3. INTRODUCERE ÎN KIT-UL DE DEZVOLTARE

3.1 INTRODUCERE

Kit-ul de dezvoltare format din **SiBRAIN for ATmega1280** și **UNI Clicker de la Mikroe** este proiectat pentru a oferi o platformă versatilă și ușor de utilizat, dedicată dezvoltării rapide de aplicații embedded. Acest kit include un microcontroler **ATmega1280** pe placa **SiBRAIN**, care este completat de funcționalitățile multiple ale plăcii **UNI Clicker**. Acestea permit extinderea prin conectarea rapidă a diferitelor module periferice prin sloturi **mikroBUS**, asigurând astfel compatibilitate cu o gamă variată de senzori și dispozitive externe.

3.2 PREZENTAREA MICROCONTROLERULUI ATMEGA1280

ATmega1280 este un microcontroler **AVR** de **8 biți** cu arhitectură **RISC** avansată, care permite execuția majorității instrucțiilor în cicluri de ceas **unice**, oferind performanță ridicată. Printre caracteristicile sale principale se numără:

- frecvențe de **0-8 MHz** atunci când tensiunea de alimentare este între **2.7 V și 5.5 V;**
- frecvențe de **0-16 MHz** atunci când tensiunea de alimentare este între **4.5 V** și **5.5 V**;
- **32 KB** de memorie **Flash** și **2 KB** de **SRAM**;
- **1 KB** de **EEPROM** pentru stocarea datelor care trebuie păstrate după oprirea sistemului;
- **JTAG** pentru **debug** și **programare**;
- până la **53 de linii I/O programabile**, configurabile pentru a interacționa cu diverse periferice;
- interfețe integrate de comunicare serială: **USART**, **SPI**, **I²C** și altele, ideale pentru comunicarea cu module externe;
- **32 de registre** de uz general;
- **2 timere** de **8 biți** și **un timer** de **16 biți;**
- **4** canale de **PWM**;
- **ADC** cu **8 canale** de **10 biți**;
- **5 moduri** de **sleep**: **Idle**, **ADC Noise Reduction**, **Power-Save**, **Power-Down** și **Stand-By**.

3.3 PREZENTAREA PLĂCII SIBRAIN PENTRU ATMEGA1280

Placa **SiBRAIN for ATmega1280** oferă o platformă compactă, dar puternică, care facilitează dezvoltarea aplicațiilor embedded. Aceasta include conectori **dedicați** pentru acces la **toate interfețele** microcontrolerului, precum și pini de alimentare **stabilă** pentru funcționarea la **3.3 V** și **5 V**. Placa include, de asemenea, un **cristal extern** de **16 MHz** pentru precizie în generarea ceasului sistemului, precum și facilități pentru conectarea la interfețele **JTAG / SPI** pentru **debug** și **programare**.

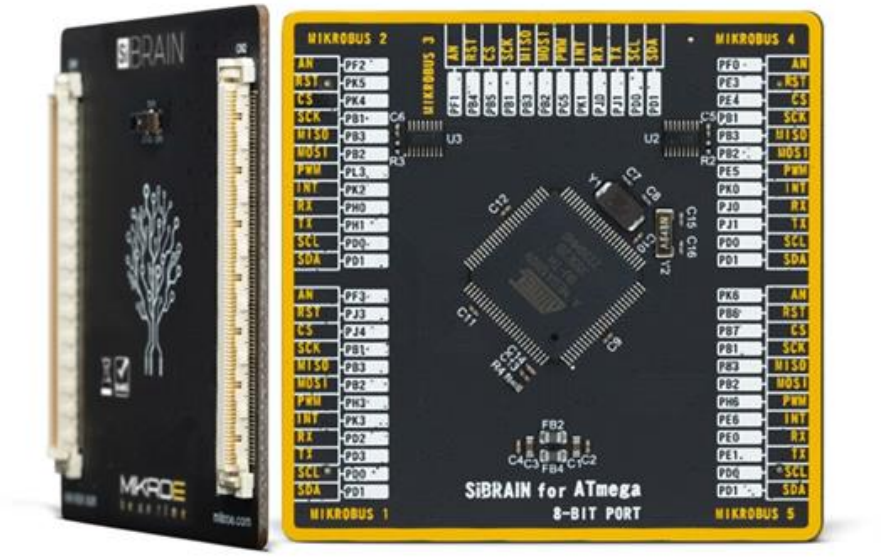


Figura 3.1 - SiBRAIN for ATmega 1280

3.4 PREZENTAREA PLĂCII UNI CLICKER

Placa **UNI Clicker** adaugă **funcționalități suplimentare** prin intermediul conectorilor **mikroBUS**. Aceasta permite conectarea rapidă a modulelor **click** care extind capabilitățile plăcii de bază. Printre funcțiile oferite de această placă se numără:

- **Managementul alimentării** pentru tensiuni de **3.3 V** și **5 V**, esențial pentru integrarea senzorilor și dispozitivelor externe. În plus, placa **UNI Clicker** include un **conector USB** utilizat pentru alimentare și programare, precum și un **conector de debug** care facilitează depanarea firmware-ului în timpul dezvoltării;
- Oferă acces la pini pentru interfețele de comunicare precum **UART**, **SPI**, **I²C** și **PWM**, prin intermediul sloturilor **mikroBUS**. Aceste interfețe sunt utilizate de modulele **click** conectate, care implementează efectiv funcționalitățile dorite.

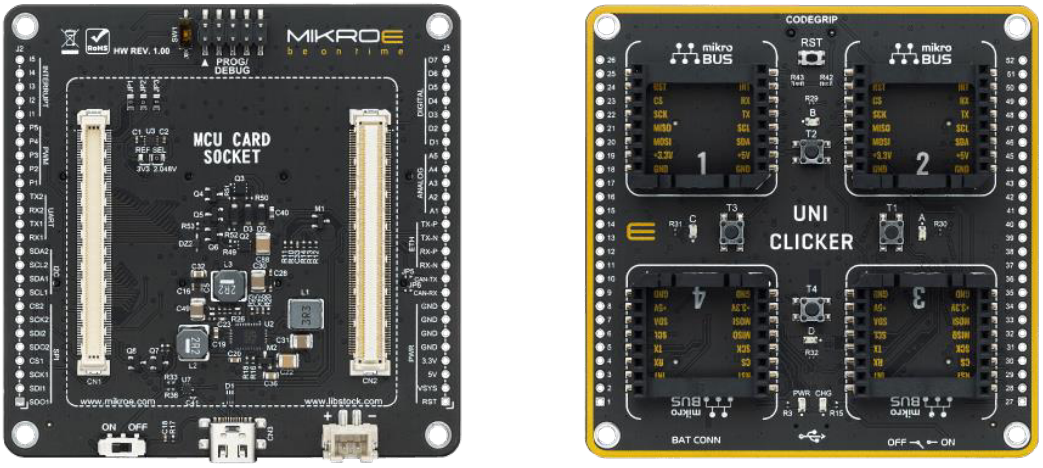


Figura 3.2 – UNI Clicker

3.5 NOȚIUNI DESPRE GPIO – PINI DE INTRARE / IEȘIRE GENERALĂ

3.5.1 DESCRIERE GENERALĂ

Microcontrolerul **ATmega1280** dispune de un număr mare de pini **GPIO** (**General Purpose Input / Output**) care pot fi configurați fie ca **intrări**, fie ca **ieșiri digitale**, în funcție de necesitățile aplicației. Acești pini sunt organizați în mai multe porturi (ex: **PORTA**, **PORTB**, **...**, **PORTL**), fiecare port având **8 biți** (pini numerotați de la **0** la **7**).

Exemple de utilizare:
- **Ieșire digitală**: controlul unui LED, a unui releu sau a unui tranzistor;
- **Intrare digitală**: citirea stării unui buton, a unui senzor digital sau a unui comutator.

Configurarea pinilor **GPIO** se face din firmware folosind registrele:
- **DDRn** (**Data Direction Register**): pentru stabilirea direcției (**intrare** sau **ieșire**);
- **PORTn**: pentru scrierea valorii logice (**HIGH / LOW**) pe pini când sunt configurați ca ieșire;
- **PINn**: pentru citirea stării logice de pe pini când sunt configurați ca intrare.

Acești pini sunt conectați fizic la exterior prin conectorii **CN1** și **CN2** de pe placa **SiBRAIN**, iar apoi, prin placa **UNI Clicker**, pot fi direcționați către sloturile **mikroBUS**, **LED-uri**, **butoane** sau **pini laterali**. Prin urmare, înțelegerea funcționării **GPIO** este **esențială** pentru **dezvoltarea aplicațiilor embedded** cu acest kit.

3.5.2 DESCRIERE PE SCURT A PINILOR ȘI UTILIZAREA LOR FRECVENTĂ

- **VCC – alimentarea digitală a microcontrolerului**

- **GND – masa = 0 V**

- **PortA (PA7...PA0) – Port I/O bidirecțional pe 8 biți**
  Servește și ca intrare analogică pentru **ADC** (**Analog to Digital Converter**). Poate fi configurat individual cu rezistențe de pull-up interne. Buffer-ele de ieșire au caracteristici de amplificare.

- **PortB (PB7...PB0) – Port I/O bidirecțional pe 8 biți cu rezistențe de pull-up interne**
  Poate îndeplini funcții speciale, inclusiv semnale pentru comunicație **SPI** sau semnale **PWM**.

- **PortC (PC7...PC0) – Port I/O bidirecțional pe 8 biți, cu funcții generale sau speciale.**
  Include linii pentru comunicare paralelă și alte funcții alternative.

- **PortD (PD7...PD0) – Port I/O bidirecțional pe 8 biți**
  Are funcții suplimentare ca semnale pentru temporizatoare sau intrări de întrerupere externă.

- **PortE (PE7...PE0) – Port I/O bidirecțional pe 8 biți**
  Include funcții speciale precum **UART** (**RXD**, **TXD**), semnale pentru temporizatoare și întreruperi externe.

- **PortF (PF7...PF0) – Port I/O bidirecțional pe 8 biți**
  Utilizat în principal ca intrare analogică pentru **ADC**. Merge și ca **I/O** general dacă **ADC-ul** nu este utilizat.

- **PortG (PG5...PG0) – Port I/O bidirecțional pe 6 biți**
  Are funcții speciale precum **controlul magistralei externe** sau **generare de semnale PWM**.

- **PortH (PH7...PH0) – Port I/O bidirecțional pe 8 biți**
  Utilizat și pentru funcții speciale precum semnale **PWM** sau **UART** suplimentar.

- **PortJ (PJ7...PJ0) și PortK (PK7...PK0) – Porturi I/O bidirecționale cu 8 biți**
  Utilizate ca **I/O** general sau pentru funcții de **comunicație** și **control**.

- **PortL (PL7...PL0) – Port I/O bidirecțional pe 8 biți**
  Utilizat pentru semnale de control pentru periferice.

- **RESET – Pin de resetare**
  O menținere a nivelului **0 logic** pentru un timp prestabilit declanșează **resetarea** microcontrolerului.

- **XTAL1**
  Intrare pentru oscilatorul extern sau pentru clock-ul intern.

- **XTAL2**
  Ieșire din oscilator.

- **AVCC – Alimentare pentru porturile analogice și pentru ADC**
  Trebuie conectat la **VCC**. Dacă **ADC** este utilizat, se recomandă conectarea printr-un filtru **trece-jos**.

- **AREF**
  Tensiune de referință pentru convertorul **analog-digital**.

- **GND și GND_ANALOG**
  Terminale de masă pentru partea **digitală** și partea **analogică**, respectiv.

- **PE0 / RXD0, PE1 / TXD0**
  Pini dedicați pentru comunicația **UART0**. **ATmega1280** dispune de **4 UART-uri** (**RXD0-3**, **TXD0-3**) disponibile pe diferite porturi.

3.5.3 PORTURILE DE INTRARE / IEȘIRE

Atunci când sunt folosite ca porturi generale digitale **I/O**, toate porturile au funcţionalitate de citire-modificare-scriere ("**True Read-Modify-Write**"). Aceasta înseamnă că, direcţia unui port poate fi schimbată fără schimbarea neintenţionată a direcţiei oricărui alt pin prin instrucţiunile **SBI** şi **CBI**. Toţi pinii porturilor au rezistenţă de pull-up individuală, selectabilă, cu rezerve de putere. Toţi pinii **I/O** au diodă de protecţie atât pentru **VCC** cât şi pentru masă, aşa cum se poate vedea în figura următoare:

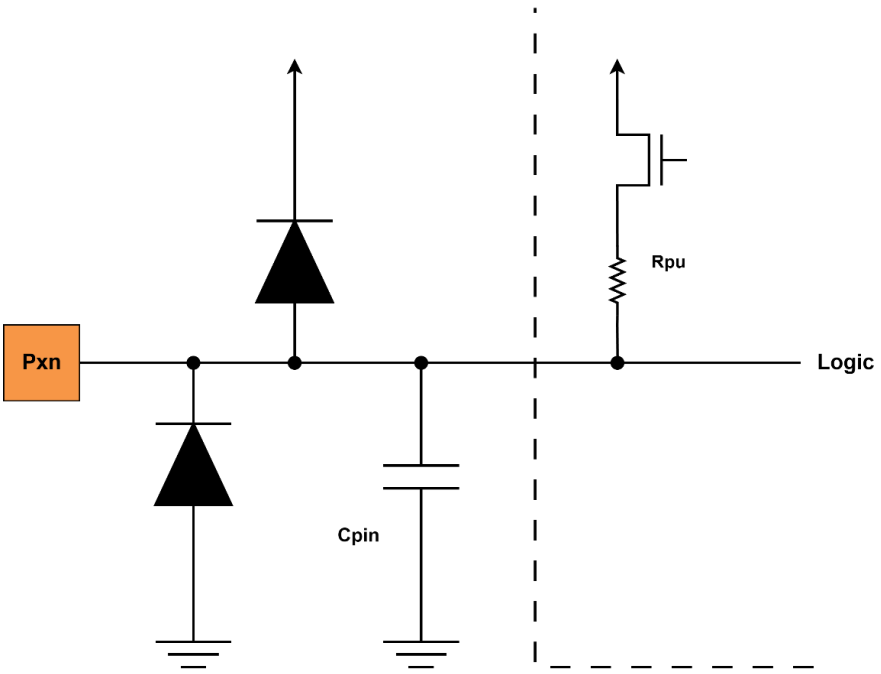


Figura 3.3 – Schema pinilor I/O

Porturile sunt de **I/O bidirecţionale** cu rezistenţe interne de pull-up **opţionale**.

3.5.4 CONFIGURAREA PINILOR

Pentru fiecare port de **I/O**, sunt alocate 3 locații de memorie, sub formă de registre pe **8 biți**:
- **PORTx (Data Register):** Registrul de date;
- **DDRx (Data Direction Register):** Registrul de direcție a datelor;
- **PINx (Port Input Pins):** Registrul de intrare al pinilor portului.

Registrul **PINx** poate fi doar citit, în timp ce **DDRx** și **PORTx** permit atât operații de citire, cât și de scriere. O diferență notabilă la modelele mai vechi era prezența bitului **PUD (Pull-up Disable)** în registrul **SFIOR**. La **ATmega1280**, acest bit a fost mutat în registrul **MCUCR (MCU Control Register)**, dar funcționalitatea sa de a dezactiva **global** rezistențele de pull-up pentru toate porturile rămâne aceeași.

Notația generică pentru acești pini este **DDxn**, **PORTxn** și **PINxn**, unde **"x"** reprezintă litera portului (de ex. A, B, C...), iar **"n"** este numărul bitului (0-7). Într-un program, trebuie folosită denumirea exactă, de exemplu **PORTB3** pentru **bitul 3** al **portului B**.

3.5.4.1 DDRX (DATA DIRECTION REGISTER)

Acest registru stabilește direcția fiecărui pin al portului, fie ca intrare, fie ca ieșire.
- Scrierea unui **0 logic** într-un bit din **DDRx** configurează pinul corespunzător ca **intrare**.
- Scrierea unui **1 logic** într-un bit din **DDRx** configurează pinul corespunzător ca **ieșire**.

Implicit, după un reset, toți pinii sunt configurați ca intrări (**DDRx = 0x00**).

Exemple:
- Pentru a seta **toți** pinii **portului A** ca intrări: **DDRA = 0x00**;
- Pentru a seta **toți** pinii **portului A** ca ieșiri: **DDRA = 0xFF**;
- Pentru a seta pinii **0**, **4**, **5** și **7** ai **portului B** ca ieșiri: **DDRB = 0xB1** (10110001 în binar).

3.5.4.2 PORTX (DATA REGISTER)

Registrul **PORTx** are un rol dublu, în funcție de direcția pinului setată în **DDRx**:
1. Când pinul este configurat ca **ieșire** (**DDxn = 1**):
   Valoarea scrisă în bitul corespunzător din **PORTxn** stabilește nivelul logic al pinului de ieșire.
   - Scrierea unui **1** în **PORTxn** va seta pinul de ieșire în stare logică **HIGH**.
   - Scrierea unui **0** în **PORTxn** va seta pinul de ieșire în stare logică **LOW**.
2. Când pinul este configurat ca **intrare** (**DDxn = 0**):
   Valoarea scrisă în **PORTxn** controlează rezistența de pull-up internă.
   - Scrierea unui **1** în **PORTxn activează** rezistența de pull-up internă pentru acel pin.
   - Scrierea unui **0** în **PORTxn dezactivează** rezistența de pull-up, iar pinul intră în starea de înaltă impedanță (**Hi-Z** sau **tri-state**).

