

1 Introducere în echipamentele de laborator

1.1 OBIECTIVE

Lucrarea prezintă echipamentele utilizate în cadrul laboratorului:

- Sursa de tensiune programabilă HAMEG HM8040-3;
- Sursa de tensiune programabilă HAMEG HM7042-5;
- Generatorul de funcții programabil HAMEG HM 8030-6;
- Multimetru digital programabil HAMEG HM8012;
- Frecvențmetru/Periodometru numeric 1.6GHz HAMEG HM8021-4;
- Osciloscop TEKTRONIX TDS 2024/TDS 3054.

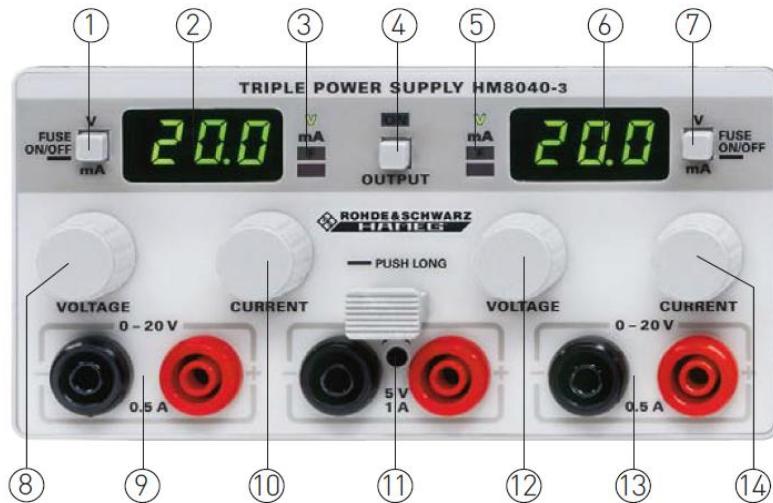
1.2 SURSA DE TENSIUNE PROGRAMABILĂ HAMEG HM8040-3

Sursa de tensiune stabilizată generează la ieșire o tensiune constantă, independentă de eventualele fluctuații ale: tensiunii de alimentare, sarcinii sau temperaturii. Sursa de tensiune se va folosi pentru alimentarea circuitelor studiate în cadrul laboratorului.

Caracteristicile sursei HM8040-3 sunt:

- Două surse de tensiune de ieșire reglabilă între 0 și 20V/0.5A și o sursă fixă 5V/1A;
- Rezoluție afișată 0.1V/1mA;
- Posibilitate de conectare în paralel sau serie;
- Buton pentru activarea/dezactivarea simultană a tuturor canalelor;
- Limitare ajustabilă pentru curent și siguranță electronică.

Panoul frontal al sursei HM8040-3 este prezentat în *Figura 1.1*.

**Figura 1.1 Panoul frontal HM8040-3****(1) & (7) - V/mA/Siguranță electronică**

Butoanele (1) (afişaj stânga) şi (7) (afişaj dreapta) sunt butoane cu ajutorul cărora se poate selecta afişarea tensiunii sau curentului și pentru activarea siguranței electronice individual pentru fiecare parte. Curentul este indicat cu o rezoluție de 1mA, iar tensiunea este afişată cu o rezoluție de 0.1V. Schimbarea între afişajele curentului și tensiunii se face printr-o apăsare scurtă a butonului ce are ca efect schimbarea indicatorilor (3) și (5), iar pentru o apăsare lungă activează siguranța electronică, semnalată prin aprinderea indicatorului F din indicatorii (3) și (5).

(2) & (6) – Afisaj tensiune/curent

Cele două afişaje cu 3 daci, 7 segmente oferă afişarea selectabilă a tensiunii de ieșire sau a curentului de ieșire. Afişajul din stânga indică tensiunea sau curentul pentru terminalele de ieșire din partea stângă (9), iar afişajul din partea dreaptă indică parametrii pentru terminalele de ieșire din partea dreaptă (13).

(3) & (5) - LED

V/mA - LED-uri pentru mărimea afişată;

F - LED ce semnalează activarea siguranței electronice;

I_{max} - LED ce semnalează depășirea limitei impuse pentru curent. Dacă siguranța electronică este activată, ieșirea sursei de tensiune se va închide când se detectează depășirea limitei de curent.

(4) – Activare/dezactivare ieșire sursă

Comanda ieșirii DC activează simultan toate cele 3 ieșiri DC (buton apăsat la HM8040-3). Afişajele de tensiune vor indica tensiunea de ieșire, chiar și atunci când LED-ul pentru ieșire indică faptul că ieșirea este deconectată.

(8) & (12) - Ajustare tensiune

Butoanele rotative pentru ajustarea tensiunii sunt folosite pentru varierea tensiunii în domeniul 0-20V. Butonul rotativ (8) din stânga setează sursa din stânga, iar butonul rotativ (12) din dreapta setează sursa din dreapta.

(9) & (13) - Ieșire 0-20V

Terminalele de ieșire pentru sursele 0-20V constau din două mufe banană mamă la care se pot conecta fire sau mufe banană tată. Circuitul electronic asigură protecția împotriva scurtcircuitului.

(10) & (14) – Ajustare limită curent

Butoanele rotative pentru ajustarea limitării de curent pentru ieșirea din partea stângă (10) și ieșirea din partea dreaptă (14). Domeniul de reglare este între 0-0.5A.

(11) - Ieșire 5V

Terminalele de ieșire pentru sursa de tensiune de +5V constau din două mufe banană mamă de 4mm la care se pot conecta fire sau mufe banană tată. Circuitul electronic asigură protecție împotriva scurtcircuitului. Un orificiu de acces (aflat deasupra, între cele două terminale de 5V) permite un reglaj fin între 4.5V - 5.5V.

1.3 SURSA DE TENSIUNE PROGRAMABILĂ HAMEG HM7042-5

Hameg HM7042-5 este o sursă de tensiune triplă, programabilă.

Caracteristici:

- Trei surse de tensiune de ieșire independente programabile:
2x0-32V, 2A; 1x2.7-5.5V, 5A;
 - Rezoluție afișată:
10 mV/1 mA pe canalul 1+3; 10 mV/10 mA pe canalul 2 ;
 - Posibilitate de conectare în paralel(pana la 9A) sau serie(pana la 69.5V);
 - Buton pentru activarea/dezactivarea simultană a tuturor canalelor;
 - Limitare ajustabilă pentru curent și siguranță electronică.
- ‘ Panoul frontal al sursei HM7042-5 este prezentat în *Figura 1.2*.

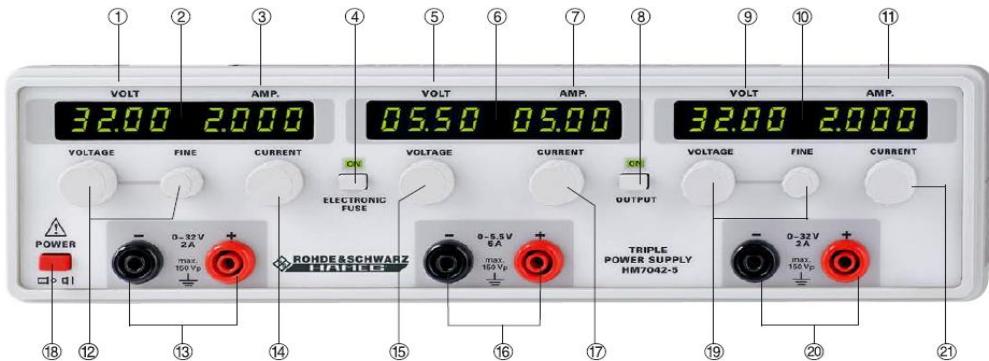


Figura 1.2 Panoul frontal HM7042-5

(1) & (5) & (9) - Afișaj tensiune

Cele trei afișaje cu 4 dígiți, 7 segmente oferă afișarea selectabilă a tensiunii de ieșire.

(2) & (6) & (10) - LED

LED = indicator limită de curent (se aprind când limita de curent este atinsă).

(3) & (7) & (11) - Afișaj curent

Cele trei afișaje cu 4 dígiți, 7 segmente oferă afișarea selectabilă a curentului de ieșire.

(4) - Siguranță Electronică

Acest buton va activa siguranța electronică, starea fiind indicată de LED-ul adiacent.

(8) – Activare/dezactivare ieșire

Activează/dezactivează simultan toate cele trei ieșiri DC, starea fiind indicată de LED-ul adiacent.

(12) & (19) - Ajustarea tensiunii

Butoanele rotative pentru ajustarea grosieră/fină a tensiunii sunt folosite pentru varierea tensiunii în domeniul 0-32V.

(13) & (20) - 0 – 32V / 2A

Ieșiri, conectori de 4 mm.

(14) & (17) & (21) - Limită curent

Butoanele rotative pentru ajustarea limitării de curent pentru ieșirea din partea stângă (14), ieșirea centrală (17) și ieșirea din partea dreaptă (21). În cazul în care „Limită curent” a fost selectată, LED-urile (2) și (10) se vor aprinde, iar tensiunea va scădea la zero.

(15) - Ajustare tensiune

Butoanele rotative pentru ajustarea tensiunii sunt folosite pentru varierea tensiunii în domeniul 0-5.5V.

(16) - 0 – 5.5V / 5A

Ieșiri, conectori de 4 mm.

(18) - ON/OFF

Buton pentru deschiderea/inchiderea sursei de tensiune.

Limitarea curentului

După pornire, sursa de alimentare nu va avea ultimele setări făcute înainte de deconectarea ei, butoanele pentru reglaj putând fi modificate cu sursa deconectată. Pentru protejarea componentelor alimentate de la sursă, este obligatorie activarea siguranței electronice. Utilizând butoanele 14,17,21 curentul maxim I_{max} poate fi setat pentru fiecare din cele 3 canale. Limitarea curentului pe un canal nu va influența pe celelalte. În caz de atingere a limitei curentului, se aprinde LED-ul corespunzător canalului utilizat ((2),(6) sau (10)).

Siguranță electronică

Înainte de selectarea acestui mod, limitele curentului trebuie să fie stabilite folosind butoanele 14,17,21. După setarea I_{max} , se apasă butonul 4 (ELECTRONIC FUSE), LED-ul [ON] se va aprinde indicând faptul că HM7042-5 este în modul Siguranță Electronică. În acest mod, toate ieșirile vor fi dezactivate imediat dacă pe un canal se ajunge la I_{max} .

1.4 GENERATORUL DE FUNCȚII PROGRAMABIL HAMEG HM8030-6

Generatorul de funcții este un aparat electronic ce furnizează semnale variabile de diferite forme (sinus, dreptunghi, triunghi, impuls, etc.), permitând modificarea după dorință a unor parametri: amplitudine, frecvență, factor de umplere, formă. Generatorul se folosește la aplicarea de semnale variabile în circuitele electronice, care se studiază experimental.

Panoul frontal al generatorului HM8030-6 este prezentat în Figura 1.3.

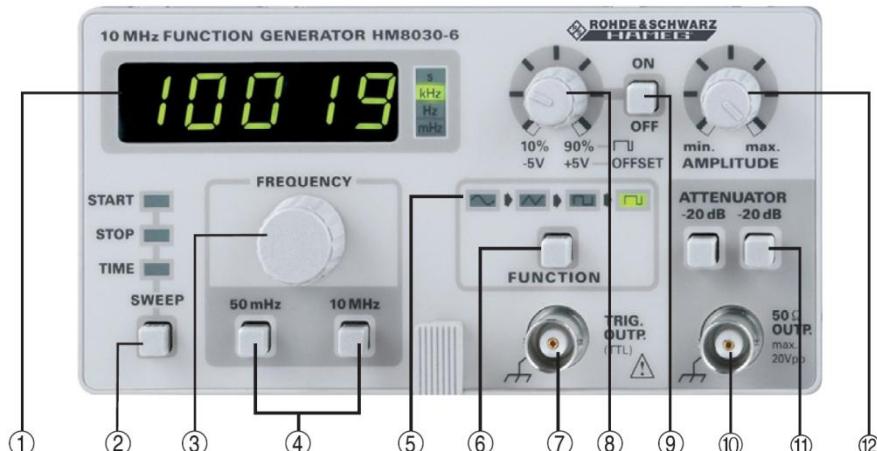


Figura 1.3 Panoul frontal HM8030-6

Generatorul de funcții HM8030-6 conține:

- partea de reglare a frecvenței semnalului generat (FREQUENCY), cu două tipuri de reglaje: în trepte (butoanele 50mHz și 10MHz (4) – ce micș orează, respectiv, măresc ordinul măsurătorii) și un reglaj fin (un potențiometru (3));
- partea de reglare a amplitudinii semnalului generat (AMPLITUDE), cu două tipuri de reglaje: brut (se modifică valoarea atenuării introdusă de aparat – două butoane de atenuare ”-20dB” fiecare (11)) și un reglaj fin (un potențiometru (12)). De asemenea, se poate regla componenta continuă a semnalului generat (OFFSET (8)) prin apăsarea butonului ON/OFF (9) și acționarea potențiometrului (8);
- un buton pentru selectarea formei semnalului de ieșire (FUNCTION (6)) indicată de LED-urile (5);
- două ieșiri de semnal: una pentru semnalul dorit de utilizator (10), având forma de undă și valorile reglate pentru frecvență și amplitudine (de ex. un semnal sinusoidal cu frecvență 20Hz și amplitudinea 4V) și o ieșire pentru semnal de sincronizare TTL (7).

1.5 MULTIMETRU DIGITAL PROGRAMABIL HAMEG HM8012

Un multimetru este un instrument de măsurare electronic care combină mai multe funcții de măsurare într-o singură unitate. Un multimetru include caracteristici de bază, cum ar fi: capacitatea de a măsura tensiunea, curentul și rezistența. Multimetrele digitale (DMM, DVOM) afișează valoarea măsurată în cifre.

Caracteristici ale HAMEG HM8012:

- Display 4 ¾ digiti 50000 unități;
- 42 domenii de măsurare;
- Selectare domeniu automat sau manual;
- Între 3 și 6 măsurători pe secundă;
- Precizie de 0.05%;
- Rezoluții: 10µV, 10nA, 10mΩ, 0,01 dBm și 0,1°.



Figura 1.4 Panou frontal HM8012

(1) - Afisaj

Afisajul digital indică valoarea de măsurare cu o rezoluție de 4 ¾ cifre, unde cea mai mare cifră folosită este "5". Acesta va afișa, de asemenea, diverse mesaje de avertizare. Valoarea de măsurare va fi afișată cu puncte zecimale și polaritate.

(2) – Martor continuitate

(3) - Activare/dezactivare martor pentru continuitate

Activează/dezactivează semnalul acustic pentru funcția de continuitate.

(4) & (5) & (7) & (9) - Conectori de 4mm

4 – conector pentru măsurarea a maximum 10 A în conjuncție cu intrarea COM (7);

5 - conector pentru măsurarea a maximum 500 mA în conjuncție cu intrarea COM (7);

7 – conector comun pentru toate măsurătorile care sunt apropiate cu masa (0V) cantitatea măsurată.

9 – conector pentru măsurarea voltajelor, rezistențelor, temperaturilor și jonctiunilor de diode în conjuncție cu intrarea COM (7).

(6) – HOLD (LED)

LED ce indică faptul că valoarea indicată de afişaj este blocată. Această funcție este activată de butonul (10).

(8) – OFFSET (LED)

LED ce indică faptul că afişajul arată o măsurătoare relativă. Valoarea indicată reprezintă diferența între valoarea de la intrare și cea prezentă la activarea modului OFFSET. Activarea acestei funcții se face prin apăsarea butonului (10).

(11) & (12) - Domeniu

Butoane pentru schimbarea domeniului de măsurare.

(15) - Auto-Range

Buton pentru activarea/dezactivarea funcției de auto-range.

(16) - Unitatea de măsurare

Această zonă afișează unitatea de măsură pentru funcția de măsurare selectată.

(17) - AC/DC

Selectare pentru măsurare în DC sau AC.

(18) & (19) - Functie măsurare

Selectarea funcției de măsurare.

1.6 FRECVENTMETRU NUMERIC 1.6 GHZ HAMEG HM8021-4

Cu ajutorul acestui frecvențmetru, se pot măsura frecvențe de până la 1.6 GHz.

Caracteristici:

- Domeniu de până la 1.6GHz;
- Senzitivitate 20 mV;
- Funcții de măsurare;
- Trei perioade de eșantionare;
- Auto-trigger.

Panoul frontal al multimetrului HM8012 este prezentat în *Figura 1.5*.

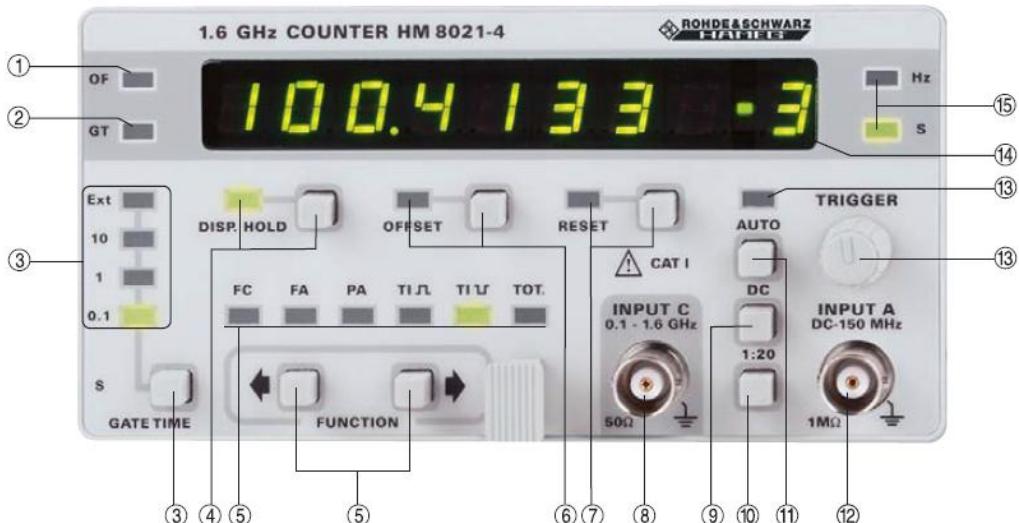


Figura 1.5 Panou frontal HM 8021-4

(1) - OF (LED)

LED-ul este aprins când apare o depășire (overflow). Aceasta depinde de perioada de eșantionare.

(2) - GT (GATE OPEN, LED)

Indicatorul GT este aprins atunci când intrarea este deschisă pentru măsurători.

(3) - Gate Time

Selectează perioada de eșantionare (0.1s, 1s, 10s).

(4) - Hold

Pe afișaj rămâne ultima valoare măsurată până la apăsarea butonului **RESET**.

(5) - Function

Martorii LED indică funcția de măsurare selectată cu ajutorul butoanelor.

(6) - Offset

La apăsarea butonului, valoarea afișată de display devine nouă valoare de referință.

(7) - Reset

Buton RESET. Resetul este activ atât timp cât butonul este apăsat.

(8) - Input C

Gama de frecvențe: 100 MHz - 1,6 GHz.

Impedanță de intrare: 50Ω .

(9) - DC

Selectarea semnalului de intrare.

(10) - 1:20

Selectează atenuarea semnalului de intrare. Semnalul de intrare este atenuat cu 26 dB.

(11) - Auto

Declanșează semnalul de intrare.

(12) - Input A

Sensibilitate semnal de intrare: 20mV - 80 MHz și 60mV - 150MHz.

(13) - Trigger Level

Potențiometru de ajustare a trigger-ului. LED-ul martor clipește atunci când trigger-ul este corect.

(14) - Afisaj 8 digitii

(15) - Hz (LED): unitatea de măsură a frecvenței;

Sec(LED): unitatea de măsură a timpului.

1.7 OSCILOSCOP TEKTRONIX TDS 2024/TDS 3054

Osciloscopul este un aparat electronic de măsură care servește la observarea și măsurarea unui semnal de tensiune electrică cu variație (frecvență) constantă, sau a mai multor semnale simultane de tensiune ce evoluează discret, folosind pentru asta în mod uzual un cimp grafic vizualizator (ecran), unde axa 'X' (abscisa) este axa timpului, iar axa 'Y' (ordonata) este axa reprezentării amplitudinilor semnalelor de măsurat (observat). Imaginele obținute pe ecran se numesc oscilogramme.

Osciloscopul se utilizează pentru:

- vizualizarea variației în timp a tensiunilor electrice, precum și măsurarea parametrilor acestora: valoare vârf la vârf, amplitudine, valoarea componentei continue, perioada (frecvența);
- vizualizarea relației dintre două tensiuni variabile în timp, putând determina raportul frecvențelor tensiunilor și defazajul dintre ele;
- trasarea curbelor caracteristice ale unor dispozitive sau materiale (caracteristici statice ale unor dispozitive sau circuite electronice, ciclu de histerezis al materialelor feromagnetice, etc.).

În laborator se folosesc osciloscoapele TEKTRONIX TDS 2024 și TEKTRONIX TDS 3054. Deși au denumiri diferite, principiul de funcționare este identic. Panoul frontal al osciloscopului TDS 2024 este prezentat în *Figura 1.6*.

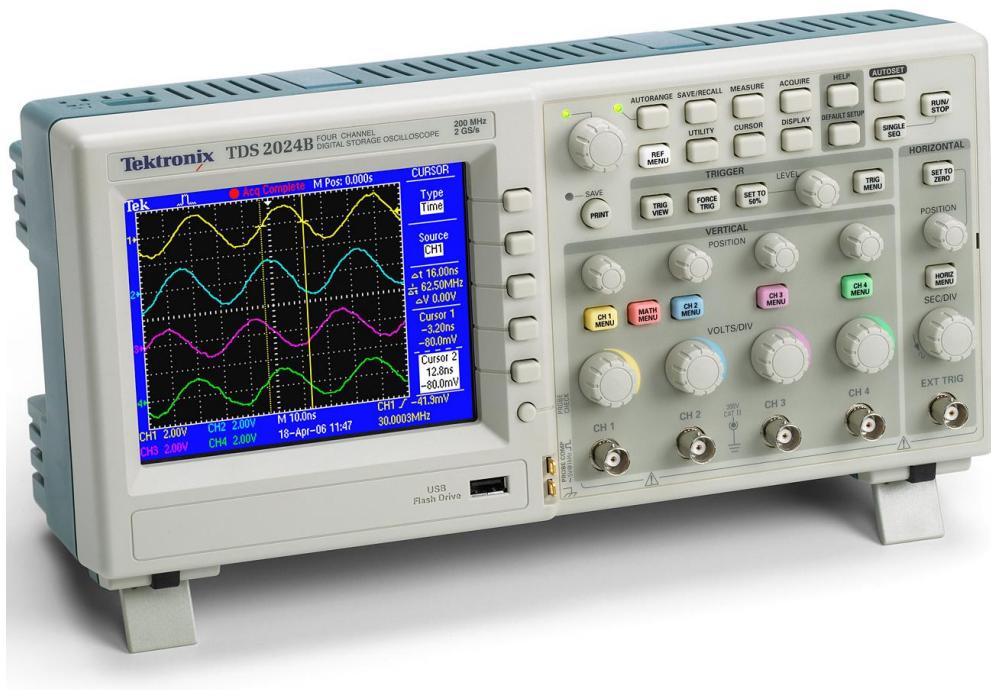


Figura 1.6 Panou frontal Tektronix TDS 2024

Osciloscopul TEKTRONIX TDS 2024 permite vizualizarea alternativă sau concomitentă a patru semnale de tensiune, având patru canale pentru aplicarea acestora (CH 1, CH 2, CH 3 și CH 4).

Fiecare canal are:

- bornă de intrare semnal;
- un potențiometru de modificare a scării de reprezentare pe verticală (VOLTS/DIV.);
- câte un potențiometru pentru poziționarea axelor X (X-POS) și Y (Y-POS).

2 Introducere în AVR Studio 4.19 și IAR Embedded Workbench (5.51 – 6.11). Realizarea unui proiect

2.1 INTRODUCERE ÎN AVR STUDIO 4.19

AVR Studio 4 este un mediu de dezvoltare gratuit ce permite realizarea de programe, încărcarea și depanarea lor pe microcontrolerle produse de către Atmel. La pornirea programului apare o listă din cadrul căreia se poate alege microcontrolerul cu care se lucrează, după care, dacă este conectat corespunzător la PC, are loc încărcarea propriu zisă a proiectului.

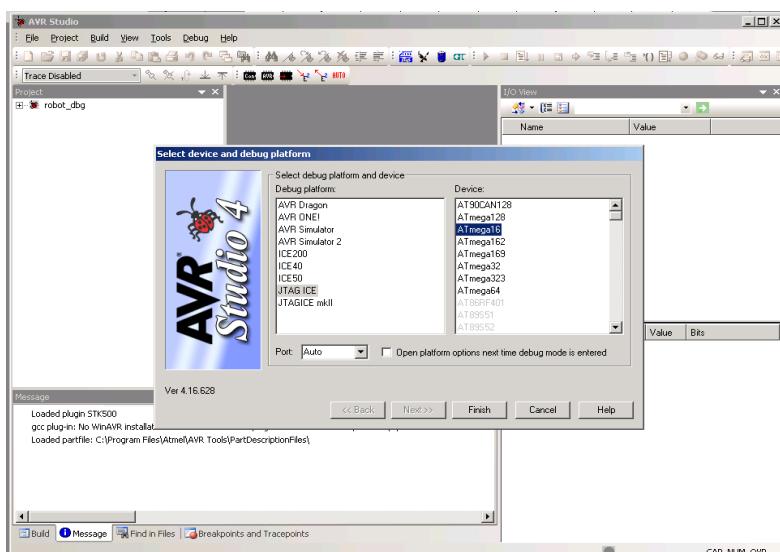


Figura 2.1 Pagina Start AVR Studio 4

2.2 AVR JTAG ICE

AVR JTAG ICE este un emulator puternic pentru toate microcontroler-ele AVR de 8 biți cu interfață JTAG IEEE 1149.1, cu suport de emulare inclus în cip (on-chip debugging). Practic, această facilitate este disponibilă pe noile ATmega (ATmega128, ATmega8, ATmega16, etc.). JTAG ICE și depanatorul software AVR Studio oferă utilizatorului control asupra resurselor interne ale microcontroler-ului, ajutând la reducerea timpului de dezvoltare și făcând depanarea mai ușoară. JTAG ICE realizează emularea microcontroler-ului în timp real, în timp ce se rulează aplicația în sistemul său. JTAG ICE oferă posibilitatea de emulare la jumătate de cost în comparație cu emulatoarele tradiționale.

Caracteristici:

- Interfață software cu AVR Studio 3.52 sau versiuni mai noi;
- Emulează toate funcțiile în cip, atât digitale cât și analogice;
- Se poate insera Breakpoint chiar când se modifică programul, fără a fi necesară recompilarea;
- Interfață spre PC: RS-232;
- Suport pentru Breakpoint pe program și pe date;
- Depanare pe cod sursă sau pe cod C;
- Programare în sistem prin interfață JTAG;
- Este alimentat direct de pe placă de bază sau printr-o sursă externă de 9-15Vcc.

JTAG ICE permite accesul către toate caracteristicile puternice oferite de microcontroler-ul AVR. Toate resursele AVR pot fi monitorizate: memoria Flash, memoria EEPROM, memoria SRAM, setul de registri, numărătoarele, biți de siguranță, biți de blocare și toate modulele I/O. JTAG ICE oferă de asemenea suport de depanare extensiv pentru condiții de întrerupere, inclusiv întrerupere în timp ce se modifică programul (break on change) fără a fi necesară recompilarea, condiții de întrerupere ale memoriei program de unică adresă sau frecvență de adresă și condiții de întrerupere ale memoriei de date de unică adresă sau frecvență de adresă.

JTAG ICE susține următoarele microcontrolere AVR:

- Atmega323;
- Atmega16;
- Atmega32 (disponibil în Q2 2002);
- Atmega162 (disponibil în Q1 2001).

JTAG ICE va fi actualizat automat prin versiuni viitoare ale AVR Studio pentru a susține viitoarele dispozitive cu suport JTAG pe măsură ce acestea vor fi scoase pe piață. Interfața JTAG este integrată în AVR Studio. Toate fazele dezvoltării AVR pot fi făcute în acest mediu de dezvoltare integrată.

2.3 INFORMAȚII GENERALE DESPRE ATMEGA16

ATmega16 este un microcontroler CMOS de 8 biți realizat de firma Atmel, cu un consum de curent mic, bazat pe arhitectura RISC AVR îmbunătățită.

Dispune de un set de 131 instrucțiuni și 32 de registre de uz general. Cele 32 de registre sunt direct adresabile de Unitatea Logică Aritmetică (ALU) permitând accesarea a două registre independente într-o singură instrucțiune. Se obține astfel o eficiență sporită în execuție (de până la zece ori mai rapide decât microcontroler-ele convenționale CISC).

Caracteristicile acestuia sunt:

- 16KB de memorie Flash reînscriptibilă pentru stocarea programelor;
- 1KB de memorie RAM;
- 512B de memorie EEPROM;
- două numărătoare/temporizatoare de 8 biți;
- un numărător/temporizator de 16 biți;
- conține un convertor analog-digital de 10 biți, cu intrări multiple;
- conține un comparator analogic;
- conține un modul USART pentru comunicație serială (port serial);
- dispune de un cronometru cu oscilator intern;
- oferă 32 de linii I/O organizate în patru porturi (PA, PB, PC, PD).

Structura internă generală a controller-ului este prezentată în *Figura 2.2*. Se poate observa că există o magistrală generală de date la care sunt conectate mai multe module:

- unitatea aritmetică și logică (ALU);
- registrele generale;
- memoria RAM și memoria EEPROM;
- liniile de intrare (porturile – linii I/O) și celelalte blocuri de intrare/ieșire. 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 instrucțiunilor, decodare și execuție, comunică printr-o magistrală proprie, separată de magistrala de date menționată mai sus. Acest tip de organizare este conform principiilor unei arhitecturi Harvard și permite controler-ului să execute instrucțiunile foarte rapid.

Modul *POWER-DOWN* salvează conținutul regiszrelor, dar blochează oscilatorul, dezactivând toate celelalte funcții ale chip-ului pana la următoarea întrerupere externă sau reset hardware. În modul *POWER-SAVE*, timer-ul asincron continuă să meargă, permitând utilizatorului să mențină o bază de timp, în timp ce restul dispozitivului este oprit.

În modul *STANDBY*, oscilatorul funcționează în timp ce restul dispozitivului este oprit. Acest lucru permite un start foarte rapid combinat cu un consum redus de energie. În modul *STANDBY EXTINS* (Extended Standby Mode), atât oscilatorul principal cât și timer-ul asincron continuă să funcționeze.

Memoria *Flash* (On-chip) permite să fie reprogramată printr-o interfață serială SPI, de către un programator de memorie nevolatilă convențional, sau de către un program de boot On-chip ce rulează pe baza AVR. Programul de boot poate folosi orice interfață să încarce programul aplicație în memoria Flash de aplicație. Combinând un CPU RISC de 8 biți cu un Flash In-system auto-programabil pe un chip monolitic, ATmega 16 este un microcontroler

puternic ce oferă o soluție extrem de flexibilă și cu un cost redus în comparație cu multe altele de pe piață. ATmega16 AVR este susținut de o serie completă de instrumente de program și de dezvoltare a sistemului, care include: compilatoare C, macroasamblare, programe depanare/simulare etc.

Diagrama bloc :

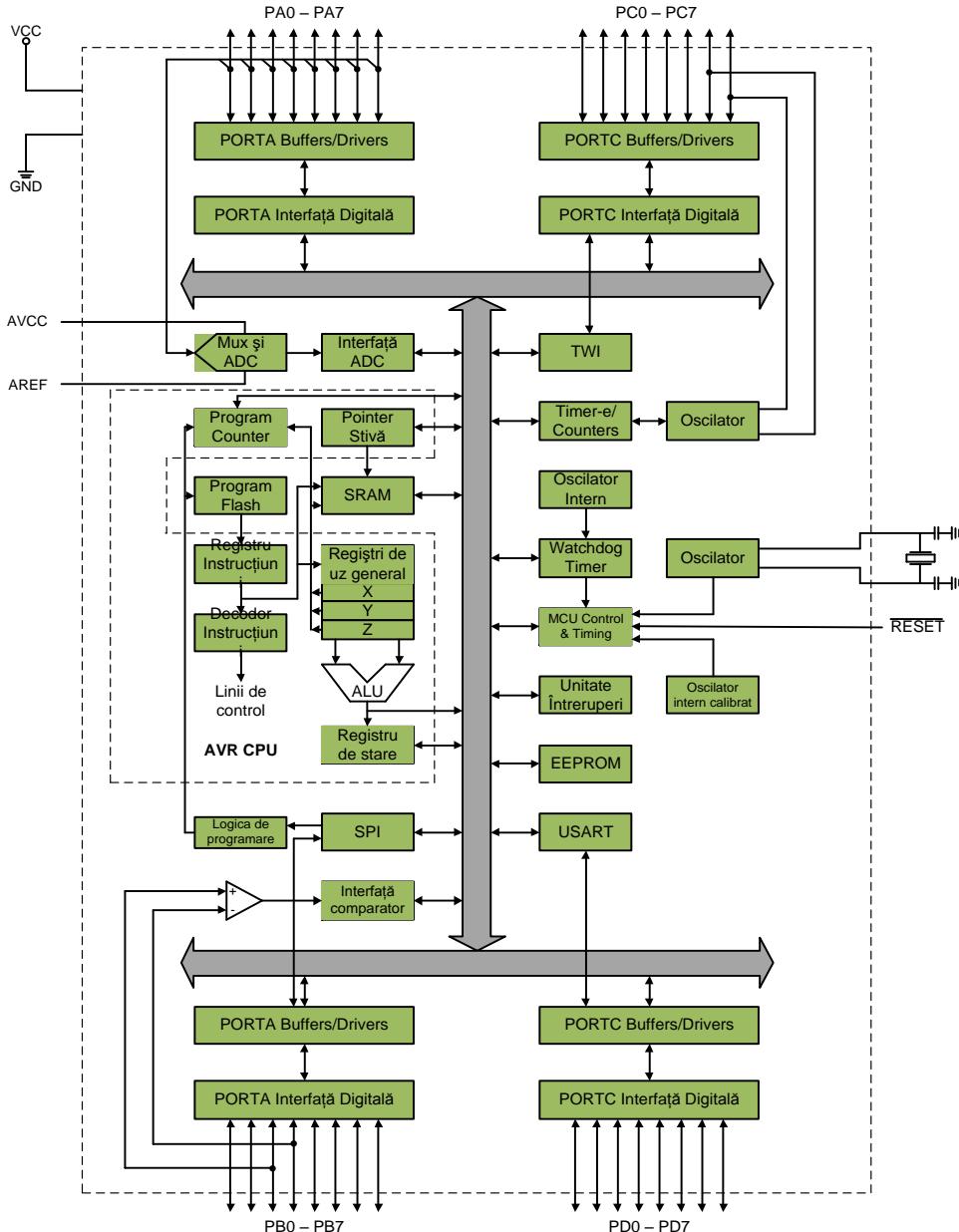


Figura 2.2 Diagrama bloc ATmega16

2.4 INTRODUCERE ÎN IAR EMBEDDED WORKBENCH 5.51

2.4.1 PERSPECTIVĂ ASUPRA LIMBAJULUI IAR

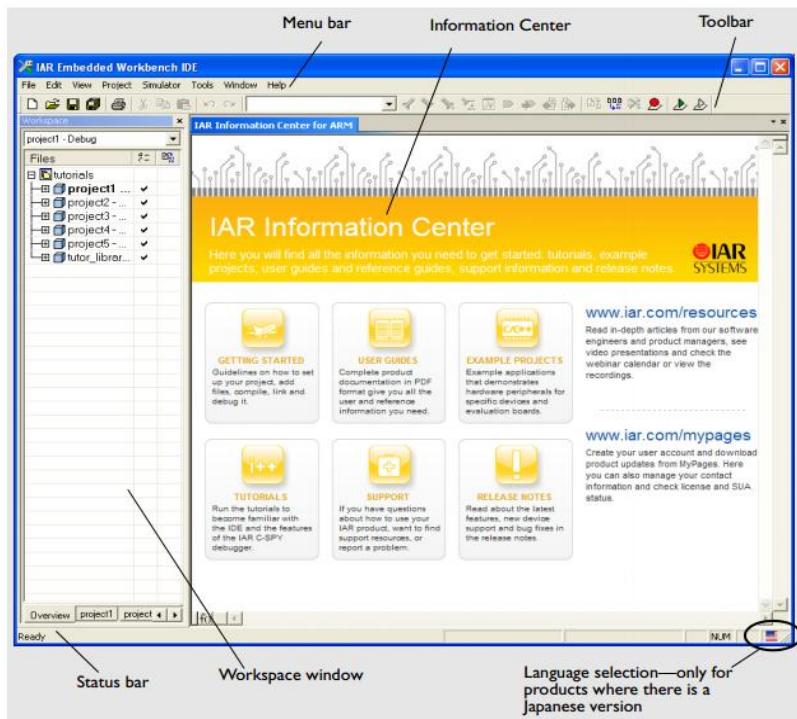


Figura 2.3 Pagina de start

Există două limbi de programare de nivel înalt disponibile cu compilatorul AVR®IAR C/C++ :

1. C, cel mai răspândit limbaj de nivel înalt de programare folosit în industria de sisteme încorporate. Folosind compilatorul AVR®IAR se pot construi aplicații de sine stătătoare ce urmează standardul ISO 9899:1990. Acest standard este cunoscut ca ANSI C.
2. C++, un limbaj modern orientat obiect, cu o librărie ce dispune de toate caracteristicile necesare pentru o programare modulară. Sistemele IAR suportă două nivele ale limbajului C++:
 - Embedded C++ (EC++) este un subset al standardului de programare C++ destinat programării sistemelor încorporate. Este definit de un consorțiu industrial, Embedded C++ Technical Comitee.
 - IAR Extended EC++, cu caracteristici suplimentare cum ar fi suportul total pentru şabloane, suportul pentru spaţiile de nume, operatorii de cast, precum şi Standard Template Library (STL).

Fiecare din cele două limbi de programare suportate pot fi folosite fie într-un mod strict, fie în unul relaxat, fie în unul relaxat cu extensiile IAR activate. Modul strict aderă la standard, pe când modul relaxat permite anumite deviații de la acest standard. Este de asemenea posibil ca anumite părți ale aplicației să fie implementate în limbaj de asamblare.

- **Construirea de aplicații – ansamblu**

O aplicație tipică este construită din fișiere sursă și librării. Fișierele sursă pot fi scrise în C, C++ sau limbaj de asamblare și pot fi compilate în fișiere obiect de către compilatorul AVR@IAR sau AVR@IAR assembler.

O librărie este o colecție de fișiere obiect. Un exemplu de librărie tipică este librăria compilatorului ce conține mediul de rulare și librăria standard C/C++. Librăriile pot fi de asemenea construite folosind IAR XAR Library Builder, IAR XLIB Librarian sau să fie oferite de furnizori externi.

Link-editor-ul IAR XLINK este folosit pentru a construi aplicația finală. XLINK folosește, în mod normal, un fișier de comandă pentru link-editare.

Compilarea: În interfața linei de comandă, linia următoare compilează fișierul sursă `myfile.c` în fișierul obiect `myfile.r90`, folosind setările implicate.

`Iccavr myfile.c`

În plus se pot specifica câteva opțiuni critice.

Linkeditarea: Linkeditorul IAR XLINK este folosit pentru a construi aplicația finală. În mod normal, XLINK necesită următoarele informații la intrare:

- Fișiere obiect și librăriile necesare;
- Librăria standard ce conține mediul de rulare și funcțiile standard ale limbajului;
- Eticheta de start a programului;
- Un fișier de comandă a linkeditorului ce descrie schema memoriei sistemului țintă;
- Informații despre formatul de ieșire;

În linia de comandă, linia următoare poate fi folosită pentru pornirea XLINK :

```
xlink myfile.r90 myfile2.r90 -s __program_start -f lnmk128s.xcl  
cl3s-ec.r90 -o aout.a90 -FIntel-extended
```

În acest exemplu, `myfile.r90` și `myfile2.r90` reprezintă fișiere obiect, `lnmk128s.xcl` este fișierul de comandă al linkeditorului, iar `cl3s-ec.r90` este librăria de rulare. Opțiunea `-s` specifică locația din care aplicația porneste. Opțiunea `-o` specifică numele fișierului de ieșire iar opțiunea `-F` poate fi folosită pentru a specifica formatul fișierului de ieșire. (Formatul fișierului de ieșire implicit este Motorola.)

Link-editorul IAR XLINK produce ieșirea conform specificațiilor alese. Formatul de ieșire se alege conform scopului dorit. Se poate dori încărcarea ieșirii la un depanator, ceea ce înseamnă că este nevoie la ieșire de informații ale depanatorului. Ca alternativă, se poate încerca ieșirea la un flash loader, caz în care este nevoie de o ieșire fără informații ale depanatorului cum ar fi Intel-hex sau Motorola S-records.

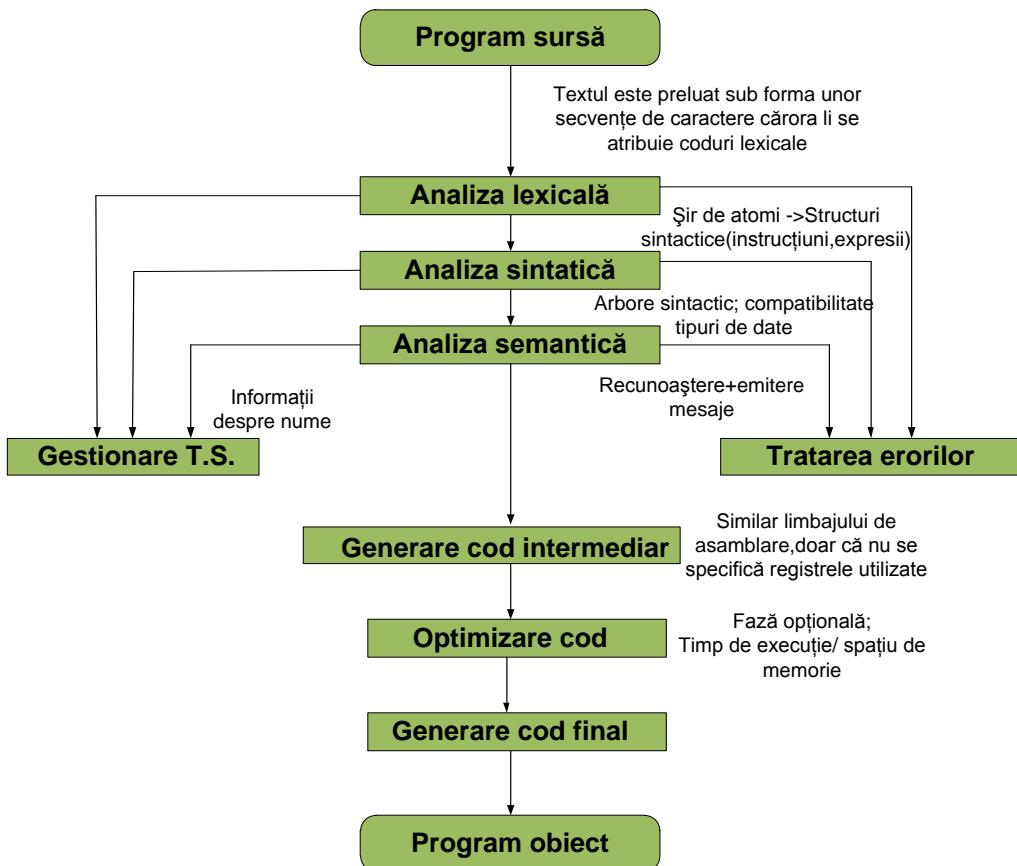


Figura 2.4 Etapele realizării unei aplicații

- **Setările de bază pentru configurarea proiectului**

Setările de bază disponibile pentru microcontroler-ul AVR sunt :

- Configurația procesorului;
- Modelul de memorie;
- Reprezentarea numărului double cu virgulă mobilă;
- Optimizări pentru viteză și dimensiune;
- Mediul de rulare.

a. **Configurarea procesorului:** Pentru un cod optim generat de compilator, setările trebuie realizate pentru microcontroler-ul folosit.

Există două opțiuni ale procesorului care pot fi folosite pentru configurarea suportului procesorului: `--cpu=derivative` și `-vn`.

Ambele opțiuni setează comportamentul implicit dar opțiunea `--cpu` este mai precisă deoarece conține mai multe informații despre target, decât mult mai generala opțiune `-v`.

Următorul tabel arată maparea opțiunilor procesor și ce microcontroler AVR suportă:

Variantă procesor	Opțiune generică	Derivate AVR suportate
<code>--cpu=1200</code>	<code>-v0</code>	AT90S1200
<code>--cpu=2313</code>	<code>-v0</code>	AT90S2313
<code>--cpu=2323</code>	<code>-v0</code>	AT90S2323
<code>--cpu=2333</code>	<code>-v0</code>	AT90S2333
<code>--cpu=2343</code>	<code>-v0</code>	AT90S2343
<code>--cpu=4414</code>	<code>-v1</code>	AT90S4414
<code>--cpu=4433</code>	<code>-v0</code>	AT90S4433
<code>--cpu=4434</code>	<code>-v1</code>	AT90S4434
<code>--cpu=8515</code>	<code>-v1</code>	AT90S8515
<code>--cpu=8534</code>	<code>-v1</code>	AT90S8534
<code>--cpu=8535</code>	<code>-v1</code>	AT90S8535
<code>--cpu=at43usb320a</code>	<code>-v3</code>	AT43USB320A
<code>--cpu=at43usb325</code>	<code>-v3</code>	AT43USB325
<code>--cpu=at43usb326</code>	<code>-v3</code>	AT43USB326
<code>--cpu=at43usb351m</code>	<code>-v3</code>	AT43USB351m
<code>--cpu=at43usb353m</code>	<code>-v3</code>	AT43USB353m
<code>--cpu=at43usb355</code> <code>--cpu=at94k</code>	<code>-v3</code> <code>-v3</code>	AT43USB355 FpSLic
<code>--cpu=at86rf401</code>	<code>-v0</code>	AT86RF401
<code>--cpu=can128</code> <code>--cpu=m8</code>	<code>-v3</code> <code>-v1</code>	AT90CAN128 ATmega8
<code>--cpu=m16</code>	<code>-v3</code>	ATmega16
<code>--cpu=m32</code> <code>--cpu=m48</code>	<code>-v3</code> <code>-v1</code>	ATmega32 ATmega48
<code>--cpu=m64</code>	<code>-v3</code>	ATmega64
<code>--cpu=m88</code> <code>--cpu=m103</code>	<code>-v1</code> <code>-v3</code>	ATmega88 ATmega103
<code>--cpu=m128</code>	<code>-v3</code>	ATmega128
<code>--cpu=m161</code> <code>--cpu=m162</code>	<code>-v3</code> <code>-v3</code>	ATmega161 ATmega162
<code>--cpu=m163</code>	<code>-v3</code>	ATmega163
<code>--cpu=m165</code> <code>--cpu=m168</code>	<code>-v3</code> <code>-v3</code>	ATmega165 ATmega168
<code>--cpu=m169</code>	<code>-v3</code>	ATmega169

--cpu=m2560 --cpu=m2561	-v5 -v5	ATmega2560 ATmega2561
--cpu=m323	-v3	ATmega323
--cpu=m325	-v3	ATmega325
--cpu=m3250	-v3	ATmega3250
--cpu=m329	-v3	ATmega329
--cpu=m3290	-v3	ATmega3290
--cpu=m406 --cpu=m645	-v3 -v3	ATmega406 ATmega645
--cpu=m6450	-v3	ATmega6450
--cpu=m649 --cpu=m6490	-v3 -v3	ATmega649 ATmega6490
--cpu=m8515	-v1	ATmega8515
--cpu=m8535 --cpu=tiny10	-v1 -v0	ATmega8535 ATtiny10
--cpu=tiny11	-v0	ATtiny11
--cpu=tiny12 --cpu=tiny13	-v0 -v0	ATtiny12 ATtiny13
--cpu=tiny15	-v0	ATtiny15
--cpu=tiny25 --cpu=tiny26	-v0 -v0	ATtiny25 ATtiny26
--cpu=tiny28	-v0	ATtiny28
--cpu=tiny45 --cpu=tiny85	-v1 -v1	ATtiny45 ATtiny85
--cpu=tiny2313	-v0	ATtiny2313

Tabel 2.1 Maparea opțiunilor procesorului

Următorul tabel rezumă caracteristicile memoriei pentru fiecare opțiune –v:

Opțiune generică procesor	Modele memorie	Atribut	Data max.	Dim. maximă a modulului și/sau a programului
-v0	Tiny	<u>nearfunc</u>	≤ 256 octeți	≤ 8 kocete
-v1	Tiny, Small	<u>nearfunc</u>	≤ 64 octeți	≤ 8 kocete
-v2	Tiny	<u>nearfunc</u>	≤ 256 octeți	≤ 128 kocete
-v3	Tiny, Small	<u>nearfunc</u>	≤ 64 kocete	≤ 128 kocete
-v4	Small, Large	<u>nearfunc</u>	≤ 16 Mo	≤ 128 kocete
-v5	Tiny, Small	<u>farfunc*</u>	≤ 64 kocete	≤ 8 Mo
-v6	Small, Large	<u>farfunc*</u>	≤ 16 Mo	≤ 8 Mo

Tabel 2.2 Rezumat al configurării procesorului

Note:

* Când este folosită opțiunea –v5 sau –v6, este posibil, pentru funcții individuale, să treacă peste atributul farfunc și să folosească în schimb atributul nearfunc.

Opțiunea –v nu reflectă volumul de date folosit, ci volumul maxim de date adresabil. Asta înseamnă că, de exemplu, dacă folosiți un microcontroler cu 16 Mo de date adresabili și nu folosiți mai mult de 256 o sau 64 Ko de date, trebuie să folosiți ori opțiunea –v4 ori –v6 pentru 16Mo date.

b. Modelul de memorie: Una dintre caracteristicile microcontroler-ului AVR este că există un compromis privind modul de accesare a memoriei variind între acces puțin costisitor, limitat la zone mici de memorie și metode de acces costisitor prin care este accesată orice locație de memorie. În compilatorul C/C++ AVR®IAR puteți seta o metodă implicită de acces la memorie selectând un anumit model de memorie. Există trei modele de memorie : Tiny, Small și Large. Alegerea opțiunii de procesor determină ce modele de memorie sunt disponibile. Dacă nu se specifică opțiunea pentru modelul de memorie, modelul Tiny va fi cel selectat în mod implicit pentru toate opțiunile procesorului cu excepție în cazul –v4 și –v6, unde se va folosi modelul Small. Programul poate folosi doar un singur model de memorie și același model trebuie folosit în toate modulele utilizatorului și în toate modulele librărie. Următorul tabel rezumă caracteristicile fiecărui model de memorie:

Modelul de memorie	Opțiunea generică procesor	Atribut de memorie implicit	Pointer implicit	Dimensiunea maximă a stivei
Tiny	-v0, -v1, -v2, -v3, -v5	<u>tiny</u>	<u>tiny</u>	≤ 256 o
Small	-v1, -v3, -v4, -v5, -v6	<u>near</u>	<u>near</u>	≤ 64 ko
Large	-v4, -v6	<u>far</u>	<u>far</u>	≤ 16 Mo

Tabel 2.3 Rezumatul modulelor de memorie

c. **Dimensiunea tipului double în virgulă mobilă**

Valorile în virgulă mobilă sunt reprezentate prin numere pe 32, respectiv 64 biți în formatul standardului IEEE754. Prin activarea opțiunii compilatorului `--64bit_doubles`, puteți alege dacă datele declarate ca fiind double să fie prezentate pe 64 de biți. Tipul de dată float este mereu reprezentat pe 32 de biți.

d. **Optimizări pentru viteză și dimensiune**

Compilatorul C/C++ AVR@IAR este un compilator state-of-the-art cu un optimizator care efectuează, pe lângă alte operații, eliminarea „codului mort”, propagare constantă și reducerea preciziei. În același timp efectuează optimizări de buclă. De cele mai multe ori optimizările vor face aplicația și mai rapidă și o vor și aduce la dimensiuni mai mici. Cu toate acestea, când nu este cazul, compilatorul utilizează ținta de optimizare selectată pentru decidiere asupra optimizării pe care o va efectua.

Nivelul și ținta de optimizare pot fi specificate pentru întreaga aplicație, pentru fiecare fișier în parte și pentru anumite funcții. În plus, anumite optimizări individuale pot fi dezactivate, cum ar fi plasarea în linie a funcțiilor.

e. **Mediul de rulare**

Pentru crearea mediului de rulare necesar va trebui să alegeti o librărie de rulare și să setați opțiunile de librărie. Ați putea de asemenea să doriți să treceți peste modulele de librării folosind versiunile personalizate proprii de librării.

Există două seturi de librării de rulare puse la dispoziție:

- *IAR DLIB Library*, care suportă ISO/ANSI C și C++. Această librărie suportă de asemenea numere în virgulă mobilă în format IEEE 754 și poate fi configurată pentru a include diferite nivele de suport pentru locale, descriptori de fișier, caractere multibyte, etc.
- *IAR CLIB Library*, este o librărie din categoria ușoară, care nu este compilată în totalitate cu ISO/ANSI C. De asemenea nu oferă suport deplin pentru numere în virgulă mobilă în format IEEE 754 sau suport pentru Embedded C++(această librărie este folosit implicit).

Librăria de rulare pe care o alegeti poate fi una dintre librării pre-built, sau o librărie pe care ați customizat-o sau ați construit-o. IAR Embedded Workbench IDE oferă template-uri pentru librăriile de proiect pentru ambele librării, care le puteți folosi pentru construirea propriilor tipuri de librării. Acest lucru vă oferă control deplin asupra mediului de rulare. Dacă proiectul conține doar cod sursă în assembler nu este nevoie alegerea unei librării de rulare.

Pentru alegerea unei librării, se alege Project>Options și se dă click pe tab-ul Library Configuration din categoria General Options. Se alege tipul de librărie adecvat din meniu drop-down.

Alegere unei librării din linia de comandă:

Linia de comandă	Descriere
<code>-I\avr\inc</code>	Specifică calea pentru includere
<code>-I\avr\inc\{clib dlib}</code>	Specifică calea pentru fișiere librărie specifice. Utilizați <code>clib/dlib</code> depinzând de ce librărie folosiți.
<code>libraryfile.r90</code>	Specifică fișierul obiect.
<code>--dlib_config C:\...\configfile.h</code>	Specifică fișierul de configurare pentru librărie (doar pentru biblioteca DLIB).

Tabel 2.4 Opțiuni ale liniei de comandă pentru specificarea bibliotecii și a fișierelor dependente

Tabelul arată de asemenea cum fișierul obiect corespunde opțiunilor dependente de proiect.

2.4.2 SUPORT SPECIAL PENTRU SISTEME EMBEDDED

Aceasta secțiune descrie pe scurt extensiile oferite de compilatorul de C/C++ AVR IAR pentru a suporta caracteristicile microcontroler-ului AVR.

- Directivele pragma** controlează comportamentul compilatorului, de exemplu modul în care se alocă memorie, dacă permite cuvinte cheie extinse sau dacă emite mesaje de eroare. Directivele pragma sunt tot timpul activate în compilatorul AVR IAR. Ele se potrivesc cu ISO/ANSI C și sunt foarte folositoare în momentul în care se dorește asigurarea portabilității.

Următorul tabel arată directivele pragma ale compilatorului :

<code>#pragma šablon_de_bază</code>	Face o funcție šablon pe deplin conștientă de memorie
<code>#pragma bitfields</code>	Controlează ordinea câmpului de biți
<code>#pragma constseg</code>	Plasează variabile constante într-un câmp numit
<code>#pragma data_alignment</code>	Selectează modul de aliniere a datelor în memorie
<code>#pragma dataseg</code>	Plasează variabile într-un segment numit
<code>#pragma diag_default</code>	Modifică nivelul de severitate al mesajelor de diagnosticare
<code>#pragma diag_error</code>	Modifică nivelul de severitate al mesajelor de diagnosticare
<code>#pragma diag_remark</code>	Modifică nivelul de severitate al mesajelor de diagnosticare
<code>#pragma diag_suppress</code>	Suprimă mesajele de diagnosticare
<code>#pragma diag_warning</code>	Modifică nivelul de severitate al mesajelor de diagnosticare
<code>#pragma include_alias</code>	Specifică un alias pentru un fișier inclus
<code>#pragma inline</code>	Face o funcție inline
<code>#pragma language</code>	Controlează extensiile de limbaj IAR
<code>#pragma location</code>	Specifică adresa absolută a unei variabile
<code>#pragma message</code>	Afișează un mesaj

#pragma object attribute	Schimbă definiția unei variabile sau a unei funcții
#pragma optimize	Specifică tipul și nivelul de optimizare
#pragma pack	Specifică alinierea membrilor unei structuri sau a unei uniuni
#pragma required	Asigură faptul că un simbol care este folosit de un alt simbol este prezent în ieșire legată.
#pragma rtmode1	Adaugă un atribut model de execuție modulului
#pragma segment	Declară un nume de segment pentru a fi utilizat de către funcțiile intrinseci
#pragma type_attribute	Modifică declarația și definiția unei variabile sau a unei funcții
#pragma vector	Specifică vectorul unei funcții de întinerupere

Tabel 2.5 Rezumatul directivelor pragma

- Cu **simbolurile predefinite ale preprocesorului**, se poate inspecta mediul din timpul compilării, de exemplu timpul de compilare, varianta de procesor și modelul de memorie folosit. Rezumatul simbolurilor predefinite:

Simbolul predefinit	Identificări
ALIGNOF ()	Accesează alinierea unui obiect
BASE_FILE	Identifică numele fișierului să fie compilat. Dacă fișierul este un fișier antet, numele de fișier care include fișierul header este identificat.
CORE	Identifică varianta procesorului în uz
CPU	Identifică varianta procesorului în uz
cplusplus	Stabilește dacă se execută în compilator modul C ++
DATE	Determină data de compilare
derivative	Coresponde procesorul specificat cu opțiunea compilatorului – cpu
embedded_cplusplus	Stabilește dacă se execută în compilator modul C ++
FILE	Identifică numele fișierului fiind compilat
func	Se extinde într-un string cu numele funcției și contextul
FUNCTION	Se extinde într-un string cu numele funcției și contextul
HAS EEPROM	Determină dacă există un EEPROM disponibil
IAR SYSTEMS ICC	Identifică platforma compilatorului IAR
ICCAVR	Identifică compilatorul AVR IAR C / C ++
LINE	Determină numărul liniei sursei de curent
MEMORY MODEL	Identifică modelul de memorie în uz
NDEBUG	Determină dacă afirmațiile trebuie să fie incluse sau nu în cererea construită
Pragma ()	Poate fi utilizat în preprocesor, definește și are efect echivalent

	cu directiva Pragma
<u><u>PRETTY_FUNCTION</u></u>	Se extinde într-un string cu numele funcției, inclusiv tipuri de parametru și de tip întoarcere, ca context
<u><u>STDC</u></u>	Identifică ISO / standardului ANSI C
<u><u>STDC_VERSION</u></u>	Identifică versiunea standardului ISO / ANSI standardului C în uz
<u><u>TID</u></u>	Identifică procesor țintă al IAR compilator în uz
<u><u>TIME</u></u>	Determină timpul de compilare
<u><u>VER</u></u>	Identifică procesor țintă al IAR compilator în uz
<u><u>VERSION_1_CALLS</u></u>	Identifică convenția de apelare în uz

Tabel 2.6 Rezumatul simbolurilor predefinite

- **Fișiere header pentru I/O**

Unitățile periferice standard sunt definite în fișierele header specifice dispozitivului cu extensia .h . Pachetul produsului pune la dispoziție fișiere I/O pentru toate dispozitivele disponibile la timpul respectiv la punerea pe piață a produsului. Fișierele se găsesc în directorul avr/inc. Dacă este nevoie de fișiere header I/O adiționale, ele pot fi foarte ușor create folosind unul dintre cele puse la dispoziție ca template-uri.

- **Accesarea regiștrilor speciali de funcții**

Fișiere header specifice pentru mai multe deriveate AVR sunt incluse în compilatorul AVR IAR C/C++ de la lansare. Fișierele header se numesc `iodependent.h` și definesc regiștrii de funcții specifici procesorului (*special function registers - SFRs*).

În IAR Embedded Workbench, activarea definițiilor pe bit se face selectând opțiunea **General Options>System>Enable bit definitions in I/O include files**.

SFR-urile cu câmpuri de biți sunt declarați în fișierele header. Următorul exemplu este din `iom128.h`:

```

__io union
{
    unsigned char PORTE;      /* The sfrb as 1 byte */
    struct
    {
        unsigned char PORTE_Bit0:1,
        PORTE_Bit1:1,
        PORTE_Bit2:1,
        PORTE_Bit3:1,
        PORTE_Bit4:1,
        PORTE_Bit5:1,
        PORTE_Bit6:1,
        PORTE_Bit7:1;
    };
} @ 0x1F;

```

Prin includerea fișierului potrivit la cod, este posibilă accesarea fie a întregului registru, fie a oricărui bit individual (sau câmp de bit) din cod C după cum urmează:

```
/* whole register access */
```

```
PORTE = 0x12;  
/* Bitfield accesses */  
PORTE.PORTE_Bit0 = 1
```

2.5 REALIZAREA UNUI PROIECT (PENTRU ATMEGA16)

2.5.1 CREAREA UNUI PROIECT ÎN IAR EMBEDDED WORKBENCH 5.51

Se vor utiliza următoarele medii de dezvoltare:

IAR Embedded Workbench 5.51	editare cod sursă și compilare
AVR Studio 4.19	execuție și debug

- se deschide mediul IAR 5.51
- menu=Project → Create New Project → option=Empty Project → button=OK

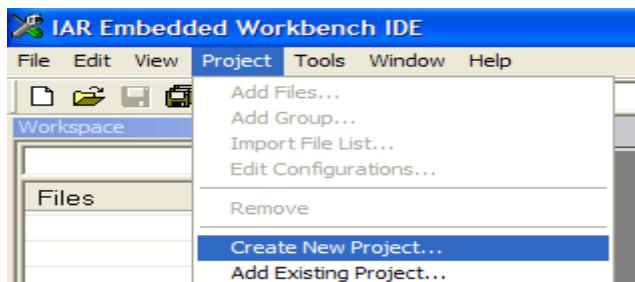


Figura 2.5

- se indică locația și numele proiectului (într-un director nou)

1. Adăugarea fișierelor sursă

- menu=File → menu>New → menu=File
- menu=File → menu=Save
- menu=Project → menu>Add files...

2. Deschiderea unui proiect existent în IAR

- menu=File → menu=Open → menu=Workspace...

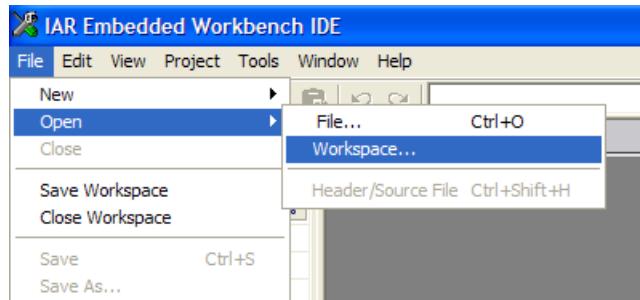


Figura 2.6

- se indică fișierul spațiului de lucru (cu extensia .eww) din directorul corespunzător proiectului

3. Configurarea proiectului în IAR 5.51

- menu=Project → menu=Options...

În fereastra ce se va deschide vor fi alese următoarele opțiuni:

- General Options → Target → Processor Configuration = -cpu=m16, ATmega16

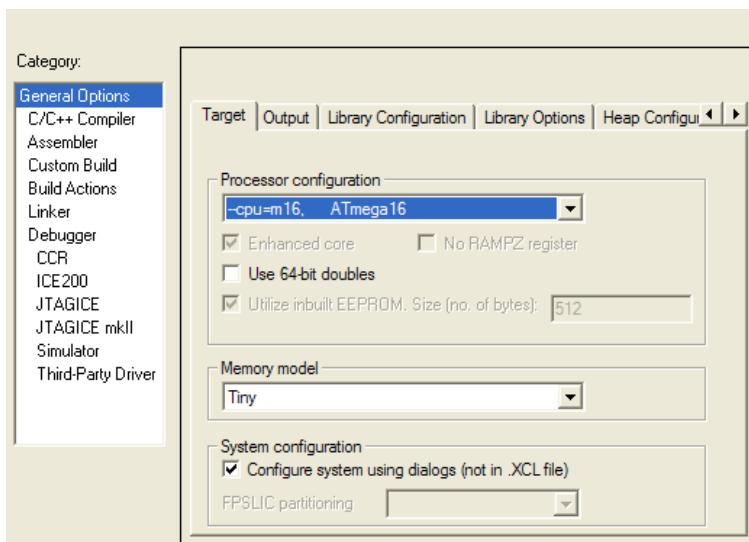


Figura 2.7

- General Options → System → Enable bit definitions in I/O-Include files = enabled

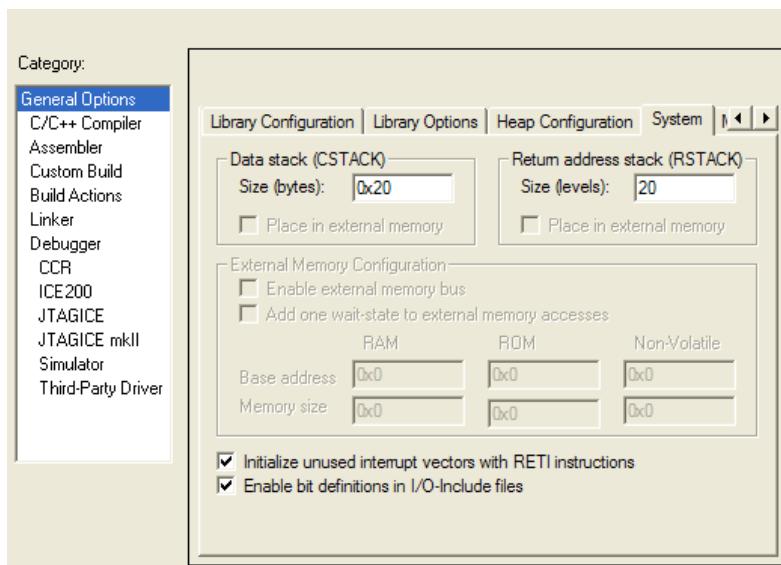


Figura 2.8

- Linker → Output → Output Format = ubrof 8 (forced)

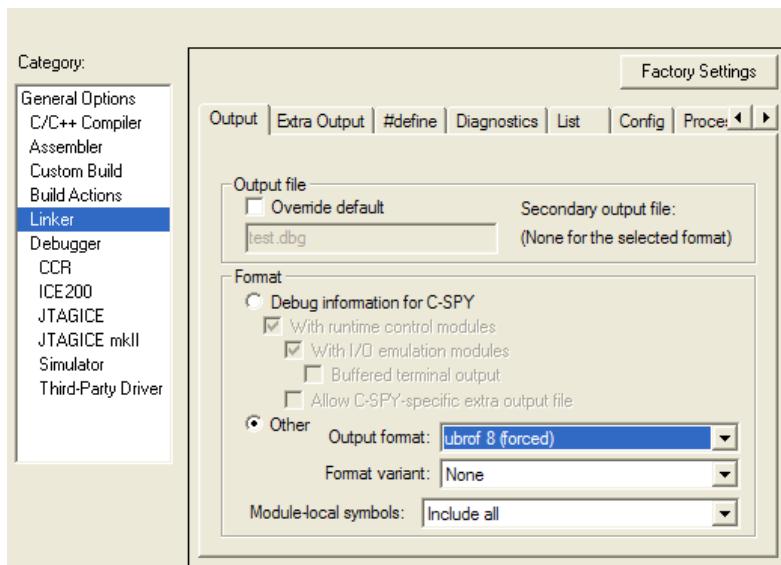


Figura 2.9

- C/C++ Compiler → Optimizations = None

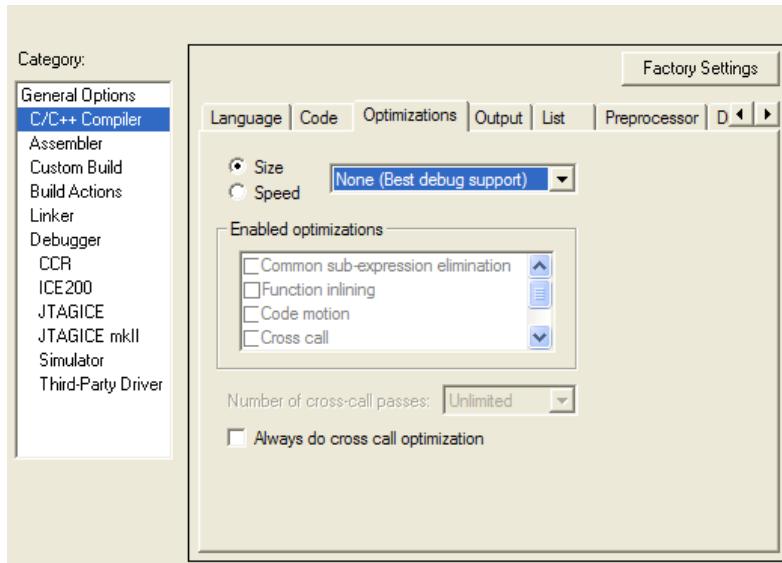


Figura 2.10

- C/C++ Compiler → List → Output list file = enabled (pentru a obține fișierul lst)

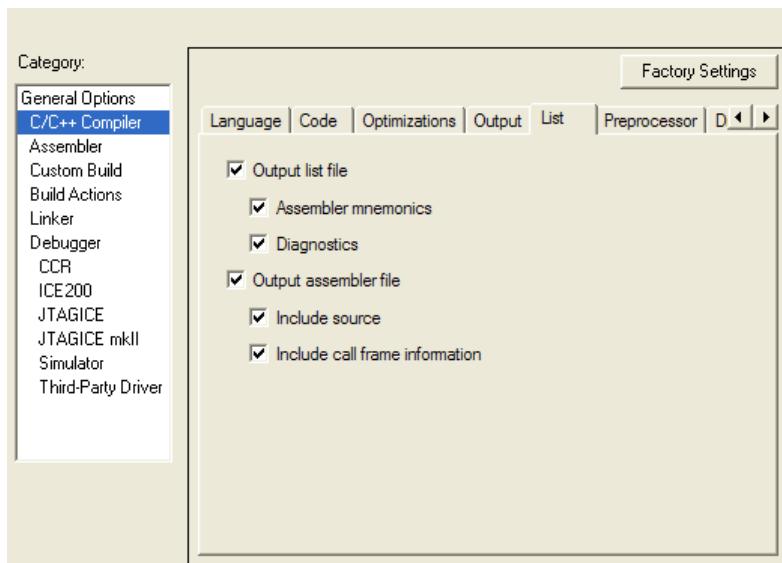


Figura 2.11

- Linker → List → Generate linker listing = enabled, Segment Map = enabled, Module map = enabled (pentru a vedea fișierul map)

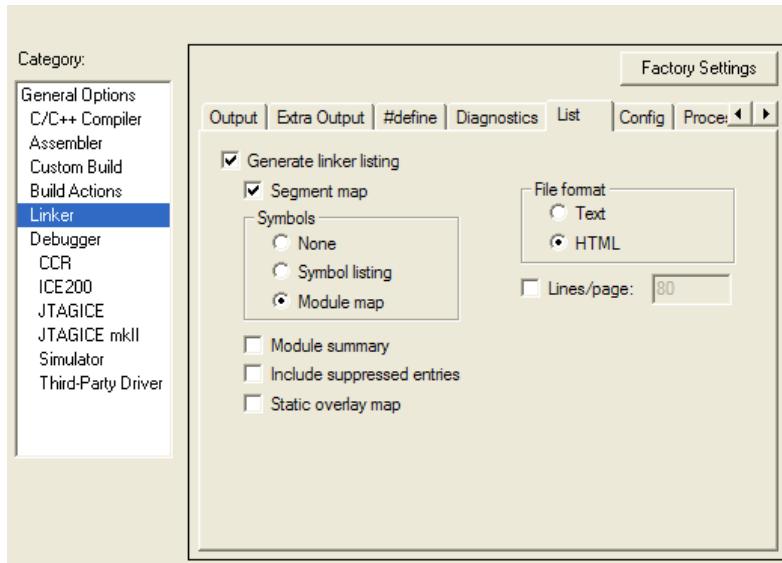


Figura 2.12

Înainte de a compila proiectul, se va defini spațiul de lucru astfel:

- menu=Project → menu=Make
- se va indica locația fișierului cu informații despre spațiul de lucru.

