi atinge valoarea de prag B
1	1	0	Semnalul de întrerupere generat când timerul pe 16 biţi atinge valoarea maximă
1	1	1	Semnalul de întrerupere generat de unitatea de captură a timerului pe 16 biţi

3. Prezentarea modulelor LCD alfanumerice

Un modul LCD alfanumeric este format dintr-un ecran LCD şi o serie de circuite integrate care asigură comanda şi controlul acestuia. Astfel, se pot afişa caractere şi cifre prin intermediul unei interfeţe de tip magistrală formată din 8 sau 4 biţi de date şi 3 semnale de comandă: RS, R/W şi E.

Orice modul LCD are prestabilit un anumit set de caractere şi dispune de două tipuri de memorii RAM: una pentru afişare de caractere şi una pentru a memora modele pentru caractere noi.

Ficărui caracter de pe ecran îi corespunde o locaţie prestabilită din memoria RAM pentru afişare. Astfel, atunci când utilizatorul doreşte să afişeze un caracter, el trebuie să scrie codul caracterului în locaţia RAM corespunzătoare. După ce s-a efectuat o scriere, adresa RAM este automat incrementată de modulul de afişare astfel că, dacă se afişează pe poziţii succesive, nu mai este nevoie să se seteze de fiecare dată adresa. Pentru un afişor cu 2 rânduri a câte 16 caractere fiecare, corespondenţa între adresa din memoria RAM de afişare (valori în hexazecimal) şi poziţia caracterului pe ecran este dată în tabelul de mai jos.

Poziţia pe ecran	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Adrese rând 1 (cel de sus)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Adrese rând 2 (cel de jos)	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Setul prestabilit de caractere defineşte codul asociat fiecărui caracter pe care îl poate afişa modulul respectiv şi diferă de la un producător la altul. Cu toate acestea, în majoritatea cazurilor cifrele, literele mari şi mici, semnele de punctuaţie precum şi simbolurile matematice uzuale au acelaşi cod ca cel din tabela ASCII.

Comunicarea cu un modul de afişare LCD se face prin intermediul unor comenzi descrise în documentaţia producătorului. Aceste comenzi sunt definite de combinaţia semnalelor RS, R/W şi a unui număr de 8 biţi de date. În plus, atunci când semnalul E trece din 0 în 1, datele de pe magistrala de comunicaţie (având 8 sau 4 biţi) sunt preluate de către modulul de afişare. Atunci când se lucrează cu o magistrală de 4 biţi, se transmite mai întâi partea superioară din cei 8 biţi de date ai unei comenzi şi apoi partea inferioară.

După ce a primit toţi biţii corespunzători unei comenzi, modulul LCD începe execuţia acesteia. În această perioadă nu se mai poate transmite nici o altă comandă, prin urmare utilizatorul trebuie fie să aştepte timpul de execuţie minim specificat de producător, fie să citească în permanenţă indicatorul Busy al modulului LCD până când acesta semnalează finalizarea comenzii.

Setul de comenzi al unui modul LCD poate fi împărţit în următoarele categorii generale:

- comenzi de iniţializare şi configurare
- comanda de selecţie a unei adrese din memoria RAM pentru afişare de caractere

- comanda de selecţie a unei adrese din memoria RAM pentru memorarea de modele pentru caractere noi
- comanda de citire a indicatorului Busy
- comanda de scriere a datelor în una din memoriile RAM selectate
- comanda de citire a datelor din una dintre memoriile RAM selectate

Pentru a selecta o adresă din memoria RAM pentru afişare de caractere, semnalele R/W şi RS trebuie să fie 0, bitul de date 7 trebuie să fie 1 iar biţii de date 0...6 trebuie să conţină valoarea adresei care se selectează.

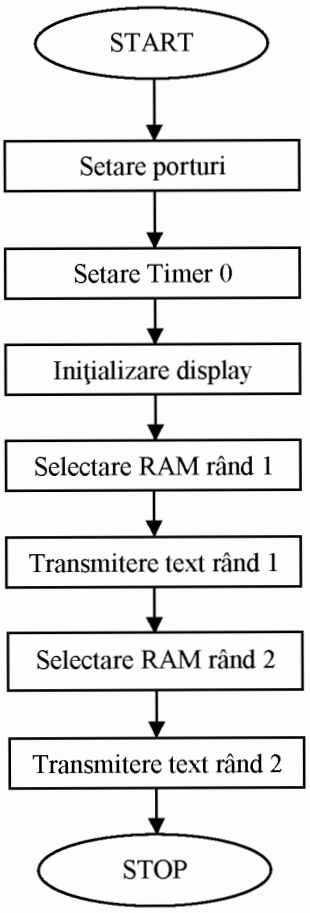
Pentru a scrie o valoare la o adresă selectată din oricare memorie RAM, semnalul R/W trebuie să fie 0, semnalul RS trebuie să fie 1 iar biţii de date trebuie să conţină valoarea care se doreşte a fi scrisă.

În cadrul acestui laborator se va utiliza un modul LCD cu 2 linii a câte 16 caractere fiecare, conectat pe o magistrală de 4 biţi: PC4, PC5, PC6 şi PC7. Semnalul RS este comandat de PA2, semnalul R/W este conectat în permanenţă la 0V iar semnalul E este comandat de PD6.

Pe perioada de execuţie a unei comenzi de către modulul de afişare se vor genera intervale de aşteptare cu aproximativ 10% mai mari decât timpii de execuţie specificaţi de producător.

4. Exemple de programe

a) Să se scrie un program care afişează pe rândul 1 al modulului LCD textul „Lab. microC 4” şi pe rândul 2 textul „Mesaj test”.



