BANCOMAT ATMEGA164P

Arhitectura Microprocesoarelor 2

2023-2024

MIHAI ADRIAN-PETRU
422E

CUPRINS

Lista de figuri și tabele

Introducere

Scopul acestui proiect constă în programarea softwear a unui bacomat și înțelegerea procesului de creare a codului. Pentru realizarea si simularea codului voi volosi CodeVisionAVR, respectiv AVRStudio.

Pentru a programa bancomatul va trebui să ținem cont de cerințele date. Bancomatul va avea o bază de date ce conține 20 de clienți, dintre care 13 au rezidență națională și 7 o rezidență internațională. Dintre cei 13 cu rezidență națională aceștia sunt împărțiți între băncile: ING, BCR și BT, iar cei cu rezidența internațională sunt clienți ai bănci Revolut. Prin urmare va trebui să creăm un ID care să conțină toate aceste date; ID_rezidenta-ID_banca-ID_client.

După identificarea clientului, va urma autentificarea printr-un pin personalizat pentru fiecare client. Pentru transmiterea semnalelor necesare ca microcontrolerul să realizeze cerința dorită va trebui să realizăm o tastatură ce constă din 12 care vor îndeplini următoarele funcții: 10 butoane de la 0-9 ce ne vor ajuta la scrierea pinului si a ID-ului, iar celelalte 2 butoane ne ajută la realizarea anumitor activități specifice ale bancomatului.

Aceste activități specifice, care vor fi disponibile după autentificarea cu succes, sunt în număr de 4 și anume: consultarea sold-ului, retragerea banilor, emiterea chitanței și retragerea cardului. Fiecare acțiune va fi semnalată prin aprinderea unui led specific, astfel sunt necesare 4 led-uri.

Pentru implementarea softwear a bancomatului va trebui sa programam prin embedded C microcontroller-ul ATMega 164. Acesta dispune de 4 porturi: A, B, C, D și un semnal timer setat la o perioadă de 20ms. 2 dintre cele 4 porturi vor fi folosite pentru semnalele de intrare primite de la tastatură, iar un port din cele rămase va fi folosit drept ieșire pentru aprinderea unor din cele patru leduri. Un alt detaliu ce trebuie implementat în codul bancomatului este blocarea tastaturii pentru 3 secunde, caz în care trebuie să folosim semnalul ceas ca să măsurăm timpul cerut.

Din punct de vedere al codului va trebui mai întâi să identificăm ce tastă e apăsată prin verificarea cărui port e activ. De exemplu dacă vrem să vedem dacă tasta 1 e apăsată trebuie să vedem dacă PIND.1 este apăsat.

După definirea tastelor, o să folosim procesul secvențial pentru a realiza verificarea ID-ului, PIN-ului și realizarea activităților specifice. În total vom avea 6 stări: ID_rezidență, ID_bancă, ID_client, in_pin starea asociată introduceri PIN-ului, operații și starea de blocare ce este specificat să fie semnalată prin blocarea tastaturii și aprinderea led-urilor.

Pentru personalizarea PIN-ului o să definim câte un vector pentru fiecare client care va stoca fiecare cod. Toate acești vectori vor fi reținuți într-o matrice de 4 dimensiuni. Din această cauză ca să nu definim prea multe variabile fără rost, am decis ca indecși aleși sa fie ID respectivului client, prin urmare ID va fi scris în decimal și va începe de la 0. De exemplu, clientul 6 va avea indecși 0-0-3-d, unde d reprezintă counter-ul PIN-ului.

Toate aceste procese for fi semnalate prin aprinderea PORT-ului A, astfel o să avem nevoie de 4 led-uri care se vor aprinde/închide în funcție de situație.

Capitolul 1

Schema bloc a sistemului

Pentru realizarea schemei bloc o să luăm în cosiderare elementele principale pentru funcționarea proiectului.