2.5.2 CREAREA PROIECTULUI ÎN AVR 4.19

De fapt proiectul specific mediului AVR se va construi peste proiectul creat mai devreme în IAR, urmând pașii:

- se deschide mediul AVR
- menu=File → menu=Open File... → Open As=Auto
- se va indica fișierul cu extensia .dbg din directorul /Debug/Exe al proiectului creat cu IAR
- button=Open
- va apărea un mesaj în care ni se va spune că mediul AVR își va construi fișierul de proiect (cu extensia .aps)

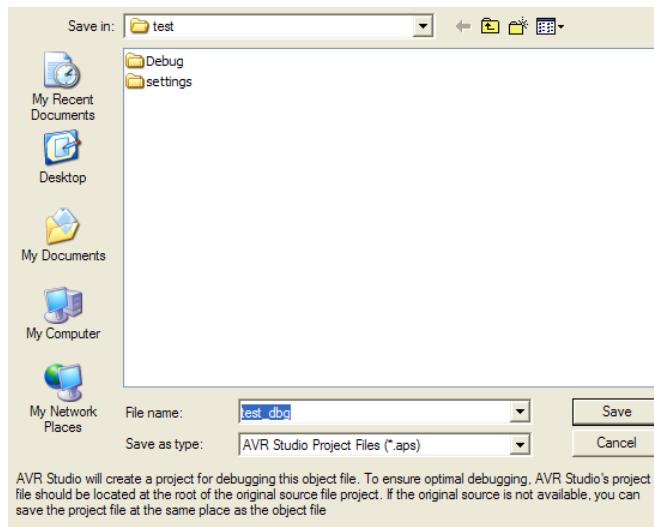


Figura 2.13

- button=Save
- va apărea o fereastră în care se va cere să se aleagă platforma pe care va rula codul executabil

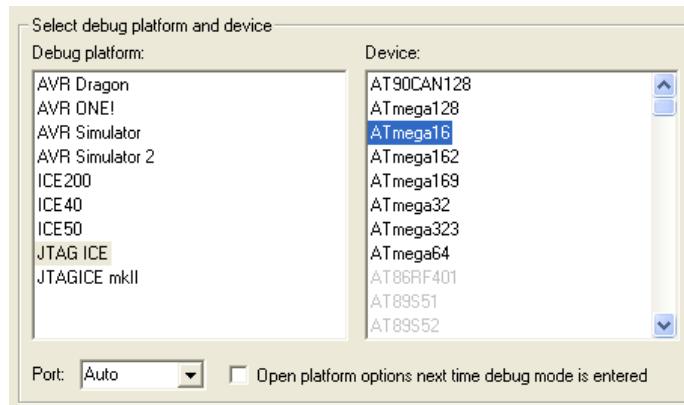


Figura 2.14

- Debug platform=JTAG ICE, Device=ATmega16
- button=Finish

Notă: codul sursă vizibil în AVR poate fi doar citit.

1. Deschiderea proiectului în AVR

- menu=File → menu=Open
- se va indica fișierul aplicației (cu extensia .aps) din directorul proiectului.

2. Înfruntarea greutăților de ordin tehnic

- în cazul în care adaptorul USB to Serial nu este recunoscut de către sistemul de operare, se va căuta un driver din clasa ftdi 232 (<http://www.ftdichip.com/Drivers/VCP.htm>)
- în general: dacă o componentă nu este recunoscută, se deconectează de la portul de comunicație sau de la sursa de alimentare dacă este cazul, și se reconectează, urmărindu-se comportamentul acesteia

3. Execuția programului

- menu=Debug → menu=Start Debugging
- menu=Debug → menu=Run (sau altă opțiune)

Este notabil faptul că se pot crea *breakpoint*-uri în program. Astfel putem verifica, de exemplu:

- dacă o secvență de cod se execută sau nu (o rutină de îintrerupere)
- valoarea regiștrilor și conținutul memoriei la un moment dat

4. Vizualizarea conținutului memoriei

Pentru a urmări valoarea unei variabile în program se va adăuga numele variabilei în fereastra Watch:

- în fișierul sursă se selectează variabila
- click-dreapta
- option>Add variable <nume variabilă> to Watch. În continuare valoarea poate fi citită oprind programul din execuție (folosind un *breakpoint*, de exemplu)

Name	Value	Type	Location
n	6	unsigned short volatile	0x007E [SRAM]

Figura 2.15

Se poate studia conținutul memoriei de la o adresă oarecare:

- menu=View → menu=Memory
- în fereastra deschisă se va putea selecta tipul de memorie (Data, Program, etc) și adresa locației de memorie

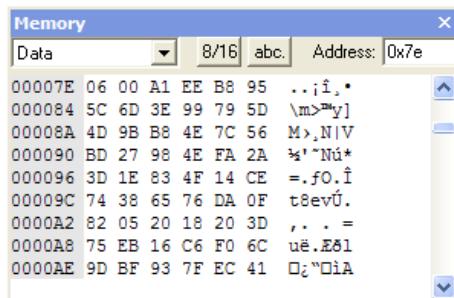


Figura 2.16

5. Programarea folosind AVR Studio 4:

- Tools-Program AVR –Connect
- Se alege numele portului corespunzător USB (a se vedea **Execuția programului**) sau dacă nu se știe se alege Auto -> Connect

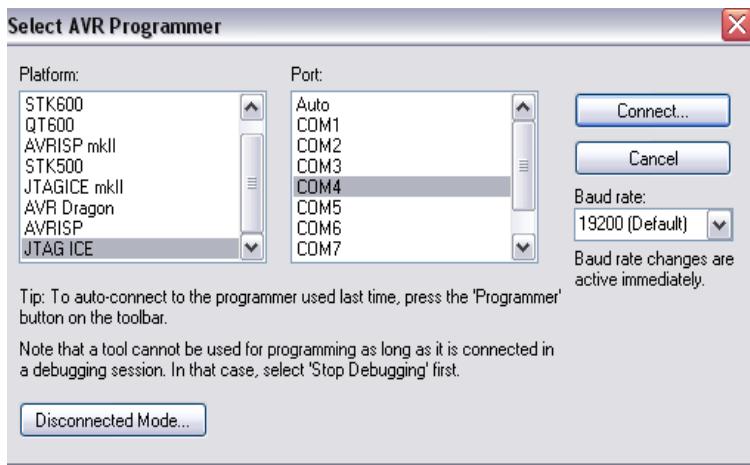
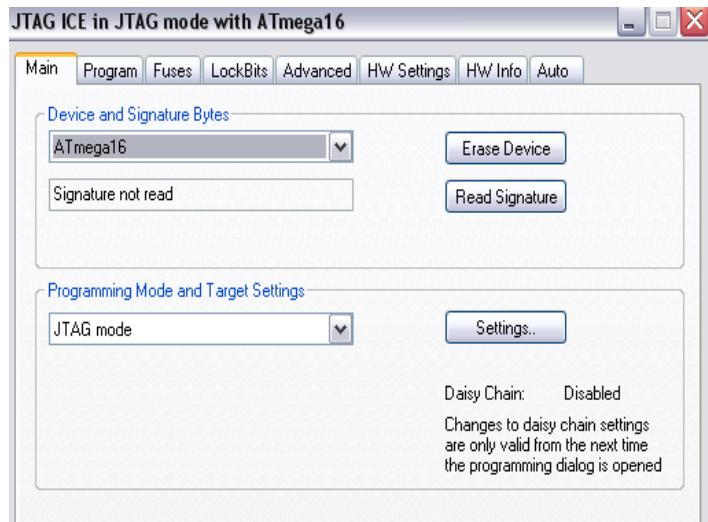


Figura 2.17

- În fila Main → Device and Signature Bytes ->Se alege Atmega16 din lista de modele AVR



Figură 2.18

- Pentru a testa conexiunea la Atmega16 : Main -> Read Signature. Astfel se trimit o comandă la Atmega16 care cere semnătura dispozitivului. Dacă totul e conectat corect va apărea: Signature matches selected device(Semnătura se potrivește cu dispozitivul ales)

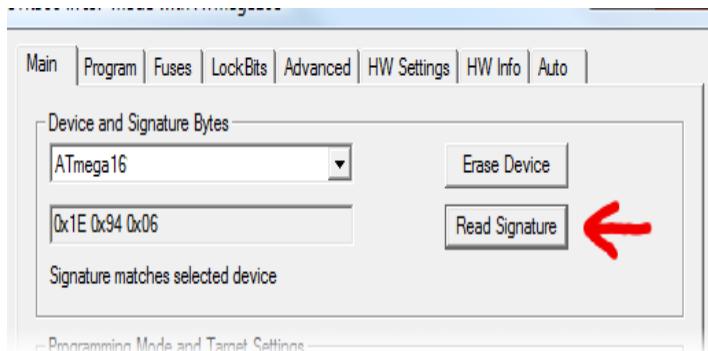
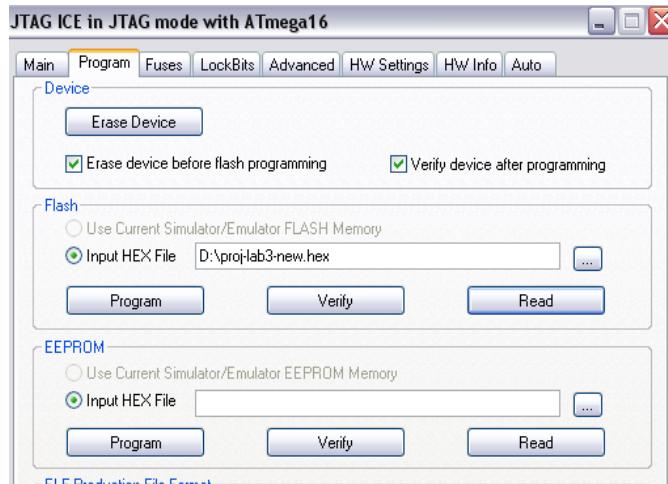


Figura 2.19

- Pentru a programa dispozitivul întă: fila Program – Secțiunea Flash- În Input HEX File se pune fișierul hex generat la construirea proiectului(fișierul se află în default\<numele_proiectului.hex>). Se apasă butonul Program din secțiunea Flash.



Figură 2.20

6. Caracteristici avansate ale AVR Studio 4:

- Apăsând fila Fuses(siguranțe) se citesc automat setările de siguranță ale dispozitivului AVR întă. Dacă dispozitivul nu e conectat atunci se va afișa un mesaj de eroare. Sigurantele vă permit să configurați mai multe aspecte persistente, fundamentale ale dispozitivului AVR. Pentru a afla mai multe despre sigurantele și ceea ce fac ele, vezi fișa tehnică pentru Atmega16.

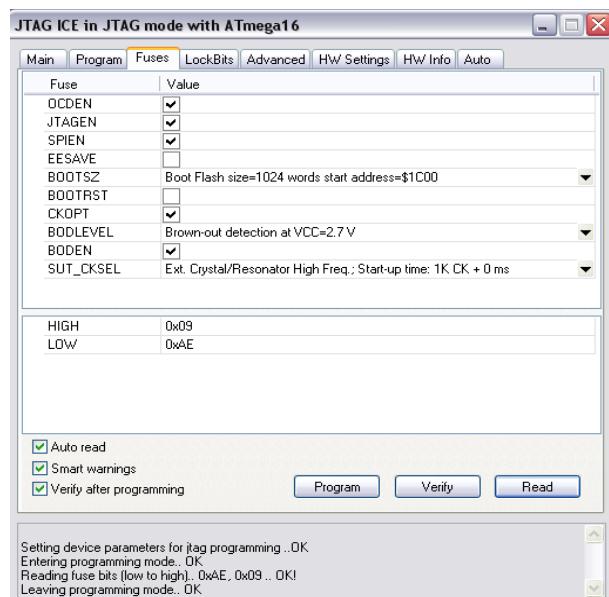


Figura 2.21

- Apăsând fila LockBits se citesc automat biții de blocare a dispozitivului AVR. Dacă dispozitivul nu e conectat atunci se va afișa un mesaj de eroare. Acești biți permit securizarea dispozitivului prevenind citirea sau scrierea ulterioară de pe flash. Biții de blocare pot fi resetați la o stare neblocantă prin ștergerea chip-ului(fila Main – butonul Erase Device). Biții de blocare sunt importanți atunci când se dorește cedarea produsului altor persoane fără a le da acces la program sau dacă se dorește împiedicarea rescrierii accidentale a chip-ului programat.

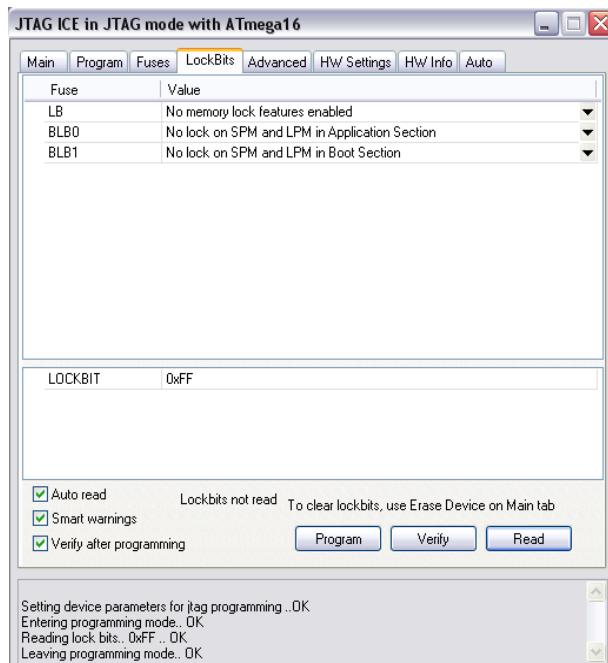


Figura 2.22

2.5.3 CREAREA UNUI PROIECT ÎN IAR 6.11

- se deschide mediul IAR 6.11
- menu=Project → menu=Create New Project → option=C→ main→ buton=OK

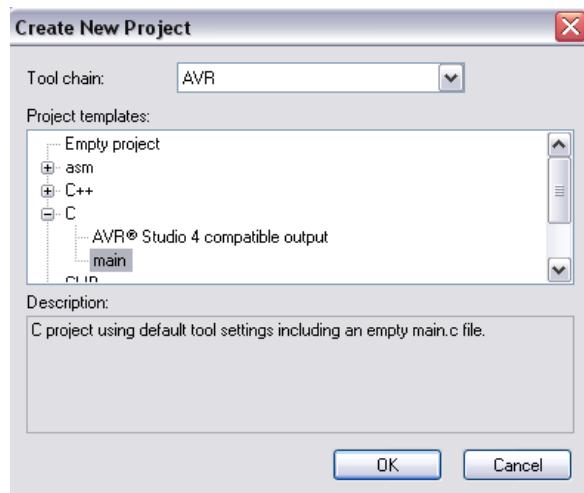


Figura 2.23

- se indică locația și numele proiectului (într-un director nou)

1. Adăugarea fișierelor sursă

- menu=File → menu>New → menu=File
- menu=File → menu=Save
- menu=Project → menu>Add files...

2. Configurarea proiectului în IAR 6.11

- menu=Project → menu=Options...

În fereastra ce se va deschide vor fi alese următoarele opțiuni:

- General Options → Target → Processor Configuration = Atmega16

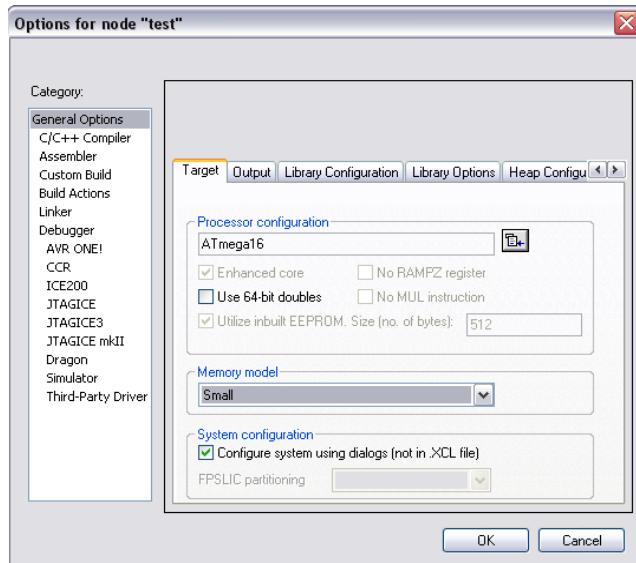


Figura 2.24

- General Options → System → Enable bit definitions in I/O-Include files = enabled

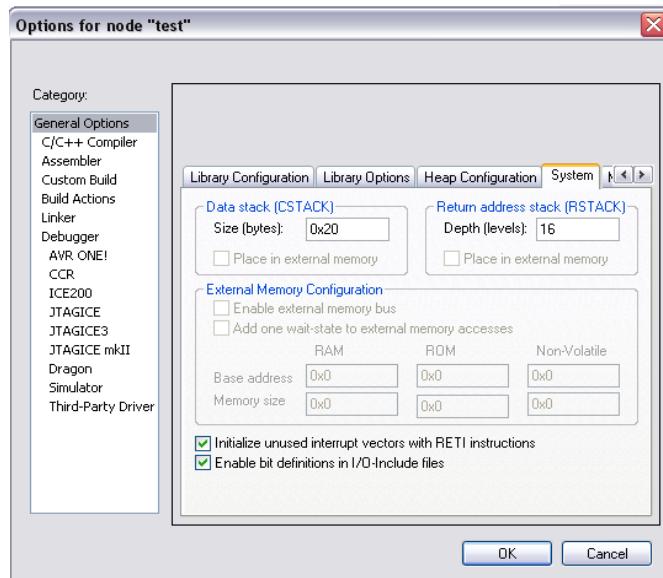


Figura 2.25

- Linker → Output → Output Format = ubprof 8 (forced)

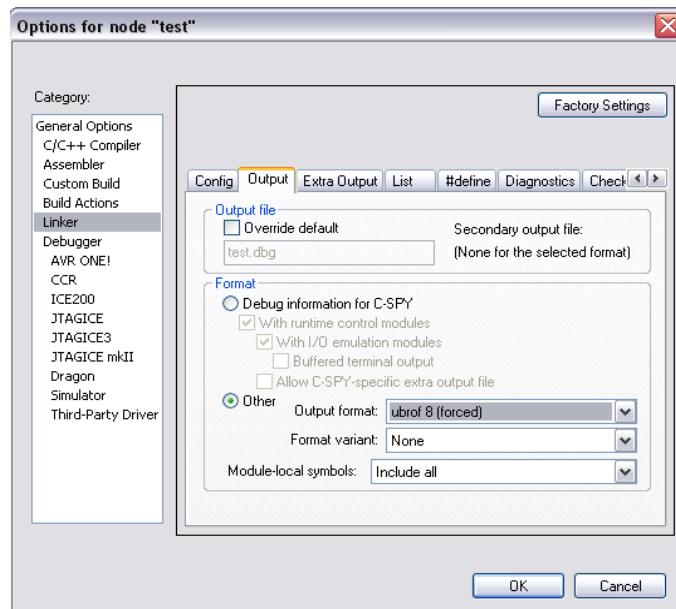


Figura 2.26

- C/C++ Compiler → Optimizations = None

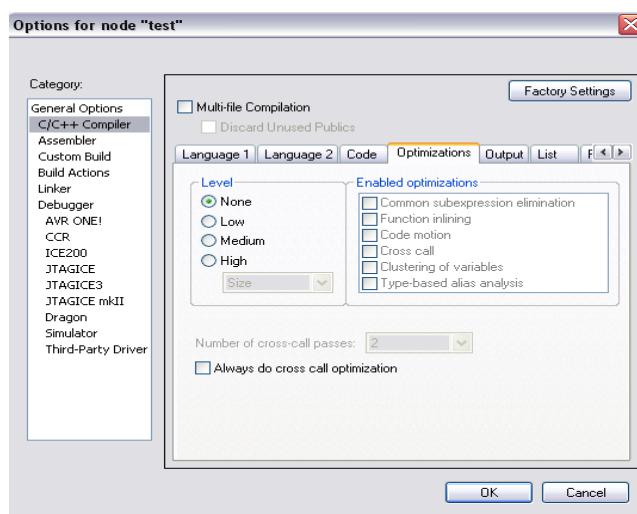


Figura 2.27

- C/C++ Compiler → List → Output list file = enabled (pentru a obține fișierul lst)

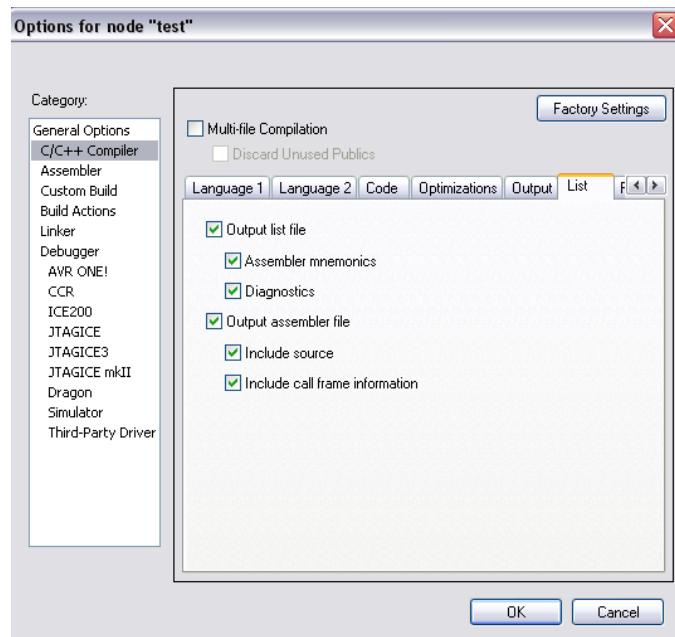


Figura 2.28

- Linker → List → Generate linker listing = enabled, Segment Map = enabled, Module map = enabled (pentru a vedea fișierul map)

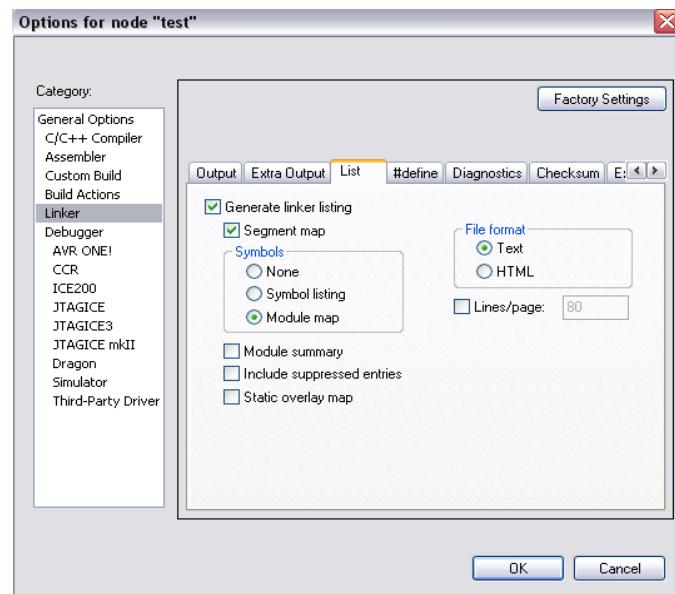


Figura 2.29

- Debugger→Setup→Driver→JTAGICE

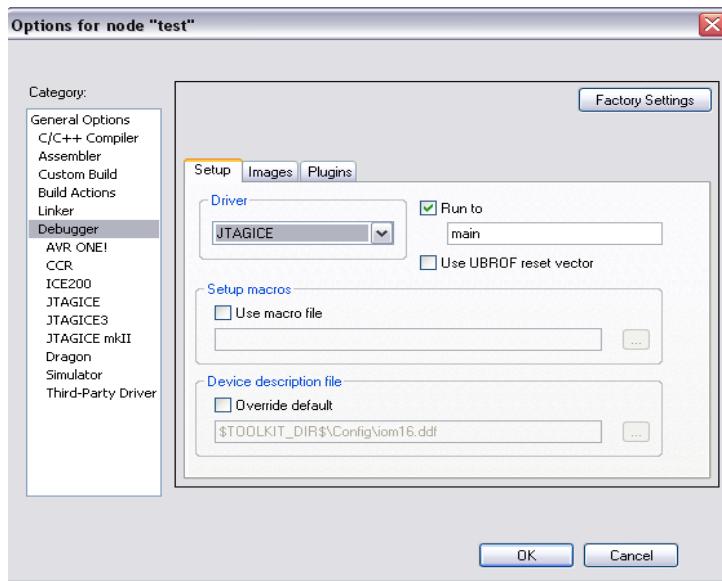


Figura 2.30

- JTAGICE→JTAGICE1→bifat Default communication→Se alege COM-ul corespunzător USB serial port ((aici COM 4)a se vede 4.2.4)
- JTag Port→Frequency in Hz: 100 KHz

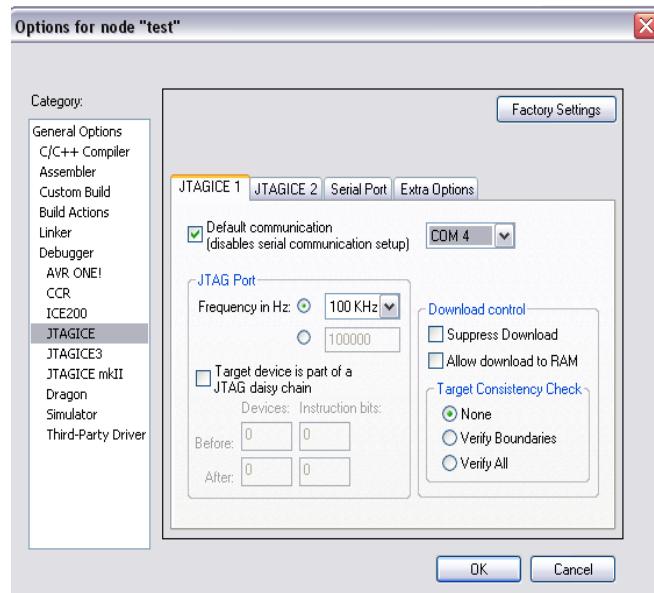


Figura 2.31

Se va defini spațiul de lucru astfel:

- menu=File → menu=Save workspace
- se va indica locația fișierului cu informații despre spațiul de lucru

3. Execuția programului

Pentru a vedea care e COM-ul corespunzător portului serial USB:

- Control Panel - System-Hardware-Device manager-Ports

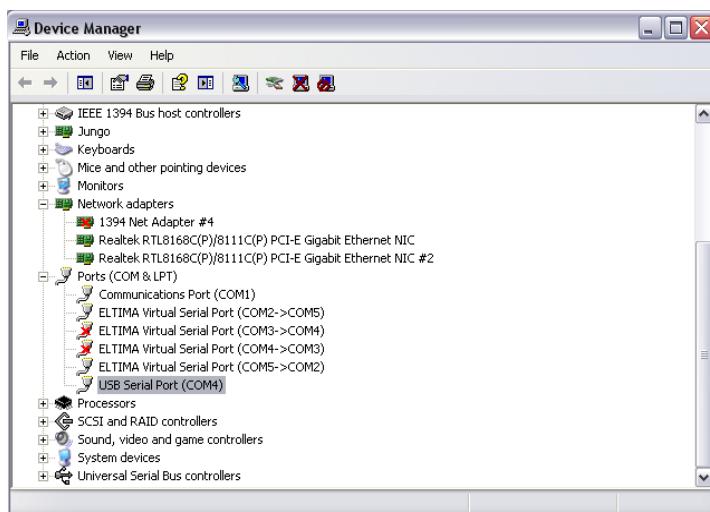


Figura 2.32

- Pentru a executa programul: Project->Download and Debug

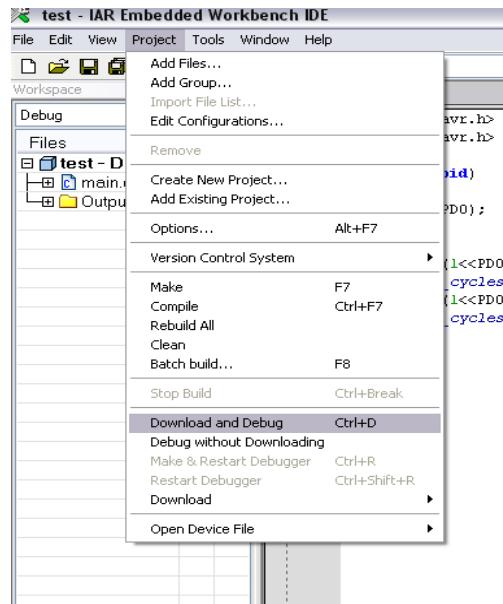


Figura 2.33

Este notabil faptul că se pot crea *breakpoint*-uri în program. Astfel putem verifica, de exemplu:

- dacă o secvență de cod se execută sau nu (o rutină de întrerupere)
- valoarea reștrînilor și conținutul memoriei la un moment dat

4. Vizualizarea conținutului memoriei

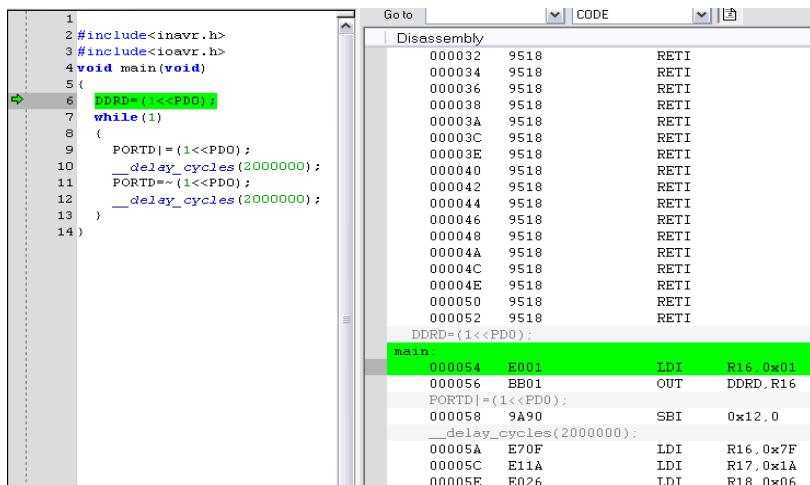


Figura 2.34

Pentru a urmări valoarea unei variabile în program se va adăuga numele variabilei în fereastra Watch:

- Meniu=View ->Watch

Watch 1			
Expression	Value	Location	Type
DDRD	'.' (0x01)	DATA: 0x000031	unsigned char
<click to ...			

Figura 2.35

Se poate studia conținutul memoriei de la o adresă oarecare:

- menu=View → menu=Memory
- În fereastra deschisă se va putea selecta tipul de memorie (Data, Program, etc.) și adresa locației de memorie

Go to	CODE
0000 Go to memory address	95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 ..
0010 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 ..	
0020 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 ..	
0030 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 ..	
0040 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 ..	
0050 18 95 18 95 c2 50 06 e0 10 e0 08 83 19 83 01 e0 ..	
0060 01 bb 90 9a 0f e7 1a e1 26 e0 01 50 10 40 20 40 ..	
0070 e1 f7 00 c0 00 00 0e ef 02 bb 0f e7 1a e1 26 e0 ..	
0080 01 50 10 40 20 40 e1 f7 00 c0 00 00 ea cf 00 00 ..P	
0090 88 95 fe cf 0f e9 0d bf 00 e0 0e bf c0 e8 d0 e0 ..	
00a0 0e 94 58 00 0e 94 2a 00 0e 94 47 00 0c 94 47 00 ..	
00b0 01 e0 08 95 ff ..	

Figura 2.36

3 Aprinderea unui LED folosind delay_cycles(). Funcții intrinseci.

3.1 CUM FUNCȚIONEAZĂ UN LED

3.1.1 CE ESTE UN LED?

Un **LED** (eng. Light-Emitting Diode) este o diodă semiconductoare ce emite lumină la polarizarea directă a joncțiunii p-n. Efectul este o formă de electroluminescență.

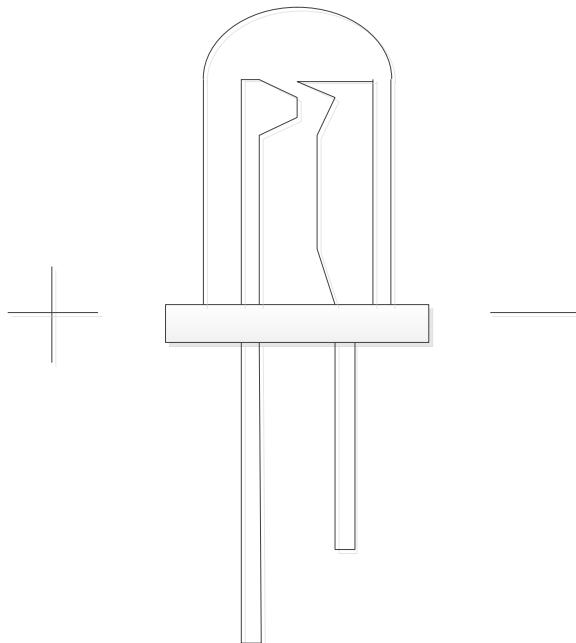


Figura 3.1 Schema simplificată a unui LED

Un LED este o sursă de lumină mică, de cele mai multe ori însorită de un circuit electric ce permite modularea formei radiației luminoase. Deseori, acestea sunt utilizate ca indicatori în cadrul dispozitivelor electronice, dar din ce în ce mai mult au început să fie utilizate în aplicații de putere ca surse de iluminare. Culoarea luminii emise depinde de compoziția și de starea materialului semiconductor folosit, și poate fi în spectrul infraroșu, vizibil sau ultraviolet. Pe lângă iluminare, LED-urile sunt folosite din ce în ce mai des într-o serie mare de dispozitive electronice.

3.1.2 ANOD, CATOD, TENSIUNI DE ALIMENTARE. REZISTENȚA DE LIMITARE A CURENTULUI

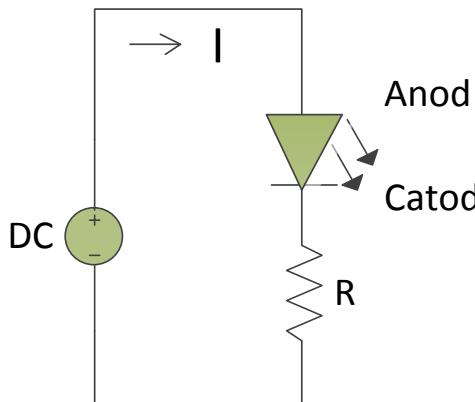


Figura 3.2 Schema simplificată a conexiunii

Înainte de conectarea unui led în cadrul unui circuit electronic, trebuie să avem în vedere unele din caracteristicile electrice și optice ale acestora:

- **Curentul maxim (mA):** Pentru a evita deteriorarea LED-ului trebuie întotdeauna să limităm curentul maxim prin LED, de regulă cu ajutorul unei rezistențe; pentru marea majoritate a LED-urilor această valoare este de 20 mA.
- **Tensiunea de alimentare (V):** Tensiunea necesară la borne pentru obținerea curentului maxim admis; de remarcat că această valoare este o caracteristică proprie fiecărui LED, de aceea în catalog vom găsi un interval de valori sau o valoare tipică.
- **Lungimea de undă:** Lungimea de undă arată culoarea exactă a LED-ului (de exemplu, 660 nm este roșu iar 625 nm este roșu - portocaliu). LED-urile albe sunt caracterizate prin temperatura de culoare (de exemplu 5500K - alb rece, 3300K - alb cald).

La realizarea și celui mai simplu circuit cu LED-uri, se va ține cont de montarea unei rezistențe de limitare a curentului pentru fiecare LED (grup de LED-uri legate în serie). Valoarea acesteia se va stabili în felul următor:

$$R = (U - U_b) / I, \text{ unde:}$$

- U este tensiunea de alimentare a circuitului;
- U_b este tensiunea la bornele LED-ului (uzual: 2V pentru LED roșu, galben, verde și 3,5V pentru albastru și alb). În cazul mai multor LED-uri montate în serie, aceasta se multiplică cu numărul de LED-uri;
- I este curentul maxim prin LED (uzual 20 mA).

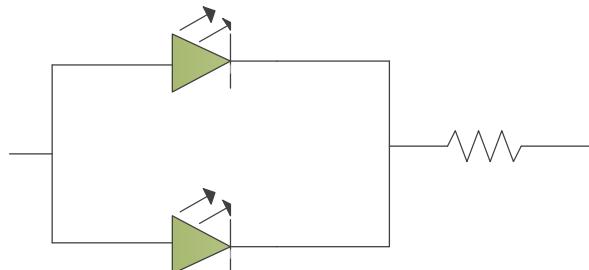
De exemplu, dacă led-ul HLMP1790 de la Farnell, de culoare verde, având următoarele parametrii: lungimea de undă (565nm), curentul continuu maxim admis (20mA), tensiunea la bornele led-ului (2V), se alimentează la sursa de curent continuu de 5V, atunci rezistența de limitare a curentului va fi:

$$U_R = 5V - 2V = 3V$$

$$R = U_R / I = 3V / 0.02mA = 150\Omega$$

Pentru legarea în paralel se va ține cont de următoarea schemă de montaj:

Greșit:



Corect:

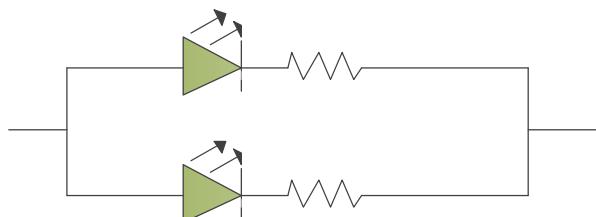


Figura 3.3 Conectarea în paralel

3.2 CONFIGURAȚIA PINILOR DIN ATMEGA16

3.2.1 PORTURILE DIN ATMEGA16 (PORTA, PORTB, PORTC, PORTD)

ATmega16 este un microcontroler bazat pe arhitectura RISC AVR îmbunătățită, ce lucrează pe 8 biți și are 40 de pini de ieșire. Dintre aceștia, 32 sunt pini de I/O (Input/Output) grupați în 4 porturi (PORTA, PORTB, PORTC, PORTD). Fiecare din cele 4 porturi au asociate câte 3 registre pe 8 biți ce conțin configurațiile fiecărui pin în parte (DDR_x, PIN_x, PORT_x).

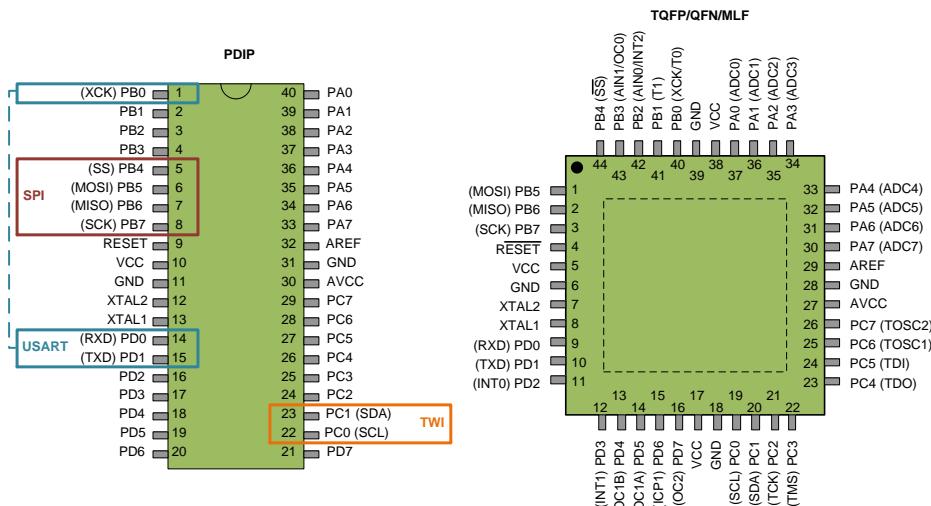


Figura 3.4 Descrierea pinilor

Microcontroler-ul dispune de un set de 131 instrucțiuni și 32 de registri de uz general (cei menționați mai sus). Cele 32 de registre sunt direct adresabile de Unitatea Logică Aritmetică (ALU), permitând accesarea a două registre independente într-o singură instrucțiune. Se obține astfel o eficiență sporită în execuție (de până la zece ori mai rapide decât microcontrolerele convenționale CISC).

3.2.2 DESCRIEREA PE SCURT A PINILOR ȘI UTILIZAREA LOR FRECVENȚĂ

VCC - Alimentarea digitală

GND – Masa – 0V

PortA (PA7...PA0) - Portul A servește ca intrare analogică pentru ADC (Analog to Digital Converter). Atunci când ADC nu este folosit, portul A poate fi folosit și ca port I/O bidirecțional pe 8 biți. Pinii de port pot fi conectați optional la VCC prin rezistori interni (selectați pentru fiecare bit). Buffer-ele de ieșire ale Portului A au caracteristici de amplificare.

PortB (PB7...PB0) - Portul B este un port I/O bidirecțional pe 8 biți cu rezistență de pull-up internă. Buffer-ele de ieșire ale portului B au caracteristici de amplificare. Portul B îndeplinește de asemenea funcții speciale ale microcontroler-ului ATmega16.

PortC (PC7...PC0) - Portul C este un port I/O bidirecțional pe 8 biți cu rezistență de pull-up internă. Portul C este folosit de asemenea și pentru unele funcții ale interfeței JTAG. Dacă interfața JTAG (de *depanare, debug*) este activată, rezistorii pinilor PC5(TDI), PC3(TMS) și PC2(TCK) vor fi activați, chiar dacă are loc o resetare.

PortD (PD7...PD0) - Portul D este un port I/O bidirectional pe 8 biți cu rezistență de pull-up internă. Buffer-ele de output ale Port-ului D au caracteristici de amplificare. Port-ul D îndeplinește de asemenea funcții speciale ale ATmega 16.

RESET - Pinul de reset. Un nivel de 0V la acest pin mai mare ca durată decât o valoare prestabilită, va genera o resetare a controllerului după restabilirea nivelului la VCC.

XTAL1 - intrare pentru oscilator și pentru circuitul intern de clock

XTAL2 - ieșire din oscilator

AVCC - AVCC este pinul de alimentare pentru portul A și pentru ADC. Trebuie conectat extern la VCC, chiar dacă ADC-ul nu este folosit. Dacă ADC-ul este folosit, trebuie conectat la VCC printr-un Filtru Trece Jos

AREF - AREF este pinul de referință analogă pentru ADC

3.2.3 PORTURILE DE INTRARE/IEȘIRE

Atunci când sunt folosite ca porturi generale digitale I/O, toate porturile au funcționalitate de citire-modificare-scriere („true Read-Modify-Write”). Aceasta înseamnă că, direcția unui port poate fi schimbată fără schimbarea neintenționată a direcției oricărui alt pin prin instrucțiunile SBI și CBI. Toți pinii porturilor au rezistență de pull-up individuală, selectabilă, cu rezerve de putere. Toți pinii I/O au diodă de protecție atât pentru Vcc cât și pentru masă, așa cum se poate vedea în figura următoare:

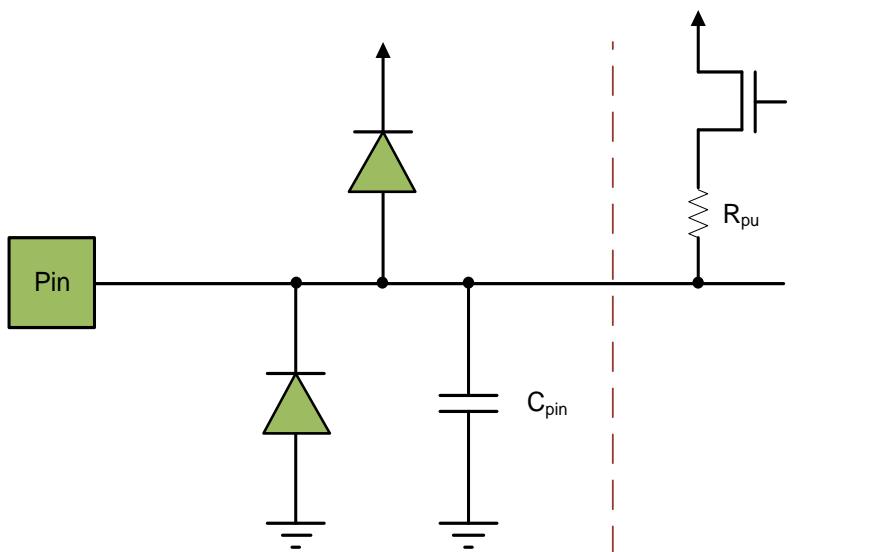


Figura 3.5 Schema echivalentă a pinilor de I/O

Porturile sunt de I/O bidirectionale cu rezistențe interne de pull-up opționale.

3.2.4 CONFIGURAREA PINILOR

Trei locații de memorie (câte 3 registre pe 8 biți) sunt alocate pentru fiecare port în parte: pentru Registrul de date (*Data Register*) – PORTx, Registrul de direcție a datelor (*Data Direction Register*) – DDRx și Registrul de intrare (*Port Input Pins*) – PINx. Locația registrului PINx poate fi doar citită nu și scrisă, în timp ce DDRx și PORTx pot fi atât scrise cât și citite. Pentru a dezactiva rezistența de *pull-up* trebuie setat (1 logic) bitul PUD (*Pull-up Disable*) din registrul SFIOR.

Notația generică pentru acești regiștri este DDxn, PORTxn, și PINxn, unde “xn” se referă la denumirea pinului respectiv (“x” reprezintă numărul literei portului (A, B, C, D) și “n” reprezintă numărul bit-ului (1-8)). La utilizarea regiștrilor și bițiilor într-un program, trebuie precizată forma exactă, de exemplu PORTB3 pentru bitul nr. 3 din portul PORTB. Biții DDxn se găsesc la adresa registrului DDRx, biții PORTxn se găsesc la adresa registrului PORTx, iar biții PINxn se găsesc la adresa registrului PINx.

DDRx (Data Direction Register) setează direcția pinilor, și anume dacă aceștia sunt de intrare sau de ieșire. Scrierea valorii de ‘0’ pe un bit din DDRx face ca pinul corespunzător din portul x să fie pin de intrare, iar scrierea valorii de ‘1’ îl setează ca pin de ieșire. Implicit toți pinii sunt pini de intrare. De exemplu:

- pentru a seta toți pinii portului A pini de intrare vom scrie:

DDRA = 0x00;

- pentru a seta toți pinii portului A pini de ieșire vom scrie:

DDRA = 0xFF;

- pentru a seta pinii 0, 4, 5 și 7 a portului B ca pini de ieșire vom scrie:

DDRB = 0xB1;

PORTx este folosit în două scopuri: pentru a asigna valori pinilor de ieșire (low sau high) și pentru a activa și dezactiva rezistența de pull-up pentru pinii de intrare. Dacă PORTxn este scris ca 1 logic atunci când pinul este configurat ca pin de intrare, rezistența de pull-up este activată. Pentru a dezactiva rezistența de pull-up, PORTxn trebuie resetat (pus pe 0 logic), sau pinul trebuie să fie configurat ca pin de ieșire. Pinii porturilor au valoarea HiZ atunci când este activată o condiție de reset, chiar dacă nu este nici un clock activ. Dacă PORTxn este scris ca 1 logic, atunci când pinul este configurat ca pin de ieșire, pinul portului este trecut în 1 logic. Dacă PORTxn este scris ca 0 logic atunci când pinul este configurat ca pin de ieșire pinul portului este trecut în 0 logic. Rolul acestui registru este astfel în legătură cu registrul DDRx. De exemplu:

- pentru a seta pinii 0, 4, 5 și 7 a portului B ca pini de ieșire cu valori de 1 logic se va scrie:

DDRB = 0xB1; //se setează pinii ca ieșiri

PORTB = 0xB1; //se setează pinii cu valori de 1 logic

- pentru a seta pinii 0 și 2 a portului A ca pini de ieșire cu valori de 1 logic se va scrie:

DDRA = 0x03; // se setează pinii ca ieșiri

PORTA = 0x03; // se setează pinii cu valori de 1 logic

PINx este folosit pentru a citi valorile pinilor de intrare. Astfel după ce se setează pinii portului x ca pini de intrare, se pot afla valorile intrărilor citind registrul PINx. De exemplu:

- pentru a citi valorile pinilor de la portul C se poate scrie:

DDRC = 0x00; // se setează pinii portului C ca pini de intrare
 x = PINC; //se copiază valorile de intrare ale pinilor portului C în variabila x
 Următorul tabel arată starea unui pin pentru diferite combinații de configurare:

DDxn	PORTxn	PUD (în SFIOR)	I/O	Pull-up	Descriere
0	0	x	Input	Nu	Tri-state (Hi-Z)
0	1	0	Input	Da	Va trece curent prin pin doar dacă în exterior avem 0 logic.
0	1	1	Input	Nu	Tri-state (Hi-Z)
1	0	x	Output	Nu	Ieșirea va fi 0 logic
1	1	x	Output	Nu	Ieșirea va fi 1 logic

Figura 3.6 Combinații de configurare

3.2.5 CITIREA VALORILOR PINILOR

Independent de setările bitului DDxn, portul pinului poate fi citit prin PINxn Register bit. Din figura se arată că PINxn Register bit, împreună cu circuitul latch ce îl precede, constituie un sincronizator. Acesta introduce o întârziere dacă pinul fizic își schimbă valoarea aproape de maximul ceasului intern. Figura următoare prezintă o diagramă de timp a sincronizării atunci când se citește o solicitare externă a valorii unui pin. Maximul și minimul propagării unei întârzieri sunt indicate de $t_{pd,max}$ și $t_{pd,min}$.

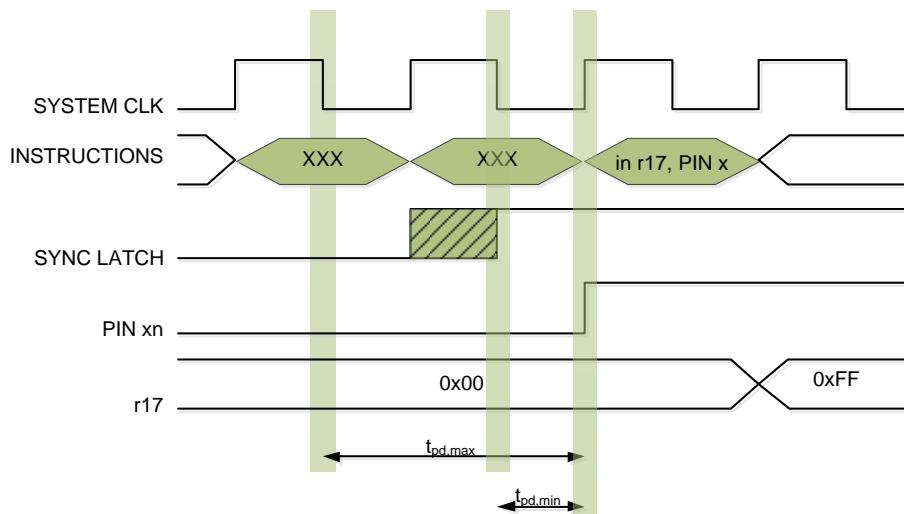


Figura 3.7 Sincronizarea în momentul citirii unei valori externe

Se consideră perioada ceasului începând de la prima front negativ al clock-ului sistemului. Latch-ul este închis atunci când clock-ul este la un nivel scăzut și funcționează normal la un nivel ridicat aşa cum se indică în partea hașurată a regiunii "SYNC LATCH" a semnalului. Valoarea semnalului este schimbată atunci când mecanismul clock-ului funcționează la un nivel scăzut. Fiecare succesiune pozitivă a clock-ului se contorizează în registrul PINxn.

Cele două săgeți $t_{pd,max}$ și $t_{pd,min}$ indică o singură tranziție a semnalului asupra pinului ce va fi întârziată între $\frac{1}{2}$ și $1\frac{1}{2}$ din perioada timpului impus. La citirea valorii pinului trebuie executată instrucția 'nop' aşa cum se arată în figura de mai jos. Instrucția 'out' setează "SYNC LATCH" pe partea pozitivă a ceasului. În acest caz, întârzierea t_{pd} ce trece prin sincronizator este de o perioadă.

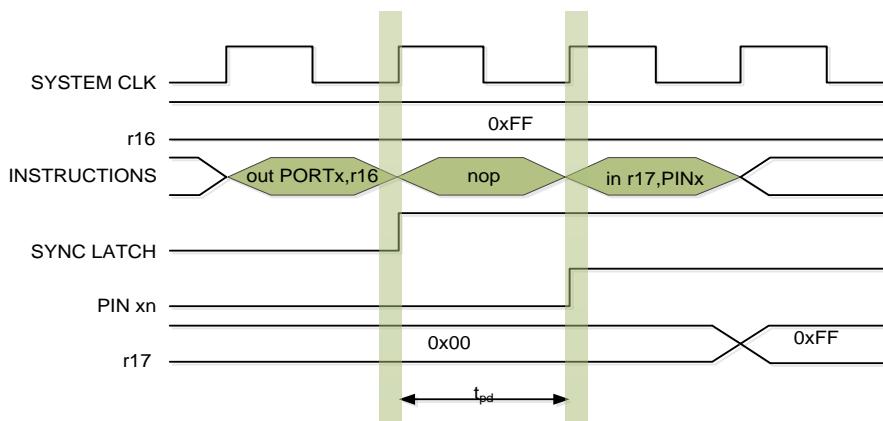


Figura 3.8 Sincronizarea software

Următorul exemplu de cod prezintă modul de setare pentru pinii 0 și 1 pe high, pinii 2 și 3 pe low din portul B; de asemenea definește pinii portului de la 4 la 7 ca intrare cu pull-ups asociate pinilor 6 și 7 ai portului. Valorile pinilor rezultate sunt citite din nou, însă pentru păstrarea valorilor precedente este necesară instrucțunea ‘nop’.

```
unsigned char i;
/*...*/
PORTB = (1<<PB7) | (1<<PB6) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
i = PINB;
/*...*/
```

Observație: În programare sunt folosite două registre temporare pentru minimizarea duratei de timp la setările pinilor 0,1,6 și 7 cu pull-up și definirea bițiilor 2 și 3 la nivel scăzut precum și redefinirea bițiilor 0 și 1 ca driver la nivel înalt.

3.2.6 FUNCȚII ALTERNATIVE ALE PORTURILOR

Majoritatea pinilor porturilor au și alte funcții în afară de faptul că sunt digitali de I/O. Semnalele de control ale pinilor pot fi suprascrise de către alte funcții, însă semnalele de suprascriere pot să nu fie prezente la toți pinii porturilor.

Tabelul următor conține funcțiile semnalelor de suprascriere. Aceste semnale sunt generate intern în modulele ce conțin funcțiile alternative.

Numele semnalului	Numele întreg	Descriere
PUOE	Pull-up Override Enable	Dacă acest semnal este setat, pull-up enable este controlat de către semnalul PUOV. Dacă acest semnal este resetat pull-up-ul este activ atunci când {DDxn, PORTxn, PUD}=0b010
PUOV	Pull-up Override Value	Dacă PUOE este setat, pull-up-ul este activ/inactiv atunci când PUOV este setat/resetat, indiferent de valorile bițiilor DDxn, PORTxn și PUD
DDOE	Data Direction Override Enable	Dacă aceste semnal este setat, Output Driver Enable este controlat de către semnalul DDOV. Dacă acest semnal este resetat Output Driver Enable este controlat de către bitul DDxn
DDOV	Data Direction Override Value	Dacă DDOE este setat Output Driver este activ/inactiv atunci când DDOV este setat/resetat, indiferent de setările bitului DDxn
PVOE	Port Value Override Enable	Dacă acest semnal este setat și Output Driver este activ, valoarea portului este controlată de către semnalul PVOV. Dacă PVOE este resetat și Output Driver este activ, valoarea portului este controlată de către bitul PORTxn

PVOV	Port Value Override Value	Dacă PVOE este setat, valoare portului este dată de PVOV, indiferent de setările bitului PORTxn
DIEOE	Digital Input Enable Override Enable	Dacă acest bit este setat, Digital Input Enable este controlat de către semnalul DIEOV. Dacă acest semnal este resetat, Digital Input Enable este determinat de starea MCU (mod normal, mod sleep)
DIEOV	Digital Input Enable Override Value	Dacă DIEOE este setat, Digital input este activ/inactiv atunci când DIEOV este setat/resetat, indiferent de starea MCU
DI	Digital Inputs	Aceasta este intrarea digitală a funcțiilor alternative
AIO	Analog Input/Output	Aceasta este intrarea/ieșirea analogă a funcțiilor alternative. Semnalul poate fi folosit bidirectional

Figura 3.9 Descrierea generică a semnalelor de suprascriere pentru alte funcții ale porturilor

3.2.6.1 Funcții alternative ale portului A

Portul A mai are și funcția de intrare analogică pentru ADC, așa cum se poate vedea în Figura 3.. Dacă unii pini ai Portului A sunt configurați ca ieșire, este esențial ca acest lucru să nu se schimbe atunci când are loc o conversie. Acest fapt ar duce la o conversie eronată.

Pinul Portului	Altă Funcție
PA7	ADC7 (canalul 7 de intrare ADC)
PA6	ADC6 (canalul 6 de intrare ADC)
PA5	ADC5 (canalul 5 de intrare ADC)
PA4	ADC4 (canalul 4 de intrare ADC)
PA3	ADC3 (canalul 3 de intrare ADC)
PA2	ADC2 (canalul 2 de intrare ADC)
PA1	ADC1 (canalul 1 de intrare ADC)
PA0	ADC0 (canalul 0 de intrare ADC)

Figura 3.10 Alte funcții ale Portului A

Figura 3. face legătura între aceste funcții ale Portului A și semnalele de suprascriere din *Figura 3..*

Numele semnalului	PA7/ ADC7	PA6/ ADC6	PA5/ ADC5	PA4/ ADC4	PA3/ ADC3	PA2/ ADC2	PA1/ ADC1	PA0/ ADC0
PUOE	0	0	0	0	0	0	0	0
PUOV	0	0	0	0	0	0	0	0
DDOE	0	0	0	0	0	0	0	0
DDOV	0	0	0	0	0	0	0	0
PVOE	0	0	0	0	0	0	0	0
PVOV	0	0	0	0	0	0	0	0
DIEOE	0	0	0	0	0	0	0	0
DIEOV	0	0	0	0	0	0	0	0
DI	-	-	-	-	-	-	-	-
AIO	Intrare ADC7	Intrare ADC6	Intrare ADC5	Intrare ADC4	Intrare ADC3	Intrare ADC2	Intrare ADC1	Intrare ADC0

Figura 3.11 Semnale de suprascriere pentru Portul A

3.2.6.2 Funcții alternative ale portului B

Funcțiile pe care le mai are Portul B, în afară de cea de General Digital I/O se pot vedea în *Figura 3..*

Pinul Portului	Altă funcție
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	\overline{SS} (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negativ Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Pozitiv Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

Figura 3.12 Alte funcții ale Portului B

Figura 3.. și *Figura 3.* fac legătura între aceste funcții ale Portului B și semnalele de suprascriere din *Figura 3..*. SPI MSTR INPUT și SPI SLAVE OUTPUT constituie semnalul MISO, în timp ce MOSI este împărțit în SPI MSTR OUTPUT și SPI SLAVE INPUT.

Numele semnalului	PB7/SCK	PB6/MISO	PB5/MOSI	PB4/ \overline{SS}
PUOE	$SPE \cdot \overline{MSTR}$	$SPE \cdot MSTR$	$SPE \cdot \overline{MSTR}$	$SPE \cdot \overline{MSTR}$
PUOV	$PB7 \cdot \overline{PUD}$	$PB6 \cdot \overline{PUD}$	$PB5 \cdot \overline{PUD}$	$PB4 \cdot \overline{PUD}$
DDOE	$SPE \cdot \overline{MSTR}$	$SPE \cdot MSTR$	$SPE \cdot \overline{MSTR}$	$SPE \cdot \overline{MSTR}$
DDOV	0	0	0	0
PVOE	$SPE \cdot MSTR$	$SPE \cdot \overline{MSTR}$	$SPE \cdot MSTR$	0
PVOV	SCK OUTPUT	SPI SLAVE OUTPUT	SPI MSTR OUTPUT	0
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	SCK INPUT	SPI MSTR INPUT	SPI SLAVE INPUT	$SPI \cdot \overline{SS}$
AI0	-	-	-	-

Figura 3.13 Semnale de suprascriere pentru Portul B

Numele Semnalului	PB3/OC0/AIN1	PB2/INT2/AIN0	PB1/T1	PB0/T0/XCK
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC0 ENABLE	0	0	UMSEL
PVOV	OC0	0	0	XCK OUTPUT
DIEOE	0	INT2 ENABLE	0	0
DIEOV	0	1	0	0
DI	-	INT2 INPUT	T1 INPUT	XCK INPUT / T0 INPUT
AIO	AIN1 INPUT	AIN0 INPUT	-	-

Figura 3.14 Semnale de suprascriere pentru Portul B

3.2.6.3 Funcții alternative ale portului C

Funcțiile pe care le mai are Portul C, în afară de cea de General Digital I/O se pot vedea în *Figura 3..*. Dacă interfața JTAG este activată, rezistență de pull-up pe pinii PC5 (TDI), PC3 (TMS) și PC2 (TCK) va fi activată, chiar dacă va avea loc un reset.

Pinul Portului	Altă Funcție
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input / Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

Figura 3.15 Alte funcții ale Portului C

Figura 3.. și *Figura 3.2* fac legătura între aceste funcții ale Portului C și semnalele de suprascriere din *Figura 3..*.

Numele Semnalului	PC7/TOSC2	PC6/TOSC1	PC5/TDI	PC4/TDO
PUOE	AS2	AS2	JTAGEN	JTAGEN
PUOV	0	0	1	0
DDOE	AS2	AS2	JTAGEN	JTAGEN
DDOV	0	0	0	SHIFT_IR + SHIFT_DR
PVOE	0	0	0	JTAGEN
PVOV	0	0	0	TDO
DIEOE	AS2	v	JTAGEN	JTAGEN
DIEOV	0	-	0	0
DI	-	-	-	-
AIO	T/C2 OSC OUTPUT	T/C2 OSC INPUT	TDI	-

Figura 3.16 Semnalele de suprascriere pentru Portul C

Numele Semnalului	PC3/TMS	PC2/TCK	PC1/SDA	PC0/SCL
PUOE	JTAGEN	JTAGEN	TWEN	TWEN
PUOV	1	1	$\overline{PC1 \cdot PUD}$	$PC0 \cdot PUD$
DDOE	JTAGEN	JTAGEN	TWEN	TWEN
DDOV	0	0	DA_OUT	SCL_OUT
PVOE	0	0	TWEN	TWEN
PVOV	0	0	0	0
DIEOE	JTAGEN	JTAGEN	0	0
DIEOV	0	0	0	0
DI	-	-	-	-
AIO	TMS	TCK	SDA INPUT	SCL INPUT

Figura 3.2 Semnalele de suprascriere pentru Portul C

3.2.6.4 Funcții alternative ale portului D

Funcțiile pe care le mai are Portul D, în afară de cea de General Digital I/O se pot vedea în *Figura 3..*

Pinul Portului	Altă funcție
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

Figura 3.18 Alte funcții ale Portului D

Figura 3. și Figura 3.3 fac legătura între aceste funcții ale Portului D și semnalele de suprascriere din Figura 3..

Numele Semnalului	PD7/OC2	PD6/ICP	PD5/OC1A	PD4/OC1B
PUOE	0	0	0	0
PUOV	0	0	0	0
DDOE	0	0	0	0
DDOV	0	0	0	0
PVOE	OC2 ENABLE	0	OC1A ENABLE	OC1B ENABLE
PVOV	OC2	0	OC1A	OC1B
DIEOE	0	0	0	0
DIEOV	0	0	0	0
DI	-	ICP INPUT	-	-
AIO	-	-	-	-

Figura 3.19 Semnalele de suprascriere pentru Portul D

Numele Semnalului	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
PUOE	0	0	TXEN	RXEN
PUOV	0	0	0	PD0•PUD
DDOE	0	0	TXEN	RXEN
DDOV	0	0	1	0
PVOE	0	0	TXEN	0
PVOV	0	0	TXD	0
Numele Semnalului	PD3/INT1	PD2/INT0	PD1/TXD	PD0/RXD
DIEOE	INT1 ENABLE	INT0 ENABLE	0	0
DIEOV	1	1	0	0
DI	INT1 INPUT	INT0 INPUT	-	RXD
AIO	-	-	-	-

Figura 3.3 Semnalele de suprascriere pentru Portul D

3.3 REZISTENȚA DE PULL-UP

3.3.1 BITUL PUD DIN SFIOR (SPECIAL FUNCTION I/O REGISTER)

Bit	7	6	5	4	3	2	1	0	SFIOR
	ADTS2	ADTS1	ADTS0	ADHSM	ACME	PUD	PSR2	PSR10	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Valoare inițială

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

Figura 3.4 Registrul SFIOR

Când acest bit este setat ‘1’, porturile I/O sunt dezactivate chiar dacă registrele PORTxn și DDxn sunt configurate pentru a le activa ($\{DDxn, PORTxn\} = 0b01$). Vezi configurarea pinilor pentru mai multe detalii despre aceste caracteristici.