Exemple:
- Setarea pinilor 0, 4, 5 și 7 ai portului B ca ieșiri și setarea lor la nivel logic HIGH:
  - setează pinii ca ieșiri.
    DDRB = 0xB1;
  - setează ieșirile la nivel logic 1
    PORTB = 0xB1;
- Setarea pinilor 0 și 1 ai portului A ca ieșiri și a pinului 2 ca intrare cu pull-up activat:
  - Pinii 0 și 1 ca ieșiri, restul intrări
    DDRA = 0x03;
  - Setează ieșirile la LOW și activează pull-up pentru pinul 2
    PORTA = 0x04;

3.5.4.3 PINX (PORT INPUT PINS)

Acest registru este folosit exclusiv pentru a citi **starea logică** a pinilor unui port, indiferent dacă sunt configurați ca **intrări** sau **ieșiri**. Registrul **PINx** este **read-only**. Citirea acestui registru **returnează** valoarea logică actuală de pe fiecare pin fizic.

Exemplu:

- Pentru a citi valorile pinilor portului C și a le stoca în variabila x:
  - Se setează toți pinii portului C ca intrări
    DDRC = 0x00;
  - Se copiază starea logică a pinilor portului C în variabila x
    x = PINC;

Tabelul de mai jos prezintă starea unui pin pentru diferite combinații de configurare ale registrelor. Bitul **PUD** se află în registrul **MCUCR** la ATmega1280.

DDxn	PORTxn	PUD (în MCUCR)	I/O	Pull-up	Descriere
0	0	X	Intrare	Nu	Tri-state (Hi-Z)
0	1	0	Intrare	Da	Pull-up activat
0	1	1	Intrare	Nu	Hi-Z (pull-up dezactivat global)
1	0	X	Ieșire	Nu	**LOW** (0 logic)
1	1	X	Ieșire	Nu	**HIGH** (1 logic)

Tabelul 3.1 - Starea unui pin pentru diferitele combinații ale regiștrilor

3.5.5 FUNCȚII INTRINSECI

Definiție

Funcțiile intrinseci oferă acces direct la instrucțiuni specifice ale procesorului, fiind extrem de utile pentru operațiuni critice din punct de vedere al timpului sau pentru optimizarea codului. Aceste funcții sunt de obicei compilate inline, adică sunt înlocuite direct cu secvența de instrucțiuni corespunzătoare, fără a implica un apel de funcție.

Funcție Intrinsecă	Descriere
`__delay_cycles(unsigned long)`	Inserează o întârziere precisă, dependentă de numărul de cicluri de ceas specificat.
`__disable_interrupt()`	Dezactivează întreruperile globale (generează instrucțiunea **CLI**).
`__enable_interrupt()`	Activează întreruperile globale (generează instrucțiunea **SEI**).
`__extended_load_program_memory(...)`	Returnează **un octet** din memoria de program. Este utilizată pentru a accesa adrese de memorie **mai mari** de **64 KB**.
`__load_program_memory(...)`	Returnează **un octet** din memoria de program. Se folosește pentru a accesa primii **64 KB** de memorie.
`__no_operation()`	Nu execută nicio operație (generează instrucțiunea **NOP**).
`__sleep()`	Pune microcontrolerul într-un mod de consum redus (generează instrucțiunea **SLEEP**).
`__swap_nibbles(unsigned char)`	Schimbă între ei cei 4 biți **inferiori** cu cei 4 biți **superiori** ai unui octet.
`__watchdog_reset()`	Resetează timer-ul watchdog pentru a preveni resetarea sistemului (generează instrucțiunea **WDR**).

Tabelul 3.2 - Funcțiile intrinseci

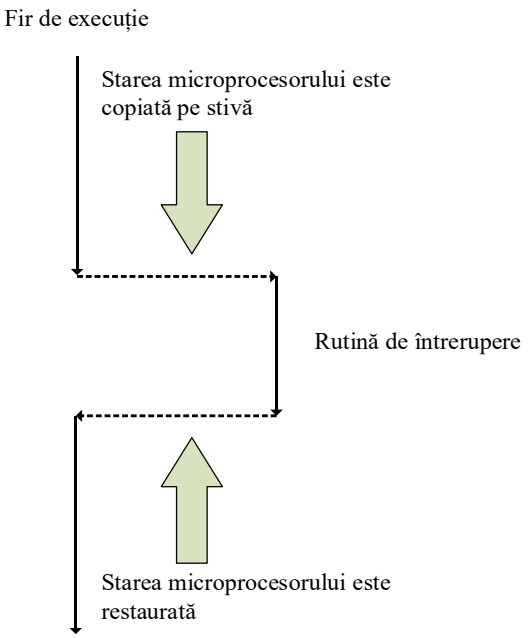


Figura 4.2 – Mecanismul de tratare al întreruperilor la nivel de microprocesor

Pentru a asocia o întrerupere cu o anumită rutină din program, procesorul folosește tabela vectorilor de întrerupere (TVI). Aceste adrese sunt predefinite și sunt mapate în memoria din program într-un spațiu contiguu (consecutiv, fără întreruperi) care alcătuiește TVI. Adresele sunt setate în funcție de prioritatea lor, cu cât adresa este mai mică cu atât prioritatea este mai mare.

Nr. vector	Adresa programului (cuvinte)	Sursa	Definiția întreruperii
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3
6	0x000A	INT4	External Interrupt Request 4
7	0x000C	INT5	External Interrupt Request 5
8	0x000E	INT6	External Interrupt Request 6
9	0x0010	INT7	External Interrupt Request 7
10	0x0012	PCINT0	Pin Change Interrupt Request 0
11	0x0014	PCINT1	Pin Change Interrupt Request 1
12	0x0016	PCINT2	Pin Change Interrupt Request 2
13	0x0018	WDT	Watchdog Time-out Interrupt
14	0x001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	0x001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	0x001E	TIMER2 OVF	Timer/Counter2 Overflow
17	0x0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	0x0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	0x0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	0x0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	0x0028	TIMER1 OVF	Timer/Counter1 Overflow
22	0x002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	0x002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	0x002E	TIMER0 OVF	Timer/Counter0 Overflow
25	0x0030	SPI, STC	SPI Serial Transfer Complete

Tabelul 4.1 – Tabela vectorilor de întrerupere (TVI)

Se observă că **TVI** este mapat începând cu **adresa 0** a memoriei de program. Deși fiecare vector de întrerupere corespunde logic unei instrucțiuni (adică unui cuvânt de **2 octeți**), în practică, adresele sunt exprimate în **octeți**, nu în **cuvinte**. Astfel, fiecare **vector** ocupă **2 octeți** (**1 WORD**), iar adresele din tabel cresc din **2** în **2** (în octeți).

Prioritatea întreruperilor este determinată de ordinea acestora în tabel, astfel: cea **mai mare** prioritate este atribuită întreruperii de **RESET** (**adresa 0x0000**), urmând apoi întreruperea externă **INT0**, apoi restul întreruperilor în ordinea crescătoare a adreselor. Perifericele care pot genera întreruperi în cazul microcontrolerului **ATmega1280** includ: temporizatoare (**Timer0**, **Timer1**, etc.), interfața serială **USART**, convertorul analog-digital **ADC**, **EEPROM** (la finalizarea scrierii), comparatorul analogic, interfața serială **TWI**.

4.7 ACTIVAREA / DEZACTIVAREA ÎNTRERUPERILOR

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.3 – Registrul de stare (SREG)

O întrerupere este dezactivată daca 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 ATmega1280, bitul ce se referă la toate întreruperile se numește **I** și se află în registrul de stare al microprocesorului (bitul 7 din **Figura 4.3**). Pentru a-l modifica se pot utiliza următoarele instrucțiuni:

Mnemonică	Descriere	Operație	Număr de cicli	Funcții IAR
SEI	Global Interrupt Enable	I = 1	1	**__enable_interrupt()**
CLI	Global Interrupt Disable	I = 0	1	**__disable_interrupt()**

Tabelul 4.2 – Instrucțiunile de modificare ale bitului I

Instrucțiunile prezentate în tabel sunt utilizate pentru gestionarea întreruperilor la nivel global în cadrul arhitecturii AVR. Mnemonicile SEI (**Set Interrupt Enable**) și CLI (**Clear Interrupt Enable**) sunt comenzi în limbaj de asamblare care modifică bitul **I** (**Interrupt Enable**) din registrul de stare **SREG**, activând (I ← 1) sau dezactivând (I ← 0) procesarea întreruperilor. Aceste comenzi sunt executate într-un singur ciclu de ceas, fiind esențiale în controlul secvențelor critice de cod unde este necesară blocarea temporară a întreruperilor.

Funcțiile **`__enable_interrupt()`** și **`__disable_interrupt()`** oferă echivalentul în limbaj de programare C, specifice compilatoarelor precum IAR Embedded Workbench sau AVR-GCC, asigurând un mod portabil și lizibil de gestionare a întreruperilor globale în aplicații embedded. Activarea întreruperilor globale prin SEI nu are efect dacă întreruperile individuale nu sunt activate în registrele de control corespunzătoare (ex. EIMSK, PCICR). Astfel, utilizarea corectă a acestor instrucțiuni este esențială pentru sincronizarea precisă între perifericele controlate prin întreruperi.

Pentru activarea unei întreruperi externe pe microcontrolerul ATmega1280, este necesară configurarea corespunzătoare a registrelor de control implicate. În cazul întreruperii externe **INT0**, aceasta poate fi setată să răspundă la diverse condiții ale semnalului aplicat pe pinul **INT0**, inclusiv front crescător, front descrescător, nivel logic scăzut sau tranziție logică *(***toggle**). Modul de declanșare este determinat de valorile setate în biții **ISC01** și **ISC00** din registrul **EICRA**. Primul pas constă în configurarea modului de declanșare prin registrul **EICRA** (**External Interrupt Control Register A**), unde biții **ISC01** și **ISC00** determină tipul de tranziție ce va genera întreruperea. Pentru declanșare pe front crescător, ambii biți trebuie setați la **1**.

Apoi, întreruperea trebuie activată la nivel individual prin setarea bitului corespunzător din registrul EIMSK (**External Interrupt Mask Register**). Pentru **INT0**, acesta este bitul **0**. În cele din urmă, este necesară activarea întreruperilor la nivel global, folosind funcția **`__enable_interrupt()`**, care corespunde instrucțiunii SEI în limbajul de asamblare. Acest pas permite procesorului să răspundă la întreruperi.

`int0_avr.c`

Fragmentul de cod alăturat reprezintă un exemplu de configurare a unei întreruperi externe (INT0) pe AVR. Programul setează ca întreruperea să fie declanșată la front crescător al semnalului și activează întreruperile globale. Bucla infinită **`while(1)`** permite procesorului să aștepte evenimente externe.

```
/*-------------------------------------------------------------------------
 * Fișier: int0_avr.c
 * Utilizat pentru exemplificarea configurării întreruperii externe INT0
 *-------------------------------------------------------------------------*/

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

int main( void )
{
  // Configurarea întreruperii externe INT0 la frontul crescător
  EICRA |= (1<< ISC01) | (1 << ISC00);
  // Activarea întreruperii externe INT0
  EIMSK |= (1<< INT0);
  // Activarea întreruperilor globale
  __enable_interrupt();
  while(1)
  {
    /*
     * Buclă infinită pentru blocarea procesorului până la apariția unei
     * întreruperi
     */
  }
}
```

Se remarcă faptul că ultimul pas din procedura de inițializare este activarea întreruperilor globale, ceea ce respectă o ordine **bottom-up** în configurare: întâi se configurează evenimentul, apoi sursa, și la final se permite procesorului să răspundă global.

Fiecare tip de întrerupere este asociat cu un bit de stare care este setat de hardware atunci când apare evenimentul corespunzător. Pentru INT0, bitul respectiv este INTF0, aflat în registrul EIFR (**External Interrupt Flag Register**), pe poziția **0**. Resetarea acestui bit se face scriind valoarea **1** în poziția respectivă.

Notă: Contraintuitiv, resetarea bitului INTF0 se face scriind valoarea 1, nu 0.

Întreruperile au o prioritate determinată de poziția lor în tabela vectorilor de întrerupere. În cazul în care mai multe întreruperi sunt active simultan, cea cu **adresa de vector mai mică** va fi tratată prima. Funcțiile de tratare a întreruperilor nu pot fi întrerupte de altele cu prioritate mai mare în timpul execuției lor, decât dacă se activează manual acest comportament (prin nesting sau reentrancy). Se spune că o întrerupere este **reentrantă** dacă poate fi întreruptă de o alta cu prioritate superioară în timp ce ea este în execuție.

O rutină de întrerupere trebuie să se termine cu instrucțiunea RETI (**Return from Interrupt**), care restabilește starea anterioară a execuției și reia programul din punctul în care a fost întrerupt. Aceasta funcționează deoarece adresa curentă este salvată automat pe stivă de către microcontroler, înainte de saltul către vectorul de întrerupere.

4.8 REGISTRE PENTRU TRATAREA ÎNTRERUPERILOR EXTERNE

4.8.1 REGISTRUL EICRA

Bit	7	6	5	4	3	2	1	0	
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.4 – Registrul de control al întreruperilor externe A

Acest registru conține biții de control pentru configurarea sensibilității întreruperilor.

Biții **ISC01** și **ISC00** din registrul **EICRA** controlează sensibilitatea întreruperii externe **INT0**. Aceasta este activată atunci când bitul **I** din registrul **SREG** și masca corespunzătoare din **EIMSK** sunt setate. Nivelul logic sau tranzițiile de pe pinul **INT0** care declanșează întreruperea sunt definite conform tabelului de mai jos. Semnalul este eșantionat înainte de a fi evaluat, iar pentru modurile de tip front sau toggle, doar impulsurile care durează mai mult de o perioadă de ceas sunt sigure pentru a genera o întrerupere. Dacă se selectează modul „nivel logic 0", semnalul trebuie menținut pe **0** până la finalizarea execuției instrucțiunii curente pentru ca întreruperea să fie declanșată.

ISC01	ISC00	Descriere
0	0	Nivelul 0 logic pe INT0 generează o cerere de întrerupere.
0	1	Orice schimbare logică pe INT0 generează o cerere de întrerupere.
1	0	Pe frontul negativ al lui INT0 se generează o cerere de întrerupere.
1	1	Pe frontul pozitiv al lui INT0 se generează o cerere de întrerupere.

Tabelul 4.3 – Stările de declanșare ale întreruperii externe INT0

4.8.2 REGISTRUL EIMSK

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.5 – Registrul de mascare a întreruperilor externe

Acest registru este răspunzător pentru activarea și dezactivarea întreruperilor externe.

- **Bitul 0 – INT0: Activare cerere de Întrerupere Externa 0**
  Când bitul **INT0** este setat (valoare logică 1) și bitul **I** din **SREG** este setat (valoare logică 1), întreruperea externă a pinului este activată. Biții de control al sensului întreruperii 0 / 1 (**ISC01** și **ISC00**) din registrul de control al întreruperilor externe A (**EICRA**) definesc dacă întreruperea externă este activată la frontul pozitiv și / sau negativ al pinului **INT0** sau la schimbare de nivel. Activarea pe pin va genera o cerere de întrerupere chiar dacă **INT0** este configurat ca pin de ieșire. Întreruperea corespunzătoare a cererii de întrerupere **0** este executate din vectorul de întrerupere **INT0**.

- **Bitul 4 – PCIE0: Bitul de activare întrerupere la schimbarea pinului 0**
  Când bitul **PCIE0** este setat (valoare logică 1) și bitul **I** din **SREG** este setat (valoare logică 1), întreruperea la schimbarea pinului **0** este activată. Orice schimbare pe oricare dintre pinii **PCINT7:0** activați va genera o întrerupere. Întreruperea corespunzătoare a cererii de întrerupere la schimbarea pinului este executată din vectorul de întrerupere **PCINT0**. Pinii **PCINT7:0** sunt activați individual prin registrul **PCMSK0**.

- **Bitul 5 – PCIE1: Activare întrerupere la schimbarea pinului 1**
  Când bitul **PCIE1** este setat (valoare logică 1) și bitul **I** din **SREG** este setat (valoare logică 1), întreruperea la schimbarea pinului **1** este activată. Orice schimbare pe oricare dintre pinii **PCINT15:8** activați va genera o întrerupere. Întreruperea corespunzătoare a cererii de întrerupere la schimbarea pinului este executată din vectorul de întrerupere **PCINT1**. Pinii **PCINT15:8** sunt activați individual prin registrul **PCMSK1**.