Tastatura ne va ajuta să citim numerele transmise de client. Tastatura va fi compusă din 12 butoate: Tasta0, Tasta1...Tasta9, TastaE și TastaD. Un lucru important de care trebuie să ținem cont este faptul că noi am definit intrarea portului de tip pulldown, astfel va trebui să ținem cont de aceasta dacă vrem să construim butonul, unde va trebui să adăgăm rezistențe pulldown.

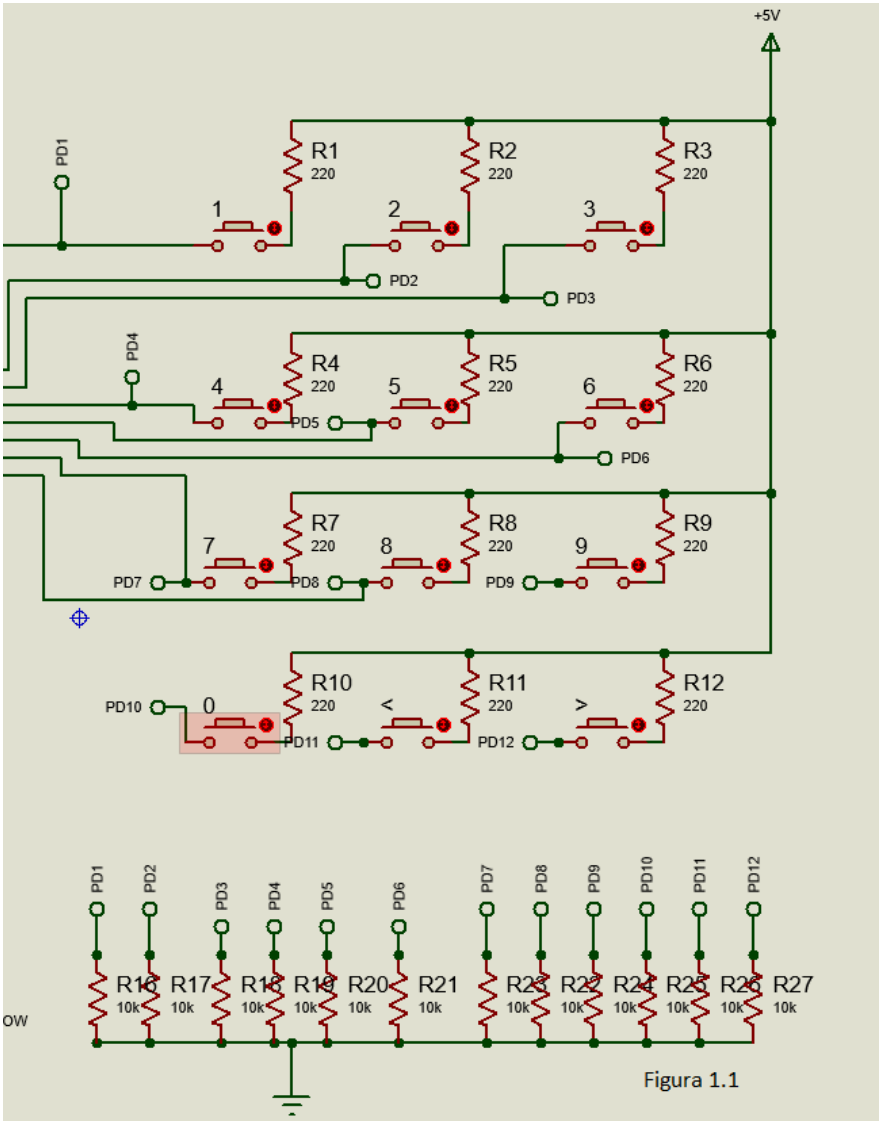

Figura 1.1

Microcontroller-ul are drept rol prelucrarea datelor primite la intrare și activarea portului corect la ieșire.

Ultimul bloc funcțional sunt led-urile care au ca rol semnalarea diferitor funcții ale bacomatului sau semnalarea funcției de blocare.