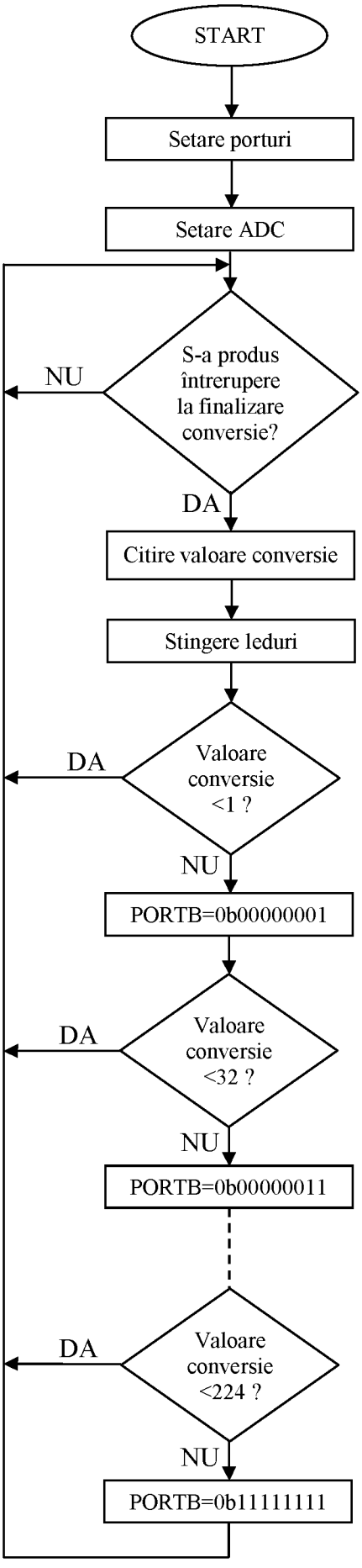

```
.include "m32def.inc"
.equ rs=PA2
.equ e=PD6
.equ ctrl=PORTD
.equ ctrl2=PORTA

jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
```

```
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b11110000
out DDRC,r16        ; pinii PC7...PC4 sunt iesiri
ldi r16,0b01000000
out DDRD,r16        ; pinul PD6 este iesire
ldi r16,0b00000100
out DDRA,r16        ; pinul PA2 este iesire
main:
cli
cbi ctrl,e
cbi ctrl2,rs
ldi r16,0b00001000  ;setare timer0:nu se utiliz. pinul OC0, timerul este oprit deocamdata
out TCCR0,r16       ;modul va fi CTC cu prag dat de OCR0
in r16, TIMSK
andi r16,0b11111100 ;nu se utiliz. nici o intrer., fara a modifica alti biti din TIMSK
out TIMSK, r16

call init_display
ldi r17,0b10000000
call set_ram        ;prin r17 se seteaza adresa 0x00 pentru RAM de afisare
ldi r17,'L'
call put_char
ldi r17,'a'
call put_char
ldi r17,'b'
call put_char
ldi r17,'.'               ;punct
call put_char
ldi r17,' '               ;spatiu
call put_char
ldi r17,'m'
call put_char
ldi r17,'i'
call put_char
ldi r17,'c'
call put_char
ldi r17,'r'
call put_char
ldi r17,'o'
call put_char
ldi r17,'C'
call put_char
ldi r17,' '
call put_char
ldi r17,'4'
call put_char

ldi r17,0b11000011  ;prin r17 se seteaza adresa 0x43 pentru RAM de afisare
call set_ram
ldi r17,'M'
call put_char
ldi r17,'e'
call put_char
ldi r17,'s'
call put_char
ldi r17,'a'
call put_char
ldi r17,'j'
call put_char
ldi r17,' '
call put_char
ldi r17,'t'
call put_char
ldi r17,'e'
call put_char
ldi r17,'s'
call put_char
ldi r17,'t'
call put_char
```

```
bucla:
rjmp bucla

init_display:
cbi ctrl2,rs
ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e

ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b10000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b11000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b00010000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms
ret


set_ram:
cbi ctrl2,rs
mov r16,r17
andi r16,0xF0              ;se retine doar nibble (4 biti) superior
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
mov r16,r17
```

```
andi r16,0x0F             ;se retine doar nibble (4 biti) inferior
swap r16                  ;interschimba nibble (4 biti) sup. cu nibble (4 biti) inf.
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
call wait_48us
ret

put_char:
sbi ctrl2,rs
mov r16,r17
andi r16,0xF0             ; se retine doar nibble (4 biti) superior
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
mov r16,r17
andi r16,0x0F             ; se retine doar nibble (4 biti) inferior
swap r16                  ; interschimba nibble (4 biti) sup. cu nibble (4 biti) inf.
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
call wait_48us
ret

wait_48us:
ldi r16,0x00
out TCNT0,r16
ldi r16, 0x06       ;se incarca valoarea de prag: 0x06=6
out OCR0,r16        ;6 * 1/(8MHz/64) =48us
in r16,TCCR0
andi r16,0b11111000       ;se porneste timerul si este setat sa numere
ori r16,0b00000011        ;      la fiecare 64 perioade de ceas, fara a modif. alti biti
out TCCR0,r16
wait:
in r16,TIFR
sbrs r16,OCF0       ;se asteapta atingerea pragului OCR0
rjmp wait
in r16,TIFR
ori r16,0b00000010
out TIFR,r16        ;se reseteaza flagul
in r16,TCCR0
andi r16,0b11111000 ;se opreste timerul
out TCCR0,r16
ret

wait_30ms:
ldi r16,0x00
out TCNT0,r16
ldi r16, 0xF0       ;se incarca valoarea de prag: 0xF0=240
out OCR0,r16        ;240 * 1/(8MHz/1024) =~ 30ms
in r16,TCCR0
andi r16,0b11111000       ;se porneste timerul si este setat sa numere
ori r16,0b00000101        ;      la fiecare 1024 per. de ceas, fara a modif. alti biti
out TCCR0,r16
wait1:
in r16,TIFR
sbrs r16,OCF0       ;se asteapta atingerea pragului OCR0
rjmp wait1
in r16,TIFR
ori r16,0b00000010
out TIFR,r16        ;se reseteaza flagul
in r16,TCCR0
andi r16,0b11111000 ;se opreste timerul
out TCCR0,r16
ret
```