3.4 PINII NECONECTAȚI

Dacă unii pini rămân neconectați, este recomandat să se asigure faptul că acești pini au valori definite. Cel mai simplu mod de a se asigura acest lucru este prin activarea rezistenței de pull-up interne. În acest caz, rezistența va fi dezactivată în momentul reset-ului. Dacă este important un consum redus de curent în momentul reset-ului, este recomandată folosirea unui pull-up sau pull-down extern. Conectarea pinilor nefolosiți direct la VCC sau la masă nu se recomandă, deoarece va produce curenți inutili dacă pinul este setat ca pin de ieșire.

3.5 FUNCȚII INTRINSECI

Funcțiile intrinseci oferă acces direct la operațiunile de procesor de nivel scăzut și pot fi foarte utile, de exemplu, în rutinele critice de timp. Funcțiile intrinseci sunt compilate în cod inline, fie ca există o singură instrucțiune sau o secvență scurtă de instrucțiuni.

Următorul tabel rezumă funcțiile intrinseci:

Funcții Intrinseci	Descriere
<code>delay_cycles</code>	Insertii de un timp de întârziere
<code>disable_interrupt</code>	Dezactivează întreruperile
<code>enable_interrupt</code>	Activează întreruperile
<code>extended_load_program_memory</code>	Returnează un octet din codul memoriei
<code>fractional_multiply_signed</code>	Generează o instrucțiune FMULS
<code>fractional_multiply_signed_with_unsigned</code>	Generează o instrucțiune FMULSU
<code>fractional_multiply_unsigned</code>	Generează o instrucțiune FMUL
<code>indirect_jump_to</code>	Generează o instrucțiune IJMP
<code>insert_opcode</code>	Atribuie o valoare într-un registru procesor
<code>load_program_memory</code>	Returnează un octet din codul memoriei
<code>multiply_signed</code>	Generează o instrucțiune MULS
<code>multiply_signed_with_unsigned</code>	Generează o instrucțiune MULSU
<code>multiply_unsigned</code>	Generează o instrucțiune MUL
<code>no_operation</code>	Generează o instrucțiune NOP
<code>sleep</code>	Introduce o instrucțiune SLEEP
<code>swap_nibbles</code>	Swap-uri bit cu bit 0-3 cu 4-7
<code>watchdog_reset</code>	Resetare watchdog

Figura 3.22 Funcții intrinseci

Secțiunea următoare oferă informații de referință despre fiecare funcție intrinsecă.

3.5.1 __DELAY_CYCLES(UNSIGNED LONG);

```
void __delay_cycles(unsigned long);
```

Această funcție generează cod care consumă exact numărul de ciclii specificat ca parametru, fără alte efecte secundare. Valoarea specificată trebuie să fie o constantă cunoscută la compilare.

Datorită faptului că procesorul se blochează când este apelată funcția, aceasta nu este o funcție optimă. Momentele în care procesorul parcurge acel cod generat sunt niște timpi nefolosiți. Pentru o optimizare a acestui proces, se pot folosi întreruperi de procesor, ce vor fi studiate într-un laborator viitor. Această funcție este preferată în instrucțiunile simple, deoarece este foarte ușor de folosit.

3.5.2 __DISABLE_INTERRUPT (VOID);

```
void __disable_interrupt(void);
```

Dezactivează întreruperile prin introducerea de instrucțiuni CLI.

3.5.3 __ENABLE_INTERRUPT (VOID);

```
void __enable_interrupt(void);
```

Activează întreruperile prin introducerea de instrucțiuni SEI.

3.5.4 __EXTENDED_LOAD_PROGRAM_MEMORY(UNSIGNED CHAR __FARFLASH *);

```
unsigned char __extended_load_program_memory(unsigned char __farflash *);
```

Returnează un octet din memoria cod. Se utilizează această funcție intrinsecă pentru accesul la datele constante din memoria cod.

3.5.5 __LOAD_PROGRAM_MEMORY(UNSIGNED CHAR __FLASH *);

```
unsigned char __load_program_memory(const unsigned char __flash *);
```

Returnează un octet din memoria cod. Constantele trebuie să fie plasate în primii 64 Kb de memorie.

3.5.6 __NO_OPERATION (VOID);

```
void __no_operation(void);
```

Generează o instrucțiune NOP.

3.5.7 __SLEEP (VOID);

```
void __sleep(void);
```

Inserții de o instrucțiune sleep. Pentru a utiliza această funcție intrinsecă, trebuie asigurat faptul că instrucțiunea a fost activată în registrul MCUCR.

3.5.8 __SWAP_NIBBLES (UNSIGNED CHAR);

```
unsigned char __swap_nibbles(unsigned char);
```

Această funcție face bit swap-urile biților 0-3 cu biții 4-7 din parametrul specificat și returnează valoarea interschimbată.

3.5.9 __WATCHDOG_RESET (VOID);

```
void __watchdog_reset(void);
```

Inserții de o instrucțiune de resetare watchdog.

3.6 APLICAȚII

Enunț

Aprinderea unui led o dată pe secundă (frecvența de 1Hz, perioadă ON de 0.5s, perioadă OFF de 0.5s).

Rezolvare

Trebuie urmată o secvență logică de pași:

1. trebuie incluse bibliotecile de lucru cu simbolurile, constantele și funcțiile intrinseci:

```
#include <inavr.h>
#include <ioavr.h>
```

2. trebuie setată direcția pinului. Așa cum a fost menționat în laborator, implicit toți pinii sunt pini de intrare. Se va seta pinul ales (de exemplu, PD1 din portul D) ca pin de ieșire prin următoarea instrucțiune: `DDRD = (1<<PD1);`

3. se vor realiza operații de toggle asupra portului de ieșire selectat. Astfel, pentru operația de aprindere și stingere succesivă a led-ului, va trebui scrisă valoarea de ‘1’ logic, respectiv ‘0’ logic pe portul de ieșire (PORTD=0), cu o anumită frecvență; acea frecvență va fi generată cu ajutorul funcției prezentate mai sus:

`_delay_cycles();` Parametrul trimis funcției va fi calculat în funcție de frecvența oscilatorului intern (în acest caz, 4Mhz).

Cum calculăm valoarea transmisă ca parametru pentru `delay_cycles()`:

- frecvența oscilatorului intern este de 4Mhz (teoretic)
- calculăm durata unui ciclu de instrucțiune:

$$\begin{array}{l} \text{1 ciclu.....} 1/4\text{Mhz}=0,25 \mu\text{s} \\ \times \text{ ciclii.....} 0,5\text{s} \\ \hline x=0,5\text{s}/0,25\mu\text{s}=2000000 \text{ ciclii} \end{array}$$

4. aşa cum a fost prezentată în prima parte a laboratorului, conectarea led-ului în cadrul circuitului este esențială pentru funcționarea acestuia. Astfel, “minusul” led-ului va fi conectat la unul din pinii de GND (masă) ai microcontroler-ului, iar “plusul” led-ului va fi conectat, prin intermediul unei rezistențe de valoare ce va fi calculată cu formula dată, la pin-ul setat ca ieșire (PD1).

Cod sursă

```
#include <inavr.h> //această bibliotecă conține prototipurile
                    funcțiilor delay_cycles(),
                    enable_interrupt(), disable_interrupt() etc.
#include <ioavr.h> //conține definițiile funcțiilor pentru
                    input/output

int main(void)
{
    DDRD = (1<<PD1); //setează pinul PD1 ca pin de ieșire
    while(1)
    {
        PORTD=2;// se setează valoarea ‘1’ logic pe pinul de ieșire
                  //2(DEC)=0b00000010, ceea ce setează pinul PD1 (al
                  doilea pin din portul D) cu valoarea de ‘1’ logic
                  //PORTD=0x02; //echivalentul în hexa
        _delay_cycles(2000000); //numărul de cicli echivalenți pentru
                               0,5s
        PORTD=0;// setăm toți pinii portului D pe ’0’ logic, inclusiv
                  pinul PD1. Avem astfel 0 Volți la ieșirea acestuia ceea
                  ce va determina stingerea ledului. În acest mod se
                  realizează operația de toggle(inchis/deschis)
        _delay_cycles(2000000);
    }
    return 0;
}
```

4 Comunicația serială

4.1 INTRODUCERE

Transmisia digitală de date a evoluat de la conexiunea între un calculator și echipamentele periferice, la calculatoare care comunică în rețele internaționale complexe. Însă sunt multe de învățat pornind de la simpla legătură punct la punct sau RS232 după standardul EIA. Cu toate că transferul paralel este mai rapid, majoritatea transmisiilor de date între calculatoare sunt făcute pe cale serială pentru a reduce costul cablurilor și conectorilor. Există și limitări fizice de distanță, care nu pot fi depășite de magistralele paralele. În comunicația serială, datele sunt transmise bit cu bit.

Toate comunicațiile sunt caracterizate de trei elemente principale:

- Date - înțelegerea lor, scheme de codificare, cantitate;
- Temporizări - sincronizarea între receptor și emițător, frecvență și fază;
- Semnale - tratarea erorilor, controlul fluxului și rutarea interfețelor seriale.

4.2 MODELUL COMUNICAȚIEI SERIALE

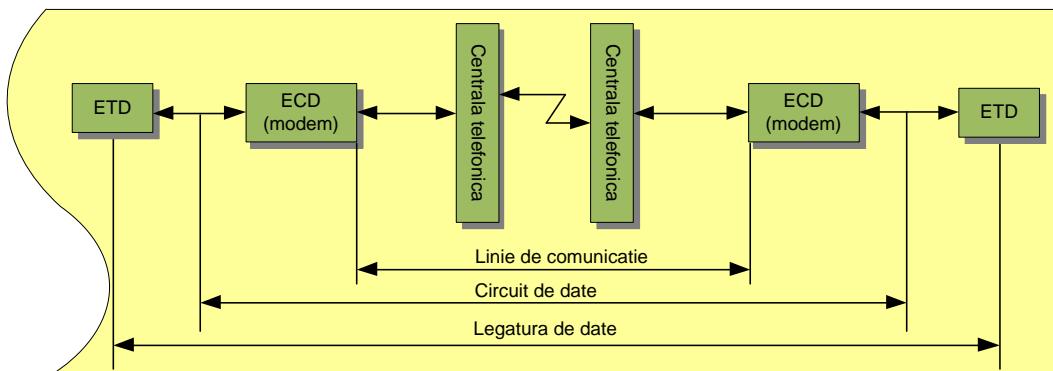


Figura 4.1 Sistem de comunicație serială

Componentele unui sistem de comunicație serială sunt următoarele:

1. ETD (Echipamente terminale de date: calculatoare, terminale de date). Acestea conțin și interfețele seriale sau controlerile de comunicație;
2. ECD (Echipamente pentru comunicația de date). Aceste echipamente se numesc modeme și permit calculatorului să transmită informații printr-o linie telefonică analogică. Funcțiile principale realizate de un modem sunt următoarele:
 - Conversia digital/analogică a informațiilor din calculator și conversia analog/digitală a semnalelor de pe linia telefonică analogică.

- Modularea/demodularea unui semnal purtător. La transmisie, modemul suprapune (modulează) semnalele digitale ale calculatorului peste semnalul purtător al liniei telefonice. La recepție, modemul extrage (demodulează) informațiile transportate de semnalul purtător și le transferă calculatorului;
3. Linia de comunicație reprezintă o linie fizică sau o linie telefonică. Linia telefonică poate fi o linie comutată (conectată la o centrală telefonică) sau o linie închiriată (dedicată);
 4. Circuitul de date cuprinde porțiunea dintre două echipamente terminale de date, modem-urile și linia de comunicație. Pe distanțe reduse, este posibilă comunicația serială directă între două echipamente terminale de date prin linii fizice, fără utilizarea unor modem-uri. În acest caz, circuitul de date este reprezentat de aceste linii;
 5. Legătura de date conține circuitul de date și interfețele seriale ale echipamentelor terminale de date.

În funcție de numărul de echipamente interconectate, o legătură serială poate fi **punct la punct** (două echipamente) sau *multi-punct* (mai mult de două echipamente).

4.3 TIPURI DE COMUNICAȚIE SERIALĂ

Din punctul de vedere al direcției de transfer, se pot distinge următoarele tipuri de comunicație serială:

- Simplex;
- Semiduplex;
- Duplex.

În cazul comunicației *simplex*, datele sunt transferate întotdeauna în același direcție, de la echipamentul transmițător la cel receptor. La comunicația *semiduplex*, fiecare echipament terminal de date funcționează alternativ ca transmițător, iar apoi ca receptor. Pentru acest tip de conexiune, este suficientă o singură linie de transmisie (două fire de legătură). Într-o comunicație *duplex* (numită și *duplex integral*), datele se transferă simultan în ambele direcții. Primele conexiuni duplex necesitau două linii de transmisie (patru fire de legătură), dar conexiunile ulterioare necesită o singură linie.

Din punctul de vedere al sincronizării dintre transmițător și receptor, există două tipuri de comunicație serială:

- Asincronă;
- Sincronă.

4.3.1 COMUNICAȚIA ASINCRONĂ

Pentru a asigura sincronizarea dintre transmițător și receptor, fiecare caracter transmis este precedat de un bit de START, cu valoarea logică 0 (“space”) și este urmat de cel puțin un bit de STOP, cu valoarea logică 1 (“mark”). Biții de START și de STOP încadrează fiecare caracter transmis; caracterul transmis între acești doi biți reprezintă un cadru de date. Un asemenea cadru reprezintă informația digitală de bază într-un sistem de comunicație serială. În cazul comunicației asincrone, intervalul de timp între transmisia a două caractere succesive este variabil, pe durata

acestui interval linia de comunicație fiind în starea 1 logic. Acest mod de comunicație este numit și start-stop.

Sincronizarea la nivel de bit se realizează cu ajutorul semnalelor de ceas locale cu aceeași frecvență. Atunci când receptorul detectează începutul unui caracter indicat prin bitul de START, pornește un oscilator de ceas local, care permite eșantionarea corectă a bițiilor individuali ai caracterului. Eșantionarea bițiilor se realizează aproximativ la mijlocul intervalului corespunzător fiecarui bit.

Figura 4.2 ilustrează transmisia caracterului cu codul ASCII 0x61. După bitul de START, având durată T corespunzătoare unui bit, transmisia caracterului începe cu bitul cel mai puțin semnificativ b0. După transmisia bitului cel mai semnificativ b7, se transmite un bit de paritate p; în acest exemplu, paritatea este impară. Bitul de paritate este optional, iar în cazul în care se adaugă la caracterul transmis, paritatea poate fi selectată pentru a fi pară sau impară. Există și posibilitatea ca bitul de paritate să fie setat la 0 sau 1, indiferent de paritatea efectivă a caracterului. În exemplul ilustrat, la sfârșitul caracterului se transmit doi biți de STOP s1 și s2, după care linia rămâne în starea 1 logic un timp nedefinit. Acest timp corespunde unui interval de pauză.

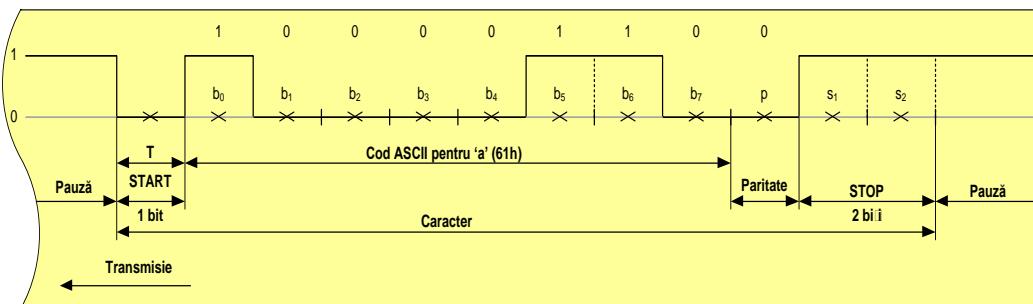


Figura 4.2 Comunicația asincronă

În cazul comunicației asincrone, sincronizarea la nivel de bit este asigurată numai pe durata transmisiei efective a fiecarui caracter. O asemenea comunicație este orientată pe caractere individuale și are dezavantajul că necesită informații suplimentare în proporție de cel puțin 25% pentru identificarea fiecarui caracter.

4.3.2 COMUNICAȚIA SINCRONĂ

În cazul comunicației sincrone, un cadru nu conține un singur caracter, ci un bloc de caractere sau un mesaj. Sincronizarea la nivel de bit trebuie asigurată permanent, nu numai în timpul transmisiei propriu-zise, ci și în intervalele de pauză. De aceea, timpul este divizat în mod continuu în intervale elementare la transmițător, intervale care trebuie regăsite apoi la receptor. Aceasta pune anumite probleme. Dacă ceasul local al receptorului are o frecvență care diferă într-o anumită măsură de frecvența transmițătorului, vor apărea erori la recunoașterea caracterelor, din cauza lungimii blocurilor de caractere.

Pentru a se evita asemenea erori, ceasul receptorului trebuie resincronizat frecvent cu cel al transmițătorului. Aceasta se poate realiza dacă se asigură că există suficiente tranziții de la 1 la 0 și de la 0 la 1 în mesajul transmis. Dacă datele de transmis constau din siruri lungi de 1 sau de 0, trebuie inserate tranziții suficiente pentru resincronizarea ceasurilor. Asemenea tehnici sunt dificil de implementat, astfel încât se utilizează de obicei o tehnică numită comunicație asincronă sincronizată (numită în mod simplu comunicație sincronă).

Acest tip de comunicație este caracterizat de faptul că, deși mesajul este transmis într-un mod sincron, nu există o sincronizare în intervalul de timp dintre două mesaje. Informația este

transmisă sub forma unor blocuri de caractere sau a unor biți succesivi, fără biți de START și STOP. Pentru ajustarea oscillatorului local la începutul unui mesaj, fiecare mesaj este precedat de un număr de caractere speciale de sincronizare, de exemplu, caracterul SYN (0x16). Pentru menținerea sincronizării, se pot insera caractere de sincronizare suplimentare în mesajul transmis, la anumite intervale de timp.

La receptor există trei nivele de sincronizare:

- Sincronizare la nivel de bit, utilizând circuite cu calare de fază PLL (Phase-Locked Loop), pe baza tranzițiilor existente în semnalul recepționat;
- Sincronizare la nivel de caracter, asigurată prin recunoașterea unui caracter de sincronizare;
- Sincronizare la nivel de bloc sau mesaj, care depinde de protocolul de date utilizat.

4.3.3 STANDARDUL RS-232C

Specificațiile electrice ale portului serial au fost definite în standardul RS-232C (Reference Standard No. 232, Revision C), elaborat în anul 1969 de către Comitetul de Standarde din SUA, cunoscut azi sub numele de Asociația Industriei Electronice (EIA – Electronic Industries Association). Standardul a fost elaborat pentru comunicația digitală între un calculator și un terminal aflat la distanță sau între două terminale fără utilizarea unui calculator. Terminalele erau conectate prin linii telefonice, astfel încât erau necesare modem-uri la ambele capete ale liniei de comunicație.

Standardul RS-232C a suferit diferite modificări, fiind elaborate mai multe revizii ale acestuia. De exemplu, în anul 1987 a fost elaborată o nouă revizie a standardului, numită EIA RS-232D. În anul 1991, EIA și Asociația Industriei de Telecomunicații (TIA – Telecommunications Industry Association) au elaborat revizia E a standardului (EIA/TIA RS-232E). Revizia curentă este EIA RS-232F, publicată în anul 1997. Totuși, indiferent de revizia acestuia, standardul este numit de cele mai multe ori RS-232C sau RS-232.

În Europa, versiunea echivalentă standardului RS-232C este V.24, elaborată de comitetul CCITT (*Comité Consultatif International pour Téléphonie et Télégraphie*). Denumirea acestui comitet a fost schimbată la începutul anilor 1990 în *International Telecommunications Union* (ITU). Ambele standarde specifică semnalele utilizate pentru comunicație, nivelele de tensiune, protocolul utilizat pentru controlul fluxului de date și conectorii interfeței seriale.

Standardul RS-232C definește atât o comunicație asincronă, cât și una sincronă. Nu sunt definite detalii cum sunt codificarea caracterelor (ASCII, Baudot, EBCDIC), încadrarea caracterelor (lungimea caracterului, numărul bițiilor de stop, paritatea) și nici vitezele de comunicație, deși standardul este destinat pentru viteze mai mici de 20.000 biți/s. Echipamentele actuale permit însă viteze superioare de comunicație, utilizând nivele de tensiune care sunt compatibile cu cele specificate de standard. Porturile seriale ale calculatoarelor permit, de obicei, selecția uneia din următoarele viteze de comunicație: 150; 300; 600; 1.200; 2.400; 4.800; 9.600; 19.200; 38.400; 57.600; 115.200 biți/s.

O legătură de bază RS-232C necesită doar trei conexiuni: una pentru transmisie, una pentru recepție și una pentru masa electrică comună. Cele mai multe legături seriale utilizează însă și semnale pentru controlul fluxului de date.

Spre deosebire de alte tipuri de comunicație serială care sunt diferențiale, comunicația RS-232C este una obișnuită, utilizând câte un fir pentru fiecare semnal. Deși astfel se simplifică

circuitele necesare interfeței, în același timp se reduce și distanța maximă de comunicație în cazul unei legături directe, fără utilizarea modem-urilor. Standardul RS-232C specifică o distanță maximă de 15 m. Distanța poate fi mărită dacă se utilizează viteze de comunicație mai reduse. Tensiunile electrice specificate de standardul RS-232C sunt următoarele:

- Valoarea logică 0 corespunde unei tensiuni pozitive între +3 V și +25 V;
- Valoarea logică 1 corespunde unei tensiuni negative între -3 V și -25 V.

4.4 CONTROLUL FLUXULUI DE DATE

Pentru a fi posibilă comunicația între dispozitive cu viteze diferite, proiectanții interfeței seriale au prevăzut semnale speciale pentru controlul fluxului de date. Aceste semnale permit unui echipament oprirea și apoi reluarea transmiterii datelor la cererea echipamentului de la celălalt capăt al liniei de comunicație serială. Pe lângă această *metodă hardware* pentru controlul fluxului de date, există și o *metodă software*, bazată pe transmiterea unor caractere speciale între cele două echipamente. Atunci când echipamentul receptor (de exemplu, o imprimantă) nu mai poate primi date deoarece bufferul acestuia este plin, transmite un anumit caracter de control echipamentului transmițător (de exemplu, calculatorului). Atunci când echipamentul receptor poate primi noi date, transmite un alt caracter de control care semnalează echipamentului transmițător că poate relua transmiterea datelor.

De obicei, metoda de control care va fi utilizată de calculator poate fi selectată prin intermediul driver-ului software al controlerului serial. Unele programe pot utiliza în mod implicit o anumită metodă. În cazul perifericelor, metoda de control poate fi selectată fie prin program, fie printr-un comutator. Este important să se utilizeze aceeași metodă de control atât pentru calculator, cât și pentru periferic pentru a evita pierderile de date.

4.4.1 METODA DE CONTROL HARDWARE

Metoda de control hardware presupune utilizarea unui protocol de comunicație cu ajutorul semnalelor de control ale interfeței seriale. Protocolul utilizat se bazează pe comunicația serială prin intermediul unor modem-uri și a unei linii telefonice, pentru care a fost elaborată interfața serială originală. Acest protocol implică stabilirea conexiunii între două modem-uri prin linia telefonică și menținerea fluxului de date dintre acestea cât timp conexiunea este activă. Etapele acestui protocol sunt descrise în continuare. Într-o formă simplificată, acest protocol este utilizat și în cazul comunicației seriale directe între două echipamente, fără utilizarea unor modem-uri și a unei linii telefonice.

1. Atunci când un modem aflat la distanță dorește stabilirea conexiunii cu modemul local, transmite semnalul de apel pe linia telefonică. Acest semnal este detectat de către modemul local, care activează semnalul *RI* pentru a informa calculatorul local asupra existenței unui apel telefonic.
2. La detectarea activării semnalului *RI*, pe calculatorul local se lansează în execuție un program de comunicație. Acest program indică disponibilitatea calculatorului de a începe comunicația prin activarea semnalului *DTR*.
3. Atunci când modemul local sesizează faptul că terminalul de date (calculatorul) este pregătit, răspunde la apelul telefonic și așteaptă activarea semnalului purtător de către modemul aflat la distanță. Atunci când modemul local detectează semnalul purtător, activează semnalul *CD*.
4. Modemul local negociază cu modemul aflat la distanță o conexiune cu anumiți parametri. De exemplu, cele două modem-uri pot determina viteza optimă de

- comunicație în funcție de calitatea legăturii telefonice. După această negociere, modemul local activează semnalul *DSR*.
5. La sesizarea activării semnalului *DSR*, programul de pe calculatorul local activează semnalul *RTS* pentru a indica modemului că poate transmite date către calculator.
 6. Atunci când modemul sesizează activarea semnalului *RTS*, activează semnalul *CTS* pentru a indica faptul că este pregătit pentru receptia datelor de la calculator.
 7. În continuare, datele sunt transferate în ambele sensuri între echipamentele aflate la distanță, pe liniile *TD* și *RD*.
 8. Deoarece viteza liniei telefonice este mai redusă decât cea a legăturii dintre calculator și modemul local, bufferul modemului se va umple. Modemul local solicită calculatorului oprirea transmiterii datelor prin dezactivarea semnalului *CTS*. La golirea bufferului, modemul reactivează semnalul *CTS*.
 9. În cazul în care calculatorul nu mai poate primi date de la modem, dezactivează semnalul *RTS*. Atunci când calculatorul poate primi din nou date de la modem, reactivează semnalul *RTS*.
 10. La încheierea sesiunii de comunicație, semnalul purtător este dezactivat, iar modemul local dezactivează semnalele *CD*, *CTS* și *DSR*.
 11. Atunci când sesizează dezactivarea semnalului *CD*, calculatorul local dezactivează semnalele *RTS* și *DTR*.

Din protocolul descris mai sus, rezultă următoarele:

- Calculatorul trebuie să detecteze activarea semnalelor *DSR* și *CTS* înainte de a transmite date către modem. Dezactivarea oricărui din aceste semnale va opri, de obicei, fluxul de date de la calculator;
- Modemul trebuie să detecteze activarea semnalelor *DTR* și *RTS* înainte de a transmite date pe linia serială sau către calculator. Dezactivarea semnalului *DTR* va opri transmiterea datelor pe linia serială, iar dezactivarea semnalului *RTS* va opri transmiterea datelor către calculator;
- Starea semnalului *CD* nu este interpretată de toate sistemele de comunicație serială. La anumite sisteme, semnalul *CD* trebuie să fie activat înainte ca terminalul de date să înceapă transmiterea datelor. La alte sisteme, starea semnalului *CD* este ignorată.

4.4.2 METODA DE CONTROL SOFTWARE

Metoda software pentru controlul fluxului de date presupune transmiterea unor caractere de control între cele două echipamente. De exemplu, perifericul va transmite un anumit caracter de control pentru a indica faptul că nu mai poate primi date de la calculator și va transmite un alt caracter de control pentru a indica faptul că transmiterea datelor poate fi reluată de calculator. Există două variante ale acestei metode. Prima variantă utilizează caracterele de control XON/XOFF, iar a doua variantă utilizează caracterele de control ETX/ACK.

În cazul utilizării variantei XON/XOFF, perifericul transmite caracterul XOFF pentru a indica faptul că bufferul său este plin și transmiterea datelor trebuie oprită de calculator. Acest caracter mai este denumit DC1 (*Device Control 1*) și are codul ASCII 0x13, fiind echivalent cu caracterul Ctrl-S. Caracterul Ctrl-S poate fi introdus și de utilizator la anumite programe de comunicație pentru a opri transmiterea datelor de către un echipament cu care este conectat calculatorul. Atunci când perifericul este pregătit pentru a primi noi date, transmite calculatorului caracterul XON. Acest

caracter mai este denumit DC3 (*Device Control 3*) și are codul ASCII 0x11, fiind echivalent cu caracterul Ctrl-Q. La anumite programe de comunicație, introducerea caracterului Ctrl-Q anulează efectul caracterului Ctrl-S.

În cazul utilizării variantei ETX/ACK, transmiterea caracterului ETX (*End of Text*) de către periferic indică faptul că transmiterea datelor trebuie oprită de calculator. Acest caracter are codul ASCII 0x03 și este echivalent cu caracterul Ctrl-C. Transmiterea caracterului ACK (*ACKnowledge*) indică posibilitatea reluării transmiterii datelor de către calculator. Acest caracter are codul ASCII 0x06 și este echivalent cu caracterul Ctrl-F.

4.5 CONECTORI

Porturile seriale pot utiliza unul din două tipuri de conectori. Conectorul DB-25 cu 25 de pini a fost utilizat la calculatoarele din generațiile anterioare. La calculatoarele mai noi se utilizează conectorul DB-9 cu 9 pini. Pentru porturile seriale ale calculatoarelor se utilizează conectori tata, iar pentru porturile seriale ale echipamentelor periferice se utilizează conectori mamă.

Conectorul DB-25 al portului serial are o formă similară cu conectorul DB-25 al portului paralel. Portul serial care utilizează un conector DB-25 se poate deosebi de portul paralel prin faptul că pentru portul serial se utilizează un conector tata, în timp ce pentru portul paralel se utilizează un conector mamă. *Figura 4.3* ilustrează conectorul DB-25 al portului serial.



Figura 4.3 Conectorul DB-25

Din cele 25 de semnale ale conectorului DB-25, se utilizează cel mult 10 semnale pentru o conexiune serială obișnuită. *Tabel 4.1*

indică numele acestor semnale și asignarea lor la pinii conectorului DB-25.

Pin	Semnal	Semnificație	In Out
1	PG	Protective Ground	←
2	TD	Transmit Data	←
3	RD	Receive Data	→
4	RTS	Request To Send	→
5	CTS	Clear To Send	
6	DSR	Data Set Ready	
7	SG	Signal Ground	←
8	CD	Carrier Detect	→
20	DTR	Data Terminal Ready	←
22	RI	Ring Indicator	←

Tabel 4.1

Pentru a se reduce spațiul ocupat de conectorul portului serial, conectorul DB-25 a fost înlocuit cu un conector de dimensiuni mai reduse, conectorul cu 9 pini DB-9 (Figura 4.4).

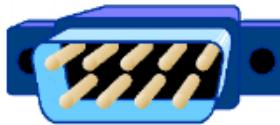


Figura 4.4 Conectorul DB-9

4.6 CABLURI

Există mai multe variante de cabluri care se pot utiliza pentru comunicația serială. Pentru viteze de comunicație reduse și lungimi scurte, se pot utiliza cabluri obișnuite, care nu sunt ecranate. Pentru a reduce interferențele cu alte echipamente, trebuie utilizate cabluri ecranate care conțin un înveliș sub formă unei folii de aluminiu. În mod ideal, ecranul cablului trebuie conectat la masa de protecție a conectorului, dacă acesta este de tip DB-25. Conectorul DB-9 nu conține un pin pentru masa de protecție. În cazul utilizării conectorilor de acest tip, ecranul cablului se poate conecta la masa electrică.

Observații:

În cazul cablului serial care utilizează conectori DB-25, masa electrică sau masa de semnal *SG* (*Signal Ground*) este separată de masa mecanică sau masa de protecție *PG* (*Protective Ground*). Masa de protecție este conectată direct la carcasa conectorului (și a echipamentului), având un rol de protecție. Prin realizarea acestei conexiuni, carcasele metalice ale celor două echipamente conectate prin cablul serial se vor afla la același potențial, evitându-se formarea unor diferențe de tensiune între cele două echipamente, tensiuni care pot fi periculoase pentru acestea. Deseori, conexiunea mesei de protecție lipsește din cablurile seriale.

Masa de protecție *PG* nu trebuie conectată niciodată la masa electrică *SG*.

Semnalele interfeței seriale au fost prevăzute în scopul conectării unui echipament terminal de date (ETD) la un echipament pentru comunicația de date (ECD). Atunci când se conectează două asemenea echipamente, de exemplu, un calculator cu un modem, care dispun de conectori de același tip (de exemplu, DB-25), este necesar un cablu care conectează pinii cu același număr ai conectorilor de la cele două capete. Acesta este un *cablu direct*. Dacă se conectează două echipamente cu conectori diferiți, este necesar un *cablu adaptor*. Dacă se conectează două echipamente terminale de date, de exemplu, două calculatoare, datele transmise pe pinul *TD* al unui echipament trebuie recepționate pe pinul *RD* al celuilalt echipament. De aceea, conexiunile acestor pini trebuie inversate la cele două capete ale cablului; un asemenea cablu este numit *cablu inversor*.

4.7 MODULUL USART

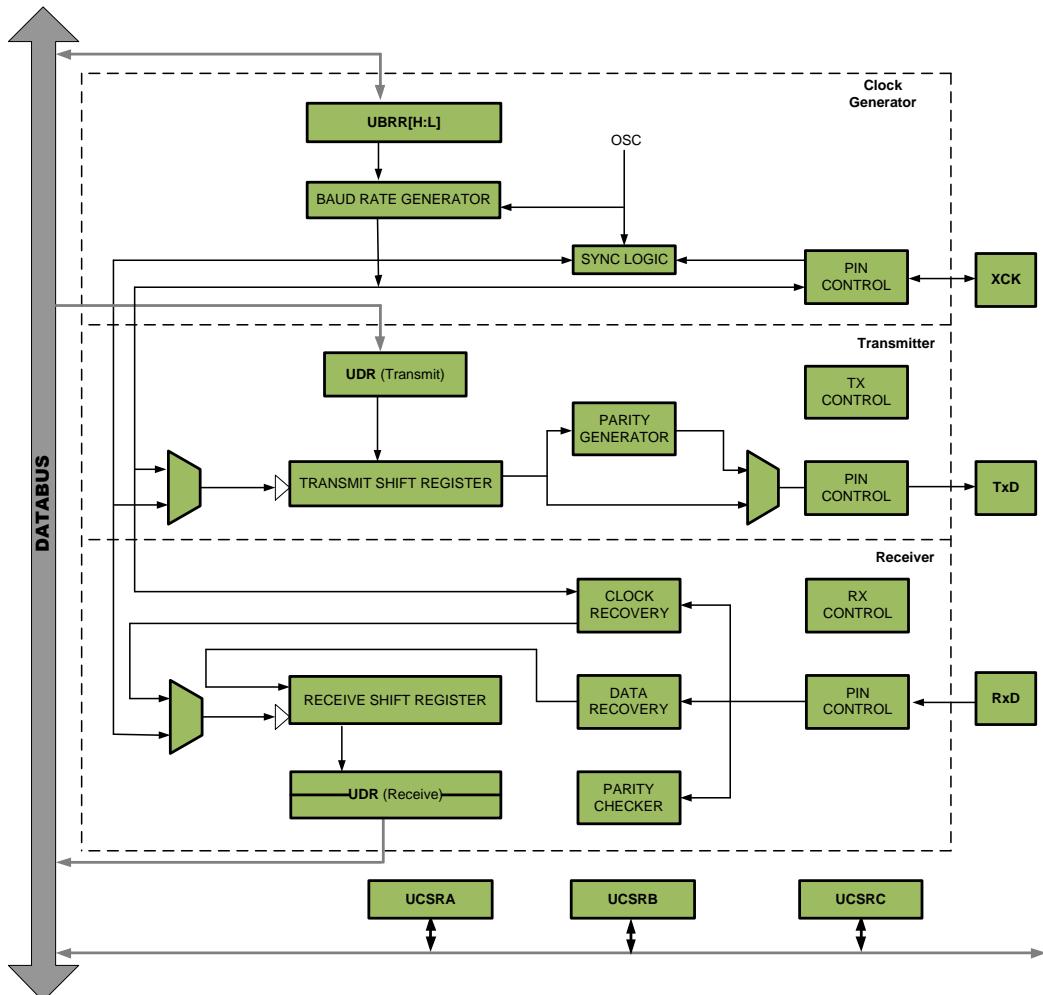


Figura 4.3 USART Block Diagram

ATMega16 dispune de trei subsisteme pentru comunicația serială:

1. Universal Synchronous & Asynchronous Serial Receiver & Transmitter (USART);
2. Serial Peripheral Interface (SPI);
3. Two-wire Serial Interface (TWI).

4.7.1 MODULUL USART

Componenta principală a unui port serial este un circuit UART (*Universal Asynchronous Receiver/Transmitter*). Acest circuit realizează conversia datelor paralele de la calculator în formatul necesar pentru transmisia serială și conversia datelor seriale recepționate în formatul paralel utilizat de calculator. Circuitul adaugă bitul de start, bitul de stop și bitul de paritate la datele seriale transmise și detectează acești biți în cadrul datelor seriale recepționate.

Modulul UART (Universal Asynchronous Receiver Transmitter) efectuează receptia/transmisia datelor de la/către un dispozitiv periferic cu acces serie. Principalele caracteristici sunt:

- Funcționare full-duplex completă atât în mod sincron cât și în mod asincron;
- Posedă generator de rată de baud propriu de rezoluție mare;
- Formate de date seriale diverse;
- Detectează automat erorile de transmisie;
- Execută comunicații de tip multiprocesor;

Modulul execută conversia serie/paralel a datelor la recepție, respectiv conversia paralel/serie la transmisie. Transmiterea datelor este inițializată prin scrierea datelor care trebuie transmise în registrul UDR (UART Data Register). Datele sunt transferate de la UDR la registrul Transmit Shift când:

- Un nou caracter este scris în UDR și caracterul precedent a fost deja transferat. Registrul de deplasare este încărcat imediat;
- Un nou caracter este scris în UDR înainte ca un caracter precedent să fi fost transferat complet. Registrul de deplasare este încărcat după ce prima operație a fost finalizată.

4.7.2 INTERFAȚA SPI

Interfața de comunicație SPI (Serial Peripheral Interface) asigură transferul rapid sincron de date între microcontroler și dispozitivele periferice sau cu alte microcontrolere AVR.

4.7.3 INTERFAȚA TWI

Interfața TWI (Two Wire Interface) implementează un protocol de comunicație pe două fire permitând interconectarea a până la 128 de dispozitive diferite. Interfața folosește două linii bidirectionale, una pentru tact (SCL) și una pentru date (SDA). Toate dispozitivele conectate la acest bus au propria adresă.

4.8 REGIȘTRII USART

Pentru a comunica via USART trebuie să fie setate anumite valori pentru regiștrii USART, informații care se pot obține din datasheet-ul microcontroler-ului.

UDR : USART Data Register : conține date receptată sau transmisă;

UCSRA / USCRB / UCSRC : USART Control and Status Registers : folosit pentru a configura USART-ul și a stoca statusul acestuia;

UBRRH / UBRL : USART Baud Rate Register : stochează valoarea corespunzătoare baud rate-ului folosit.

4.8.1 USCRA: USART CONTROL AND STATUS REGISTER A:

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

RXC : Este setat când USART a terminat de primit date.

TXC : Este setat când USART a terminat de transmis un byte către celălalt dispozitiv.

4.8.2 USCRB: USART CONTROL AND STATUS REGISTER B:

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

RXCIE: Receive Complete Interrupt Enable – dacă este setat pe 1, întreruperea pe flagul RXC este activată.

TXCIE: Transmit Complete Interrupt Enable – dacă este setat pe 1, întreruperea pe flagul TXC este activată.

RXEN: Receiver Enable – pentru a activa receptia trebuie setat pe 1.

TXEN: Transmitter Enable – pentru a activa transmisia trebuie setat pe 1.

UCSZ2: USART Character Size – este utilizat pentru a stabili numărul de biți per pachet.

4.8.3 USCRC: USART CONTROL AND STATUS REGISTER C:

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	UCSRC							
Valoare inițială	1	0	0	0	0	1	1	0	

UMSEL: USART Mode Select – acest bit selectează între modurile sincron și asincron.

UMSEL	Mod
0	Asincron
1	Sincron

UPM1:0: Parity Mode Acești biți activează și setează tipul generării și verificării parității. Dacă este activată, transmițătorul va genera automat și va trimite paritatea bițiilor din dată. Receptorul va genera o valoare de paritate pentru datele primite și o va compara cu setarea UPM0. Dacă se detectează o nepotrivire, va fi setat flagul PE din UCSRA.

UPM1	UPM10	Modul parității
0	0	Dezactivată
0	1	Rezervată
1	0	Activată, paritate pară
1	1	Activată, paritate impară

USBS: USART Stop Bit Select – Acest bit selectează numărul de biți de stop din transferul de date.

USBS	Biți de stop
0	1 BIT
1	2 BIT

UCSZ: USART Character size – Acești trei biți (unul în UCSRB) selectează numărul de biți din dată. Întotdeauna vom opta pentru 8 biți, însătrucăt acesta este standardul.

UCSZ2	UCSZ1	UCSZ0	Dimensiune
0	0	0	5Bit
0	0	1	6Bit
0	1	0	7Bit
0	1	1	8Bit
1	0	0	Rezervat
1	0	1	Rezervat
1	1	0	Rezervat
1	1	1	9Bit

4.8.4 UBRR: USART BAUD RATE REGISTER

Acest regisztr are o dimensiune de 16 biți, aşadar **UBRRH** este High Byte și **UBRRL** Low byte.

Acest regisztr este folosit de către USART pentru a genera baud rate (de exemplu 9600Bps).

Valoarea UBRR este calculată în funcție de următoarea formulă:

Operating Mode	Equation for calculating Baud Rate	Equation for calculating UBRR value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{16(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{8(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{2(UBRR + 1)}$	$UBRR = \frac{f_{osc}}{2BAUD} - 1$

4.9 APLICAȚII

Enunț

Să se transmită prin comunicație serială un sir dacă se primește litera s sau S.

Codul sursă

uart.h

```
#ifndef __USART__
#define __USART__

#include <avr.h>
#include <iom16.h>

#define F_OSC 4000000
#define BAUD 19200
#define BAUD_RATE (F_OSC/16/BAUD - 1)

void USART_initialize(unsigned short int baud_rate);
void USART_transmit(unsigned char data);
unsigned char USART_Receive( void );
//#pragma vector = USART_RXC_vect
//__interrupt void interrupt_routine_USART_RXC(void);

#endif
```

uart.c

```
#include "uart.h"

void USART_initialize(unsigned short int baud_rate)
```

```

{
    /* setează baud rate */
    UBRRH = (unsigned char)(baud_rate >> 8);
    UBRL = (unsigned char)(baud_rate & 0xFF);

    UCSRB = (1 << RXEN) | (1 << TXEN); /* activează transmisia și
    receptia la ieșire */

    /* setează pinul TXD: ieșire */
    DDRD |= (1 << PD1);

    /* setează pinul RXD: intrare */
    DDRD &= ~(1 << PD0);

    /* activează întreruperea */
    //UCSRB |= (1 << RXCIE);
}

void USART_transmit(unsigned char data)
{
    /* așteaptă până ce se termină de transmis toate datele și după trece
    la următoarele informații */
    while (!(UCSRA & (1 << UDRE)))
    {
        ;
    }
    UDR = data;
}

unsigned char USART_Receive( void )
{
    /* Așteaptă recepționarea datelor */
    while ( !(UCSRA & (1<<RXC)) )
    {
        ;
    }
    /* Preia și returnează datele recepționate din buffer */
    return UDR;
}

```

main.c

```

#include "uart.h"

void main( void )
{
    unsigned char string[]="Hello world!", aux;
    unsigned int i=0;
    USART_initialize(BAUD_RATE);
    while(1)
    {

```

```
aux=USART_Receive();
i=0;
if(aux=='s'||aux=='S')
{
    while(string[i]!='\0')
    {
        USART_transmit(string[i]);
        i++;
    }
}
}
```

5 Funcții IAR. Funcția Printf.

5.1 INTRODUCERE

5.1.1 PERSPECTIVĂ ASUPRA MEDIULUI DE DEZVOLTARE IAR

Există două limbaje de programare de nivel înalt disponibile cu compilatorul AVR®IAR C/C++ :

- **C**, cel mai răspândit limbaj de nivel înalt de programare folosit în industria de sisteme embedded. Folosind compilatorul AVR®IAR puteți construi aplicații de sine stătătoare, ce urmează standardul ISO 9899:1990. Acest standard este cunoscut ca ANSI C;
- **C++**, un limbaj modern orientat obiect, cu o bibliotecă ce dispune de toate caracteristicile necesare pentru o programare modulară.

Sistemele IAR suportă două nivele ale limbajului C++:

1. **Embedded C++ (EC++)**, un subset al standardului de programare C++, care este destinat programării sistemelor embedded. Este definit de un consorțiu industrial, Embedded C++ Technical Comitee;
2. **IAR Extended EC++**, cu caracteristici suplimentare cum ar fi suportul total pentru template-uri, suportul pentru namespace-uri, operatorii de cast, precum și Standard Template Library (STL).

Fiecare din cele două limbaje de programare suportate pot fi folosite fie într-un mod strict, fie în unul mai puțin strict, fie în unul mai puțin strict cu extensiile IAR activate. Modul strict aderă la standard, pe când celălalt mod permite anumite deviații de la acest standard.

Este de asemenea posibil ca anumite părți ale aplicației să fie implementate în limbaj de asamblare.

5.1.2 MEDIUL DE RULARE

Pentru crearea mediului de rulare este necesară alegerea unei biblioteci de rulare și setarea opțiunilor de bibliotecă.

Există două seturi de biblioteci de rulare puse la dispoziție:

- **IAR DLIB Library**, care suportă ISO/ANSI C și C++. Această bibliotecă suportă de asemenea numere în virgulă mobilă în format IEEE 754 și poate fi configurată pentru a include diferite nivele de suport pentru locale, descriptori de fișier, caractere multibyte, etc.
- **IAR CLIB Library**, este o bibliotecă din categoria ușoară, care nu este compilată în totalitate cu ISO/ANSI C. De asemenea nu oferă suport deplin pentru numere în virgulă mobilă în format IEEE 754 sau suport pentru Embedded C++ (această librărie este folosită implicit).

Librăria de rulare aleasă poate fi una dintre librăriile prebuilt, sau o librărie pe care ati customizat-o sau ati construit-o chiar dumneavoastră. IAR Embedded Workbench IDE oferă template-uri pentru librăriile de proiect pentru ambele librării, care se pot folosi pentru construirea

propriilor tipuri de librării. Acest lucru oferă control deplin asupra mediului de rulare. Dacă proiectul conține doar cod sursă în assembler nu este necesară alegerea unei librării de rulare.

5.1.2.1 Alegerea unei biblioteci de rulare în IAR Embedded Workbench

Pentru alegerea unei librării, se alege **Project → Options**, click pe tab-ul **Library Configuration** din categoria **General Options**. Se alege tipul de librărie adecvat din meniul drop-down.

Pentru DLIB library există două configurații diferite – Normal și Full, ce includ diferite nivele de suport pentru locale, descriptori de fișier, caractere multibyte, etc.

5.1.2.2 Alegerea unei biblioteci de rulare de la linia de comandă

Se folosesc următoarele opțiuni ale liniei de comandă pentru specificarea bibliotecii și fișierele de dependență:

Linia de comandă	Descriere
<code>-I\avr\inc</code>	Specifică calea pentru includere
<code>-I\avr\inc\{clib dlib}</code>	Specifică calea pentru fișiere specifice bibliotecii. Utilizați clib/dlib depinzând de ce biblioteca folosiți.
<code>libraryfile.r90</code>	Specifică fișierul obiect.
<code>--dlib_config C:\...\configfile.h</code>	Specifică fișierul de configurare pentru biblioteca(doar pentru biblioteca DLIB).

5.1.2.3 Setarea opțiunilor bibliotecii și a mediului de rulare

Se pot seta anumite opțiuni pentru a reduce dimensiunile bibliotecii și a mediului de rulare:

- Funcțiile de intrare/ieșire (cel mai des utilizate sunt scanf și printf);
- Dimensiunea stivei și heap-ului.

5.1.3 MEDIUL RUNTIME DLIB

Descrie mediul de execuție în care o cerere se execută. În special, se referă la biblioteca runtime DLIB și modul în care se poate modifica, la opțiunile de setare a modulelor implicate ale bibliotecii, sau construirea bibliotecii proprii pentru a putea fi optimizată aplicația.

Acesta se referă atât la inițializarea sistemului, cât și la funcționarea sistemului; modul în care o aplicație poate controla ceea ce se întâmplă înainte de funcția principală.

5.1.3.1 Introducere în mediul Runtime

Mediul de rulare (IAR DLIB) este mediul în care cererile se execută. Acesta depinde de destinația hardware și software. IAR DLIB poate fi folosit împreună cu IAR C-SPY Debugger.

Această secțiune oferă o privire de ansamblu asupra:

- Mediului de execuție și a componentelor sale;
- Biblioteca de selecție.

5.1.3.2 Funcționalitatea mediului Runtime

Mediul de rulare (RTE) sprijină ISO/ANSI C și C++, inclusiv biblioteca standard de şablonane. Mediul de rulare este format din *runtime library*, care conține funcțiile definite de aceste standarde, și includ fișiere care definesc biblioteca interfață.

Biblioteca Runtime este disponibilă atât în biblioteci precompilate, cât și ca fișiere sursă.

Mediul de rulare amintit mai sus, cuprinde:

- Suport pentru caracteristici hardware;
- Acces direct la operațiunile de procesor low-level prin intermediul funcțiilor *intrinseci*;
- Registrele de unitate și definițiile incluse în fișiere;
- Suportul compilatorului special pentru accesarea șirurilor de caractere în memoria flash ;
- Suportul mediului de runtime, care este, cod de intrare, cod de ieșire și interfață low-level la unele funcții de bibliotecă.

Unele părți, cum ar fi codul de intrare și de ieșire și mărimea heap-urilor trebuie să fie adaptate atât pentru hardware-ul specificat, cât și pentru cerințele aplicației.

5.1.3.3 Selectarea bibliotecii

Pentru a configura cât mai eficient mediul de rulare al codului, trebuie să fie cunoscute cerințele hardware ale aplicației.

IAR Embedded Workbench vine cu un set de biblioteci runtime precompilate. Pentru a se obține mediul de execuție necesar, sunt necesare următoarele:

- Setarea opțiunilor de bibliotecă, de exemplu, pentru alegerea lui scanf și printf, și pentru a preciza dimensiunea stivei și heap-ului;
- Setarea funcțiilor bibliotecă, de exemplu cstartup.s90, cu propria versiune;
- Alegerea nivelului de sprijin pentru anumite funcționalități a bibliotecii standard, de exemplu: locale, descriptori de fișiere și multibytes, prin alegerea unui *library configuration*: normal sau complet.

În plus, se pot face, de asemenea modificări proprii bibliotecii, fapt care necesită reconstruirea bibliotecii. Acest lucru permite obținerea controlului deplin a mediului runtime.

Notă: Aplicația proiectului trebuie să poată localiza biblioteca, inclusiv fișierele de configurare ale bibliotecii.

5.1.3.4 Situații care necesită construirea bibliotecii

Construirea unei biblioteci proprii este un proces complex. Prin urmare, ar trebui să se ia în considerare doar ceea ce este cu adevărat necesar.

Propria bibliotecă se construiește:

- Atunci când nu există nici o bibliotecă precompilată pentru combinația necesară de opțiuni a compilatorului sau a suportului hardware;
- Pentru a defini configurația proprie a bibliotecii, cu suport pentru locale, descriptori de fișier, caractere multiocet, etc.

5.1.3.5 Configurarea bibliotecii

Este posibilă configurarea nivelului de sprijin pentru: locale, descriptori de fișiere,multibytes. Biblioteca de configurație runtime este definită *library configuration file*. Aceasta conține informații despre ceea ce înseamnă funcționalitatea unei părți a mediului de rulare. Fișierul de configurație este utilizat pentru realizarea unei biblioteci runtime, precum și realizarea antetului, folosit atunci când se face compilarea aplicației.

Sunt disponibile următoarele configurații bibliotecă DLIB:

Biblioteca de configurare	Descriere
Normal DLIB	Nu conține: interfață locală, C locale, suport de descriptor de fișier, caractere multibyte în printf și scanf, hex floats în strtod.
Full DLIB	Conține interfață locală completă, C locale, suport descriptor de fișier, caractere multibyte în printf și scanf și hex floats în strtod.

Tabel 5.1 Configurații bibliotecă

În plus, față de aceste configurații, se pot defini configurații proprii, ceea ce înseamnă că trebuie modificat fișierul de configurație existent. Deci, fișierul de configurație a bibliotecii descrie modul în care o bibliotecă a fost construită, și, prin urmare nu poate fi schimbată decât dacă se reconstruiește biblioteca.

Bibliotecile precompilate sunt bazate pe configurații implice. Există, de asemenea, template-uri ale proiectului bibliotecă gata făcute care se pot utiliza, pentru reconstruirea bibliotecii runtime.

5.1.3.6 Suportul debug în mediul bibliotecii

Puteți oferi bibliotecii diferite nivele de depanare-suport de bază, runtime și depanare I/O.

Următorul tabel descrie diferite niveluri a suportului de depanare:

Suport de depanare	Opțiune linker în IAR Embedded Workbench	Comanda linker opțiune de linie	Descriere
Depanare de bază	Informații C-SPY	-Fubrof	Asistență pentru depanare C-SPION fără nici un sprijin de funcționare
Depanarea funcționării	Cu module de control a funcționării	-r	La fel ca -Fubrof, dar, de asemenea, include suport debugger pentru manipularea programului
Depanare I/O	Cu module de emulare I/O	-rt	La fel ca -r, doar că include suport pentru manipulare I/O (stdin și stdout sunt redirecționate la Terminalul C-SPY)

Tabel 5.2 Niveluri de depanare suport în bibliotecile runtime

Dacă se construiește aplicația proiectului cu opțiunile **XLINK With runtime control modules** sau **With I/O emulation modules**, anumite funcții din bibliotecă vor fi înlocuite cu funcții care comunică cu C-SPY IAR Debugger. Pentru a seta opțiunile linker pentru depanarea suportă în IAR Embedded Workbench, se alege **Project>Options** și categoria **Linker**. Pe pagina **Output**, se selectază opțiunea corespunzătoare **Format**.

5.1.3.7 Utilizarea bibliotecii precompilate

Bibliotecile precompilate runtime sunt configurate pentru diferite combinații de caracteristici cum ar fi:

- Tipul de bibliotecă;
- Opțiunea procesor (-v);
- Opțiunea modelului de memorie (--memory_model);
- Opțiune de bază AVR (--enhanced_core);
- Mici opțiuni ale memoriei flash (--64k_flash);
- Opțiune 64-bit doubles (--64bit_doubles);
- Biblioteca de configurație-normal sau completă.

Pentru AVR IAR C/C++ Compiler și configurația bibliotecă Normal, există precompilate biblioteci runtime pentru toate combinațiile acestor opțiuni.

Următorul tabel prezintă numele bibliotecilor și modul în care sunt reflectate setările utilizate:

Fișierul de Biblioteca	Opțiune Generic Procesor	Opțiune Generic Procesor	Modelul memoriei	Enhanced core	Small flash	64-bit doubles	Configurarea bibliotecii
dlavr-3s-ecsf-n.r90	-v3	-v3	Small	X	X	--	Normal
dlavr-3s-ec-64-f.r90	-v3	-v3	Small	X	--	X	Full

Tabel 5.3 Biblioteci precompilate

Numele bibliotecilor sunt construite în felul următor:

<*library*><*target*>-<*cpu*><*memory_model*>-<*enhanced_core*>-<*small_flash*>-<*64-bit_doubles*>-<*library_configuration*>.r90

unde:

- <*library*> este dl pentru biblioteca DLIB IAR sau cl pentru biblioteca CLIB IAR (pentru o listă de fișiere bibliotecă CLIB, a se vedea *Runtime environment*)
- <*target*> este avr
- <*cpu*> este o valoare de 0-6, opțiunea-v
- <*memory_model*> este fie t, e, sau l pentru modelul de memorie respectiv(Tiny, Small, Large)

- *<enhanced_core>* este CE atunci când baza consolidată este folosită. În cazul în care baza consolidată nu este folosită, această valoare nu este specificată
- *<small_flash>* este SF atunci când memoria flash *small* este disponibilă. Când memoria flash nu este disponibilă, această valoare nu este specificată
- *<64-bit_doubles>* este de 64 biți când se utilizează 64 biți doubles. Când sunt utilizati 32 de biți doubles, această valoare nu este specificată
- *<library_configuration>* N=normal sau F=full

Notă: Fișierul de configurare al bibliotecii are același nume de bază ca biblioteca.

5.1.4 MEDIUL RUNTIME CLIB

Se descrie mediul runtime în care o aplicație este executată. În particular aceasta acoperă biblioteca mediului CLIB runtime și cum se poate optimiza aplicația.

Mediul CLIB descrie sistemul de inițializare și oprire. Acesta prezintă cum o aplicație poate fi controlată și ceea ce se întâmplă înainte de pornirea funcției main, precum și metoda de personalizare a interfeței. În cele din urmă, interfața de runtime C-SPY este inclusă.

5.1.4.1 Mediul Runtime

Mediul de rulare CLIB include biblioteci standard C. Link-editarea va include numai acele rulări care sunt necesare direct sau indirect de către aplicație.

IAR Embedded Workbench vine cu un set de biblioteci de rulare precompilate, care sunt configurate pentru diferite combinații:

- Tipul bibliotecii;
- Opțiunile procesorului (*-v*) ;
- Opțiunile modelului de memorie (*--memory_model*) ;
- Opțiunile consolidate ale nucleului AVR-ului (*--enhanced_core*) ;
- Opțiunile memoriei flash mai mici (*--64k_flash*) ;
- Opțiunile în dublă precizie pe 64-biți (*--64bit_doubles*) .

Pentru compilatorul AVR IAR C/C++, înseamnă că este precompilată biblioteca de rulare pentru diferite combinații ale acestor opțiuni. În tabelul 5.5, este arătat numele bibliotecii și cum se reflectă asupra setărilor utilizate.

Fișierul librărie	Opțiunea procesorului	Modelul memoriei	Bază consolidată	Small flash	64-bit doubles
c10t.r90	-v0	Tiny	--	--	--
c11s-64.r90	-v1	Small	--	--	X
c161-ec-64.r90	-v6	Large	X	--	X

Tabel 5.4 Precompilarea bibliotecilor

Numele bibliotecii este construit în următorul mod:

```
<library><cpu><memory_model>-<enhanced_core>-<small_flash>-<64-bit_doubles>.r90
```

Unde:

- <library> este cl pentru biblioteca IAR CLIB, sau dl pentru biblioteca IAR DLIB;
- <cpu> este o valoare de la 0 la 6;
- <memory_model> este t, s, sau l pentru modelele memoriei (Tiny,Small, Large);
- <enhanced_core> este ec și este folosit atunci când se consolidează nucleele. Când consolidarea nucleelor nu este folosită , această valoare nu este specificată;
- <small_flash> este sf când memoria flash este disponibilă. Când memoria flash nu este disponibilă, această valoare nu este specificată;
- <64-bit_doubles> este 64 când este utilizată dubla precizie pe 64 biți. Când este utilizată dubla precizie pe 32 biți, această valoare nu este specificată.

5.2 FUNCȚII

Se găsește o privire de ansamblu la mecanismele de control asupra funcțiilor.

5.2.1 CUVINTE EXTINSE PENTRU FUNCȚII

Cuvintele cheie care pot fi folosite pentru funcții pot fi împărțite în trei categorii:

- Cuvinte cheie care pot controla locația și tipul funcțiilor. Cuvintele cheie din acest grup trebuie specificate și când funcțiile sunt definite, și când funcțiile sunt declarate: __nearfunc și __farfunc
- Cuvinte cheie care controlează tipul funcțiilor. Cuvintele cheie din acest grup trebuie folosite doar atunci când funcția este definită: __interrupt, __task, și __version_1
- Cuvinte cheie care controlează doar funcțiile definite : __root, __monitor, și __noreturn

Cuvintele cheie care controlează locația și tipul funcțiilor sunt de asemenea denumite type attributes (attribute de tip). De obicei, aceste funcții controlează aspecte ale funcției vizibile din contextul exterior. Cuvintele cheie care controlează doar comportamentul funcției și nu afectează interfața funcției, sunt numite object attributes (attribute de obiect).

5.2.2 SINTAXA

Cuvintele cheie extinse sunt menționate înainte de tipul returnat, de exemplu:

```
__interrupt void alpha(void);
```

Cuvintele cheie care sunt attribute de tip trebuie specificate atât atunci când sunt definite, cât și în declarație. Atributele de Obiect trebuie să fie specificate doar când sunt definite, atât timp cât nu afectează modul în care funcția este folosită.

În plus, față de regulile prezentate aici – pentru a plasa cuvântul cheie direct în cod – pot fi folosite directivele #pragma type_attribute și #pragma object_attribute

pentru specificarea cuvintelor cheie.

5.2.3 STOCAREA FUNCȚIILOR

Există două atrbute de memorie pentru controlarea stocării funcțiilor : `__nearfunc` și `__farfunc`.

Următorul tabel rezumă caracteristicile fiecărui atribut de memorie:

Atribut de memorie	Interval de adresă	Dimensiune pointer	Folosit în opțiunea procesor
<code>__nearfunc</code>	0–0x1FFFFE (128 Kbytes)	16 biți	<code>-v0, -v1, -v2, -v3, -v4</code>
<code>__farfunc</code>	0–0x7FFFFFFE (8 Mbytes)	24 biți	<code>-v5, -v6</code>

Tabel 5.5 Atribute de memorie pentru funcții

Când este folosită opțiunea `-v5` sau `-v6` este posibil ca, pentru anumite funcții, să treacă peste atributul `__farfunc` și în locul său să folosească atributul `__nearfunc`. Atributul de memorie implicit poate fi ignorat prin specificarea explicită a atributului de memorie în declarația funcției folosind directiva `#pragma type_attribute`:

```
#pragma type_attribute=__nearfunc void MyFunc(int i)
{
    ...
}
```

Este posibil să fie apelată o funcție cu atributul `__nearfunc` dintr-o funcție `__farfunc` și viceversa. Doar dimensiunea pointerului la funcție este afectată. Pointerii cu atrbute ale funcțiilor de memorie au restricții în cast-urile implicate și explice la cast-urile dintre pointeri, și de asemenea la cast-urile dintre pointeri și valori integer.

Este posibilă plasarea funcțiilor în segmente, folosind operatorul `@` sau directiva `#pragma location`.

5.2.4 FUNCȚIA MYPRINT

Se va crea o funcție proprie pentru comunicația pe serială. Prin intermediul acestei funcții, se vor putea trimite pe serială caractere sau numere. Funcția va primi trei parametri după cum urmează:

- „tip” – semnifică tipul argumentului „valoare”. Poate lua una din următoarele valori:
 - 0 - caz în care pe serială se va transmite un număr de tip integer;
 - 1 - caz în care pe serială se va transmite un număr de tip long long în formă hexazecimală;
 - 2 - caz în care pe serială se va transmite un număr de tip double;
 - 3 - caz în care pe serială se va transmite un sir de caractere.

2. „nr_car” – semnificață numărul efectiv de caractere de transmis pe serială, inclusiv semnul „-”, în cazul unui număr;
3. „val” - reprezintă pointer la valoarea efectivă de transmis pe serială. Tipul acestei variabile este „`void*`” .

Deci, **prototipul** funcției este următorul:

```
void myprint(unsigned int tip, unsigned int nr_car, void * val)
```

Implementarea acestei funcții:

```
void myprint(unsigned int tip, unsigned int nr_car, void * val)
{
    switch(tip)
    {
        case 0:
            integerTransmit(tip, nr_car, val);
            break;

        case 1:
            hexadecimalTransmit(tip, nr_car, val);
            break;

        case 2:
            doubleTransmit(tip, nr_car, val);
            break;

        case 3:
            characterTransmit(tip, nr_car, val);

            break;
    }
}

//transmiterea unui întreg pe serială
void integerTransmit (unsigned int p1, unsigned int p2, void * p3)
{
    int index=0;
    char aux[5];
    int x=*((int *) (p3));
    if(x<0)
    {
        USART_transmit(' - ');
        x*=(-1);
    }
    while(x!=0)
    {
        aux[index]=x%10+'0';
        x/=10;
        index++;
    }
}
```

```
    index++;
    x=x/10;
}
while(p2>0 )
{
    USART_transmit(aux[index-1]);
    index--;
    p2--;
}
}

//transmiterea unui număr hexazecimal pe serială
void hexadecimalTransmit (unsigned int p1, unsigned int p2, void * p3)
{
    long long x=*((long long *) (p3));
    int index=0;
    USART_transmit('0');
    USART_transmit('x');
    while(x!=0)
    {
        aux[index]=x&0x0F;
        if(aux[index]<=9)
        {
            aux[index]+='0';
        }
        else
        {
            aux[index]=aux[index]+'A'-10;
        }
        index++;
        x>>=4;
    }
    while(p2>0 && index>0)
    {
        USART_transmit(aux[index-1]);
        index--;
        p2--;
    }
}

//transmiterea unui număr de tip double pe serială
void doubleTransmit(unsigned int p1, unsigned int p2, void * p3)
{
    int index=0;
    int dataIntreg;
    double x=*((double *) (p3));
    if(x<0)
    {
```

```
    USART_transmit(' - ');
    x*=-1;
}
dataIntreg=(int)x;
while((int)dataIntreg!=0)
{
    aux[index]=(int)dataIntreg%10+'0';
    index++;
    dataIntreg=dataIntreg/10;
}
while(index>0 )
{
    USART_transmit(aux[index-1]);
    index--;
}

dataIntreg=(int)x;
x=x-dataIntreg;
if(x>0)
{
    USART_transmit('.');
}
while(x>0 && p2>0)
{
    x=x*10;
    dataIntreg=(int)x;
    USART_transmit(dataIntreg+'0');
    x=x-dataIntreg;
    p2--;
}
}

//transmiterea unui numar de tip float
void floatTransmit(unsigned int p1, unsigned int p2, void * p3)
{
    char buff[30]={0};
    int j, nrCaract;
    long long nr;
    char nrNeg;
    int i;
    float floatNr;
    floatNr=*((float *)p3);
    nrCaract=0;
    nr=(long long)floatNr;

    nrNeg=(nr<0);
    if (nrNeg)
        nr*=-1;

    while(nr!=0)
```

```
{  
    j=nr%10;  
    nr=nr/10;  
    buff[nrCaract]=j+'0';  
    nrCaract++;  
}  
if(nrCaract ==0)  
{  
    buff[0]='0';  
    nrCaract=1;  
}  
if(nrNeg)  
{  
    buff[nrCaract]='-';  
    nrCaract++;  
}  
for(i=nrCaract-1; i>=0; i--)  
{  
    USART_transmit(buff[i]);  
}  
USART_transmit('.');  
if (floatNr<0)  
    floatNr*=-1;  
for (i=0; i<p2; i++)  
{  
    floatNr=floatNr-(long long)floatNr;  
    floatNr*=10;  
    nr=(int)floatNr;  
    USART_transmit(nr+'0');  
}  
}  
  
//transmiterea unui sir de caractere pe serială  
void characterTransmit (unsigned int p1, unsigned int p2, void *  
p3)  
{  
    unsigned char *x=(unsigned char *) (p3);  
    int index=p2;  
    while(index!=0)  
    {  
        USART_transmit(x[index]);  
        index--;  
    }  
}
```

Exemplu de utilizare al acestei funcții:

```
#include"uart.h"  
#include<iom16.h>  
#include<inavr.h>
```

```

void main( void )
{
    USART_initialize(BAUD_RATE);
    unsigned int tip=2, nr_car=4;
    double a = -2.13;
    double *val = &a;
    USART_transmit(0xd0);
    USART_transmit(0xa0);
    myprint(tip, nr_car, val);
}

```

Notă: Pentru crearea și utilizarea acestei funcții este nevoie de fișierele: „usart.h” și „usart.c”.

5.3 CREAREA UNEI BIBLIOTECI ÎN IAR EMBEDDED WORKBENCH

După ce am creat proiectul și avem toate fișierele ce vrem să le includem în bibliotecă se va naviga către **Project>Options>General Options>Output**. Aici se va alege **Output file** de tipul **Library**.

În continuare se va rula proiectul: **Project>Make**, iar apoi se va merge în directorul proiectului, iar apoi se vor accesa următoarele foldere: **Debug>Exe**. În acesta din urmă se va găsi un fișier cu extensia .r90. Acest fișier reprezintă biblioteca creată de noi.