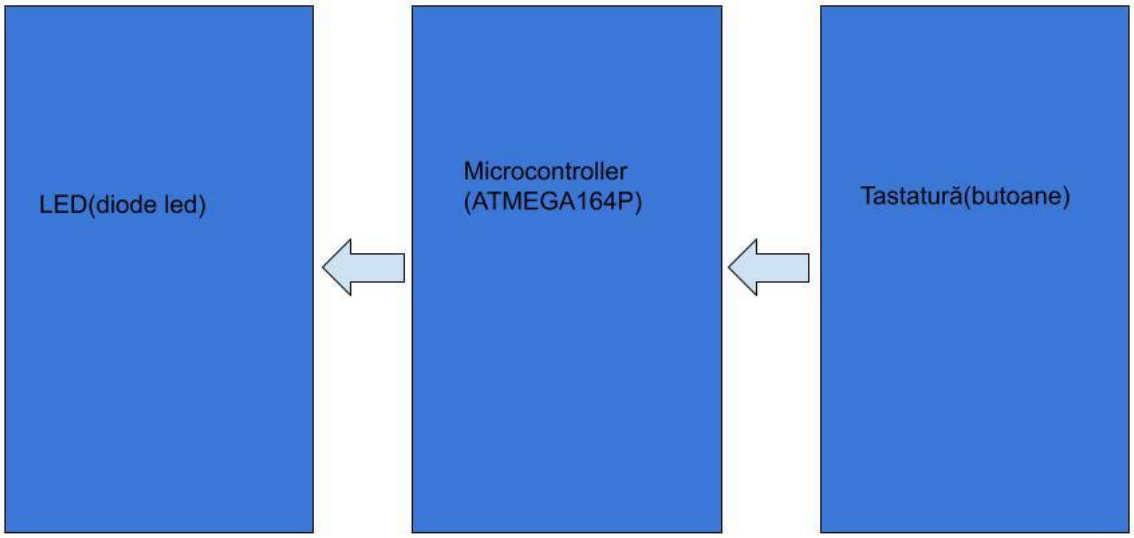

Figura 1.2

Diagrama cazurilor de utilizare a sistemului

Cazurile prin care poate trece un utilizator la folosirea bancomatului sunt: cele 3 stări de introducere a ID-ului, dacă este apăsată o tastă necorespunzătoare, ca de exemplu Tasta4 când sunt numai 3 clienți la o bancă, bancomatul o să intre într-o stare de blocare pentru 3 secunde urmând ca acesta să revină la starea 0, starea in_pin de adăugare a pinului ce constă în autentificarea cu ajutorul PIN-ului, orice greșeală a acestuia va rezulta în intrarea stării de blocare, dar după terminarea celor 3 secunde programul o să revină la starea precedentă și ultima funcție este cea de operații unde se poate apela funcția RESET ce ne duce înapoi la începutul programului.