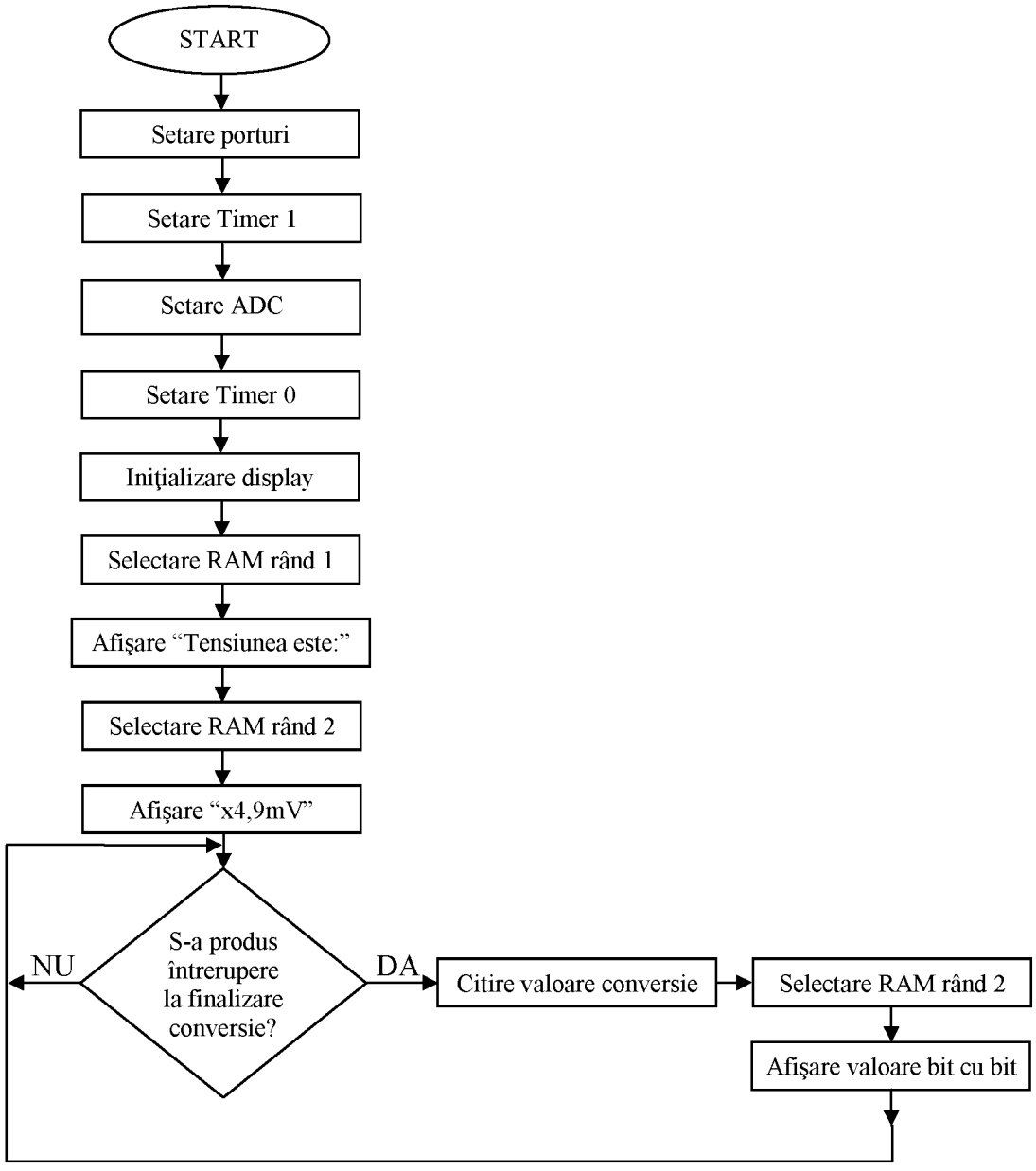
b) Să se scrie un program de tip „VU-metru” care aprinde succesiv ledurile conectate pe pinii PB0...PB7 în funcţie de valoarea tensiunii aplicată pe pinul ADC6 al microcontrolerului.



```
.include "m32def.inc"
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp gata_conversia
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0xFF
out DDRB,r16 ; portul B este iesire
main:
cli
ldi r16,0b00100110  ;setare ADC: AREF este referinta, canal ADC6
out ADMUX,r16       ;rezultatul se aliniaza la stanga
ldi r16,0b10101110  ;activare intrer., setare ceas ca fosc/64
out ADCSRA,r16             ;deocamdata nu s-a initiat conversia
in r16, SFIOR
andi r16,0b00011111 ;setarea modului de funct. perm., fara a modifica alti biti
out SFIOR, r16
sei
sbi ADCSRA,ADSC            ;se initiaza conversia
bucla:
rjmp bucla

gata_conversia:
in r20,SREG
in r17,ADCH         ;se citeste rezultatul conversiei, pastrand precizie de numai 8 biti
ldi r16,0x00
out PORTB,r16       ;se sting toate ledurile
cpi r17,0x01        ;primul nivel de comparatie, echiv. la 19,5mV
brlo end
ldi r16,0x01
out PORTB,r16

cpi r17,32          ;al doilea nivel, echiv. la 624mV
brlo end
ldi r16,0x03
out PORTB,r16

cpi r17,64          ;al treilea nivel, echiv. la 1248mV
brlo end
ldi r16,0x07
out PORTB,r16

cpi r17,96
brlo end
ldi r16,0x0F
out PORTB,r16

cpi r17,128
brlo end
ldi r16,0x1F
```

```
out PORTB,r16

cpi r17,160
brlo end
ldi r16,0x3F
out PORTB,r16

cpi r17,192
brlo end
ldi r16,0x7F
out PORTB,r16

cpi r17,224
brlo end
ldi r16,0xFF
out PORTB,r16
end:
out SREG,r20
reti
```