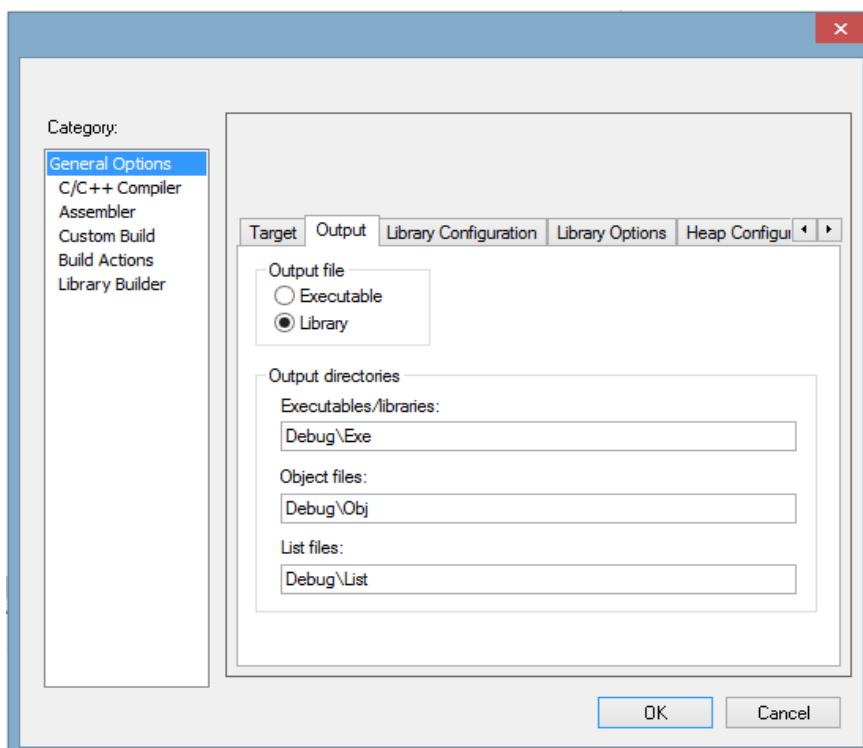


Figura 5.1 Setarea fișierelor de ieșire

5.4 IMPORTAREA UNEI BIBLIOTECI ÎNTR-UN NOU PROIECT

Pentru importarea unei biblioteci într-un nou proiect este nevoie de fișierul header ce conține toate prototipurile funcțiilor care pot fi apelate prin intermediul bibliotecii importate.

Primul pas pentru importarea unei biblioteci într-un nou proiect este de a include fișierul header corespunzător. Acest lucru se poate face fie prin intermediul directivei `#include`, fie prin navigarea către **Project>Options>C/C++ Compiler>Preprocessor>Preinclude File** unde se alege fișierul header corespunzător.

Următorul pas reprezintă navigarea către **Project>Options>Linker>Extra options**. Aici se va bifa opțiunea **Use command line options**. În zona text nou activată se va include biblioteca creată astfel:

-I\avr\inc\cib <calea completă către fișierul .r90 >

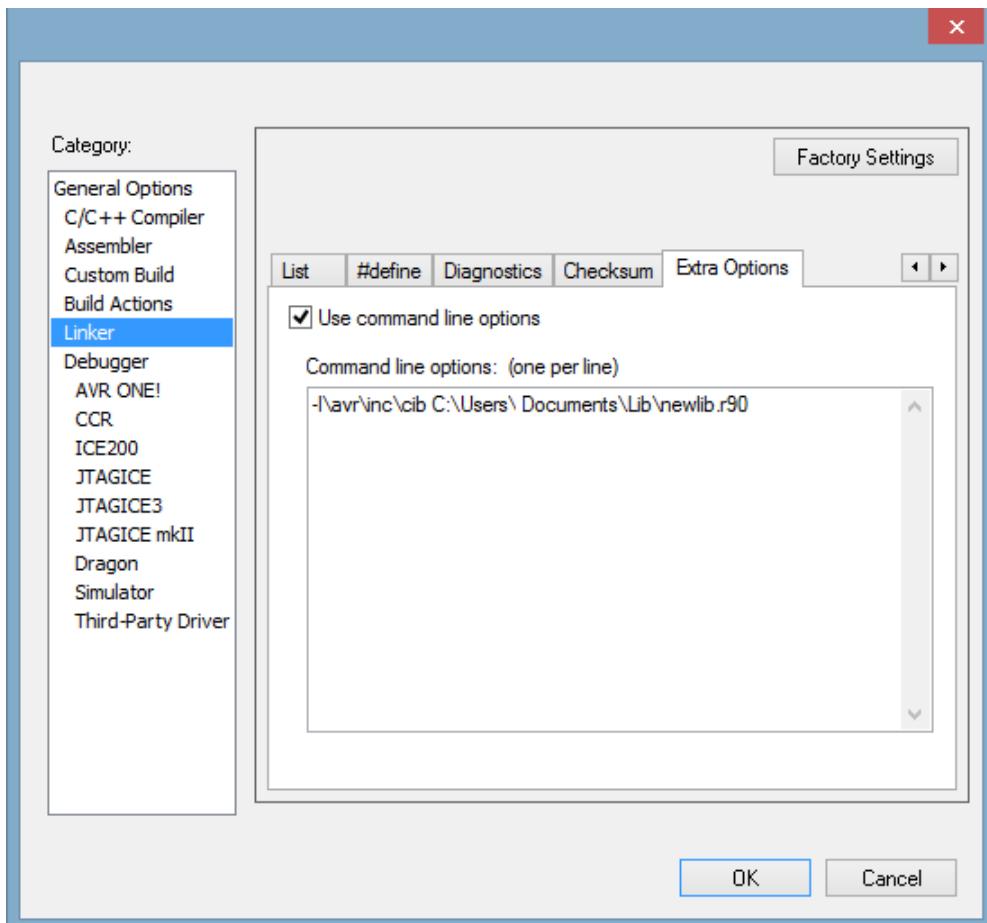


Figura 5.2 Adăugare cale bibliotecă

6 Întreruperi

6.1 ÎNTRERUPERI. TABELA VECTORILOR DE ÎNTRERUPERI.

6.1.1 INTRODUCERE

O întrerupere reprezintă un semnal sincron sau asincron de la un periferic ce semnalizează apariția unui eveniment care trebuie tratat de către procesor. Tratarea întreruperii are ca efect suspendarea firului normal de execuție al unui program și lansarea în execuție a unei rutine de tratare a întreruperii (RTI).

Întreruperile *hardware* au fost introduse pentru a se elimina buclele pe care un procesor ar trebui să le facă în aşteptarea unui eveniment de la un periferic. Folosind un sistem de întreruperi, perifericele pot atenționa procesorul în momentul producerii unei întreruperi (IRQ), acesta din urmă fiind liber să ruleze programul normal în restul timpului și să înceapă execuția doar atunci când este necesar.

Înainte de a lansa în execuție o RTI, procesorul trebuie să aibă la dispoziție un mecanism prin care să salveze starea în care se află în momentul apariției întreruperii. Aceasta se face prin salvarea într-o memorie, de cele mai multe ori organizată sub forma unei stive, a registrului contor de program (Program Counter), a registrelor de stare precum și a tuturor variabilelor din program care sunt afectate de execuția RTI. La sfârșitul execuției RTI starea anterioară a registrelor este refăcută și programul principal este reluat din punctul de unde a fost întrerupt.

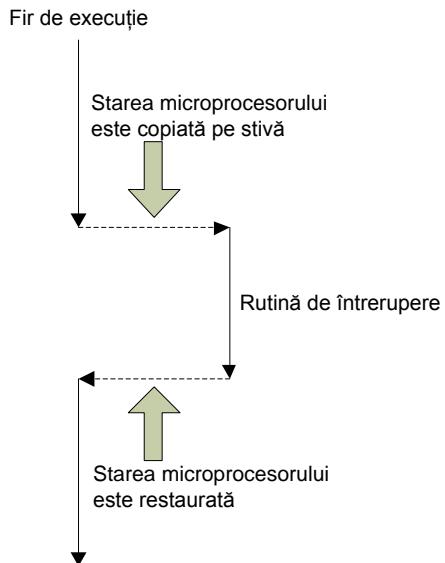
Întreruperile sunt indispensabile în proiectarea unui sistem care să reacționeze corect și eficient în raport cu lumea exterioară. Faptul că au suport hardware oferă un timp de răspuns și *overhead* minimal.

În același timp, din punct de vedere *software*, întreruperile au un neajuns intrinsec. În primul rând, ele nu sunt portabile pe diferite procesoare și chiar pe diferite compilatoare. În al doilea rând, întreruperile pot determina numeroase erori *software* greu de identificat.

6.2 NOTIUNI

O întrerupere are două înțelesuri apropiate:

- transferul *hardware* al controlului (saltul firului de execuție) către un vector de întrerupere pe baza înregistrării unui fenomen exterior procesorului;
- funcția de tratare a întreruperii – secvența de cod la care se ajunge plecând de la vectorul de întrerupere.

**Figura 6.1** Întrerupere

Pentru a asocia o întrerupere cu o anumită rutină din program, procesorul folosește tabela vectorilor de întrerupere (Tabel 6.1). Fiecărei întreruperi îi este asociată o adresă la care programul va face salt în cazul apariției acesteia. Aceste adrese sunt predefinite și sunt mapate în memoria de program într-un spațiu contigu care alcătuiește TVI. Adresele întreruperilor în TVI sunt setate în funcție de prioritatea lor, cu cât adresa este mai mică cu atât prioritatea este mai mare.

Pentru ATMega16, TVI este dată în tabelul de mai jos:

Număr vector	Adresa programului	Sursa	Definiția întreruperii
1	\$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI,STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

Tabel 6.1 Tabela vectorilor de întrerupere (TVI)

Se observă că TVI este plasată de la prima adresă a memoriei de program și că întreruperile sunt puse din două în două adrese consecutive. Prioritatea cea mai mare o are întreruperea de RESET, de la adresa 0, apoi întreruperea externă 0 (INT0).

Perifericele care pot genera întreruperi la ATMega16 sunt timer-ele, interfața serială (USART), convertorul analog-digital (ADC), controlerul de memorie EEPROM, comparitorul analog și interfața serială I2C. De asemenea, procesorul poate să primească cereri de întreruperi externe din trei surse (INT0, 1 și 2), ce corespund unui număr egal de pini exteriori.

6.3 ACTIVAREA/DEZACTIVAREA ÎNTRERUPERILOR

Întreruperile pot fi activate sau dezactivate de utilizator în program prin setarea individuală a biților de interrupt enable pentru fiecare periferic folosit și prin setarea flagului de “Global Interrupt Enable” (I) din Status Register (*Figura 6.2*).

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Valoare inițială	0	0	0	0	0	0	0	0	

Figura 6.2 Registrul de stare (SREG)

O întrerupere este dezactivată dacă s-a utilizat un suport *hardware* pentru a preveni declanșarea întreruperii. Suportul *hardware* este dat de 2 biți: unul specific fiecărui tip de întrerupere și unul ce se referă la toate întreruperile. Întreruperea de Reset face abatere de la această regulă prin faptul că nu poate fi prevenită. Saltul către un vector oarecare de întrerupere poate avea loc doar dacă cei doi biți au valoarea 1.

Pentru ATmega16, bitul ce se referă la toate întreruperile se numește I și se află în registrul de stare al microprocesorului (bitul 7 din Figura 6.2). Pentru a-l modifica se pot utiliza următoarele instrucțiuni:

Mnemonică	Descriere	Operatie	Număr de cicli	Funcții IAR
SEI	<i>Global Interrupt Enable</i>	$I \leftarrow 1$	1	<code>_enable_interrupt</code>
CLI	<i>Global Interrupt Disable</i>	$I \leftarrow 0$	1	<code>_disable_interrupt</code>

Pentru a modifica bitul individual de validare a întreruperii, acesta trebuie căutat pentru fiecare tip de întrerupere. De exemplu, pentru întreruperea externă INT0, acesta se regăsește în registrul Figura , pe poziția 6. Astfel, pentru a admite întreruperi de tip INT0, pe frontul crescător al semnalului extern, putem utiliza următorul cod:

Cod sursă 1:

```
#include <inavr.h> //include biblioteca inavr.h
#include <iom16.h> //include biblioteca iom16.h

int main(void)
{
```

```

    // începutul inițializărilor
    MCUCR |= ((1 << ISC01) | (1 << ISC00)); // setează ca front pozitiv
    GICR |= (1 << INT0); // activează INT0 (External Interrupt Request 0
Enable)
    __enable_interrupt(); /* activează întreruperea globală */
    // sfârșitul inițializărilor
}

```

Se remarcă faptul că ultimul lucru din procedura de inițializare a întreruperilor este validarea întreruperilor la nivel global (ordine *bottom-up*).

Pentru fiecare tip de întrerupere există un bit ce este setat ori de către platforma *hardware* înregistrează producerea fenomenului cauză specific și, de regulă, este pus pe 0 de către microprocesor la intrarea în rutina de întrerupere. De exemplu, în cazul întreruperii INT0 acest bit se numește INTFO și se află în registrul GIFR pe poziția 6.

Observație: Pentru a reseta bitul ce semnifică producerea fenomenului de întrerupere i se va asigna acestui bit valoarea 1.

Prioritatea întreruperilor are justificare în cazul în care mai multe întreruperi se găsesc în așteptare. Întreruperea cu numărul de ordine cel mai mic va fi următoarea tratată (a se vedea *Tabel 6.1*).

O funcție de întrerupere în curs nu poate fi întreruptă la rândul ei de către o întrerupere cu prioritate mai înaltă.

Latența întreruperii reprezintă practic timpul de răspuns al microprocesorului. Se definește ca fiind intervalul de timp dintre momentul de timp în care condiția de întrerupere a avut loc și momentul de timp în care s-a intrat în rutina de întrerupere.

De regulă, latența întreruperii nu este o mărime constantă și se poate vorbi despre „cea mai defavorabilă latență”.

O funcție de întrerupere se termină cu instrucțiunea RETI (*return from interrupt*). Această instrucțiune realizează de fapt o revenire în program în punctul în care acesta a fost întrerupt. Adresa de revenire (2 octetii) este stocată (pe stivă, de regulă) înainte de a se efectua saltul către vectorul de întrerupere (deci în momentul în care întreruperea se află în stare de așteptare și s-a decis tratarea ei).

În general, întreruperile pot fi sau nu reentrantă. Se spune despre o întrerupere că este reentrantă dacă execuția ei poate fi întreruptă la apariția unei întreruperi cu o prioritate mai mare. Întreruperile de pe microprocesorul ATmega16 nu sunt reentrantă.

6.4 REGISTRE PENTRU TRATAREA ÎNTRERUPERILOR EXTERNE

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valoare inițială	0	0	0	0	0	0	0	0	

Figura 6.3 Registrul de control a întreruperilor (GICR = General Interrupt Control Register)

Acest regisztr este răspunzător pentru plasarea tabelului vectorului de întreruperi.

Bitul 0 din acest regisztr se numește **IVCE** (Interrupt Vector Change Enable). Acest bit trebuie inițializat cu valoarea logică “1” pentru a permite schimbarea următorului bit din regiszru și anume **IVSEL**. Setând acest bit, nu se vor mai putea genera întreruperi.

Bitul 1 din acest regisztr se numește **IVSEL** (Interrupt Vector Select). Când acest bit are valoarea logică “0” vectorii de întreruperi sunt plasați la începutul memoriei de program (Flash). Când acest bit este setat cu valoarea logică “1”, vectorii de întreruperi sunt mutați la începutul zonei de boot .

Bitul 7 –INT1:External Interrupt Request 1 Enable:

Când INT1 e setat cu valoarea logică “1” și bitul “I” din Figura 6.2 este setat cu aceeași valoare, pinul destinat întreruperii externe INT1 este activat. Biții ISC11 și ISC10 definesc logica de generare a întreruperii, respectiv dacă aceasta este generată pe front crescător sau descrescător. Întreruperea va fi setată conform rutinei asociate vectorului de întrerupere.

Bitul 6 –INT0:External Interrupt Request 0 Enable:

Când INT0 e setat cu valoarea logică “1” și bitul “I” din Figura 6.2 este setat cu aceeași valoare pinul destinat întreruperii externe 0 este activat. Biții ISC01 și ISC00 definesc logica de generare a întreruperii, respectiv dacă aceasta este generată pe tranziția de creștere sau pe cea de scădere.

Bitul 5 –INT2:External Interrupt Request 2 Enable:

Când INT0 e setat cu valoarea logică “1” și bitul “I” din Figura 6.2 este setat cu aceeași valoare, pinul destinat întreruperii externe 2 este activat. Bitul ISC2 definește logica de generare a întreruperii, respectiv dacă aceasta este generată pe tranziția de creștere sau pe cea de scădere.

6.4.1 ÎNTRERUPERILE EXTERNE

Aceste întreruperi sunt generate prin intermediul pinilor INT0, INT1 și INT2. Ele sunt activate chiar dacă acești pini sunt setați ca fiind de output (ieșire). Modul în care se pot genera întreruperile externe poate fi setat prin configurarea regiszrelor Figura 6.4 și Figura 6.5 .

Bit	7	6	5	4	3	2	1	0	MCUCR
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00		
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Figura 6.4 MCU Control Register

ISC11	ISC10	Descriere
0	0	Nivelul scăzut al lui INT1 generează o cerere de întrerupere.
0	1	Orice schimbare logică pe INT1 generează o cerere de întrerupere.
1	0	Pe frontul negativ al lui INT1 se generează o cerere de întrerupere.
1	1	Pe frontul pozitiv al lui INT1 se generează o cerere de întrerupere.

Tabel 6.3

Bit	7	6	5	4	3	2	1	0	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Valoare inițială	0	0	0	0	0	0	0	0	

Figura 6.5 MCU Control and Status Register

De exemplu, pentru a folosi întreruperea externă INT2 sunt necesare următoarele configurații:

1. Bitul I din Figura 6.2 trebuie să fie setat (global interrupt enable);
 2. Bitul INT2 din Figura trebuie setat (INT2 enable) ;
 3. Dacă ISC2 este inițializat cu valoarea zero, INT2 va fi activată pe front descrescător (tranzitie din 1 în 0 a pinului INT2), în caz contrar ea va fi activată pe front crescător.
- Perifericele care pot genera întreruperi sunt: timerele, convertorul analog-digital, interfețele seriale (RS232, I2C, SPI) etc.

6.5 PROBLEME

6.5.1 CORECTITUDINEA CONCURENTIALĂ

Întreruperile pot provoca un comportament neașteptat al microprocesorului dacă nu s-au prevăzut toate aspectele legate de concurență.

Se presupune că o întrerupere poate avea loc oricând. Deci o funcție de întrerupere va avea structura următoare:

1	Fă o copie a tuturor regiștrilor ce vor fi eventual modificați în corpul funcției de întrerupere
2	Realizează toate operațiile proprii rutinei de întrerupere
3	„Refă” toti regiștrii referiți la pasul 1

Tabel 6.4 Structura unei întreruperi

Într-o funcție de întrerupere scrisă în C nu se specifică pașii 1 și 3, deoarece compilatorul are grija să studieze ce registri sunt modificați în corpul funcției de întrerupere și să-i salveze. Oricum, estimările compilatorului pot fi altele decât cele făcute de programator (compilatorul va lăua cel mai probabil niște precauții mai mari decât cele necesare), astfel încât, la un moment dat, să apară dorința de a scrie funcția de întrerupere în limbaj de asamblare.

6.5.1.1 Depășirea stivei

Se presupune că o rutină de întrerupere are nevoie de un spațiu propriu peste stiva programului. Dimensiunea acestui spațiu reflectă un consum de memorie și, poate cel mai important, existența unor instrucțiuni de scriere-citire ce se execută în întrerupere. Din acest motiv, în rutinile de întrerupere se evită apelurile de funcție.

Ceea ce rămâne de făcut din partea proiectantului este să scrie rutine de întrerupere care să utilizeze cât mai puțină stivă și să dimensioneze stiva astfel încât să nu se ajungă la coruperea ei.

6.5.1.2 Supraîncărcarea procesorului

La proiectarea unui sistem se va lua în considerație frecvența de întrerupere și timpul utilizat de către rutinele de tratare a întreruperilor.

De exemplu, conectarea directă a unui buton la un pin de întrerupere externă ar putea însemna generarea unui număr mare de întreruperi inutile (din cauza fenomenului de bouncing).

6.5.1.3 Tipuri: *hard* și *soft*

Atât timp cât fenomenul cauză al întreruperii este de natură externă întreruperea este de tip *hard*. Există și posibilitatea ca printr-o metodă *software* să se satisfacă condiția de întrerupere. O întrerupere declanșată pe această cale va fi de tip *soft*.

De exemplu, dacă o întrerupere externă este activată (INT0, INT1, INT2) și pinul corespunzător este setat ca ieșire, atunci întreruperea poate fi declanșată prin scrierea pinului.

O altă cale prin care se poate genera *software* o întrerupere este de a seta bitul folosit de către platforma *hardware* pentru înregistrarea satisfacerii condiției de întrerupere.

6.5.1.4 Vectori de întrerupere

Vectorii de întrerupere sunt definiți în fișierul iom16.h, după cum urmează (adresele sunt exprimate în octeți):

```
#define RESET_vect      (0x00)
#define INT0_vect        (0x04)
#define INT1_vect        (0x08)
#define TIMER2_COMP_vect (0x0C)
#define TIMER2_OVF_vect  (0x10)
#define TIMER1_CAPT_vect (0x14)
#define TIMER1_COMPA_vect (0x18)
#define TIMER1_COMPB_vect (0x1C)
#define TIMER1_OVF_vect  (0x20)
#define TIMER0_OVF_vect  (0x24)
#define SPI_STC_vect     (0x28)
#define USART_RXC_vect   (0x2C)
#define USART_UDRE_vect  (0x30)
#define USART_TXC_vect   (0x34)
#define ADC_vect          (0x38)
#define EE_RDY_vect       (0x3C)
#define ANA_COMP_vect    (0x40)
#define TWI_vect          (0x44)
#define INT2_vect         (0x48)
#define TIMER0_COMP_vect (0x4C)
#define SPM_RDY_vect      (0x50)
```

6.5.2 DEFINIȚIA UNEI FUNCȚII DE ÎNTRERUPERE ÎN LIMBAJUL C

Pentru a defini o funcție de întrerupere pentru *Timer1 Overflow*, în limbajul C, se vor crea fișierele isr.h și isr.c după cum urmează:

Cod sursă 1:

```
isr.h
#ifndef __ISR__
#define __ISR__

#include <iom16.h> //include biblioteca iom16.h
#pragma vector = TIMER1_OVF_vect
//asocierea dintre o funcție și un vector de întrerupere
//vectorul întreruperii este dat sub forma unui simbol
//definit de regulă într-un fișier header aflat în biblioteca
//compilatorului
__interrupt void isr_TIMER1_overflow(void); //declararea funcției de
//întrerupere
#endif
```

isr.c

```
#include "isr.h" //include fișierul header de mai sus

__interrupt void isr_TIMER1_overflow(void)
{
    /* to do */
}
```

Se poate observa de mai sus că asocierea dintre o funcție și un vector de întrerupere se face cu ajutorul directivei `#pragma vector = [vectorul întreruperii]`, unde vectorul întreruperii este dat sub forma unui simbol definit de regulă într-un fișier *header* aflat în biblioteca compilatorului.

6.5.3 VERIFICĂRI LA NIVEL DE COD MAȘINĂ

În continuare se va considera exemplul funcției de întrerupere pentru fenomenul de *Timer1 Overflow* (a se vedea codul sursă 1).

Observație: în fișierul iom16.h `TIMER1_OVF_vect` este definit ca fiind numărul 0x20.

Specificația `#pragma vector = TIMER1_OVF_vect` face legătura între funcția de întrerupere `isr_TIMER1_overflow` și vectorul de întrerupere aflat la adresa 0x20.

Observație: În documentația tehnică a microprocesorului ATmega16 se spune că întreruperea *TIMER1 OVF* are asociată în zona de program adresa \$010. Această adresă are semnificația de număr de cuvinte.

Pentru fenomenul de *Timer1 Overflow*, dacă bitul TOIE1 din registrul TIMSK1 este 1 (întreruperea este validată), procesorul va executa instrucțiunea aflată la adresa 0x20 din memoria

Flash. Această instrucțiune reprezintă un salt la funcția de întrerupere corespunzătoare. Pentru a verifica acest lucru vom studia fișierul generat la compilare:

```
In segment CODE, align 2, keep-with-next
3      __interrupt void isr_TIMER1_overflow(void)
\          isr_TIMER1_overflow:
4  {
\ 00000000  930A      ST    -Y, R16
\ 00000002  B70F      IN    R16, 0x3F
5  /* to do */
6  }
\ 00000004  BF0F      OUT   0x3F, R16
\ 00000006  9109      LD    R16, Y+
\ 00000008  9518      RETI

\          In segment INTVEC, offset 0x20, root
\          `?isr_TIMER1_overflow??INTVEC 32`:
\ 00000020  .......   JMP   isr_TIMER1_overflow
7
```

Observăm faptul că, în segmentul vectorilor de întrerupere, la adresa 0x20, se află o instrucțiune Tabel , ce va efectua un salt la funcția `isr_TIMER1_overflow`, aflată în segmentul de cod, la o adresă relocabilă.

Adresa efectivă (adresa fizică din memoria Flash) a acestei instrucțiuni Tabel și a funcției de întrerupere poate fi căutată în fișierul Disassembler pus la dispoziție de mediul AVR Studio (menu = View, option = Disassembler):

```
+0000000F: 9518      RETI           Interrupt return
@00000010: _..X_CSTACK_SIZE
+00000010: 940C002A  JMP   0x0000002A  Jump
+00000012: 9518      RETI           Interrupt return
+00000013: 9518      RETI           Interrupt return
```

Se observă aşadar faptul că în memoria de cod la adresa 0x10 (număr de cuvinte) urmează o instrucțiune Tabel cu operandul 0x43 (număr de cuvinte). La adresa 0x2A se găsește de fapt funcția de întrerupere:

```
@0000002A: isr_TIMER1_overflow
---- isr.c -----
__interrupt void isr_TIMER1_overflow(void)
4  {
+0000002A: 930A      ST    -Y,R16     Store indirect and predecrement
+0000002B: B70F      IN    R16,0x3F   In from I/O location
6  }
+0000002C: BF0F      OUT   0x3F,R16   Out to I/O location
+0000002D: 9109      LD    R16,Y+     Load indirect and postincrement
+0000002E: 9518      RETI           Interrupt return
@0000002F: main
```

În mediul AVR Studio putem vizualiza direct memoria de cod (*menu = View, option = Memory Window*):

Memory										
Program	8/16	abc.	Address:	0x10	C					
000010	0C94	2A00	1895	1895	1895	1895	1895	1895	1895	1895
000019	1895	1895	1895	1895	1895	1895	1895	1895	1895	1895
000022	1895	1895	1895	1895	1895	1895	1895	1895	1895	0A93
00002B	0FB7	0FBF	0991	1895	00E0	10E0	0895	0000	8895	
000034	FECF	07EA	0DBF	00E0	0EBF	COE8	DOE0	OE94	4300	
00003D	0E94	2F00	0E94	3200	0C94	3200	01E0	0895	FFFF	
000046	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
00004F	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
000058	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
000061	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
00006A	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
000073	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	
00007C	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	

Observăm faptul că la adresa 0x10 (număr de cuvinte) se găsește un dublu-cuvânt cu valoarea 0C94 2A00. Această valoare capătă sens dacă se studiază codificarea instrucțiunii Tabel cu operandul 0x2A.

Remarcă: Cuvintele din codul mașină sunt scrise două câte două în ordine inversă.

6.5.3.1 Instrucțiunea JMP

Notă: aceste informații pot fi regăsite în fișierul *Instruction Set Nomenclature*.

Sintaxă	Operanți	Cod mașină			
JMP k	$0 \leq k < 4M$	1001	010k	kkkk	110k
		kkkk	kkkk	kkkk	kkkk

Tabel 6.5

Observație: Instrucțiunea Tabel se codifică într-un dublu-cuvânt.

Notă: În cazul în care există vectori de întrerupere pentru care nu s-au definit funcții de întrerupere, în zona de cod corespunzătoare se pun instrucțiuni Tabel (mai puțin pentru întreruperea de Reset pentru care se definește implicit o funcție de întrerupere). Acest lucru se poate observa în fișierul Disassembler sau analizând memoria de program și codificarea instrucțiunii Tabel .

6.5.3.2 Instrucțiunea RETI

Sintaxă	Operanți	Cod mașină			
RETI		1001	0101	0001	1000

Tabel 6.6

Observație: instrucțiunea Tabel se codifică într-un cuvânt.

6.5.3.3 Definiția unei funcții de întrerupere în limbaj de asamblare

Pentru a defini în limbaj de asamblare o rutină de întrerupere corespunzătoare, de exemplu, vectorului INT0, se poate utiliza codul sursă 2 (unde rutina de întrerupere va seta și, imediat, va reseta pinul 0 de la portul B):

Cod sursă 2 :

```
NAME EXT_INT0      // numele funcției
#include <iom16.h> // include biblioteca iom16.h

extern isr_INT0    //funcția externă utilizată

COMMON INTVEC(1)   //codul din segmentul vectorului de întrerupere
ORG INT0_vect     //regiunea care conține codul din vectorul de întrerupere
    jmp isr_INT0 //salt la funcția de întrerupere
ENDMOD

NAME CODE_int0 //numele funcției
#include <iom16.h> // include biblioteca iom16.h

set_B0 MACRO //macrodefiniția care setează pinul 0 de la portul B
    sbi 0x18, 0x00
ENDM

reset_B0 MACRO //macrodefiniția care resetează pinul 0 de la portul B
    cbi 0x18, 0x00
ENDM

PUBLIC isr_INT0 //declară isr_INT0 publică pentru a fi exportată într-o
                 //funcție C

RSEG CODE:ROOT // codul este relocabil
isr_INT0: //eticheta la care se face jump
    set_B0 //apeleză funcția care setează pinul 0 de la portul B
    reset_B0 // apeleză funcția care resetează pinul 0 de la portul B
reti
ENDMOD
END
```

6.5.4 MĂSURAREA LATENȚEI ÎNTRERUPERII

A măsura latența întreruperii înseamnă practic a examină intervalul de timp după care microprocesorul răspunde la o întrerupere externă (care este probabil același pentru toate tipurile de întreruperi).

Răspunsul la o întrerupere durează minim 4 cicli, timp în care se salvează pe stivă PC (*Program Counter*) și se face posibilă executarea instrucțiunii aflate în vectorul de întrerupere

corespunzător. Această instrucțiune este de regulă Tabel (deci încă 3 cicli) către rutina propriu-zisă de întrerupere.

Pentru a măsura timpul de răspuns la o întrerupere externă s-a utilizat o funcție de întrerupere scrisă în limbaj de asamblare (codul sursă 2).

O descriere **aproximativă** în domeniul timp a ceea ce se întâmplă până la apariția răspunsului funcției de întrerupere este dată în Diagrama 6.1:

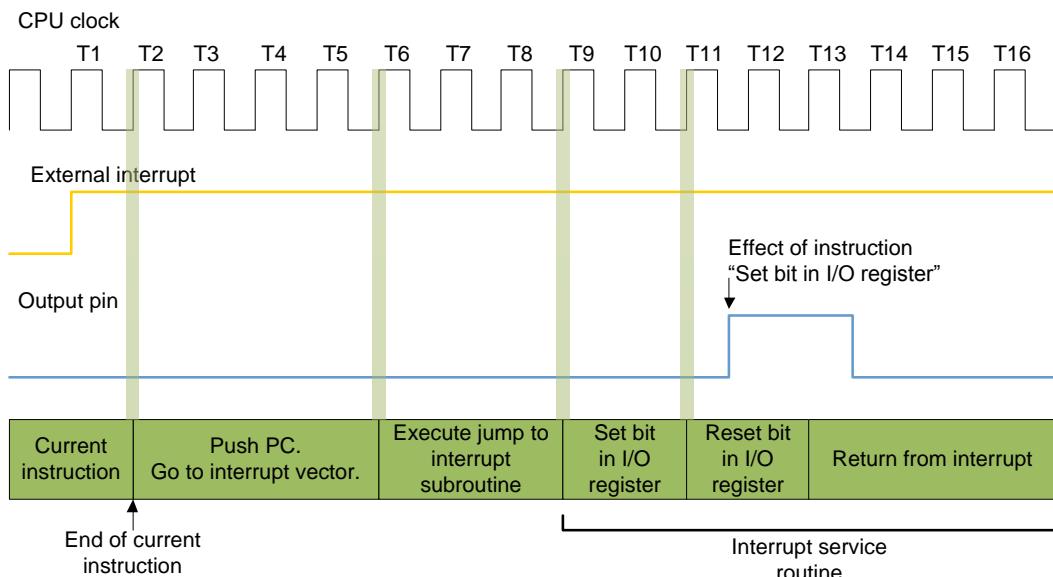


Diagrama 6.1

S-a măsurat durata de timp dintre fronturile crescătoare a 2 semnale:

- ✓ intrarea pentru INT0 (port D, pin 2);
- ✓ ieșirea modificată în rutina de întrerupere (port B, pin 0).

Notă: Frecvența procesorului este de 4MHz.

Tensiunea de intrare pentru INT0 a fost luată de la un generator de pulsuri cu o frecvență de 10KHz și un factor de umplere de 20% (frecvența pulsurilor trebuie să fie mai mică de 500MHz, iar durata pulsului trebuie să fie mai mare de 50ns).

S-au obținut următoarele imagini (unda de culoare albastră este semnalul de excitație, iar unda de culoare roșie este semnalul răspuns):

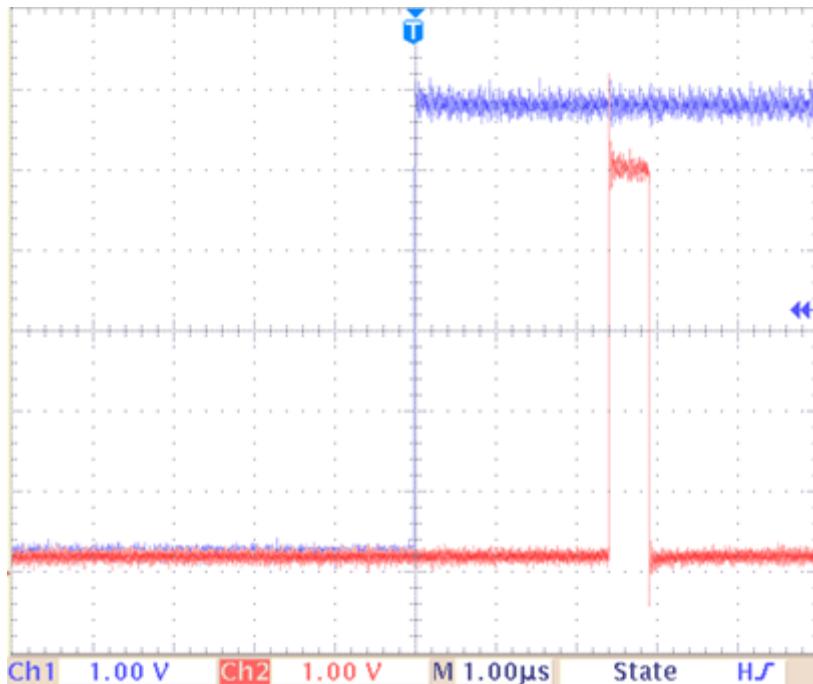


Figura 6.6

Observații:

- efectul funcției de întrerupere se produce cu aproximativ $2.4\mu s$ după condiția de întrerupere (fenomenul cauză). Acest timp corespunde a 9-10 cicli mașină. Scrierea unui pin de ieșire durează 2 cicli;
- intervalul de timp cât pinul de ieșire se află pe 1 logic este egală cu aproximativ 500ns (a se vedea figura 6.6). Acest timp corespunde a 2 cicli mașină, ceea ce pare a fi normal, dacă se consideră faptul că instrucțiunile SBI (*set bit in I/O register*) și CBI (*clear bit in I/O register*) durează, fiecare, 2 cicli mașină.

În *Figura* este redat rezultatul același program, doar că a fost mărită persistența în timp a semnalelor (există o astfel de opțiune la osciloscop).

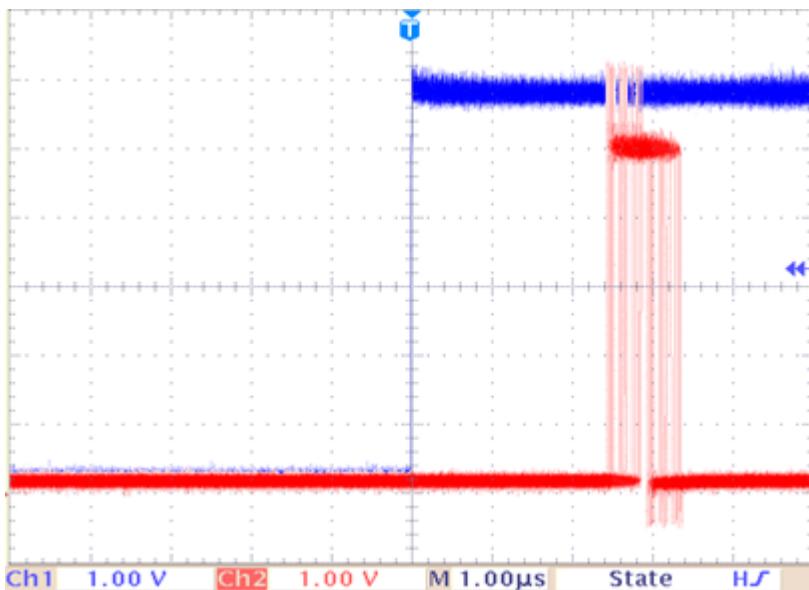


Figura 6.7

Observații:

- timpul de răspuns al procesorului nu este constant, ci variază într-un interval de circa $0.5\mu s$ (aşa-zisul *jitter*). *Jitter*-ul este dat de faptul că microprocesorul întotdeauna va termina execuția curentă și apoi va trata întreruperea, iar timpul când se înregistrează condiția de întrerupere nu este sincronizat cu sfârșitul execuției instrucțiunii curente. Instrucțiunea ce se execută la infinit este (se poate vedea în fișierul *Disassembler*) RJMP (*relative jump* – se sare înapoi cu o instrucțiune) și durează 2 cicli. Rezultă deci, că ne putem aștepta la un *jitter* de maxim 500ns, ceea ce corespunde cu rezultatele obținute (unde se poate observa un interval de aprox. 450ns).

În cazul în care vom defini o funcție de întrerupere în limbajul C, compilatorul va „îmbrăcă” corpul funcției cu instrucțiuni ce vor avea ca efect „conservarea” stării microprocesorului (contextul de execuție).

În *Figura* se poate vedea efectul unei funcții de întrerupere definită în C în care se setează și se resetează pinul 0 de la portul B. Funcția de întrerupere are deci aceeași funcționalitate cu rutina definită în limbaj de asamblare mai sus, însă modificarea pinului se va produce mai târziu, deoarece este salvat registrul de stare al microprocesorului (Figura 6.2).

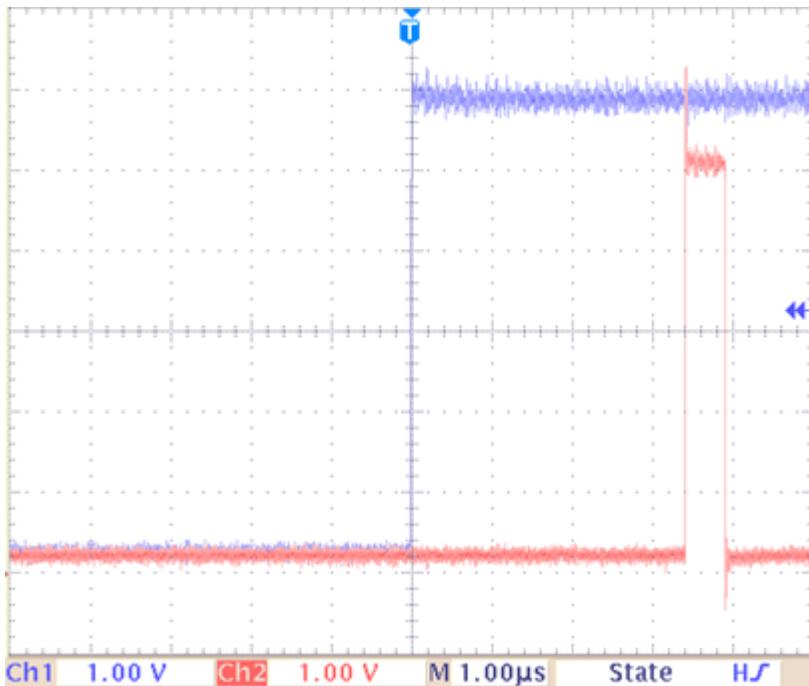


Figura 6.8

De regulă, într-o funcție de întrerupere scrisă în C, compilatorul va salva:

- ✓ registrul de stare, întotdeauna;
- ✓ regiștrii cu funcție generală, dacă sunt modificați.

6.5.5 DECLARAREA UNEI FUNCȚII DE ÎNTRERUPERE ÎN C

Pentru a declara o funcție de întrerupere în asamblare și a defini-o în C, de exemplu pentru vectorul INT1, se poate utiliza ca suport:

Cod sursă 3 :

main.c

```
#include <inavr.h> //include biblioteca inavr.h
#include <iom16.h> //include biblioteca iom16.h

int main( void )
{
    /* INT1 este pe pinul PD3 */
    DDRD = 0xFF; // setează ca ieșire
    PORTD = 0xFF;

    MCUCR |= ((1 << ISC11) | (1 << ISC10)); /* Pe frontul pozitiv al lui INT1
                                                se generează o cerere de întrerupere */
```

```
GICR |= (1 << INT1); /* activează întreruperea externă INT1 */  
    __enable_interrupt(); //activează întreruperea globală  
  
    while (1)  
    {  
        PORTD = ~PORTD; // generează o întrerupere software  
    }  
}
```

INT1_definition.c

```
#include <iom16.h> //include biblioteca iom16.h  
  
/* declararea acestei funcții se găsește în „INT1_declaration.asm” */  
_interrupt void isr_INT1(void)  
{  
    unsigned char test = 0;  
    test += 1; // putem pune un breakpoint aici  
}
```

Notă: Compilatorul dă un mesaj de avertisment în fișierul **INT1_definition.c** referitor la faptul că funcția **isr_INT1** este definită că funcție de întrerupere, însă nu se cunoaște vectorul de întrerupere asociat.

7 Coduri redundante ciclice

7.1 INTRODUCERE

Codurile Redundante Ciclice sunt o metodă de detectare a erorilor folosită pe larg în rețelistică și dispozitive de stocare pentru a detecta modificările accidentale ale datelor. Blocurile de date binare care circulă în interiorul acestor sisteme au atașate câte o valoare de control, care este de fapt restul unei împărțiri polinomiale a conținutului lor; la recepție calculul se repetă și se pot face corecții împotriva coruperii datelor în cazul în care valorile de control nu coincid.

Codurile Redundante Ciclice se numesc astfel pentru că valoarea de control este o redundanță (aceasta mărește lungimea mesajului fără a adăuga informație), iar algoritmul se bazează pe coduri ciclice. CRC sunt atât de populare pentru că sunt simplu de implementat în dispozitivele digitale, ușor de analizat matematic și, în particular, foarte eficiente la detectarea erorilor produse din cauza zgromotului în canalele de transmitere. Datorită faptului că valoarea de control are o lungime fixă, funcția care generează această valoare este adesea folosită ca o funcție hash. Controlul redundant ciclic a fost inventat de către W. Wesley Peterson în 1961; polinomul pe 32 biți utilizat în funcția CRC a standardului Ethernet și multor altora este rezultatul muncii mai multor cercetători și a fost publicat în 1975.

Codurile ciclice nu sunt doar simplu de implementat, dar sunt și foarte convenabile pentru detectarea erorilor de tip „explozie”, a secvențelor continue de date eronate, în diferite tipuri de canale de comunicație, inclusiv dispozitive de stocare optice și magnetice. De obicei, un CRC pe n biți, aplicat unui bloc de date de lungime arbitrară, va detecta oricare eroare cu lungimea mai mică de n biți și o mică parte $1-2^{-n}$ din erorile cu o lungime mai mare.

Specificațiile unui CRC necesită definirea unui așa-numit „polinom generator”. Acest polinom reprezintă împărțitorul în cadrul împărțirii polinomiale, unde mesajul este deîmpărțitul, câtul este ignorat, iar restul reprezintă valoarea de control (codul CRC propriu-zis). Deosebirea cea mai importantă în cadrul acestei împărțiri este folosirea aritmetică modulo 2, adică a operatorului XOR. Lungimea „restului” obținut este tot timpul mai mică decât lungimea polinomului generator, ceea ce determină lungimea mesajului rezultat. Cel mai simplu mod de detectare a erorilor, bitul de paritate, este de fapt un CRC pe 1 bit ce folosește polinomul generator $x + 1$.

Un dispozitiv ce utilizează CRC, calculează o secvență binară de lungime fixă pentru fiecare bloc de date care urmează să fie transmis și să atașezează acestuia, formând astfel un cuvânt de cod. Când un cuvânt de cod este recepționat sau citit, dispozitivul compară valoarea de control cu valoarea rezultată din calculul CRC asupra blocului de date, sau efectuează o împărțire polinomială asupra întregului mesaj (împreună cu valoarea de control) și compară rezultatul cu zero. Dacă aceste valori nu coincid, atunci mesajul conține o eroare. Dispozitivul poate acționa în diferite moduri pentru a corecta eroarea, de exemplu să mai citească o dată blocul de date sau să ceară ca blocul de date să fie transmis din nou.

7.2 CRC ȘI INTEGRITATEA DATELOR

Codurile ciclice au fost proiectate să protejeze datele de erorile cel mai des întâlnite în canalele de comunicație. În acest domeniu, CRC oferă o siguranță rezonabilă a mesajelor transmise. Totuși, aceste coduri nu sunt convenabile contra alterării intenționate a datelor.

În primul rând, CRC nu necesită nici un fel de autentificare, adică un atacator poate edita mesajul și recalcule suma de control fără ca această „substituție” să fie detectată. Dacă sunt stocate alături de date, codurile redundante ciclice și funcțiile criptografice hash nu protejează împotriva alterării intenționate a datelor. Orice aplicație care necesită protecție împotriva acestui tip de atacuri trebuie să utilizeze mecanisme de autentificare, ca mesaje de autentificare sau semnături digitale (care sunt adesea bazate pe funcții criptografice hash).

În al doilea rând, spre deosebire de funcțiile criptografice hash, CRC este o funcție ușor reversibilă, ceea ce o face inconvenabilă pentru a fi folosită în semnăturile digitale.

În al treilea rând, CRC este o funcție liniară cu proprietatea că $crc(x \oplus y) = crc(x) \oplus crc(y)$ și, ca rezultat, chiar dacă CRC-ul ar fi criptat, mesajul și CRC-ul asociat ar putea fi manipulate fără cunoașterea cheii de criptare; acesta este un neajuns bine-cunoscut al protocolului Wired Equivalent Privacy (WEP).

7.3 SPECIFICAȚII CRC

Conceptul de CRC referitor la coduri de detectare a erorilor se complică atunci când un dezvoltator sau comisie de standardizare îl folosesc la proiectarea unui sistem practic. Acestea sunt unele dintre complicațiile care apar:

- Uneori implementarea adaugă un sir fix de biți înaintea șirului care trebuie verificat. Acest fapt este util atunci când erorile de sincronizare „insereză” biți de zero în fața unui mesaj fără a modifica șirul ce va fi verificat;
- De obicei, dar nu întotdeauna, o implementare atașează încă N biți (N fiind dimensiunea CRC-ului) la șirul de biți înainte ca acesta să fie divizat cu polinomul generator. Acest fapt este convenabil, pentru că astfel restul împărțirii șirului de biți original la valoarea de control este exact zero, deci CRC-ul poate fi verificat prin simpla împărțire polinomială a șirului de biți recepționat și compararea valorii obținute cu zero. Datorită proprietăților de asociativitate și comutativitate a operatorului XOR, implementările ce utilizează tabele pot obține un rezultat numeric egal cu zero fără a atașa zerourile explicit, prin folosirea unui algoritm echivalent mai rapid care combină mesajul inițial cu șirul rezultat din registrul pentru CRC;
- Unele implementări aplică operatorul XOR asupra restului divizării polinomiale;
- Ordinea bițiilor: Unele scheme „văd” cel mai nesemnificativ bit dintr-un octet ca fiind „primul”, ceea ce înseamnă „cel mai din stânga” în timpul împărțirii polinomiale. Această convenție are sens atunci când mesajele transmise pe interfața serială sunt verificate cu CRC în interiorul dispozitivelor, datorită răspândirii pe larg a convenției conform căreia cel mai nesemnificativ bit este transmis primul;
- Ordinea octetelor: La CRC-urile pe mai mulți octeți, poate apărea o confuzie cu privire la primul octet transmis (sau octetul stocat la adresa mai mică în memorie) dacă acesta este cel mai semnificativ octet (LSB) sau cel mai semnificativ octet (MSB). De exemplu, unele implementări ale CRC interschimbă octetii polinomului generator;
- Omiterea celui mai semnificativ bit al polinomului generator. Datorită faptului că cel mai semnificativ bit este tot timpul 1 și pentru că un CRC pe N biți trebuie definit printr-un

- divizor pe $N+1$ biți, unii dezvoltatori cred că este inutil de menționat bitul cel mai semnificativ al divizorului;
- Omiterea celui mai nesemnificativ bit al polinomului generator. Datorită faptului că cel mai nesemnificativ bit este tot timpul 1, unii autori ca Philip Koopman reprezintă polinomul divizor cu bitul cel mai semnificativ intact, dar fără cel mai nesemnificativ bit (x^0). Această convenție codează polinomul împreună cu gradul său într-un singur întreg.
- Aceste complicații înseamnă că există trei modalități uzuale de a exprima polinomul ca un întreg: primele două sunt constantele regăsite în cod, cea de-a treia este numărul regăsit în materialele lui Koopman. În orice caz, un termen este omis. Deci, polinomul $x^4 + x + 1$ poate fi transcris ca:
- $0x3 = 0b0011$, reprezentând $x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0$ (MSB-first code)
 - $0xC = 0b1100$, reprezentând $1x^0 + 1x^1 + 0x^2 + 0x^3 + x^4$ (LSB-first code)
 - $0x9 = 0b1001$, reprezentând $1x^4 + 0x^3 + 0x^2 + 1x^1 + x^0$ (notația Koopman)

Reprezentări		
Normală	Inversată	Inversată reciproc (Koopman)
0x3	0xC	0x9

Tabel 7.1

7.4 CRC STANDARDIZATE

Numerose variații de control redundant ciclic au fost încorporate în standarde tehnice. Niciodată un anumit algoritm nu poate satisface toate scopurile/necesitățile. De aceea, Koopman și Chakvarty recomandă selectarea unui polinom ce corespunde cerințelor aplicației și lungimii mesajelor. Numărul mare al CRC-urilor distințe a zăpăcit dezvoltatorii și acești autori au încercat să rezolve această situație. Există trei polinoame pentru CRC-12, șaisprezece polinoame conflictuale pentru CRC-16 și șase pentru CRC-32.

Tabelul de mai jos prezintă doar polinoamele utilizate în diferiți algoritmi. Diferite variații ale aceluiași protocol pot implica inversarea, post-inversarea și inversarea completă a biților. De exemplu, CRC-32 folosit în Gzip și Bzip2 au același polinom, dar Bzip2 implică inversarea ordinii biților.

Nume	Reprezentări		
	Normală	Inversată	Inversată reciproc
CRC-1	0x1	0x1	0x1
CRC-4-ITU	0x3	0xC	0x9
CRC-5-EPC	0x09	0x12	0x14
CRC-5-USB	0x05	0x14	0x12
CRC-6-ITU	0x03	0x30	0x21
CRC-7	0x09	0x48	0x44
CRC-8-CCITT	0x07	0xE0	0x83
CRC-8	0xD5	0xAB	0x8E
CRC-8-Dallas/Maxim	0x31	0x8C	0x98
CRC-8-SAE J1850	0x1D	0xB8	0x8E
CRC-8-WCDMA	0x9B	0xD9	0xCD
CRC-10	0x233	0x331	0x319
CRC-11	0x385	0x50E	0x5C2
CRC-12	0x80F	0xF01	0xC07
CRC-15-CAN	0x4599	0x4CD1	0x62CC
CRC-15-MPT1327	0x6815	0x540B	0x740A
CRC-16-IBM (ANSI)	0x8005	0xA001	0xC002
CRC-16-CCITT	0x1021	0x8408	0x8810
CRC-16-T10-DIF	0x8BB7	0xEDD1	0xC5DB
CRC-16-DNP	0x3D65	0xA6BC	0x9EB2
CRC-16-DECT	0x0589	0x91A0	0x82C4
CRC-16-ARINC	0xA02B	0xD405	0xD015
CRC-24	0x5D6DCB	0xD3B6BA	0xAEB6E5
CRC-32	0x04C11DB7	0xEDB88320	0x82608EDB

Tabel 7.2

7.5 NOȚIUNI DESPRE CALCULUL CRC

7.5.1 SOFTWARE

Pentru a realiza o aplicație software pentru calculul CRC există mai multe metode de implementare, în funcție de:

- ipotezele de la care se pornește calculul;
- dimensiunea polinomului generator;
- dimensiunea și structura mesajului pentru care se calculează CRC-ul;
- timpul în care se generează CRC-ul;
- dimensiunea spațiului de memorie alocat;
- performanțele procesorului de calcul.

Metodele de implementare se pot clasifica în două categorii:

- algoritmi de viteză redusă;
- algoritmi de mare viteză.

Pentru implementarea software a unui algoritm CRC, va trebui realizată implementarea împărțirii în binar folosite de aritmetică CRC. Instrucțiunea de împărțire a unui calculator nu poate fi folosită deoarece împărțirea CRC nu este același lucru cu împărțirea normală și datorită dimensiunii mesajului, întrucât acesta poate ajunge la dimensiuni de ordinul MB, iar procesoarele actuale nu folosesc registre atât de mari. Pentru implementare, trebuie să existe un registru de deplasare, având dimensiunea egală cu gradul polinomului generator în care să se afle biții mesajului. Prelucrarea mesajului se va face bit cu bit. O altă metodă de implementare presupune existența unui tabel în care se găsesc biții polinomului CRC deplasați. Pentru a micșora timpul de execuție s-a trecut la procesarea în același timp cantități mai mari de biți (prelucrare paralelă): semi-octeți (4biți), octeți (8biți), cuvinte (16biți) și dublu-cuvinte (32biți). Dintre acestea, prelucrarea semi-octetilor este evitată deoarece calculatoarele operează cu octeți. Pentru mărirea vitezei de execuție majoritatea implementărilor operează cu octeți sau cu multipli ai acestora.

7.5.2 HARDWARE

Implementarea hardware a CRC sub forma unui sistem ce are la bază un microcontroler are următoarele avantaje:

- Permite modificarea cu ușurință a metodei de calcul;
- Permite o introducere și prelucrare simplă a datelor înainte de a fi supuse calculului;
- Permite comunicația cu diferite periferice;
- Permite comunicația cu calculatorul prin intermediul porturilor cum ar fi cel serial;
- Pentru obținerea unor tempi foarte mici se folosește o implementare de tip hardware cu bistabile în varianta paralelă.

7.5.3 EXEMPLU DE CALCUL CRC

Pentru calculul unui CRC pe n biți, se vor poziționa în partea stângă a mesajului inițial cei $n+1$ biți ai polinomului generator.

Se pornește de la mesajul inițial:

```
11010011101100
```

Se completează mesajul inițial cu n zerouri corespunzătoare celor n biți de la CRC. Mai jos sunt prezentate calculele pentru un CRC pe 3 biți:

```
11010011101100 000 <--- completăm mesajul cu cei 3 biți
1011           <--- divizorul (4 biți) =  $x^3+x+1$ 
-----
01100011101100 000 <--- rezultatul
```

Dacă bitul deasupra bitului cel mai semnificativ din divizor este 0, nu trebuie făcute calcule. Dacă bitul din mesajul inițial deasupra bitului cel mai semnificativ din divizor este 1, se face un XOR între mesajul inițial și divizor. Apoi divizorul este mutat cu o poziție în dreapta și procesul se repetă până când divizorul ajunge în partea dreaptă a mesajului inițial.

Mai jos este prezentat calculul complet:

```
11010011101100 000 <--- mesajul este completat cu 3 biți
1011           <--- divizor
01100011101100 000 <--- rezultat
1011           <--- divizor ...
00111011101100 000
 1011
00010111101100 000
 1011
00000001101100 000
 1011
00000000110100 000
 1011
00000000011000 000
 1011
00000000001110 000
 1011
00000000000101 000
 101 1
-----
0000000000000000 100 <--- rest (3 biți)
```

Restul obținut reprezintă valoarea propriu-zisă a funcției CRC. Pentru verificarea validității unui mesaj primit, acesta este divizat cu polinomul generator. Dacă nu există erori detectabile, restul obținut trebuie să fie zero.

```
11010011101100 100 <--- mesajul cu valoarea de control
1011           <--- divizor
01100011101100 100 <--- rezultat
1011           <--- divizor ...
```

```

00111011101100 100
.....
00000000001110 100
      1011
00000000000101 100
      101 1
-----
          0 <--- rest

```

7.6 POINTERI ÎN IAR

7.6.1 POINTERI ȘI TIPURI DE MEMORIE

Pointerii sunt folosiți pentru a referi locația datelor. În general, pointerii au un tip. De exemplu, un pointer de tipul `int *` indică către un întreg.

În compilator, pointerul indică către un anumit tip de memorie. Tipul memoriei este specificat utilizând un cuvânt cheie înainte de asterisc. De exemplu un pointer care indică către un întreg stocat în memoria „far” este declarat astfel:

```
int _ _far * MyPtr;
```

Trebuie menționat faptul că locația variabilei pointer `MyPtr` nu este afectată de cuvântul-cheie care precede asteriscul. În exemplul următor variabila `MyPtr2` este plasată în memoria „tiny”. Ambele variabile, `MyPtr` și `MyPtr2`, indică către o dată de tip caracter din memoria „far”.

```
char _ _far * _tiny MyPtr2;
```

Oricând este posibil, pointerii trebuie declarați fără atribute de memorie. De exemplu, funcțiile din biblioteca standard toate sunt declarate fără specificarea explicită a tipului de memorie.

7.6.2 DIFERENȚE ÎNTRE TIPURI DE POINTERI

Un pointer trebuie să conțină informația necesară pentru a specifica locația unui anumit tip de memorie. Acest fapt înseamnă că dimensiunile pointerilor sunt diferite pentru diferite tipuri de memorie. În IAR C/C++ Compiler for AVR este interzisă conversia pointerilor de tipuri diferite fără utilizarea unui cast explicit.

7.6.2.1 Pointeri la funcții

Dimensiunea unui pointer la funcție este tot timpul 16 sau 24 de biți și aceștia pot adresa întreaga memorie. Reprezentarea internă a unui pointer la funcție reprezintă adresa de la care începe funcția împărțită la doi.

În IAR sunt disponibile următoarele tipuri de pointeri la funcții:

Cuvânt-cheie	Interval de memorie	Dimensiune pointer	Tip index	Descriere
<code>_nearfunc</code>	0-0x1FFFFE	2 octeți	signed int	Poate fi apelat din oriunde în memoria program, dar trebuie să refere o locație din primii 128KB ai aceluia spațiu de memorie.
<code>_farfunc</code>	0-0x7FFFFFFE	3 octeți	signed long	Poate fi apelat de oriunde.

Tabel 7.3

7.6.2.2 Pointeri la date

Pointerii la date pot avea trei dimensiuni: 8, 16 sau 24 biți. Pointerii la date disponibili sunt:

Cuvânt-cheie	Dimensiune pointer	Spațiul de memorie	Tipul indicelui	Intervalul de memorie
<code>_tiny</code>	1 octet	Data	signed char	0x0-0xFF
<code>_near</code>	2 octeți	Data	signed int	0x0-0xFFFF
<code>_far</code>	3 octeți	Data	signed int	0x0-0xFFFFFFFF
<code>_huge</code>	3 octeți	Data	signed long	0x0-0xFFFFFFFF
<code>_tinyflash</code>	1 octet	Code	signed char	0x0-0xFF
<code>_flash</code>	2 octeți	Code	signed int	0x0-0xFFFF
<code>_farflash</code>	3 octeți	Code	signed int	0x0-0xFFFFFFFF
<code>_hugeflash</code>	3 octeți	Code	signed long	0x0-0xFFFFFFFF
<code>_eeprom</code>	1 octet	EEPROM	signed long	0x0-0xFF
<code>_eeprom</code>	2 octeți	EEPROM	signed int	0x0-0xFFFF

Tabel 7.3

7.7 IMPLEMENTARE CRC ÎN IAR/AVR

7.7.1 CONFIGURAREA MEDIULUI IAR

Pentru a genera în mod automat CRC-ul pentru toată memoria flash, mediul IAR trebuie configurat ca în figura următoare:

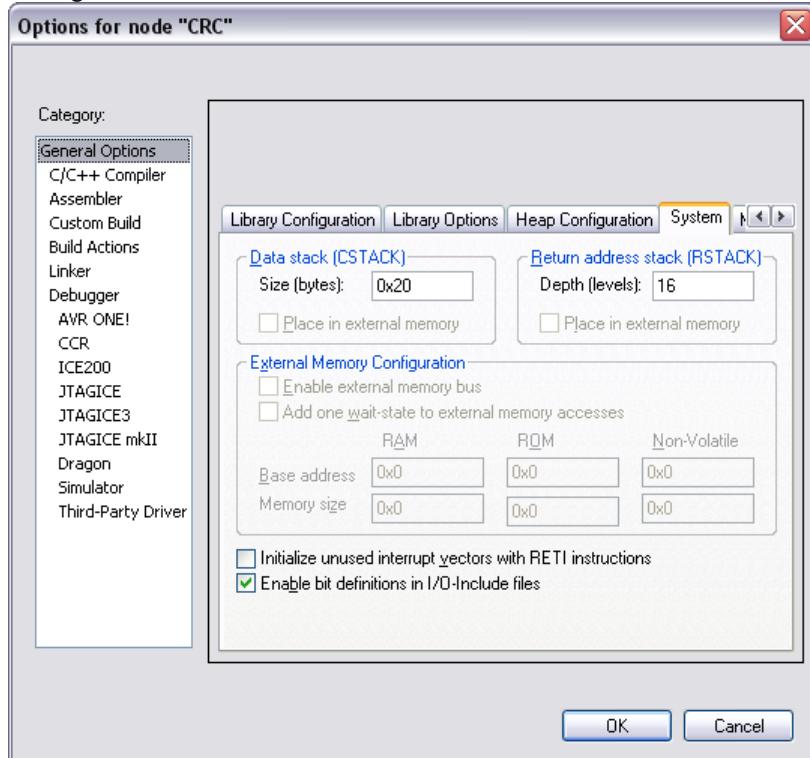


Figura 7.1

Se lasă debifată căsuța *Initialize unused interrupt vectors with RETI instructions* pentru a evita un conflict cu opțiunea *Fill unused code memory* din categoria Linker.

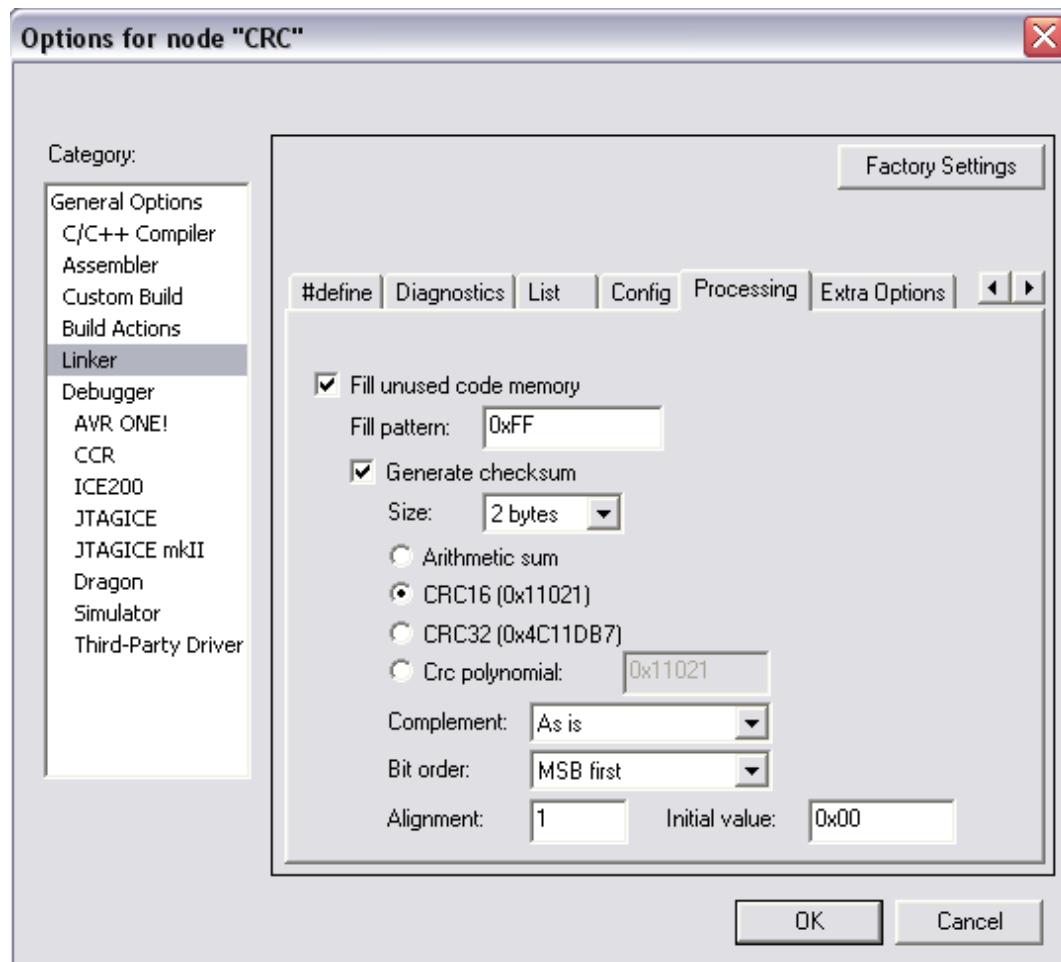


Figura 7.2

- **Fill unused code memory** – setează valoarea cu care se completează memoria neutilizată.
- **Generate checksum** – setează generarea CRC-ului.
- **Size** – setează dimensiunea CRC-ului generat în octeți.
- **Arithmetic sum** – calculează suma tuturor biților de 1.
- **CRC polynomial** – setează un polinom propriu pentru generarea CRC-ului.
- **Complement** – setează modul de reprezentare al CRC-ului generat:
 - **As is** – rezultatul rămâne neschimbat.
 - **1's Complement** – complement față de 1.
 - **2's Complement** – complement față de 2.
- **Bit order** – modul de reprezentare al CRC-ului generat.
- **LSB first** – primul bit reprezintă coeficientul termenului la puterea 0.
- **MSB first** – primul bit reprezintă coeficientul termenului la puterea cea mai mare.
- **Initial value** – setează valoarea inițială a CRC-ului.