- **Bitul 6 – PCIE2: Activare întrerupere la schimbarea pinului 2**
  Când bitul **PCIE2** este setat (valoare logică 1) și bitul **I** din **SREG** este setat (valoare logică 1), întreruperea la schimbarea pinului **2** este activată. Orice schimbare pe oricare dintre pinii **PCINT23:16** activați va genera o întrerupere. Întreruperea corespunzătoare a cererii de întrerupere la schimbarea pinului este executată din vectorul de întrerupere **PCINT2**. Pinii **PCINT23:16** sunt activați individual prin registrul **PCMSK2**.

- **Bitul 7 – PCIE3: Activare întrerupere la schimbarea pinului 3**
  Când bitul **PCIE3** este setat (valoare logică 1) și bitul **I** din **SREG** este setat (valoare logică 1), întreruperea la schimbarea pinului **3** este activată. Orice schimbare pe oricare dintre pinii **PCINT30:24** activați va genera o întrerupere. Întreruperea corespunzătoare a cererii de întrerupere la schimbarea pinului este executată din vectorul de întrerupere **PCINT3**. Pinii **PCINT30:24** sunt activați individual prin registrul **PCMSK3**.

4.8.3 REGISTRUL EIFR

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.6 – Registrul de indicatori ai întreruperilor externe

- **Bit 0 – INTF0: Indicator al Întreruperii Externe 0**
  Când o tranziție sau o schimbare logică pe pinul **INT0** declanșează o cerere de întrerupere, bitul **INTF0** se setează (**1**), microcontrolerul va sări la vectorul de întrerupere corespunzător. Indicatorul este resetat când rutina de întrerupere este executată. Alternativ, indicatorul poate fi resetat prin scrierea unei valorii de 1 logic în acesta. Acest indicator este întotdeauna resetat când **INT0** este configurat ca întrerupere pe nivel.

- **Bit 4 – PCIF0: Indicator al Întreruperii la Schimbarea Pinului 0**
  Când o schimbare logică pe oricare dintre pinii **PCINT7:0** declanșează o cerere de întrerupere, bitul **PCIF0** se setează (**1**). Dacă bitul **I** din **SREG** și bitul **PCIE0** din **EIMSK** sunt setați (**1**), microcontrolerul va sări la vectorul de întrerupere corespunzător. Indicatorul este resetat când rutina de întrerupere este executată. Alternativ, indicatorul poate fi resetat prin scrierea valorii de 1 logic în acesta.

- **Bit 5 – PCIF1: Indicator al Întreruperii la Schimbarea Pinului 1**
  Când o schimbare logică pe oricare dintre pinii **PCINT15:8** declanșează o cerere de întrerupere, bitul **PCIF1** se setează (**1**). Dacă bitul **I** din **SREG** și bitul **PCIE1** din **EIMSK** sunt setați (**1**), microcontrolerul va sări la vectorul de întrerupere corespunzător. Indicatorul este resetat când rutina de întrerupere este executată. Alternativ, indicatorul poate fi resetat prin scrierea valorii de 1 logic în acesta.

- **Bit 6 – PCIF2: Indicator al Întreruperii la Schimbarea Pinului 2**
  Când o schimbare logică pe oricare dintre pinii **PCINT23:16** declanșează o cerere de întrerupere, bitul **PCIF2** se setează (**1**). Dacă bitul **I** din **SREG** și bitul **PCIE2** din **EIMSK** sunt setați (**1**), microcontrolerul va sări la vectorul de întrerupere corespunzător. Indicatorul este resetat când rutina de întrerupere este executată. Alternativ, indicatorul poate fi resetat prin scrierea valorii de 1 logic în acesta. Acest bit este rezervat în ATmega1280 și va fi citit întotdeauna ca zero.

- **Bit 7 – PCIF3: Indicator al Întreruperii la Schimbarea Pinului 3**
  Când o schimbare logică pe oricare dintre pinii **PCINT30:24** declanșează o cerere de întrerupere, bitul **PCIF3** se setează (**1**). Dacă bitul **I** din **SREG** și bitul **PCIE3** din **EIMSK** sunt setați (**1**), microcontrolerul va sări la vectorul de întrerupere corespunzător. Indicatorul este resetat când rutina de întrerupere este executată. Alternativ, indicatorul poate fi resetat prin scrierea valorii de 1 logic în acesta.

4.8.4 REGISTRUL PCMSK0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.7 – Registrul de mască pentru schimbarea pinului 0

- **Biții 7:0 – PCINT7:0: Mască de Activare a Întreruperii la Schimbarea Pinului 7:0**
  Fiecare bit **PCINT7:0** selectează dacă întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT7:0** este setat și bitul **PCIE0** din **EIMSK** este setat, întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT7:0** este șters, întreruperea la schimbarea pinului pe pinul de **I/O** corespunzător este dezactivată.

4.8.5 REGISTRUL PCMSK1

Bit	7	6	5	4	3	2	1	0	
(0x6C)	PCINT15	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.8 – Registrul de mască pentru schimbarea pinului 1

- **Biții 7:0 – PCINT15:8: Mască de Activare a Întreruperii la Schimbarea Pinului 15:8**
  Fiecare bit **PCINT15:8** selectează dacă întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT15..8** este setat și bitul **PCIE1** din **EIMSK** este setat, întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT15:8** este șters, întreruperea la schimbarea pinului pe pinul de **I/O** corespunzător este dezactivată.

4.8.6 REGISTRUL PCMSK2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 4.9 – Registrul de mască pentru schimbarea pinului 2

- **Biții 7:0 – PCINT23:16: Mască de Activare a Întreruperii la Schimbarea Pinului 23..16**
  Fiecare bit **PCINT23:16** selectează dacă întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT23:16** este setat și bitul **PCIE2** din **EIMSK** este setat, întreruperea la schimbarea pinului este activată pe pinul de **I/O** corespunzător. Dacă bitul **PCINT23:16** este șters, întreruperea la schimbarea pinului pe pinul de **I/O** corespunzător este dezactivată.

4.8.7 ANALIZA COMPARATIVĂ A ÎNTRERUPERILOR INTX VS PCINTX

În microcontrolerul ATmega1280, gestionarea evenimentelor externe se poate face prin două tipuri principale de întreruperi: întreruperile externe **INTx** și întreruperile la schimbare de pin **PCINTx**. Deși ambele permit detectarea modificărilor de semnal pe pini de intrare, ele diferă semnificativ ca funcționalitate, flexibilitate și mod de configurare.

Întreruperile **INTx** (de la **INT0** la **INT7**) sunt asociate cu pini dedicați și permit configurarea sensibilității la front ascendent, front descendent, schimbare logică sau nivel logic scăzut. Acestea oferă un control mai precis al detecției și sunt adesea utilizate în aplicații critice de timp real.

Pe de altă parte, întreruperile **PCINTx** (**Pin Change Interrupt**) sunt mai flexibile din punct de vedere al acoperirii, putând fi configurate pentru majoritatea pinilor de **I/O**, dar reacționează doar la orice schimbare logică (fie de la 0 la 1, fie de la 1 la 0), fără posibilitatea de a selecta tipul de tranziție. **PCINTx** sunt utile atunci când este necesară monitorizarea simultană a mai multor pini fără a consuma resurse suplimentare.

Prin urmare, alegerea între **INTx** și **PCINTx** depinde de cerințele aplicației, nivelul de precizie necesar în detecția evenimentelor și disponibilitatea pinilor corespunzători.

4.9 LATENȚĂ, JITTER ȘI TIMP DE RĂSPUNS LA ÎNTRERUPERI

4.9.1 LATENȚĂ

Definiție

Latența unei întreruperi reprezintă timpul scurs între momentul în care sursa de întrerupere solicită serviciu (prin activarea unei condiții de întrerupere) și momentul în care microcontrolerul începe execuția efectivă a rutinei de tratare (ISR – Interrupt Service Routine).

Latența este influențată de următorii factori:
- instrucțiunea curentă aflată în execuție (în special dacă este una multi-ciclu),
- activarea globală a întreruperilor (**SREG.I = 1**),
- prioritățile relative ale întreruperilor multiple aflate în așteptare.

Conform documentației oficiale (datasheet ATmega1280), **răspunsul minim la o întrerupere necesită 5 cicluri de ceas**, timp în care:
- program counter-ul este salvat pe stivă,
- execuția sare către adresa vectorului de întrerupere.

Această adresă vector indică de regulă un salt (**jmp**) către rutina de întrerupere, care mai consumă **încă 3 cicluri de ceas**, rezultând un total minim de **8 cicluri pentru intrarea efectivă în ISR**. Dacă întreruperea apare în timpul unei instrucțiuni multi-ciclu, aceasta este finalizată complet înainte de declanșarea ISR-ului.

În cazul în care microcontrolerul se află într-un mod de stand-by / sleep, timpul de răspuns este mărit cu încă **5 cicluri de ceas**, la care se adaugă și timpul specific de „wake-up” corespunzător modului de somn selectat.

La revenirea din întrerupere (care durează **5 cicluri de ceas**), procesorul:
- restaurează adresa salvată (3 octeți pentru PC),
- incrementează SP corespunzător,
- reactivează **flag-ul** global de întreruperi (setează bitul **I** din **SREG**).

Configurațiile software precum activarea / dezactivarea întreruperilor globale sau utilizarea modurilor de repaus pot introduce variații în latență, afectând astfel **jitter-ul** sistemului.

5.7 TIPURI DE COMUNICAȚII SERIALE

Din punctul de vedere al direcţiei de transfer, se pot distinge următoarele tipuri de comunicaţie serială:
- Simplex;
- Semiduplex;
- Duplex.

În cazul comunicaţiei **simplex**, datele sunt transferate întotdeauna în aceeaşi direcţie, de la echipamentul transmițător la cel receptor. La comunicaţia **semiduplex**, fiecare echipament terminal de date funcţionează 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 comunicaţie **duplex** (numită şi **duplex integral**), datele se transferă simultan în ambele direcţii. 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 şi receptor, există două tipuri de comunicaţie serială:
- Asincronă (**UART**);
- Sincronă (**USART**).

5.7.1 COMUNICAȚIA ASINCRONĂ – UART

Pentru a asigura sincronizarea dintre transmițător şi receptor, fiecare caracter transmis este precedat de un bit de **START**, cu valoarea logică **0** (**"space"**) şi este urmat de cel puţin un bit de **STOP**, cu valoarea logică **1** (**"mark"**).

Biţii de **START** şi de **STOP** încadrează fiecare caracter transmis; caracterul transmis între aceşti 2 biţi reprezintă un cadru de date. Un asemenea cadru reprezintă informaţia digitală de bază într-un sistem de comunicaţie serială. În cazul comunicaţiei asincrone, intervalul de timp între transmisia a 2 caractere succesive este variabil, pe durata acestui interval linia de comunicaţie fiind în starea **1 logic**. Acest mod de comunicaţie este numit şi **START-STOP**.

Sincronizarea la nivel de bit se realizează cu ajutorul semnalelor de ceas locale cu aceeaşi frecvenţă. Atunci când receptorul detectează începutul unui caracter indicat prin bitul de **START**, porneşte un oscilator de ceas local, care permite eşantionarea corectă a biţilor individuali ai caracterului. Eşantionarea biţilor se realizează aproximativ la mijlocul intervalului corespunzător fiecărui bit.

Figura 5.3 ilustrează transmisia caracterului cu codul ASCII **0x61**. După bitul de **START**, având durata T corespunzătoare unui bit, transmisia caracterului începe cu bitul cel mai puţin semnificativ (**b0**). După transmisia bitului cel mai semnificativ (**b8**), se transmite un bit de paritate **p**; în acest exemplu, paritatea este **impară**. Bitul de paritate este opţional, iar în cazul în care se adaugă la caracterul transmis, paritatea poate fi selectată pentru a fi pară sau impară. Există şi posibilitatea ca bitul de paritate să fie setat la **0** sau **1**, indiferent de paritatea efectivă a caracterului. În exemplul ilustrat, la sfârşitul caracterului se transmit doi biţi de **STOP s1** şi **s2**, după care linia rămâne în starea **1 logic** un timp nedefinit. Acest timp corespunde unui interval de pauză.

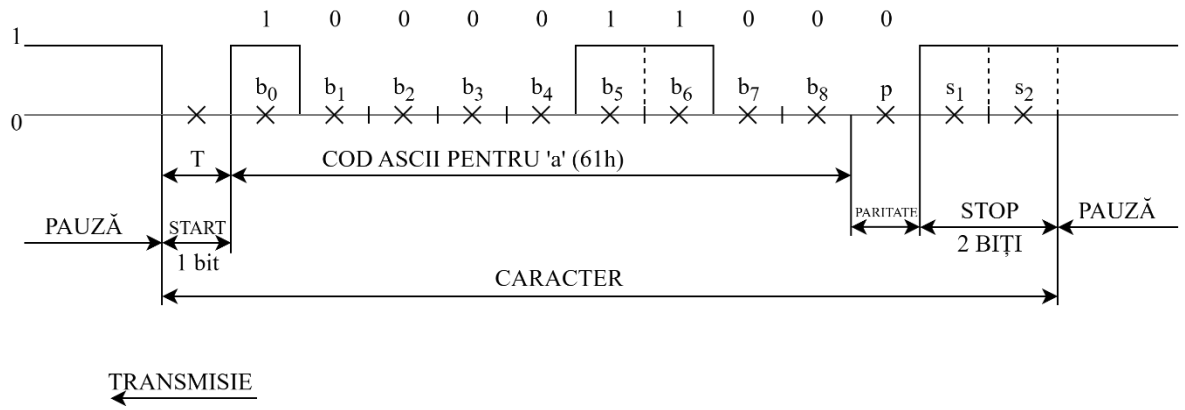


Figura 5.3 – Comunicația asincronă

În cazul comunicaţiei asincrone, sincronizarea la nivel de bit este asigurată numai pe durata transmisiei efective a fiecărui caracter. O asemenea comunicaţie este orientată pe caractere individuale şi are dezavantajul că necesită informaţii suplimentare în proporţie de cel puţin 25% pentru identificarea fiecărui caracter.

5.7.2 COMUNICAȚIA SINCRONĂ – USART

În cazul comunicaţiei sincrone, un cadru nu conţine 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 şi 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 frecvenţa transmițătorului, vor apare erori la recunoaşterea 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 tranziţii de la **1** la **0** şi de la **0** la **1** în mesajul transmis. Dacă datele de transmis constau din şiruri lungi de **1** sau de **0**, trebuie inserate tranziţii suficiente pentru resincronizarea ceasurilor. Asemenea tehnici sunt dificil de implementat, astfel încât se utilizează de obicei o tehnică numită **comunicaţie asincronă sincronizată** (numită în mod simplu comunicaţie sincronă).

Acest tip de comunicaţie este caracterizat de faptul că, deşi mesajul este transmis într-un mod sincron, nu există o sincronizare în intervalul de timp dintre două mesaje. Informaţia este transmisă sub forma unor blocuri de caractere sau a unor biţi succesivi, fără biţi de **START** şi **STOP**. Pentru ajustarea oscilatorului local la începutul unui mesaj, fiecare mesaj este precedat de un număr de caractere speciale de sincronizare, de exemplu, caracterul **SYN** (**0x16**). Pentru menţinerea 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 tranziţiilor existente în semnalul recepţionat;
- Sincronizare la nivel de caracter, asigurată prin recunoaşterea anumitor caractere de sincronizare;
- Sincronizare la nivel de bloc sau mesaj, care depinde de protocolul de date utilizat.

5.7.3 STANDARDUL RS-232C

Specificaţiile 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 **Asociaţia Industriei Electronice** (**EIA – Electronic Industries Association**). Standardul a fost elaborat pentru comunicaţia digitală între un calculator şi un terminal aflat la distanţă 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 comunicaţie.

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** şi **Asociaţia Industriei de Telecomunicaţii** (**TIA – Telecommunications Industry Association**) au elaborat revizia **E** a standardului (**EIA/TIA RS-232E**). Revizia curentă este **EIA RS-232F**, publicată în anul 1997. Totuşi, 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 comunicaţie, nivelele de tensiune, protocolul utilizat pentru controlul fluxului de date şi conectorii interfeţei seriale.

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

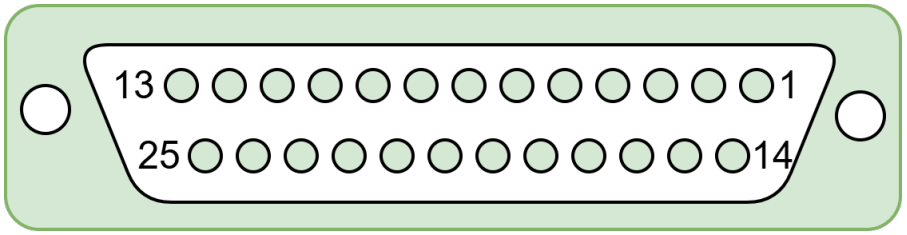


Figura 5.4 – Conectorul DB-25 male (tată)

Din cele 25 de semnale ale conectorului **DB-25**, se utilizează cel mult 10 semnale pentru o conexiune serială obişnuită. **Tabelul 5.1** indică numele acestor semnale şi 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	DCD	Data Carrier Detect	→
20	DTR	Data Terminal Ready	←
22	RI	Ring Indicator	←

Tabelul 5.1 – Semnalele portului

Pentru a se reduce spaţiul ocupat de conectorul portului serial, conectorul DB-25 a fost înlocuit cu un conector de dimensiuni mai reduse, și anume conectorul cu 9 pini **DB-9** (Figura 1.5).

