Universitatea Națională de Știință și Tehnologie POLITEHNICA București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem de Sortare și Depozitare a Pachetelor pe o Bandă

Nume: Drugea Alina-Nicoleta
Grupa: 422E

mai 2025

Cuprins

Lista figurilor

Lista tabelelor

Introducere

În fiecare zi, milioane de pachete sunt procesate în depozite și în centre logistice de curierat din întreaga lume. Fiecare pachet trebuie sortat, direcționat și depozitat corect fără erori sau întârzieri. Într-un mediu atât de dinamic, contribuția umană poate încetini procesul, iar automatizarea devine esențială.

Proiectul prezintă proiectarea și realizarea unui Sistem de Sortare și Depozitare a Pachetelor pe o Bandă, care funcționează pe baza unui cod binar de identificare și a unui senzor de greutate. Sistemul detectează fiecare pachet, îi analizează caracteristicile și decide zona de destinație.

Pentru scrierea codului am utilizat programul CodeVisionAVR V4.03, iar pentru testarea lui am utilizat AVR Studio. Pentru configurarea proiectului am ales un microcontroler de tip ATmega164 si frecvența ceasului de 10MHz.

Pentru a asigura desfășurarea corectă a pașilor din procesul de sortare și depozitare, am implementat un circuit logic secvențial cu un număr finit de stări. Astfel, circuitul logic secvențial are 5 stări:

1. Starea 0 - detectarea pachetului
2. Starea 1 - citirea codului și a greutății coletului
3. Starea 2 - decizia și activare direcționare
4. Starea 3 - verificare
5. Starea 4 - procesarea următorului colet

Odată cu sosirea primului colet, sistemul utilizează senzorii montați pe porturile A și B pentru a extrage codul pachetului și a măsura greutatea. Pe baza informațiilor, programul va determina zona corespunzătoare sau va identifica eventualele nereguli asociate coletului. Acesta va afișa cu ajutorul LED-urilor etapele de procesare. După verificare și redirecționarea sa, se va prelua un alt pachet.

Capitolul I Date generale despre proiect

Sistem de Sortare și Depozitare a Pachetelor pe o Bandă este conceput pentru a automatiza procesul de identificare, clasificare și direcționare a coletelor.

Specificații tehnice:

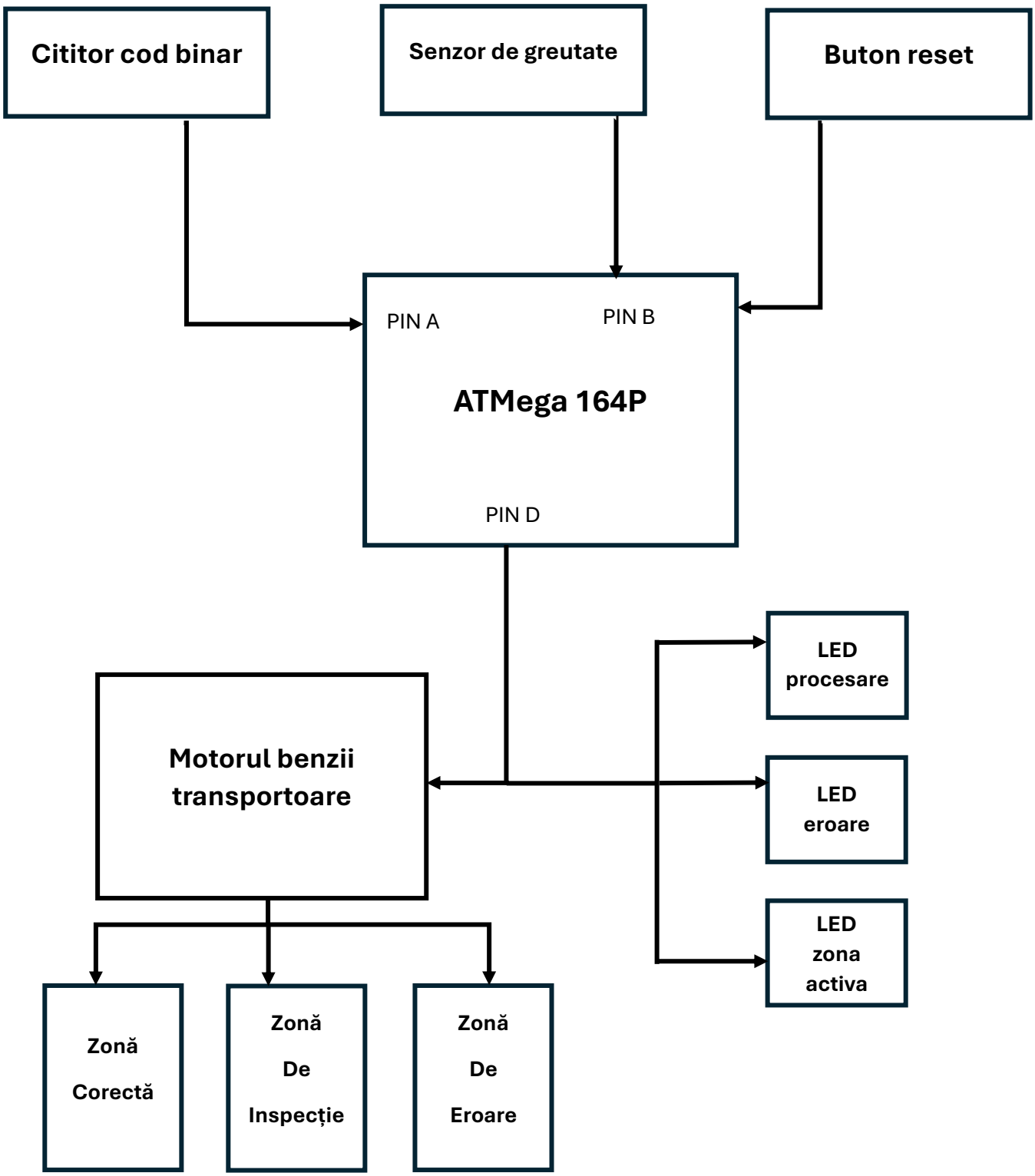
- Fiecare colet este identificat printr-un cod pe 4 biți, care indică zona de destinație.
- Limita maximă admisă pentru greutatea coletului este de 5 kg , detectată printr-un senzor de greutate.
- Sistemul utilizează un microcontroler ATmega164p pentru procesarea datelor și controlul componentelor.

Componenta centrală este reprezentată de circuitele logice care interacționează cu senzorii de greutate și de cod, motorul benzii transportoare, afișajul LED și butonul de reset. Datele de intrare sunt colectate de la senzori, procesate de automatul cu stări finite (FSM), iar comanda rezultată controlează direcționarea pachetelor pe banda transportoare.

La detectarea unui colet, sistemul preia codul său binar pe 4 biți și măsoară greutatea acestuia. Aceste date sunt analizate de FSM care decide direcția corectă de redirecționare. Dacă greutatea depășește limita admisă (5 kg), coletul este trimis către zona de inspecție. Dacă codul nu este valid, coletul este redirecționat spre zona de eroare. Motorul benzii transportoare este controlat automat pentru a realiza aceste operațiuni, iar afișajul LED indică în timp real starea sistemului (procesare, eroare, direcție activă).