Implementare CRC-16

Setări IAR:

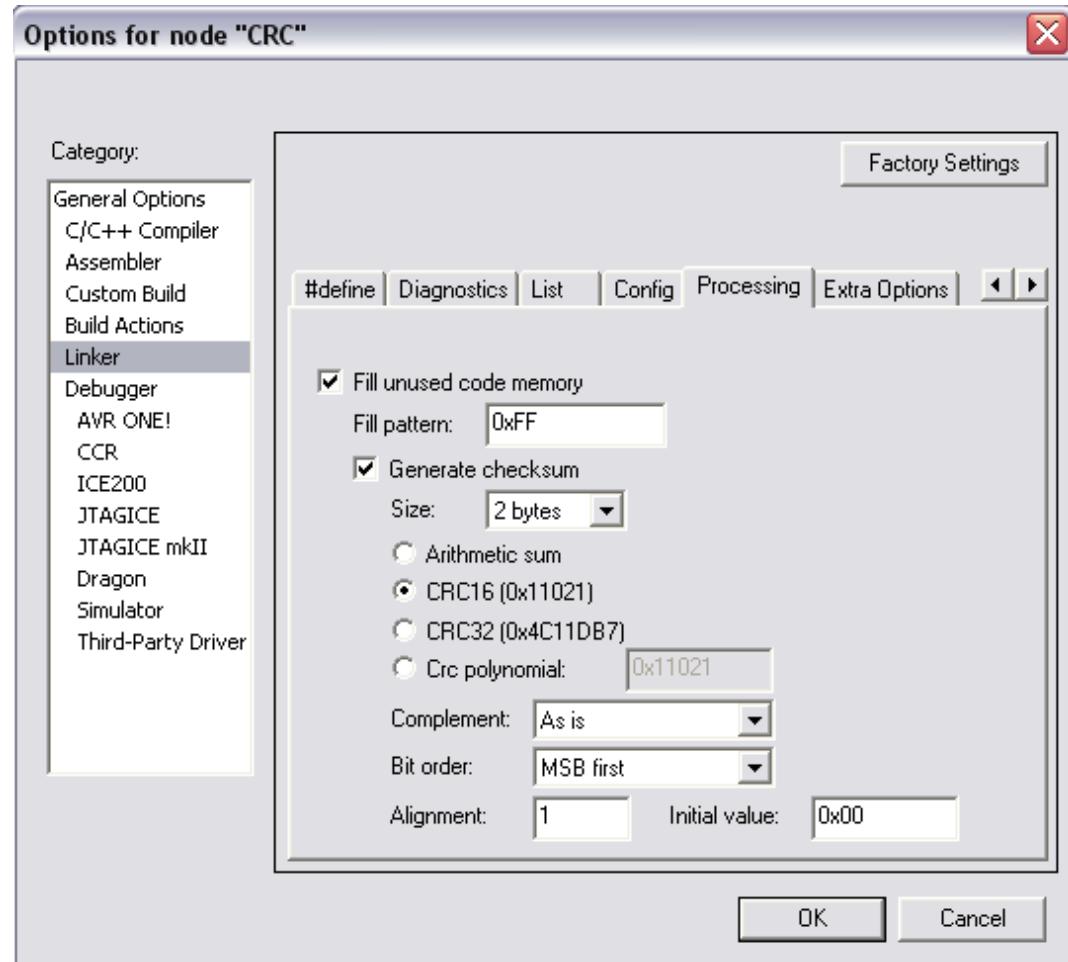


Figura 7.3

Rezultatele rulării în AVR:

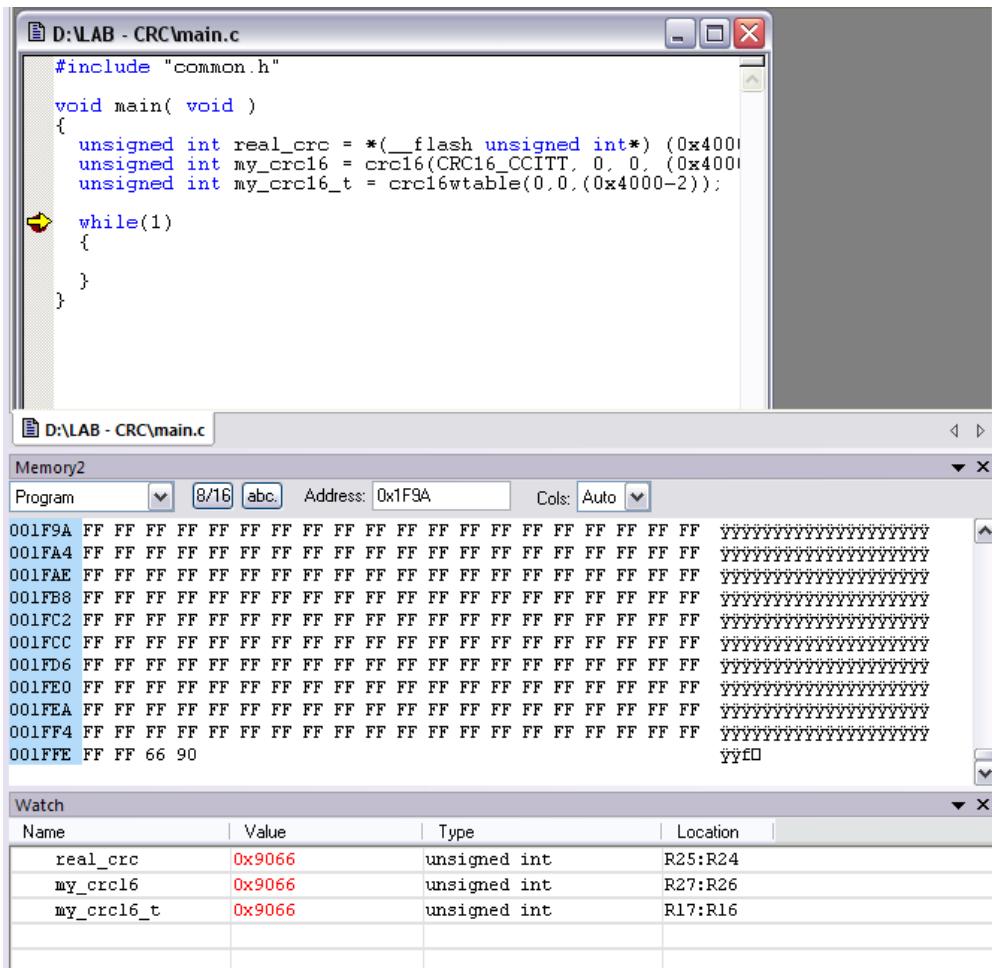


Figura 7.4

Cod sursă 7.1:

```
#define CRC16_CCITT 0x1021
enum BitOrder { LSBF, MSBF };

unsigned int crc16(unsigned int polinom16, unsigned int init_val_16,
                  unsigned int adr_start, unsigned int len, enum BitOrder ord)
{
    //rezultatul final
    unsigned int crc = init_val_16;
    //reține în octetul cel mai semnificativ datele
    //extrase din memoria flash
    unsigned int data = 0;
```

```

while( len-- ) {
    unsigned int i;
    //se extrage valoarea unui octet de la adresa de start
    //din memoria flash
    data = *(__flash char *)adr_start;
    if ( ord == MSBF ) //optiunea cu shiftare spre MSB
    {
        //octetul este shiftat la stanga pentru a se alinia
        //cu polinomul generator
        data <<= 8;
        //datele sunt "transferate" in rezultat
        crc ^= data;
        adr_start++;
        //pentru bitii de date se face XOR cu polinomul generator,
        //daca bitul cel mai semnificativ este 1 sau se shifta datele
        //la stanga, daca bitul cel mai semnificativ este 0
        for( i = 0; i < 8; ++i ) {
            //se verifică dacă bitul cel mai semnificativ este 1
            if( crc & 0x8000 )
                crc = (crc << 1) ^ polynom16;
            else
                crc = crc << 1;
        }
    }
    else
        //optiunea cu shiftare spre LSB
        //parametrul polynom16 al functiei trebuie sa aiba bitii inversati in
        //prealabil
    {
        crc ^= data;
        adr_start++;
        //se verifică bitul cel mai putin semnificativ si daca
        //acesta este 1 se face XOR cu polinomul generator,
        //altfel datele sunt shiftate la dreapta
        for( i = 0; i < 8; ++i ) {
            //se verifică daca cel mai putin semnificativ bit este 1
            if( crc & 0x0001 )
                crc = (crc >> 1) ^ polynom16;
            else
                crc = crc >> 1;
        }
    }
}
return crc;
}

```

```

//aceasta implementare utilizează o tabelă cu valori pre-calculate ale
//funcției CRC16

__flash const unsigned int crc16tab[256] = {
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x2210, 0x3273, 0x4252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x4042, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbff, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0cfffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfbfa, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

unsigned int crc16wtable(unsigned int init_val_16, unsigned int adr_start,
                        unsigned int len)
{
    unsigned int counter;
    unsigned int crc = init_val_16;
    for( counter = 0; counter < len; counter++)
        crc = (crc<<8) ^ crc16tab[((crc>>8) ^ *(__flash char
*)adr_start++) & 0x00FF];
    return crc;
}

```

7.7.2 IMPLEMENTARE CRC-32

Setări IAR:

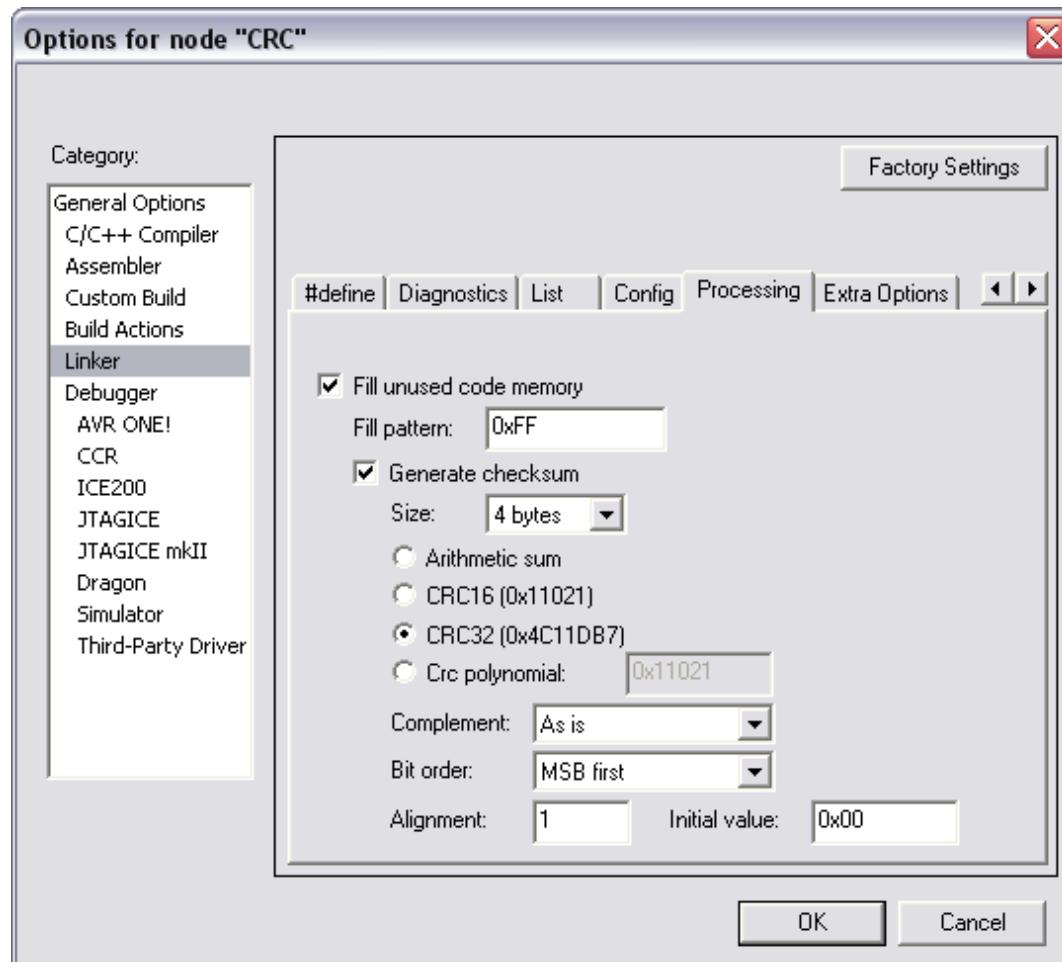


Figura 7.5

Rezultatele rulării în AVR:

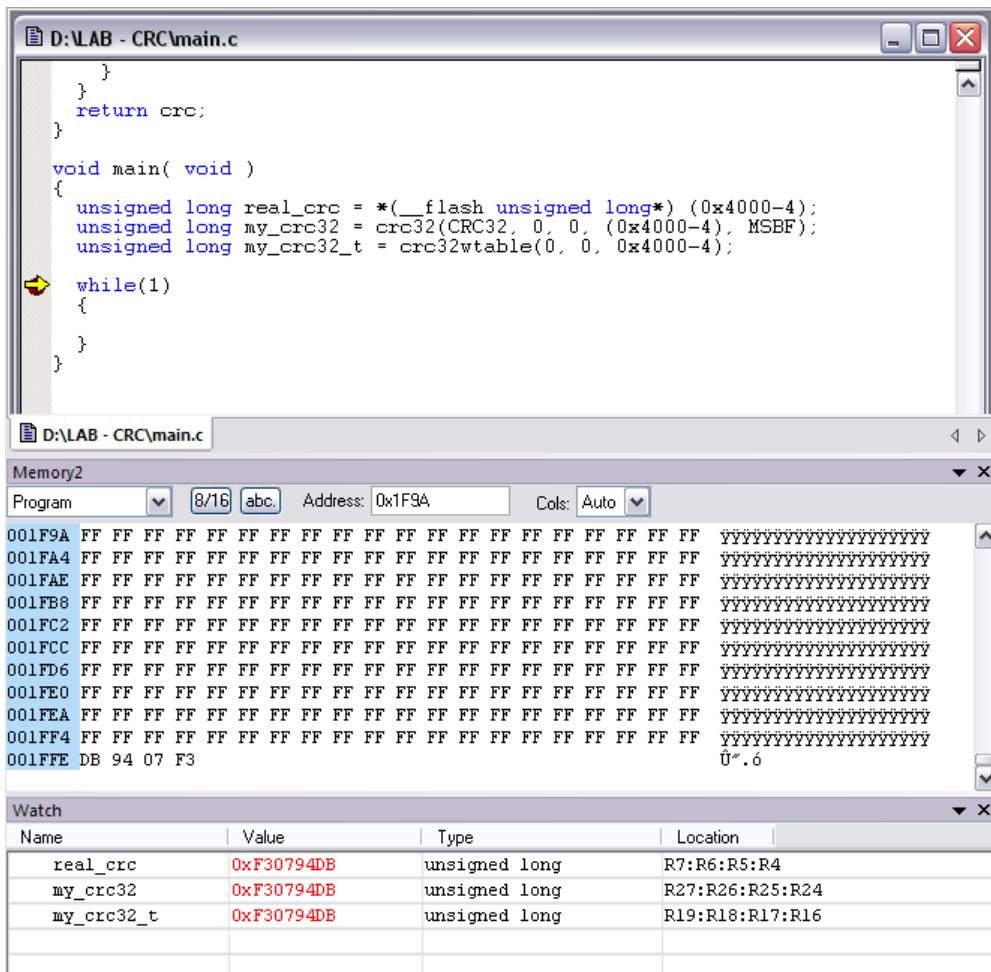


Figura 7.6

Cod sursă 7.2:

```
unsigned long crc32(unsigned long polinom32, unsigned long init_val_32,
                     unsigned int adr_start, unsigned int len, enum BitOrder
                     ord)
{
    //variabile pe 32 biți
    unsigned long crc = init_val_32;
    unsigned long data = 0;

    while( len-- ) {
        int i;
        //se extrage un octet din memoria flash
```

```

data = *(__flash char *)adr_start;
if ( ord == MSBF ) //optiunea cu shiftare spre MSB
{
    //octetul extras este shiftat pana la cel mai semnificativ bit
    data <= 24;
    crc ^= data;
    adr_start++;
    for( i = 0; i < 8; ++i ) {
        //se verifică dacă cel mai semnificativ bit este 1
        if( crc & 0x80000000 )
            crc = (crc << 1) ^ polinom32;
        else
            crc = crc << 1;
    }
}
else
//optiunea cu shiftare spre LSB
//parametrul polinom32 al funcției trebuie să aibă biții inversați în
prealabil
{
    crc ^= data;
    adr_start++;
    for( i = 0; i < 8; ++i ) {
        //se verifică dacă cel mai puțin semnificativ bit este 1
        if( crc & 0x00000001 )
            crc = (crc >> 1) ^ polinom32;
        else
            crc = crc >> 1;
    }
}
return crc;
}

//implementare ce utilizează un tabel cu valori pre-calculate ale
funcției CRC
__flash unsigned long crc32tab[256] = {
0x00000000, 0x04c11db7, 0x09823b6e, 0x0d4326d9, 0x130476dc, 0x17c56b6b,
0x1a864db2, 0x1e475005, 0x2608edb8, 0x22c9f00f, 0x2f8ad6d6, 0x2b4bcb61,
0x350c9b64, 0x31cd86d3, 0x3c8ea00a, 0x384fbdbd, 0x4c11db70, 0x48d0c6c7,
0x4593e01e, 0x4152fd9, 0x5f15adac, 0x5bd4b01b, 0x569796c2, 0x52568b75,
0x6a1936c8, 0x6ed82b7f, 0x639b0da6, 0x675a1011, 0x791d4014, 0x7ddc5da3,
0x709f7b7a, 0x745e66cd, 0x9823b6e0, 0x9ce2ab57, 0x91a18d8e, 0x95609039,
0x8b27c03c, 0x8fe6dd8b, 0x82a5fb52, 0x8664e6e5, 0xbe2b5b58, 0xbaea46ef,
0xb7a96036, 0xb3687d81, 0xad2f2d84, 0xa9ee3033, 0xa4ad16ea, 0xa06c0b5d,
0xd4326d90, 0xd0f37027, 0xddb056fe, 0xd9714b49, 0xc7361b4c, 0xc3f706fb,
0xceb42022, 0xca753d95, 0xf23a8028, 0xf6fb9d9f, 0xfb8bb46, 0xff79a6f1,
0xe13ef6f4, 0xe5ffeb43, 0xe8bccd9a, 0xec7dd02d, 0x34867077, 0x30476dc0,
0x3d044b19, 0x39c556ae, 0x278206ab, 0x23431b1c, 0x2e003dc5, 0x2ac12072,
0x128e9dcf, 0x164f8078, 0x1b0ca6a1, 0x1fcdbb16, 0x018aeb13, 0x054bf6a4,
}

```

```

0x0808d07d, 0x0cc9cdca, 0x7897ab07, 0x7c56b6b0, 0x71159069, 0x75d48dde,
0x6b93dddb, 0x6f52c06c, 0x6211e6b5, 0x66d0fb02, 0x5e9f46bf, 0x5a5e5b08,
0x571d7dd1, 0x53dc6066, 0x4d9b3063, 0x495a2dd4, 0x44190b0d, 0x40d816ba,
0xacac5c697, 0xa864db20, 0xa527fdf9, 0xa1e6e04e, 0xbfa1b04b, 0xbb60adfc,
0xb6238b25, 0xb2e29692, 0x8aad2b2f, 0x8e6c3698, 0x832f1041, 0x87ee0df6,
0x99a95df3, 0xd684044, 0x902b669d, 0x94ea7b2a, 0xe0b41de7, 0xe4750050,
0xe9362689, 0xedf73b3e, 0xf3b06b3b, 0xf771768c, 0xfa325055, 0xfef34de2,
0xc6bcf05f, 0xc27dede8, 0xcf3ecb31, 0xcbfffd686, 0xd5b88683, 0xd1799b34,
0xdc3abded, 0xd8fba05a, 0x690ce0ee, 0x6dcfd59, 0x608edb80, 0x644fc637,
0x7a089632, 0x7ec98b85, 0x738aad5c, 0x774bb0eb, 0x4f040d56, 0x4bc510e1,
0x46863638, 0x42472b8f, 0x5c007b8a, 0x58c1663d, 0x558240e4, 0x51435d53,
0x251d3b9e, 0x21dc2629, 0x2c9f00f0, 0x285e1d47, 0x36194d42, 0x32d850f5,
0x3f9b762c, 0x3b5a6b9b, 0x0315d626, 0x07d4cb91, 0xa97ed48, 0xe56f0ff,
0x1011a0fa, 0x14d0bd4d, 0x19939b94, 0x1d528623, 0xf12f560e, 0xf5ee4bb9,
0xf8ad6d60, 0xfc6c70d7, 0xe22b20d2, 0xe6ea3d65, 0xeba91bbc, 0xef68060b,
0xd727bbb6, 0xd3e6a601, 0xdea580d8, 0xda649d6f, 0xc423cd6a, 0xc0e2d0dd,
0xcda1f604, 0xc960ebb3, 0xbd3e8d7e, 0xb9ff90c9, 0xb4bcb610, 0xb07daba7,
0xae3afba2, 0xaafbe615, 0xa7b8c0cc, 0xa379dd7b, 0xb3660c6, 0x9ff77d71,
0x92b45ba8, 0x9675461f, 0x8832161a, 0x8cf30bad, 0x81b02d74, 0x857130c3,
0x5d8a9099, 0x594b8d2e, 0x5408abf7, 0x50c9b640, 0x4e8ee645, 0x4a4ffb2,
0x470cdd2b, 0x43cdc09c, 0x7b827d21, 0x7f436096, 0x7200464f, 0x76c15bf8,
0x68860bfd, 0x6c47164a, 0x61043093, 0x65c52d24, 0x119b4be9, 0x155a565e,
0x18197087, 0x1cd86d30, 0x029f3d35, 0x065e2082, 0xb1d065b, 0x0fdc1bec,
0x3793a651, 0x3352bbe6, 0x3e119d3f, 0x3ad08088, 0x2497d08d, 0x2056cd3a,
0x2d15ebe3, 0x29d4f654, 0xc5a92679, 0xc1683bce, 0xcc2b1d17, 0xc8ea00a0,
0xd6ad50a5, 0xd26c4d12, 0xdf2f6bcb, 0dbe767c, 0xe3a1cbc1, 0xe760d676,
0xea23f0af, 0xeee2ed18, 0xf0a5bd1d, 0xf464a0aa, 0xf9278673, 0xfde69bc4,
0x89b8fd09, 0xd79e0be, 0x803ac667, 0x84fbdbd0, 0x9abc8bd5, 0x9e7d9662,
0x933eb0bb, 0x97ffad0c, 0afb010b1, 0xab710d06, 0xa6322bdf, 0xa2f33668,
0xbcb4666d, 0xb8757bda, 0xb5365d03, 0xb1f740b4,
};

unsigned long crc32wtable(unsigned int init_val_32, unsigned int adr_start,
                           unsigned int len)
{
    unsigned int counter;
    unsigned long crc = init_val_32;
    for( counter = 0; counter < len; counter++)
        crc = (crc<<8) ^ crc32tab[((crc>>24) ^ *(__flash char
*)adr_start++) & 0xff];
    return crc;
}

```

7.7.3 COMPARAREA TIMPIILOR DE EXECUȚIE

CRC-16 lent:

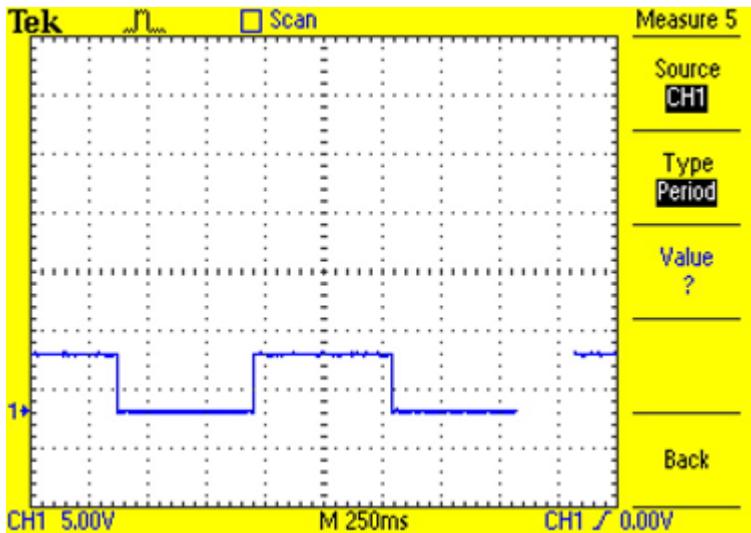


Figura 7.7

CRC-16 ce utilizează tabel cu valori pre-calulate:

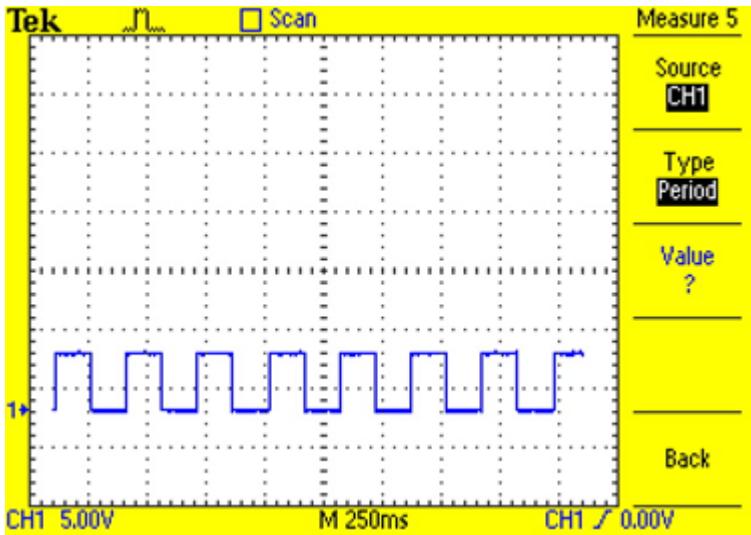


Figura 7.8

7.8 ANEXĂ

Cod sursă 7.3:

```
unsigned int reverse_bits_16(unsigned int input)
{
    unsigned int output = 0;
    unsigned int n = sizeof(input) << 3;
    unsigned int i = 0;

    for (i = 0; i < n; i++)
        if ((input >> i) & 0x1)
            output |= (0x1 << (n - 1 - i));
    return output;
}

unsigned long reverse_bits_32(unsigned long input)
{
    unsigned int left = input>>16;
    left = reverse_bits_16(left);
    unsigned int right = input;
    right = reverse_bits_16(right);
    unsigned long output = 0;
    output |= right;
    output <<= 16;
    output |= left;
    return output;
}
```

8 Frecvențmetru. Timer/Counter 0

8.1 FRECVENȚMETRUL

8.1.1 DEFINIȚIE

Un frecvențmetru este un instrument electronic sau o componentă dintr-un instrument electronic folosit pentru măsurarea frecvenței. Acesta măsoară numărul de evenimente (oscilații sau pulsuri) pe o anumită perioadă de timp a semnalului primit.

8.1.2 PRINCIPIU DE OPERARE

Frecvențmetrele folosesc un numărător pentru a acumula numărul de evenimente (pulsării) înregistrate într-o anumită perioadă de timp. După acea perioadă specificată valoarea din numărător este transferată pe display și numărătorul este resetat la 0. Există două moduri de măsurare a frecvenței: numărare directă și numărare reciprocă.

8.1.2.1 Numărarea directă

Frecvențmetrele care folosesc acest principiu acumulează numărul de dăți în care semnalul de input (intrare) traversează în creșterea sau în descreșterea lui o anumită valoare prestabilită.

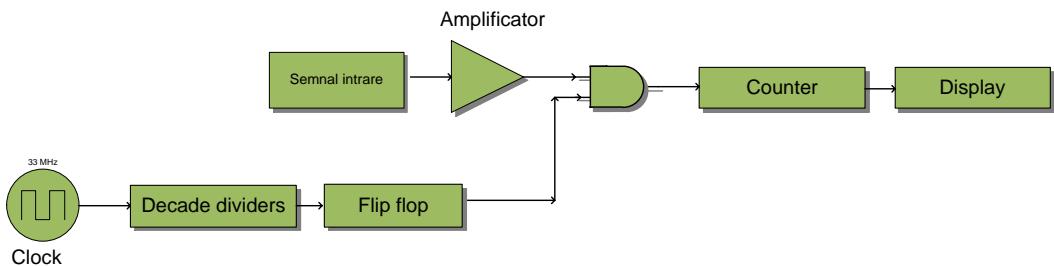


Figura 8.1 Diagrama bloc a unui numărător cu numărare directă

Când un semnal intră într-un frecvențmetru prima dată intră într-un amplificator unde este convertit în semnal rectangular pentru a putea fi procesat de restul circuitului. De obicei, în această fază a procesării este întâlnit un circuit de tip trigger Schmitt pentru a putea atenua zgomotele semnalului de intrare.

Pentru a putea crea semnalele din numărător este necesar un clock. De obicei acesta este un oscillator cu cristal care poate fi intern sau extern. Semnalul preluat de la oscillator se împarte cu ajutorul unor decade dividers trecând apoi prin flip flop pentru a obține pulsul pentru poarta principală.

Poarta principală primește semnalul de la flip flop și de la input și are ca output un număr de pulsuri pe o perioadă precisă de timp. De exemplu, dacă semnalul de input este de 1 MHz și poarta a fost deschisă pentru o secundă aceasta va avea ca output 1 milion de pulsuri.

Numărătorul preia pulsațiile de la poarta principală și le împarte la 10 în multiple faze, numărul de faze fiind egal cu numărul de cifre afișate minus 1. De exemplu, în prima fază împarte la 10, în faza a două împarte la 10×10 , și.a.m.d.

Latch-ul este folosit pentru a memora ultimul rezultat cât timp counter-ul primește alt input pentru a putea păstra un rezultat static pe display. Display-ul preia output-ul de la latch și îl afișează.

În acest caz :

$$\text{frecvența} = \frac{\text{numărul de pulsuri}}{\text{unitatea de timp}} = \frac{\text{valoarea dată de numărător}}{\text{unitatea de timp}}$$

8.1.2.2 Numărarea reciprocă

O altă metodă de a măsura frecvența unui semnal este de a măsura perioada unui ciclu din forma de undă și de a calcula reciproca acesteia. De exemplu, numărătorul începe să calculeze la primul front pozitiv al semnalului de input și se oprește la următorul front pozitiv.

În acest caz :

$$\text{frecvența} = \frac{1}{\text{perioadă}}$$

8.2 TIMER/COUNTER 0

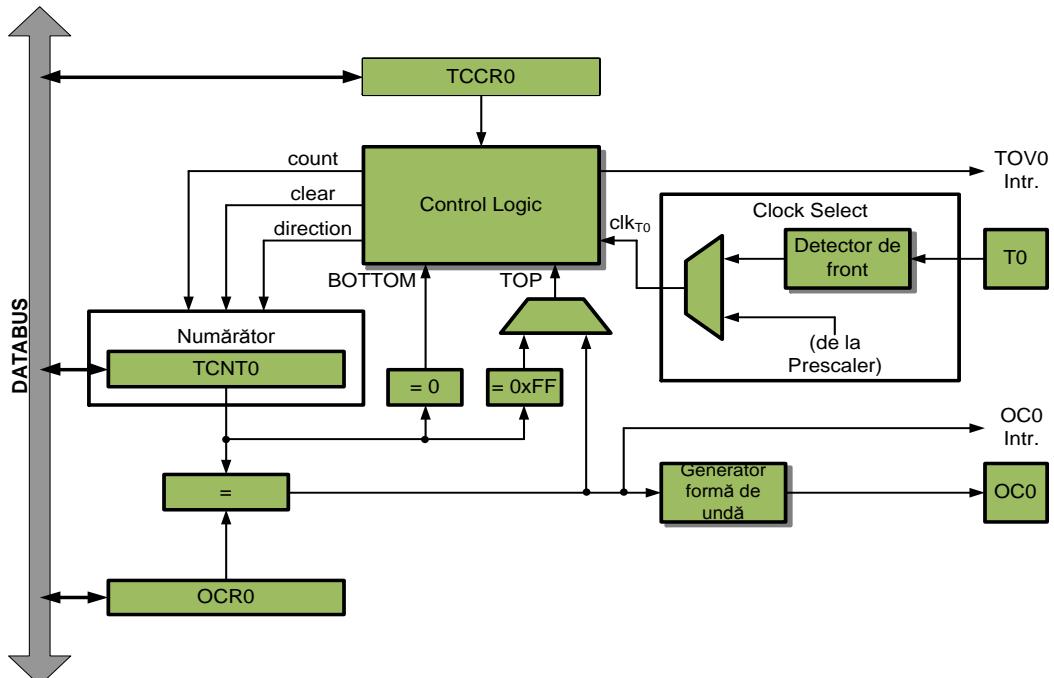


Figura 8.2 Diagrama bloc Timer/Counter0

8.2.1 NOȚIUNI INTRODUCTIVE

Atmega16 dispune de trei timere dintre care două sunt pe 8 biți (Timer/Counter0 , Timer/Counter2) și unul este pe 16 biți (Timer/Counter1) .

Sursa semnalului de clock poate fi selectată intern (de la Prescaler), sau de la o sursă de clock externă conectată la pinul T0. Timer -ul este inactiv când nu este selectată nici o sursă de clock. Ieșirea din blocul *Clock Select* reprezintă semnalul de clock la care va opera Timer/Counter 0 (clk_{T0}).

8.2.1.1 Regiștri

În Error! Reference source not found.2 putem observa următorii registri :

- TCCR0 (Timer/Counter Control Register) : biții acestui registru de control configurează modul de operare al timerului;
- TCNT0 este un registru numărător;
- OCR0 (Output Compare Register) este comparat tot timpul cu valoarea lui **TCNT0**. Rezultatul comparării poate fi folosit de generatorul de undă pentru a genera la ieșire un semnal PWM sau semnal de clock cu frecvențe variabile pe pinul Output Compare (**OC0**). Rezultatul comparării va seta, de asemenea, și *Compare Flag* (**OCF0**), care poate fi folosit pentru a genera o cerere de întrerupere la ieșire.

8.2.1.2 Unitatea de numărare

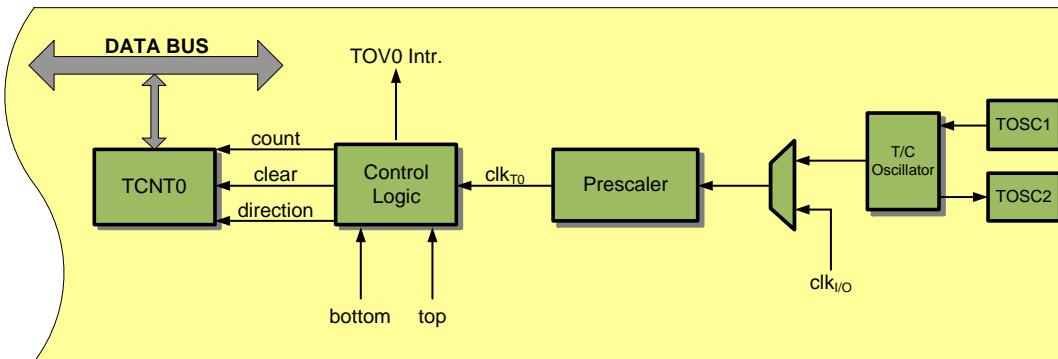


Figura 8.3 Diagrama bloc a unității de numărare

Unitatea de numărare este pe 8 biți, bidirecțională și programabilă. În

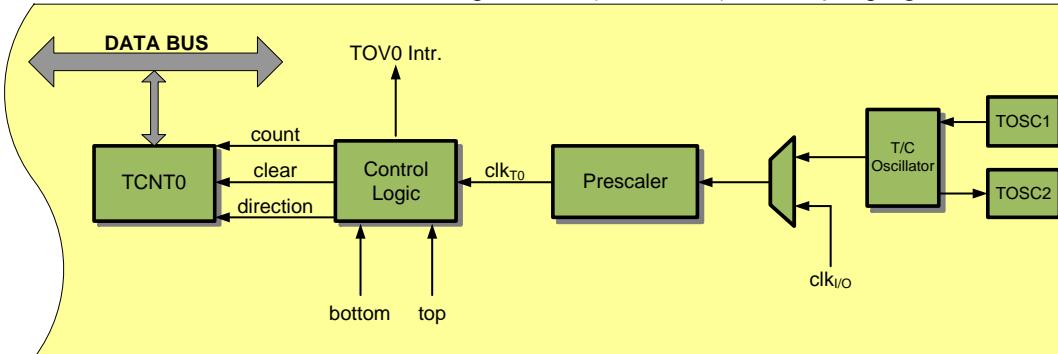


Figura 8.3 se poate vedea structura acestei unități.

Semnificația semnalelor :

count	incrementarea sau decrementarea registrului TCNT0 cu 1 unitate;
direction	selecția între incrementare și decrementare;
clear	resetare registru TCNT0;
clk_{T0}	clock-ul Timer/Counter;
top	TCNT0 a ajuns la valoarea sa maximă;
bottom	TCNT0 a ajuns la valoarea minimă.

8.2.1.3 Unitatea de comparare (Output Compare)

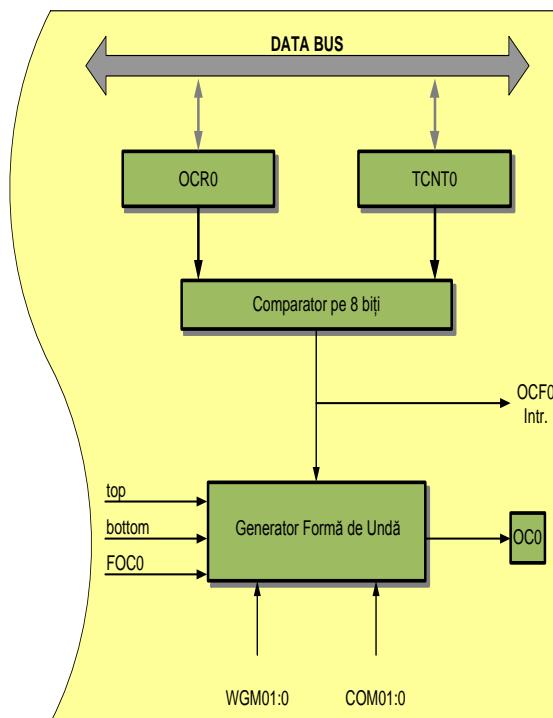


Figura 8.4 : Diagrama bloc a unității de comparare

Această unitate compară TCNT0 cu registrul Output Compare (OCR0). Dacă TCNT0 este egal cu OCR0 se va seta flag-ul Output Compare (OCF0) la următorul ciclu de clock. Acest flag va genera o întrerupere de output compare dacă această întrerupere este activată. Flag-ul Output Compare (**OCF0**) poate fi resetat prin soft scriind la locația sa un 1 logic. Generatorul semnalului de ieșire (Wave Generation Mode) folosește semnalul de ieșire al comparatorului pentru a genera forma de undă corespunzătoare cu modul de operare setat prin biții **WGM01:00** și biții Compare Output Mode (**COM01:00**).

8.2.2 Moduri de funcționare

Definiții:

- BOTTOM : reprezintă valoarea minimă a counterului 0x00
- MAX : reprezintă valoarea maximă a counterului 0xFF (aceasta diferă în funcție de numărul de biți al counter-ului)
- TOP : counterul ajunge la valoarea de TOP când devine egal cu cea mai mare valoare din secvența de numărat. Aceasta poate fi o valoare fixă (MAX) sau valoarea stocată în registrul OCR în funcție de modul de operare.

Un timer poate funcționa în următoarele moduri:

- **Normal** – se numără periodic de la BOTTOM la MAX

- Clear Timer on Compare (CTC) – se numără periodic de la BOTTOM la TOP
- **Fast PWM** – se numără periodic de la BOTTOM la TOP cu posibilitatea de a modula în durată ieșirea de la pinul OCn (sau OCnx). De exemplu :
 - atunci când valoarea din registrul numărător devine egală cu BOTTOM, pinul OCn este resetat;
 - atunci când valoarea din registrul numărător devine egală cu valoarea din registrul de comparare pentru ieșire, pinul OCn este setat.

8.3 PROBLEMĂ REZOLVATĂ

Enunț

Generarea unui semnal PWM de 50Hz având un factor de umplere de 45 %.

Notări

F = frecvență

T = perioada

DC = Duty cycle = factorul de umplere

Rezolvare

$$F = 50\text{Hz}$$

$$T = \frac{1}{F} = \frac{1}{50} = 0.02\text{ s} = 20\text{ ms}$$

$$T = T(\text{on}) + T(\text{off})$$

$$\text{DC} = 45\%$$

$$T_{\text{on}} = \frac{45}{100} 20 = 9\text{ ms}$$

$$T(\text{off}) = 20\text{ ms} - 9\text{ ms} = 11\text{ ms}$$

Configurare Timer/Counter 0

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TCCR0							
Valoare inițială	0	0	0	0	0	0	0	0	

Figura 8.5 : Timer/Counter Control Register – TCCR0

Din această figură bițiile de interes sunt :

Biți 6,3: WGM01:0 - Waveform Generation Mode. Aceștia pot fi setați pe 0 sau pe 1, în funcție de tipul de pwm pe care dorim să îl obținem , conform **Tabel 8.1 Waveform Generation Mode Bit Description**

Mod	WGM01 (CTC0)	WGM00 (PWM0)	Mod de operare	TOP	Actualizare OCR0	Setare Flag TOV0
0	0	0	Normal	0xFF	Imediat	MAX
1	0	1	PWM, <i>Phase Correct</i>	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Imediat	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Tabel 8.1 Waveform Generation Mode Bit Description

=> **Biți** COM01:0 . Cu ajutorul lor putem controla dacă PWM poate fi inversabil sau nu.

COM01	COM00	Descriere
0	0	Mod normal de operare, OC0 deconectat
0	1	Toggle OC0 la <i>compare match</i>
1	0	Resetare OC0 la <i>compare match</i>
1	1	Setare OC0 la <i>compare match</i>

Tabel 8.2

Codul sursă :

```
#include <avr.h>
#include <iom16.h>
void timer0_init()
{
    // inițializare TCCR0
    TCCR0 |= (1<<WGM00) | (1<<COM01) | (1<<WGM01) | (1<<CS00);
    // setare pinul OC0 ca output (este pinul PB3)
    DDRB |= (1<<PB3);
}
void main()
{
    uint8_t duty;
    duty = 115; // duty cycle = 45% și deoarece Timer 0 este pe
                // 8 biți acesta poate număra până la 255. Deci
                // 45% din 255 = 114.75 = 115
    timer0_init(); // inițializarea timer-ului
    while(1) // crearea unei bucle infinite
    {
        OCR0 = duty; //Output Compare register
    }
}
```

Rezultate

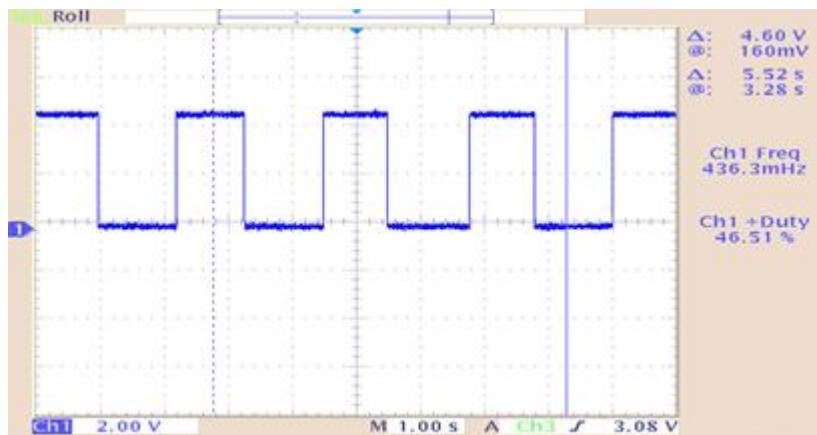


Figura 8.6 Captură de imagine osciloscop

8.4 PROBLEMĂ PROPUȘĂ

Folosind Timer 0 cu modul fast pwm să se folosească semnalele pentru a varia luminozitatea unui LED de la minim la maxim și apoi de la maxim la minim . Indicație: pentru a putea vedea schimbarea în luminozitatea LED-ului se va folosi funcția `__delay_cycles()` .

```
#include <inaavr.h>
#include <iom16.h>

void Init()
{
    /*
    Timer Clock = CPU Clock (fără prescalare)
    Mode        = Fast PWM
    */

    TCCR0 |= (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS00);

    //Setare pin OC0 ca output
    DDRB |= (1<<PB3);
}
```

```
void SetOutput(unsigned int duty)
{
    OCR0=duty; //Output Compare register
}

void Wait() //este necesară o funcție de așteptare
{
    __delay_cycles(3200);
}

void main()
{
    unsigned int brightness=0;

    Init();
    while(1)
    {

        for(brightness=0;brightness<255;brightness++)
        {

            SetOutput(brightness);
            Wait();
        }

        for(brightness=255;brightness>0;brightness--)
        {

            SetOutput(brightness);
            Wait();
        }
    }
}
```

9 Watchdog. Calculul timpului, frecvența oscilatorului.

9.1 NOȚIUNI INTRODUCTIVE. FAMILIA AVR XMEGA.

Ideea de bază ce stă în spatele existenței unui *Watchdog timer*, este de a avea un mecanism care să reseteze microcontroler-ul în cazul în care dintr-un motiv exceptional, aplicația nu răspunde un timp îndelungat.

Familia *AVR XMEGA* oferă un *watchdog* intern foarte robust cu o sursă de clock separată față de microcontroler. O eroare a clock-ului principal nu afectează *watchdog – ul*.

Clarificarea unor termeni folosiți în acest laborator:

- *Watchdog timer (WDT)* –modul periferic care poate fi configurat să genereze un semnal de reset , dacă este resetat prea devreme sau prea târziu potrivit unei perioade specificate.
- *Watchdog timer reset WDT reset* –resetarea registrului *WDT*(revenirea la o valoare inițială stabilită).
- *Resetarea sistemului* –resetarea microcontroler-ului *AVR*.

9.1.1 SCHEMA DE PRINCIPIU

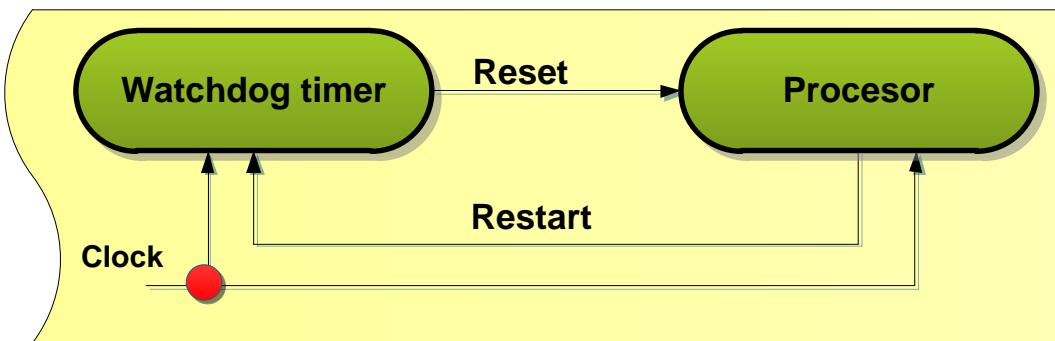


Figura 9.1 Schema de principiu

9.1.2 SURSA DE CLOCK

Watchdog timer – ul din familia AVR XMEGA are o sursă de clock internă de 1kHz separată, reprezentată de un oscilator RC cu un consum redus. Acest oscilator poate fi folosit și de timer-ul *RTC* sau *Brown Out Detection circuit* dacă este configurat în modul eșantionare. Dacă unul din aceste module este configurat să folosească acest oscilator, atunci este pus în funcțiune.

Oscilatorul RC nu este foarte precis. Acest lucru se datorează faptului că este proiectat pentru un consum redus de energie pentru a putea fi folosit în aplicații pe o perioadă mai mare de timp. Dezavantajul unui consum redus este precizia redusă. Precizia tipică este de $\pm 30\%$. De aici rezultă că frecvența oscilatorului poate varia de la un dispozitiv la altul. Atunci când se proiectează aplicații care folosesc *WDT*-ul, variația dispozitiv-la-dispozitiv trebuie păstrată pentru a se asigura că perioadele de *time – out* folosite sunt valabile pentru toate dispozitivele, nu numai pentru dispozitivele din laboratorul de dezvoltare a aplicațiilor. În plus sursa de clock este sensibilă și la variația temperaturii sau a tensiunii de alimentare – deși variația este semnificativ mai mică decât $\pm 30\%$.

9.1.3 SINCRONIZAREA ÎNTRU DOMENII DIFERITE DE CLOCK

WDT și *UCP* operează în domenii de clock diferite, iar sincronizarea între aceste domenii trebuie luată în considerare atunci când folosim *watchdog*-ul. Pentru a configura *WDT* – *ul* sunt necesare 2-3 perioade de clock ale *WDT* – *ului*. Setările sunt scrise în registrele de control ale *WDT* – *ului*, ele sunt efective la următorul front pozitiv al clock-ului *watchdog* – *ului*, adică după 2 – 3 ms după ce setările sunt scrise în registrul. De aici rezultă că perioada inițială de *time – out* este cu 3 ms mai lungă. Dacă perioada de *time – out* este de 8 ms, perioada actuală este între 10 și 11 ms. Acest lucru este relevant atunci când se folosește *modul fereastră* și perioade de *time – out* mici. Această caracteristică nu este specifică doar *watchdog* – *ului*, toate timer-ele asincrone operează în acest fel pentru sincronizarea clock-ului între diferite domenii. *WDT* – *ul* este resetat atunci când are loc o scriere validă în registrele de control.

O altă caracteristică de sincronizare este diferența de timp dintre executarea instrucțiunii de *WDT reset* și resetarea acestuia efectiv, problemă ce ține de sincronizare dintre domenii de clock. *WDT* este resetat după 3 perioade de clock după ce instrucțiunea de reset este executată, adică între 2 – 3 ms după executarea instrucțiunii. Dacă se folosește o perioadă de *time – out* de 8 ms prima instrucțiune de *WDT reset* va fi executată după 5 ms de la activarea *watchdog* – *ului*. Înțând cont de precizia oscilatorului $\pm 30\%$, instrucțiunea trebuie să fie executată în 3.5 ms sau mai puțin. Intervalul dintre două instrucțiuni de *WDT reset* poate fi de 4.9 ms sau mai mic (8 ms-1 ms incertitudine-30% precizia oscilatorului). Efectul acestei sincronizări este minimizat atunci când perioada de *time – out* este mai mare.

Dacă *WDT* generează un semnal de reset (de ex. după expirarea perioadei de *time – out*) resetarea sistemului are loc la următorul front al clock-ului *watchdog* – *ului*. Resetarea sistemului are loc la 1 ms după ce perioada de *time – out* a expirat. Acest lucru în mod normal nu ar trebui să creeze nici o problemă, dar este important de știut dacă se dorește să se măsoare perioada *watchdog*-ului urmărind nivelului logic de tensiune al unui pin. Cea mai bună metodă de calcul a perioadei efective a *watchdog*-ului este folosirea XMEGA Real Time Clock timer-ului. Toate aceste caracteristici sunt valabile în ambele moduri, normal și fereastră.

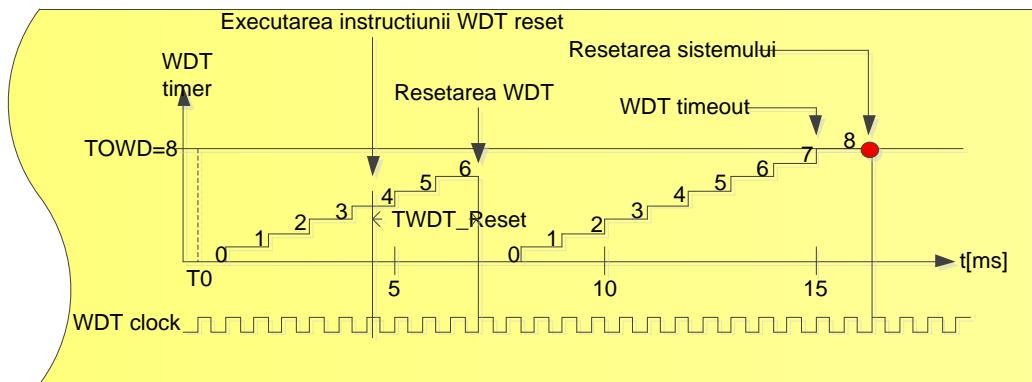


Figura 9.2 Sincronizarea clock-ului

9.1.4 MODURILE DE FUNCȚIONARE

Watchdog timer – ul XMEGA pe lângă modul normal de lucru în care trebuie resetat înainte de o perioadă stabilită, oferă și modul fereastră în care *WDT – ul* poate fi resetat doar într-o anumită perioadă de timp (*fereastră*). În modul fereastră dacă *watchdog – ul* este resetat prea devreme sau prea târziu se generează o condiție de resetare a microcontroler-ului.

9.1.4.1 Modul normal

Atunci când *WDT – ul* este configurat în modul normal este setată o singură perioadă de *time – out*. Dacă *watchdog – ul* nu este resetat înainte de expirarea acestei perioade, va genera o condiție de reset. Intervalul marcat „deschis” pe grafic indică faptul că *WDT – ul* poate fi resetat oricând înainte de expirarea perioadei de *time – out*. (T_{WDT}). Modul de funcționare normal este ilustrat în **Figura 9.3**.

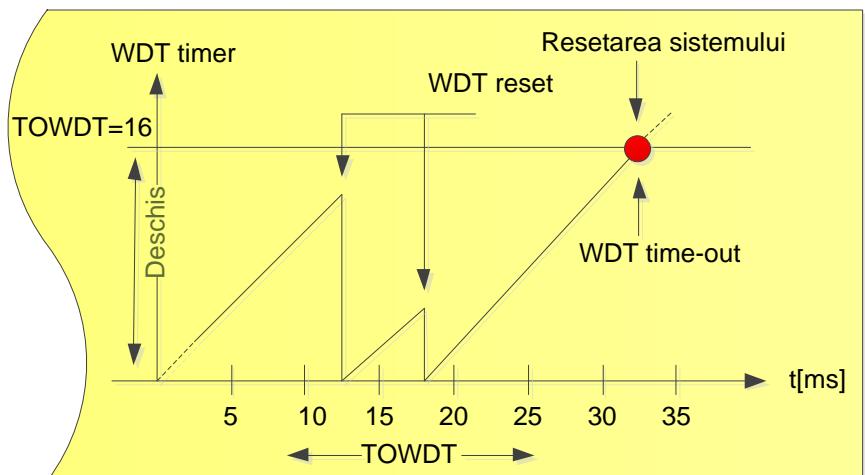


Figura 9.3 Modul Normal

9.1.4.2 Modul fereastră

Atunci când WDT-ul este folosit în modul fereastră sunt setate două perioade de $time - out$, o perioadă „închisă” (T_{WDTW}) și o perioadă „deschisă” (T_{WDT}). Prima perioadă (T_{WDTW}) reprezintă un interval de la 8ms până la 8s în care watchdog – ul nu poate fi resetat, dacă este resetat atunci va genera o condiție de reset. A doua perioadă (T_{WDT}) reprezintă un interval de la 8ms până la 8s în care WDT – ul poate fi resetat. Perioada totală de $time - out$ este calculată ca fiind suma celor două perioade menționate mai sus. Modul de funcționare fereastră este ilustrat în **Figura 9.4**.

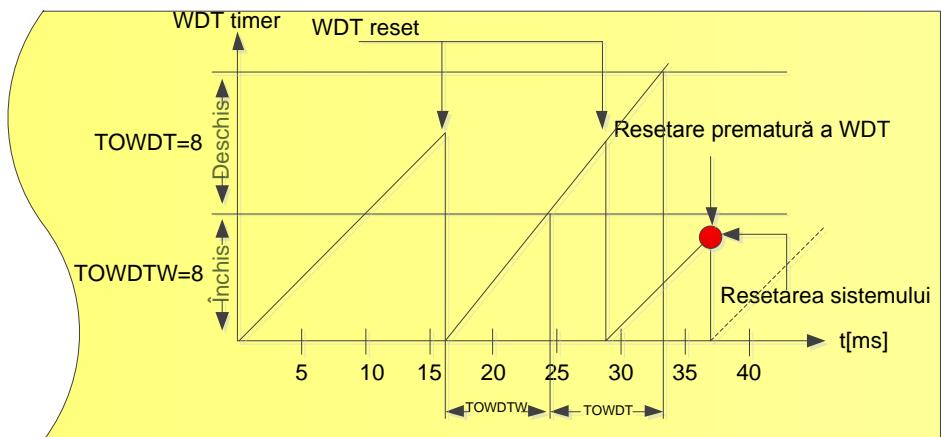


Figura 9.4 Modul fereastră

9.2 STANDARDUL INTERNAȚIONAL IEC 60730

IEC 60730 este un standard de siguranță pentru aparatele de uz casnic, care abordează aspectele de design și de funcționare a produselor. Acest standard este menționat și de alte standarde care vizează siguranța dispozitivelor, de exemplu standardul IEC 60335. Pentru siguranță este foarte important ca sistemul să fie certificat de acest standard.

9.2.1 CLASELE STANDARDULUI IEC 60730

Anexa H a standardului IEC 60730 definește trei clase de control al soft-ului pentru diferite aparate electrocasnice :

- *Clasa A* – funcții de control care nu sunt destinate a fi invocate pentru siguranța echipamentelor;
- *Clasa B* – aplicații care includ cod cu scopul de a preveni alte erori decât cele software;
- *Clasa C* – aplicații care includ cod destinat să prevină erorile fără utilizarea altor dispozitive de protecție.

9.2.2 WATCHDOG-UL DE CLASĂ B

Arhitectura XMEGA este prevăzută cu un mecanism de protecție care asigură faptul că setările WDT nu pot fi modificate accidental. Pentru o siguranță sporită este prevăzută o siguranță (fuse) pentru a bloca setările WDT.

Deoarece WDT este un element de siguranță integrat în familia Atmel AVR XMEGA s-a conceput o rutină de autodiagnosticare care testează ambele moduri de funcționare, normal și în fereastră. Aceasta se execută după resetare, în partea de pre-initializare a aplicației înaintea funcției main. Diagrama de test este reprezentată în *Figura 9.5* Diagrama de test

Rutina de autodiagnosticare ne asigură că:

- Resetarea sistemului este realizată după expirarea perioadei de time-out a *watchdog – ului*;
- *Watchdog timer – ul* poate fi resetat;
- Sistemul este resetat la resetarea prematură a *watchdog – ului* în modul de funcționare fereastră.

Conform diagramei logice dispozitivul este resetat de câteva ori în timpul testului. Prin urmare variabila SRAM și flag-urile de reset ale dispozitivului sunt utilizate de rutina de autodiagnosticare, pentru a urmări fază de testare. În continuare utilizatorul poate configura ce să facă în cazul unui reset software, brown-out sau cum să proceseze resetul cauzat de watchdog, atunci când testul se află în starea *WDT_OK*.

Rutina de autodiagnosticare folosește un *Real Time Counter RTC* pentru a verifica perioada *Watchdog – ului*. *RTC – ul* are o sursă de clock independentă față de *CPU* și *WDT*. Ambele module au oscilatoare care funcționează la 32.768kHz . Cu toate acestea oscilatorul *WDT – ului* este optimizat pentru un consum redus de energie. *RTC – ul* este folosit pentru estimarea perioadei *WDT – ului*, iar programul verifică dacă această perioadă este în intervalul $(T/2, 3T/2)$, unde T este perioada nominală a *WDT – ului*.

Fluxul de execuție fără erori este :

1. După punerea sub tensiune sau reset extern se verifică dacă *WDT – ul* poate reseta sistemul. Se setează starea *WDT_1*. Sistemul este resetat de *WDT*.
2. Se verifică dacă *WDT – ul* poate fi resetat. Se setează starea *WDT_2*. Sistemul este resetat de *WDT*.
3. Se verifică dacă modul window funcționează corect. Se setează *WDT_3*. Sistemul este resetat de *WDT*.
4. Se configurează *WDT – ul* conform setărilor. Se setează starea testului *WDT_OK*. Se continuă cu funcția main.

Primul pas este asigurarea faptului că *WDT – ul* poate genera semnalul de reset. Acest lucru se realizează configurând *Watchdog – ul* conform datasheet-ului și așteptând până când procesul de resetare a avut loc. În plus *RTC – ul*¹ este folosit pentru a estima perioada watchdog-ului, care este necesară în fazele ulterioare ale testului. Acest lucru se realizează setând perioada *RTC – ului* cu o valoare destul de mică (aproximativ $3ms$) și numărând perioadele *RTC – ului* până la resetare. Există un timp maxim de așteptare care poate fi configurat, după expirarea acestui timp programul intră într-o stare de eroare.

¹ *RTC – ul* este testat implicit de această rutină de autodiagnosticare, în cazul în care diferența de frecvență dintre *RTC* și *WDT* este mai mare de 50% se trece în starea de eroare.

Al doilea pas este asigurarea faptului că *WDT – ul* poate fi resetat și verificarea perioadei *watchdog – lui*. Starea de eroare este setată temporar, iar apoi se verifică dacă perioada *WDT – lui* este mai mare decât un minim stabilit. Se verifică dacă diferența de frecvență între *WDT* și *RTC* este situată într-un interval care ne asigură că ambele module funcționează după așteptări. Apoi *WDT – ul* este configurat, se folosește *RTC* pentru a aștepta $\frac{3}{4}$ din perioada *watchdog – lui*, care a fost estimată la pasul anterior, astfel se verifică dacă perioada *WDT – lui* nu expiră mai devreme decât se așteaptă. După aceea *WDT – ul* este resetat, iar programul așteaptă din nou $\frac{3}{4}$ din perioada *WDT – lui*. Dacă mecanismul de resetare a *watchdog – lui* a descurat fară probleme sistemul ar trebui să se reseteze în timp ce programul este în a doua așteptare. Acest lucru se datorează faptului că perioada totală de așteptare este 1.5 din perioada totală a *WDT – lui* estimată anterior. În plus această resetare prematură ar duce testul în starea de eroare. Presupunând că *WDT – ul* a fost resetat corect, sistemul trebuie să fie resetat în aproximativ $\frac{1}{4}$ din perioada *WDT – lui*. Testul trece în starea următoare, iar programul este în așteptarea resetului.

Al treilea pas este asigurarea faptului că *WDT – ul* funcționează corect în modul fereastră. Acest lucru are la bază setarea *WDT – lui* și trecerea testului în următoarea stare și apoi resetarea lui prematură. Având în vedere că perioada nu este respectată *WDT – ul* ar trebui să genereze un semnal de reset. Astfel, după resetarea prematură a *WDT – lui* programul așteaptă resetarea sistemului $\frac{1}{4}$ din perioada totală a *watchdog – lui*. Dacă a apărut o problemă sistemul nu se va reseta, iar programul va semnaliza starea de eroare.

Al patrulea și ultimul pas: programul pur și simplu setează *WDT – ul* în modul fereastră, iar testul în starea *WDT_OK*. După acesta, aplicația este responsabilă de resetarea *WDT – lui* în funcție de setări. Dacă testul se află în starea de eroare va fi apelat un sistem de tratare a erorilor predefinit de utilizator. În mod implicit dispozitivul va fi “suspendat” deoarece un *WDT* funcțional este esențial pentru o aplicație software sigură.

RTC – ul și variabilele de stare sunt declarate în aşa fel încât compilatorul nu le inițializează după resetare. Acest lucru permite folosirea lor pe durata tuturor resetărilor.

9.2.3 TESTAREA RUTINEI DE AUTODIAGNOSTICARE

În primul rând rutina de autodiagnosticare va seta starea de eroare dacă nu are loc resetarea sistemului. O posibilă problemă care poate împiedica *WDT – ul* să genereze un semnal de reset ar putea fi dezactivarea lui. Starea de eroare va fi stabilită, iar sistemul pur și simplu va fi “suspendat”.

O eroare a frecvenței *RTC – lui* sau a *WDT – lui* poate fi simulată printr-un break point și modificând variabila *rtc_count*, astfel încât să fie în afara intervalului prestabilit.

O eroare în mecanismul de resetare a *WDT – lui* poate fi simulată prin înlăturarea liniei de cod în care *WDT – ul este resetat*. Fiind dată structura programului, în al doilea rând rutina de autodiagnosticare va seta starea de eroare înainte ca *watchdog – ul* să urmeze constrângerile de timp care sunt impuse.

O eroare în modul fereastră al *watchdog*-ului poate fi simulată prin înlăturarea setărilor acestui mod. Codul va reseta *WDT – ul* și apoi va aștepta $\frac{1}{4}$ din perioada de time-out. La momentul respectiv perioada *watchdog*-ului va fi mai mare sau egală cu perioada estimată (perioada totală este suma perioadelor “închisă” și “deschisă”). Prin urmare sistemul nu va fi resetat înainte de setarea stării de eroare.

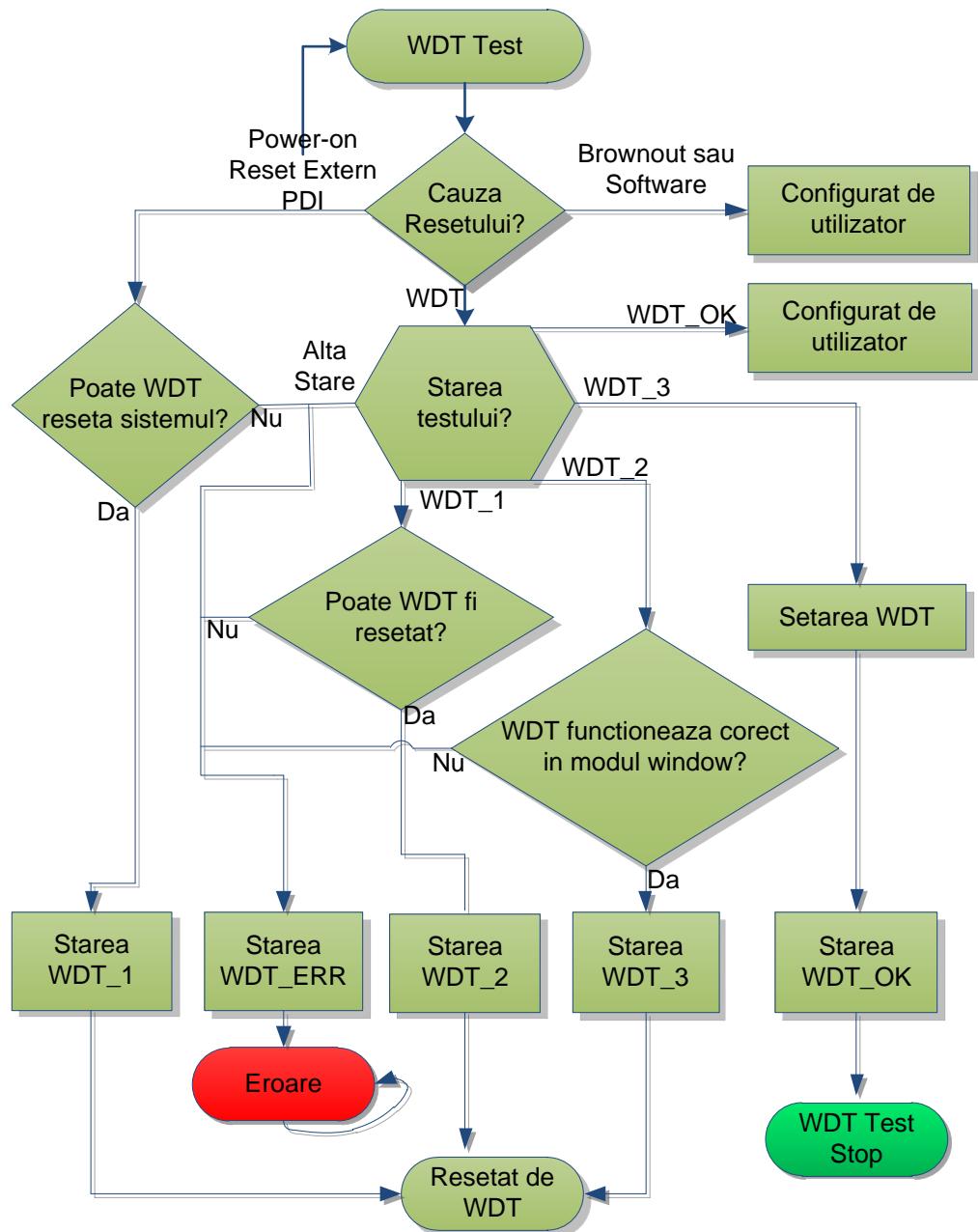


Figura 9.5 Diagrama de test

9.3 ATMEGA16

9.3.1 ÎNTRERUPERA DE RESET

Întotdeauna când are loc o condiție de *Reset*, microcontroler-ul va aștepta un număr de cicli (o durată de timp (t_{TOUT})), după care se va executa instrucțiunea aflată în locația de memorie corespunzătoare vectorului de întrerupere *RESET* (0x000000), care este de regulă un salt (JMP) la o rutină de întrerupere adecvată.

9.3.2 RUTINA DE ÎNTRERUPERE DE RESET

În rutina de întrerupere de Reset se fac anumite inițializări necesare aplicației. De regulă, pentru un proiect scris în limbajul C, compilatorul de la IAR generează automat o rutină de întrerupere de Reset în care inițializează vârful stivei (implicit și dimensiunea ei). Scrierea unei rutine de Reset proprii nu este posibilă. De fapt, compilatorul de la IAR construiește 2 stive:

- *stiva de date* – segmentul CSRACK;
- *stiva de program* – segmentul RSTACK (return address stack). Compilatorul utilizează acest spațiu pentru adresele de revenire salvate de instrucțiunile de apel de procedură și la tratarea întreruperilor.

Întotdeauna înainte de funcția main, va fi executată rutina de întrerupere de Reset (de fapt main-ul este apelat din rutina de Reset).