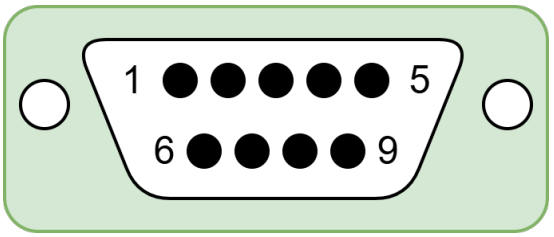


Figura 5.5 – Conectorul DB-9 female (mamă)

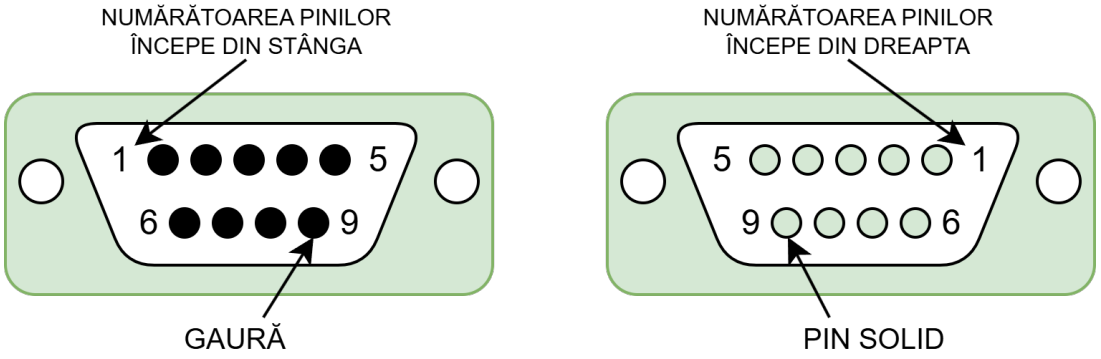


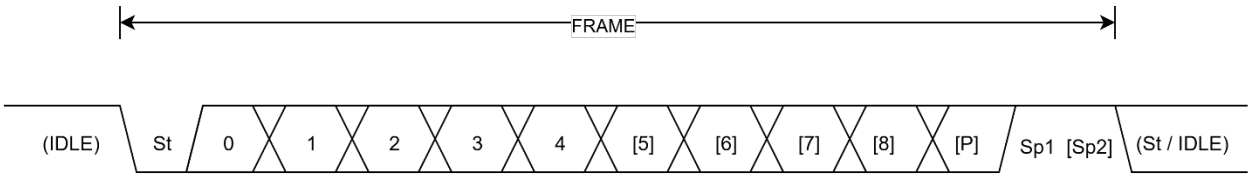
Figura 5.6 – Dierențele între conectorul mamă (stânga) și conectorul tată (dreapta)

5.11.3 FORMAT FRAME-URI

Definiție

Un frame serial reprezintă un caracter transmis, împreună cu biții de control. Acesta conține:

1. **Un bit de START: întotdeauna 0 logic.**
2. **Biți de date: între 5 și 9 biți.**
3. **Un bit de paritate (opțional): Pentru detecția erorilor.**
4. **Biți de STOP: Unul sau doi biți, întotdeauna 1 logic.**



St	Start bit, always low
(n)	Data bits (0 to 8)
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high
IDLE	No transfers on the comunication line (RxDn or TxDn). An IDLE line must be high

Figura 5.15 - Frame Format

5.11.4 INIȚIALIZAREA USART

Pentru a inițializa modulul **USART** trebuie urmați următorii pași:

1. Setarea **Baud Rate-ului**: se calculează valoarea **UBRR** și se încarcă registrele **UBRRH** și **UBRRL**.
2. Setarea **Formatului de Cadru**: se configurează numărul de biți de date, biți de paritate și biții de stop în registrul **UCSRC**.
3. Activarea Transmițătorului și / sau a Receptorului: se setează biții **TXEN** și / sau **RXEN** din reg. **UCSRB**.

5.12 DESCRIEREA REGIȘTRILOR

5.12.1 UDR – USART I/O DATA REGISTER

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

Figura 5.16 – Registrul UDR

Este un registru pe 8 biți care servește ca buffer atât pentru transmisie, cât și pentru recepție:

- La scriere, datele sunt puse în buffer-ul de transmisie (**TXB**).
- La citire, se accesează datele din buffer-ul de recepție (**RXB**).

5.12.2 UCSRNA – USART CONTROL AND STATUS REGISTER A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial value	0	0	1	0	0	0	0	0	

Figura 5.17 – Registrul UCSRnA

- **RXCn (Bitul 7): Receive Complete**
  Acest flag este 1 când în buffer-ul de receptive există date necitite. Poate genera o întrerupere.

- **TXCn (Bitul 6): Transmit Complete**
  Acest flag este 1 când un frame a fost transmis și buffer-ul de transmisie este gol. Poate genera o întrerupere.

- **UDREn (Bitul 5): USART Data Register Empty**
  Acest flag este **1** când buffer-ul de transmisie este gol și gata să primească un nou caracter pentru a fi transmis. Poate genera o întrerupere.

- **FEn (Bitul 4): Frame Error**
  Setat pe **1** dacă este detectată o eroare la un frame (ex: bitul de stop este 0).

- **DORn (Bitul 3): Data OverRun**
  Setat pe **1** dacă o suprascriere de date are loc în buffer-ul de receptive.

- **UPEn (Bitul 2): Parity Error**
  Setat pe **1** dacă este detectată o eroare de paritate.

- **U2Xn (Bitul 1): Double the USART Transmission Speed**
  Când este setat pe **1** se dublează viteza de transfer în modul asincron.

- **MPCMn (Bitul 0): Multi-Processor Comunication Mode**
  Activează modul de comunicare multi-procesor.

5.12.3 UCSRNB – USART CONTROL AND STATUS REGISTER B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial value	0	0	0	0	0	0	0	0	

Figura 5.18 – Registrul UCSRnB

- **RXCIEn (Bit 7): RX Complete Interrupt Enable**
  Activează întreruperea pe flag-ul **RXC**.

- **TXCIEn (Bit 6): TX Complete Interrupt Enable**
  Activează întreruperea pe flag-ul **TXC**.

- **UDRIEn (Bit 5): USART Data Register Empty Interrupt Enable**
  Activează întreruperea pe flag-ul **UDRE**.

- **RXENn (Bit 4): Receiver Enable**
  Activează receptorul **USART**.

- **TXENn (Bit 3): Transmitter Enable**
  Activează transmițătorul **USART**.

- **UCSZn2 (Bit 2): Character Size**
  Folosit împreună cu biții **UCSZ1:0** pentru a seta numărul de biți de date (5-9).

- **RXB8n (Bit 1): Receive Data Bit 8**
  Al 9-lea bit de date recepționat (în modul 9 biți de date).

- **TXB8 (Bit 0): Transmit Date Bit 8**
  Al 9-lea bit de date de transmis (în modul 9 biți de date).

5.13 UCSRNC – USART CONTROL AND STATUS REGISTER C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	1	1	0	

Figura 5.19 – Registrul UCSRnC

- **UMSELn1:0 (Bit 7:6): USART Mode Select**
  Setează modul de operare după următorul tabel:

UMSELn1	UMSELn0	Mode
0	0	USART asincron
0	1	USART sincron
1	0	Reserved
1	1	Master SPI

Tabelul 5.3 – Configurarea modului USART

- **UPMn1:0 (Bit 5:4): Parity Mode**
  Setează modul de paritate după următorul tabel:

UPMn1	UPMn0	Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Tabelul 5.4 – Configurarea modului de paritate

- **USBSn (Bit 3): Stop Bit Select**
  Setează numărul de biți de stop:

USBSn	Stop Bit(s)
0	1-bit
1	2-bit

Tabelul 5.5 – Configurarea numărului de biți de stop

- **UCSZn1:0 (Bit 2:1): Character Size**
  Folosiți împreună cu UCSZ2 pentru a seta numărul de biți de date:

UCSZn2	UCSZn1	UCSZn0	Character size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Tabelul 5.6 – Configurarea numărului de biți de date

- **UCPOLn (Bit 0): Clock Polarity**
  Folosit doar în modul sincron pentru a defini polaritatea clock-ului:

UCPOLn	Transmited Data Changed (output of TxDn Pin)	Received Data Sampled (input on RxDn Pin)
0	Rising XCKn Edge	Falling XCKn Edge
1	Falling XCKn Edge	Rising XCKn Edge

Tabelul 5.7 – Configurarea parității clock-ului

5.14 PROBLEME

5.14.1 APLICAȚIE ECHO

Se cere implementarea unei aplicații software care realizează funcționalitatea de **echo** prin intermediul unei comunicații seriale **USART**. Aplicația trebuie să primească un **caracter transmis** de la PC sau alt dispozitiv conectat pe interfața serială și să îl retransmită imediat către sursă, asigurând astfel afișarea lui înapoi pe terminal.

Sugestii: *Configurarea presupune inițializarea vitezei de comunicație (**baud rate**) conform formulei standard, activarea receptorului și transmițătorului serial și configurarea pinilor corespunzători **TX** și **RX** (**TX** ca **ieșire**, **RX** ca **intrare**). Transmiterea și recepția datelor se realizează în mod blocant, prin verificarea flag-urilor din registrele de stare: pentru transmitere, se așteaptă eliberarea buffer-ului de trimitere (**UDRE3 = 1**), iar pentru recepție, se așteaptă sosirea unui caracter (**RXC3 = 1**). Astfel, fiecare caracter primit este preluat din registrul **UDR3** și retransmis imediat prin același registru.*

Conectări hardware: *Conectarea se face pe portul **microBUS 3**, unde pinii **TXD** și **RXD** ai microcontrolerului sunt legați la interfața de comunicație serială a plăcii. În cazul testării cu PC-ul, se poate folosi un cablu **USB-UART** pentru a deschide un terminal serial la **9600 baud**. Conectarea se poate face, la voia studentului, pe orice alt microBUS, cu condiția modificării pinilor aferenți microBUS-ului respectiv.*

6.5 NOȚIUNI INTRODUCTIVE

Definiții

Frecvența este numărul de apariții ale unui eveniment repetitiv pe unitatea de timp. Perioada este timpul necesar pentru a completa un ciclu al unei oscilații sau rotații. Relația dintre frecvență și perioadă este:

$$T = \frac{1}{\text{frecvență}}$$

Factorul de umplere (duty cycle) este valoarea care indică cât din întreaga perioadă semnalul are valoarea "1" și se exprimă în procente.

$$T = \frac{p}{p+q} \; x \; 100\%, \text{ unde:}$$

- **p este timpul de "1" (timpul în care ne aflăm pe nivelul 1 logic);**
- **q este timpul de "0" (timpul în care ne aflăm pe nivelul 0 logic);**

Notă: T = p + q.

PWM (Pulse Width Modulation) este o tehnică esențială în controlul dispozitivelor electronice care permite varierea eficientă a tensiunii medii aplicate unei sarcini.

Aceasta este realizată prin alternarea rapidă între starea **HIGH** și **LOW** a unui semnal digital. Durata în care semnalul este **HIGH** (**duty cycle**) determină puterea medie aplicată. **PWM** acolo unde este necesar un control precis al puterii fără a risipi energie. În implementarea hardware se ține cont, în aproape toate microcontrolerele, de registrele **Timer / Counter**. Astfel, generarea unei frecvențe prin **PWM** ține cont de frecvența microcontrolerului și capacitățile registrelor **Timer / Counter**: **rezoluție**, posibilitate de **output compare**, posibilitate de **modificare a duty cycle-ului**.

Pentru a calcula **perioada**, **frecvența** și **factorul de umplere** al unui semnal **PWM** generat de un microprocesor **ATmega1280** folosind un timer de **8 sau 16 biți**, se vor folosi următoarele formule, ținând cont de frecvența microprocesorului (**8 MHz în mod normal**, **16 MHz frecvența maximă**) și de **factorul de prescaler** care poate fi **selectat de utilizator**.

6.5.1 FRECVENȚA (HZ)

Frecvența unui semnal **PWM** depinde de valoarea **contorului / timer-ului**, de **prescaler** și de **frecvența clock-ului** sistemului. Formula generală este:

$$f_{PWM} = \frac{\text{frecvența microcontrolerului}}{\text{prescaler} * \text{valoarea maximă a timer-ului}} \quad (1)$$

De exemplu, pentru **ATmega1280** care funcționează la o frecvență la **8 MHz**, valoarea frecvenței microcontrolerului (sau frecvența ceasului – f_{CLK}) este:

$$f_{CLK} = 8 \text{ MHz} = 8 \times 10^6 \text{ Hz}$$

Pentru un timer pe **8 biți**:

$$f_{PWM} = \frac{f_{CLK}}{\text{prescaler} * 256} \quad (2)$$

Pentru un timer pe **16 biți**:

$$f_{PWM} = \frac{f_{CLK}}{\text{prescaler} * 65536} \quad (3)$$

6.5.2 PERIOADA (MILISECUNDE)

Perioada unui semnal **PWM** poate fi calculată astfel:

$$\textbf{Perioada } (T_{PWM}) = \frac{1}{f_{PWM}} \quad \textbf{(4)}$$

Notă: Atenție la unitățile de măsură! În cazul anterior, perioada este exprimată în **secunde**. Dacă se dorește obținerea perioadei în **milisecunde** (**ms**), multiplicăm cu **1000**:

$$\textbf{Perioada } (T_{PWM}) = \frac{1000}{f_{PWM}} \quad \textbf{(5)}$$

6.5.3 FACTORUL DE UMPLERE (DUTY CYCLE)

Definiție

Factorul de umplere (Duty Cycle) este procentul de timp în care semnalul PWM este "activ" (adică are valoare logică 1).

Dacă numărul de incrementări ale contorului **timer-ului** până când semnalul **PWM** se dezactivează este **OCRxn (Output Compare Register)**, atunci factorul de umplere se calculează astfel:

$$\textbf{Duty Cycle} = \left(\frac{\text{OCRxn}}{\text{valoarea maximă a timer-ului}}\right) \textbf{ x 100\% } \textbf{(6)}$$

Pentru un timer de **8 biți**:

$$\textbf{Duty Cycle} = \left(\frac{\text{OCRxn}}{255}\right) \textbf{ x 100\% } \textbf{(7)}$$

Pentru un timer de **16 biți**:

$$\textbf{Duty Cycle} = \left(\frac{\text{OCRxn}}{65535}\right) \textbf{ x 100\% } \textbf{(8)}$$

6.6 MODURI DE OPERARE PWM

Microcontrolerul **ATmega1280** suportă generarea semnalelor **PWM** prin intermediul mai multor timere integrate. Acesta oferă mai multe moduri principale de generare a semnalelor PWM. Valorile counter-ului sunt clasificate astfel:

- **BOTTOM*:*** reprezintă valoarea **minimă** a numărătorului (**counter**);
- **MAX*:*** reprezintă valoarea **maximă** a numărătorului (**counter**);
- **TOP*:*** counter-ul ajunge la valoarea **TOP** când devine **egal cu cea mai mare valoare din secvența de numărat**, iar această valoare poate fi **fixă** (**MAX**) sau **valoarea stocată** în registrul **OCRnx** în funcție de modul de operare.

6.6.1 CLEAR TIMER ON COMPARE MATCH (CTC)

În modul de operare **CTC** numărătorul crește de la **0**, dar în loc să ajungă la valoarea **MAX**, se resetează automat la **0** atunci când valoarea sa devine **egală** cu cea din registrul **OCRnx** (sau **ICRn** în cazul **TCNT** pe **16 biți**). Astfel, **OCRnx** acționează ca o valoare maximă personalizată (**TOP**).

Frecvența semnalului generat în modul **CTC** este determinată de valoarea **OCR** și de **prescaler-ul** utilizat pentru timer, fiind calculată cu formula:

$$F_{CTC} = \frac{F_{CLK}}{2 \text{ x N x } (OCRnx + 1)} \quad \textbf{(9)}, \text{ unde:}$$

- **F_{CTC}** este frecvența semnalului generat în modul **CTC**;
- **F_{CLK}** este frecvența de lucru a microcontrolerului;
- **N** este valoarea factorului de scalare;
- **OCRxn** este valoarea din registrul de comparație.

Astfel, modul **CTC** este ideal pentru aplicații de generare a unor semnale **precise** și **periodice**, fără necesitatea unui control fin asupra **duty cycle-ului**.