c) Să se scrie un program care converteşte la fiecare 2s valoarea tensiunii de pe pinul ADC6 şi afişează rezultatul conversiei pe modulul LCD. Mesajul afişat va avea pe prima linie textul „Tensiunea este:” iar pe a doua linie textul „*valoare*x4,9mV” unde *valoare* reprezintă cei 10 biţi ai rezultatului conversiei.



```
.include "m32def.inc"
.equ rs=PA2
.equ e=PD6
.equ ctrl=PORTD
.equ ctrl2=PORTA


jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp gata_conversia
jmp reset
jmp reset
jmp reset
jmp reset
reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,0b11110000
out DDRC,r16        ; pinii PC7...PC4 sunt iesiri
ldi r16,0b01000000
out DDRD,r16        ; pinul PD6 este iesire
ldi r16,0b00000100
out DDRA,r16        ; pinul PA2 este iesire
main:
cli
cbi ctrl,e
cbi ctrl2,rs
ldi r16,0b00000110  ;setare ADC: AREF este referinta, canal ADC6
out ADMUX,r16       ;rezultatul se aliniaza la dreapta
ldi r16,0b00101110  ;activare intrer., setare ceas ca fosc/64
out ADCSRA,r16            ;deocamdata nu s-a initiat conversia si nu s-a pornit modulul
in r16, SFIOR
andi r16,0b00011111       ;setarea modului de funct. pe semnal timer 16b prag B
ori r16,0b10100000        ;      fara a modifica alti biti
out SFIOR, r16


ldi r16,0b00000000  ;setare timer 16 biti: nu se utiliz. OC1A si OC1B, oprit deocamdata
out TCCR1A,r16            ;modul va fi CTC
ldi r16,0b00001000
out TCCR1B,r16
in r16, TIMSK
andi r16,0b11000011 ;nu se utiliz. nici o intrer., fara a modifica alti biti din TIMSK
out TIMSK, r16
ldi r16, 0x3D       ;se incarca valoarea de prag A: 0x3D0A=15626
out OCR1AH,r16            ;15626 * 1/(8MHz/1024) =~ 2s
ldi r16, 0x0A
out OCR1AL,r16
ldi r16, 0x3D       ;se incarca valoarea de prag B: 0x3D09=15625
out OCR1BH,r16            ;15625 * 1/(8MHz/1024) =~ 2s
ldi r16, 0x09
out OCR1BL,r16


ldi r16,0b00001000  ;setare timer0: nu se utiliz. pin OC0, timerul este oprit deocamdata
```

```
out TCCR0,r16       ;      modul va fi CTC cu prag dat de OCR0
in r16, TIMSK
andi r16,0b11111100 ;nu se utiliz. nici o intrer., fara a modifica alti biti din TIMSK
out TIMSK, r16

call init_display
ldi r17,0b10000000
call set_ram        ;prin r17 se seteaza adresa 0x00 pentru RAM de afisare
ldi r17,'T'
call put_char
ldi r17,'e'
call put_char
ldi r17,'n'
call put_char
ldi r17,'s'
call put_char
ldi r17,'i'
call put_char
ldi r17,'u'
call put_char
ldi r17,'n'
call put_char
ldi r17,'e'
call put_char
ldi r17,'a'
call put_char
ldi r17,' '
call put_char
ldi r17,'e'
call put_char
ldi r17,'s'
call put_char
ldi r17,'t'
call put_char
ldi r17,'e'
call put_char
ldi r17,':'
call put_char


ldi r17,0b11001010  ;prin r17 se seteaza adresa 0x4A pentru RAM de afisare
call set_ram

ldi r17,'x'
call put_char
ldi r17,'4'
call put_char
ldi r17,','
call put_char
ldi r17,'9'
call put_char
ldi r17,'m'
call put_char
ldi r17,'V'
call put_char

sei
sbi ADCSRA,ADEN           ;se porneste ADC
in r16,TCCR1B
andi r16,0b11111101       ;se porneste timerul 16b care va numara la 1024 imp. de ceas
ori r16,0b00000101
out TCCR1B,r16
bucla:
rjmp bucla

gata_conversia:
in r20,SREG
in r21,ADCL  ;se citeste rezultatul conversiei
in r22,ADCH
in r16,TIFR
ori r16,0b00001000  ;se reseteaza flagul timerului 16biti
```

```
out TIFR,r16
ldi r17,0b11000000  ;prin r17 se seteaza adresa 0x40 pentru RAM de afisare
call set_ram

ldi r17,'0'
sbrc r22,1          ;se trimite bitul 9 al conversiei
ldi r17,'1'
call put_char

ldi r17,'0'
sbrc r22,0          ;se trimite bitul 8 al conversiei
ldi r17,'1'
call put_char

ldi r18,0x09
bucla1:
ldi r17,'0'
dec r18
breq end
rol r21
brcc PC+2
ldi r17,'1'
call put_char
rjmp bucla1

end:
out SREG,r20
reti


init_display:
cbi ctrl2,rs
ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e

ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b10000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b11000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b00010000
out PORTC,r16
sbi ctrl,e
call wait_48us
```

```
cbi ctrl,e
call wait_30ms

ldi r16,0b00000000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
ldi r16,0b00100000
out PORTC,r16
sbi ctrl,e
call wait_48us
cbi ctrl,e
call wait_30ms
ret


set_ram:
cbi ctrl2,rs
mov r16,r17
andi r16,0xF0              ;se retine doar nibble (4 biti) superior
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
mov r16,r17
andi r16,0x0F              ;se retine doar nibble (4 biti) inferior
swap r16                   ;interschimba nibble (4 biti) sup. cu nibble (4 biti) inf.
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
call wait_48us
ret

put_char:
sbi ctrl2,rs
mov r16,r17
andi r16,0xF0              ; se retine doar nibble (4 biti) superior
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
mov r16,r17
andi r16,0x0F              ; se retine doar nibble (4 biti) inferior
swap r16                   ; interschimba nibble (4 biti) sup. cu nibble (4 biti) inf.
out PORTC,r16
sbi ctrl,e
nop
nop
cbi ctrl,e
call wait_48us
ret

wait_48us:
ldi r16,0x00
out TCNT0,r16
ldi r16, 0x06       ;se incarca valoarea de prag: 0x06=6
out OCR0,r16        ;6 * 1/(8MHz/64) =48us
in r16,TCCR0
andi r16,0b11111000        ;se porneste timerul si este setat sa numere
ori r16,0b00000011         ;      la fiecare 64 perioade de ceas, fara a modif. alti biti
out TCCR0,r16
wait:
in r16,TIFR
sbrs r16,OCF0       ;se asteapta atingerea pragului OCR0
rjmp wait
in r16,TIFR
```

```
ori r16,0b00000010
out TIFR,r16        ;se reseteaza flagul
in r16,TCCR0
andi r16,0b11111000 ;se opreste timerul
out TCCR0,r16
ret

wait_30ms:
ldi r16,0x00
out TCNT0,r16
ldi r16, 0xF0       ;se incarca valoarea de prag: 0xF0=240
out OCR0,r16        ;240 * 1/(8MHz/1024) =~ 30ms
in r16,TCCR0
andi r16,0b11111000       ;se porneste timerul si este setat sa numere
ori r16,0b00000101        ;      la fiecare 1024 per. de ceas, fara a modif. alti biti
out TCCR0,r16
wait1:
in r16,TIFR
sbrs r16,OCF0       ;se asteapta atingerea pragului OCR0
rjmp wait1
in r16,TIFR
ori r16,0b00000010
out TIFR,r16        ;se reseteaza flagul
in r16,TCCR0
andi r16,0b11111000 ;se opreste timerul
out TCCR0,r16
ret
```

Lucrarea 5
Utilizarea microcontrolerului ATmega32 pentru a comanda senzorul de temperatură DS1820 Prezentarea informaţiilor folosind un afişor LED multiplexat tip 7 segmente

Scopul lucrării

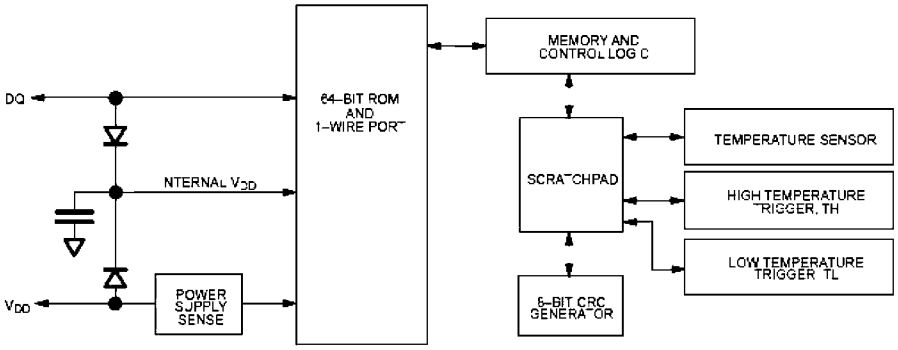
a) Prezentarea senzorului de temperatură DS1820 şi a modului în care se poate realiza comunicaţia cu acesta
b) Descrierea modului de utilizare a unui modul de afişare LED tip 7 segmente folosind tehnica multiplexării
c) Prezentarea unor opţiuni avansate de compilare oferite de mediul Atmel Studio
d) Programarea unei aplicaţii utilizând placa de dezvoltare EasyAVRv7