Sistemul include următoarele componente esențiale:

- Microcontrolerul ATmega164p- reprezintă unitatea centrală de control, responsabilă pentru citirea datelor de la senzori, interpretarea acestora .
- Senzorul de greutate -conectat la un pin, acesta transmite microcontrolerului informația privind masa pachetului.
- Cititor cod binar pe 4 biți – colectează informațiile despre destinația coletului, codificată sub forma unui număr binar de 4 biți.
- Buton de reset – permite revenirea sistemului în starea inițială în caz de blocaj, asigurând continuitatea funcționării și minimizarea timpilor de nefuncționare.
- Motor al benzii transportoare – direcționează pachetul către zona corectă, în funcție de condițiile coletului.
- Afișaj cu LED-uri indicatoare - semnalează starea sistemului (procesare, eroare, zona de destinație activă).



Figură 1.1 Schema bloc a sistemului

Capitolul II Descrierea cazurilor de utilizare

Funcționarea sistemului de sortare și depozitare a pachetelor este guvernată de un automat cu stări finite, care controlează fluxul de procesare într-o manieră secvențială.

Starea 0: Sistemul va detecta coletul care intră pe banda transportoare.

Starea 1: La apariția unui colet, sistemul citește informațiile de la senzorul de greutate respectiv, de la codul binar, acestea fiind utilizate în etapele următoare ale procesului. În cazul în care un alt colet este deja în curs de procesare, sistemul va aștepta finalizarea acestuia înainte de a prelua noul colet.

Starea 2: În această etapă are loc luarea deciziei privind direcționarea coletului, pe baza datelor furnizate de senzori. Mai întâi, se verifică lungimea codului binar asociat coletului. Dacă acesta depășește 4 biți, coletul este considerat invalid și este direcționat către zona de eroare. Dacă codul este corect, se evaluează și greutatea coletului. În cazul în care acesta se încadrează în limita admisă, coletul este trimis către zona de destinație indicată de cod. În caz contrar, coletul este direcționat către zona de inspecție pentru o verificare suplimentară.

Starea 3: Sistemul aplică decizia asupra coletului procesat și transmite informația la led-urile indicatoare.

Starea 4: Sistemul revine la starea de repaus sau reset, pregătindu-se pentru procesarea următorului colet.

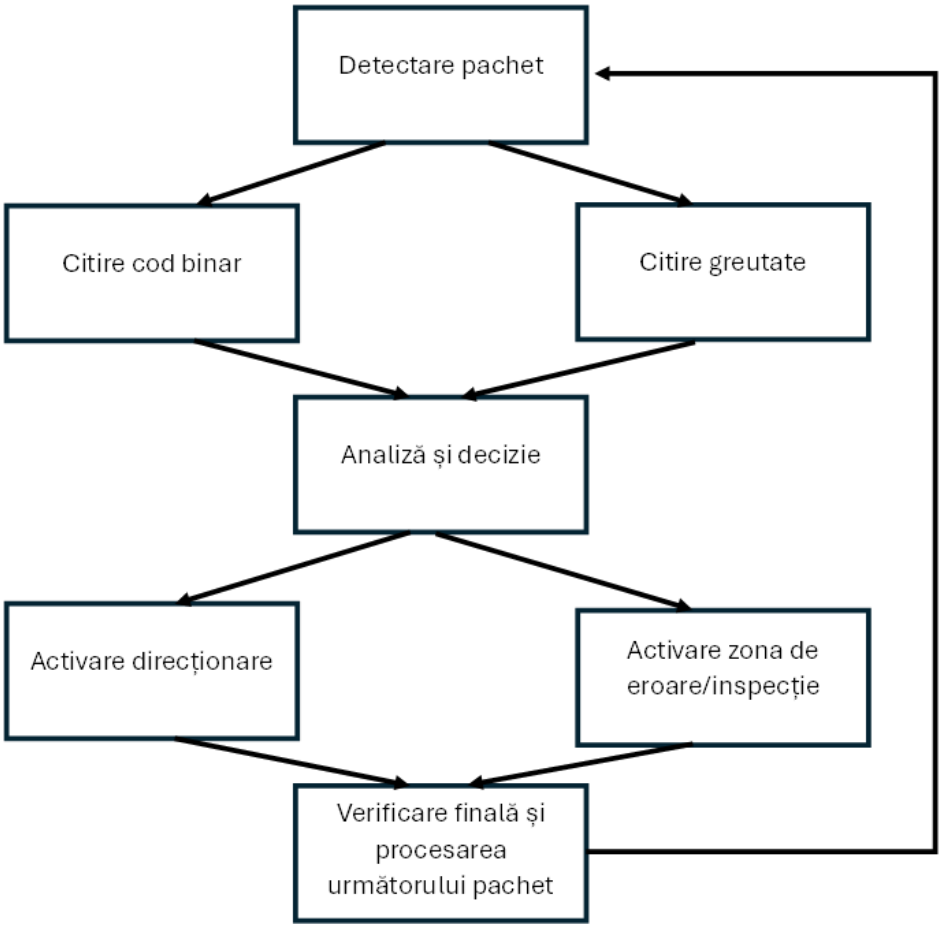


Figura 1.2 Diagrama cazurilor de utilizare a sistemului

Capitolul III Proiectarea și implementarea sistemului

3.1 Etapa de proiectare

Fiecare colet din depozit este marcat cu un cod binar, care trebuie sa corespundă uneia din valorile de mai jos. Aceste trebuie să aibă și o greutate cuprinsă între 0 și 5 kilograme pentru a fi validate.

COD BINAR	DESTINAȚIE
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

Tabel 3.1 Cod binar – Destinație

Codul sistemului a fost implementat în CodeVision AVR, folosind limbajul C și cu următoarele setări:

- Chip:
  -Selectat din meniul Chip Settings: ATmega164P
  -Frecvența ceasului (Clock): 10 MHz

- Timers/Counters:
  -Clock value: 9.766 kHz
  -Overflow Interrupt activat (bifat)
  -Timer initial value: 3C (hexazecimal)

- Port Settings:

PORT A								
Data Direction	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	IN	IN	IN	IN	IN	IN	IN	IN
Pullup/Output Value	P	P	P	P	P	P	P	P

Tabel 3.2 Configurare port A

PORT B								
Data Direction	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	IN	IN	IN	IN	IN	IN	IN	IN
Pullup/Output Value	P	P	P	P	P	P	P	P

Tabel3.3 Configurare port B

PORT D								
Data Direction	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	OUT	OUT	OUT	OUT	OUT	OUT	OUT	OUT
Pullup/Output Value	1	1	1	1	1	1	1	1

Tabel 3.4 Configurare port D

Tabelul pentru circuitul logic combinațional care stă la baza decodificării codului pachetului și deplasarea acestuia în etapa următoare.

COD BINAR CU „0” ACTIV								GREUTATE	IEȘIRE LED	SEMNIFICAȚIE
SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0			
1	1	1	1	1	1	1	1	<11111010	01010101	VALID
1	1	1	1	1	1	1	0	<11111010	01010101	VALID
1	1	1	1	1	1	0	1	<11111010	01010101	VALID
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	<11111010	01010101	VALID
1	1	1	1	0	0	0	0	<11111010	01010101	VALID
1	1	1	1	0	0	0	0	>11111010	00000000	INVALID masă depășită
1	1	1	0	0	0	0	0	orice masă	00000000	INVALID cod greșit

Tabel 3.5 Tabel de adevăr pentru clc

3.2 Alegerea porturilor

Portul A este configurat ca intrare și are rolul de cititor al codului binar inscripționat pe colet. Acesta preia informația necesară pentru identificarea zonei de destinație.