9.3.3 SURSELE DE RESETARE ALE MICROCONTROLER-ULUI

Există 5 surse de resetare a microcontroler-ului:

- *Power – on Reset* – atât timp cât tensiunea de alimentare V_{cc} este sub valoarea prag V_{POT} microcontroler-ul nu funcționează. În momentul când V_{cc} crește peste V_{POT} microcontroler-ul va genera o condiție de *Reset*;
- *External Reset* – menținerea pinului \overline{RESET} pe 0 logic pentru o durată de timp mai mare decât $t_{RST} = 1.5\mu s$ va rezulta în resetarea microcontroler-ului;
- *Watchdog Reset* – expirarea numărătorului *Watchdog* – ului, dacă este activat va genera o condiție de reset;
- *Brown – out Reset* – coborârea tensiunii de alimentare sub pragul V_{BOT} va avea ca efect resetarea microcontroler-ului, dacă unitatea *Brown – out Detector* este activată;
- *JTAG Reset* – interfața JTAG poate să reseteze microcontroler-ul.

9.3.3.1 Watchdog Reset

Atunci când perioada watchdog-ului expiră se generează un impuls de reset scurt pe durata unui ciclu de clock. Pe frontul negativ al acestui impuls *Delay Counter – ul* va începe a contoriza perioada de time-out (t_{TOUT}).

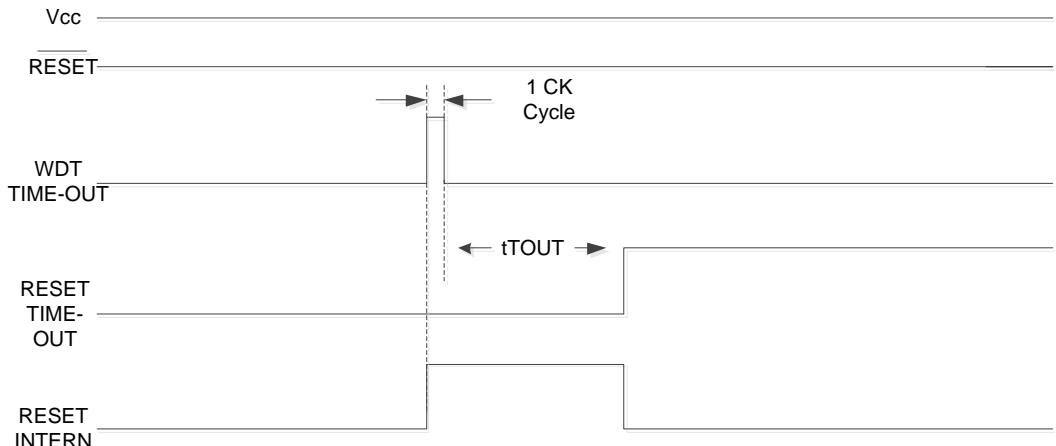


Figura 9.6 Watchdog reset

9.3.4 REGISTRUL MCUCSR

Acest registru oferă informații despre cauza resetării microcontroler-ului.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Valoare inițială

0	0	0
---	---	---

JTRF: JTAG Reset Flag – acest bit este setat dacă resetarea sistemului este cauzată de scrierea de 1 logic în registrul **JTAG Reset** selectat de instrucțiunea JTAG, **AVR_RESET**. Acest bit este resetat printr-un power-on reset sau prin scrierea de “0” logic.

- **WDRF:** Watchdog Reset Flag - acest bit este setat dacă resetarea sistemului este cauzată de *watchdog*. Acest bit este resetat printr-un power-on reset sau prin scrierea de “0” logic.
- **BORF:** Brown-out Reset Flag - acest bit este setat dacă se produce un brown-out reset. Bitul este resetat prin power-on reset sau prin scrierea de “0” logic.
- **EXTRF:** External Reset Flag – acest bit este setat dacă se produce un reset extern. Bitul este resetat prin power-on reset sau prin scrierea de “0” logic.
- **PORF:** Power-on Reset Flag – Acest bit este setat dacă se produce un power-on reset. Bitul este resetat numai prin scrierea de “0” logic.

Pentru a folosi acești biți ei trebuie să citiți și resetați cât mai devreme în program.

9.3.5 WATCHDOG TIMER

Watchdog timer de pe microcontroler-ul ATmega16, Figura 9.7, are o sursă de clock internă proprie de 1 MHz . Aceasta este o valoare tipică pentru $V_{cc} = 5V$. Perioada de time-out a watchdog-ului poate fi ajustată conform tabelului 9.1.

Tabelul 9.1 Modurile de configurare a perioadei de time-out. Dacă perioada de time-out expiră fără ca watchdog-ul să fie resetat atunci ATmega16 va fi resetat.

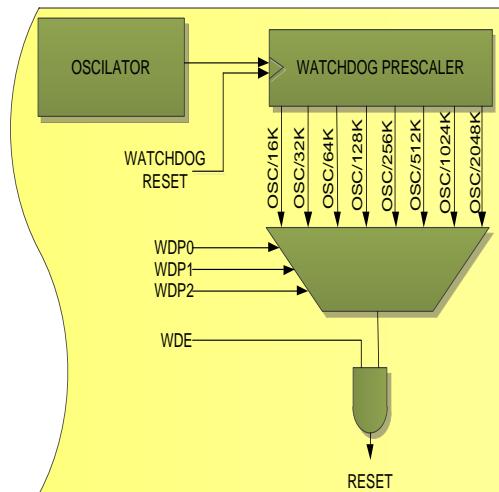


Figura 9.7 Watchdog timer

9.3.5.1 Registrul WDTCR

Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	WDTCR
Valoare inițială	0	0	0	0	0	0	0	0	

- **WDTOE: Watchdog Turn-off Enable** – este un bit pentru siguranță – în ideea de a nu dezactiva accidental Watchdog timer-ul, această procedură a fost făcută mai complicată – pentru a dezactiva Watchdog timer-ul este nevoie să fie setat bitul WDTOE și apoi, cât mai repede (cel mult în 4 cicli) trebuie resetat bitul WDE
- **WDE: Watchdog Enable** – setarea acestui bit echivalează cu activarea Watchdog timer-ului; resetarea acestui bit va determina dezactivarea Watchdog timer-ului, dacă bitul WDTOE are în acest timp valoarea 1
- **WDP2:0 : Watchdog Timer Prescaler** – acești biți determină practic timpul de expirare a Watchdog timer-ului, conform tabelului următor:

WDP2	WDP1	WDP0	Numărul de cicli ai WDT	Timpul de expirare pentru VCC = 3.0V	Timpul de expirare pentru VCC = 5.0V
0	0	0	16K(16,384)	17.1 ms	16.3 ms
0	0	1	32K(32,768)	34.3 ms	32.5 ms
0	1	0	64K(65536)	68.5 ms	65 ms
0	1	1	128K(131,072)	0.14 s	0.13 s
1	0	0	256K(262,144)	0.27 s	0.26 s
1	0	1	512K(524,288)	0.55 s	0.52 s
1	1	0	1024K(1,048,576)	1.1 s	1.0 s
1	1	1	2048K(2,097,152)	2.2 s	2.1 s

Tabelul 9.1 Modurile de configurare a perioadei de time-out.

9.4 APLICAȚII

9.4.1 APLICAȚIA 1

9.1.1.1 Enunț

Să se creeze o aplicație care să demonstreze resetarea microprocesorului de către *Watchdog timer*.

9.1.1.2 Rezolvare

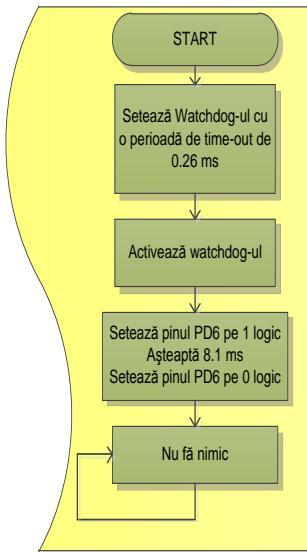
În primul rând se va activa *Watchdog timer*-ul (attenție și la timpul de expirare).

În al doilea rând se va “bloca” microprocesorul, de exemplu într-o buclă infinită, lăsând *Watchdog timer*-ul să expire.

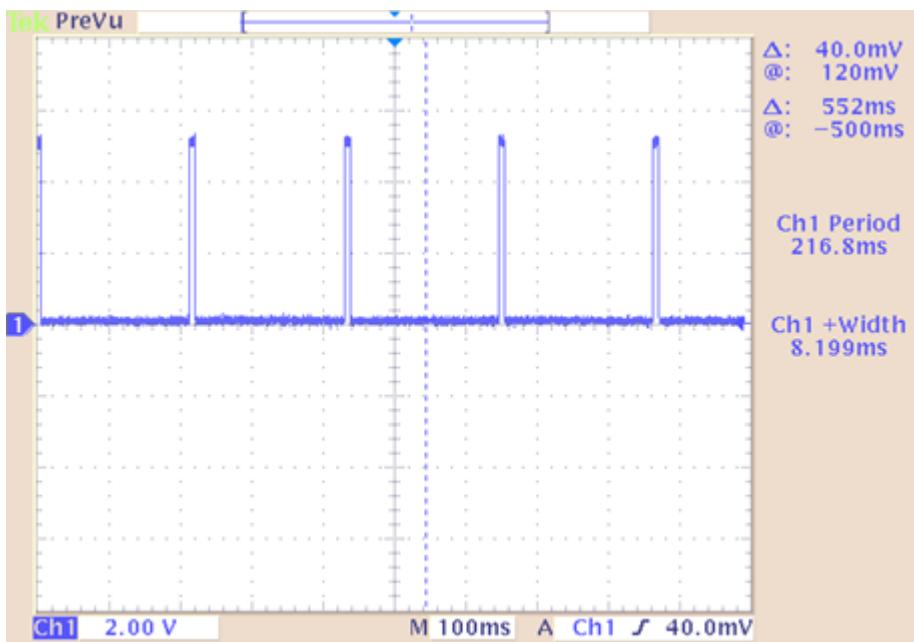
Pentru a observa din exterior pornirea din nou a aplicației (ceea ce se va întâmpla după resetarea microprocesorului), se poate, de exemplu, să se seteze și apoi să se reseteze un pin de ieșire, de exemplu PB0, înainte de a “bloca” microprocesorul.

Notă: Deoarece pinul PB0 va fi urmărit pe o durată de timp de ordinul sutelor de milisecunde, este posibil durata cât acesta va avea valoarea 1 logic să fie inobservabilă pe osciloscop. Din acest motiv este binevenită introducerea unei întârzieri înainte de a reseta pinul PB0.

9.1.1.3 Schema logică



Rezultate



9.1.1.4 Codul sursă

main.c

```
#include <iom16.h>
#include <inavr.h>
#define DELAY_CYCLES 32768

int main( void )
{
    __disable_interrupt();
    /*Setare perioadă de time-out 0.26s*/
    WDTCR|=(1<<WDP2);
    /*Activare watchdog timer*/
    WDTCR|=(1<<WDE);
    /*Setare PD6 ca pin de ieșire*/
    DDRD|=(1<<PD5);
    __enable_interrupt();
    /*Setare PD6 pe 1 logic*/
    PORTD|=(1<<PD5);
    __delay_cycles(DELAY_CYCLES);
    /*Setare PD6 pe 0 logic*/
    PORTD&=~(1<<PD5);
    while(1);
    return 0;
}
```

9.4.2 APLICAȚIA 2

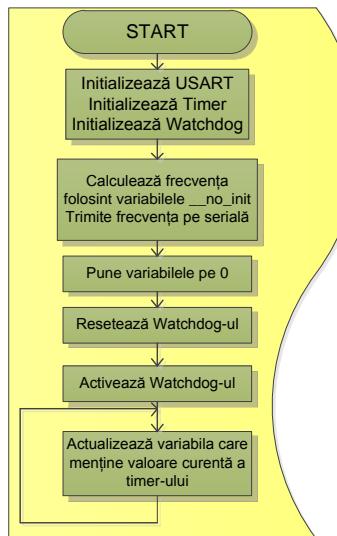
9.1.1.5 Enunț

Să se creeze o aplicație care să calculeze perioada efectivă a *watchdog – ului*.

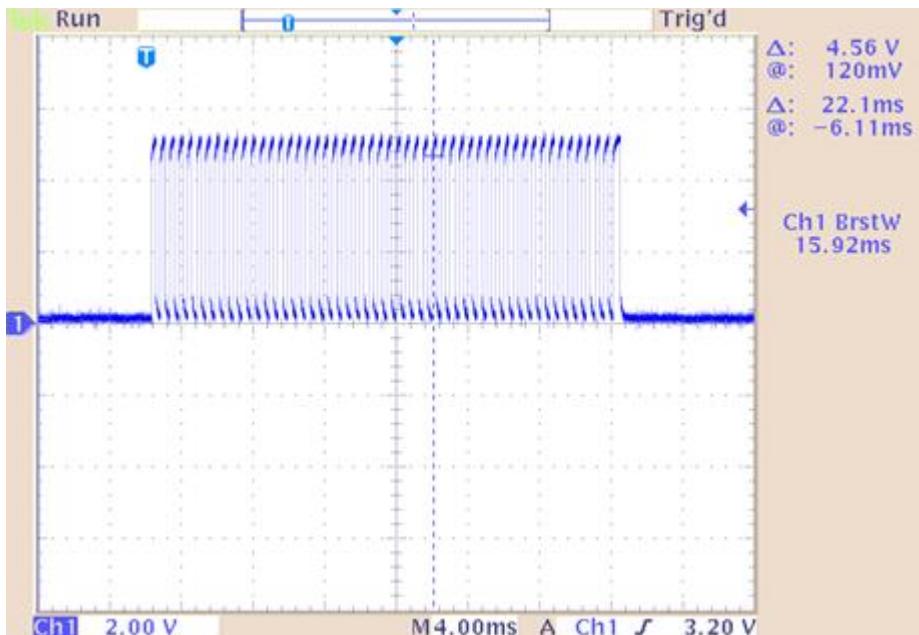
9.1.1.6 Rezolvare

Pentru a putea aproxima această perioadă cu ajutorul osciloscopului se folosește un pin de ieșire *PC1*. Pe durata perioadei de time-out a *watchdog – ului* acest pin își va schimba valoarea. Astfel diferența de tensiune va fi sesizată de oscilloscop. Prin urmare perioada aproximată a *watchdog – ului* va fi de fapt perioada de la primul front pozitiv până la ultimul front negativ al semnalului generat de pinul *PC1*. Pentru claritate înainte de resetarea *watchdog – ului* se va “bloca” microprocesorul pe durata a 100000 de cicli de clock.

9.1.1.7 Schema logică



9.1.1.8 Rezultate



9.1.1.9 Codul sursă

```
#include <iom16.h>
#include <inavr.h>

int main( void )
```

```
{  
    __enable_interrupt();  
    /*Initializare Watchdog*/  
    /*Setare time-out de 16.3 ms*/  
    WDTCR|= (~(1<<WDP0))| (~(1<<WDP1))| (~(1<<WDP2));  
    __delay_cycles(100000);  
    /*Resetare WDT*/  
    asm("WDR");  
    /*Activare WDT*/  
    WDTCR|=(1<<WDE);  
    DDRC|=(1<<DDC1);  
    PORTC|=(1<<PC1);  
    while(1)  
    {  
        PORTC^=(1<<PC1);  
        __delay_cycles(1000);  
    }  
    return 0;  
}
```

9.4.3 APLICAȚIA 3

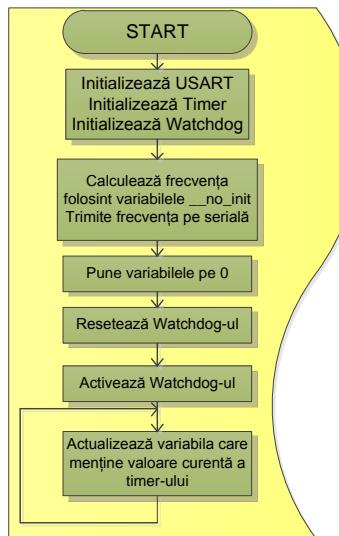
9.1.1.10 Enunț

Să se scrie o aplicație care calculează frecvența efectivă a *watchdog*-ului. Pentru calculul exact al frecvenței se va folosi unul dintre timer-ele microcontroler-ului. Frecvența va fi trimisă pe serială. **Nu** se va folosi tipul de dată **double**.

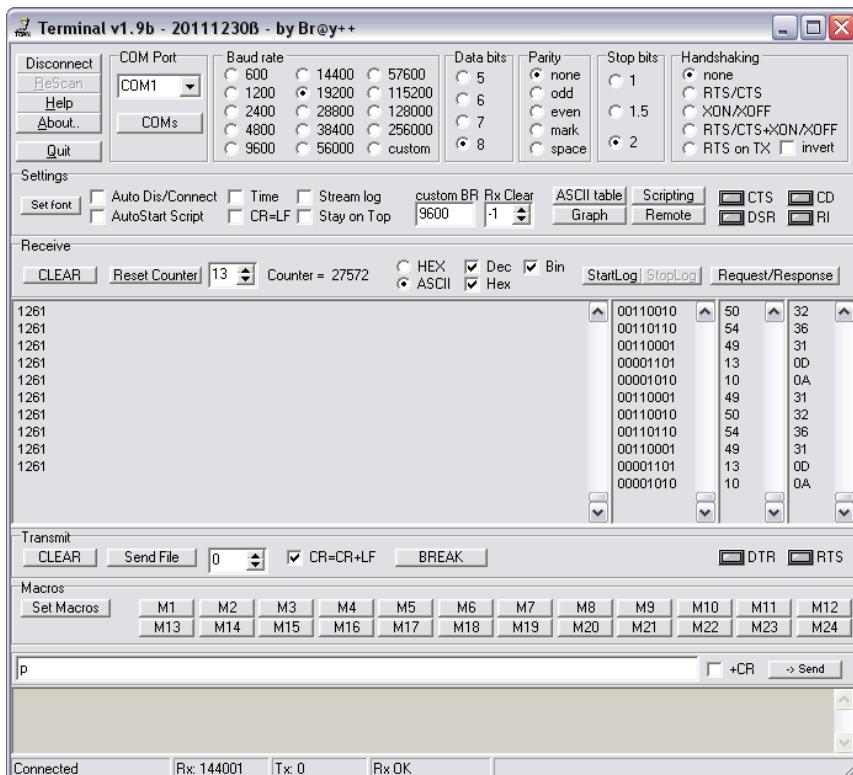
9.1.1.11 Rezolvare

Pentru a păstra valoarea curentă a timer-ului și numărul de overflow-uri se folosesc două variabile *_no_init* pentru a-și păstra valoarea și după reset. Dacă în momentul activării *watchdog*-ului se startează un timer, după resetarea microcontroler-ului se va ști perioada de time-out. Știind perioada conform formulei $T = \frac{1}{f}$ se poate calcula frecvența.

9.1.1.12 Schemă logică



9.1.1.13 Rezultate



9.1.1.14 Codul sursă

```
#include <iom16.h>
#include <inavr.h>
#include "uart.h"

__no_init unsigned int Timer1_currentValue;
__no_init unsigned int Timer1_numberOverflows;

#pragma vector = TIMER1_OVF_vect
_isr void T1_OVF()
{
    Timer1_numberOverflows++;
}

int main( void )
{
    /*initializare usart*/
    USART_initialize(BAUD_RATE);
    /*initializare Timer 1*/
    /*Mod de funcționare Normal*/
    /*Setare prescaler */
    TCCR1B|=(1<<CS10);
    /*Activare intrerupere*/
    TIMSK|=(1<<TOIE1);
    __enable_interrupt();
    /*initializare Watchdog*/
    /*Setare time-out de 32.5 ms*/
    WDTCR|= (1<<WDP0);

    if(Timer1_numberOverflows >0 || Timer1_currentValue>0)
    {
        /*calculul perioadei efective de time-out a watchdog-ului*/
        unsigned long number=Timer1_numberOverflows*65535+Timer1_currentValue;
        /*calculul perioadei în ms*/
        unsigned long period=number*0.00025;
        /*calculul perioadei unui ciclu de clock al watchdog-ului*/
        unsigned long time_per_clock=(unsigned long)((period*1000000)/32768);
        /*calculul frecvenței în Hz*/
        unsigned long freqv=(unsigned long)((1000000./time_per_clock));
        print(freqv);
    }
    Timer1_numberOverflows=0;
    Timer1_currentValue=0;
    TCNT1=0;
    /*Resetare WDT*/
    asm("WDR");
    /*Activare WDT*/
    WDTCR|=(1<<WDE);
    while(1)
    {
        Timer1_currentValue=TCNT1;
    }
    return 0;
}
```

10 Generare de frecvențe. Fast PWM, factor de umplere.

Pulse Width Modulation (PWM) este o tehnică de modulare a perioadei frecvenței unui semnal dreptunghiular prin controlarea timpului cât semnalul are valoarea “1” și cât “0” într-o perioadă.

Factorul de umplere (eng. *Duty cycle*) este valoarea ce indică cât din întreaga perioadă semnalul are valoarea “1”. Factorul de umplere se exprimă în procente.

$$\text{Duty Cycle} = \frac{p}{p+q} * 100 \%$$

unde p este timpul de “1”, q este timpul de “0”, iar $p+q$ este actuala perioadă a semnalului modulat.

PWM este considerat o implementare simplistă a unui convertor numeric-analogic. Multe aplicații de control necesită semnale analogice de control pentru diverse task-uri, astfel sistemele PWM pe microcontrolere dă posibilitatea de generare de semnale electrice de frecvențe dorite, evident limitate de capacitatele microcontroler-ului.

PWM este folosit ca metodă eficientă de generare a tensiunilor variabile pentru a controla componente externe precum motoare AC și DC sau sisteme de încălzire, fără a fi necesare componente adiționale cum ar fi convertoarele numeric-analogice. În funcție de necesitate un modul PWM poate fi implementat hardware și/sau software. Implementarea hardware diferă de la un microcontroler la altul, pe când cea software se bazează pe folosirea intreruperilor și poate fi generalizată pentru o gamă mai mare de implementări fizice.

În implementarea hardware se ține cont, în aproape toate microcontrolerele, de registrele *Timer/Counter*. Astfel generearea unei frecvențe prin PWM ține cont de frecvența microcontroler-ului și capacitatele registrelor *Timer/Counter*: rezoluție, posibilitate de output compare, posibilitate de modificare a factorului de umplere. De exemplu dacă avem o frecvență a microcontroler-ului de 1kHz și folosim un *Timer/Counter* pe 8 biți, perioada de semnal maximă ce poate fi generată este $256 \times 1\text{ms}$ adică 256ms, iar frecvența minimă este de ~4Hz.

În cazul microcontroler-ului *ATmega16* se poate genera un semnal PWM cu toate 3 registrele *Timer/Counter*. În plus, acestea pot fi comandate atât de clock-ul microcontroler-ului cât și de un clock extern.

10.1 GENERAREA UNUI SEMNAL PWM CU ATMEGA16

În funcție de frecvență ce se dorește a fi generată, microcontrolerul *ATmega16* pune la dispoziție 3 registre *Timer/Counter*, 2 din ele pe 8 biți și unul pe 16 biți, ce pot fi configurate pentru a satisface această cerință cât mai bine. Astfel utilizatorul poate alege care din cele 3 registre poate genera cel mai corect frecvență dorită.

Aceste registre pot fi controlate de clock-ul intern al microcontroler-ului, dar li se poate furniza și un clock extern.

10.2 UTILIZARE TIMER/COUNTER1

Timer/Counter1 este o unitate de timer pe 16 biți care permite acuratețe la sincronizarea execuției programului (management al evenimentelor), generare de undă. Principalele caracteristici ale timer-ului *Timer/Counter1* sunt: design pe 16 biți (permite PWM pe 16 biți), două unități independente de *Output Compare*, regiștri dublu buffer de *Output Compare*, o unitate *Input Capture*, auto reload, PWM, perioadă variabilă de PWM, generator de frecvență, patru surse independente de întreruperi (**TOV1**, **OCF1A**, **OCF1B**, **ICF1**).

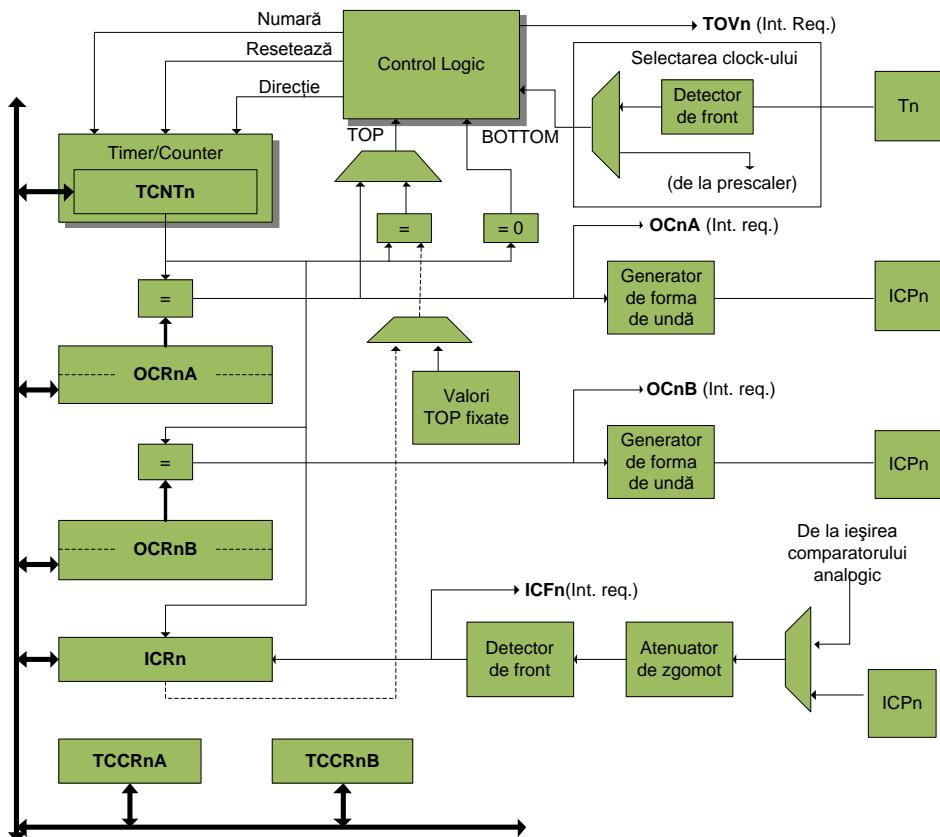


Figura 10.1 Diagrama bloc a numărătorului pe 16 biți

În generarea de frecvențe intervin *Timer/Counter* (**TCNT1**), *Input Capture Register* (**ICR1**) și *Output Compare Register* (**OCR1A/B** sau **OCR1x**), toate 3 fiind registre pe 16 biți. *Timer/Counter1* folosește 2 registre de 8 biți *Timer Counter Control Register* (**TCCR1A/B**), fără restricții de acces la procesor, pentru configurarea modulului în care vrem să utilizăm Timer-ul. Semnalele cererilor de întrerupere sunt vizibile în registrul *Timer Interrupt Flag* (**TIFR**). Toate întreruperile sunt mascate de registrul *Time Interrupt Mask* (**TIMSK**).

10.2.1 ACCESAREA REGIȘTRILOR PE 16 BIȚI

Regiștrii pe 16 biți **TCNT1**, **ICR1**, **OCR1A/B** pot fi accesăți de AVR CPU prin intermediul unei magistrale de 8 biți. Registrul pe 16 biți trebuie accesat pe octet folosindu-se 2 operații de citire sau scriere. Fiecare registru pe 16-biți are un registru temporar pe 8 biți pentru stocarea octetului High. Registrul temporar este folosit de toate registrele de 16 biți ale aceluiși timer. Accesarea octetului Low lansează o operație de citire sau scriere pe 16 biți. Când octetul Low al unui registru pe 16 biți este scris, octetul High stocat în registru temporar și octetul Low sunt copiate în registrul de 16 biți în același ciclu de clock. Când octetul Low este citit, octetul High este copiat în registrul temporar în același ciclu de clock în care octetul Low este citit. Nu toate registrele de 16 biți utilizează registrul temporar pentru octetul High. Citirea regisitrelor **OCR1A/B** nu implică utilizarea acestuia.

La o operație de scriere pe 16 biți, octetul High trebuie scris înaintea octetului Low. La o operație de citire, octetul Low trebuie citit înaintea octetului High.

Următorul exemplu de cod prezintă cum se accesează regiștrii timer pe 16 biți, presupunând că nici o întrerupere nu actualizează registrul temporar. Același principiu poate fi folosit la accesarea directă a regiștrilor **OCR1A/B** și **ICR1**. Când se utilizează limbajul C, compilatorul se ocupă de accesul pe 16 biți.

Exemplu de cod în asamblare

```
...
; Set TCNT1 to 0x01FF
ldi r17, 0x01
ldi r16, 0xFF
out TCNT1H, r17
out TCNT1L, r16
; Read TCNT1 into r17:r16
in r16, TCNT1L
in r17, TCNT1H
...
```

- Exemplu de cod C

```
unsigned int i;
...
/* Setează valoarea din TCNT1 la 0x01FF */
TCNT1 = 0x01FF;
/* Citește valoarea din TCNT1 în i */
i = TCNT1;
...
```

Codul în limbaj de asamblare returnează valoarea **TCNT1** din perechea de registre r17:r16.

Important de reținut este că accesarea regiștrilor pe 16 biți este o operație atomică. Dacă o întrerupere intervine între 2 instrucțiuni de accesare a unui registru pe 16 biți, iar codul întreruperii actualizează registrul temporar prin accesarea aceluiși sau a oricărui alt registru pe 16 biți al timer-ului, rezultatul accesării din afara întreruperii va fi eronat. Astfel, când și codul principal și cel al întreruperii actualizează registrul temporar, codul principal trebuie să dezactiveze întreruperile în timpul accesării pe 16 biți.

10.3 URMĂTORUL EXEMPLU ARATĂ CUM SE FACE O CITIRE ATOMICĂ A CONȚINUTULUI REGISTRULUI TCNT1. CITIREA REGISTRELOR ICR1 I OCR1A/B SE POATE FACE SIMILAR.

- Exemplu de cod în asamblare

TIM16_ReadTCNT1:

```
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Read TCNT1 into r17:r16
in r16,TCNT1L
in r17,TCNT1H
; Restore global interrupt flag
out SREG,r18
ret
```

- Exemplu de cod C

```
unsigned int TIM16_ReadTCNT1( void )
{
    unsigned char sreg;
    unsigned int i;
    /* Salvează flagul global de întreruperi */
    sreg = SREG;
    /* Dezactivează întreruperile */
    _CLI();
    /* Citește valoarea din TCNT1 în i */
    i = TCNT1;
    /* Restaurează flagul de întreruperi */
    SREG = sreg;
    return i;
}
```

Codul în limbaj de asamblare returnează valoarea **TCNT1** din perechea de registre r17:r16.

Următorul exemplu arată cum se face o scriere atomică a conținutului registrului **TCNT1**. Citirea registrelor **ICR1** și **OCR1A/B** se poate face similar.

- Exemplu de cod în asamblare

TIM16_WriteTCNT1:

```
; Save global interrupt flag
in r18,SREG
; Disable interrupts
cli
; Set TCNT1 to r17:r16
out TCNT1H,r17
out TCNT1L,r16
; Restore global interrupt flag
out SREG,r18
ret
```

- Exemplu de cod C

```
void TIM16_WriteTCNT1 ( unsigned int i )
{
    unsigned char sreg;
    unsigned int i;
    /* Salvează flagul global de întreruperi */
    sreg = SREG;
    /* Dezactivează întreruperile */
    _CLI();
    /* Setează valoarea din TCNT1 în i */
    TCNT1 = i;
    /* Restaurează flagul de întreruperei */
    SREG = sreg;
}
```

Codul în asamblare necesită ca regiștri r17:r16 să conțină valoarea ce trebuie scrisă în **TCNT1**.

10.3.1 UNITATEA DE NUMĂRARE

Partea principală a Timer/Counter-ului pe 16 biți este unitatea de numărare pe 16 biți, bidirecțională și programabilă. În Figura 10.2 se poate vedea structura acestei unități, unde semnificația semnalelor este următoarea:

count	incrementare sau decrementare a TCNT1 cu 1
direction	selectează între incrementare sau decrementare
clear	resetare TCNT1
clkT1	clock-ul Timer/Counter1
top	TCNT1 a ajuns la valoarea sa maximă
bottom	TCNT1 a ajuns la valoarea minimă (zero)

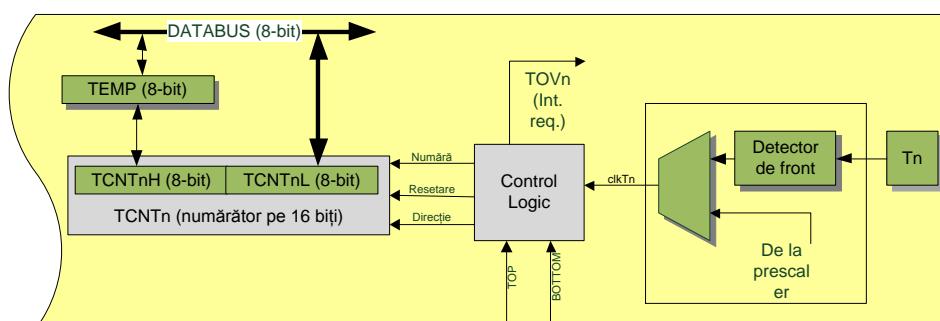


Figura 10.2 Unitatea de numărare, Diagrama Bloc

Numărătorul pe 16 biți este mapat la două locații de 8 biți de memorie I/O : *Counter High* (**TCNT1H**) conținând primii 8 biți ai numărătorului, și *Counter Low* (**TCNT1L**) conținând ultimii 8 biți. Registrul **TCNT1H** poate fi accesat doar indirect de către CPU. Atunci când CPU realizează un acces la locația **TCNT1H**, CPU accesează registrul **TEMP**. Acesta din urmă este actualizat cu valoarea **TCNT1H** atunci când **TCNT1L** este citit, iar **TCNT1H** este actualizat cu valoarea **TEMP** când **TCNT1L**

este scris. Acest lucru face posibilă scrierea sau citirea de către CPU a întregii valori de 16 biți a numărătorului într-un singur ciclu de clock prin intermediu magistralei de 8 biți.

În funcție de modul de operare folosit, numărătorul este resetat, incrementat, sau decrementat la fiecare ciclu de clock $\text{clk}_{\text{T}1}$. $\text{Clk}_{\text{T}1}$ poate fi generat extern sau intern, în funcție de biții de selecție a clock-ului (**CS12:10**). Atunci când nu este selectată nici o sursă de clock (**CS12:10=0**) timer-ul este opri. Indiferent dacă $\text{clk}_{\text{T}1}$ este prezent sau nu valoarea **TCNT1** poate fi accesată de către CPU.

Secvența de numărare este determinată de biții ce determină Modul De Generare A Formei De Undă (**WGM13:10**), aflați în regiștrii **TCCR1A** și **TCCR1B**. Flag-ul *Timer/Counter Overflow* (**TOV1**) este setat în funcție de modul de operare selectat de biții **WGM13:10**. **TOV1** poate fi folosit pentru a genera întreruperi CPU.

10.3.2 UNITATEA INPUT CAPTURE

Timer-ul încorporează o unitate *Input Capture* ce poate capta evenimente externe și le poate da o marcă de timp ce indică momentul producerii. Semnalul extern ce indică un eveniment sau mai multe evenimente poate fi aplicat pe pinul **ICP1** sau prin comparatorul analogic. Mărcile de timp pot fi folosite în calcularea frecvenței, factorului de umplere și altor caracteristici aplicate semnalelor. Mărcile de timp mai pot fi folosite în crearea unui eveniment log.

Unitatea *Input Capture* este ilustrată în diagrama bloc din Figura 10.3. Elementele ce nu fac parte direct din acest ansamblu sunt colorate cu gri. Litera “**n**” din registre și numele biților indică numărul *Timer/Counter*.

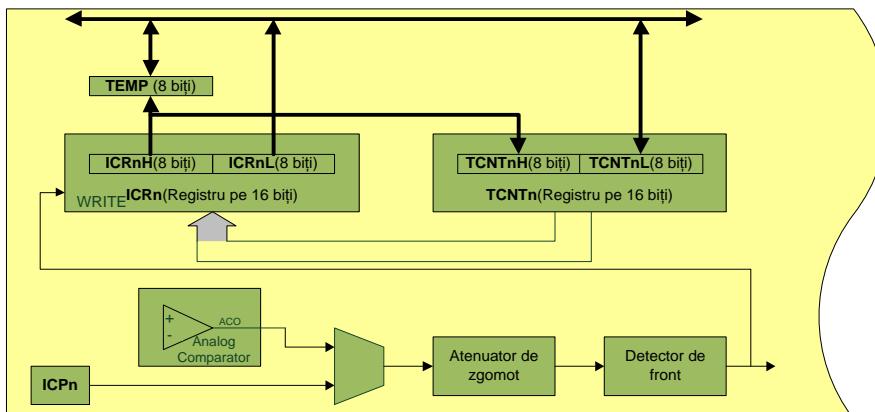


Figura 10.3 Unitatea Input Capture, Diagrama bloc

Când apare o schimbare a nivelului logic (un eveniment) apare pe *Input Capture pin* (**ICP1**), sau pe *Analog Comparator output* (**ACO**), iar schimbarea este confirmată de modul în care a fost setat detectoarea de front, o captură va fi declanșată. În acest moment, valoarea pe 16 biți a counterului **TCNT1** este scrisă în **ICR1**. Flag-ul **ICF1** este setat în aceeași perioadă de clock în care valoarea din **TCNT1** este scrisă în **ICR1**. Flag-ul **ICF1** este pus pe 1 imediat ce întreruperea este executată.

Citirea valorii pe 16 biți din **ICR1** se face prin citirea mai întâi a octetului Low (**ICR1L**) și apoi a octetului High (**ICR1H**). Când se citește octetul Low, octetul High

este copiat în registrul temporar (**TEMP**). Când procesorul citește locația I/O a **ICR1H** va accesa registrul **TEMP**.

Registrul **ICR1** poate fi scris doar în modurile de generare de forme de undă ce folosesc acest registru pentru redefinirea valorii TOP a numărătorului. În aceste cazuri modul de generare a formei de undă trebuie setat înaintea ca valoarea de TOP să poată fi scrisă în registrul **ICR1**. Atunci când se scrie registrul **ICR1** octetul High trebuie scris în **ICR1H** înaintea octetului Low în **ICR1L**.

Registrul **ICR1** poate fi scris doar în modurile de generare de forme de undă pentru redefinirea valorii de TOP a registrului **TCNT1**. Astfel biții de mod **WGM13:10** trebuie setați înaintea de scrierea unei valori în **ICR1**.

10.1.1.1 Input Capture Pin Source

Sursa principală de declanșare pentru unitatea *Input Capture* este *Input Capture pin (ICP1)*. *Timer/Counter1* poate folosi unitatea *Analog Comparator output (ACO)* că sursă de declanșare pentru unitatea *Input Capture*. Această unitate este selectată ca sursă de declanșare prin setarea bitului *Analog Comparator Input Capture (ACIC)* din registrul *Analog Comparator Control and Status Register (ACSR)*. A se avea grija la schimbarea sursei de declanșare deoarece aceasta poate declanșa o captură. Astfel flag-ul **ICF1** trebuie eliberat după efectuarea schimbării.

Atât **ICP1** cât și **ACO** sunt evaluate folosind aceeași tehnică ca pentru pinul **T1**. Detectorul de front este de asemenea identic. De altfel, dacă atenuatorul de zgomot este activ, o logică adițională este inserată înaintea acestuia, fapt ce crește întârzierea cu 4 cicli de clock. Intrarea atenuatorului de zgomot și detectorului de front este întotdeauna activă cu excepția modurilor de generare de forme de undă ce folosesc **ICR1** pentru a redefini valoarea de TOP.

O captură de intrare poate fi declanșată software prin controlarea portului în care se găsește pinul **ICP1**.

10.3.2.2 Utilizarea unității Input Capture

Principala provocare în utilizarea unității *Input Capture* este alocarea de suficiente resurse procesor pentru tratarea evenimentelor. Timpul între 2 evenimente este critic. Dacă procesorul nu a citit valoarea captată în registrul **ICR1** înainte de apariția evenimentului următor, **ICR1** va fi suprascris cu noua valoare. În acest caz, rezultatul capturii va fi incorect.

Când se folosește întreruperea *Input Capture*, registrul **ICR1** ar trebui citit cât mai repede posibil în rutina de tratare a întreruperii. Chiar dacă întreruperea *Input Capture* are o prioritate relativ ridicată, timpul maxim de răspuns al întreruperii este dependent numărul de cicli de clock necesari tratării altor cereri de întrerupere.

Utilizarea unității *Input Capture* în alte moduri de operare în afară de cele în care valoarea de TOP trebuie schimbată nu este recomandat.

Măsurarea factorului de umplere a unui semnal extern necesită ca frontul de declanșare să fie schimbat după fiecare captură. Schimbarea detectării frontului trebuie făcută pe cât de repede posibil după ce registrul **ICR1** a fost citit. După o schimbare de front, flag-ul **ICF1** trebuie eliberat prin software (scrierea a “1” logic la adresa acestuia). Pentru măsurarea frecvenței, eliberarea flag-ului **ICF1** nu este necesară (dacă se folosește o întrerupere).

10.3.3 UNITATEA OUTPUT COMPARE

Comparitorul face continuu comparări între **TCNT1** și **Output Compare Register (OCR1x)**, iar în cazul în care **TCNT1** este egal cu **OCR1x**, se va genera o potrivire între semnale. Potrivirea va seta **Output Compare Flag (OCF1x)** la următorul ciclu de clock. Dacă este activat (**OCIE1x = 1**), **OCF1x** va genera o întrerupere la ieșire și se va reseta automat în caz că întreruperea este executată. Generatorul de undă folosește potrivirea de semnal în concordanță cu modul de operare setat de bițiii Waveform Generator mode(**WGM13:0**) și Compare Output mode (**COM1x1:0**).

O caracteristică specială a **OCR1A** este ca permite definirea valorii TOP a timerului, valoare ce definește perioada de timp pentru unda generată de Waveform Generator.

În Figura 10.4 este prezentată diagrama bloc a *Output Compare*:

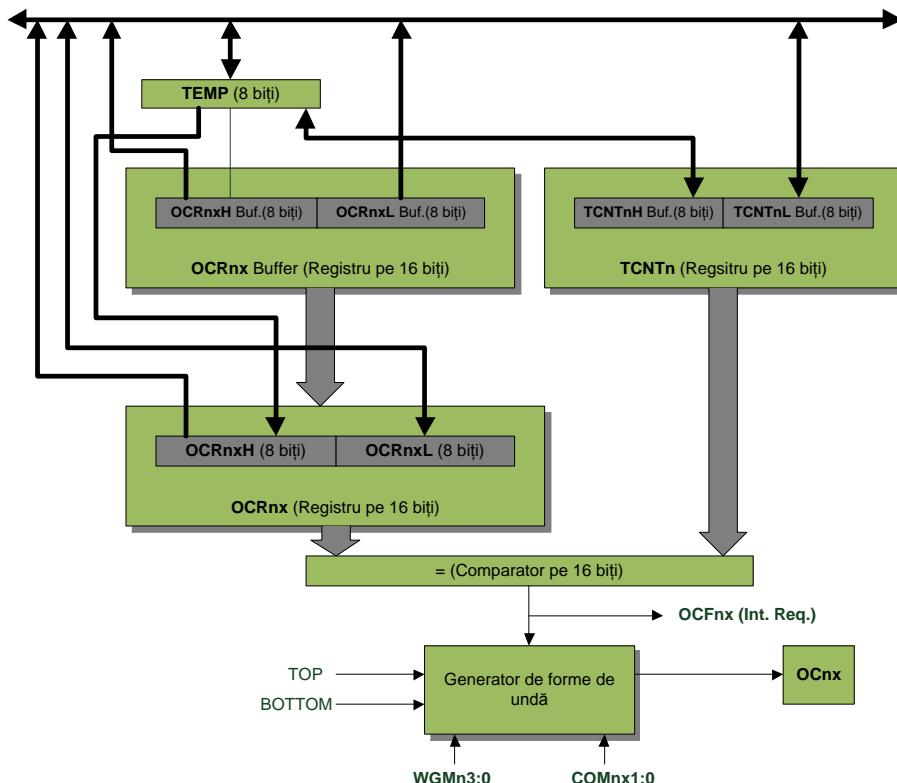


Figura 10.4 Diagrama bloc Unitatea Output Compare

10.3.3.1 Utilizarea unității Output Compare

Cum scrierea **TCNT1** în oricare mod de operare blochează toate potrivirile la comparare pentru un ciclu de timer clock, există riscuri în schimbarea **TCNT1** când sunt utilizate oricare din unitățile *Output Compare*, chiar dacă *Timer/Counter* este în funcțiune sau nu. Dacă valoarea scrisă în **TCNT1** este egală cu valoarea din **OCR1x**, potrivirea la comparare va fi sărită rezultând un semnal generat incorrect. A nu se scrie

TCNT1 egal cu **TOP** în modurile PWM cu valori de **TOP** variabile. Potrivirea la comparare pentru **TOP** va fi ignorată și numărătorul va continua până la **0xFFFF**. Similar a nu se egala **TCNT1** cu **BOTTOM** când numărătorul funcționează descrescător.

Setarea **OC1x** trebuie făcută înainte de setarea registrului Data Direction pentru pinul de ieșire. Cea mai ușoară modalitate de setare a valorii **OC1x** este folosirea biților Force Output Compare (**FOC1x**) în modul Normal. Registrul **OC1x** își păstrează valoarea la schimbarea modurilor de generare de forme de undă.

A se avea grijă că biții **COM1x1:0** nu sunt double buffered împreună cu valoarea comparată. Schimbarea acestora va declanșa efectul dorit imediat.

10.3.3.2 Unitatea Compare match output

Biții *Compare Output mode* (**COM1x1:0**) au două funcții. Generatorul de forme de undă îi folosește pentru definirea stării *Output Compare* (**OC1x**) la următoarea potrivire. În al doilea rând aceștia controlează sursa de ieșire a pinului **OC1x**. Figura 10.5 prezintă o schemă simplificată a logicii afectate de setarea biților **COM1x1:0**. Registrele, biții și pinii I/O din figură sunt evidențiate prin bold. Sunt prezentate doar părțile generale ale regiștrilor de *I/O Port Control* (**DDR** și **PORT**) care sunt afectate. Când spunem starea **OC1x**, ne referim la registrul intern **OC1x** și nu pinul **OC1x**. La apariția unui reset de sistem, registrul **OC1x** este resetat la „0”.

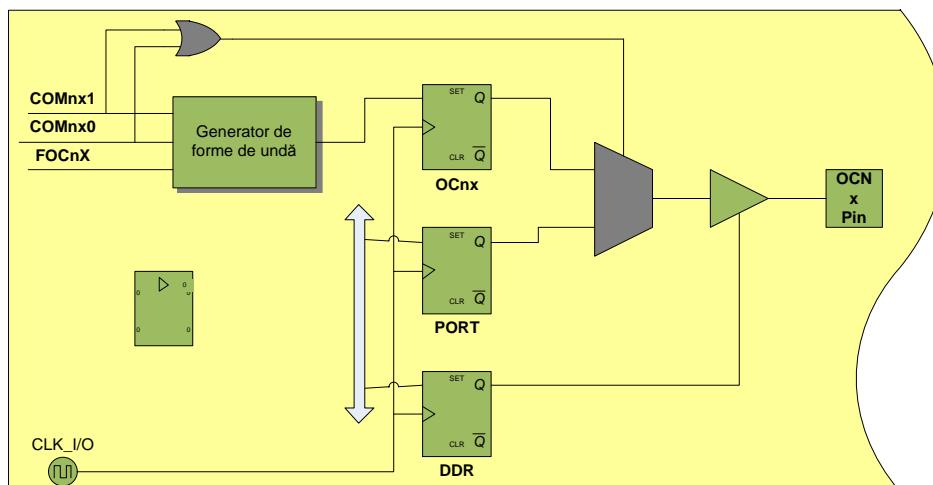


Figura 10.5 SchemăUnitate Compare match output

Functia generală a portului I/O este încălcată de **OC1x** din generatorul de forme de undă dacă oricare din biții **COM1x1:0** sunt setați. De altfel, direcția pinului **OC1x** (intrare sau ieșire) este încă controlată de *Data Direction Register* (**DDR**). Bitul pentru pinul **OC1x** din **DDR** (**DDR_OC1x**) trebuie setat înainte ca valoarea de ieșire pe pinul **OC1x** să fie vizibilă.

Logica pinului de output compare permite initializarea stării **OC1x** înainte ca ieșirea să fie activată. Unele setări ale bițiilor **COM1x1:0** sunt rezervate pentru anumite moduri de operare. Aceștia nu au nici un efect asupra unității Input Capture.

10.3.3.3 Modul Compare Output si Generarea de forme de undă

Generatorul de forme de undă utilizează biții **COM1x1:0** diferit în modurile normal, CTC și PWM. Pentru toate modurile **COM1x1:0** semnalează generatorului de forme de undă că nici o acțiune nu va fi făcută asupra registrului **OC1x** la următoarea potrivire. O schimbare a stării bițiilor **COM1x1:0** va lua efect la prima potrivire după ce biții au fost scrisi.

10.3.4 MODURI DE OPERARE

Timer-ul 1 poate funcționa în cinci moduri de operare: mod normal, mod Clear Timer on Compare Match(CTC), mod Fast PWM, mod Phase Correct PWM, mod Phase and Frequency Correct PWM.

10.3.4.1 Modul Fast PWM

Modul Fast Pulse Width Modulation sau modul Fast PWM (**WGM13:0** = 5,6,7,14, sau 15) furnizează o opțiune de generare a unei unde PWM de frecvență mare. Fast PWM diferă de alte PWM-uri prin funcționarea pe o singură pantă. Counter-ul numără de la BOTTOM la TOP apoi se restartează de la BOTTOM. La modul non-inversat Compare Output, Output Compare (**OC1x**) este resetat când valorile lui **TCNT1** și **OCR1x** sunt egale și este setat la BOTTOM.

Datorită funcționării pe o singură pantă, frecvența de operare a modului Fast PWM poate fi de două ori mai mare decât în cazul modurilor care funcționează pe două pante. Frecvența mare a modului Fast PWM face acest mod potrivit pentru regulările, rectificările de tensiune și aplicațiile DAC. Funcționarea la frecvențe înalte permite și componentelor de mici dimensiuni să funcționeze (bobine, condensatori), reducându-se, prin urmare, consumul sistemului.

Rezoluția PWM-ului pentru modul Fast PWM poate fi fixată la 8, 9 sau 10 biți sau definită de **ICR1** sau **OCR1A**. Rezoluția minimă admisă este de 2 biți (**ICR1** sau **OCR1A** setat la **0x0003**) și cea maximă de 16 biți (**ICR1** sau **OCR1A** setat la MAX). Rezoluția PWM-ului în biți poate fi calculată folosind următoarea ecuație:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

În modul Fast PWM counterul este incrementat până când valoarea lui atinge una din valorile fixate **0x00FF**, **0x01FF** sau **0x03FF** (**WGM13:0** = 5, 6 sau 7), valoarea în **ICR1** (**WGM13:0** = 14), sau valoarea în **OCR1A** (**WGM13:0** = 15). Counter-ul este resetat la următorul ciclu de clock al timer-ului.

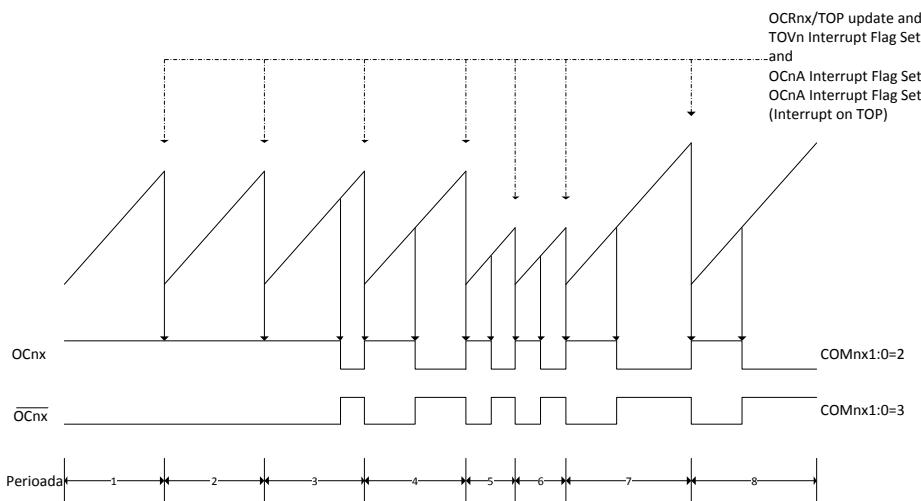


Figura 10.6 Modul Fast PWM, Diagrama de timp

În Figura 10.6 este descris modul de operare Fast PWM când **OCR1A** sau **ICR1** sunt utilizati ca TOP. Valorile lui **TCNT1** din diagramă ilustrează modul de operare pe o singură pantă. Linia orizontală dintre pantele lui **TCNT1**, reprezintă comparările dintre **OCR1x** și **TCNT1**. **OC1x** va fi setat atunci când apare o potrivire de valori.

Timer/Counter Overflow Flag (TOV1) este setat de fiecare dată când counterul ajunge la TOP. În plus, **OC1A** sau **ICF1** Flag este setat la același ciclu de clock al timer-ului ca **TOV1** atunci când **OCR1A** sau **ICR1** este folosit pentru a defini valoarea **TOP**.

Dacă vreuna dintre întreruperi este permisă, rutina de întreruperi poate fi folosită pentru updatarea valorii **TOP** și compararea valorilor.

Când se schimbă valoarea **TOP**, programul trebuie să se asigure că noua valoare TOP este mai mare sau egală cu valoarea tuturor celorlalți registrii. Dacă valoarea TOP este mai mică decât una dintre valorile regiștrilor Compare nu va apărea nici o potrivire de valori între **TCNT1** și **OCR1x**.

Când sunt folosite valori fixe TOP, biții nefolosiți sunt reduși la zero când oricare dintre valorile regiștrilor **OCR1x** este stabilită.

Procedura de updatare pentru **ICR1** diferă de updatarea pentru **OCR1A** când este folosită pentru definirea valorii TOP. Registrul **ICR1** nu este dublu buffered. Asta înseamnă că, dacă **ICR1** este schimbat la o valoare mai mică când counter-ul funcționează fără nici o valoare sau cu o valoare mică a prescaler-ului, atunci există riscul ca noua valoare stabilită pentru **ICR1** să fie mai mică decât valoarea curentă a **TCNT1**. În acest fel counter-ul nu va atinge nici o valoare comună la valoarea **TOP**. Counter-ul va trebui atunci să numere până la valoarea MAX (**0xFFFF**) și să revină la **0x0000** pentru a atinge o valoare comună.

Registrul **OCR1A** este dublu buffered. Această caracteristică permite ca locația I/O pentru **OCR1A** să fie scrisă oricând. Când locația I/O pentru **OCR1A** este stabilită

aceasta va fi pusă pe buffer-ul registrului **OCR1A**. Registrul **OCR1A** va fi updatat la următorul ciclu al timer-ului de clock când **TCNT1** atinge TOP. Updatarea este făcută în același ciclu al timer clock-ului în care **TCNT1** este resetat și **TOV1** Flag este setat.

Folosirea registrului **ICR1** pentru definirea valorii TOP este potrivită atunci când sunt folosite valori TOP fixe. Folosind **ICR1**, registrul **OCR1A** este liber să fie folosit pentru a genera o ieșire PWM pe **OC1A**. Cu toate acestea, în cazul în care frecvența de bază a PWM este schimbată activ (prin schimbarea valorii TOP), folosirea lui **OCR1A** pentru TOP este o alegere mai bună datorită buffer-ului dublu.

În modul Fast PWM, unitățile Compare permit generarea undelor PWM pe pinii **OC1x**. Setarea bițiilor **COM1x1:0** la 2 va produce un PWM non-inversat, iar un PWM inversat poate fi obținut prin setarea lui **COM1x1:0** la 3. Valoarea actuală a lui **OC1x** va fi vizibilă doar pe pinul portului dacă direcția pentru acesta este setată ca ieșire (**DDR_OC1x**). Unda PWM este generată prin setarea (resetarea) registrului **OC1x** la o valoarea comună între **OCR1x** și **TCNT1** și setarea (resetarea) registrului **OC1x** la ciclul timer clock-ului când counterul este setat (se schimbă din TOP în BOTTOM). Frecvența PWM pentru ieșire poate fi calculată prin următoarea formulă:

$$f_{OCnxPWM} = \frac{f_{clk_I/O}}{N * (1 + TOP)}$$

Variabila N reprezintă divizorul pentru prescaler (1, 8, 64, 256, or 1024).

Valorile extreme pentru registrul **OCR1x** reprezintă cazuri speciale când este generată o undă de ieșire PWM în modul Fast PWM. Setând valoarea lui **OCR1x** egală cu **TOP** va rezulta o ieșire înaltă sau joasă (depinzând de polaritatea ieșirii setată de biții **COM1x1:0**).

O anumită frecvență a undei de ieșire (cu 50% duty cycle) în modul Fast PWM poate fi obținută prin setarea lui **OC1A** să-și comute level-ul logic la fiecare valoare comună (**COM1A1:0 = 1**). Asta se aplică doar dacă **OCR1A** este folosit pentru a defini valoarea TOP (**WGM13:0 = 15**). Unda generată va avea frecvență maximă la

$f_{OC1A} = f_{clk_I/O} / 2$ când **OCR1A** este setat la zero (**0x0000**). Această caracteristică este similară și la comutarea **OC1A** în modul CTC, exceptând caracteristica că bufferul dublu al unității *Output Compare* există doar în modul Fast PWM.

10.3.5 DESCRIEREA REGIȘTRILOR

Timer-ul pe 16 biți Timer1 conține un set de regiștri care trebuie configurați pentru ca să funcționeze în modul de operare dorit, în cazul nostru Fast PWM. O descriere a regiștrilor timerului *Timer/Counter1* este făcută în continuare.

10.3.5.1 Timer/Counter1 Control Register A – TCCR1A

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TCCR1A							
Valoare inițială	0	0	0	0	0	0	0	0	

Biți 7:6 - COM1A1:0 : Modul Compare Output pentru canalul A

Biți 5:4 - COM1B1:0 : Modul Compare Output pentru canalul B

Biții **COM1A1:0** și **COM1B1:0** controlează comportamentul pinilor de *Output Compare* (**OC1A** și respectiv **OC1B**). Dacă unul sau amândoi din biții **COM1A1:0** sunt scriși pe 1, ieșirea **OC1A** suprascrie funcționalitatea normală a portului I/O la care este conectat pinul. Același lucru și în cazul **COM1B1:0**, cu precizarea că cel care suprascrie este **OC1B**. În orice caz bitul coresponzător lui **OC1A** sau **OC1B** din *registrul Data Direction* (**DDR**) trebuie setat ca ieșire. Atunci când **OC1A** sau **OC1B** este conectat la pin, funcționalitatea **COM1x1:0** este dependentă de setările din **WGM13:10**.

COM1 A1/CO M1B1	COM1A0/ COM1B0	Descriere
0	0	Operație normală pe port, OC1A/OC1B deconectate.
0	1	WGM13:0 = 15: Folosește OC1A la potrivire, OC1B deconectat, operație normală pe port). Pentru toate celelalte moduri WGM13:0, operație normală pe port, OCnA/OCnB deconectate.
1	0	Resetează OC1A/OC1B la potrivire, setează OC1A/OC1B la BOTTOM, (mod neinversabil)
1	1	Setează OC1A/OC1B la potrivire, resetează OC1A/OC1B la BOTTOM, (mod inversabil)

Tabel 100.1 Modul Compare Output, Fast PWM

Bitul 3 – FOC1A: Force Output Compare pentru canalul A

Bitul 2 – FOC1B: Force Output Compare pentru canalul B

Acești biți sunt activi numai când **WGM13:0** specifică un mod non-PWM. În orice caz, pentru a asigura compatibilitate cu viitoarele dispozitive, acești biți trebuie setați pe zero atunci când **TCCR1A** este scris, când se operează în moduri PWM. Când se scrie 1 logic la acești biți este forțată o potrivire imediată la unitatea de generare a formelor de undă.

Biți 1:0 – WGM11:10 :Modul de generare a formelor de undă

Mode	WG M13	WGM 12	WG M11	WGM 10	Moduri de operare ale Timer/Counter	TOP	Actualizare OCR1x	Flag-ul TOV1 setat pe
0	0	0	0	0	Normală	0xFFFF	Imediată	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	Imediată	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	Imediată	MAX
13	1	1	0	1	Rezervat	-	-	-
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Tabel 100.2 Moduri de operare

Notă: FPWM = Fast PWM
 PFC = Phase and Frequency Correct
 PC = Phase Correct

10.3.5.2 Timer/Counter1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	TCCR1B
Read/Write	R/W								
Valoare inițială	0	0	0	0	0	0	0	0	

Bitul 7 – ICNC1 : Input Capture Noise Canceler

Setând acest bit (valoarea 1) se activează Input Noise Canceler.

Bitul 6 – ICES1 : Input Capture Edge Select

Bitul 5 – Rezervat

Biți 4:3 – WGM13:12: Modul de generare formei de undă

A se vedea registrul **TCCR1A**

Biți 2:0 – CS12:10 : Selectia clock-ului

CS12	CS11	CS10	Descriere
0	0	0	Fără sursă clock (Timer/Counter oprit)
0	0	1	clkI/O (fără prescaler)
0	1	0	clkI/O/8 (de la prescaler)
0	1	1	clkI/O/64 (de la prescaler)
1	0	0	clkI/O/254 (de la prescaler)
1	0	1	clkI/O/1024 (de la prescaler)
1	1	0	Sursă externă de clock pe pinul T0. Clock pe front negativ.
1	1	1	Sursă externă de clock pe pinul T0. Clock pe front pozitiv.

Tabel 100.3 Descrierea bițiilor de selecție a clock-ului

10.3.5.3 Timer/Counter1 – TCNT1H și TCNT1L

Bit	7	6	5	4	3	2	1	0	TCNT1H
TCNT1[15:8]									
TCNT1[7:0]									
Read/Write	R/W	TCNT1L							
Valoare inițială	0	0	0	0	0	0	0	0	

Cele două locații dău acces direct la numărătorul pe 16 biți a *Timer/Counter*-ului, atât pentru operații de scriere, cât și pentru operații de citire. Modificarea valorii din **TCNT1** în timp ce counter-ul este activ introduce riscul de a pierde o potrivire între **TCNT1** și unul din registrii **OCR1x**. Scriind în **TCNT1** blochează potrivirea în următorul ciclu de clock pentru toate unitățile de comparare.

10.3.5.4 Output Compare Register 1 A – OCR1AH și OCR1AL

Bit	7	6	5	4	3	2	1	0	OCR1AH
OCR1A[15:8]									
OCR1A[7:0]									
Read/Write	R/W	OCR1AL							
Valoare inițială	0	0	0	0	0	0	0	0	

10.3.5.5 Output Compare Register 1 B – OCR1BH și OCR1BL

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	OCR1BH							
Valoare inițială	0	0	0	0	0	0	0	0	OCR1BL

Registrul Output Compare conține o valoare pe 16 biți ce este comparată permanent cu valoarea **TCNT1**. O potrivire poate fi folosită să genereze o întrerupere, sau să genereze o formă de undă la ieșirea pinului **OC1x**.

10.3.5.6 Input Capture Register 1 – ICR1H și ICR1L

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	ICR1H							
Valoare inițială	0	0	0	0	0	0	0	0	ICR1L

Input Capture se updatează cu valoarea lui **TCNT1** de fiecare dată când intervine un eveniment pe pinul **ICP1**. Input Capture poate fi utilizat pentru definirea valorii TOP a counter-ului.

10.3.5.7 Timer/Counter Interrupt Mask Register – TIMSK

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TIMSK							
Valoare inițială	0	0	0	0	0	0	0	0	

Bitul 5 – TICIE1 : Timer/Counter1 Input Capture Interrupt Enable

Bitul 4 – OCIE1A: Timer/Counter1 Output Compare A Match Interrupt Enable

Bitul 3 – OCIE1B: Timer/Counter1 Output Compare B Match Interrupt Enable

Bitul 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable

Atunci când unul din acești biți este setat pe 1 , iar întreruperile sunt activate global, se activează întreruperea corespunzătoare. Vectorul corespunzător întreruperii este executat atunci când flag-ul corespunzător întreruperii, din registrul **TIFR** (*Timer/Counter Interrupt Flag Register*) este setat.

10.4 PAȘ II PENTRU GENERAREA UNEI FORME DE UNDĂ CU ATMEGA16

Exemplu: Să se genereze un semnal cu frecvență de 1kHz utilizând Timer/Counter1 în modul Fast PWM (14) fără prescaler, cu factor de umplere de 50%. Trebuie specificat că ATmega16 are un clock intern de 4MHz, dar acesta este cazul ideal. În realitate acesta funcționează la ~4MHz. Scopul este de a genera o frecvență de 1kHz reală, pornind de la câteva date ideale.

Pasul 1. Se verifică câte perioade de clock ale microcontroler-ului sunt necesare pentru a compune o perioadă din semnalul cerut.

$$\frac{1}{4000000} = 0.00000025 \text{ s}$$
$$\frac{1}{1000} = 0.001 \text{ s}$$

Rezultă că perioada noastră ar trebui să cuprindă exact 4000 perioade de clock interne. Acest număr va fi stocat în registrul ICR1 (va fi valoarea de TOP a TCNT1). Factorul de umplere fiind de 50% va avea valoarea OCR1A=(ICR1)/2.

Pasul 2. Construirea programului de generare a frecvenței

```
#include<avr.h>
#include<iom16.h>

int cnt=0; //variabila pentru întrerupere

//întreruperea de overflow la timer1
#pragma vector=TIMER1_OVF_vect
__interrupt void T1int(void)
{
    cnt++;
}

//configurarea timer 1
void timer1_INIT()
{
    /*
    Modul Fast PWM: WGM13:10=1110;
    Fără prescaler: CS12:10=001;
    Compare Output Mode: 10
    */
    TCCR1B|=(1<<WGM13)|(1<<WGM12)|(1<<CS10);
    TCCR1A|=(1<<COM1A1)|(1<<WGM11);
    TIMSK|=(1<<TOIE1);
}
```

```
void main()
{
    //Alegerea pinului de ieșire
    DDRD|=(1<<PD5);
    PORTD&=(1<<PD5);

    //Initializarea timer-ului
    timer1_INIT();

    //Valoarea care se încarcă în registrul în funcție de
    //frecvența necesară
    ICR1=4000;
    //Factor de umplere de 50%
    OCR1A=ICR1>>1;

    //Pornirea întreruperii
    __enable_interrupt();
    while(1)
    {
    }
}
```

Pasul 3. Se execută programul, se verifică cu COUNTER HM 8021-4 și se constată că frecvența generată nu este exact 1kHz.

Pasul 4. Se calculează perioada frecvenței obținute și se utilizează regula de 3 simplă pentru a calcula noua valoare ce trebuie înregistrată în ICR1.

Perioada.....	ICR1
Calculată.....	4000
1ms.....	x

Este posibil ca și de această data să nu rezulte exact 1kHz. Se va reaplica pasul 4 sau dacă frecvența rezultată este deja foarte apropiată de rezultatul dorit se va modifica ICR1 intuitiv.



Figura 100.7 Forma finală a semnalului generat

10.5 APLICAȚII

Să se genereze un semnal Fast PWM utilizând Timer/Counter1 (mod 14).

- Frecvența semnalului inițială este de 1kHz. Factorul de umplere este de 50%. Utilizând terminalului la trimiterea caracterului "+" frecvența va crește cu 1kHz, iar la trimiterea caracterului "-" frecvența va scade cu 1kHz intervalul de variație fiind [1kHz,10kHz]. Factorul de umplere trebuie să rămână la 50%.
- Având un factor de umplere initial de 5%, utilizând terminalului la trimiterea caracterului "+" factorul de umplere va crește cu 5%, iar la trimiterea caracterului "-" va scade cu 5%, intervalul de variație fiind [5%,95%]. Frecvența rămâne constantă.
- Frecvența inițială este de 1kHz și factorul de umplere inițial de 5%. Utilizând terminalului la trimiterea caracterului "+" frecvența va crește cu 500Hz și factorul de umplere cu 3%, iar la trimiterea caracterului "-" frecvența va scade cu 500Hz și factorul de umplere cu 3%. Intervalul de variație pentru frecvență: [1kHz,15kHz], iar pentru factorul de umplere [5%,95%].