1. Senzorul de temperatură DS1820

DS1820 este un circuit integrat produs de firma Dallas Semiconductor cu funcţionare specializată pentru măsurarea temperaturilor mediului ambiant sau a suprafeţei cu care se află în contact. Dintre caracteristicile acestui circuit se pot menţiona:

- este disponibil şi în capsulă de mici dimensiuni tip TO92 cu numai 3 pini;
- interfaţa de comunicaţie se bazează pe un protocol proprietar „1-Wire" care necesită o conexiune de date cu un singur fir, în afară de masă, la care se pot conecta simultan mai multe dispozitive;
- poate fi telealimentat prin conexiunea de date;
- gamă de măsură cuprinsă între -55^{0}C şi +125^{0}C cu pas de 0,5^{0}C
- poate memora într-o memorie nevolatilă valori limită de temperatură la depăşirea cărora va genera automat stare de alarmă

Schema bloc a circuitului DS1820 este prezentată în figura de mai jos.



1.1. Modul de funcţionare

Conversia temperaturii se realizează după un mecanism hardware proprietar imediat după ce se primeşte o comandă de iniţiere a acesteia. La finalizarea conversiei, care durează aproximativ 200ms, rezultatul va fi reprezentat în formă binară pe un număr total de 16biţi.

Primul octet al rezultatului indică semnul conversiei. Astfel, valoarea FFh semnifică o temperatură negativă care va fi reprezentată în cel de-al doilea octet (cel mai puţin semnificativ) în complement faţă de doi. Dacă temperatura este pozitivă, octetul cel mai semnificativ va fi 00h şi valoarea acesteia va avea o reprezentare directă în cel de-al doilea octet.

Un exemplu de legătură între rezultatul conversiei şi valoarea temperaturii este prezentat în tabelul de mai jos.

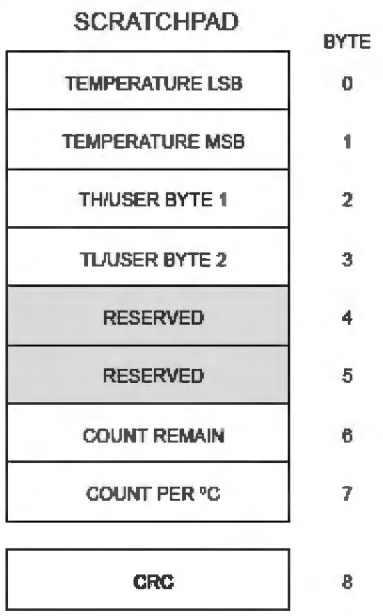
TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	00000000 11111010	00FA
+25°C	00000000 00110010	0032h
$+^1/_2$°C	00000000 00000001	0001h
+0°C	00000000 00000000	0000h
$-^1/_2$°C	11111111 11111111	FFFFh
–25°C	11111111 11001110	FFCEh
–55°C	11111111 10010010	FF92h

Indiferent de semnul temperaturii, doar biţii 1...7 din octetul cel mai puţin semnificativ reprezintă valoarea efectivă a conversiei, în timp ce bitul 0 este indicator pentru jumătate de grad. Astfel, dacă bitul 0 are valoarea 1 logic, temperatura se citeşte prin adunarea valorii $0,5^0$C la valoarea reprezentată de biţii 1...7. În caz contrar, temperatura se obţine direct din valoarea biţilor 1..7. Pentru temperaturile negative, mai întâi se face trecerea din complementul faţă de doi şi apoi se separă şi se interpretează biţii.

În tabelul de mai jos sunt exemplificate două cazuri de citire a temperaturii.

Rezultatul conversiei		Interpretare
Octet MS	Octet LS	
00000000	00010101	- MS este 00h prin urmare temperatura este pozitivă iar LS se citeşte direct - bitul 0 din LS este 1 deci se va adăuga $0,5^0$C - din LS se reţin doar biţii 1...7 adică 00001010 = 10_z - în final temperatura se interpretează ca $+10,5^0$C
11111111	11001111	- MS este FFh prin urmare temperatura este negativă iar LS se citeşte în complement faţă de doi - LS este în complement faţă de doi, scădem 1 şi complementăm obţinând 00110001 - bitul 0 din noul LS este 1 deci se va adăuga $0,5^0$C - din noul LS se reţin doar biţii 1...7 adică 00011000 = 24_z - în final temperatura se interpretează ca $-24,5^0$C

DS1820 dispune de o memorie RAM numită „scratchpad” cu dimensiunea de 9 octeţi. Rezultatul conversiei se memorează întotdeauna cu octetul LS la adresa 0 şi octetul MS la adresa 1, aşa cum se poate vedea în figura de mai jos.



1.2. Protocolul de comunicaţie

Comunicaţia între DS1820 şi alte dispozitive este de tip serial şi necesită o singură conexiune bidirecţională pe care se pot conecta mai multe echipamente. Dintre acestea, unul singur va avea rol de master şi toate celelalte vor fi slave. Dispozitivul master este cel care iniţiază comunicaţia iar dispozitivele slave răspund la comenzile primite, fiind identificate prin intermediul unei adrese unice astfel că la un moment dat comunicaţia se desfăşoară numai între master şi un singur slave, cel care a fost adresat.

Din punct de vedere hardware, protocolul necesită o rezistenţă de pull-up pe linia de date şi respectarea strictă a unor secvenţe de generare a următoarelor tipuri de semnale, privite din punctul de vedere al dispozitivului master:

- puls de reset la care dispozitivul slave răspunde prin aducerea liniei în 0 logic;
- bit 0 logic;
- bit 1 logic;
- citire bit de la slave

Transmiterea sau recepţionarea unui octet de către master se face prin compunerea semnalelor menţionate şi respectând regula „LSB first”. Secvenţa de generare standard a semnalelor de bază este descrisă în tabelul de mai jos.