Portul B este configurat ca intrare și are două funcții principale: senzorul de greutate și detectarea stării butonului de reset.

Port B							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Buton reset	Senzor de greutate						

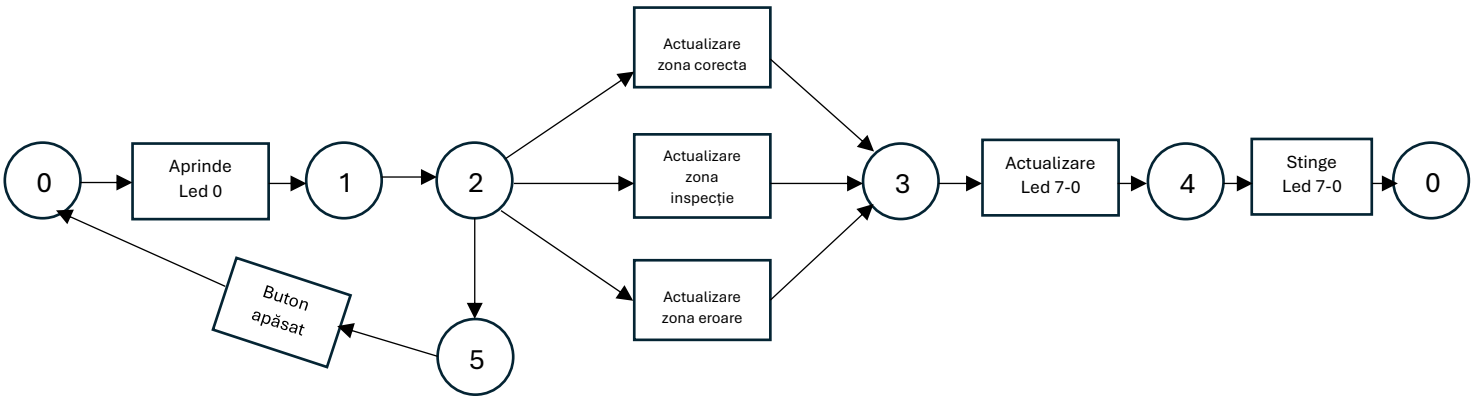
Tabel 3.6 Semnificația intrării portului B

Portul D este configurat ca ieșire și este utilizat pentru controlul LED-urilor, care indică vizual rezultatul procesării fiecărui colet.

Port D								
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit0	Semnificație
1	1	1	1	1	1	1	0	Procesare
0	0	0	0	0	0	0	0	Eroare
0	1	0	1	0	1	0	1	Destinație corectă a coletului
1	1	1	1	1	1	1	1	Procesarea următorului colet

Tabel 3. 7 Semnificația LED-urilor

3.3 Diagramele secvențiale corespunzătoare codului sursă



Figură 3.1 Diagrama secvenţială corespunzătoare codului sursă

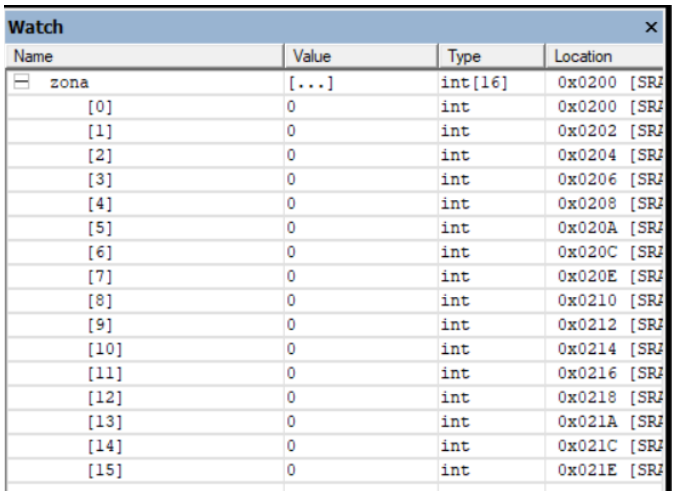
0- Detectare pachet
1- Citire greutate si cod
2- Decizie
3- Verificare finala
4- Procesarea următorului colet
5- Eroare

3.4 Implementarea codului

La începutul codului, au fost definite variabilele necesare pentru implementarea programului. Variabilele greutate, N, masca_g, masca_b sunt descrise prin directive #define și sunt etichete pentru valori fixe. Variabila stare_sistem este folosită în automatul finit de stare pentru a controla logica de funcționare a programului. Variabilele greutate_colet, colet, cod_g_b, cod sunt utilizate pentru a reține informații specifice fiecărui colet.

Tabloul zona [16] este un vector cu 16 elemente, în care fiecare poziție reprezintă o zonă de destinație posibilă. La fiecare trimitere a unui colet către una dintre aceste zone, valoarea

corespunzătoare din vector este incrementată, având astfel rolul de contor pentru numărul de colete direcționate către fiecare zonă.



Watch

Name	Value	Type	Location
zona	[...]	int[16]	0x0200 [SRJ
[0]	0	int	0x0200 [SRJ
[1]	0	int	0x0202 [SRJ
[2]	0	int	0x0204 [SRJ
[3]	0	int	0x0206 [SRJ
[4]	0	int	0x0208 [SRJ
[5]	0	int	0x020A [SRJ
[6]	0	int	0x020C [SRJ
[7]	0	int	0x020E [SRJ
[8]	0	int	0x0210 [SRJ
[9]	0	int	0x0212 [SRJ
[10]	0	int	0x0214 [SRJ
[11]	0	int	0x0216 [SRJ
[12]	0	int	0x0218 [SRJ
[13]	0	int	0x021A [SRJ
[14]	0	int	0x021C [SRJ
[15]	0	int	0x021E [SRJ

Figură 3.2 Vector zona de destinatie

Zona_de_eroare și zona_de inspecție sunt contoare care înregistrează câte colete au fost trimise în zona de eroare respectiv, zona de inspecție.

Tabloul OUT este utilizat pentru a controla semnalizarea led-urilor în funcție de starea sistemului.

Variabila buton este conectată cu butonul din sistem, de la pinul B.

În acest cod am utilizat un mecanism switch-case pentru a implementarea procesului secvențial, gestionând trecerea prin diferitele etape ale procesării coletului în funcție de variabila stare_sistem. Fiecare caz(case) corespunde unei stări specifice în care se realizează operații distincte, iar trecerea între stări se face prin modificarea valorii lui stare_sistem.

În „ case 0” ,se citesc variabilele din pinii A și B pentru a fi detectat coletul.

În „case 1”, se semnalează că pachetul se procesează prin actualizarea variabilei stare_procesare. Variabila cod primește valoarea din colet, iar pentru greutatea coletului se face o operație logică cu masca_g.

În „case 2”, se analizează datele coletului. Dacă codul binar este în intervalul validă[0,15] și greutatea coletului este sub limita prestabilită(<=5), atunci se actualizează numărul coletelor pentru zona de destinație corespunzătoare și se marchează starea procesării. Dacă codul depășește limita validă, se consideră că este o eroare, starea procesării este setată pentru zona de eroare, iar contorul erorilor crește. În cazul în care codul e valid, dar greutatea depășește limita admisă, coletul este direcționat spre inspecție, starea procesării este tot de eroare, dar se incrementează contorul pentru zona de inspecție.

În „case 3”, se afișează pe Led-urile indicatoare de pe pinul D, rezultatul procesării coletului.

În „case 4”, se resetează starea sistemului pentru a permite procesarea unui nou pachet, iar portul D se setează valoarea 0xFF, ce indică resetarea afișajului.