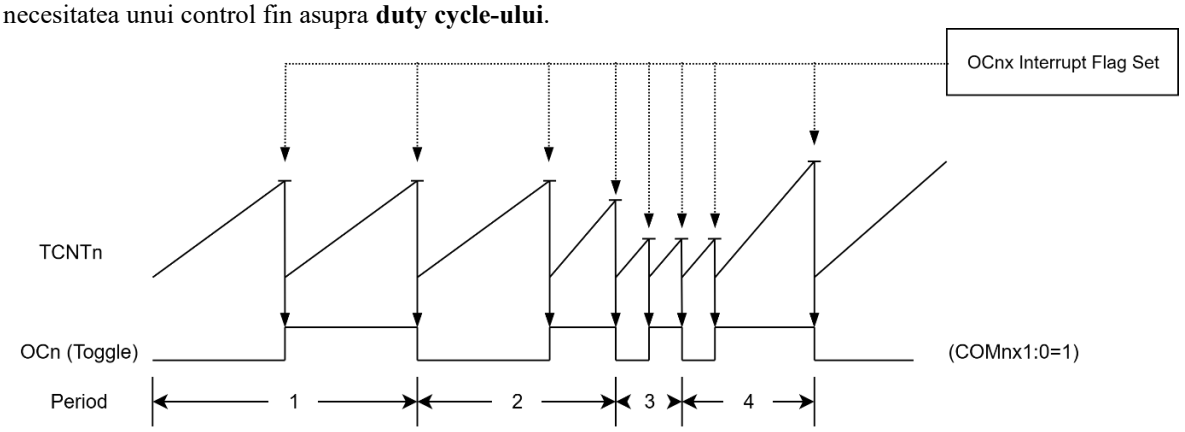


Figura 6.2 – Diagrama de timp pentru modul CTC

6.6.2 FAST PWM

Modul **Fast PWM** se bazează pe o numărare cu o singură pantă (**single-slope**): numărătorul crește de la **0** la valoarea **TOP**, apoi este resetat brusc înapoi la **0**, formând o undă de tip **"fierăstrău"**. Ieșirea **PWM** își schimbă starea la **începutul** ciclului (când numărătorul este la **0**) și la **potrivirea** dintre **TCNTn** și **OCRnx**.

Principalele caracteristici ale modului **Fast PWM** sunt:

1. Factorul de umplere a semnalului este controlat prin valoarea **OCRxn**. O valoare **OCRxn** mai **mică** înseamnă un factor de umplere mai **mic**, iar o valoare mai **mare OCRxn** înseamnă un factor de umplere mai **mare**.
2. Frecvența semnalului este determinată de valoarea **maximă** a contorului și de **factorul de scalare** a timer-ului. Spre deosebire de modul **CTC**, frecvența rămâne **constantă** în timpul funcționării, iar singura variabilă este factorul de umplere.

Acest mod este utilizat în aplicații unde sunt necesare semnale de înaltă frecvență și unde modificarea rapidă a factorului de umplere este **esențială**, cum ar fi în controlul motoarelor de curent continuu (**PWM controlat de microcontroler**), generarea de tonuri audio sau aplicațiile de iluminat controlate electronic.

Frecvența semnalului PWM în modul Fast PWM se calculează astfel:

$$\mathbf{F_{FastPWM} = \frac{F_{CLK}}{N \times (TOP + 1)}} \quad \textbf{(10)}, \text{ unde:}$$

- **$F_{FastPWM}$** este frecvența semnalului generat;
- **F_{CLK}** este frecvența de lucru a microcontrolerului;
- **N** este valoarea factorului de scalare;
- **TOP** este valoarea maximă la care ajunge contorul înainte de a fi resetat.

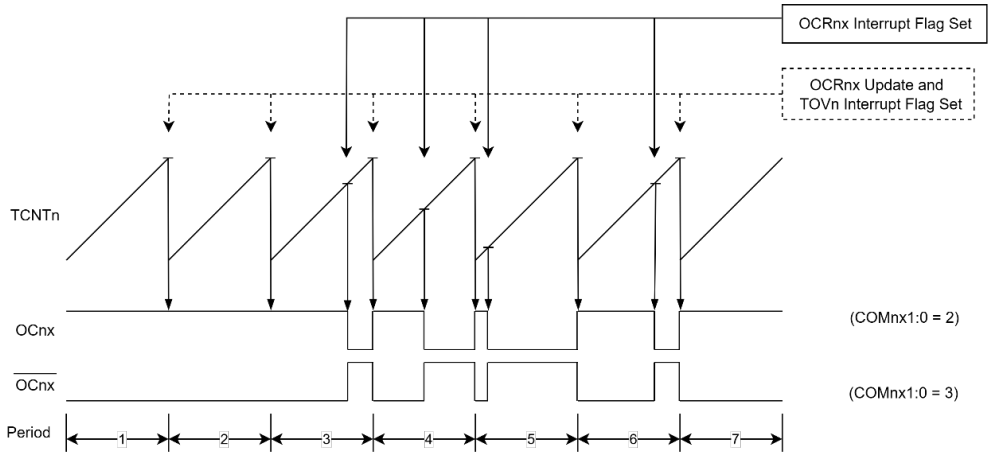


Figura 6.3 – Diagrama de timp pentru modul Fast PWM

6.6.3 PHASE CORRECT PWM

Acest mod de operare generează un semnal PWM **simetric**. **Phase Correct PWM** funcționează pe principiul numărării cu **pantă dublă (dual-slope)**: numărătorul crește de la **0** (**BOTTOM**) la **TOP**, apoi scade **simetric** înapoi la **0**, formând o undă **triunghiulară**. Ieșirea **PWM** își schimbă starea o dată pe panta **crescătoare** și a doua oară pe panta **descrescătoare**, la potrivirea cu **OCRnx**.

Acest mod este caracterizat de:

- Factorul de umplere este determinat de valoarea registrului **OCRnx**, la fel ca în modul **Fast PWM**, dar faza semnalului este ajustată pentru a minimiza **distorsiunea** și **zgomotul de comutație**. Aceasta asigură o **simetrie perfectă** în distribuția timpului între stările **HIGH** și **LOW**.
- Frecvența semnalului este de obicei mai **mică** decât în modul **Fast PWM**, deoarece contorul efectuează atât o contorizare **ascendentă**, cât și una **descendentă** într-un ciclu complet.

Modul **Phase Correct PWM** este ideal pentru aplicații care necesită o comutație mai lină a semnalului și minimizarea variațiilor rapide de tensiune, cum ar fi în controlul motoarelor, unde este importantă reducerea zgomotului și a interferențelor electromagnetice (**EMI**).

Frecvența semnalului **PWM** în modul **Phase Correct** este calculată astfel:

$$\mathbf{F_{PhaseCorrect}} = \frac{\mathbf{F_{CLK}}}{\mathbf{2x\ N\ x\ (TOP\ +1)}} \quad \textbf{(11)}, \text{ unde:}$$

- $\mathbf{F_{PhaseCorrect}}$ este frecvența semnalului generat,
- $\mathbf{F_{CLK}}$ este frecvența de lucru a microcontrolerului,
- **N** este valoarea factorului de scalare,
- **TOP** este valoarea maximă la care contorul oscilă înainte de a schimba direcția.

6.12 REGIȘTRI

6.12.1 TEMPORIZATOARE PE 8 BIȚI

6.12.1.1 TCCRNA – TIMER / COUNTER CONTROL REGISTER A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B1	-	-	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Valoare Inițială	0	0	0	0	0	0	0	0	

Figura 6.15 - Registrul TCCR0A

- **Biții 7:6 – COMnA1:0: Compare Match Output A Mode**
  Sunt biții de control pentru pinul de **Output Compare** (**OCnA**). Dacă cel puțin un bit dintre aceștia este setat, ieșirea **OCnA** va **suprascrie** funcționalitatea portului pinului **I/O** la care este conectat.

COM0A1	COM0A0	Descriere
0	0	Funcționarea normală a portului, **OC0A** deconectat
0	1	Activează **OC0A** la Compare Match
1	0	Șterge **OC0A** la Compare Match
1	1	Setează **OC0A** la Compare Match

Tabelul 6.1 - Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Descriere
0	0	Funcționare normală a portului, **OC0A** deconectat
0	1	Când **WGM02 = 0**: funcționare normală a portului, **OC0A** deconectat; Când **WGM02 = 1**: se activează **OC0A** la Compare Match
1	0	Șterge **OC0A** la Compare Match, setează **OC0A** la valoarea **BOTTOM** (modul neinversat)
1	1	Setează **OC0A** la Compare Match, șterge **OC0A** când ajunge la valoarea **BOTTOM** (modul inversat)

Tabelul 6.2 - Compare Output Mode, Fast PWM Mode

COM0A1	C0M0A0	Descriere
0	0	Funcționare normală a portului, **OC0A** deconectat
0	1	Când **WGM02 = 0**: funcționare normală a portului, **OC0A** deconectat; Când **WGM02 = 1**: se activează **OC0A** la Compare Match
1	0	În momentul incrementării se va șterge **OC0A** la Compare Match. În momentul decrementării, se va seta **OC0A** la Compare Match.
1	1	În momentul incrementării se va seta **OC0A** la Compare Match. În momentul decrementării, se va șterge **OC0A** la Compare Match.

Tabelul 6.3 - Compare Output Mode, Phase Correct PWM Mode

- **Biții 5:4 – C0MnB1:0 Compare Match Output Mode**
  Sunt biții de control pentru pinul de **Output Compare** (**OCnB**). Dacă cel puțin un bit dintre aceștia este setat, ieșirea **OCnB** va suprascrie funcționalitatea portului pinului I/O la care este conectat.

COM0B1	COM0B0	Descriere
0	0	Funcționarea normală a portului, **OC0B** deconectat
0	1	Activează **OC0B** la Compare Match
1	0	Șterge **OC0B** la Compare Match
1	1	Setează **OC0B** la Compare Match

Tabelul 6.4 - Compare Output Mode, non-PWM Mode

COM0B1	COM0B0	Descriere
0	0	Funcționare normală a portului, **OC0B** deconectat
0	1	Rezervat
1	0	Șterge **OC0B** la Compare Match, setează **OC0B** la valoarea **BOTTOM** (modul neinversat)
1	1	Setează **OC0B** la Compare Match, Șterge **OC0B** când ajunge la valoarea **BOTTOM** (modul inversat)

Tabelul 6.5 - Compare Output Mode, Fast PWM Mode

COM0B1	C0M0B0	Descriere
0	0	Funcționare normală a portului, **OC0B** deconectat
0	1	Rezervat
1	0	În momentul incrementării se va șterge **OC0B** la Compare Match. În momentul decrementării, se va seta **OC0B** la Compare Match.
1	1	În momentul incrementării se va seta **OC0B** la Compare Match. În momentul decrementării, se va șterge **OC0B** la Compare Match.

Tabelul 6.6 - Compare Output Mode, Phase Correct PWM Mode

- **Biții 3:2 – Res (Reserved Bits)**
  Acești biți sunt rezervați și mereu vor avea valoarea **0** la citire.

- **Biții 1:0 – WGMn1:0 Waveform Generation Mode**
  Împreună cu bitul **WGMn2** din registrul **TCCRnB**, acești biți controlează secvența de numărare a contorului, sursa pentru valoarea maximă (**TOP**) a contorului și **tipul de generare** a formei de undă care va fi utilizat.

Mode	WGM2	WGM1	WGM0	Timer / Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	1	1	1	Fast PWM	0xFF	TOP	MAX
4	1	0	0	Reserved	-	-	-
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	-	-	-
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Tabelul 6.7 - Moduri de operare pentru TCNT0

Notă: Modul Reserved nu este un mod de funcționare propriu-zis. Reserved (Rezervat) indică o combinație specifică a biților de configurare pe care producătorul a decis să nu o documenteze pentru uz public.

6.12.1.2 TCCRNB – TIMER / COUNTER CONTROL REGISTER B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Valoare Inițială	0	0	0	0	0	0	0	0	

Figura 6.16 - Registrul TCCRNB

- **Bitul 7 – FOCnA: Force Output Compare A**
  Acest bit este activ doar atunci când biții **WGM** specifică modul **non-PWM**, este implementat ca un impuls și va avea mereu valoarea **0**.

- **Bitul 6 – FOCnB: Force Output Compare B**
  Acest bit este activ doar atunci când biții **WGM** specifică modul **non-PWM**, este implementat ca un impuls și va avea mereu valoarea **0**.

- **Biții 5:4 – Reserved Bits**

- **Bitul 3 – WGMn2: Waveform Generation Mode**

- **Biții 2:0 – CSn2:0: Clock Select**
  Acești biți selectează **sursa de ceas** utilizată de temporizator.

CS02	CS01	CS00	Descriere
0	0	0	No clock source – temporizatorul este oprit
0	0	1	$clk_{I/O}$ (fără prescaler)
0	1	0	$clk_{I/O}$ / 8 (from prescaler)
0	1	1	$clk_{I/0}$ / 64 (from prescaler)
1	0	0	$clk_{I/O}$ / 256 (from prescaler)
1	0	1	$clk_{I/O}$ / 1024 (from prescaler)
1	1	0	Sursa externă de clock prin intermediul pinului **T0** pentru front descrescător
1	1	1	Sursa externă de clock prin intermediul pinului **T0** pentru front crescător

Tabelul 6.8 – Descrierea biților de Clock Select

- **Bitul 4 – TCN2UB: Timer / Counter2 Update Busy**
  Atunci când **TCNT2** funcționează în modul **asincron** și **TCNT2** este scris se activează acest bit. Dacă **TCNT2** a fost actualizat dintr-un registru de stocare temporar, acest bit va fi **șters de hardware**. Valoarea **0 logic** indică că **TCNT2** este „pregătit” să fie **actualizat** cu o valoare nouă.

- **Bitul 3 – OCR2AUB: Output Compare Register2 Update Busy – Idem.**

- **Bitul 2 – OCR2BUB: Output Compare Register2 Update Busy – Idem.**

- **Bitul 1 - TCR2AUB: Timer / Counter Control Register2 Update Busy – Idem.**

- **Bitul 0 - TCR2BUB: Timer / Counter Control Register2 Update Busy – Idem.**

6.12.1.9 GTCCR – GENERAL TIMER / COUNTER CONTROL REGISTER

Bit	7	6	5	4	3	2	1	0	
0x23 (0x43)	TSM	-	-	-	-	-	PSRASY	PSRSYNC	GTCCR
Read/Write	R/W	R	R	R	R	R	R/W	R/W	
Valoare Inițială	0	0	0	0	0	0	0	0	

Figura 6.23 - Registrul GTCCR

- **Bitul 7 – TSM: Timer / Counter Synchronization Mode**
  Atunci când valoarea acestui bit este **1 logic** se activează modul **sincron** al temporizatorului. Când acest bit este setat pe valoarea **0 logic**, contoarele încep să numere **simultan**.

- **Bitul 1 – PSRASY: Prescaler Reset Timer / Counter2**
  Atunci când acest bit are valoarea **1 logic**, factorul de scalare pentru **TCNT2** va fi **resetat**.

- **Bitul 0 – PSRSYNC: Prescaler Reset for Synchronous Timer / Counter**
  Atunci când acest bit este setat pe valoarea **1 logic**, valoarea factorului de scalare pentru **TCNT0**, **TCNT1**, **TCNT3**, **TCNT4** și **TCNT5** va fi **resetată**.

6.12.2 TEMPORIZATOARE PE 16 BIȚI

6.12.2.1 TCCRNA – TIMER / COUNTER N CONTROL REGISTER A

Bit	7	6	5	4	3	2	1	0	
	COMnA1	COMnA0	COMnB1	COMnB1	COMnC1	COMnC0	WGMn1	WGMn0	TCCRnA
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 6.24 - Registrul TCCRnA

- **Biții 7:6 – COMnA1:0: Compare Output Mode for Channel A**

- **Biții 5:4 – COMnB1:0: Compare Output Mode for Channel B**

- **Biții 3:2 – COMnC1:0: Compare Output Mode for Channel C**
  Acești biți sunt biții de control pentru pinii **Output Compare** (**OCnA**, **OCnB**, **OCnC**). Dacă cel puțin un bit dintre aceștia este setat, ieșirea **OCnA** va suprascrie funcționalitatea portului pinului **I/O** la care este conectat.

- **Biții 1:0 – WGMn1:0: Waveform Generation Mode**
  Împreună cu bitul **WGMn3:2** din registrul **TCCRnB**, acești biți controlează secvența de numărare a contorului, sursa pentru valoarea maximă (**TOP**) a contorului și tipul de generare a formei de undă care va fi utilizat.

COMnA1 COMnB1 COMnC1	COM0A0 COM0B0 COM0C0	Descriere
0	0	Funcționarea normală a portului, **OCnA** / **OCnB** / **OCnC** deconectat
0	1	Activează **OCnA** / **OCnB** / **OCnC** la Compare Match
1	0	Șterge **OCnA** / **OCnB** / **OCnC** la Compare Match (setează ieșirea pe **0** logic)
1	1	Setează **OCnA** / **OCnB** / **OCnC** la Compare Match (setează ieșirea pe **1** logic)

Tabelul 6.9 – Compare Output Mode, non-PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Descriere
0	0	Funcționare normală a portului, **OCnA** / **OCnB** / **OCnC** deconectat
0	1	Când **WGM13:0 = 14** sau **15**: se activează **OC1A** la Compare Match și **OC1B** și **OC1C** sunt deconectate. Pentru celelalte setări ale pinilor **WGM1** portul va funcționa normal, iar **OCnA** / **OCnB** / **OCnC** sunt deconectate
1	0	Șterge **OCnA** / **OCnB** / **OCnC** la Compare Match, setează **OCnA** / **OCnB**/ **OCnC** la valoarea **BOTTOM** (modul neinversat)
1	1	Setează **OCnA** / **OCnB** / **OCnC** la Compare Match, Șterge **OCnA** / **OCnB** / **OCnC** când ajunge la valoarea **BOTTOM** (modul inversat)

Tabelul 6.10 – Compare Output Mode, Fast PWM

COM0A1	C0M0A0	Descriere
0	0	Funcționare normală a portului, **OCnA** / **OCnB** / **OCnC** deconectat
0	1	Când **WGM13:0 = 9** sau **11**: se activează **OC1A** la Compare Match și **OC1B** și **OC1C** sunt deconectate. Pentru celelalte setări ale pinilor **WGM1** portul va funcționa normal, iar **OCnA** / **OcnB** / **OCnC** sunt deconectate
1	0	În momentul incrementării se va șterge **OCnA** / **OCnB** / **OCnC** la Compare Match. În momentul decrementării, se va seta **OCnA** / **OCnB** / **OCnC** la Compare Match.
1	1	În momentul incrementării se va seta **OCnA** / **OCnB** / **OCnC** la Compare Match. În momentul decrementării, se va șterge **OCnA** / **OCnB** / **OCnC** la Compare Match.