11 Periodmetrul. Calcul timp.

11.1 PERIODMETRUL

11.1.1 DEFINIȚIE

Periodmetru este un aparat electronic folosit pentru măsurarea perioadelor semnalelor periodice. Pentru frecvențele mici când eroarea frecvenț metrului crește foarte mult datorită numărului mic de impulsuri contorizate pe durata timpului de măsurare se folosește schema unui periodmetru care este asemănătoare cu a frecvenț metrului doar că timpul de măsurare este definit de semnalul necunoscut iar impulsurile de numărat sunt cele provenite de la oscilator. Impulsurile de la oscilator au o foarte mare importanță incrementarea numărătorului iar restul este asemănător frecvenț metrului.

De observat că, *frecvențmetru numeric singular* se utilizează, de regulă, ca aparat de tablou și mai rar ca aparat de laborator, deoarece, în acest ultim caz, este mai economică includerea lui într-un aparat cu funcționalități multiple: *frecvențmetru/periodmetru*, numărător/temporizator universal (foarte răspândit până la apariția unor instrumente similare, dar bazate pe microprocesor), generator de semnal, osciloscop numeric,etc. Frecvențmetrul numeric poate fi utilizat și pentru măsurarea tensiunilor sau curentilor, prin asocierea sa cu un convertor tensiune-frecvență.

Un exemplu de periodmetru este aparatul de măsură Hameg 8021-4, care poate fi folosit și ca frecvenț metru astăzi cum s-a prezentat în laboratorul 8.

Așadar, la frecvențe mici, singura modalitate de a măsura precis frecvența, fără a mări exagerat timpul de măsură, este folosirea modului periodmetru.

Principala caracteristica a periodmetrului este că, tactul numărătorului este furnizat de baza de timp, din care sunt selectate de această dată frecvențe mari, începând cu frecvența oscilatorului ;

Așadar un semnal se numește periodic dacă se reproduce identic la intervale de timp egale. Cel mai mic dintre aceste intervale este perioada T , iar numărul de perioade ce au loc într-o secundă reprezintă frecvența $f=1/T$.

Există două metode de măsurare a frecvențelor : *analogice*(metode ce se bazează pe compararea cu o alta frecvență), *numerice* (se bazează pe numărarea ciclilor într-un interval de timp dat).

Măsurarea numerică a perioadei, este o metodă care constă în numărarea a numărului de cicli de clock trecuți de la startul perioadei până la sfârșitul acesteia.

Așadar marea majoritate a aplicațiilor de control se bazează pe măsurarea timpului. Mai exact, este necesar să știm când s-a scurs un anumit timp.

Există trei moduri în care se poate face acest lucru:

- Primul mod în care se poate aştepta un anumit interval de timp este să se numere. De exemplu, se poate determina când a trecut un minut numărând de la 120 la 180, făcând câte un pas la fiecare secundă. În software aceasta metodă are drept corespondent incrementarea unei variabile într-o buclă for:

```
for(i=120; i<180; i++){wait(o_secunda);}
```

Metoda are dezavantajul că atât timp cât se numără nu se mai pot executa alte acțiuni; Toată forța de calcul a procesorului este folosită pentru incrementare. În plus metoda este imprecisă. Cu toate acestea metoda este uneori utilizată.

- Al doilea mod în care se poate aştepta presupune să avem un ceas sau un temporizator.
În acest caz aşteptarea este mult mai precisă și în plus se pot executa și alte acțiuni în timp ce temporizatorul funcționează. Într-un sistem de calcul rolul ceasului sau al temporizatorului este îndeplinit de numărătoare.
- Al treilea mod reprezintă o îmbunătățire a modului doi. Se va adăuga la temporizator o întârziere care se va activa la o anumită valoare dată. La procesare acest mod de lucru se numește execuție pe întârziere și necesită tot numărătoare, ca la modul 2.

11.2 MĂSURAREA INTERVALELOR DE TEMP

O altă aplicație frecventă a numărătoarelor constă în măsurarea perioadei unui semnal: de exemplu se dorește construirea unui turometru electronic. Pe axul motorului se montează un senzor care oferă un puls la fiecare rotație completă a motorului, notat „Puls de la traductor – PT” în Figura 11.1. Perioada acestuia este notată T_x în figură.

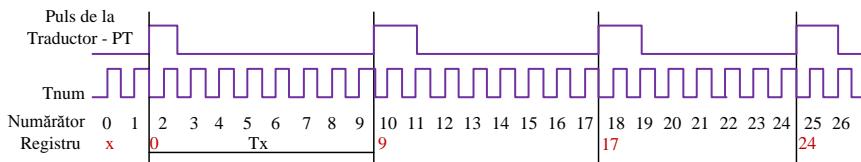


Figura 11.1

Pentru a calcula perioada semnalului PT se folosește metoda ceasului pentru măsurarea duratei unei activități sau fenomene: se notează timpul la care a început activitatea și timpul la care aceasta s-a sfârșit și apoi se face diferența celor doi timpi. Numărătorul joacă rolul ceasului. În Figura 11.1 prima perioadă a lui PT începe când numărătorul are valoarea 1 și se termină când are valoarea 9. Perioada T_x a lui PT este $9-1=8$ perioade ale ceasului numărătorului. Pentru un T_{num} de 1us obținem $T_x=8$ us.

Măsurarea perioadei lui TP se face încontinuu. Pentru următoarele perioade diferența timpilor este $17-9=8$ și $24-17=7$.

Desigur, valoarea 8 are scop didactic. În realitate turația unui motor variază între 800 și 10000 de rotații pe minut, ceea ce însemnă 13 – 167 rotații pe secundă, adică o perioadă între 77 ms și 6 ms. Pe baza acestor două valori se alege perioada Tnum. Tnum trebuie să fie suficient de mare ca 77 ms să se reprezinte pe cel mult 16 biți, cât este dimensiunea numărătoarelor din majoritatea microcontrolerelor. De exemplu, pentru $T_{num} = 1\text{us}$, 77 ms înseamnă o valoarea a numărătorului de 77000 de pulsuri. Această valoarea înseamnă un numărător pe mai mult de 16 biți, așa că $T_{num} = 1\text{us}$ este prea mică.

Pe de altă parte, pentru ca măsurarea să fie precisă, perioada lui Tnum trebuie să fie de cel puțin 100 de ori mai mică decât Tx. Astfel se obține o precizie mai bună de 1%. Valoarea inferioară a lui Tx, și anume 6 ms, impune valoarea maximă $6\text{ms}/100=60\text{ us}$ pentru Tnum.

Oricare numărător (timer/counter) poate fi configurat să numere fie impulsuri interne (timer), fie impulsuri externe (counter). Cele trei numărătoare pot fi programate să funcționeze în 4 moduri: normal, CTC(Clear Timer on Compare Match), PWM rapid și PWM cu fază corectă.

În modul normal numărătorul 0 poate fi folosit pentru aplicațiile detaliate anterior: generarea de intervale de timp și numărarea evenimentelor. Există o categorie de aplicații care nu poate fi implementată corect în modul normal, cum ar fi divizorul de impulsuri pentru care se folosește metoda CTC.

Tehnica variată ieftină de umplere se numește PWM-Pulse With Modulation. Pe lângă controlul turației motoarelor de curent continuu, metoda este folosită și pentru controlul luminosității LED-urilor.

11.3 APLICAȚII

Enunț

Măsurare impuls fără întreruperi.

```
#include <inavr.h>
#include <iom16.h>

unsigned long nr_us;
unsigned int nr_cicli;
//măsurare impuls fără întrerupere
void main( void )
{
    TCCR1A=0;//initializare timer mod normal și dezactivare clock
    TCCR1B=0;
    DDRD&=~(1<<PD2);//pinul PD2 este de input

    while(PIND & (1<<PD2)==0);//așteptăm frontul pozitiv
```

```

//a apărut frontul pozitiv
TCCR1B |= (1<<CS12); //pornire clock timer cu prescaler de 256

while(PIND & (1<<PD2)==1); //așteptăm frontul negativ

//a apărut frontul negativ
nr_cicli=TCNT1; //salvăm numărul de ciclii

//calculam perioada în microsecunde
nr_us=nr_cicli/4*256; //nr_us=nrciclii* (perioada unui ciclu,
adică 0.25 microsecunde) * prescaler

//aici se poate trimite pe serială perioada.

return;
}

```

Măsurarea impulsului cu întreruperi.

```

#include <inavr.h>
#include <iom16.h>
unsigned int per;
unsigned int nr_cicli;
unsigned int flag_per;

#pragma vector=INT0
__interrupt void isr_INT0(void){
    if(PIND&(1<<PD2)==1){
        TCNT1=0; //reset timer
        TCCR1B=(1<<CS12); //start timer cu prescaler de 256
    }
    else{
        nr_cicli=TCNT1; //salvează nr cicli
        per=(nr_cicli/4)*256; //calculează perioada în microsecunde
        flag_per=1;
    }
}

int main( void )
{
    unsigned int x;

    // initializare întrerupere externă
    MCUCR |= (1<<ISC00);
    MCUCR &= ~(1<<ISC01);
    GICR |= (1<<INT0); //activare întrerupere externă
}

```

```

while(1){
    while(flag_per==0); //așteaptă măsurare întrerupere

    flag_per=0; //reseteză flag
    x=per; //se face ceva cu valoarea măsurată
}

```

Măsurare perioadei fără întrerupere

```

#include <inavr.h>
#include <iom16.h>

unsigned long nr_us;
unsigned int nr_cicli;
//măsurare perioada fără întrerupere
void main( void )
{
    TCCR1A=0; //initializare timer mod normal și dezactivare clock
    TCCR1B=0;
    DDRD&=~(1<<PD2); //pinul PD2 este de input

    while(PIND & (1<<PD2)==0); //așteptăm frontul pozitiv

    //a apărut frontul pozitiv
    TCCR1B|=(1<<CS12); //pornire clock timer cu prescaler de 256

    while(PIND & (1<<PD2)==1); //așteptăm frontul negativ

    //a apărut frontul negativ
    while(PIND & (1<<PD2)==0); //așteptăm frontul pozitiv pentru a completa perioada

    nr_cicli=TCNT1; //salvăm numărul de ciclii

    //calculam perioada în microsecunde
    nr_us=nr_cicli/4*256; //nr_us=nrciclii* (perioada unui ciclu, adică 0.25 microsecunde) * prescaler

    //aici se poate trimite pe serială perioada

    return;
}

```

12 Combinarea codului C și ASM în IAR

12.1 INTRODUCERE

Se va învăța modul de folosire a compilatorului pentru C pus la dispoziție de IAR pentru controlerele AVR în proiecte ce conțin atât cod C, cât și cod ASM. Combinând C și ASM se pot combina instrucțiunile puternice puse la dispoziție de limbajul C împreună cu eficacitatea instrucțiunilor scrise în ASM specifice pentru fiecare componentă în parte.

Assembly	C
<ul style="list-style-type: none"> +Control total al modului de utilizare a resurselor + Compact și foarte rapid în aplicații mici - Ineficient în aplicații de dimensiuni mari -Greu de înțeles -Greu de menținut -Nu e portabil 	<ul style="list-style-type: none"> +Eficient în aplicații mari +Cod structurat +Ușor de menținut +Ușor de portat -Control limitat al modului de utilizare a resurselor -Cod de dimensiuni mai mari și mai încet în aplicații mici

Figura 12.1 Plusurile și minusurile ale C și ASM

12.2 TRANSMITEREA DE VARIABILE ÎNTRE FUNCȚIILE SCRISE ÎN C ȘI FUNCȚIILE SCRISE ÎN ASM

Când compilatorul IAR este folosit împreună cu controlerele AVR, fișierele registru sunt segmentate precum în Figura 122.2.

Registrele Scratch sunt acele registre ce pot fi folosite de către modulele ASM, sau în alte obiecte externe fără a fi necesară salvarea conținutului.

Registrele Scratch nu sunt păstrate între apelurile de funcții, însă registrele locale sunt păstrate. Registrul Y (R28:R29) este folosit ca stack pointer pentru SRAM. Registrele Scratch sunt folosite pentru transmiterea parametrilor și a valorilor de return între funcții.

Când o funcție este apelată parametrii necesari vor fi salvați în Reșiștri R16-R23. Când o funcție returnează o valoare, această valoare va fi salvată în Reșiștri R16-R19 în funcție de dimensiunea parametrilor și a variabilei returnate.

Figura 122.3 arată câteva exemple de salvare a parametrilor, pentru diferite apele de funcții.

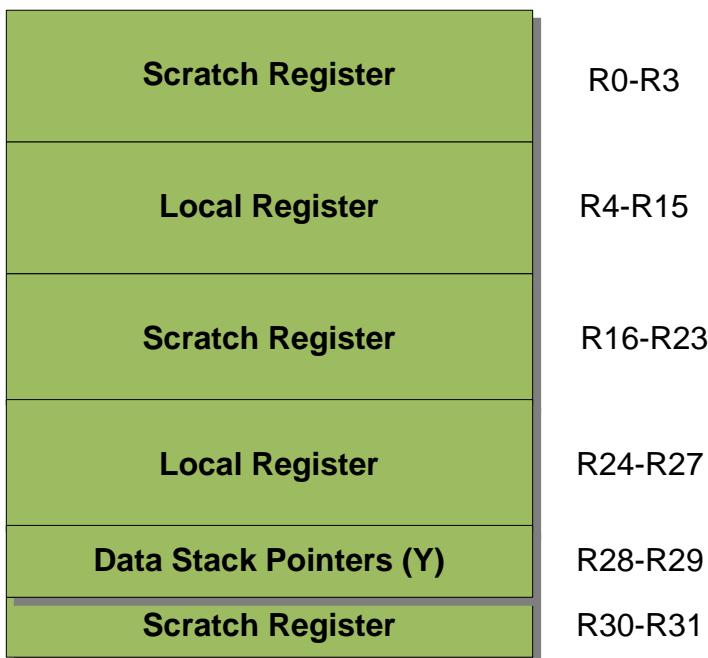


Figura 122.2 Fișierele Registru

Function	Parameter 1 Registers	Parameter 2 Registers
Func (char , char)	R16	R20
Func (char , int)	R16	R20, R21
Func (int , long)	R16,R17	R20, R21, R22 , R23
Func (long , long)	R16,R17,R18,R19	R20, R21, R22 , R23

Figura 122.3 Exemplu de salvare a parametrilor la apelul unei funcții

Un exemplu de apel de funcție în C :

```
int get_port (unsigned char temp , int num)
```

Când se apelează această funcție parametrul temp, pe 1 octet, este salvat în R16 , parametrul num, pe 2 octeți, este salvat în R20:R21 . Funcția returnează o valoare pe 2 octeți care este stocată în R16:R17 când se ieșe din funcție.

Dacă o funcție este chemată cu mai mult de 2 parametri, primii 2 parametri sunt trimiși către funcție, cum am arătat mai sus, iar următorii sunt transmiși funcției cu ajutorul stivei de date. Dacă o funcție este chemată cu o structură sau o uniune ca parametru, un pointer către structură sau uniune este trimis către funcție folosind stiva de date.

Dacă o funcție este nevoie să folosească un registru local, prima dată introduce valoarea din registru în stiva de date. Valoarea de return din funcție este salvată la adresa R16-R19, în funcție de dimensiunea valorii de return.

12.2.1 EXEMPLU DE APEL AL UNEI FUNCȚII ASM DINTR-UN PROGRAM C

12.2.1.1 Fără parametri și fără variabilă de return

Exemplu

Program scris în C cheamă o funcție scrisă în ASM

```
#include "io8515.h"
extern void get_port(void);/* Prototipul funcției ASM */
void main(void)
{
    DDRD = 0x00; /* Initializează porturile de I/O*/
    DDRB = 0xFF;
    while(1)/* Bucla infinită*/
    {
        get_port();/* Apelul funcției scrise în ASM */
    }
}
```

Codul ASM ce va fi chemat în modulul C

Function

NAME get_port

```
#include "io8515.h" ; Fișierul #include trebuie să fie în interiorul modulului
```

```
PUBLIC get_port ; Declararea simbolurilor ce vor fi chemate de funcția C
```

```
RSEG CODE ; Aceasta bucata de cod este relocată,
RSEG
```

```
get_port: ; Etichetă, aici începe execuția
```

```
in R16, PIND           ; Citește valoarea pinului D  
swap R16               ; Interschimbă valorile  
out PORTB, R16         ; Setează pe PORTB valoarea din R16  
ret                    ; Întoarcere în main  
END
```

12.2.1.2 Cu parametri și valoare de return

În exemplul de mai jos se apelează o funcție ASM. Variabila mask pe un octet este trimisă ca parametru către funcția ASM , mask este salvat în R16 înainte de apelul funcției . După terminarea funcției valoarea de return este salvată în R16 și apoi salvată în variabila value.

```
#include "io8515.h"  
char get_port(char mask); /*Prototipul funcției ASM */  
void C_task main(void)  
{  
    DDRB=0xFF  
    while(1) /* Buclă infinită*/  
    {  
        char value, temp; /*Declarare variabile locale */  
        temp = 0x0F;  
        value = get_port(temp); /* Apelul funcției scrise în ASM */  
        if(value==0x01)  
        {  
            /* Se intră în if */  
            PORTB=~(PORTB); /* Neagă valoarea din PORTB */  
        }  
    }  
}
```

Codul ASM ce va fi chemat în modulul C Function

NAME get_port

```
#include "io8515.h"          ; Fișierul #include trebuie să fie în interiorul modulului
```

```
PUBLIC get_port             ; Declararea simbolurilor ce vor fi chemate de funcția C
```

```
RSEG CODE                  ; Aceasta bucătă de cod este relocată, RSEG
```

```

get_port:                                ; Etichetă, aici începe execuția
    in R17, PIND                         ; Citește valoarea pinului D
    xor R16, R17                          ; XOR cu valoarea din mask(în R16) din
                                            ; main()
    swap R16                             ; Interschimbă valorile
    rol R16                             ; Rotește R16 la stânga
    brcc ret0                           ; Jump, dacă flagul carry e gol
    ldi r16,0x01                         ; Pune valoarea de 1 în R16
    ret                                ; Return
    ret0: clr R16                         ; Pune valoarea de 0 în R16
    ret                                ; Return
END

```

12.3 APELAREA DE FUNCȚII SCRISE ÎN C DE CĂTRE PROGRAME SCRISE ÎN ASM

În acest exemplu, programul scris în ASM va apela funcția de standard de librărie rand() pentru a primi un număr aleatoriu. Funcția rand() returnează un întreg pe 2 octeți. Exemplul de mai jos scrie doar primii 8 biți.

```

NAME get_port
#include "io8515.h"                      ; Fișierul #include trebuie să fie în
interiorul                                     ; modulului
EXTERN rand, max_val                         ; Simboluri externe ce vor fi folosite în
                                              ; funcție
PUBLIC get_port                               ; Declararea simbolurilor ce vor fi chemate
                                              ; de funcția C
RSEG CODE                                    ; Această bucată de cod este relocată, RSEG
get_port:                                     ; Etichetă, aici începe execuția
    clr R16                                 ; Clear R16
    sbis PIND, 0                            ; Testează dacă PIND0 e 0
    rcall rand                            ; Apelează RAND() dacă PIND0 este 0
    out PORTB,R16                          ; În PORTB salvează valoarea random
    lds R17,max_val                       ; Încarcă valoarea din main, max_val
    cp R17,R16                            ; Testează dacă valoarea e mai mare ca max_val
    brlt nostore                          ; Sari peste dacă e adevărat
    sts max_val,R16                        ; Salvează noua valoare

```

```

        ret          ; Return
END

```

12.4 FUNCȚII DEDICATE ÎNTRERUPERILOR ÎN ASM

În ASM se pot scrie și funcții dedicate îintreruperilor. Funcțiile dedicate îintreruperilor nu pot avea parametri și nu pot avea valoare de return. Fiindcă o îintrerupere poate avea loc în orice moment în timpul executării unui program este nevoie ca în momentul apariției să salveze valorile din toți regiștrii pe stivă.

Trebuie avută mare grijă când se folosește ASM-ul pentru a scrie astfel de funcții pentru a evita incompatibilități cu funcțiile scrise în C.

Exemplu de funcție ASM salvată în vectorul de îintreruperi:

```

NAME ext_int1
#include "io8515.h"
extern c_int1
COMMON INTVEC(1) ; Cod localizat în segmentul vectorilor de îintrerupere
ORG INT1_vect    ; Localizează codul în zona vectorilor de îintrerupere
RJMP c_int1       ; Jump la funcția în asm pentru îintrerupere
ENDMOD           ; Codul din vectorul de îintrerupere face un jump la
                  ; funcția c_int1 de mai jos:

NAME c_int1
#include "io8515.h"
PUBLIC c_int1      ; Declararea simbolurilor ce vor fi chemate de
                    ; funcția C
RSEG CODE          ; Această bucată de cod este relocată, RSEG
c_int1:
        st -Y,R16   ; Push regiștri pe stivă
        in R16,SREG  ; Citește status register
        st -Y,R16   ; Citește status register
        in R16,PIND  ; Salvează valoare din port D
        com R16      ; Inversare
        out PORTB,R16 ; Se salvează valoarea în port B
        ld R16,Y+     ; Pop status register
        out SREG,R16  ; Salvează status register
        ld R16,Y+     ; Pop Register R16
        reti
END

```

12.5 ACCESAREA DE VARIABILE GLOBALE ÎN INTERIORUL MODULELOR ASM

Dacă în programul principal există variabile globale spre exemplu max_val pentru a accesa această variabilă în modulul ASM, variabila trebuie declarată ca EXTERN max_val. Pentru a accesa această variabilă modulul ASM folosește instrucțiunile LDS(Load direct from SRAM) și STS(Store direct to SRAM).

```

#include "io8515.h"
char max_val ;
void get_port(void) ; /* Prototipul funcției ASM */
void C_task main(void)
{
    DDRB = 0xFF; /* Setează ca output port B */
    while(1) /* Buclă infinită */
    {
        get_port(); /* Apeleză funcția ASM */
    }
}

NAME get_port
#include "io8515.h"      ; Fișierul #include trebuie sa fie in interiorul
                        ; modulului
EXTERN rand, max_val   ; Simboluri externe ce vor fi folosite în funcție
PUBLIC get_port         ; Simboluri externe ce vor fi folosite în funcție
RSEG CODE               ; Această bucată de cod este relocată, RSEG
get_port:               ; Etichetă, aici începe execuția
    clr R16              ; Clear R16
    sbis PIND,0           ; Testează dacă PIND0 este 0
    rcall rand             ; Apeleză RAND() dacă PIND0 = 0
    out PORTB,R16          ; Setează valoarea de output în PORTB
    lds R17, max_val       ; Încarcă valoarea max_val din main
    cp R17, R16             ; Verifică dacă numărul este mai mare ca max_val
    brlt nostore            ; Sari peste dacă nu este mai mare ca max_val
    sts max_val, R16         ; Salvează noua valoare dacă este mai mare
    nostore:
    ret                     ; Return
END

```

12.6 PARALELE ÎNTRE C ȘI ASM

Acest exemplu setează pinii portului B astfel: pinii 0 și 1 pe high iar pinii 2 și 3 pe low și definește porturile de la 4 la 7 ca input:

```

ldi r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)      unsigned char i;
                                                       ...
ldir17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)    PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
                                                       ...
out PORTB,r16                                         DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
                                                       ...
out DDRB,r17                                         /* Apeleză nop pentru sincronizare */
                                                       ...
; Apeleză nop pentru sincronizare
nop
; Citește valorile din pini
in r16, PINB                                         /* Citește valorile din pini */
                                                       i = PINB;

```

În exemplul de mai jos se arată modul de accesare a registrului timer-ului pe 16 biți:

```

TIM16_ReadTCNT1:
; Salvează flagul global de întreruperi
    in r18,SREG
; Dezactivează întreruperile
    cli
; Read TCNT1 into r17:r16
    in r16,TCNT1L
    in r17,TCNT1H
; Restaurează flagul de întreruperi
    out SREG,r18
ret

```

```

unsigned int TIM16_ReadTCNT1( void )
{
    unsigned char sreg;
    unsigned int i;
/* Salvează flagul global de
   întreruperi */
    sreg = SREG;
/* Dezactivează întreruperile */
    _CLI();
/* Citește valoarea din TCNT1 în i */
    i = TCNT1;
/* Restaurează flagul de întreruperi */
    SREG = sreg;
    return i;
}

```

Acest exemplu arată cum să inițializăm SPI-ul ca master și cum să realizăm o simplă transmisie:

```

SPI_MasterInit:
; Setează MOSI și SCK ca output, și restul ca input
    ldi r17,(1<<DD_MOSI)|(1<<DD_SCK)
    out DDR_SPI,r17
; Activează SPI, ca Master, setează clock rate ca fck/16
    ldi r17,(1<<SPE)|(1<<MSTR)|(1<<SPR0)
    out SPCR,r17
ret

```

```

SPI_MasterTransmit:
; Începe transmisia de date (r16)
    out SPDR,r16
Wait_Transmit:
; Așteaptă ca transmisia să fie completă
    sbis SPSR,SPIF
    rjmp Wait_Transmit
ret

```

```

void SPI_MasterInit(void)
{
/* Setează MOSI și SCK ca output, și
   restul ca input */
    DDR_SPI      = (1<<DD_MOSI)
|(1<<DD_SCK);
/* Activează SPI, ca Master, setează
   clock rate ca fck/16 */
    SPCR        = (1<<SPE)|(1<<MSTR)
|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
/* Începe transmisia de date */
    SPDR = cData;
/* Așteaptă ca transmisia să fie
   completă */
    while(!(SPSR & (1<<SPIF)))
        ;
}

```

Acest exemplu arată cum să inițializăm SPI-ul ca slave și cum să realizăm o simplă recepție:

```

SPI_SlaveInit:
; Setează MISO ca output, și restul ca input
    ldi r17,(1<<DD_MISO)
    out DDR_SPI,r17
; Enable SPI
    ldi r17,(1<<SPE)

```

```

void SPI_SlaveInit(void)
{
/* Setează MISO ca output, și restul ca
   input */
    DDR_SPI = (1<<DD_MISO);
/* Enable SPI */

```

```

out SPCR,r17                               SPCR = (1<<SPE);
ret                                         }

SPI_SlaveReceive:
;Așteaptă ca receptia să fie completă
    sbis SPSR,SPIF
    rjmp SPI_SlaveReceive
;Citește valorile recepționate
    in r16,SPDR
ret

char SPI_SlaveReceive(void)
{
/* Așteaptă ca receptia să fie completă */
/*
    while (!(SPSR & (1<<SPIF)))
;
/* Citește valorile recepționate */
    return SPDR;
}

```

Acest exemplu ne arată cum să inițializăm USART-ul:

```

USART_Init:
;Setează baud rate
    out UBRRH, r17
    out UBRRL, r16
;Enable receiver și transmitter
    ldi r16, (1<<RXEN)|(1<<TXEN)
    out UCSRB,r16
;Setează formatul frame-ului ca : 8data, 2stop bit
    ldi r16, (1<<URSEL)|(1<<USBS)|(3<<UCSZ0)
    out UCSRC,r16
ret

#define FOSC 1843200// Clock Speed
#define BAUD 9600
#define MYUBRR FOSC/16/BAUD-1
void main( void )
{
...
    USART_Init ( MYUBRR );
...
}
void USART_Init( unsigned int ubrr )
{
/* Setează baud rate-ul */
    UBRRH = ( unsigned
char )(ubrr>>8);
    UBRRL = ( unsigned char )ubrr;
/* Activează receiver și transmitter */
    UCSRB = (1<<RXEN)|(1<<TXEN);
/* Setează formatul frame-ului ca :
8data, 2stop bit */
    UCSRC = (1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
}

```

Acest exemplu ne arată cum să transmitem un mesaj simplu folosind USART:

```

USART_Transmit:
;Așteaptă ca buffer-ul de transmisie să fie gol
    sbis UCSRA,UDRE
    rjmp USART_Transmit
;Copie al 9-lea bit din r17 în TXB8
    cbi UCSRB,TXB8
    sbrc r17,0
    sbi UCSRB,TXB8
;Salvează LSB data (r16) în buffer, trimite datele
    out UDR,r16

void USART_Transmit( unsigned int data
)
{
/* Așteaptă ca buffer-ul de transmisie
să fie gol */
    while ( !( UCSRA & (1<<UDRE)));
/* Copie al 9-lea bit din r17 în TXB8*/
    UCSRB &= ~(1<<TXB8);
if ( data & 0x0100 )
    UCSRB |= (1<<TXB8);
}

```

ret

```
/* Salvează LSB data (r16) in buffer,
trimite datele */
UDR = data;
}
```

Acest exemplu ne arată cum să receptionăm un mesaj simplu folosind USART:

USART_Receive:

```
; Așteaptă ca buffer-ul de recepție să fie gol
    sbis UCSRA, RXC
    rjmp USART_Receive
; Salvează statusul și al 9-lea bit, apoi datele din buffer
    in r18, UCSRA
    in r17, UCSRB
    in r16, UDR
; În caz de eroare, return -1
    andi r18,(1<<FE)|(1<<DOR)|(1<<PE)
    breq USART_ReceiveNoError
    ldi r17, HIGH(-1)
    ldi r16, LOW(-1)
USART_ReceiveNoError:
; Filtrează al 9-lea bit, apoi return
    lsr r17
    andi r17, 0x01
ret
```

```
unsigned int USART_Receive( void )
{
    unsigned char status, resh,
    resl;
    /* Așteaptă ca buffer-ul de recepție să fie gol */
    while ( !(UCSRA & (1<<RXC)) );
    /* Salvează statusul și al 9-lea bit,
    apoi datele din buffer */
    status = UCSRA;
    resh = UCSRB;
    resl = UDR;
    /* În caz de eroare, return -1*/
    if ( (status & (1<<FE)|(1<<DOR)|(1<<PE) ) )
        return -1;
    /* Filtrează al 9-lea bit, apoi return */
    resh = (resh >> 1) & 0x01;
    return ((resh << 8) | resl);
}
```

Acest exemplu ne arată cum să golim buffer-ul de recepție:

USART_Flush:

```
    sbis UCSRA, RXC
    ret
    in r16, UDR
rjmp USART_Flush
```

```
void USART_Flush( void )
{
    unsigned char dummy;
    while ( UCSRA & (1<<RXC) )
        dummy = UDR;
}
```

Acest exemplu ne arată cum să accesăm registrul UCSRC:

USART_ReadUCSRC:

```
; Citește UCSRC
    in r16,UBRRH
    in r16,UCSRC
ret
```

```
unsigned char USART_ReadUCSRC( void )
{
    unsigned char ucsrc;
    /* Citește UCSRC */
    ucsrc = UBRRH;
    ucsrc = UCSRC;
    return ucsrc;
}
```

13 Microcontrolere AVR

13.1 CARACTERISTICI

Microcontroler-ele AVR pe 8 biți (Atmel) au la bază un nucleu RISC cu arhitectură Harvard. Aceste microcontrolere sunt destinate aplicațiilor simple: controlul motoarelor, controlul fluxului de informație pe portul USB, aplicații din domeniul automotive, controlul accesului de la distanță (Remote Access Control), și.a.. Pe baza acestui nucleu firma Atmel a dezvoltat mai multe familii de microcontrolere, cu diferite structuri de memorie și de interfețe I/O, destinate diferitelor clase de aplicații.

Familia de procesoare AVR pe 8 biți are următoarele caracteristici:

- arhitectură RISC;
- 32 de registre de lucru de 8 biți;
- multiplicator hardware;
- au o arhitectură Harvard modificată (există memorii și bus-uri separate pentru program și date). Există un nivel de *pipeline* – în timp ce o instrucțiune este executată, instrucțiunea următoare este adusă din memorie;
- frecvență de lucru de 0 - 16 MHz;
- procesoarele sunt prevăzute cu o gama largă de dispozitive I/O și de periferice încorporate;
- timer programabil cu circuit de prescalare;
- surse de întrerupere interne și externe;
- timer de urmărire (watchdog) cu oscilator independent;
- interfață JTAG (standardul IEEE 1149.1 Compliant);
- 6 moduri de operare SLEEP și POWER DOWN pentru economisirea energiei;
- oscilator integrat RC;
- densitate mare a codului și compatibilitate integrală a codului între membrii familiei;
- procesoarele sunt disponibile în capsule variate, de la circuite cu 8 pini la procesoare cu 68 de pini;
- familia AVR beneficiază de existența unui set unitar de instrumente software pentru dezvoltarea aplicațiilor.

Microcontrolerelor din familia AVR folosesc o arhitectură RISC care permite execuția celor mai multe instrucțiuni într-un singur ciclu de tact, ceea ce duce la îmbunătățirea performanței de 10 ori față de procesoarele convenționale (de exemplu, Intel 8051) care operează la aceeași frecvență.

13.2 FAMILIA AVR

Exemplu este microcontrolerul de 8 biți ATmega16, realizat în tehnologie CMOS, bazat pe arhitectura RISC AVR îmbunătățită.

Procesorul dispune de un set de 131 instrucțiuni și 32 de registre de uz general. Cele 32 de registre sunt direct adresabile de unitatea aritmetică și logică (ALU). O instrucțiune poate accesa două registre independente.

Caracteristicile procesorului ATmega16 sunt:

- 16 Kb de memorie Flash reînscriptibilă pentru stocarea programelor;
- 1 Kb de memorie RAM pentru date;
- 512 bytes de memorie EEPROM pentru constante;
- dispozitive periferice și porturi I/O:
 - două numărătoare/temporizatoare de 8 biți;
 - un numărător/temporizator de 16 biți;
 - un convertor analog/digital de 10 biți, cu 8 intrări;
 - 4 canale PWM;
 - un comparator analogic;
 - timer de urmărire (watchdog) cu oscilator propriu;
 - conține 3 interfețe pentru comunicație: USART pentru comunicație serială (port serial), interfață serială TWI, interfață serială SPI;
 - ceas de timp real cu oscilator separat;
 - 32 de linii I/O organizate în patru porturi (PA, PB, PC, PD).

Structura internă generală a microcontroler-ului este prezentată în **Error!**

Reference source not found. Se poate observa că există o magistrală generală de date la care sunt conectate mai multe module:

- unitatea aritmetică și logică (ALU);
- registrele generale;
- memoria RAM și memoria EEPROM;
- porturile I/O de uz general și celelalte blocuri de intrare/iesire. Aceste module sunt controlate de un set special de registre din spațiul I/O, fiecare modul având asociat un număr de registre specifice.

Memoria flash de program împreună cu întreg blocul de extragere a instrucțiunilor, decodare și execuție va comunica printr-o magistrală proprie, separată de magistrala de date. Acest tip de organizare este specific arhitecturii Harvard și permite controllerului să execute instrucțiunile foarte rapid.

În modul *power-down* procesorul salvează conținutul regisrelor, dar blochează oscilatorul, dezactivând toate celelalte funcții al chip-ului până la următoarea întrerupere externă sau la activarea intrării de inițializare hardware (Reset). În modul power-save, timer-ul asincron continuă să funcționeze, permitând utilizatorului să mențină o bază de timp în timp ce restul dispozitivului este oprit.

În modul *standby*, oscilatorul funcționează în timp ce restul dispozitivului este oprit. Acest lucru permite un start foarte rapid combinat cu un consum redus de energie. În modul standby extins (Extended Stand-by Mode), atât oscilatorul principal cât și timer-ul asincron continuă să funcționeze.

Memoria flash internă poate fi reprogramată printr-o interfață serială SPI, de către un programator de memorie nonvolatilă convențional, sau de către un program de pornire rezident (on-chip) ce rulează pe nucleul AVR. Acest program poate folosi orice interfață pentru a încărca programul de aplicație în memoria flash.

Combinând un nucleu CPU-RISC de 8 biți cu o memorie flash într-un singur chip, ATmega 16 este un microcontroler puternic ce oferă o soluție extrem de flexibilă și cu un cost redus. În plus ATmega16 AVR este susținut de o serie completă de instrumente de programare și de dezvoltare a sistemului, care include: compilatoare C, macroasamblare, programe de depanare și simulare, etc.

13.2.1 ATMEGA 16 - DESCRIEREA PINILOR

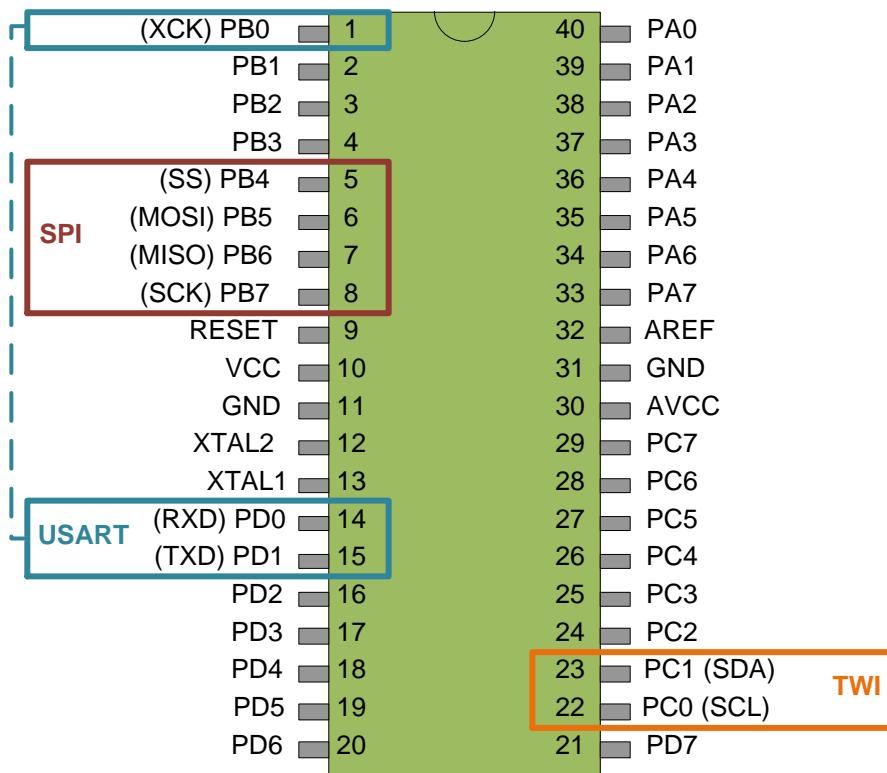


Figura 13.1

VCC – Sursa de alimentare

GND – Masa

Portul A (PA7÷PA0)

Portul A este un port bidirectional de 8 biți programabil. Liniile portului A sunt folosite și ca intrări analogice pentru convertorul A/D. Liniile portului pot fi conectate optional la VCC prin rezistențe de agățare (pull-up resistor), selectate pentru fiecare linie. Buffer-ele de ieșire ale portului A au caracteristici de amplificare.

Portul B (PB7÷PB0)

Portul B este un port I/O de 8 biți bidirectional prevăzut cu rezistențe de agățare interne (optional). Buffer-ele de ieșire ale portului B au caracteristici de amplificare. Portul B îndeplinește de asemenea funcții speciale ale microcontrolerului ATmega16.

Portul C (PC7÷PC0)

Portul C este un port I/O de 8 biți bidirectional cu prevăzut cu rezistențe de agățare interne (optional). Buffer-ele de ieșire ale portului C au caracteristici de amplificare. Dacă interfața JTAG (de depanare) este activată, rezistențele pinilor PC5(TDI), PC3(TMS) și PC2(TCK) vor fi activate, chiar dacă are loc o resetare. Portul C îndeplinește de asemenea funcții ale interfeței JTAG și alte funcții speciale ale ATmega16.

Port D (PD7...PD0)

Portul D este un port I/O de 8 biți bidirecțional prevăzut cu rezistente de agățare interne (optional). Buffer-ele de ieșire ale portului D au caracteristici de amplificare. Portul D îndeplinește de asemenea funcții speciale ale ATmega16.

Reset

Un nivel scăzut la acest pin mai mare ca durată decât o valoare prestabilită, va provoca inițializarea procesorului.

XTAL1 și XTAL2. Intrarea și respectiv ieșirea amplificatorului inversor al oscilatorului generatorului de tact.

AVCC este pinul de alimentare pentru portul A și pentru convertorul A/D. Trebuie conectat extern la Vcc chiar dacă ADC nu este folosit. Dacă ADC este folosit, trebuie conectat la Vcc printr-un filtru trece-jos.

AREF este pinul de intrare pentru referință analogică a convertorului A/D.

13.2.2 UNITATEA CENTRALĂ DE PRELUCRARE (CPU)

Arhitectura nucleului AVR este prezentată în Figura 13.2. Funcția principală a nucleului CPU-AVR este aceea de a asigura execuția corectă a programului. Pentru aceasta, nucleul este capabil să acceseze memorile, să execute calcule, să controleze perifericele și să prelucreze întreruperile.

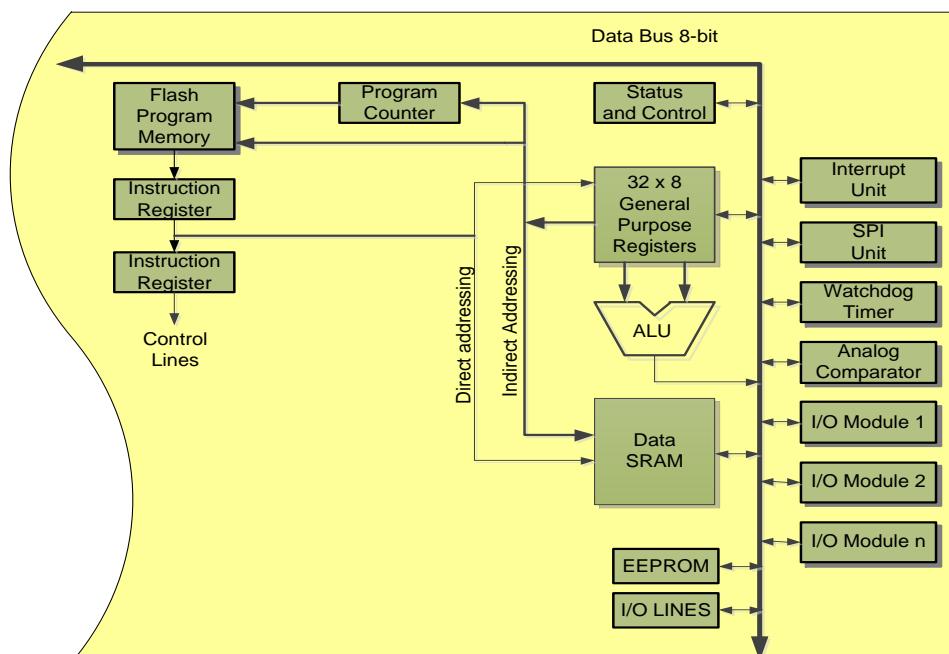


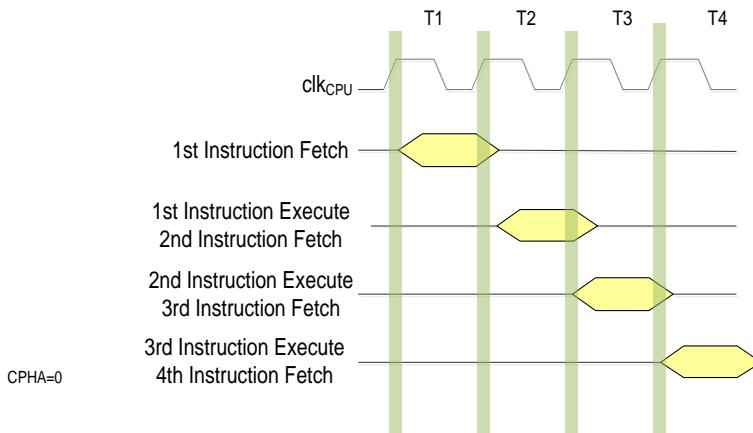
Figura 13.2 Schema bloc a nucleului CPU-AVR

13.2.3 ALU (UNITATEA ARITMETICĂ ȘI LOGICĂ)

Unitatea aritmetică și logică execută operațiile de prelucrarea a datelor. ALU-AVR lucrează direct cu cele 32 de registre. Operațiile pe care le execută unitatea ALU sunt împărțite în trei categorii: aritmetice, logice și operații pe bit. Unele implementări ale arhitecturii AVR pot efectua și multiplicări de operanzi cu sau fără semn.

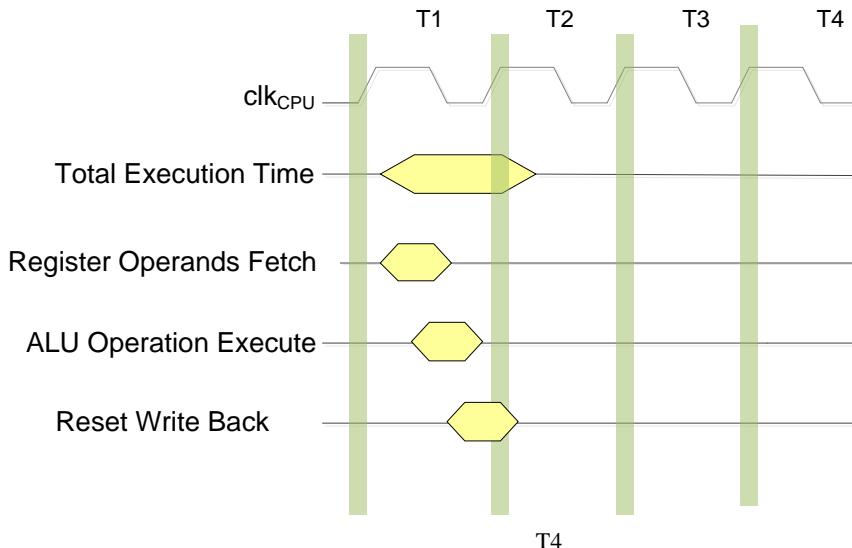
13.3 EXECUȚIA INSTRUCȚIUNILOR

Pentru obținerea unei performanțe bune CPU-AVR combină avantajul arhitecturii Harvard a spațiului de memorie și accesul rapid la registre cu execuția pipeline. Procesorul folosește un pipeline cu două etaje, unul pentru extragerea instrucțiunilor și altul pentru execuție. În timp ce o instrucțiune este executată, o nouă instrucțiune este extrasă din memorie. În felul acesta, procesorul furnizează un rezultat la fiecare tact, chiar dacă o instrucțiune durează mai mult de un tact. Această secvențiere este prezentată în Figura 13.3.

**Figura 13.3** Funcționarea pipeline

În acest mod, performanța nucleului se apropie de 1 MIPS/MHz și oferă cele mai bune rezultate din punct de vedere al raportului funcționi/cost, funcționi/timp și funcționi/unitate.

În **Error! Reference source not found.** este prezentat modul de folosire a registrelor. Într-un singur ciclu de tact ALU folosește 2 registre pentru a executa o operație, iar rezultatul este stocat înapoi în registrul de destinație.



Figură 13.4 Operații cu registre executate de ALU într-un singur ciclu de tact

Instrucțiunile care accesează memoria necesită doi cicli de tact pentru o operație după cum se vede din *Figura 13.5*.

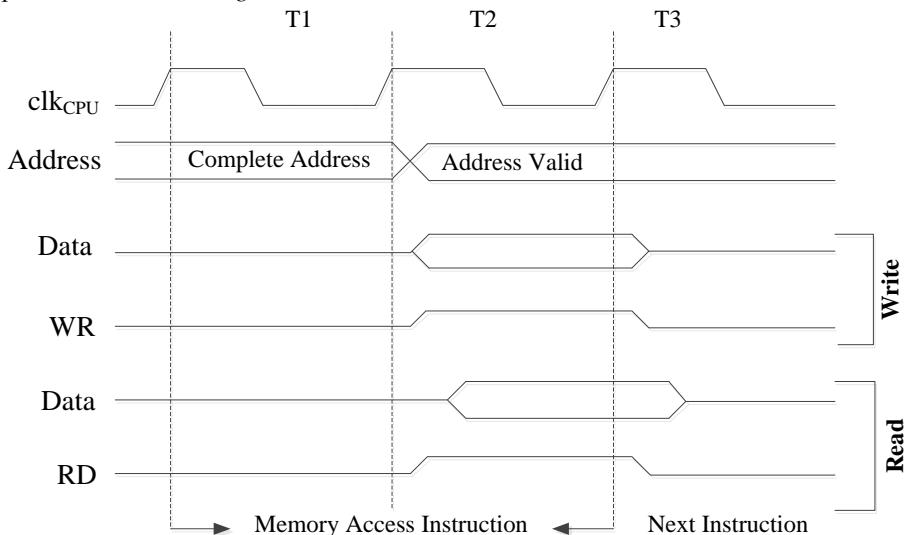


Figura 13.5 Execuția instrucțiunilor care accesează memoria SRAM

13.4 TRATAREA ÎNTRERUPERILOR ȘI INITIALIZARE

Nucleul AVR folosește mai multe tipuri de întreruperi. Întreruperile sunt vectorizate; vectorii de întrerupere și cel corespunzător inițializării procesorului reprezintă adrese ale unor locații din memoria de program. Întreruperile pot fi activate/dezactivate global (cu ajutorul bitului I din registrul de stare) sau individual. Sursele de întrerupere și vectorii corespunzători sunt prezenți în tabelul următor:

Nr. vectorului	Adresa din program	Sursa	Definirea întreruperii
1	\$000	Reset	Pin extern, Reset la pornire, Reset Watchdog, JTAG AVR Reset
2	\$002	EXT_INT0	External IRQ0
3	\$004	EXT_INT1	External IRQ1
4	\$006	TIM2_COMP	Comparare Timer2
5	\$008	TIM2_OVF	Timer2 overflow
6	\$00A	TIM1_CAPT	Capturare eveniment Timer1
7	\$00C	TIM1_COMPA	Comparare Timer1
8	\$00E	TIM1_COMPB	Comparare Timer1
9	\$010	TIM1_OVF	Timer 1 overflow
10	\$012	TIM0_OVF	Timer 0 overflow
11	\$014	SPI_STC	Transfer compet (SPI)
12	\$016	UART_RXC	UART RX complet
13	\$018	UART_DRE	UART UDR empty
14	\$01A	UART_TXC	UART TX complet
15	\$01C	ADC	Conversie ADC terminată
16	\$01E	EE_RDY	EEPROM ready
17	\$020	ANA_COMP	Comparator analogic
18	\$022	TWI	Interfață serială cu 2 fire
19	\$024	INT2	Întrerupere externă IRQ2
20	\$026	TIM0_COMP	Comparare Timer0
21	\$028	SPM_RDY	Încarcarea programului

Pentru vectorii de întrerupere sunt rezervate adresele inferioare din memoria de program. Prioritatea diferitelor surse este dată de poziția vectorului în harta de memorie. Cu cât sursa de întrerupere are vectorul mai mic cu atât prioritatea ei este mai mare.

RESET-ul are cea mai mare prioritate. Vectorii de întrerupere pot fi mutați la începutul secțiunii Boot Flash prin setarea bitului IVSEL din registrul global de control al întreruperilor (GICR). Vectorul de RESET poate fi de asemenea mutat la începutul aceleiași secțiuni prin programarea secțiunii BOOTRST.

Când apare o întrerupere bitul I din registrul de stare este resetat și toate întreruperile sunt invalidate. În cadrul subruteinei de tratare utilizatorul poate reactiva sistemul de întreruperi. Dacă sistemul de întreruperi este activat subrutina de tratare poate fi la rândul ei întreruptă de o sursă de întrerupere cu prioritate mai mare. Bitul I este automat setat când este executată instrucțiunea RETI.

Întreruperile externe pot fi controlate individual cu ajutorul registrului GIMSK (adresa 3B în spațiul I/O). Fiecare bit din acest regisztr corespunde unei întreruperi. Bitul 7 controlează INT1, iar bitul 6 controlează INT0. Setarea bitului corespunzător activează întreruperea. Registrul GIMSK poate fi citit/scris cu instrucțiunile IN/OUT. Starea întreruperilor este indicată de registrul GIFR (adresa 3A în spațiul I/O). Dacă o întrerupere externă a fost activată este setat bitul corespunzător din GIFR.

AVR are o structură complexă a sistemului de întreruperi. Aproape toate dispozitivele periferice au fost dotate cu capacitați de întrerupere, motiv pentru care programul principal nu trebuie să verifice periodic starea acestor dispozitive.

Secvența de acțiuni care se derulează când apare o întrerupere este următoarea:

1. Dispozitivul periferic emite o cerere de întrerupere care întrerupe procesorul;
2. Execuția instrucțiunii curente este finalizată;
3. Adresa următoarei instrucțiuni din programul curent este memorată în stivă;
4. Este încărcată în PC adresa subruteinei de tratare asociată;
5. Procesorul execută subrutina de tratare;
6. Subrutina se încheie cu instrucțiunea RETI (Return from Interrupt);
7. Procesorul reia execuția programului întrerupt de la adresa memorată în stivă.

Un factor important de luat în seamă când sunt folosite întreruperile, este cât de repede poate răspunde un procesor la o întrerupere. Aceasta depinde de arhitectura procesorului. Pentru controlerle AVR, răspunsul la întreruperi se face în minim 4 cicli de tact. În timpul celor 4 cicli, este salvat în stiva PC (2 bytes), iar SP este decrementat cu 2.

Transferul execuției la rutina de întrerupere necesită doi cicli. Dacă o întrerupere apare în timpul unei instrucțiuni multiciclu, această instrucțiune este finalizată înainte de a fi servită întreruperea. Revenirea dintr-o subrutină de tratare durează alți 4 cicli. În timpul acestor 4 cicli, este restaurată starea PC salvată în stivă (2 bytes), iar SP este incrementat cu 2. În același timp este setat bitul I din SREG.

13.5 MEMORIA

Memoria de program este de tip Flash și are o mărime de 256K. Este organizată pe 16 biți (128K x 16) deoarece instrucțiunile se codifică pe 2 sau 4 octeți.

Nu se poate adăuga memorie de program externă.

Din considerente de securitate *software*, spațiul memoriei de program este împărțit în 2 regiuni (*Figura 13.6*):

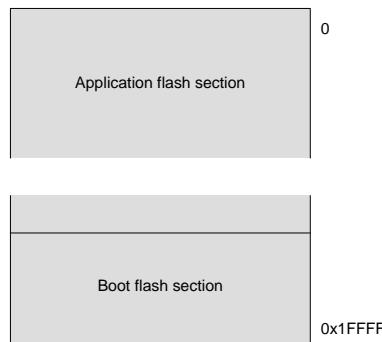


Figura 13.6 Regiunile memoriei de program

Memoria de program suportă minim 10000 de cicli de ștergere-scriere. Un astfel de ciclu reprezintă de fapt reprogramarea microcontrolerului și se întâmplă relativ rar (cel mai adesea în etapa de dezvoltare, pe echipamentele de test).

Adresa instrucțiunii ce urmează a fi executată este întotdeauna conținută în registrul PC (*Program Counter*). Modificarea registrului PC poate fi efectuată cu o instrucțiune de salt sau apel.

13.5.1 ADRESAREA MEMORIEI DE PROGRAM

Memoria de program este adresabilă liniar. În cazul altor microcontrolere memoria poate fi, de exemplu, segmentată, ceea ce implică o adresare prin intermediul registrelor de segment (acest lucru poate reprezenta o facilitate în cazul unei aplicații cu multiple fire de execuție).

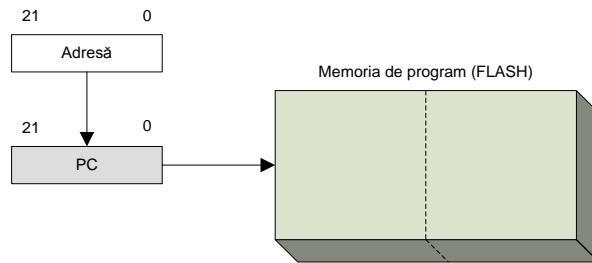
Pentru a scrie un program în limbajul C nu este nevoie cunoașterea modurilor de adresare a memoriei (nu contează de ce tip). Înțelegerea modurilor de adresare este însă necesară pentru a scrie cod sursă în limbaj de asamblare (ceea ce se întâmplă adeseori în dezvoltarea sistemelor *embedded*).

Există 2 clase de instrucțiuni ce pot referi memoria de cod:

- instrucțiunile de salt sau apel – adresa efectivă se calculează în cuvinte (pentru că o instrucțiune se codifică pe 2 sau 4 octeți);
- instrucțiunile de scriere și citire a memoriei de cod – adresa efectivă se calculează în octeți (pentru că există 2 operanzi – sursă și destinație – unul dintre care este unul sau doi registri cu funcție generală).

Instrucțiunile de salt sau apel pot adresa memoria de program în 3 moduri:

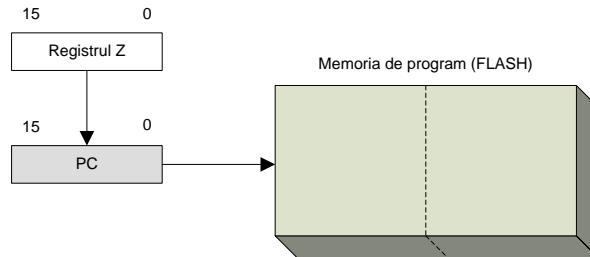
1. **direct** – adresa efectivă este operandul instrucțiunii. Acest mod de adresare este folosit de instrucțiunile JMP și CALL;



Exemplu:

```
jmp farp
...
farp:    nop
```

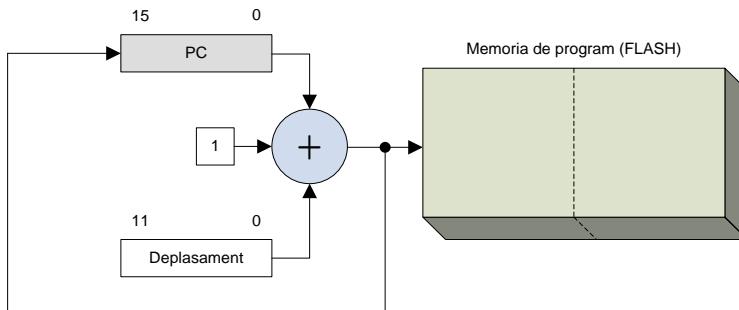
2. **indirect** – adresa efectivă este conținutul registrului Z. Acest mod de adresare este folosit de instrucțiunile IJMP și ICALL;



Exemplu:

```
ijmp      ; salt la rutina indicată în r31:r30
```

3. **relativ** – adresa efectivă este suma dintre valoarea curentă a registrului PC, conținutul registrului Z și 1. Acest mod de adresare este folosit de instrucțiunile RJMP și RCALL.

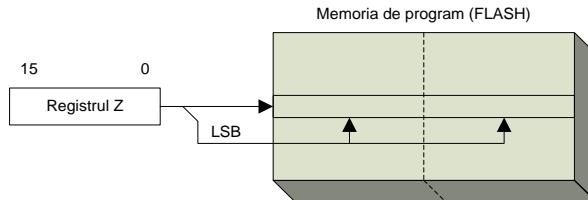


Exemplu:

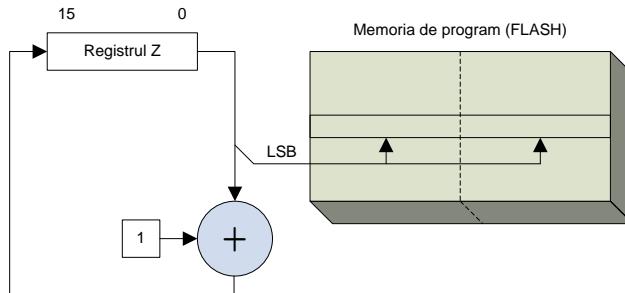
```
rjmp error
...
error:    nop
```

Instrucțiunile de scriere și citire a memoriei de program pot utiliza 2 moduri de adresare:

1. **indirect** – adresa efectivă este conținutul registrului Z. Acest mod de adresare este folosit de instrucțiunile LPM, ELPMP, și SPM;



2. **indirect cu post-decrement** – adresa efectivă este conținutul registrului Z, fiind incrementată după citirea memoriei. Acest mod de adresare este folosit de instrucțiunile LPM și ELPMP.



În memoria de program pot fi păstrate variabilele constante.

13.5.2 MEMORIA DE DATE

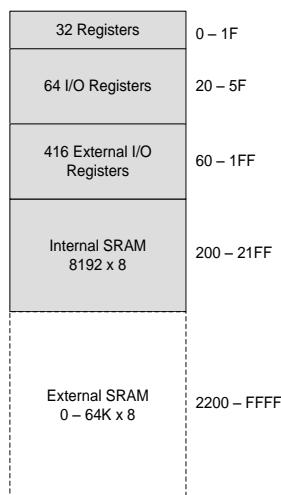
Există 2 tipuri de memorie de date: SRAM și EEPROM.

➤ **SRAM**

SRAM-ul este un tip de memorie foarte rapidă și relativ scumpă, celula de bază fiind construită din tranzistori.

ATmega2560 dispune de 8K de SRAM.

Spațiul din SRAM este organizat în mai multe regiuni (**Error! Reference source not found.**):

**Figura 13.7 Organizarea SRAM-ului**

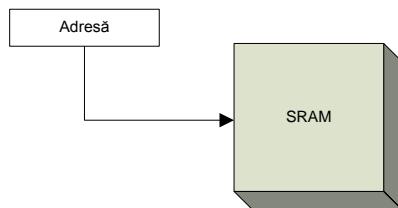
Instrucțiunile de scriere/citire pe SRAM, de tipul *load/store*, se execută în minim 2 cicli mașină. Pentru anumite porțiuni din SRAM însă, există instrucțiuni specializate, ce se execută într-un ciclu mașină:

- registrele cu funcție generală pot fi adresate cu instrucțiunile MOV (operează pe un octet) și MOVW (operează pe cuvânt)
- registrele I/O pot fi adresate cu instrucțiunile IN și OUT

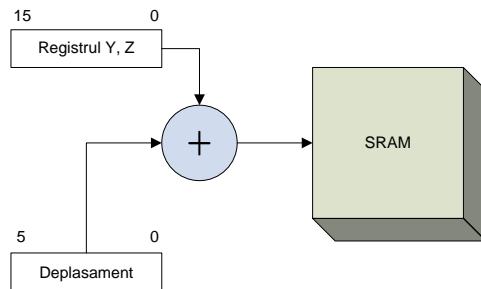
SRAM-ul extern reprezintă de fapt spațiul de adresare disponibil pentru a conecta la microcontroler o memorie externă de tip SRAM, Flash sau chiar niște periferice.

Există 5 moduri de adresare:

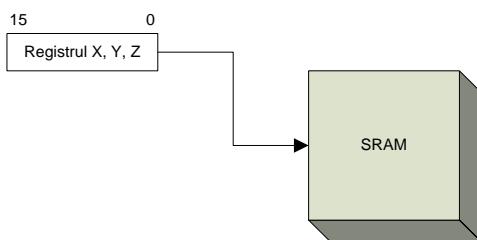
1. direct



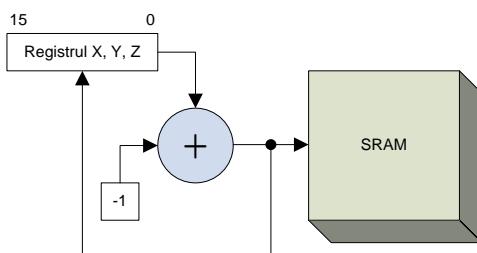
2. indirect cu deplasament



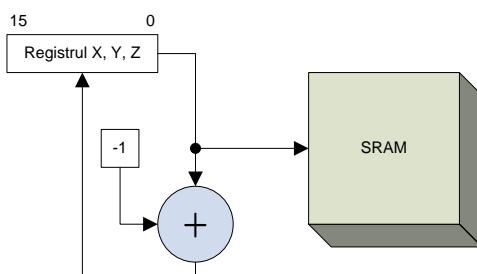
3. indirect



4. indirect cu pre-decrementare



5. indirect cu post-decrementare



Registrele mapate în SRAM (adresele mici) au scheme de adresare optimizate.

13.5.2.1 EEPROM

EEPROM-ul (4K x 8) este un tip de memorie nevolatilă, ce suportă minim 100000 cicli de scriere. De fapt, această memorie este văzută ca un periferic, care poate fi adresat prin 3 registre: de control (EECR), de date (EEDR), de adresă (EEAR).

Operațiile de scriere și citire pot fi implementate în limbajul C sub forma unor proceduri, după cum urmează:

```
void EEPROM_write(unsigned int uiAddress, unsigned char ucData)
```

```
{
```

```
    /* Așteaptă completarea scrierii anterioare */
    while(EECR & (1<<EEPE))
        ;
    /* Setează adresa și regisztrul triei de date */
    EEAR = uiAddress;
    EEDR = ucData;
    /* Scrie 1 logic în EEMPE */
    EECR |= (1<<EEMPE);
    /* Pornește scrierea în eeprom, odată cu EEPE */
    EECR |= (1<<EEPE);
```

```
}
```

```
unsigned char EEPROM_read(unsigned int uiAddress)
```

```
{
```

```
    /* Așteaptă completarea scrierii anterioare */
    while(EECR & (1<<EEPE))
        ;
    /* Setează adresa registrului */
    EEAR = uiAddress;
    /* Pornește citirea eeprom, odată cu EERE */
    EECR |= (1<<EERE);
    /* Returnează registrul de date */
    return EEDR;
```

```
}
```

Funcția de scriere a unui octet în EEPROM poate fi optimizată astfel încât să mărească durata de viață a EEPROM-ului – deoarece numărul de cicli de scriere este limitat, are sens ca:

- înainte de a efectua o operație de scriere, să se compare valoarea de scris cu cea existentă în memorie;
- gradul de utilizare a tuturor locațiilor să fie cât mai uniform.

Procesul de scriere a unei locații de memorie în EEPROM durează un timp relativ îndelungat. Operația de citire a unei locații în curs de scriere va returna valoarea

0xFF. Din acest motiv, pentru a efectua o citire corectă a unei locații de memorie ce a fost anterior scrisă, se va aștepta minim $t_{WD_EEPROM} = 9.0$ ms după scrierea ei.

13.6 POINTERI

Cunoștințe necesare:

- ✓ limbajul C
- ✓ structura și organizarea memoriei la ATmega2560

13.6.1 TIPURI DE POINTERI

Odată ce există mai multe tipuri de memorie este normal ca să existe mai multe tipuri de pointeri. Tipul unui pointer este dat de:

- organizarea memoriei – în cazul mașinilor AVR, memoria fiind adresabilă liniar, structura internă a unui pointer reprezintă doar o adresă (nu și o adresă de segment, de exemplu);
- tipul de memorie în care este stocat obiectul referit – SRAM, Flash, EEPROM, etc – deoarece, de regulă, există spații de adresare separate;
- spațiul adresabil – cu cât dimensiunea pointerului este mai mare cu atât spațiul este mai vast (dacă el există fizic);
- natura obiectului referit (funcție sau variabilă) – acest aspect determină efectul operatorului de indexare aplicat asupra pointerului.

Pentru declararea unui pointer de un anumit tip se vor folosi niște cuvinte cheie specifice compilatorului (acestea nu fac parte din limbajul C standard) cu rol de modificator al variabilei.

a) Pointeri la funcții

Pointerii la funcții pot avea o dimensiune de 2 sau 3 octeți. Valoarea unui pointer la funcție este adresa referită în octeți împărțită la 2, adică adresa exprimată în cuvinte.

Cuvânt cheie	Spațiul de adresare	Dimensiunea	Tipul indexului
<u>nearfunc</u>	0–0x1FFFFE	2 octeți	signed int
<u>farfunc</u>	0–0x7FFFFFFE	3 octeți	signed long

b) Pointeri la variabile

Pointerii la variabile pot avea o dimensiune de 1, 2 sau 3 octeți.

Cuvânt cheie	Spațiul de adresare	Dimensiunea	Tipul indexului	Tipul memoriei
<code>_tiny</code>	0x0–0xFF	1 octet	signed char	SRAM
<code>_near</code>	0x0–0xFFFF	2 octeți	signed int	SRAM
<code>_far</code>	0x0–0xFFFFFFFF	3 octeți	signed int	SRAM
<code>_huge</code>	0x0–0xFFFFFFFF	3 octeți	signed long	SRAM
<code>_tinyflash</code>	0x0–0xFF	1 octet	signed char	Flash
<code>_flash</code>	0x0–0xFFFF	2 octeți	signed int	Flash
<code>_farflash</code>	0x0–0xFFFFFFFF	3 octeți	signed int	Flash
<code>_hugeflash</code>	0x0–0xFFFFFFFF	3 octeți	signed long	Flash
<code>_eprom</code>	0x0–0xFF	1 octet	signed char	EEPROM
<code>_eprom</code>	0x0–0xFFFF	2 octeți	signed int	EEPROM
<code>_generic</code>	Cel mai semnificativ bit (MSB) indică tipul de memorie referită (1=Flash, 2=SRAM)	1-2 octeți	signed int, signed long	SRAM, Flash

13.6.2 APLICAȚII

Enunț

Să se copie 6 octeți din memoria de program de la adresa 0x0020 într-o variabilă definită în memoria de date și să se verifice dacă operația a fost făcută corect.