În cazul în care sistemul se blochează sau ajunge într-o stare neașteptată, secțiunea default permite resetarea manuală a automatelor prin apăsarea butonului de pe pinul B, asigurând funcționarea corecta a programului.

Capitolul IV Testarea codului sursă

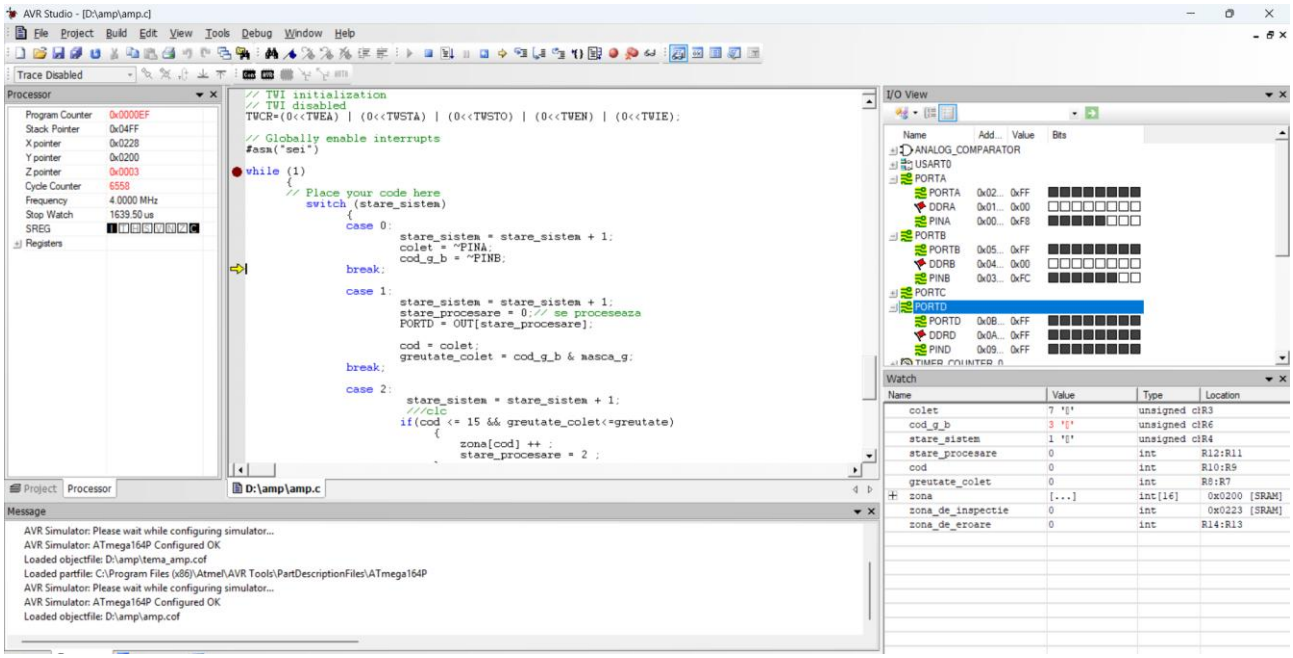
Pentru a verifica funcționalitatea codului, au fost realizate teste cu ajutorul aplicației AVR Studio, urmărind fiecare etapă.

4.1. Testare cod valid- greutate validă

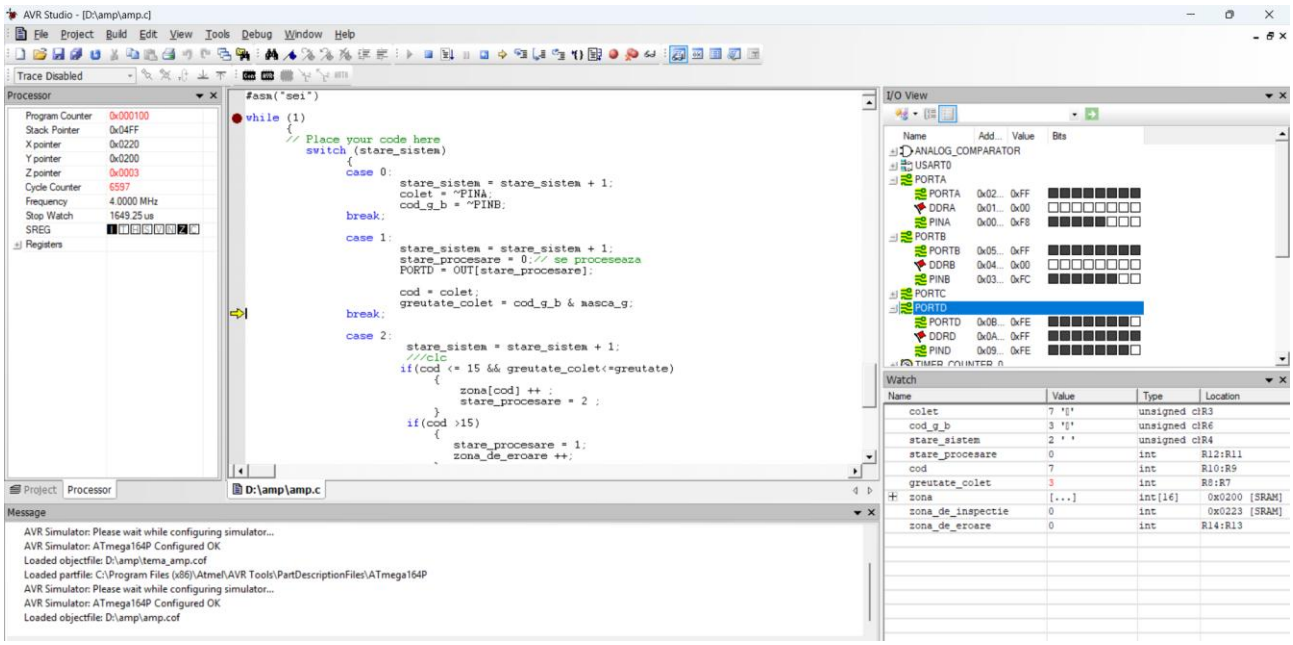
Condiții de testare:

- Cod colet între 0 și 15
- Greutate colet <=5

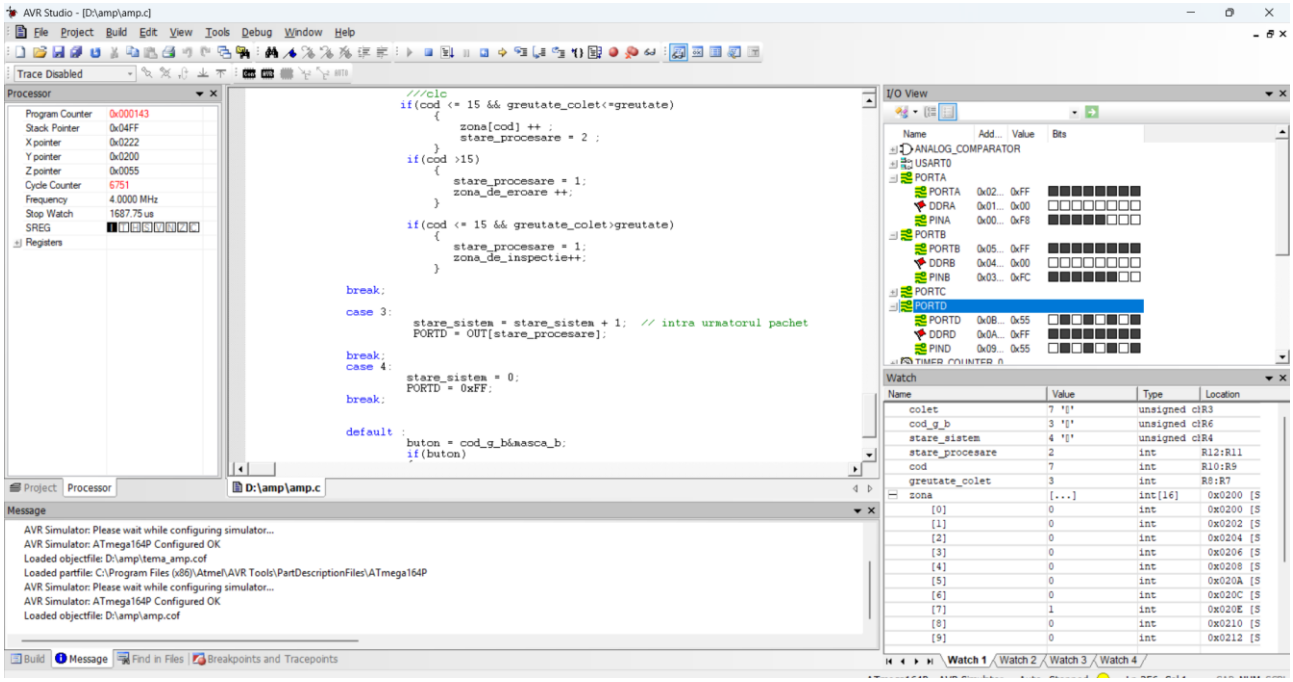
Au fost alese următoarele valori: cod = 7 ; greutate = 3;



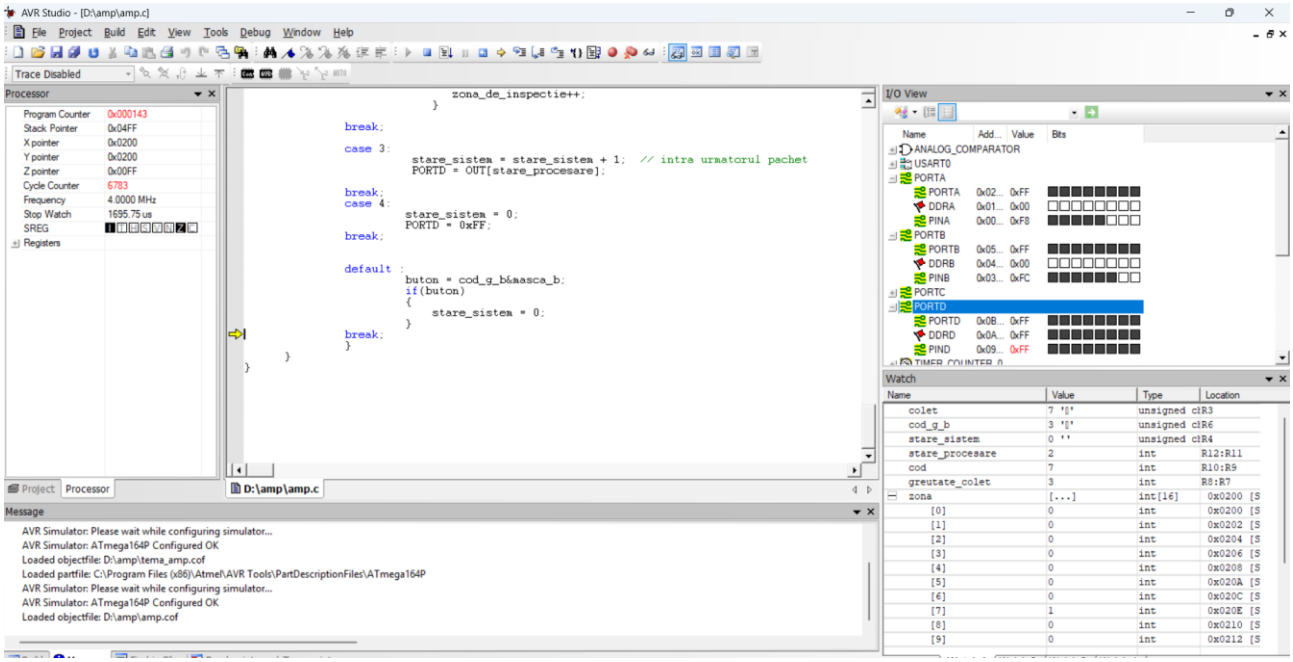
Figură 4.1 Detectarea coletului



Figură 4.2 Citirea specificațiilor



Figură 4.3 Direcționarea coletului și afișarea



Figură 4.4 Așteptarea următorului colet

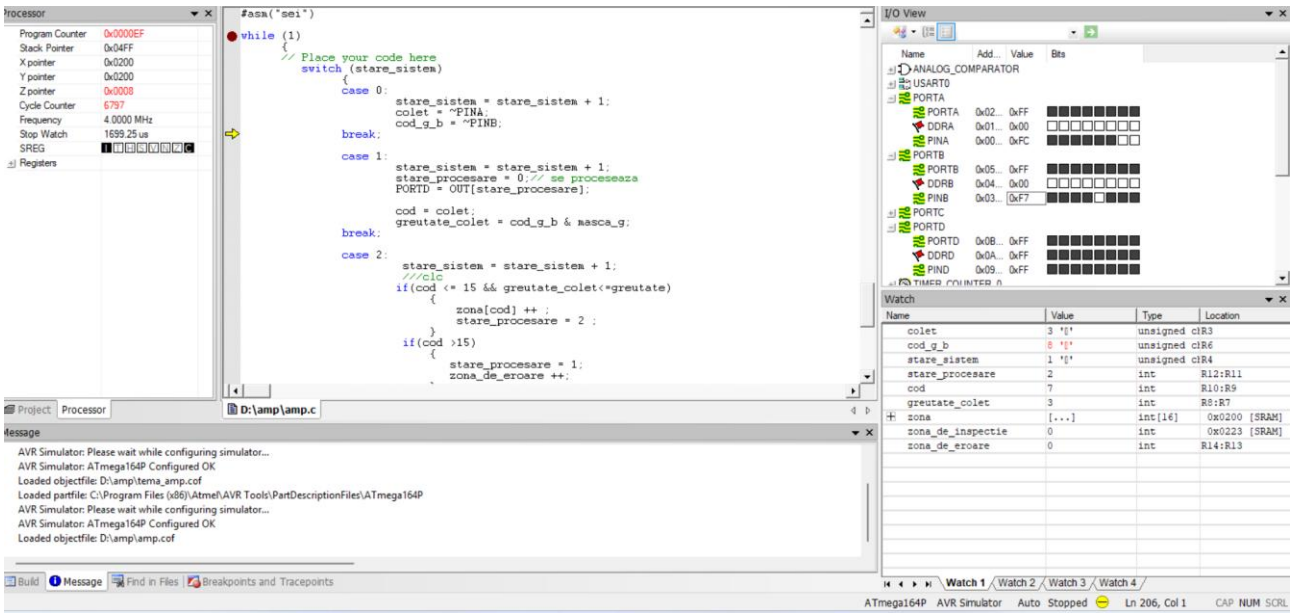
Rezultatul procesării: coletul este direcționat în zona 7 și afișajul este pentru zona corectă.