Tip de semnal	Secvenţa de generare standard
Puls de reset	- se aduce linia de date la nivel 0 şi se aşteaptă între 480µs şi 640µs - se eliberează linia de date (i se permite să fie adusă la nivel 1 prin rezistenţa de pull-up) şi se aşteaptă între 70µs şi 78µs

	- se verifică nivelul liniei de date pentru a vedea dacă a răspuns cel puţin un dispozitiv (în acest caz linia va avea nivel 0) - se aşteaptă cel puţin 410µs
Bit 0 logic	- se aduce linia de date la nivel 0 şi se aşteaptă între 60µs şi 120µs - se eliberează linia de date (i se permite să fie adusă la nivel 1 prin rezistenţa de pull-up) şi se aşteaptă cel puţin 10µs
Bit 1 logic	- se aduce linia de date la nivel 0 şi se aşteaptă între 6µs şi 15µs - se eliberează linia de date (i se permite să fie adusă la nivel 1 prin rezistenţa de pull-up) şi se aşteaptă cel puţin 64µs
Citire bit de la slave	- se aduce linia de date la nivel 0 şi se aşteaptă între 6µs şi 15µs - se eliberează linia de date (i se permite să fie adusă la nivel 1 prin rezistenţa de pull-up) şi se aşteaptă între 9µs şi 12µs - se verifică nivelul liniei de date pentru a vedea cu ce valoare a răspuns dispozitivul slave - se aşteaptă cel puţin 55µs

Din punct de vedere software, orice comunicare se face printr-o tranzacţie în timpul căreia dispozitivul slave primeşte o serie de comenzi cunoscute. Orice tranzacţie va conţine în ordine câte o singură comandă din următoarele categorii:

- iniţializare;
- identificare;
- transfer date.

Comanda de iniţializare coincide cu generarea de către master a pulsului de reset.

Comenzile de identificare permit adresarea mai multor dispozitive care împart aceeaşi conexiune. În cadrul laboratorului nu se va utiliza decât comanda cu codul 0xCC care permite comunicarea cu un dispozitiv fără a-l adresa explicit. Acest lucru este posibil doar când există un singur dispozitiv slave astfel încât nu există riscul apariţiei conflictelor în comunicaţie.

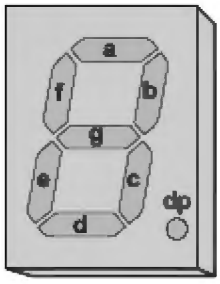
Comenzile de transfer de date reprezintă de fapt instrucţiunile pe care le poate executa circuitul DS1820. În cadrul laboratorului se va utiliza comanda 0x44 – iniţierea unei conversii şi comanda 0xBE – citire memorie scratchpad.

Trebuie precizat că în timp ce realizează o conversie, DS1820 va ţine linia de date în 0, urmând să o elibereze imediat după terminarea acesteia. De asemenea, după comanda 0xBE, DS1820 va transmite pe rând şi în ordine octeţii 0...8 ai memoriei scratchpad, până în momentul când primeşte comandă de iniţializare.

Placa EasyAVRv7 este echipată cu un senzor DS1820 a cărui linie de date poate fi conectată la pinul PA7 sau PB4.

2. Module de afişare LED tip 7 segmente

O unitate de afişare LED tip 7 segmente este formată din 7 leduri dispuse aşa cum se observă în figura de mai jos. De multe ori unităţile conţin şi un al optulea LED care este pe poziţia punctului zecimal. Întotdeauna unele dintre terminalele ledurilor, anodul sau catodul, vor fi conectate împreună astfel încât pentru a comanda o unitate sunt necesare 8 linii (segmentele a...g şi dp) + una (anodul sau catodul comun).



Atunci când mai multe unităţi sunt conectate şi comandate împreună, ele formează un modul de afişare.

Există două modalităţi fundamentale de a comanda un modul:

- direct;
- prin multiplexare.

Modul direct de conectare impune ca fiecare unitate să aibă propriile linii de comandă. Acest mod are avantajul simplităţii dar un dezavantaj major prin faptul că sunt necesare foarte multe linii de comandă (9/unitate).

Comanda prin multiplexare este puţin mai complexă dar necesită doar 8 linii de comandă plus câte o singură linie pentru fiecare unitate, care se conectează la anodul sau catodul comun al acesteia. Multiplexarea se bazează pe faptul că la o viteză suficient de mare (peste 30 de acţionări pe secundă), ochiul uman nu poate sesiza stingerea unui LED. Astfel, în loc să se aprindă simultan toate unităţile unui modul, ele se aprind pe rând, fiecare cu propria configuraţie. Selecţia se face pe baza anodului sau catodului comun pe care îl are fiecare unitate.

Placa EasyAVRv7 este echipată cu un modul LED tip 7 segmente conectate pentru comandă prin multiplexare. Cele 8 linii principale sunt comandate de portul C iar liniiile de selecţie a fiecărei unităţi sunt comandate de pinii PA0...PA3.

3. Opţiuni avansate de compilare oferite de mediul Atmel Studio

3.1. Macroinstrucţiuni

O macroinstrucţiune este reprezentată de un nume ales de utilizator şi conţine mai multe intstrucţiuni simple sau chiar alte macroinstrucţiuni. Atunci când compilatorul întâlneşte în cod numele unei macroinstrucţiuni, acesta va fi înlocuit în codul sursă cu toate instrucţiunile simple asociate. De aceea, programatorul trebuie să aibă grijă să nu utilizeze excesiv macroinstrucţiunile deoarece se poate ajunge în situaţia în care codul sursă devine prea mare şi depăşeşte capacitatea memoriei pentru cod a microcontrolerului.

Sintaxa unei macroinstrucţiuni este următoarea:

.MACRO nume
instrucţiune
instrucţiune
.ENDMACRO

3.2. Rezervarea de spaţiu în memorie şi tabele de căutare

Compilatorul mediului Atmel Studio oferă posibilitatea de a rezerva spaţiu pentru valori constante în memoria de cod sau în memoria EEPROM. Selecţia între cele două se face prin directivele .cseg şi .eseg respectiv.

Directivele utilizate pentru rezervarea şi iniţializarea memoriei sunt: .db – rezervă un octet, .dw – rezervă un cuvânt, .dd – rezervă un dublucuvânt şi .dq – rezervă 64 de biţi. Acestea sunt de obicei precedate de o etichetă şi urmate de o listă de valori, conform sintaxei:

eticheta: .db *listă_valori*

Valorile din listă pot fi exprimate în orice formă: binar, hexazecimal sau zecimal.

Rezervarea şi iniţializarea de spaţiu în memorie constituie elementul cheie pentru realizarea de tabele de căutare, utilizate frecvent atunci când sunt necesare diverse conversii de date.