Tabelul 6.11 – Compare Output Mode, Phase Correct și Phase and Frequency Correct PWM

6.12.2.2 TCCRNB – TIMER / COUNTER N CONTROL REGISTER B

Bit	7	6	5	4	3	2	1	0	
	ICNCn	ICESn	-	WGMn3	WGMn2	CSn2	CSn1	CSn0	TCCRnB
Read/Write	W	W	R	R/W	R/W	R/W	R/W	R/W	
Valoare Inițială	0	0	0	0	0	0	0	0	

Figura 6.25 - Registrul TCCRnB

- **Bitul 7 – ICNCn: Input Capture Noise Canceler**
  Este activ când are valoarea **1 logic**. Când **Noise Canceler** este activ, semnalul de la pinul **Input Capture** (**ICPn**) este filtrat. Funcția de filtrare necesită **patru eșantioane** succesive **egale** ale pinului **ICPn** pentru a schimba valoarea la ieșire. Prin urmare, captura semnalului va fi întârziată cu **patru ciclii** ale oscilatorului atunci când **Noise Canceler** este **activ**.

- **Bitul 6 – ICESn: Input Capture Edge Select**
  Acest bit permite selectarea frontului care va fi utilizat pentru declanșarea evenimentului de captură pe **Input Capture Pin** (**ICPn**). Când bitul **ICESn = 0**, se va utiliza frontul descrescător pentru a declanșa captura. Când bitul **ICESn = 1**, se va utiliza frontul crescător pentru a declanșa captura.

- **Bitul 5 – Res: Reserved**.

- **Biții 4:3 – WGMn3:2: Waveform Generation Mode**.

- **Biții 2:0 – CSn2:0: Clock Select**.

CS02	CS01	CS00	Descriere
0	0	0	No clock source – temporizatorul este oprit
0	0	1	$clk_{I/O}$ (fără prescaler)
0	1	0	$clk_{I/O}$ / 8 (from prescaler)
0	1	1	$clk_{I/0}$ / 64 (from prescaler)
1	0	0	$clk_{I/O}$ / 256 (from prescaler)
1	0	1	$clk_{I/O}$ / 1024 (from prescaler)
1	1	0	Sursa externă de clock prin intermediul pinului Tn pentru front descrescător
1	1	1	Sursa externă de clock prin intermediul pinului Tn pentru front crescător

Tabelul 6.12 – Descrierea bitului Clock Select

Mode	WGMn3	WGMn2 (CTC)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Time / Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM

11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	-	-	-
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Tabelul 6.13 – Moduri de operare pentru TCNT1, 3, 4 și 5

6.12.2.3 TCCRNC – TIMER / COUNTER N CONTROL REGISTER C

Bit	7	6	5	4	3	2	1	0	
	FOCnA	FOCnB	FOCnC	-	-	-	-	-	TCCRnC
Read/Write	W	W	W	R	R	R	R	R	
Valoare Inițială	0	0	0	0	0	0	0	0	

Figura 6.26 - Registrul TCCRnC

- **Bitul 7 – FOCnA: Force Output Compare for Channel A**
- **Bitul 6 – FOCnB: Force Output Compare for Channel B**
- **Bitul 5 – FOCnC: Force Output Compare for Channel C**
- **Biții 4:0 – Res: Reserved.**

Biții **FOCnA** / **FOCnB** / **FOCnC** sunt activi doar când **WGMn3:0** specifică un mod de operare **non-PWM**.

6.12.2.4 TCNTNH ȘI TCNTNL – TIMER / COUNTER

Bit	7	6	5	4	3	2	1	0	
	TCNTn[15:8]								TCNTnH
	TCNTn[7:0]								TCNTnL
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 6.27 - Registrul TCNTnH și TCNTnL

6.12.2.5 OCRNXH ȘI OCRNXL – OUTPUT COMPARE REGISTER

Bit	7	6	5	4	3	2	1	0	
	OCRnx[15:8]								OCRnxH
	OCRnx[7:0]								OCRnxL
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 6.28 - Registrul OCRnXH și OCRnXL

6.12.2.6 ICRNH ȘI ICRNL – INPUT CAPTURE REGISTER

Valoarea acestui registru este actualizată cu valoarea contorului (**TCNTn**) la fiecare eveniment al pinul **ICPn**.

7. RESET & WATCHDOG

7.1 UNDE SE FOLOSEȘTE RESET & WATCHDOG?

Watchdog Timer (**WDT**) este utilizat pe scară largă în sisteme electronice și embedded pentru a crește fiabilitatea și a preveni blocarea aplicațiilor. Principalele domenii de utilizare includ:

- **Automotive:** monitorizarea unităților de control electronice pentru a detectarea erorilor software critice;
- **Sisteme industriale:** protecția sistemelor de control împotriva blocajelor;
- **Electronice de consum:** mentenanța dispozitivelor precum router-elor, TV-urilor și a electrocasnicelor;
- **Aerospațial și militar:** garantarea funcționării continue a echipamentelor în medii critice;
- **Dispozitive medicale:** asigurarea siguranței pacienților prin repornirea automată a aparatelor.

7.2 CUNOȘTINȚE NECESARE

Pentru a putea parcurge acest capitol, este necesar să cunoașteți următoarele subiecte din capitolele anterioare, și nu numai:

- Utilizarea întreruperilor;
- Utilizarea timerelor pentru întreruperi sincronizate și generarea semnalelor **PWM**;
- Modul de funcționare al memoriilor **EEPROM**;
- Utilizarea kit-ului hardware Mikroe.

7.3 ABSTRACT

Acest capitol are în vedere descrierea resetării sistemului și cauzele acesteia, punând în evidență mecanismul Watchdog-ului.

7.4 HARDWARE ȘI SOFTWARE NECESAR

- ATMega1280 SiBRAIN;
- UNI Clicker;
- EEPROM Click;
- Atmel ICE;
- IAR *Embedded Workbench* IDE 7.30.5;
- Osciloscop.

7.5 INTRODUCERE ÎN CONTROLUL ȘI RESETAREA SISTEMULUI

În timpul resetării, toți regiștrii de 1/0 sunt readuși la valorile lor inițiale determinate de hardware, iar programul își începe execuția de la adresa vectorului Reset(adresa fixa 0x0000). Instrucțiunea plasată în vectorul de Reset ar trebui să fie o instrucțiune de salt, și anume o instrucțiune de salt absolut(JMP) spre rutina care gestionează resetarea.

Întreruperile sunt dezactivate implicit după reset, prin dezactivarea bitului global de întreruperi, pentru a preveni întreruperi premature care ar putea perturba procesul de inițializare. Vectorul de întreruperi este folosit doar după ce întreruperile sunt activate explicit prin setarea bitului global. Dacă întreruperile nu sunt activate, execuția programului continuă normal, secvențial, fără să fie influențată de întreruperi. Astfel, vectorul de reset asigură un punct clar și sigur de pornire a programului, iar vectorii de întreruperi sunt utilizați numai când întreruperile sunt permise și active.

7.6 SURSELE DE RESETARE

Microcontrolerul Atmega1280 are 5 surse de resetare:

- **Resetarea la pornire(Power-on Reset):** microcontrolerul este resetat, atunci când tensiunea de alimentare crește de la zero și se află sub pragul de resetare la pornire; procesorul începe execuția abia după ce V_{CC} este stabilă.
- **Resetarea externă(External Reset):** atunci când pinul RESET este menținut la nivel logic 0 pentru o durată mai mare decât perioada minimă necesară pentru semnalul de reset.
- **Resetarea prin Watchdog(Watchdog Reset): d**acă temporizatorul watchdog nu este resetat la timp de către program, el generează un reset automat pentru a evita blocarea sistemului.
- **Resetarea de tip brown-out(Brown-out Reset):** fenomenul de **brown-out** constă în scăderea valorii tensiunii într-un circuit electric sub un anumit prag, numit prag de **brown-out**; microcontrolerul este resetat când valoarea tensiunii de alimentare $\mathbf{V_{CC}}$ este inferioară valorii pragului de resetare **brown-out** ($\mathbf{V_{BOT}}$) și detectorul fenomenului de **brown-out** este activat.
- **Resetarea de tip JTAG AVR:** microcontrolerul e resetat când registrul de **Reset** are valoarea **1** logic.

7.7 INTRODUCERE ÎN WATCHDOG

Ideea de bază ce stă în spatele existenţei unui **watchdog timer** este de a avea un mecanism care să reseteze microcontrolerul în cazul în care dintr-un motiv excepțional, aplicația nu răspunde un timp îndelungat. Această funcție este esențială pentru aplicații în timp real și sisteme embedded sigure și fiabile.

Familia ATMega AVR 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 folosiţi 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 stabilită inițial;
- **Resetarea sistemului** – resetarea microcontrolereului AVR.

Prezintă următoarele aspecte:

- Semnalul de ceas este separat și provine de la un oscilator **on-chip**.
- Are 3 moduri de operare: **Interrupt**, **System Reset** și **Interrupt and System Reset**.
- O perioadă de time-out selectabilă de la **16 ms** la **8 s**.

7.7.1 SCHEMA DE PRINCIPIU

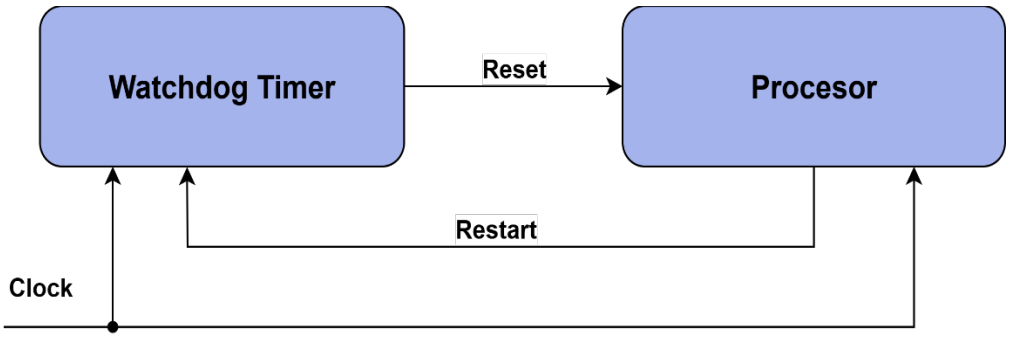


Figura 7.1 – Schema de principiu

Definiție

Watchdog-ul este un timer care contorizează ciclurile semnalului de ceas pe un oscilator on-chip separat, ce are frecvența de 128 kHz.

Acesta produce o întrerupere sau o resetare a sistemului atunci când contorul său atinge valoarea de expirare a timpului (**time-out**). În modul normal de operare, pentru a preveni resetarea, programul trebuie să execute periodic instrucțiunea specială WDR (Watchdog Timer Reset), care resetează contorul intern al WDT-ului, împiedicând astfel atingerea time-out-ului. Dacă sistemul nu îl resetează, o întrerupere sau o resetare a sistemului va fi produsă.

La nivel intern, acesta trebuie resetat de către software la intervale de timp regulate pentru ca numărătorul acestuia să nu atingă valoarea **0**, caz în care se produc întreruperi și / sau reset-uri nedorite.

Astfel, Watchdog Timer-ul servește ca o „paznic" ce resetează automat sistemul dacă software-ul nu confirmă în mod regulat că rulează corect, prevenind astfel blocările sau situațiile în care sistemul nu mai răspunde.

7.7.2 SINCRONIZAREA ÎN 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**, acestea fiind efective la următorul front pozitiv al **clock-ului watchdog-ului**, adică după **2-3 ms** după ce setările au fost scrise în registru. 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**. Ținând cont de precizia oscilatorului ± 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**:

T = 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 exemplu 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 **MEGA Real Time Clock Timer-ului**.

În modul normal, Watchdog trebuie resetat înainte ca temporizatorul să ajungă la timpul de expirare (time-out) pentru a preveni resetarea sistemului.
În modul fereastră, Watchdog poate fi resetat numai într-un interval de timp specific (fereastra). Resetarea în afara acestui interval, prea devreme sau prea târziu, va cauza resetarea procesorului. Acest mod oferă un control mai strict asupra momentului în care este făcut resetul pentru a detecta erori mai complexe.

Toate aceste caracteristici sunt valabile atât în modul normal cât și în modul fereastră.

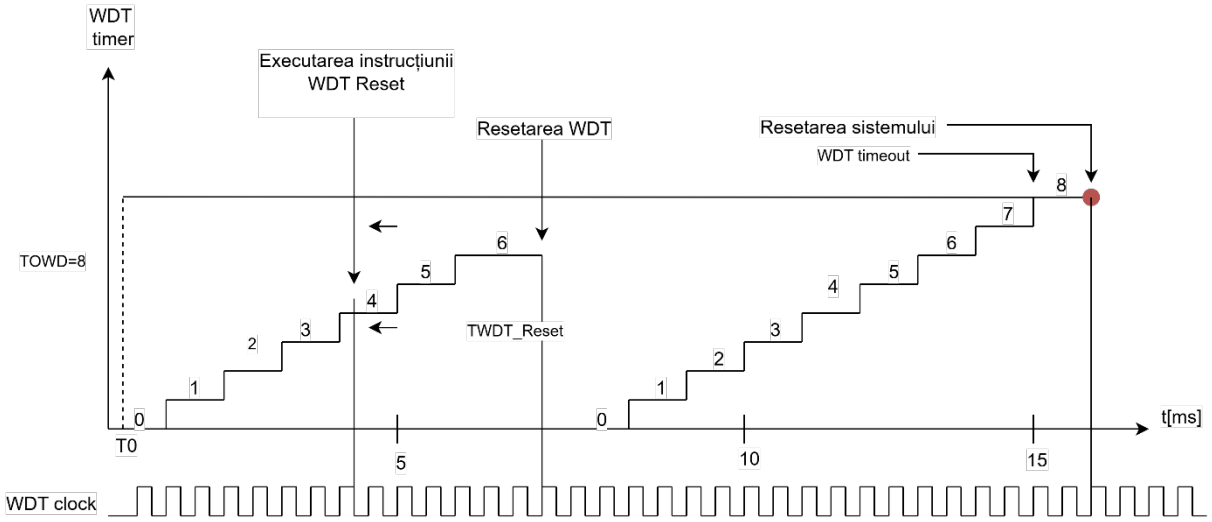


Figura 7.2 – Reprezentarea grafică a ciclurilor Watchdog Timer (WDT) de la inițializare (T0) până la resetarea manuală (TWDT_Reset)

7.7.3 MODURILE DE FUNCȚIONARE

7.7.3.1 INTERRUPT MODE (MODUL DE ÎNTRERUPERI)

Watchdog-ul produce o **întrerupere** atunci când timpul setat **expiră**, aceasta fiind folosită pentru a "trezi" anumite dispositive din modul sleep. Un exemplu de utilizare a acestui mod este de a limita timpul maxim al unei anumite operații, producându-se o întrerupere atunci când acesta este atins. Astfel, operația nu rulează mai mult timp decât este așteptat.

7.7.3.2 SYSTEM RESET MODE (MODUL DE RESETARE AL SISTEMULUI)

Watchdog-ul produce o **resetare** atunci când timpul **expiră**. Acesta este utilizat pentru a preveni blocarea sistemului în cazul în care se execută cod interminabil.

7.7.3.3 INTERRUPT AND SYSTEM RESET MODE

Interrupt and System Reset Mode combină celelalte 2 moduri de funcționare. În primul rând se produce **întreruperea**, apoi **resetarea** sistemului. Dacă bitul **WDE** este setat, **watchdog-ul** se află în modul acesta. Primul time-out al numărătorului va seta **WDIF**, iar executarea vectorului de întreruperi va reseta biții **WDIE** și **WDIF** cu ajutorul hardware-ului, watchdog-ul trecând în modul de resetare. Pentru a-l menține în modul de întreruperi, bitul **WDIE** trebuie setat după fiecare întrerupere. Acest lucru nu ar trebui realizat folosind rutina de întrerupere pentru că s-ar putea compromite funcționalitatea de siguranță a modului de reset. Dacă întreruperea nu se exexcută înainte de time-out, se va produce o resetare a sistemului. **Interrupt and System Reset Mode** este folosit pentru închiderea în manieră sigură a dispozitivului, salvând date critice (parametri critci) înainte de un reset.