4.2. Testare cod valid- greutate invalidă

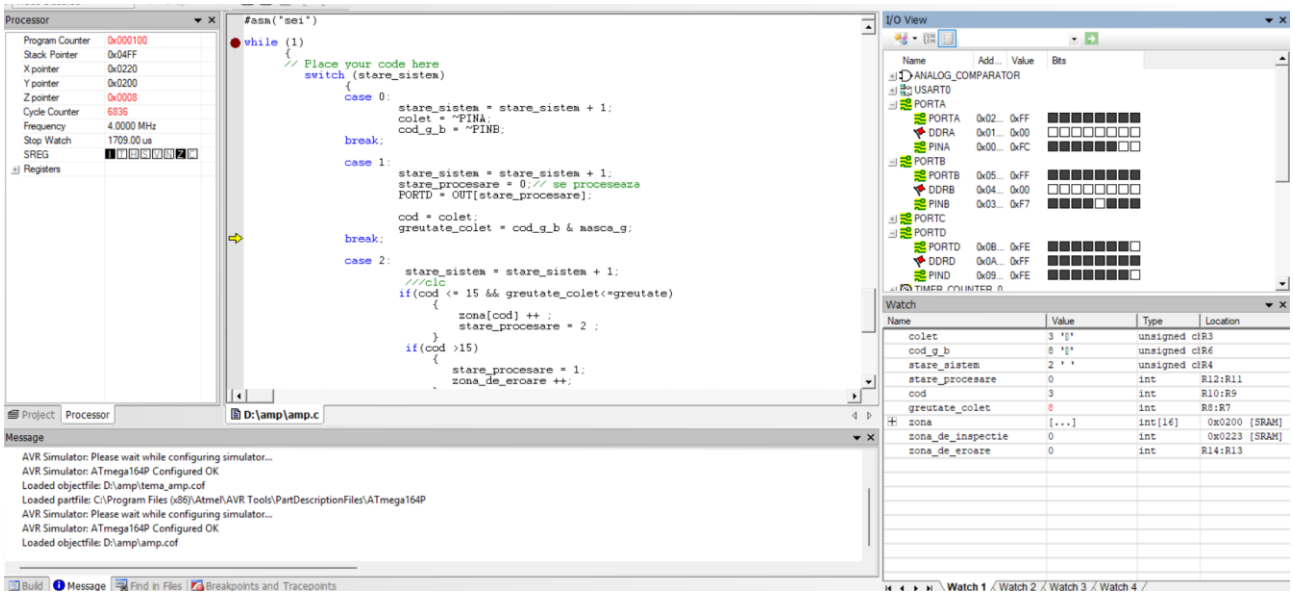
Condiții de testare:

- Cod colet între 0 și 15
- Greutate colet >5

Au fost alese următoarele valori: cod = 3; greutate = 8;



Figură 4.5 Detectarea coletului



Figură 4.6 Citirea specificațiilor

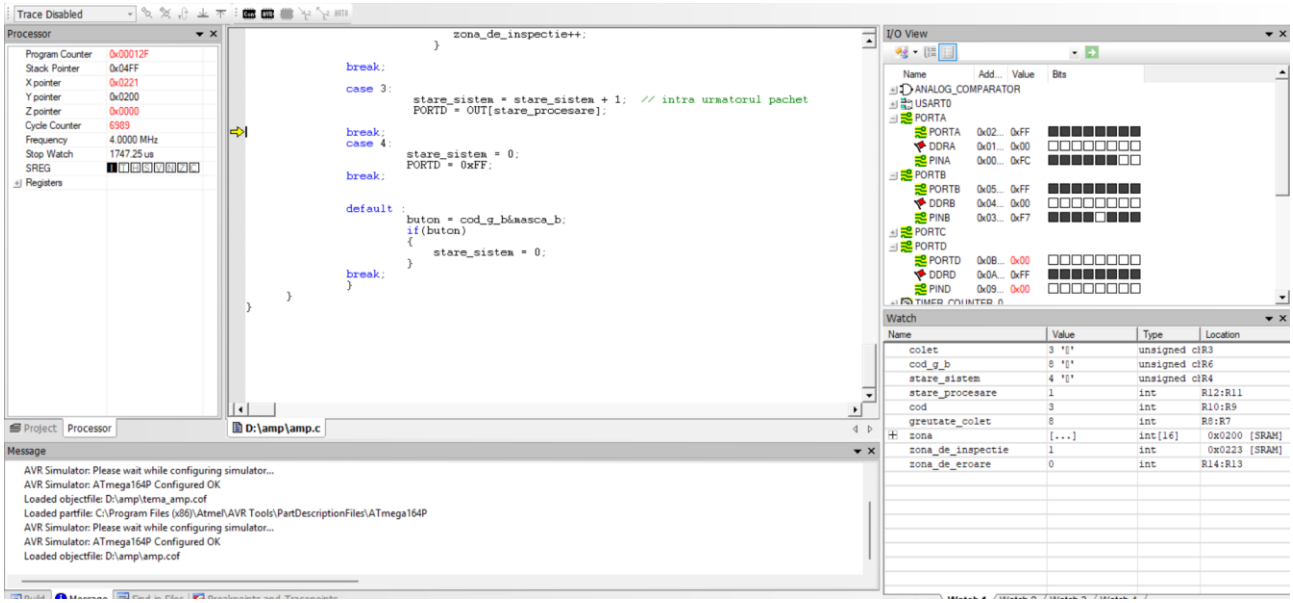


Figura 4.7 Direcționarea coletului și afișarea rezultatului pentru inspectie

Rezultatul procesării: coletul este direcționat în zona de inspecție și afișajul este pentru eroare.

4.3. Testare cod invalid

Condiții de testare:

- Cod colet >15

Au fost alese următoarele valori: cod = 16; greutate = 2;