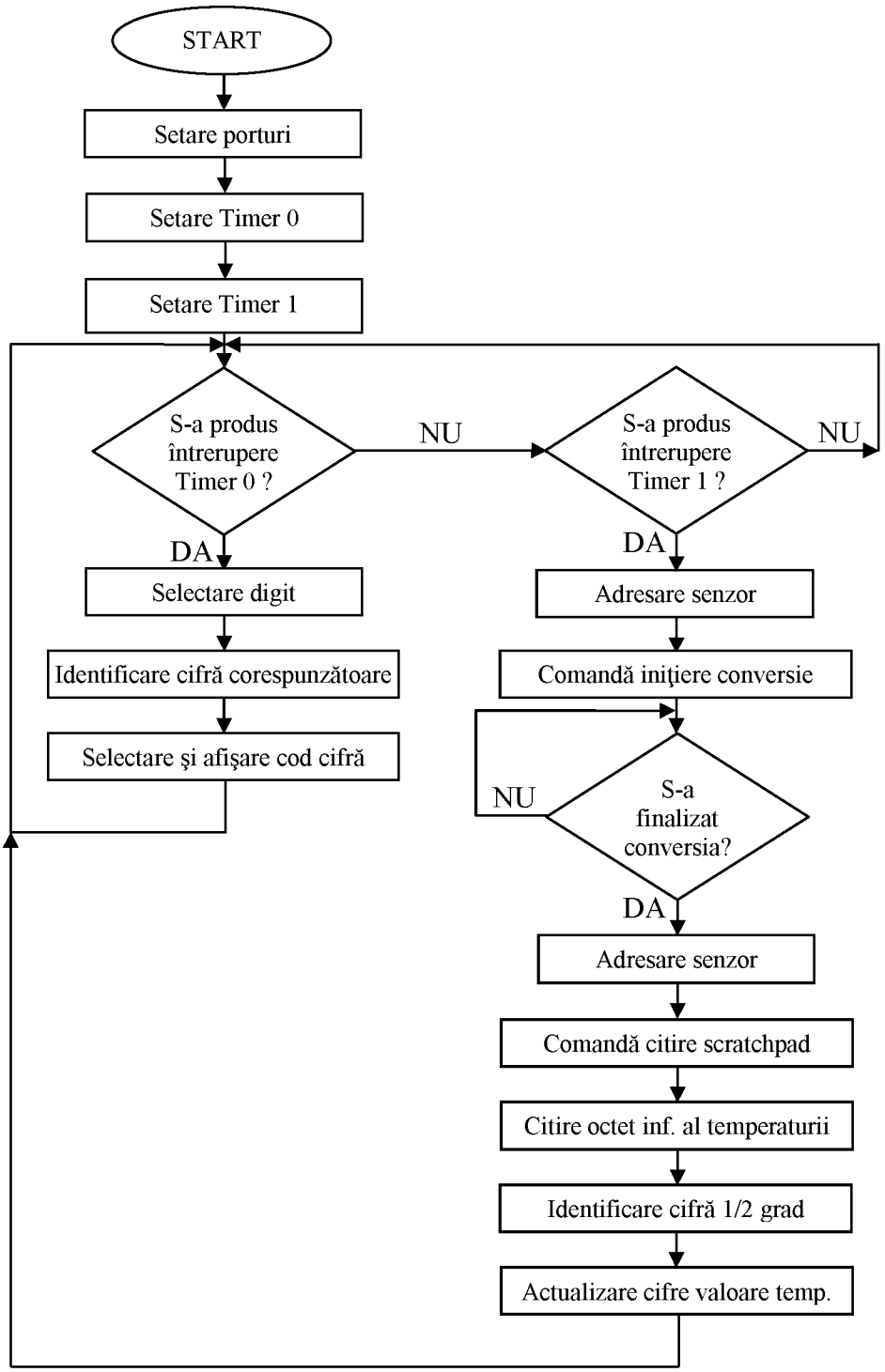
Pentru a extrage un element de 8 biţi aflat la poziţia *offset* într-o tabelă identificată prin eticheta *label*, se poate utiliza următoarea secvenţă de cod:

ldi ZH, high(2***label*)
ldi ZL, low(2***label*)
ldi *registru*, 0x00
ldi *registru1*, *offset*
add ZL, *registru1*
adc ZH, *registru*
lpm *registru*, Z

La finalul acestei secvenţe de cod elementul de la poziţia *offset* se va regăsi în registrul *registru*.