7.8 STANDARDUL INTERNAȚIONAL IEC 60730

Definiție

IEC 60730 este un standard de siguranță pentru aparatele de uz casnic, care abordează aspectele de design și 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.

7.8.1 CLASELE STANDARDULUI IEC 60730

Anexa H a standardului definește **3 clase de control** ale software-ului pentru diferite 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 cu cod destinat prevenirii erorilor fără utilizarea dispozitivelor de protecție.

.

7.8.2 WATCHDOG-UL DE CLASĂ B

Arhitectura **MEGA** 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 MEGA**, s-a conceput o rutină de autodiagnosticare care testează ambele moduri de funcționare, normal și fereastră. Aceasta se execută după resetare, în partea de pre-inițializare a aplicației înaintea funcției main.

Rutina de autodiagnosticare ne asigură că:
- **Resetarea timer-ului** 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 faza 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 **32.768 kHz**. 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**.

Figura 7.4 – Registrul de stare MCUSR

- **Bit 0 – PORF: Power-on Reset Flag**
  Acest bit este setat (este scrisă valoarea 1 logic) în cazul unui Power-on Reset și resetat scriind valoarea 0 în dreptul său.

- **Bit 1 – EXTRF: External Reset Flag**
  Acest bit este setat în cazul unui reset extern și resetat (este scrisă valoarea 0 logic) în cazul unui Power-on Reset sau dacă este scrisă valoarea 0 în dreptul său.

- **Bit 2 – BORF: Brown-out Reset Flag**
  Fenomenul de Brown-out constă în scăderea valorii tensiunii sub un anumit prag într-un circuit electronic. Acest bit este setat în cazul unui Brown-out Reset și resetat în cazul unui Power-on Reset dacă este scrisă valoarea 0 în dreptul său.

- **Bit 3 – WDRF: Watchdog Reset Flag**
  Acest bit este setat în cazul unui Watchdog Reset și resetat în cazul unui Power-on Reset dacă este scrisă valoarea 0 în dreptul său.

- **Bit 4 – JTRF: JTAG Reset Flag**
  Acest bit este setat dacă resetarea a fost cauzată de registrul de reset al JTAG-ului selectat de instrucțiunea AVR_RESET, fiind necesar ca registrul să aibă valoarea 1 logic. Este resetat în cazul unui Power-on Reset sau dacă este scrisă valoarea 0 în dreptul său.

Pentru identificarea corectă a tipului de reset, utilizatorul ar trebui să citească și să reseteze registrul MCUSR cât mai devreme posibil în program. În contextul în care se întâmplă acest lucru, se poate identifica tipul reset-ului examinând flag-urile de reset.

7.9.2 REGISTRUL WDTCSR - WATCHDOG TIMER CONTROL REGISTER

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	X	0	0	0	

Figura 7.5 – Registrul WDTCSR

- **Bit 5, 2:0 – WDP 3:0: Watchdog Timer Prescaler**
  Biții WDP 3:0 determină prescalarea Watchdog-ului când acesta este activat, adică setează numărul de cicli ai acestuia (Să se consulte **tabelul 1.2**).

- **Bit 3 – WDE : Watchdog System Reset Enable**
  Bitul WDE este suprascris de bitul WDRF în registrul MCUSR. Prin urmare, WDE este setat întotdeauna când WDRF este setat. Pentru a reseta WDE, mai întâi trebuie resetat WDRF. Acest lucru asigură resetări multiple în condițiile blocării sau nefuncționării unui program și o pornire sigură după acest lucru.
- **Bit 4 – WDCE : Watchdog Change Enable**
  Acest bit este utilizat în secvențe cronometrate pentru a schimba WDE și biții de prescalare. Pentru a reseta bitul WDE și/sau schimba biții de prescalare, WDCE trebuie setat. După ce a fost setat, hardware-ul va reseta acest bit dupa 4 cicli de ceas.

- **Bit 6 – WDIE: Watchdog Interrupt Enable**
  Când acest bit are valorea 1 și bitul de întreruperi (I-bit) din registrul de stare (nu MCUSR) este setat, se activează întreruperea Watchdog-ului. Dacă WDE este resetat, iar I-bit-ul este activat, watchdog-ul se află în modul de întreruperi și întreruperea corespunzătoare se execută în contextul în care perioada de timp prestabilită a expirat. Dacă bitul WDE este setat Watchdog-ul se află în modul Interrupt and System Reset.

- **Bit 7 – WDIF: Watchdog Interrupt Flag**
  Acest bit este setat când perioada de timp a Watchdog-ului expiră, iar acesta este configurat pentru întreruperi. WDIF este resetat de hardware când se execută vectorul de întreruperi sau când este scrisă valoarea 1 în dreptul său.

WDTON	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt Mode	Interrupt
1	1	0	System Reset Mode	Reset
1	1	1	Interrupt and System Reset Mode	Interrupt, then System Reset Mode
1	x	X	System Reset Mode	Reset

Tabelul 7.1 – Modurile de operare ale biților registrului WDTCSR

WDP	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 5.0 V
0	0	0	0	2K (2048 cycles)	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125s
0	1	0	0	32K (32768) cycles	0.25s
0	1	0	1	64K (65536) cycles	0.5s
0	1	1	0	128K (131072) cycles	1.0s
0	1	1	1	256K (262144) cycles	2.0 s
1	0	0	0	512K(524288) cycles	4.0s
1	0	0	1	1024K (1048576) cycles	-
1	0	1	0	-	-
1	0	1	1	-	-
1	1	0	0	-	-
1	1	0	1	-	-
1	1	1	0	-	-
1	1	1	1	-	-

Tabelul 7.2 – Tabela de adevăr a biților WDP

8. CODURI REDUNDANTE CICLICE – CRC

8.1 UNDE SE FOLOSEȘTE CRC?

CRC este un algoritm folosit în principal pentru detecția erorilor în transmisia sau stocarea datelor în ceea ce privește:

- **Rețele de calculatoare:** verificarea transmiterii corecte (fără erori) ale datelor printr-o rețea;
- **Dispozitive de stocare:** verificarea integrității datelor scrise sau citite pe hard disk-uri, SSD, etc;
- **Protocoale de comunicație:** verificarea comunicării între periferice prin USB, HDLC, PPP, etc;
- **Sisteme embedded:** comunicarea între microcontrollere sau între senzori și procesoare.

8.2 CUNOȘTINȚE NECESARE

Pentru a putea parcurge acest capitol, este necesar să cunoașteți următoarele subiecte din capitolele anterioare, și nu numai:

- Aritmetica binară, în special operația **XOR**.
- Reprezentarea polinoamelor în format binar.
- Noțiuni de bază despre erorile de transmisie a datelor.
- Programare în limbajul C pentru microcontrollere AVR.
- Utilizarea mediului de dezvoltare IAR Embedded Workbench.

8.3 ABSTRACT

Acest capitol detaliază metoda de detectare a erorilor folosind **CRC** (**Cyclic Redundancy Check**). Se prezintă atât fundamentele teoretice, bazate pe împărțirea polinomială în aritmetică **modulo 2**, cât și aspectele practice ale implementării. Este exemplificat modul de calcul software, atât bit cu bit, cât și folosind **tabele pre-calculate**, și demonstrează generarea automată a sumei de control folosind funcționalitățile **linker-ului** din mediul **IAR / AVR**.

8.4 HARDWARE ȘI SOFTWARE NECESAR

- ATmega 1280 SiBRAIN;
- UNI clicker;
- Atmel ICE;
- IAR Embedded Workbench 7.30.5;
- Osciloscop (opțional, pentru analiza timpilor de execuție).

8.5 INTRODUCERE ÎN CRC

Definiție

Codurile Redundante Ciclice (CRC) reprezintă o metodă eficientă de detectare a erorilor pentru blocuri de date.

Fiecărui bloc de date i se atașează o valoare de control de lungime fixă, cunoscută ca și "**checksum**" sau **cod CRC**. Această valoare este **restul** unei împărțiri polinomiale a conținutului mesajului. La recepție, calculul se repetă, iar dacă noile date de control **nu corespund** cu cele recepționate, înseamnă că a apărut o eroare.

Algoritmul se numește astfel deoarece valoarea de control este o **redundanță** (mărește mesajul fără a adăuga informație utilă) și se bazează pe proprietățile **codurilor ciclice**. Acestea sunt foarte populare datorită simplității de implementare atât în hardware, cât și în software, a eficienței în detectarea erorilor comune și a analizei matematice facile.

Principiul de bază este împărțirea polinomială în aritmetică **modulo 2** (unde adunarea și scăderea sunt echivalente cu operația logică **XOR**). Mesajul este tratat ca un **polinom**, care este **împărțit** la un **polinom prestabilit**, numit **polinom generator**. Câtul împărțirii este **ignorat**, iar restul obținut constituie **codul CRC**.

Cel mai simplu exemplu de **CRC** este **bitul de paritate**, care este un **CRC** pe **1 bit** ce folosește polinomul generator **x + 1**.

8.6 SPECIFICAȚII ȘI PARAMETRI

Implementarea practică a unui **algoritm CRC** implică definirea mai multor parametri care caracterizează în mod unic procesul de calcul.

Deși eficiente pentru detectarea erorilor accidentale, **codurile CRC** standard **NU** sunt potrivite pentru a proteja datele împotriva alterărilor intenționate:

- **Lipsa autentificării:** Un atacator poate **modifica** mesajul și **recalcula** valoarea **CRC**, astfel încât modificarea să nu fie detectată. Aplicațiile care necesită protecție la modificări intenționate trebuie să folosească **mecanisme criptografice**, precum **semnăturile digitale**.
- **Reversibilitate:** Spre deosebire de **funcțiile hash criptografice**, funcția **CRC** este **ușor reversibilă**.
- **Liniaritate: CRC** are proprietatea liniară **crc(x) ⊕ crc(y) = crc(x ⊕ y)**. Aceasta permite manipularea unui mesaj criptat și a **CRC-ului** său asociat **fără a cunoaște** cheia de criptare.

Un **sistem CRC** este definit complet de parametrii enumerați mai jos.

8.6.1 POLINOMUL GENERATOR

Acesta este elementul central al algoritmului. Adesea, **bitul cel mai semnificativ** (care este mereu **1**) este omis din reprezentarea sa numerică (coeficientul lui x^n este mereu 1, altfel polinomul nu ar mai fi de grad n).

8.6.2 REPREZENTAREA POLINOMULUI

Un polinom poate fi reprezentat numeric în mai multe moduri. De exemplu, polinomul $\mathbf{x^4 + x + 1}$ (grad 4) poate fi scris:

- **Normal (0x3):** Corespunde la **0b0011**. Se omit termenii $\mathbf{x^4}$, $\mathbf{x^3}$, $\mathbf{x^2}$ și se reprezintă biții $\mathbf{x^1}$ **și** $\boldsymbol{x^0}$. Se folosește în împărțiri unde bitul cel mai semnificativ (**MSB**) este procesat primul.
- **Inversat (0xC):** Corespunde la **0b1100**. Este reprezentarea **în oglindă** a celei normale. Se folosește în implementări unde bitul cel mai puțin semnificativ (**LSB**) este procesat primul.
- **Inversat reciproc (Koopman) (0x9):** Corespunde la **0b1001**. Reprezintă biții polinomului, dar în loc să omiți termenul x^n, omiți termenul liber x^0.

8.6.3 VALOAREA INIȚIALĂ

Registrul CRC este inițializat cu această valoare la începutul calculului. Poate fi 0 sau orice altă valoare.

8.6.4 ORDINEA BIȚILOR

Specifică dacă biții din fiecare octet al mesajului sunt procesați de la **MSB** la **LSB** (direct) sau de la **LSB** la **MSB** (inversat / reflected).

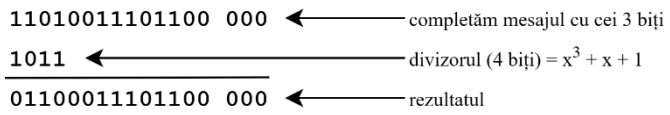
8.6.5 XOR FINAL

Valoarea calculată a **CRC-ului** este supusă unei operații **XOR** cu această valoare înainte de a fi returnată ca rezultat final.

8.7 EXEMPLU DE CALCUL

Dacă polinomul generator este de **grad n**, atunci **CRC-ul** are **n biți**. Polinomul generator are întotdeauna **n+1 biți** (de la $\mathbf{x^n}$ până la $\mathbf{x^0}$). Pentru calculul unui **CRC** pe **n biți**, se vor poziționa în partea stângă a mesajului inițial cei **n biți** ai polinomului generator. Acest lucru lasă loc pentru restul împărțirii (**CRC-ul**), care are exact $\mathbf{x^n}$ **biți**.

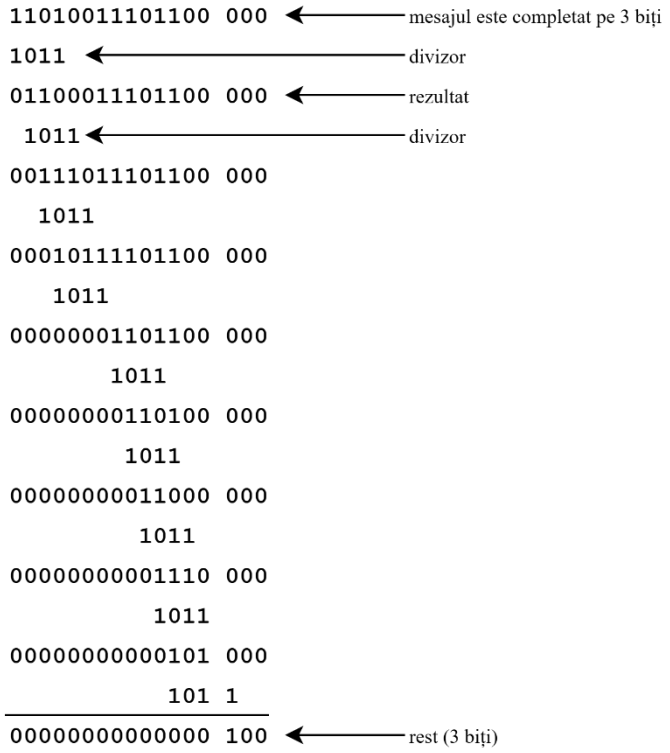
Se pornește de la mesajul inițial: **11010011101100.** Se completează mesajul inițial cu **n zerouri** corespunzătoare celor **n biți** de la **CRC**. Mai jos sunt prezentate calculele pentru un **CRC** pe **3 biți**:



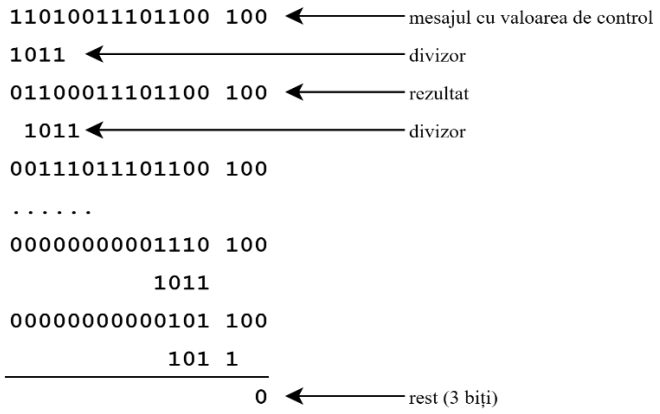
Dacă bitul **deasupra MSB-ului (**bitul din mesaj **corespunzător poziției MSB a generatorului** în segmentul curent) din divizor este **0**, nu trebuie făcute calcule.

Dacă bitul din mesajul inițial **deasupra MSB-ului** 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:



Restul obținut reprezintă **valoarea propriu-zisă** a funcției **CRC**. Pentru verificarea unui mesaj primit, acesta este **divizat cu polinomul generator**. Dacă nu există erori detectabile, restul obținut trebuie să fie **0**.



8.8 METODE DE CALCUL CRC

8.8.1 ALGORITM BIT-CU-BIT

Aceasta este implementarea **directă** a împărțirii polinomiale. Mesajul este procesat **bit cu bit**, iar pentru fiecare bit se efectuează o operație **XOR** cu polinomul generator, dacă este cazul. Deși este simplu de înțeles, este **cea mai lentă** metodă.

8.8.2 ALGORITM BAZAT PE TABELE (PRE-CALCULAT)

Pentru mărirea vitezei, se poate **pre-calcula** rezultatul împărțirii pentru toți cei **256 de octeți** posibili și stoca într-un tabel. La execuție, se procesează mesajul **octet cu octet**, folosind tabelul pentru a actualiza valoarea **CRC**, ceea ce elimină buclele de procesare per bit și **crește semnificativ** viteza.

Multe microcontrolere conțin module **hardware** dedicate pentru **calculul CRC**, oferind cea mai mare viteză. Alternativ, uneltele **software** moderne, cum ar fi **linker-ul** din IAR Embedded Workbench, pot calcula automat un **CRC** pentru o anumită secțiune de memorie (de obicei, întreaga memorie **Flash** a programului) și pot plasa rezultatul la o adresă cunoscută. Acest lucru este util pentru a verifica integritatea firmware-ului la pornire.