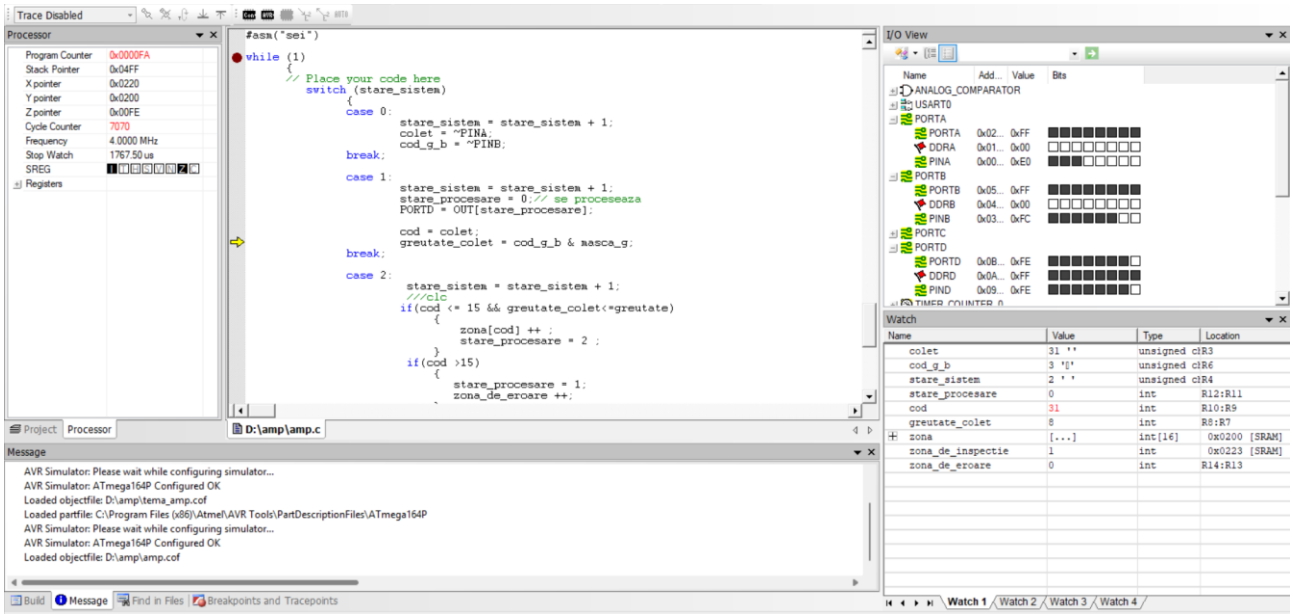


Figura 4.8 Detectarea si citirea coletului

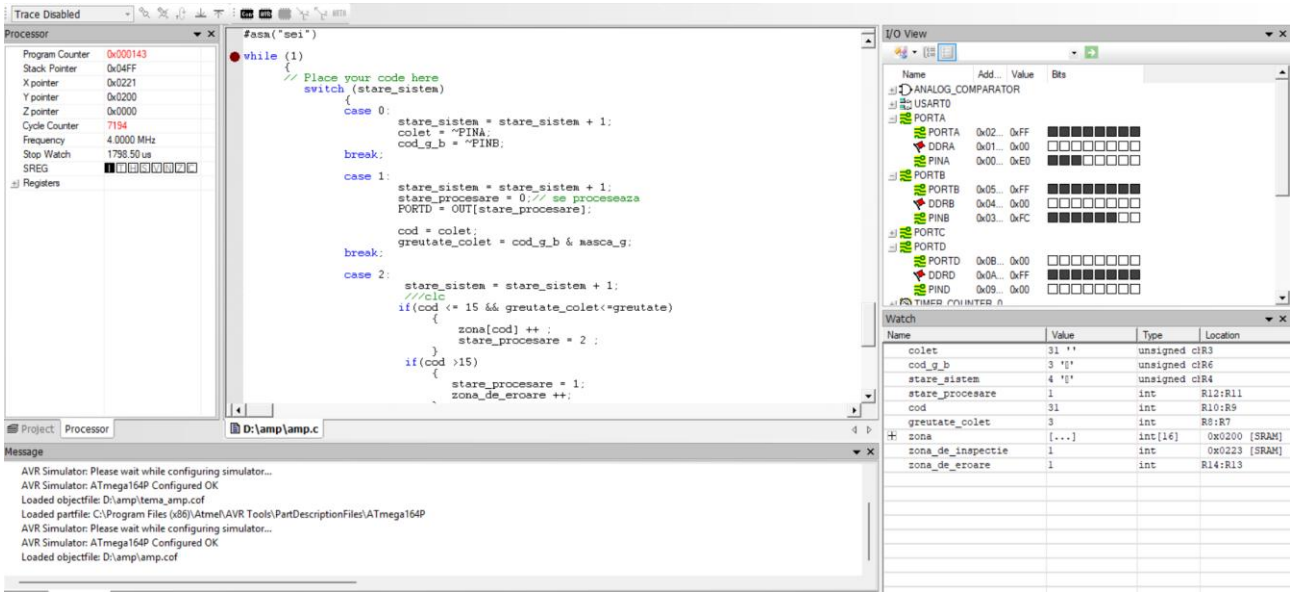


Figura 4.9 Direcționarea coletului și afișarea rezultatului pentru eroare

Rezultatul procesării: coletul este direcționat în zona de eroare și afișajul este pentru eroare.

Concluzii

În cadrul acestui proiect am proiectat și implementat un Sistem de Sortare și Depozitare a Pachetelor pe o Bandă, menit să optimizeze procesul de clasificare și depozitare a acestora pe o bandă transportoare. Scopul principal a fost realizarea unui sistem automat care să preia, să analizeze și să direcționeze pachetele în funcție de greutate și codul de identificare.

Proiectul folosește un automat finit de stare, care permite o organizare clară și structurată a fluxului operațional al sistemului, fiecare stare având un rol bine definit în preluarea, procesarea și direcționarea coletelor.

Implementarea în limbajul C, utilizând microcontrolerul, a demonstrat o interacțiune eficientă între componentele hardware și logica software.

Testele efectuate pentru cele trei scenarii posibile au confirmat corectitudinea și stabilitatea sistemului, comportamentul acestuia fiind fiabil.

În plus, proiectul a reprezentat o oportunitate de consolidare a cunoștințelor practice în domeniul electronicii digitale și programării microcontrolerelor.