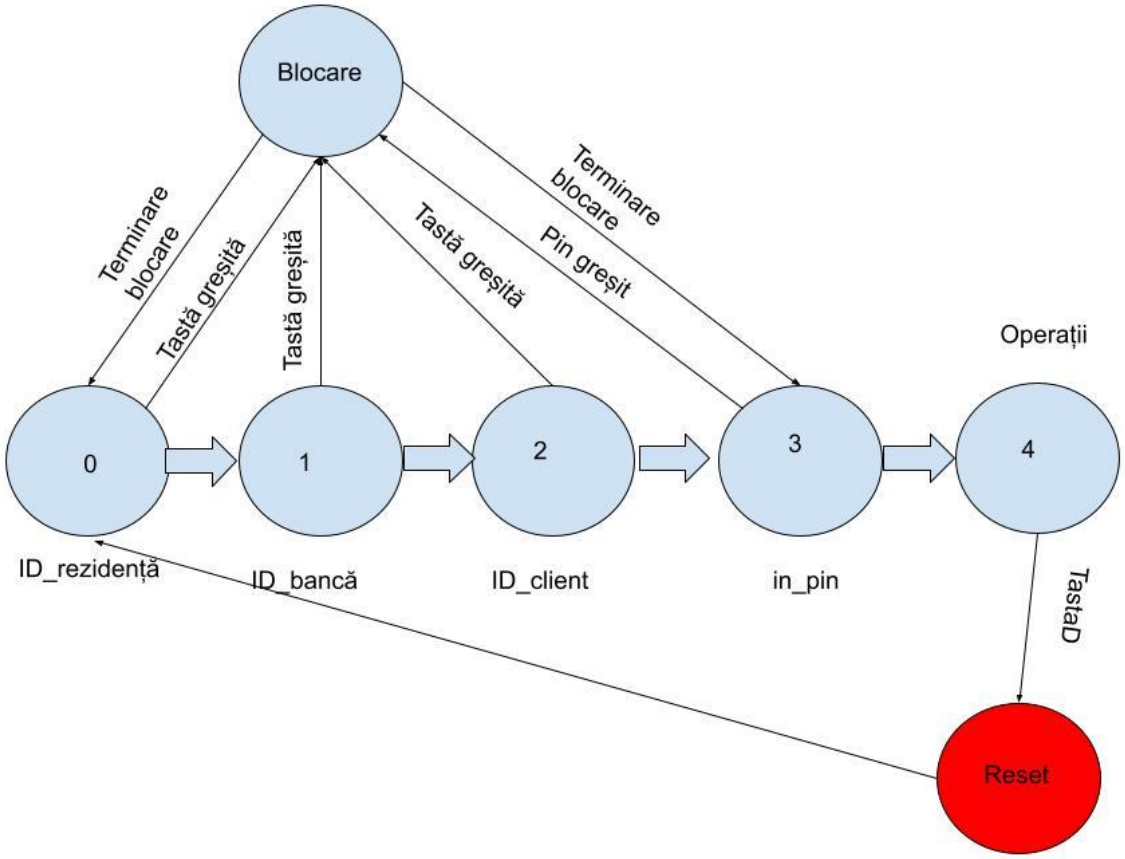

Figura 1.3

Capitolul 2

Descrierea funcționalității porturilor de intrare și ieșire

Pentru funcționarea bancomatului o să avem nevoie de 3 porturi. Astfel am ales PORT A ca ieșire și PORT C și PORT D ca intrare.

PORT D o să aibă drept rol preluarea semnalelor de la tastele 1 pană la 8, iar PORT C are rol preluarea semnalelor de la tastele 0, 9, E și D.

PORT D:

PIND.7	PIND.6	PIND.5	PIND.4	PIND.3	PIND.2	PIND.1	PIND.0
Tasta8	Tasta7	Tasta6	Tasta5	Tasta4	Tasta3	Tasta2	Tasta1

Tabel 2.1

PORT C:

PINC.7	PINC.6	PINC.5	PINC.4	PINC.3	PINC.2	PINC.1	PINC.0
///////////////	///////////////	///////////////	///////////////	TastaD	TastaE	Tasta0	Tasta9

Tabel 2.2

PORTA o să aibă drept rol aprinderea LED-urilor conectate la microcontroller.

PORT A:

PORTA.7	PORTA.6	PORTA.5	PORTA.4	PORTA.3	PORTA.2	PORTA.1	PORTA.0
///////////////	///////////////	///////////////	///////////////	LED4	LED3	LED2	LED1

Tabel 2.3

Proiectarea codului sursă corespunzător sistemului

Primul pas în proiectarea codului constă în selectarea unui microcontreller pe care îl putem programa în embedded C. O să alegem microcontroller-ul ATMega164P care ne oferă la dispoziție 4 PORT-uri programabile, un semnal ceas cu frecvența de 10 MHz și un timer pe care o să îl setăm la 20ms.