4. Exemple de programe

a) Să se scrie un program care la fiecare 2 secunde realizează o măsurare a temperaturii ambiante utilizând senzorul DS1820 şi afişează valoarea acesteia utilizând modulul de afişare LED 7 segmente de pe placa EasyAVRv7. Algoritmul de afişare va fi conceput numai pentru gama de temperaturi pozitive cuprinse între 0^0C şi 125^0C. Comunicaţia cu senzorul se face pe pinul PB4 şi este monitorizată astfel încât în cazul în care acesta nu răspunde la comenzi se va aprinde ledul PD0.



```
.include "m32def.inc"
.equ wireport = PORTB
.equ wirepin = PB4
.equ wirectrl = DDRB
.equ wirein = PINB

.def temp=r18
.def temp1=r19
.def temp2=r24
.def digit1=r0
.def digit2=r1
.def digit3=r2
.def digit4=r3
.def mux=r4

.MACRO _1us
       nop
       nop
       nop
       nop
       nop
       nop
       nop
       nop
.ENDMACRO

.MACRO wwrite_1
       sbi wirectrl,wirepin
       cbi wireport,wirepin
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       sbi wireport,wirepin
       ldi r16,50    //50 * 11 * 0,125us =~ 68,75us
       wwrite_1_loop1:
       _1us
       dec r16
       brne wwrite_1_loop1
.ENDMACRO

.MACRO wwrite_0
       sbi wirectrl,wirepin
       cbi wireport,wirepin
       ldi r16,50    //50 * 11 * 0,125us =~ 68,75us
       wwrite_0_loop1:
       _1us
       dec r16
       brne wwrite_0_loop1
       sbi wireport,wirepin
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
.ENDMACRO



jmp reset
jmp reset
jmp reset
jmp reset
```

```
jmp reset
jmp reset
jmp reset
jmp citire_temp
jmp reset
jmp reset
jmp refresh
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset
jmp reset

cifre: .DB 0b00111111, 0b00000110
       .DB 0b01011011, 0b01001111
        .DB 0b01100110, 0b01101101
        .DB 0b01111101, 0b00000111
        .DB 0b01111111, 0b01101111



reset:
ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16
in r16,SFIOR
ori r16,0b00000100
out SFIOR,r16
ldi r16,0xFF
out DDRC,r16 ; portul C este iesire
out DDRD,r16
ldi r16,0x0F
out DDRA,r16 ; pinii PA3...PA0 sunt iesiri
sbi wirectrl,wirepin
sbi wireport,wirepin
cbi PORTD,0
clr digit1
clr digit2
clr digit3
clr digit4
ldi r16,0b00001000
mov mux,r16
main:
cli
ldi r16,0b00001000  ;setare timer: nu se utiliz. pinul OC0, timerul este oprit deocamdata
out TCCR0,r16       ;modul va fi CTC cu prag dat de OCR0
in r16, TIMSK
andi r16,0b11111100 ;se utiliz. intrer. pt prag, fara a modifica alti biti din TIMSK
ori r16,0b00000010
out TIMSK, r16

ldi r16,0b00000000  ;setare timer 16 biti: nu se utiliz. OC1A si OC1B, oprit deocamdata
out TCCR1A,r16             ;modul va fi CTC
ldi r16,0b00001000
out TCCR1B,r16
in r16, TIMSK
andi r16,0b11000011 ;se utiliz. intrer. prag A, fara a modifica alti biti din TIMSK
ori r16,0b00010000
out TIMSK, r16
ldi r16, 0x3D       ;se incarca valoarea de prag A: 0x3D0A=15626
out OCR1AH,r16             ;15626 * 1/(8MHz/1024) =~ 2s
ldi r16, 0x0A
out OCR1AL,r16

in r16,TCCR1B
```

```
andi r16,0b11111101       ;se porneste timerul 16b care va numara la 1024 imp. de ceas
ori r16,0b00000101
out TCCR1B,r16

ldi r16,0x00
out TCNT0,r16
ldi r16, 0x7D       ;se incarca valoarea de prag: 0x7D=125
out OCR0,r16        ;125 * 1/(8MHz/64) = 1ms => f=1kHz
in r16,TCCR0
andi r16,0b11111000       ;se porneste timerul si este setat sa numere
ori r16,0b00000011        ;la fiecare 64 perioade de ceas, fara a modifica alti biti
out TCCR0,r16

sei
bucla:
rjmp bucla


refresh:
in r20,SREG
out PORTA,mux             ;se selecteaza digitul corespunzator
ldi r16,0b00001000
cp mux,r16                ;se incarca cifra corespunzatoare
brne PC+3                 ;      digitului selectat
mov r17,digit1
rjmp refresh_out
ldi r16,0b00000100
cp mux,r16
brne PC+3
mov r17,digit2
rjmp refresh_out
ldi r16,0b00000010
cp mux,r16
brne PC+3
mov r17,digit3
rjmp refresh_out
mov r17,digit4
refresh_out:
ldi ZH,high(2*cifre)      ;se identifica codul
ldi ZL,low(2*cifre)       ;      corespunzator cifrei
ldi temp1,0x00                  ;      alese
add ZL,r17
adc ZH,temp1
lpm temp1,Z
ldi r16,0b00000010        ;pentrul digitul 3
cp mux,r16                      ;      trebuie sa aprind si virgula
brne PC+2
ori temp1,0b10000000
out PORTC,temp1                 ;in temp1 este codul corespunzator cifrei selectate
lsr mux                               ;se pregateste selectia urmatorului digit
ldi r16,0x00
cp mux,r16
brne PC+3
ldi r16,0b00001000
mov mux,r16
out SREG,r20
reti

citire_temp:
in r20,SREG
call wire_reset
ldi temp,0xCC       ;comanda "skip ROM"
call wire_write     ;trimite la DS1820 continutul registrului temp
ldi temp,0x44       ;comanda de initiere conversie
call wire_write
cbi wirectrl,wirepin
temp_wait:
sbis wirein,wirepin       ;astept sa termine conversia
rjmp temp_wait
call wire_reset     ;a terminat conversia, voi citi rezultatul
ldi temp,0xCC
```

```
call wire_write
ldi temp,0xBE       ;comanda de citire scratchpad
call wire_write
call wire_read      ;se citeste in registrul temp primul octet din scratchpad
call wire_reset     ;      care este octetul inf al conversiei

ldi r17,0x00
lsr temp            ;se extrage in carry bitul indicator de 1/2 grad
brcc PC+2
ldi r17,0x05
mov digit4,r17      ;digitul 4 (cel mai din dreapta) afiseaza 1/2 grad

ldi r16,0x00
mov digit1,r16
mov digit2,r16
mov digit3,r16
temp_bucla1:        ;pe baza rezultatului conversiei
inc digit3          ;      se calculeaza cele 3 cifre zecimale
ldi r16,0x0A        ;      care se vor afisa
cp digit3,r16
brne bucla1_next
clr digit3
inc digit2
ldi r16,0x0A
cp digit2,r16
brne PC+3
clr digit2
inc digit1
bucla1_next:
dec temp
breq PC+2
rjmp temp_bucla1
out SREG,r20
reti

wire_reset:
sbi wirectrl,wirepin
cbi wireport,wirepin
cli
ldi r16,34                  //34 * 113 cicluri * 0,125us =~ 480us
ldi temp2,10
wr_loop1:
       _lus                 //8 cicluri
       dec temp2            //1 ciclu
       brne wr_loop1        //2 cicluri la salt inapoi; 1 ciclu altfel
       ldi temp2,10         //1 ciclu
       dec r16                     //1 ciclu
brne wr_loop1//2 cicluri la salt inapoi; 1 ciclu altfel
cbi wirectrl,wirepin        //2 cicluri
ldi r16,50          //70,5us
wr_loop2:
       _lus
       dec r16
brne wr_loop2
sbis wirein,wirepin //se verifica daca senzorul a raspuns
rjmp PC+3
sbi PORTD,0         //senzorul nu a raspuns,aprind un led
rjmp PC+2
cbi PORTD,0         //senzorul a raspuns,sting ledul

ldi r16,29          //29 * 113 cicluri * 0,125us =~ 410us
ldi temp2,10
wr_loop3:
       _lus         //8 cicluri
       dec temp2    //1 ciclu
       brne wr_loop3//2 cicluri la salt inapoi; 1 ciclu altfel
       ldi temp2,10 //1 ciclu
       dec r16                  //1 ciclu
brne wr_loop3       //2 cicluri la salt inapoi; 1 ciclu altfel
sei
ret
```

```
wire_write:
cli
ldi temp2,0x08
ww_send1:
       lsr temp
       brcs PC+2
       rjmp ww0
       wwrite_1
       rjmp ww_nextbit
       ww0:   wwrite_0
       ww_nextbit:  dec temp2
breq PC+2
rjmp ww_send1
sei
ret

wire_read:
cli
ldi temp2,0x08
wr_get1:
       sbi wirectrl,wirepin
       cbi wireport,wirepin
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       cbi wirectrl,wirepin
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       _1us
       sec           //1ciclu ; presupun ca am primit 1
       sbis wirein,wirepin
       clc
       ror temp      ;in temp se va memora octetul primit
       ldi r16,50    //50 * 11 * 0,125us =~ 68.75us
       wre_loop1:
       _1us
       dec r16
       brne wre_loop1
       dec temp2
breq PC+2
rjmp wr_get1
sei
ret
```