Rezolvare

Se va declara o variabilă de tip pointer către o zonă din memoria de cod (variabila propriu-zisă va fi stocată în memoria de date). Se va inițializa variabila de tip pointer cu adresa primului octet ce trebuie de copiat. Acum, cei 6 octeți se pot obține dereferențind de 6 ori variabila de tip pointer adunată cu un offset corespunzător.

Verificarea se face comparând conținutul memoriei din sursă (0x0020) și destinație.

Codul sursă

main.c

```
#include <avr.h>
#include <iom16.h>

unsigned char destination[6];
#define SOURCE 0x0020
int main( void )
{
    unsigned char __flash *ptr;
    unsigned char i;
    ptr = SOURCE;
    for (i = 0; i < 6; i += 1)
    {
        destination[i] = *(ptr + i); // <=> destination[i] = ptr[i];
    }
    return 0;
}
```

14 ATMega16 I²C

14.1 INTRODUCERE ÎN I²C

I²C (Inter – Integrated Circuit) este un protocol de transmisie serial de tip master – slave, folosit pentru a permite comunicarea între dispozitivele electronice integrate. Acest tip de transmisie a fost inventat în anul 1982 de către divizia de circuite semiconductoare NXP a companiei olandeze Philips. Pe parcursul dezvoltării circuitelor integrate, protocolul I²C a suferit mai multe schimbări regăsite în următoarele versiuni:

- **1982: Versiunea inițială** a sistemului I²C folosit pentru transmisia de informații între diversele circuite implementat de Philips;
- **1992: Versiunea 1.0** – prima versiune standardizată care a adăugat, pe lângă modul standard de 100 kHz și așa-numitul Fast Mode (Fm) de 400 khz. De asemenea a fost modificat și modul de adresare, acesta fiind trecut pe 10 biți crescând astfel capacitatea nodurilor suportate de protocol;
- **1998: Versiunea 2.0** – a adăugat modul High Speed 3.4 MHz;
- **2000: Versiunea 2.1** – implementează mici modificări de menenanță a versiunii anterioare;
- **2007: Versiunea 3.0** – a adăugat modul Fast-Mode Plus (Fm+) și un nou mecanism de identificare a dispozitivelor;
- **2013: Versiunea 4.0** – a adăugat modul Ultra Fast-Mode (UFm) pentru noile canale USDA și USCL care foloseau logica de tip push-pull fără a mai fi nevoie de rezistențe de pull-up.

I²C folosește ca mediu de transmisie numai două linii de bus bidirectionale, una pentru pachetele de date (SDA) și una pentru clock (SCL). De asemenea, pentru fiecare linie a bus-ului I²C este nevoie de o singură rezistență de pull-up conectată la sursa de alimentare. Voltajele tipice folosite au valorile de +5V sau de +3.3V, deși sunt permise și alte valori.

Un device conectat într-o rețea I²C poate avea două roluri: cel de master și respectiv, cel de slave, acest device purtând denumirea de nod. Astfel, un nod *master* are rolul de a iniția comunicarea cu nodurile slave și de a genera clock-ul. Un nod *slave* receptionează clock-ul de la un device de tip master și răspunde la cererile acestuia.

Pentru un nod dintr-o conexiune I²C există patru moduri de operație posibile:

1. Master transmitter – nodul master transmite mesaje către un nod slave;
2. Master receiver – nodul master receptionează date de la un nod slave;
3. Slave transmitter – nodul slave transmite mesaje către un nod master;
4. Slave receiver – nodul slave receptionează date de la un nod master.

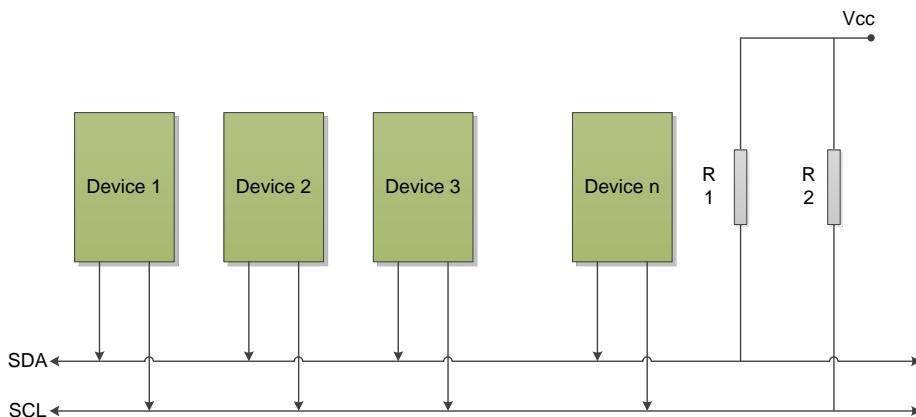


Figura 14.1 Conecțarea device-urilor în protocolul I²C

14.2 INTERFAȚA TWI (TWO WIRE INTERFACE)

Acest tip de interfață este compatibilă cu protocolul I²C, având un mod de adresare pe 7 biți, permitând utilizatorului să conecteze într-o rețea I²C până la 128 de device-uri pe același bus de date. Nodurile din această rețea sunt capabile de a transmite date atât în modul standard (<100 kBps) cât și în modul Fast (<400 kBps).

14.2.1 TRANSMISIA DE DATE FOLOSIND INTERFAȚA I²C

O transmisiune de tip TWI constă dintr-un bloc de start, un bloc indicator de read/write, confirmarea de la device-ul slave, unul sau mai multe pachete de date și un bloc de stop. Fiecare bit din mesajele vehiculate pe bus-ul TWI este însoțit de un puls pe linia de clock. Pentru a asigura validitatea datelor, voltajul liniei de transmisie a datei trebuie să rămână stabil pe nivelul logic high, exceptie făcând cazurile când sunt generate condițiile de start/stop.

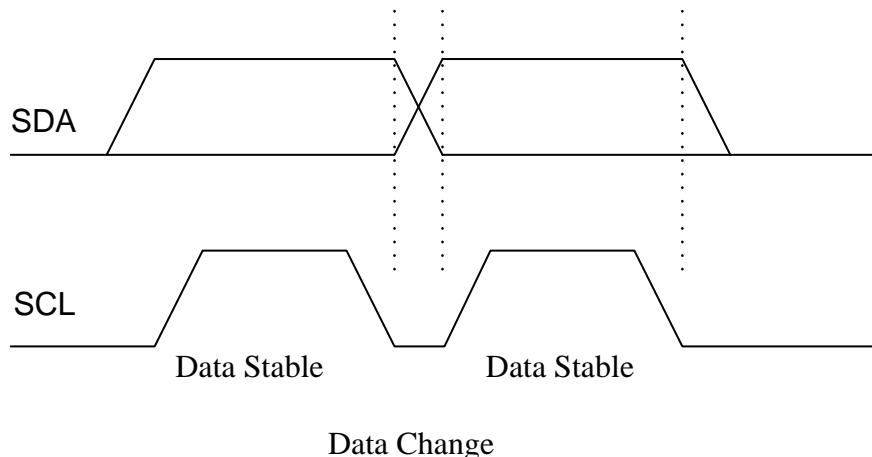


Figura 14.2 Validitatea datelor la transmisia TWI

14.2.2 CONDIȚIILE DE START/STOP

Unul din rolurile unui nod master este de a iniția și de a încheia o transmisie pe bus-ul TWI. Transmisia este inițiată când device-ul master emite o condiție de START, și este încheiată atunci când este emisă condiția de STOP. Între aceste două condiții busul este considerat ocupat, nici un alt master neputând să acceseze bus-ul în acest timp. Singura excepție în acest caz este atunci când între emisia condițiilor de START/STOP se emite o nouă condiție de START. Această condiție se numește REPEATED START și este folosită de un nod de tip master care dorește să reiniețeze un transfer fără a elibera controlul bus-ului TWI. După emisia unei condiții de tip REPEATED START busul este ocupat până la apariția condiției de STOP. Condițiile de START/STOP sunt evidențiate prin schimbarea nivelului logic a liniei SDA atunci când SCL este pe nivelul logic high, conform *Figura 14.3*.

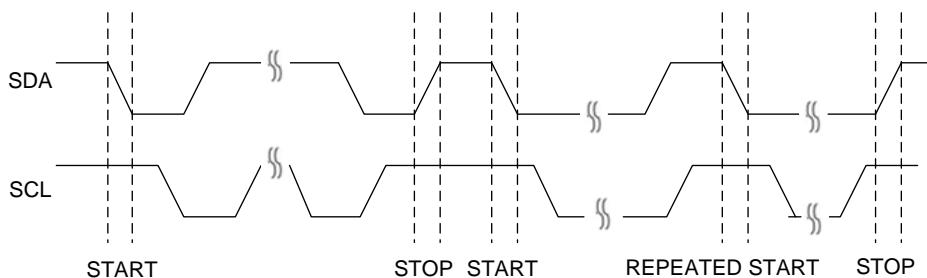


Figura 14.3 Condițiile de START, STOP și REPEATED START

14.2.3 FORMATUL PACHETELOR DE ADRESE

Toate pachetele de adresă vehiculate pe bus-ul TWI au lungimea de nouă biți, dintre care șapte biți sunt cei de adresă, un bit de control al operației de READ/WRITE și un bit de confirmare. Dacă se va seta bitul de READ/WRITE pe high, atunci se va executa o operație de citire, în caz contrar se va executa o operație de scriere. Când un device slave detectează că este adresat, atunci ar trebui să confirme adresarea setând linia SDA pe low în al nouălea ciclu SCL. Dacă nodul slave adresat este ocupat sau nu poate răspunde la cererea device-ului master, atunci linia SDA rămâne pe nivelul logic high în ciclul de confirmare a SCL. În acest caz master-ul trimite o condiție de STOP sau una de REPEATED START pentru a reînîția transmisia. Un pachet de adresă constituie din adresa unui Slave și o cerere de READ/WRITE este denumit generic SLA+R, respectiv SLA+W.

Bitul cel mai semnificativ al pachetului de adresare este transmis mai întâi. Adresele device-urilor slave sunt alocate la alegere de către utilizator dar adresa 0000 000 este rezervată pentru un apel general. Atunci când un apel general este emis, toate device-urile slave trebuie să răspundă prin setarea liniei SDA pe nivelul logic 0 în timpul ciclului de recunoaștere. Atunci când după apelul general se transmite un bit de comandă pentru scriere, toate device-urile slave răspund prin setarea liniei SDA pe low în timpul ciclului de confirmare. Astfel, pachetele de date care vor urma vor fi recepționate de către toate device-urile slave care au confirmat apelul general. În cazul în care apelul general este urmat de un bloc de read, atunci acest apel este ignorat de către sistem, deoarece acest lucru ar însemna că toate device-urile slave ar transmitese mesaje diferite concomitent.

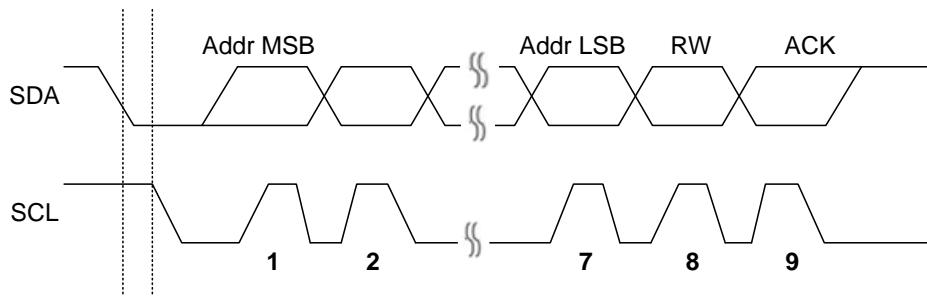


Figura 14.4 Formatul pachetelor de adrese

Pachetele de date transmise pe bus-ul TWI au lungimea fixă de nouă biți, acestea fiind alcătuite dintr-un bit de confirmare și opt biți de date. Atunci când este inițiat un transfer de date, device-ul master generează pulsul de clock, condițiile de START și de STOP, în acest timp device-ul receiver confirmând datele recepționate. Device-ul care recepționează mesajul confirmă primirea acestuia setând linia SDA pe nivelul logic 0 în timpul celui de al nouălea puls al liniei SCL. Dacă receiver-ul lasă linia SDA pe high, este semnalat cazul de NACK (No Acknowledge).

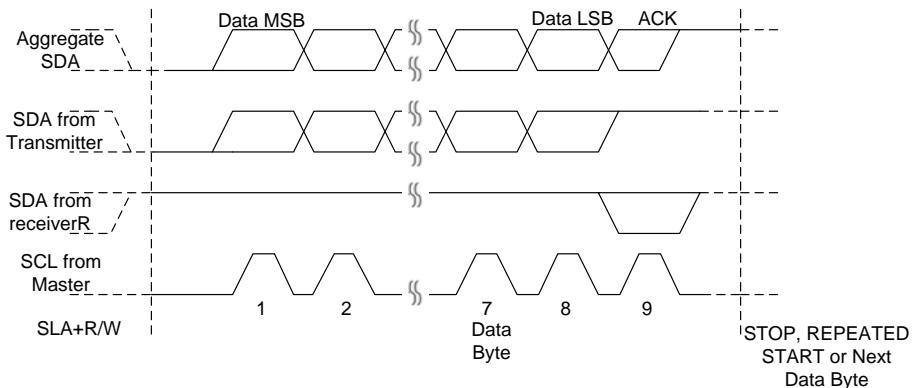


Figura 14.5 Formatul pachetelor de date

14.2.4 COMBINAREA ADRESELOR ȘI A PACHETELOR DE DATE ÎN TIMPUL TRANSMISIEI

O transmisie constă dintr-o condiție de START, un pachet de tipul SLA+R/W, unul sau mai multe pachete de date și o condiție de stop. Un mesaj gol, format dintr-un START urmat de o condiție de STOP, este incorrect.

Slave-ul poate extinde perioada low a SCL mutând linia SCL pe un nivel logic 0. Acest lucru este folositor dacă viteza clock-ului setată de device-ul master este prea mare pentru slave sau dacă slave-ul are nevoie de mai mult timp pentru procesare între transmisiile de date. Faptul că slave-ul mărește perioada de nivel logic low a SCL-ului nu va afecta perioada high a SCL-ului, aceasta fiind determinată de nodul master. Ca o consecință, slave-ul poate reduce viteza de transfer a datelor TWI prelungind ciclul SCL.

14.3 STRUCTURA MODULULUI TWI ÎN ATMEGA16

Principalele submodule componente ale blocului TWI pot fi observate în *Figura 14.6*.

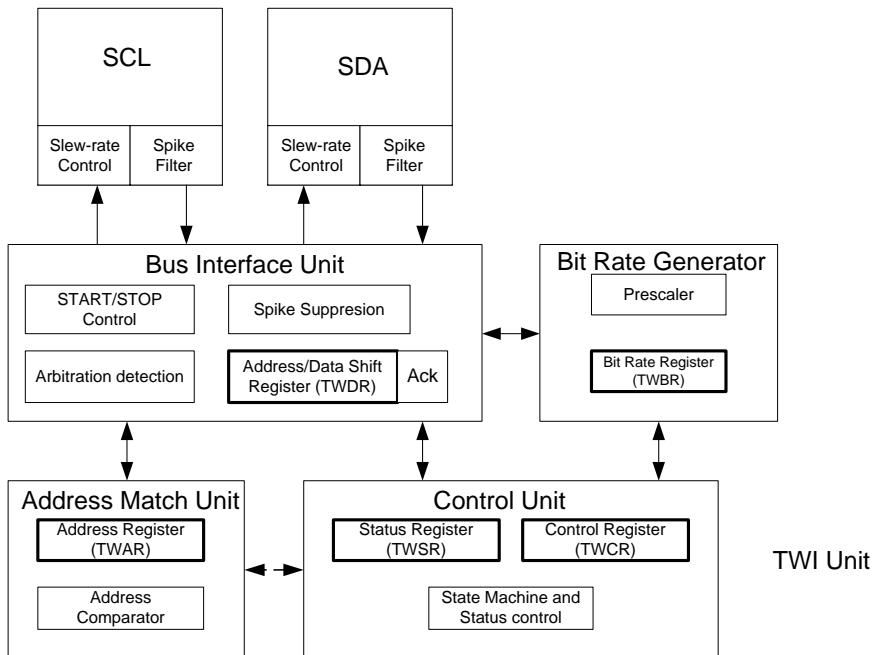


Figura 14.6. Structura modulului TWI

14.3.1 PINII SCL ȘI SDA

Acești pini au rolul de interfață între modulele TWI și cel MCU. Driverele de output conțin componente limitatoare pentru a respecta specificațiile TWI. Astfel, semnalele cu durate mai mici de 50 de nanosecunde sunt eliminate de la început.

14.3.2 MODULUL GENERATOR DE BIT RATE

Acest modul controlează perioada liniei SCL atunci când se operează în modul master. Această perioadă este controlată configurând registrul TWBR și biții de prescaler din registrul TWSR. Operațiile de tip slave nu depind de bit rate sau de configurarea prescaler-ului, dar este necesar ca frecvența de clock a device-ului slave să fie de cel puțin 16 ori mai mare decât frecvența liniei SCL. Frecvența liniei SCL este generată folosind relația următoare:

$$SCL_{freqv} = \frac{CPU_{freqv}}{16 + 2 * TWBR * 4^{TWPS}}$$

În această relație TWBR reprezintă valoarea registrului TWI Bit Rate Register iar TWPS valoarea biților de prescale din registrul TWI Status Register.

14.3.3 MODULUL DE INTERFAȚĂ BUS

Conține registrul de date și adrese TWDR și un controler al condițiilor de START/STOP. Registrul TWDR conține adresele sau bițiile de date care urmează a fi transmiși sau adresa bițiilor recepționați. Pe lângă registrul TWDR, interfața bus-ului conține registrul care semnalizează bitul (N)ACK care urmează a fi transmis/recepționat. Acest registru poate fi alterat doar la recepție modificând registrul TWCR.

Controler-ul de condiții de START/STOP este responsabil cu generarea și detecția semnalelor START, REPEATED START și STOP. Controler-ul de START/STOP poate detecta condițiile acestea chiar dacă microcontroler-ul AVR este într-un mod pasiv, permitând microcontroler-ului să poată fi adresat de modul master.

14.4 DESCRIEREA REGIȘTRILOR TWI

14.4.1 TWI BIT RATE REGISTER – TWBR

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	TWBR							
Valoare inițială	0	0	0	0	0	0	0	0	

Biți 7-0 – TWI Bit Rate Register

TWBR selectează factorul de diviziune pentru generatorul de flux a bițiilor. Generatorul de flux este un divizor de frecvență care generează o frecvență de clock pentru SCL în modul MASTER.

14.4.2 TWI CONTROL REGISTER – TWCR

TWCR este utilizat pentru a controla operațiile din TWI. Acesta este folosit pentru a activa TWI, pentru a iniția accesul la Master prin aplicarea unor condiții de Start pe bus, pentru a genera condițiile de stop și pentru a stopa controlul bus-ului cât timp datele ce trebuie scrise pe bus, sunt scrise în TWDR

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	TWCR
Valoare inițială	0	0	0	0	0	0	0	0	

Bitul 7 – TWINT : TWI Interrupt Flag

Acest bit este setat de hardware când TWI a terminat sarcina curentă și așteaptă răspunsul aplicației software.

Bitul 7 – TWEA: TWI Enable Acknowledge Bit

Bitul TWEA controlează generarea pulsului acknowledge. Dacă TWEA este setat pe 1, ACK este generat pe bus dacă sunt îndeplinite următoarele condiții:

1. Adresa dispozitivului Slave a fost recepționată;
2. Un apel general a fost primit, atât timp cât bitul TWGCE din TWAR este setat;
3. O dată a fost recepționată în modul Master Receiver sau Slave Receiver.

Dacă se setează bitul pe 0, dispozitivul poate fi virtual deconectat de la bus-ul TWI.

Bitul 5 – TWSTA: TWI START Condition

Acest bit se setează pe 1 când se dorește să devină Master pe busul TWI. Hardware-ul TWI verifică dacă bus-ul este liber și în acest caz generează o condiție de start. Oricum, dacă bus-ul nu este liber, acesta așteaptă o condiție de STOP pentru a genera o nouă condiție de START.

Bitul 4 – TWSTO: TWI STOP Condition

Setând bitul TWSTO pe 1 în mod Master, acesta va genera o condiție de STOP pe bus. Când o condiție de STOP este executată, TWSTO se resetează automat.

Bitul 3 – TWWC: TWI Write Collision Flag

Bitul 2 – TWEN: TWI Enable Bit

TWEN setează operațiile TWI și activează interfața TWI. Când TWEN este setat pe 1, TWI preia controlul asupra pinilor conectați la SCL și SDA.

Bitul 1 – Res: Reserved Bit

Acest bit este tot timpul setat pe 0.

Bitul 0 – TWIE: TWI Interrupt Enable

Când acest bit este pe 1, cererile de întrerupere vor fi activate atât timp cât flagul TWINT este high.

14.4.3 TWI STATUS REGISTER – TWSR

Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R/W	R/W	TWSR
Valoare inițială	1	1	1	1	1	0	0	0	

Biții 7-3 – TWS: TWI Status

Acești biți reflectă statusul pe TWI logic și pe bus-ul TWI

Bitul 2 – Res: Reserved Bit

Biții 1-0 – TWPS: TWI Prescaler Bits

Acești biți controlează bit rate-ul prescalerului

TWPS1	TWPS0	Valoare prescaler
0	0	1
0	1	4
1	0	16
1	1	64

14.4.4 TWI DATA REGISTER – TWDR

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W								
Valoare inițială	1	1	1	1	1	1	1	1	

Biții 7-0 – TWD:TWI Data Register

Acești biți conțin date ce trebuie transmisă, sau ultima dată primită pe bus-ul TWI.

14.4.5 TWI (SLAVE) ADDRESS REGISTER – TWAR

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W								
Valoare inițială	1	1	1	1	1	1	1	0	

Biții 7-1 – TWA: TWI (Slave) Address Register

Acești șapte biți constituie adresa Slave a unității TWI

Bitul 0 - TWGCE: TWI General Call Recognition Enable Bit

14.5 PROBLEMĂ REZOLVATĂ

Pentru a inițializa modulul TWI al controlerului ATMega16 se va folosi următoarea funcție:

```
void TWIInit()
{
    //setarea frecvenței liniei SCL la 400kHz
    TWSR = 0x00;
    TWBR = 0x0C;
    //TWI enable
    TWCR = (1<<TWEN);
}
```

Pentru a genera semnalele de START/STOP se vor crea două funcții independente, descrise mai jos:

```
void TWIStart()
{
    TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
    while ((TWCR & (1<<TWINT)) == 0);
}

void TWIStop(void)
{
    TWCR = (1<<TWINT) | (1<<TWSTO) | (1<<TWEN);
}
```

Pentru a scrie pe bus-ul TWI se va utiliza funcția de mai jos. Aceasta scrie bițiile de date în registrul TWDR, iar mai apoi așteaptă terminarea transmisiei și citirea statusului transmisiei din registrul TWSR.

```
void TWIWrite(uint8_t data)
{
    TWDR = data;
    TWCR = (1<<TWINT)|(1<<TWEN);
    while ((TWCR & (1<<TWINT)) == 0);
}
uint8_t TWIGetStatus(void)
{
    uint8_t status;
    //citirea statusului prin mascare
    status = TWSR & 0xF8;
    return status;
}
```

Folosind modulele de mai sus se poate implementa funcția de scriere a memoriei EEPROM:

```
uint8_t EEWriteByte(uint16_t addr, uint8_t data)
{
    //startarea modulului TWI
    TWIStart();
    //verificarea stării acestuia
    if (TWIGetStatus() != 0x08)
        return -1; //eroare
    //selectarea adresei device-ului slave corespunzător și
    //trimiterea datelor
    TWIWrite((uint8_t)((addr & 0x0700)>>7));
    //verificarea stării transmisiei
    if (TWIGetStatus() != 0x18)
        return -1; //eroare
    //stoparea modulului TWI
    TWIStop();
    return 0; //succes
}
```

Similar, mai jos este implementată funcția de citire din memoria EEPROM:

```
uint8_t EEReadByte(uint16_t addr, uint8_t *data)
{
    //startarea modului TWI
    TWIStart();
    if (TWIGetStatus() != 0x10)
        return -1; //eroare
    //selectarea nodului și citirea statusului
    TWIWrite(((uint8_t)((addr & 0x0700)>>7)));
    if (TWIGetStatus() != 0x40)
        return -1; //eroare
    //citirea statusului confirmării și a valorii receptionate
    * data = TWIReadNACK();
    if (TWIGetStatus() != 0x58)
```

```
    return -1;
    //stopare TWI
    TWIStop();
    return 0;
}

uint8_t TWIReadNACK(void)
{
    TWCR = (1<<TWINT)|(1<<TWEN);
    while ((TWCR & (1<<TWINT)) == 0);
    return TWDR;
}
```

15 ADC - Convertorul Analog – Numeric (Analog to Digital Converter)

15.1 CARACTERISTICI

- Rezoluție de 10 biți
- Precizie absolută de ± 2 LSB
- Timp de conversie între 13 – 260 μ s
- 16 canale de intrare nediferențiale multiplexate
- 14 canale de intrare diferențiale
- 4 canale diferențiale cu amplificare de 10x și 200x
- Interval tensiune intrare între 0 și V_{cc} pentru modul nediferential
- Interval tensiune intrare între 2.7 și V_{cc} pentru modul diferențial
- Posibilitatea de a alege între 2.56V sau 1.1V ca tensiune referință
- Moduri de conversie *Free Running* sau *Single*
- Posibilitate de generare întrerupere la sfârșitul conversiei AD
- Posibilitate de a diminua zgomotul

Microcontroler-ul ATmega16 include un convertor analog-numeric cu aproximări succesive de o rezoluție de 10 biți. Unitatea de conversie (ADC) este conectată la un multiplexor analogic cu 8/16 canale ce permite 8 sau 16 intrări nediferențiale posibile de la pinii portului A și a portului F. Referința pentru intrările nediferențiale este 0V (GND).

Dispozitivul permite deasemenea 16/32 intrări diferențiale. Patru dintre intrările diferențiale (ADC1 & ADC0, ADC3 & ADC2, ADC9 & ADC8 și ADC11 & ADC10) sunt prevăzute cu un etaj programabil de amplificare, furnizând pașii de amplificare de 0dB (1x), 20dB (10x) și 46dB (200x) la intrarea diferențială înaintea conversiei AD. Cele 16 canale sunt împărțite în 2 secțiuni de câte 8 canale, unde în fiecare secțiune câte 7 canale diferențiale au în comun terminalul negativ. Dacă se utilizează amplificarea de 1x sau 10x, este de așteptat o rezoluție de 8 biți. Pentru o amplificare de 200x este de așteptat o rezoluție de 7 biți.

Convertorul include un circuit de eşantionare-memorare ce asigură tensiunea de intrare constantă pe timpul conversiei. Diagrama bloc a convertorului este ilustrată în figura xx.

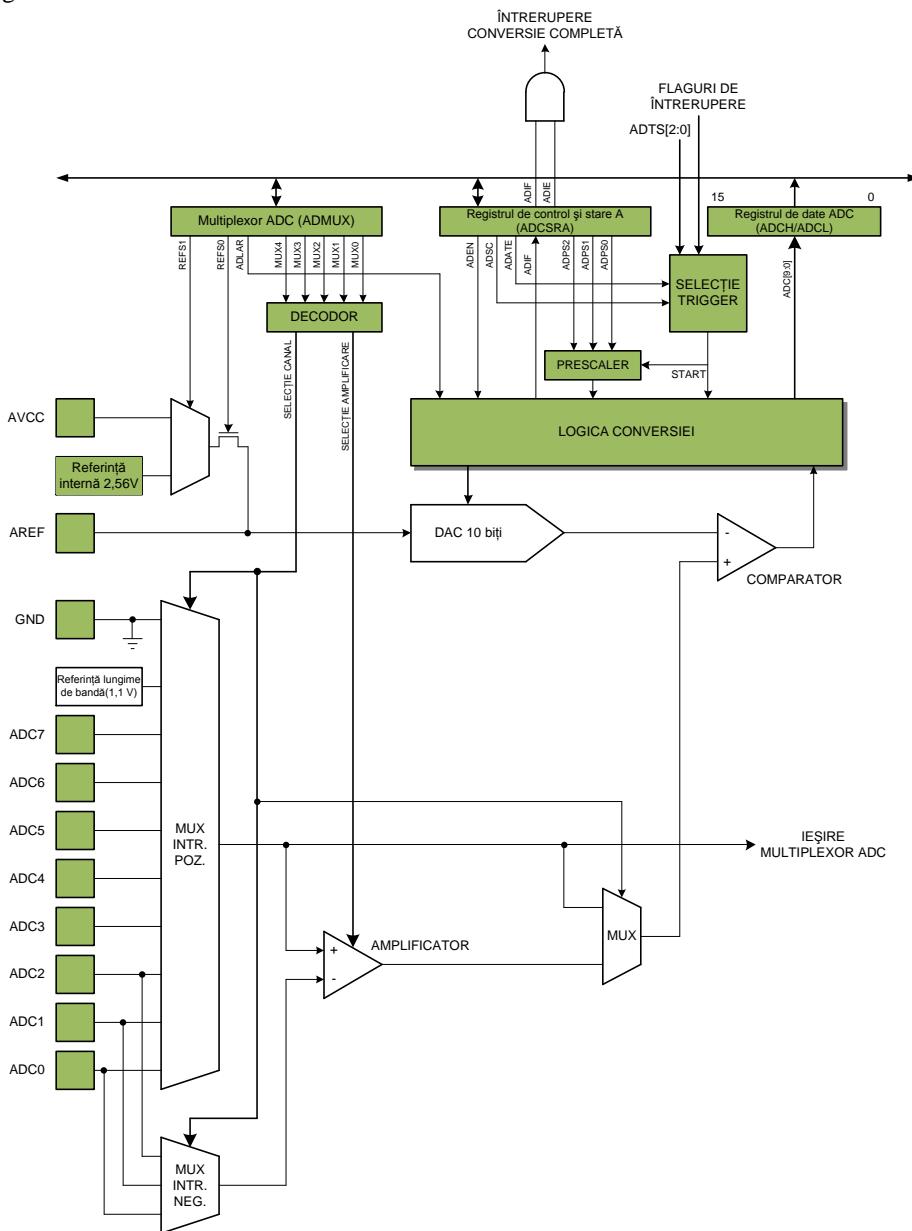


Figura 15.1 Schema bloc a convertorului analog-numeric

Convertorul are un pin de alimentare (**AVCC**). Tensiunea de pe acesta trebuie să nu difere cu mai mult de $\pm 3V$ față de V_{cc} .

Tensiunile interne de referință de valori nominale de 2.56V sau **AVCC**, sunt furnizate de pe chip. Tensiunea de referință poate fi decuplată extern de la pinul **AREF** de un condensator pentru diminuarea zgomotului.

Pentru a reduce energia consumată de convertor, bitul **PRADC** trebuie să fie dezactivat prin scrierea unui 0 logic.

15.1.1 FUNCȚIONARE

Convertorul transformă o tensiune analogică de intrare către o valoare digitală de 10 biți prin aproximări succesive. Valoarea minimă este **GND** și valoarea maximă reprezintă tensiunea pe pinul **AREF** minus 1 LSB. Optional, **AVCC** sau o referință de 1.1V sau 2.56V poate fi conectată la pinul **AREF** prin setarea biților **REFSn** din registrul **ADMUX**. Referința internă poate fi decuplată de un condensator extern conectat la pinul **AREF** pentru diminuarea zgomotului.

Canalul analogic de intrare este selectat prin setarea biților **MUX** din **ADMUX**. Oricare din pinii convertorului, ca masa și referința, pot fi selectați ca intrări nediferențiale. Pinii pot fi selectați că intrări neinversoare sau inversoare ale amplificatorului operațional.

Dacă intrările diferențiale sunt selectate, diferența de tensiune dintre perechea pinilor canalului de intrare selectat devine valoarea de intrare către convertor.

Convertorul este activat prin setarea bitului **ADEN** din registrul **ADCSRA**. Tensiunea de referință și selecția canalelor de intrare nu va avea efect până când **ADEN** nu este setat. Convertorul nu consumă energie când **ADEN** este resetat, în sensul acesta fiind recomandată oprirea convertorului înainte de intrarea în modul de *Power Saving Sleep*.

Convertorul generează un rezultat pe 10 biți care este stocat în regiștrii de date **ADCH** și **ADCL**. Implicit, rezultatul este aliniat la dreapta, dar optional poate fi aliniat la stânga prin setarea bitului **ADLAR** în registrul **ADMUX**.

Dacă rezultatul este aliniat la stânga și nu este necesară o precizie de mai mult de 8 biți, este suficientă citirea lui **ADCH** pentru a afla valoarea conversiei. În orice caz, **ADCL** trebuie citit primul, apoi **ADCH**, pentru a fi siguri că valoarea citită din cele 2 registre aparține aceleiași conversiei.

15.1.2 INITIALIZAREA UNEI CONVERSII

O conversie este inițializată prin setarea bitului **ADSC**. Acesta rămâne pe 1 atât timp cât conversia are loc, și va fi resetat hardware când aceasta este completă. Dacă un canal de date diferit este selectat în timpul unei conversii, convertorul va termina procesul curent înainte de a schimba canalul.

Alternativ, o conversie poate fi declanșată automat în mai multe moduri. Auto-declanșarea este activată prin setarea bitului **ADATE** în registrul **ADCSRA**. Sursa de declanșare este selectată prin setarea biților **ADTS** din registrul **SFIOR**. Când un front pozitiv al clockului este detectat de semnalul de declanșare, *prescaler*-ul convertorului este resetat și procesul începe. Acest mod furnizează o posibilitate de începere a conversiei la intervale fixe. Dacă alt front pozitiv este detectat în acest timp, acesta este ignorat. Flagul de înterupere trebuie resetat pentru a declanșa o nouă conversie la următorul eveniment.

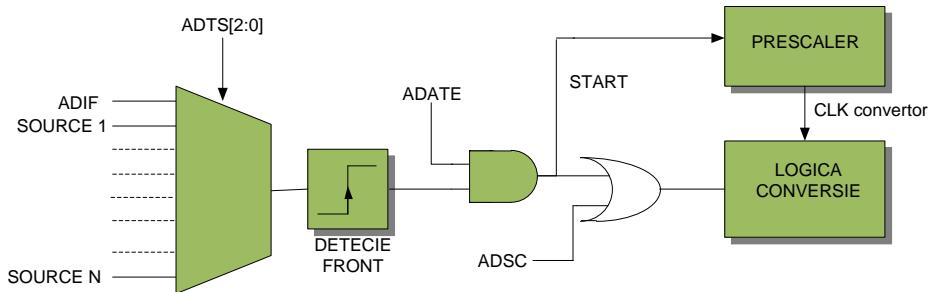


Figura 15.2 Blocul de auto-declanșare

Setarea flagului de întrerupere va duce la începerea unei noi conversii imediat ce se termină procesul curent. Convertorul operează astfel în modul *Free Running*, în mod constant eșantionând și actualizând registrul de date. Prima conversie trebuie începută prin setarea către bitul **ADSC** din registrul **ADCSCRA**.

Dacă auto-declanșarea este activată, conversiile *Single* pot fi începute prin setarea bitului **ADSC** din registrul **ADCSCRA**. Bitul **ADSC** poate fi deasemenea utilizat pentru a afla dacă o conversie este în progres. Bitul **ADSC** va fi citit ca 1 în timpul unei conversii, independent de modul cum a fost pornit procesul.

15.1.3 TIMPI DE PRESCALARE ȘI CONVERSIE

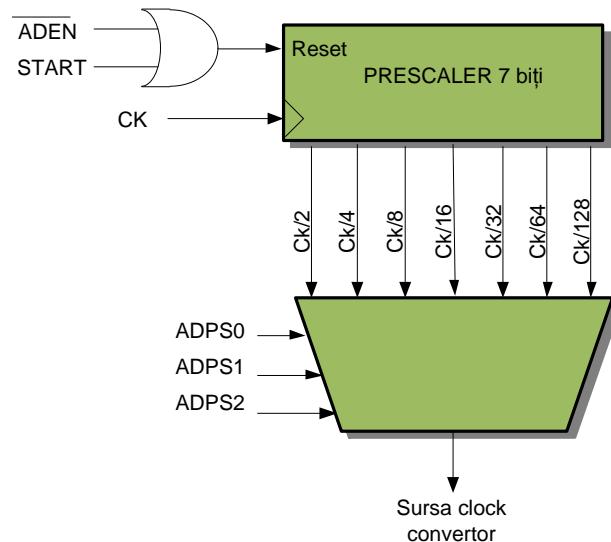


Figura 15.3 Prescaler-ul convertorului analog-numeric

Implicit, circuitul de aproximății succesive necesită un clock de frecvență între 50KHz și 200KHz. Dacă este suficientă o rezoluție mai mică de 10 biți, frecvența clock-

ului convertorului poate fi mărită până la 1000KHz, obținând astfel o rată de eșantionare mai mare.

Convertorul conține un *prescaler*, care generează o frecvență de clock acceptabilă de la orice frecvență CPU sub 100KHz. Factorul *prescaler*-ului este setat de biții **ADPS** din registrul **ADCSRA**.

Prescalerul începe să numere din momentul când ADC-ul este pornit prin setarea bitului **ADEN** din **ADCSRA**.

O conversie obișnuită durează 13 cicli clock convertor. Prima procesare după ce convertorul este pornit se face pe parcursul a 25 cicli clock pentru a inițializa circuitul analogic.

Un proces de eșantionare și memorare are loc în 1.5 cicli după începerea unei conversii obișnuite și 13.5 cicli după începerea primei conversii. Când o procesare este definitivată, rezultatul este scris în regiștrii de date și **ADIF** este setat. Software-ul poate să seteze **ADSC** din nou, și o nouă conversie va fi inițializată la primul front crescător al semnalului de tact.

Când auto-declanșarea este utilizată, valoarea din *prescaler* devine 1. Acest lucru asigură o întârziere fixă de la evenimentul declanșator până la începutul conversiei. În acest mod, eșantionarea și memorarea introduce 2 cicli de tact după frontul pozitiv al sursei de *clock* al declanșatorului. Pentru sincronizarea logică sunt necesari 3 cicli.

În modul *Free Running*, o nouă conversie va începe imediat după o conversie completă, cât timp **ADSC** este 1.

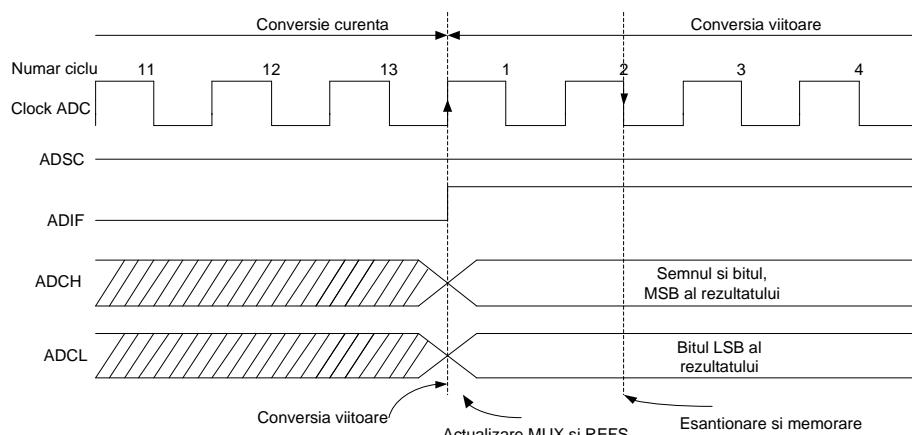


Figura 15.4 Diagrama de timp în cazul modului de conversie Free Running

	Memorare(Cicli de la Conversie(Cicli) începutul conversiei)	
Prima conversie	13.5	25
Conversii normale, nediferentiale	1.5	13
Conversii autodeclansate	2	13.5
Conversii normale, diferențiale	1.5/2.5	13/14

Tabel 4 Timpii asociați tipurilor de conversie

15.1.4 CANALELE DIFERENȚIALE

Când se utilizează canalele diferențiale, anumite aspecte ale conversiei trebuie luate în considerare.

Conversiile diferențiale sunt sincronizate cu *clock*-ul intern CK_{ADC2} cu frecvență egală cu semnalul de tact al convertorului. Această sincronizare are loc în aşa fel încât eșantionarea și memorarea se execută la o fază specificată a CK_{ADC2} .

Dacă modul diferențial este utilizat și conversiile sunt auto-declanșate, convertorul trebuie oprit între conversii. Când auto-declanșarea este utilizată, *prescaler*-ul convertorului este resetat înainte de începerea conversiei. Din moment ce acest stagiul este dependent de prioritatea stabilității semnalului de tact, această conversie nu va fi validă. Prin dezactivarea și reactivarea convertorului între fiecare conversie (punând 0, apoi 1 logic în bitul **ADEN** din **ADCSRA**) , doar conversiile extinse sunt realizate.

15.1.5 SCHIMBAREA CANALULUI SAU SELECȚIA REFERINȚEI

Biții **MUXn** și **REFS1:0** din registrul **ADMUX** sunt memorați într-un registru temporar la care procesorul are acces. Acest lucru asigură faptul că selecția referinței și a canalelor are loc la un moment de timp sigur în timpul conversiei. Canalul și selecția referinței este actualizată continuu până la începerea unei noi conversii. Odată ce conversia a început, posibilitatea de a schimba canalul și selecția referinței este blocată.

Dacă auto-declanșarea este utilizată, timpul exact al momentului când a avut loc declanșarea nu poate fi precizat cu acuratețe. Considerații speciale trebuie luate când se actualizează registrul **ADMUX**, cu scopul de a controla care conversie va fi afectată de noua setare.

Dacă biții **ADATE** și **ADEN** sunt setați, o întrerupere poate avea loc în orice moment. Dacă valoarea registrului **ADMUX** este schimbată în această perioadă, utilizatorul nu poate spune care conversie este bazată pe nouă sau vechea setare. **ADMUX** poate fi actualizat sigur în următoarele moduri:

- Când **ADATE** și **ADEN** sunt resetați
- În timpul unei conversii, minim 1 ciclu clock ADC după declanșare
- După o conversie, înainte ca flagul de întrerupere utilizat ca declanșator să fie resetat

15.1.6 CANALELE DE INTRARE

Când are loc schimbarea canalelor, utilizatorul trebuie să țină cont de următoarele precizări pentru a se asigura că selecția canalului a fost corectă:

În modul de conversie *Single*, selecția canalului se face înainte de începerea conversiei. Canalul selectat poate fi schimbat într-un ciclu de *clock* după ce a avut loc scrierea unui 1 logic în **ADSC**. În orice caz, cea mai simplă metodă este să se aștepte finalul conversiei înainte de a schimba canalul.

În modul *Free Running*, întotdeauna canalul este selectat înaintea primei conversii. Selecția canalului poate fi realizată într-un ciclu *clock* după ce a avut loc scrierea unui 1 logic în **ADSC**. În orice caz, cea mai simplă metodă este să se aștepte finalul conversiei înainte de a schimba canalul. Din moment ce următoarea conversie a început automat, următorul rezultat va reflecta canalul selectat anterior.

15.1.7 TENSIUNEA DE REFERINȚĂ

Tensiunea de referință pentru convertor indică intervalul de conversie. Canalele nediferențiale care depășesc tensiunea de referință V_{REF} vor avea valoarea 0x3FF. Tensiunea V_{REF} poate fi selectată cu **AVCC**, prin 1.1V referință internă, sau extern cu valoarea de 2.56V de la pinul **AREF**.

Pinul **AVCC** este conectat la convertor printr-un comutator pasiv. Referința internă de 1.1V este generată printr-un amplificator operațional. În celălalt caz, pinul extern **AREF** este conectat direct la convertor, și tensiunea de referință este mai bine protejată la zgomot prin conectarea unui capacitor între pinul **AREF** și masa.

Dacă utilizatorul lucrează cu o sursă de tensiune fixă conectată la **AREF**, acesta poate să nu mai utilizeze opțiunile de setare a tensiunii în aplicație, deoarece ele vor fi direcționate direct către sursa de tensiune externă. Dacă nu se aplică o tensiune externă la pinul **AREF**, utilizatorul poate alege ca tensiune de referință dintre 1.1V și 2.56V de la pinul **AVCC**.

15.1.8 DIMINUAREA ZGOMOTULUI

Convertorul pune la dispoziție un element de circuit ce realizează diminuarea semnalului de zgomot prezent la intrare, indus de periferice și de microprocesor în general. Pentru a utiliza această caracteristică, următoarea procedură trebuie să fie urmată:

1. Să se asigure că convertorul este activat și nu este în timpul unei procesări. Modul de conversie *Single* trebuie selectat și activată generarea de întrerupere la sfârșitul conversiei.
2. Se intră în modul de diminuare a zgomotului (sau modul *Idle*). Convertorul va începe conversia odată ce procesorul este dezactivat.
3. Dacă nici o întrerupere nu are loc înainte de terminarea conversiei curente, generarea întreruperii va activa procesorul și se va executa rutina de conversie. Dacă o altă întrerupere activează procesorul înaintea terminării conversiei ce are loc, aceasta va fi executată, și va avea loc o întrerupere la sfârșitul conversiei.

De notat este faptul că convertorul nu va fi automat oprit când se intră în modul *Sleep* față de modul *Idle*. Utilizatorul este sfătuit să reseteze bitul **ADEN** înainte de a intra în modul *Sleep* pentru a evita consumul excesiv de energie.

Dacă convertorul este activat în modul *Sleep* și utilizatorul vrea să realizeze conversii diferențiale, este recomandat să se oprească convertorul și pornit atunci când este necesară determinarea rezultatului unei conversii extinse.

15.1.9 CONSTRUCȚIA INTRĂRILOR ANALOGICE

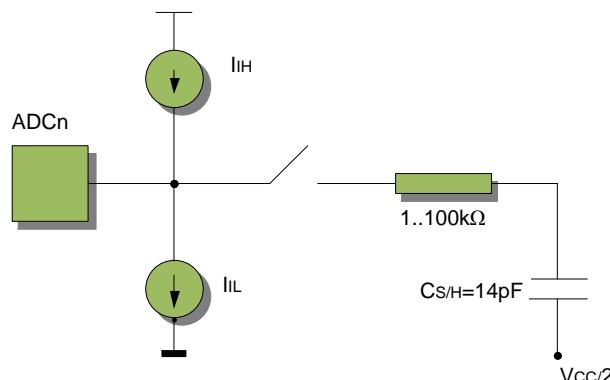


Figura 15.5 Structura electrică a unui pin de intrare în convertor

Construcția intrărilor analogice pentru canalele nediferențiale este ilustrată în figura xx. Convertorul este optimizat pentru tensiunile analogice cu o impedanță de ieșire de aproximativ $10\text{k}\Omega$ sau mai puțin. Dacă o asemenea sursă este folosită, timpul de eşantionare va fi neglijabil. Dacă o sursă cu o impedanță mai mare este folosită, timpul de eşantionare va depinde de perioada necesară condensatorului S/H (de eşantionare și memorare) să se încarce. Este recomandat ca utilizatorul să utilizeze surse cu impedanță mică și variații neglijabile, din moment ce acest lucru va minimaliza transferul sarcinii către capacitorul S/H.

Semnalele ce au în compoziție frecvențe mai mari decât frecvența Nyquist ($F_{ADC}/2$) nu sunt indicate, pentru a evita distorsiunile ce pot apărea din cauza conveleției semnalului. Utilizatorul este indicat să îndepărteze componentele de frecvență mare cu un filtru trece-jos înainte de a aplica semnalul la intrarea convertorului.

15.1.10 METODE DE DIMINUARE A ZGOMOTULUI ANALOGIC

Circuitele digitale din interiorul și exteriorul convertorului generează inducție electro-magnetică (EMI) care poate afecta acuratețea măsurărilor analogice. Dacă nivelul de precizie dorit este critic, nivelul de zgomot poate fi redus aplicând următoarele puncte:

1. Dacă este posibil, semnalul analogic trebuie distribuit către convertor pe o cale cât mai scurtă. Căile analogice să aibă o masă comună bine delimitată și acestea să fie la o distanță suficientă de traseele digitale de mare viteză.

2. Pinul AVCC să fie conectat la V_{CC} prin intermediul unei rețele LC ca în figura de mai jos.
3. Este indicat să se utilizeze funcția de diminuare a zgomotului indus de către CPU.
4. Dacă oricare din pinii convertorului sunt folosiți ca ieșiri digitale, este esențial să nu fie comutați când o conversie este în funcționare.

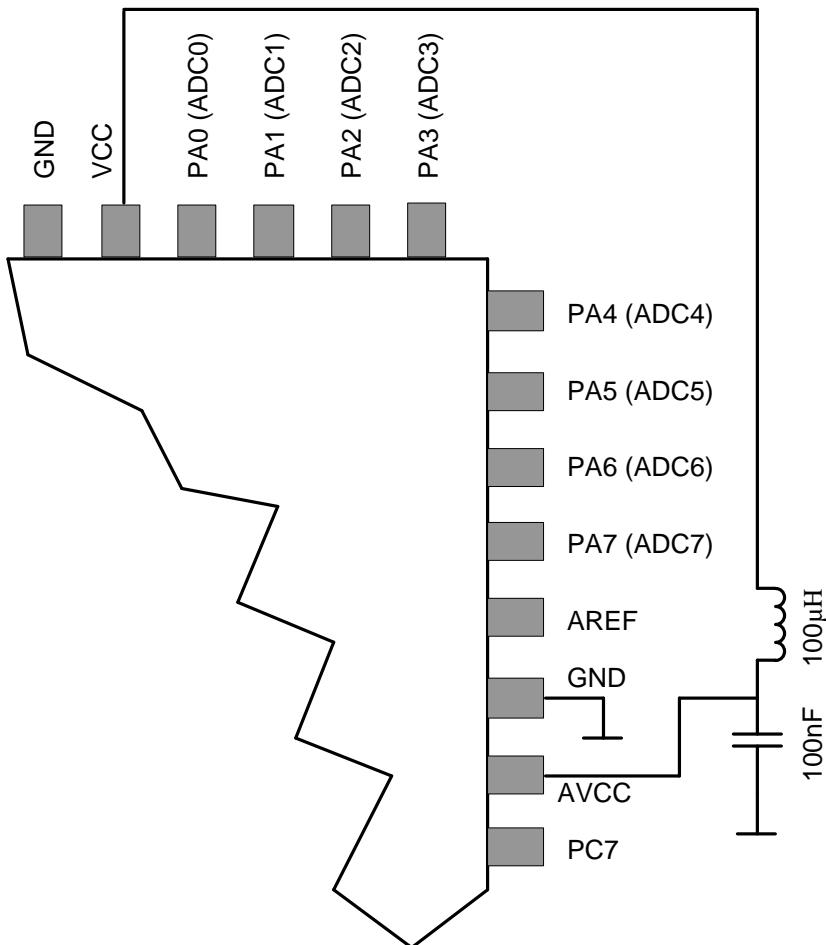


Figura 15.6 Conexiunile convertorului

15.1.11 MĂRIMI DE PRECIZIE A CONVERTORULUI

Un convertor nediferențial pe n biți transformă o tensiune liniară cuprinsă între GND și V_{REF} în 2^n pași. Codul rezultat minim poate fi 0, iar maxim $2^n - 1$.

O serie de parametri descriu deviația de la comportamentul ideal:

- Offset-ul: reprezintă deviația primei tranziții (0x000 la 0x001) comparată cu tranziția ideală (la 0.5 LSB). Valoarea ideală este 0 LSB

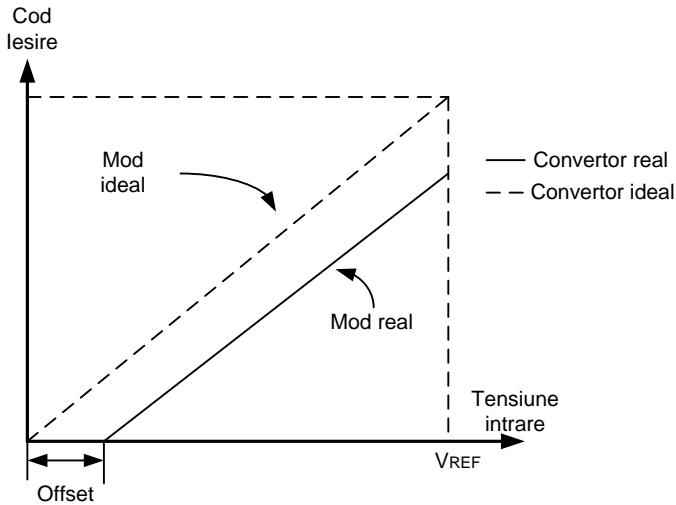


Figura 15.7 Eroarea de offset unui convertor analog-numeric

- Eroarea de amplificare: este definită ca fiind deviația ultimei tranziții (de la 0x3FE la 0x3FF) comparată cu tranziția ideală (la 1.5 LSB sub valoarea maximă admisă) în cazul în care eroarea de offset a fost calibrată. Valoarea ideală este 0 LSB.

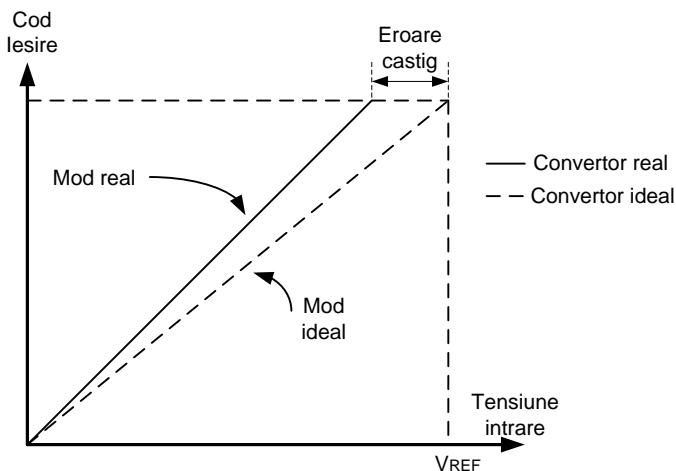


Figura 15.8 Eroarea de offset unui convertor analog-numeric

15.1.11.1 Eroarea de amplificare

- Neliniaritatea integrală (INL – *Integral Non-Linearity*): în cazul în care eroarea de offset și eroarea de amplificare sunt compensate, neliniaritatea integrală se definește ca fiind deviația tranzitiei curente comparată cu tranzitia ideală corespunzătoare oricărui cod. Valoarea ideală este 0 LSB.

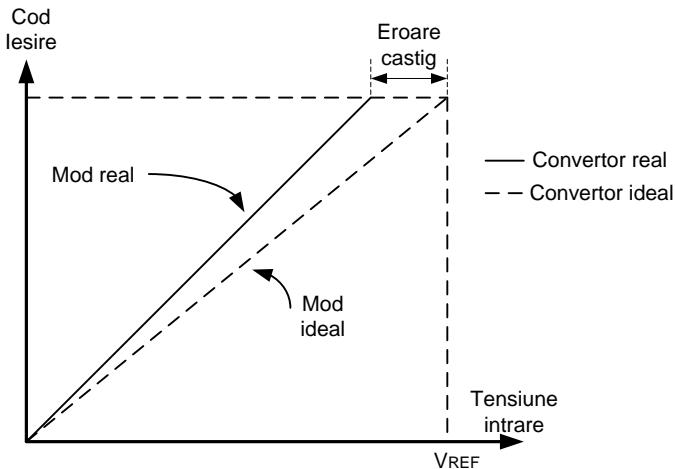


Figura 15.9 Eroarea de offset unui convertor analog-numeric

15.1.11.2 Neliniaritatea integrală

- Neliniaritatea diferențială (DNL – *Differential Non-Linearity*): reprezintă deviația maximă a codului dintre 2 tranzitii adiacente având ca referință valoarea ideală de 1 LSB. Valoarea ideală este 0 LSB.

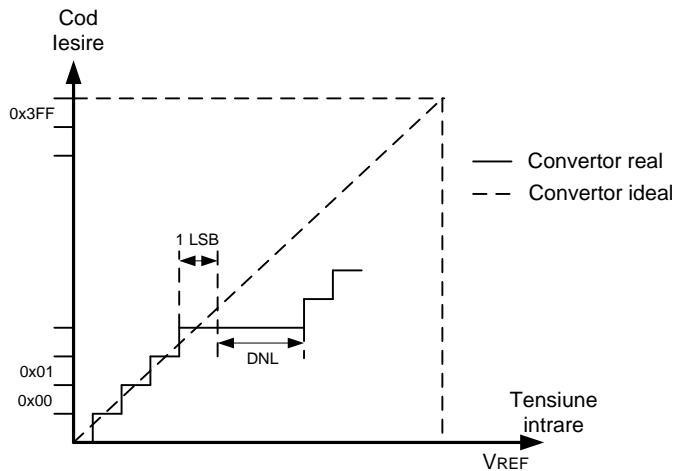


Figura 15.10 Eroarea de offset unui convertor analog-numeric

15.1.11.3 Neliniaritatea diferențială

- Eroarea de cuantizare: datorită cuantizării tensiunii de intrare într-un număr finit de coduri (de ordinul 1 LSB), unele valori apropiate, dar diferite la tensiunii de intrare vor avea aceeași codificare.
- Precizia absolută: reprezintă maximul deviației al unei tranziții curente raportat la o tranziție ideală o oricărui cod. Aceasta este un rezultat al erorii de *offset*, erorii de amplificare, neliniarității și al erorii de cuantizare. Valoarea ideală este ± 0.5 LSB.

15.1.12 REZULTATUL CONVERSIEI

După ce conversia este completă (**ADIF** este 1), rezultatul acesteia poate fi găsit în regiștri de date ale convertorului (**ADCH**, **ADCL**).

Pentru conversiile nediferențiale, rezultatul este:

$$ADC = \frac{V_{IN} * 1024}{V_{REF}}$$

unde V_{IN} este tensiunea de pe pinul de intrare și V_{REF} tensiunea de referință aleasă.

Dacă sunt folosite canalele diferențiale, rezultatul este:

$$ADC = \frac{(V_{POS} - V_{NEG}) * 512}{V_{REF}}$$

unde V_{IN} este tensiunea pe pinul de intrare pozitiv și V_{NEG} tensiunea de pe pinul negativ, iar V_{REF} tensiunea de referință aleasă. Rezultatul este păstrat în complement față de 2, de la 0x200 (-512d) până la 0x1FF (+511d). De menținut este faptul că dacă utilizatorul vrea să verifice polaritatea rezultatului, este suficient ca bitul MSB să fie citit (**ADC9** din **ADCH**). Dacă bitul este 1, atunci rezultatul este negativ, în caz contrar este pozitiv.

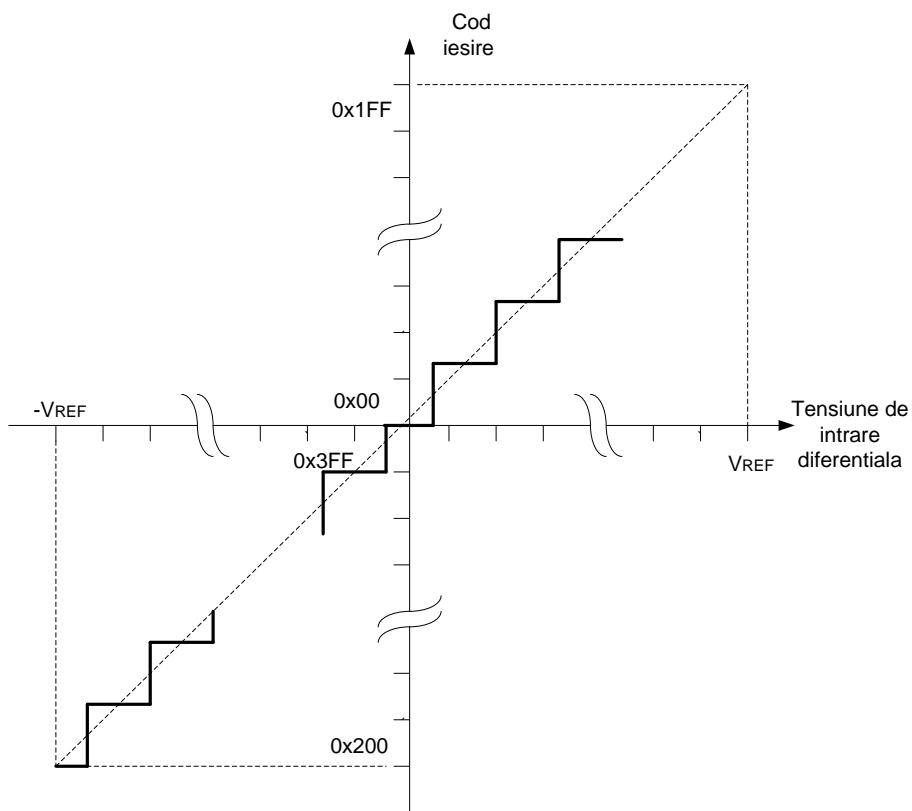


Figura 15.11 Eroarea de *offset* unui convertor analog-numeric

15.1.13 INTERVALELE DE MASURĂ ÎN CADRUL FOLOSIRII INTRĂRIILOR DIFERENȚIALE

Tabelul de mai jos conține codurile de ieșire rezultate dacă se utilizează o pereche de intrări diferențiale (ADCn-ADCm) la care se aplică amplificarea GAIN și tensiunea de referință VREF.

V_{ADCn}	Cod citit	Valoare decimală corespunzătoare
$V_{ADCm} + V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + 0.999V_{REF}/GAIN$	0x1FF	511
$V_{ADCm} + 0.998V_{REF}/GAIN$	0x1FE	510
...
$V_{ADCm} + 0.001V_{REF}/GAIN$	0x001	1

V_{ADCm}	0x000	0
$V_{ADCm} - 0.001V_{REF}/GAIN$	0x3FF	-1
...
$V_{ADCm} - 0.999V_{REF}/GAIN$	0x201	-511
$V_{ADCm} - V_{REF}/GAIN$	0x200	-512

Tabel 15.1 Corelația dintre tensiunea de intrare și codurile de ieșire**15.1.14 DESCRIERE REGIȘTRI****15.1.14.1 ADMUX – Registrul de selecție**

Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Valoare inițială	0	0	0	0	0	0	0	0	

- Biții 7:6 – **REFS1:0** : Biții de selecție ai referinței

Acești biți selectează tensiunea de referință pentru convertor, așa cum este arătat în tabela de mai jos. Dacă acești biți sunt modificați în timpul conversiei, schimbarea nu va avea efect decât după completarea conversiei curente (**ADIF** din **ADCSRA** este 1). Opțiunile pentru tensiunea de referință pot să nu fie utilizate dacă o tensiune de referință externă este aplicată la pinul **AREF**.

REFS REFS Selectia tensiunii de referinta

1	0	
0	0	Referința la pinul AREF, V_{REF} intern oprit
0	1	Referința la pinul AVCC cu condensator extern conectat la pinul AREF
1	0	1.1V tensiune internă de referință cu condensator extern conectat la pinul AREF
1	1	2.56V tensiune internă de referință cu condensator extern conectat la pinul AREF

Tabel 15.2 Selecția tensiunii de referință pentru convertorul analog-numeric

- Bitul 5 – **ADLAR**: Aliniere la stânga a rezultatului

Acet bit modifică aşezarea rezultatului conversiei în registrul de date al acestuia. Scriind 1 în ADLAR, rezultatul va fi aliniat la stânga. În caz contrar, rezultatul este aliniat la dreapta.

- Biții 4:0 – **MUX4:0** : Biții de selecție ai canalului analogic și ai amplificării

Valoarea acestor biți selectează care combinație de intrări analogice este prezentă la intrarea convertorului. Dacă acești biți sunt schimbați în timpul conversiei, efectul lor va fi vizibil la următoarea conversie (bitul **ADIF** din **ADCSRA** este 1).

MUX[4:0]	Intrare Single Ended	Intrare Diferențială Pozitivă	Intrare Diferențială Pozitivă	Ampl.
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4	N/A		
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111	N/A	ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x

10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22V (V _{BG})	N/A		
11111	0 V (GND)			

Tabel 15.3 Selecția canalului de conversie și a amplificării**15.1.14.2 ADCSRA - Registrul A de control și stare a convertorului**

Bit	7	6	5	4	3	2	1	0	ADCSRA
Read/Write	R/W								
Valoare inițială	0	0	0	0	0	0	0	0	

- Bitul 7 – **ADEN**: Activare convertor

Setând acest bit, se va activa convertorul. La resetarea acestui bit, convertorul este oprit. Dacă se oprește convertorul în momentul efectuării unei conversii, conversia este interrupță, datele rezultate fiind pierdute.

- Bitul 6 – **ADSC**: Bitul de pornire al unei conversii analog-numerice

În modul de conversie *Single*, scriind în acest bit 1, se va declanșa o nouă conversie.

În modul *Free Running*, se scrie 1 în acest bit pentru a începe prima conversie.

- Bitul 5 – **ADATE**: Bitul de activare al autodeclanșării

Când acest bit este setat, autodeclanșarea convertorului este activată. Convertorul va începe o nouă conversie la apariția unui front pozitiv al semnalului de declanșare. Sursa de declanșare este selectată prin biții **ADTS** din registrul **ADCSR**.

- Bitul 4 – **ADIF**: Bitul de detecție a intreruperii generate de o conversie

Acest bit este setat când conversia curentă este completă și rezultatul acesteia este scris în registrul de date. Generarea intreruperii este efectuată la terminarea conversiei dacă bițul **ADIE** și bițul I din registrul **SREG** sunt setați. **ADIF** este resetat hardware când are loc execuția vectorului de intrerupere.

- Bitul 3 – **ADIE**: Bitul de activare întrerupere la sfârșitul conversiei
Când în acest bit și bitul I din registrul **SREG** sunt setați, este activată o întrerupere la terminarea conversiei în curs de prelucrare.
- Biții 2:0 – **ADPS2:0**: Biții de selecție ai factorului de prescalare al convertorului
Acești biți determină factorul de divizare dintre frecvența XTAL și semnalul de tact al convertorului.

ADPS2	ADPS1	ADPS0	Factor divizare
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabel 5.4 Selecția factorului de prescalare al convertorului

15.1.14.3 ADCL și ADCH – Regiștrii de date ai convertorului

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	ADCH	ADCL
	-	-	-	-	-	-	ADC9	ADC8		
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0		
	7	6	5	4	3	2	1	0		
Read/Write	R	R	R	R	R	R	R	R		
	R	R	R	R	R	R	R	R		
Valoare inițială	0	0	0	0	0	0	0	0		
	0	0	0	0	0	0	0	0		

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Valoare inițială	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Când conversia este completă, rezultatul este stocat în aceste registre. Dacă sunt utilizate canalele diferențiale, rezultatul este prezent sub complement față de 2.

Când registrul **ADCL** este citit, registrul de date nu este actualizat până ce nu are loc citirea registrului **ADCH**. Dacă rezultatul este aliniat la stânga și nu este necesară o precizie mai mare de 8 biți, este suficientă numai citirea registrului **ADCH**. În caz contrar registrul **ADCL** trebuie citit mai întâi, apoi registrul **ADCH**.

- **ADC9:0** – Biții rezultat ai conversiei

Acești biți sunt utilizati pentru a stoca rezultatul unei conversii analog-numerice.

15.1.14.4 SFIOR - Registrul cu funcție specială

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Valoare inițială	0	0	0	0	0	0	0	0	

- Biții 2:0 – **ADTS2:0** : Biții de selecție ai sursei de autodeclanșare

Dacă bitul **ADATE** din registrul **ADCSRA** este 1, valoarea acestor biți indică care sursă va genera autodeclanșarea unei conversii analog-numerice. Dacă bitul **ADATE** este resetat, setarea bițiilor **ADTS2:0** nu va avea nici un efect. O conversie va fi declanșată de frontul pozitiv al flagului de întrerupere selectat.

ADTS2	ADTS1	ADTS0	Sursa declanșare
--------------	--------------	--------------	-------------------------

0	0	0	Modul <i>Free Running</i>
0	0	0	Comparator analogic

0	1	0	Cerere de întrerupere externă 0
0	1	1	<i>Timer/Counter0 Compare Match A</i>
1	0	0	<i>Timer/Counter0 Overflow</i>
1	0	1	<i>Timer/Counter1 Compare Match B</i>
1	1	0	<i>Timer/Counter1 Overflow</i>
1	1	1	<i>Timer/Counter1 Capture Event</i>

Tabel 15.5 Selecția sursei de autodeclanșare a conversiei