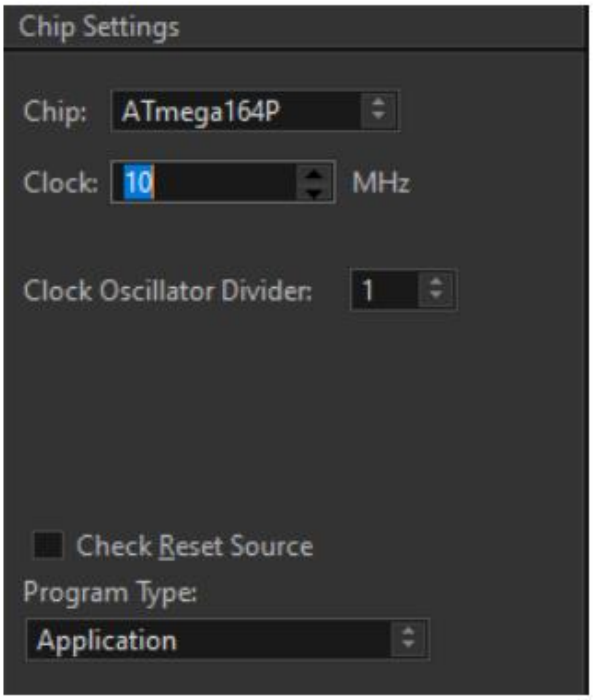

Figura 2.1

Codul trebuie să fie scris într-un mediu de dezvoltare. Prin urmare, o să folosim programul CodeVissionAVR. Pentru proiect o să ne ocupăm numai de partea softwear a programului, astfel tot ce ține de setup o să ne folosim de CodeWizardAVR.

Pentru setup-ul PORT-urilor ne vom folosi de Tabelul 2.1, 2.2, 2.3 pentru a vedea care PORT este de tip IN sau OUT. Trebuie să ținem cont că un PORT IN are un pullup value, în timp ce un PORT OUT este activ pe 1. Pentru celelalte porturi pe care nu le folosim le putem lăsa așa cum sunt.

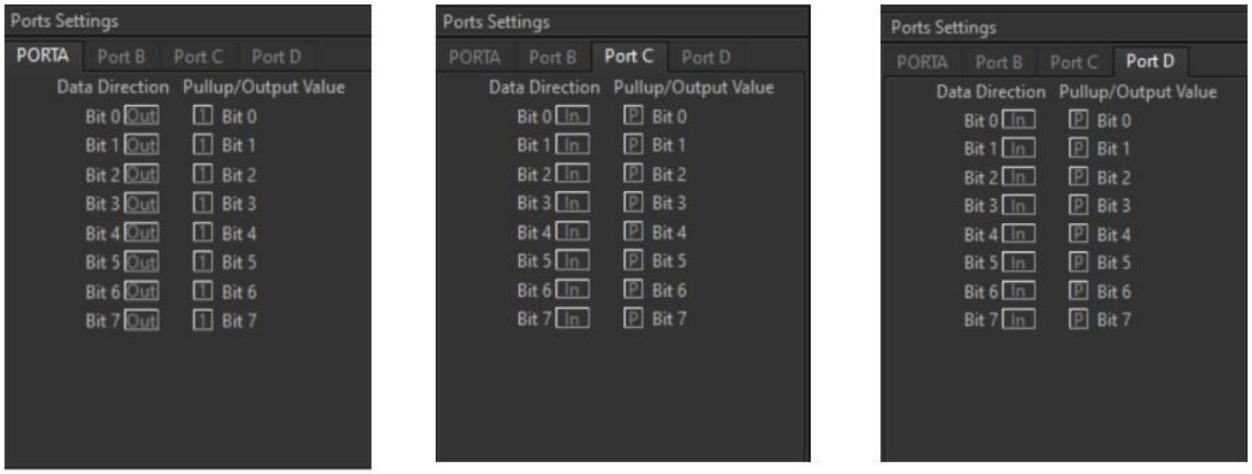

Figura 2.2

Setările pentru timer ca să avem o perioadă de 20ms sunt:

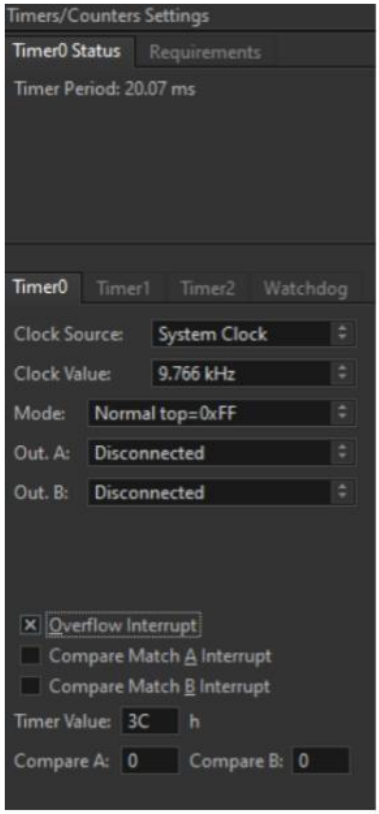

Figura 2.3

După terminarea setup-ului putem începe să programăm bancomatul. Codul principal poate fi scris în interrupt sau în bucla continuă while. Trebuie să ținem cont că un ciclu durează 20 ms așa cum a fost setat. În cazul de față o să scriem programul principal în interrupt.

Variabile globale simple pe care o să le folosim o să le declarăm în afara interrupt-ului. O să declarăm următoarele variabile cu următorul scop: stările id_rezidență, id_bancă, id_client, in_pin, operații și blocat pentru ușurarea citirii codului ca să nu stăm de fiecare dată să vedem ce număr aparține unei stări anume.

Tastele vor fi definite conform tabelului 2.1 și 2.2. Alte variabile care ne ajută sunt Starex care reprezintă starea, blocare reprezintă condiția de a intra în starea de blocare, cnt_blocare ne ajută la numărarea celor trei secunde ținând cont că avem un timer de 20 ms, Tasta_apasata ne ajută să vedem ce tastă este apăsată la momentul respectiv, select_regiune, select_banca, select_client ne ajută să identificăm pinul clientului și cnt_pin pentru a vedea la ce cifră din pin ne aflăm

Pentru personalizarea pinului am decis să declarăm o matrice de 4 dimensiuni client_pin[2][4][7][4], unde o să ne folosim de cele trei select-uri pentru a afla locația cifrei din pin pe care vrem să o verificăm dacă este corectă. Am declarat pentru fiecare client câte un vector de 4 cifre cu scopul de a pune toate valorile în matricea mare ca să putem compara tasta apăsată cu cifra corectă din vectorul client_pin. Acestea au fost declarate în main ca setup-ul deoarece acestea sunt rulate o singură dată la începutul programului.

ID si pinul corespunzător fiecărui client arată în felul următor:

Nr. Crt.	ID_reziden	ID_banca	ID_client	PIN
1	0	0	0	1234
2	0	0	1	2003
3	0	0	2	2024
4	0	0	3	1594
5	0	1	0	6478
6	0	1	1	5469
7	0	1	2	1111
8	0	1	3	2222
9	0	1	4	7563
10	0	1	5	9874
11	0	2	0	5135
12	0	2	1	8753
13	0	2	2	3256
14	1	3	0	7365
15	1	3	1	8748
16	1	3	2	2456
17	1	3	3	9853
18	1	3	4	1467
19	1	3	5	5632
20	1	3	6	6258

Tabel 2.4

Pentru rularea programului o să ne folosim în interrupt de procesele secvențiale. O să schimbăm starea cu ajutorul unui switch-case în care o să punem un if-else în care verificăm dacă o tastă corectă a fost apăsată. Dacă tasta apăsată e greșită bancomatul va intra în blocare, de ținut cont că dacă intrăm în blocare la scrierea ID o să o luăm de la început, altfel dacă greșim la pin rămânem în aceeași stare.