8.9 POINTERI ÎN IAR

8.9.1 POINTERI ȘI TIPURI DE MEMORIE

Pointerii sunt folosiţi 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 menţionat faptul că locaţia 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 **declaraţi fără** atribute de memorie. De exemplu, toate funcţiile din biblioteca standard sunt declarate fără specificarea explicită a tipului de memorie.

8.9.2 DIFERENȚE ÎNTRE TIPURI DE POINTERI

Un pointer trebuie să conţină **informaţia necesară** pentru a specifica **locaţia** unui anumit **tip de memorie**. Acest fapt denotă 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**.

8.9.2.1 POINTERI LA FUNCȚII

Dimensiunea unui pointer la funcţie este tot timpul **16** sau **24 de biţi**, iar aceștia pot adresa **întreaga memorie**. Reprezentarea internă a unui pointer la funcţie este **adresa** de la care începe **funcţia împărţită la 2**.

În IAR sunt disponibile următoarele tipuri de pointeri la funcţii:

Cuvânt-cheie	Interval de memorie	Dimensiune pointer	Tip index	Descriere
__nearfunc	0-0x1FFFE	2 octeți	`signed int`	Poate fi apelat de oriunde din memoria programului, dar trebuie să refere o locație din primii 128 kB i acelui spațiu de memorie.
__farfunc	0-0x7FFFFE	3 octeți	`signed long`	Poate fi apelat oriunde.

Tabelul 8.1 - Tipurile de pointeri la funcții disponibile în IAR

8.9.2.2 POINTERI LA DATE

Pointerii la date pot avea 3 dimensiuni: **8**, **16** sau **24 biţi**. Pointerii la date disponibili sunt:

Cuvânt-cheie	Dimensiune pointer	Spaţiul de memorie	Tipul indicelui	Intervalul de memorie
__tiny	1 octet	Data	`signed char`	0x0-0xFF
__near	2 octeţi	Data	`signed int`	0x0-0xFFFF
__far	3 octeţi	Data	`signed int`	0x0-0xFFFFFF
__huge	3 octeți	Data	`signed long`	0x0-0xFFFFFF
__tinyflash	1 octet	Code	`signed char`	0x0-0xFF
__flash	2 octeţi	Code	`signed int`	0x0-0xFFFF
__farflash	3 octeţi	Code	`signed int`	0x0-0xFFFFFF
__hugeflash	3 octeți	Code	`signed long`	0x0-0xFFFFFF
__eeprom	1 octer	EEPROM	`signed long`	0x0-0xFF
__eeprom	2 octeţi	EEPROM	`signed int`	0x0-0xFFFF

Tabelul 8.2 - Tipurile de pointeri la date disponibile în IAR

8.10 PROBLEME

8.10.1 SETĂRI IMPLEMENTARE CRC ÎN IAR

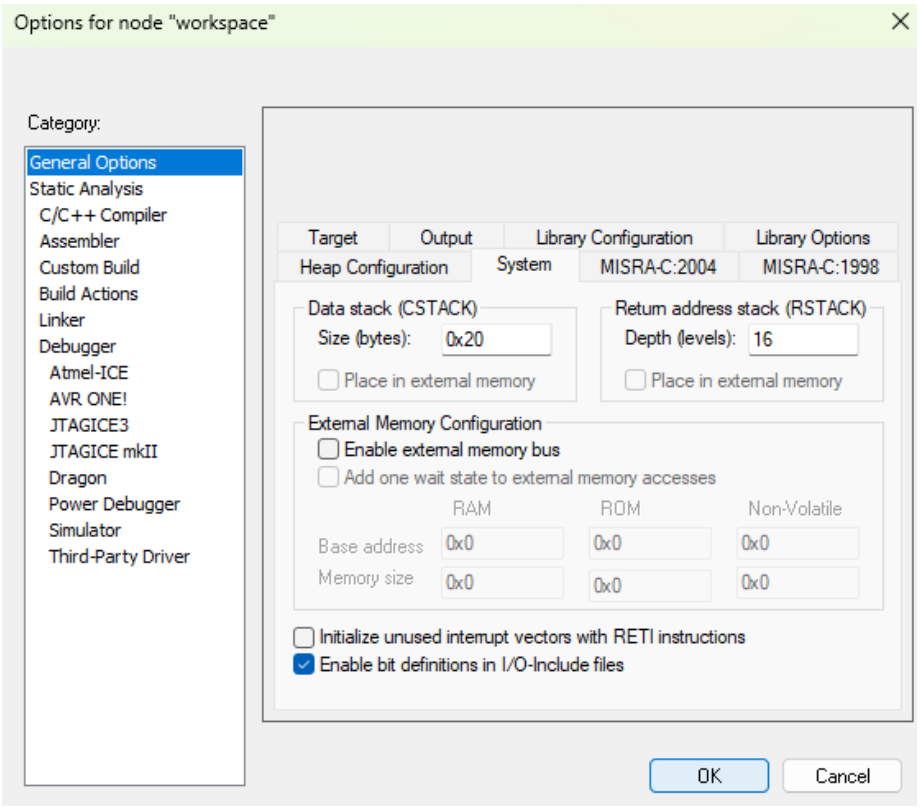


Figura 8.1 - Fereastra de opțiuni generale ale proiectului

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

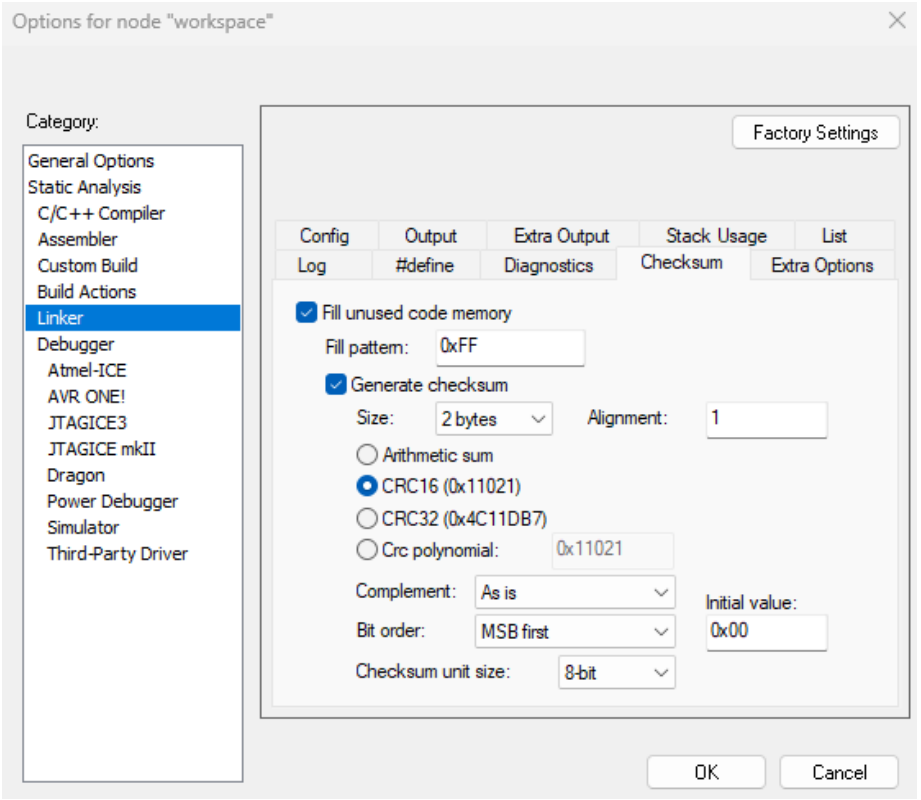


Figura 8.2 - Fereastra de opțiuni ale Linker-ului

Linker-ul din **IAR** poate fi configurat pentru a genera automat un **CRC** pentru codul programului. Acest lucru se face din **Options -> Linker -> Checksum**.

- **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**.

8.10.2 CRC-16

Cerință: Să se realizeze o aplicație care **calculează și verifică** valoarea **CRC-16** a unui bloc de date stocat în memoria **Flash**. Blocul de date are dimensiunea **128 KB** și este stocat în zona de adresă **0x000000 – 0x01FFFF** (spațiul Flash). Valoarea „reală" a **CRC-ului** pe 16 biți este salvată **imediat după blocul de date**, la adresele **0x020000 – 0x020003** (2 octeți CRC + un eventual padding / rezervă).
CRC-ul trebuie calculat folosind **două metode**:

1. **Metoda „bit cu bit"** (fără tabelă);
2. **Metoda cu tabele precalculate** (lookup table).

`crc16.h`

`crc16.h` reprezintă header-ul modulului de calcul al valorii **CRC pe 16 biți**, declarând funcțiile necesare pentru: calculul CRC-16 prin metoda clasică bit-cu-bit (**`crc16()`**), folosind un polinom specificat și o valoare inițială, calculul CRC-16 prin metoda cu tabelă precalculată (**`crc16wtable()`**), optimizată pentru performanță, selectarea ordinii de procesare a biților (**`enum BitOrder`** – **LSBF** sau **MSBF**), utilizarea polinoamelor standard definite pentru CRC-16 în reprezentare MSBF (**`0x1021`**) și LSBF (**`0x8408`**).

```
/*-------------------------------------------------------------------------
 * Fișier: crc16.h
 * Utilizat pentru declararea funcțiilor care calculează CRC pe 16 biți
 *-------------------------------------------------------------------------*/

#ifndef _CRC_H_
#define _CRC_H_

/*-------------------------------------------------------------------------
 * Includes
 *-------------------------------------------------------------------------*/

// General
#include <ioavr.h>
#include <inavr.h>
#include <stdint.h>

/*-------------------------------------------------------------------------
 * Data structures
 *-------------------------------------------------------------------------*/

/*
 * Enum care definește ordinea de procesare a biților.
 * LSBF: bitul cel mai puțin semnificativ este procesat primul.
 * MSBF: bitul cel mai semnificativ este procesat primul.
 */
enum BitOrder {LSBF, MSBF};

/*-------------------------------------------------------------------------
 * Public defines
 *-------------------------------------------------------------------------*/

// Polinomul standard pentru CRC-16 în reprezentare MSBF
#define CRC16_MSBF 0x1021

// Polinomul standard pentru CRC-16 în reprezentare LSBF
#define CRC16_LSBF 0x8408

/*-------------------------------------------------------------------------
 * Public (exported) functions
 *-------------------------------------------------------------------------*/

/*
 * Funcția calculează CRC-16 folosind o tabelă pre-calculată (lookup table)
 * și returnează valoarea finală calculată.
 */
uint16_t crc16wtable(uint16_t init_val_16, uint32_t adr_start,
                     uint32_t len, enum BitOrder ord);
```

```
/*
 * Funcția calculează CRC-16 folosind metoda clasică bit-cu-bit (fără
 * tabelă) și returnează valoarea finală calculată.
 */
uint16_t crc16(uint16_t polinom16, uint16_t init_val_16,
               uint32_t adr_start, uint32_t len, enum BitOrder ord);

#endif
```

crc16.c

Fișierul **crc16.c** definește funcțiile și tabelele necesare pentru calculul **CRC-16** atât prin metoda clasică **bit-cu-bit**, cât și prin metoda optimizată cu **tabele precalculate** (lookup table). Conține două tabele în memorie flash, **crc16tab_MSBF** și **crc16tab_LSBF**, pentru calculul rapid al **CRC-ului** în funcție de ordinea biților (**MSB-first** sau **LSB-first**). Funcția **crc16()** implementează algoritmul de calcul **bit-cu-bit**, citind datele din memoria flash și aplicând operații de **shiftare** și **XOR** cu polinomul specificat. Funcția **crc16wtable()** utilizează tabelele precalculate pentru a accelera calculul, reducând numărul de operații la fiecare octet procesat.

```
/*-------------------------------------------------------------------------
 * Fișier: crc16.c
 * Utilizat pentru definirea funcțiilor și a tablourilor folosite de CRC-16
 *-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
 * Includes
 *-------------------------------------------------------------------------*/

// General
#include "crc16.h"

/*-------------------------------------------------------------------------
 * Public variables
 *-------------------------------------------------------------------------*/

// Look-up table cu valori precalculate pentru calculul CRC-16 cu MSB-first
__flash const uint16_t crc16tab_MSBF[256] = {
  0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
  0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
  0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
  0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
  0x2462, 0x3443, 0x0420, 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, 0xfbbf, 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, 0xcffc, 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, 0xbfba, 0x8fd9, 0x9ff8,
  0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

/*
 * Look-up table cu valori precalculate pentru optimizarea calculului CRC-
 * 16 cu LSB-first
 */
__flash const uint16_t crc16tab_LSBF[256] = {
0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};

/*-------------------------------------------------------------------------
 * Public functions
 *-------------------------------------------------------------------------*/

/*
 * Funcția calculează CRC-16 folosind metoda clasică bit-cu-bit (fără
 * tabelă) și returnează valoarea finală calculată.
 */
uint16_t crc16(uint16_t polinom16, uint16_t init_val_16, uint32_t
               adr_start, uint32_t len, enum BitOrder ord)
{
  uint16_t crc = init_val_16; // Inițializează CRC cu valoarea de start
```

```
  uint16_t data = 0;          // Variabilă temporară pentru octetul curent
  while(len--) {
    uint16_t i;
   // Se extrage valoarea octetului de la adresa de start din memoria flash
    data = *(__farflash char *)adr_start;
    if (ord == MSBF) // Varianta cu shiftare spre MSB
    {
      data <<= 8; // Se aliniază octetul la MSB
      crc ^= data; // Se transferă și integrează datele în CRC
      adr_start++;
      for(i = 0; i < 8; ++i) {
        if(crc & 0x8000) // Se verifică dacă MSB este 1
          crc = (crc << 1) ^ polinom16;
        else
          crc = crc << 1;
      }
    }
    else { // Varianta cu shiftare spre LSB
      crc ^= data;
      adr_start++;
      for(i = 0; i < 8; ++i) {
        if(crc & 0x0001) // Se verifică dacă LSB este 1
          crc = (crc >> 1) ^ polinom16;
        else
          crc = crc >> 1;
      }
    }
  }
  return crc;
}

/*
 * Funcția calculează CRC-16 folosind o tabelă pre-calculată (lookup table)
 * și returnează valoarea finală calculată.
 */
uint16_t crc16wtable(uint16_t init_val_16, uint32_t adr_start,
                     uint32_t len, enum BitOrder ord)
{
  uint32_t counter;
  uint32_t crc = init_val_16; // Inițializează CRC cu valoarea de start
  for( counter = 0; counter < len; counter++)
    if(ord == MSBF) // Varianta cu MSB
    /* 1. Shiftare la stânga CRC cu 8 biți
     * 2. XOR cu valoarea din tabelul MSBF corespunzătoare.
     * Indexul în tabel se obține astfel:
     * - (crc >> 8) = octetul superior al CRC curent
     * - XOR cu byte-ul citit din memorie (de la adr_start)
     * - & 0x00FF pentru a păstra doar 8 biți
     * 3. Se adună valoarea din tabel, care reprezintă efectul acelui byte
     * asupra CRC-ului.
     */
      crc = (crc << 8) ^ crc16tab_MSBF[((crc >> 8) ^ *(__farflash
            char *)adr_start++) & 0x00FF];
    else // Varianta cu LSB
      crc = (crc >> 8) ^ crc16tab_LSBF[(crc ^ *(__farflash char *)
            adr_start++) & 0x00FF];
  return crc;
}
```

`main.c`

Fișierul **`main.c`** reprezintă punctul de intrare al aplicației **CRC-16**. În cadrul acestuia, se citește valoarea reală a CRC-ului stocată în memoria flash (**`real_crc`**), apoi se calculează CRC-ul pentru aceeași zonă de memorie prin două metode diferite:

- **`crc16()`** – metoda clasică **bit-cu-bit**, care procesează fiecare bit în funcție de polinomul specificat;
- **`crc16wtable()`** – metoda optimizată cu **tabele precalculate** (look-up table), care reduce numărul de operații necesare pentru calculul CRC-ului.

Ambele metode procesează datele din memoria flash, cu ordinea bitilor specificată (MSB-first). Programul nu conține o buclă de execuție funcțională, rămânând blocat în bucla infinită **`while(1)`** după efectuarea calculelor, permițând astfel analizarea și compararea rezultatelor obținute prin cele două tehnici.

```
/*-------------------------------------------------------------------------
 * Fișier: main.c
 * Fișierul principal de rulare a aplicației CRC-16
 *-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------
 * Includes
 *-------------------------------------------------------------------------*/

// General
#include "crc16.h"

void main( void )
{
  /*
   * Se calculează atât CRC-ul folosind funcția standard crc16(), cât și cu
   * crc16wtable(), versiunea cu look-up table.
   */
  uint16_t real_crc=*(__farflash uint16_t *)(0x020000-2);
  uint16_t my_crc16=crc16(CRC16_MSBF,0,0,(0x020000-2),MSBF);
  uint16_t my_crc16_t=crc16wtable(0,0,(0x020000-2),MSBF);

  while(1)
  {
  }
}
```