Anexe

Cod sursă

```
/*****************************************************
This program was created by the
CodeWizardAVR V4.03

Automatic Program Generator

© Copyright 1998-2024 Pavel Haiduc, HP InfoTech S.R.L.

http://www.hpinfotech.ro

Project :
Version :
Date    : 17.05.2025
Author  :
Company :
Comments:


Chip type               : ATmega164
Program type            : Application
AVR Core Clock frequency: 10,000000 MHz
Memory model            : Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

// I/O Registers definitions
#include <mega164.h>

#include <math.h>
#define greutate 5
#define N 4
#define masca_g 0x7F
#define masca_b 0x80
// Declare your global variables here
unsigned char stare_sistem=0;
char  colet, cod_g_b;
int greutate_colet, cod ;
int zona[16]; // vector cu zone de destinatie

char OUT[] = {0XFE,0X00, 0x55} ;
int stare_procesare, zona_de_eroare=0;
int zona_de_inspectie = 0;


int buton = 0;
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x3C;
// Place your code here

}
void clc()
{
}

void main(void)
{
```

```
// Declare your local variables here

// Clock Oscillator division factor: 1
#pragma optsize-
CLKPR=(1<<CLKPCE);
CLKPR=(0<<CLKPCE) | (0<<CLKPS3) |
(0<<CLKPS2) | (0<<CLKPS1) |
(0<<CLKPS0);
#ifdef _OPTIMIZE_SIZE_
#pragma optsize+
#endif

// Input/Output Ports initialization
// Port A initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In
Bit3=In Bit2=In Bit1=In Bit0=In
DDRA=(0<<DDA7) | (0<<DDA6) |
(0<<DDA5) | (0<<DDA4) | (0<<DDA3) |
(0<<DDA2) | (0<<DDA1) | (0<<DDA0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P
Bit2=P Bit1=P Bit0=P
PORTA=(1<<PORTA7) | (1<<PORTA6) |
(1<<PORTA5) | (1<<PORTA4) |
(1<<PORTA3) | (1<<PORTA2) |
(1<<PORTA1) | (1<<PORTA0);

// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In
Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) |
(0<<DDB5) | (0<<DDB4) | (0<<DDB3) |
(0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P
Bit2=P Bit1=P Bit0=P
PORTB=(1<<PORTB7) | (1<<PORTB6) |
(1<<PORTB5) | (1<<PORTB4) |
(1<<PORTB3) | (1<<PORTB2) |
(1<<PORTB1) | (1<<PORTB0);

// Port C initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In
Bit3=In Bit2=In Bit1=In Bit0=In
DDRC=(0<<DDC7) | (0<<DDC6) |
(0<<DDC5) | (0<<DDC4) | (0<<DDC3) |
(0<<DDC2) | (0<<DDC1) | (0<<DDC0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T
Bit3=T Bit2=T Bit1=T Bit0=T
PORTC=(0<<PORTC7) | (0<<PORTC6) |
(0<<PORTC5) | (0<<PORTC4) |
(0<<PORTC3) | (0<<PORTC2) |
(0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=Out Bit6=Out Bit5=Out
Bit4=Out Bit3=Out Bit2=Out Bit1=Out
Bit0=Out
DDRD=(1<<DDD7) | (1<<DDD6) |
(1<<DDD5) | (1<<DDD4) | (1<<DDD3) |
(1<<DDD2) | (1<<DDD1) | (1<<DDD0);
// State: Bit7=1 Bit6=1 Bit5=1 Bit4=1 Bit3=1
Bit2=1 Bit1=1 Bit0=1
PORTD=(1<<PORTD7) | (1<<PORTD6) |
(1<<PORTD5) | (1<<PORTD4) |
(1<<PORTD3) | (1<<PORTD2) |
(1<<PORTD1) | (1<<PORTD0);

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 9,766 kHz
// Mode: Normal top=0xFF
// OC0A output: Disconnected
// OC0B output: Disconnected
// Timer Period: 20,07 ms
```

```
TCCR0A=(0<<COM0A1) | (0<<COM0A0) |
(0<<COM0B1) | (0<<COM0B0) |
(0<<WGM01) | (0<<WGM00);

TCCR0B=(0<<WGM02) | (1<<CS02) |
(0<<CS01) | (1<<CS00);

TCNT0=0x3C;

OCR0A=0x00;

OCR0B=0x00;


// Timer/Counter 1 initialization

// Clock source: System Clock

// Clock value: Timer1 Stopped

// Mode: Normal top=0xFFFF

// OC1A output: Disconnected

// OC1B output: Disconnected

// Noise Canceler: Off

// Input Capture on Falling Edge

// Timer1 Overflow Interrupt: Off

// Input Capture Interrupt: Off

// Compare A Match Interrupt: Off

// Compare B Match Interrupt: Off

TCCR1A=(0<<COM1A1) | (0<<COM1A0) |
(0<<COM1B1) | (0<<COM1B0) |
(0<<WGM11) | (0<<WGM10);

TCCR1B=(0<<ICNC1) | (0<<ICES1) |
(0<<WGM13) | (0<<WGM12) | (0<<CS12) |
(0<<CS11) | (0<<CS10);

TCNT1H=0x00;

TCNT1L=0x00;

ICR1H=0x00;

ICR1L=0x00;

OCR1AH=0x00;

OCR1AL=0x00;

OCR1BH=0x00;

OCR1BL=0x00;


// Timer/Counter 2 initialization

// Clock source: System Clock

// Clock value: Timer2 Stopped

// Mode: Normal top=0xFF

// OC2A output: Disconnected

// OC2B output: Disconnected

ASSR=(0<<EXCLK) | (0<<AS2);

TCCR2A=(0<<COM2A1) | (0<<COM2A0) |
(0<<COM2B1) | (0<<COM2B0) |
(0<<WGM21) | (0<<WGM20);

TCCR2B=(0<<WGM22) | (0<<CS22) |
(0<<CS21) | (0<<CS20);

TCNT2=0x00;

OCR2A=0x00;

OCR2B=0x00;


// Timer/Counter 0 Interrupt(s) initialization

TIMSK0=(0<<OCIE0B) | (0<<OCIE0A) |
(1<<TOIE0);


// Timer/Counter 1 Interrupt(s) initialization

TIMSK1=(0<<ICIE1) | (0<<OCIE1B) |
(0<<OCIE1A) | (0<<TOIE1);


// Timer/Counter 2 Interrupt(s) initialization

TIMSK2=(0<<OCIE2B) | (0<<OCIE2A) |
(0<<TOIE2);


// External Interrupt(s) initialization

// INT0: Off

// INT1: Off

// INT2: Off
```

```
// Interrupt on any change on pins PCINT0-7: Off
// Interrupt on any change on pins PCINT8-15: Off
// Interrupt on any change on pins PCINT16-23: Off
// Interrupt on any change on pins PCINT24-31: Off
EICRA=(0<<ISC21) | (0<<ISC20) | (0<<ISC11) | (0<<ISC10) | (0<<ISC01) | (0<<ISC00);
EIMSK=(0<<INT2) | (0<<INT1) | (0<<INT0);
PCICR=(0<<PCIE3) | (0<<PCIE2) | (0<<PCIE1) | (0<<PCIE0);

// USART0 initialization
// USART0 disabled
UCSR0B=(0<<RXCIE0) | (0<<TXCIE0) | (0<<UDRIE0) | (0<<RXEN0) | (0<<TXEN0) | (0<<UCSZ02) | (0<<RXB80) | (0<<TXB80);

// USART1 initialization
// USART1 disabled
UCSR1B=(0<<RXCIE1) | (0<<TXCIE1) | (0<<UDRIE1) | (0<<RXEN1) | (0<<TXEN1) | (0<<UCSZ12) | (0<<RXB81) | (0<<TXB81);

// Analog Comparator initialization
// Analog Comparator: Off
// The Analog Comparator's positive input is
// connected to the AIN0 pin
// The Analog Comparator's negative input is
// connected to the AIN1 pin
ACSR=(1<<ACD) | (0<<ACBG) | (0<<ACO) | (0<<ACI) | (0<<ACIE) | (0<<ACIC) | (0<<ACIS1) | (0<<ACIS0);
ADCSRB=(0<<ACME);
// Digital input buffer on AIN0: On
// Digital input buffer on AIN1: On
DIDR1=(0<<AIN0D) | (0<<AIN1D);

// ADC initialization
// ADC disabled
ADCSRA=(0<<ADEN) | (0<<ADSC) | (0<<ADATE) | (0<<ADIF) | (0<<ADIE) | (0<<ADPS2) | (0<<ADPS1) | (0<<ADPS0);

// SPI initialization
// SPI disabled
SPCR=(0<<SPIE) | (0<<SPE) | (0<<DORD) | (0<<MSTR) | (0<<CPOL) | (0<<CPHA) | (0<<SPR1) | (0<<SPR0);

// TWI initialization
// TWI disabled
TWCR=(0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWEN) | (0<<TWIE);

// Globally enable interrupts
#asm("sei")

while (1)
    {
    // Place your code here
      switch (stare_sistem)
          {
          case 0:
```

```
            stare_sistem = stare_sistem + 1;
            colet = ~PINA;
            cod_g_b = ~PINB;
        break;

        case 1:
            stare_sistem = stare_sistem + 1;
            stare_procesare = 0;// se
proceseaza
            PORTD =
OUT[stare_procesare];

            cod = colet;
            greutate_colet = cod_g_b &
masca_g;
        break;

        case 2:
             stare_sistem = stare_sistem +
1;
             ///clc
            if(cod <= 15 &&
greutate_colet<=greutate)
                {
                  zona[cod] ++ ;
                  stare_procesare = 2 ;
                }
             if(cod >15)
                {
                  stare_procesare = 1;
                  zona_de_eroare ++;
                }
            if(cod <= 15 &&
greutate_colet>greutate)
                {
                  stare_procesare = 1;
                  zona_de_inspectie++;
                }

        break;

        case 3:
             stare_sistem = stare_sistem +
1;  // intra urmatorul pachet
             PORTD =
OUT[stare_procesare];

        break;
        case 4:
            stare_sistem = 0;
            PORTD = 0xFF;
        break;

        default :
            buton = cod_g_b&masca_b;
            if(buton)
            {
              stare_sistem = 0;
            }
        break;
        }
    }
}
```