Implementarea codului sursă

Dacă punem cap la cap tot ce am spus până acum programul realizat arată în felul următor:

```
#include <mega164.h>


#define id_rezidenta  0
#define id_banca      1
#define id_client     2
#define in_pin        3
#define operatii      4


#define Tasta1 PIND.0
#define Tasta2 PIND.1
#define Tasta3 PIND.2
#define Tasta4 PIND.3
#define Tasta5 PIND.4
#define Tasta6 PIND.5
#define Tasta7 PIND.6
#define Tasta8 PIND.7
#define Tasta9 PINC.0
#define Tasta0 PINC.1
#define TastaE PINC.2
#define TastaD PINC.3


int tmp_portC, tmp_portD;
int LED1=0, LED2=0,  LED3=0, LED4=0;


int S=0;
int blocare=0;
int asteapta=0;
int cnt_blocare=0;
int Tasta_apasata=99;


int select_regiune;
int select_banca;
int select_client;
```

```
int cnt_pin=0;
int client_pin[2][4][7][4];
/* baza de date cu clienti ca o matrice multidimensionala cu 4 dimensiuni */
 // regiune // bnca // clienti // pin pentru fiecare client //

 // national - 0 // international - 1 //
 // ING - 0   // BCR - 1   // BT - 2    // REVOLUT - 3 //
 // 4 clienti // 6 clienti // 3 clienti // 7 clienti  //


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

// aici asteptam deblocarea tastaturii timp de 20ms * 150 = 3 secunde
if(blocare)
 {
    cnt_blocare++;         // incrementam contor
    if(cnt_blocare >= 150)  // aici expira cele 3 secunde
    {
        cnt_blocare=0;     // reinitializam contor
        PORTA=0x00;        // stingem leduri
        blocare = 0;       // iesim din blocare
    }
   return;
  }

 if(asteapta)
 {
    cnt_blocare++;         // incrementam contor
    if(cnt_blocare >= 25)  // aici expira cele 0.5 sec
    {
        cnt_blocare=0;     // reinitializam contor
        asteapta = 0;       // iesim din asteptare
```

```
    }
   return;
  }

Tasta_apasata=99;
//presupunem cod pentru tasta neapasata si de la 0 la 9 pentru taste
// mai avem o tasta sters sau revenire cod 10 si una enter cod 11

        if(Tasta0)
          Tasta_apasata=0;
        else if (Tasta1)
          Tasta_apasata=1;
        else if (Tasta2)
          Tasta_apasata=2;
        else if (Tasta3)
          Tasta_apasata=3;
        else if (Tasta4)
          Tasta_apasata=4;
        else if (Tasta5)
          Tasta_apasata=5;
        else if (Tasta6)
          Tasta_apasata=6;
        else if (Tasta7)
          Tasta_apasata=7;
        else if (Tasta8)
          Tasta_apasata=8;
        else if (Tasta9)
          Tasta_apasata=9;
        else if (TastaE)
          Tasta_apasata=10;
        else if (TastaD)
          Tasta_apasata=11;

    // Stare sunt starile sistemului/ programului
    switch(S){
     case id_rezidenta:
       if (Tasta_apasata == 99)
        {
          PORTA=0x00;   //  debug stare
        }
```

```
      else if((Tasta_apasata == 0 ) || (Tasta_apasata == 1))   // citesc rezidenta trebuie sa fie 0
sau 1
        {
           //PORTA=0x00;        // stingem leduri daca venim de pe reset
           select_regiune=Tasta_apasata; // memoram regiunea selectata
           S=id_banca;               // trecem in urmatoarea stare pentru citit id_banca
           asteapta=1;
        }
        else
        {
             // alta tasta nu este valida aprindem ledurile pentru 3 secunde
             // nu mai citim tastatura 3 secunde
             PORTA=0xff;
             blocare=1;
             //nu trebuie sa resetam starea deoarece suntem deja in prima stare
        }
     break;

     case id_banca:      // stare in care citim id_banca si semnalizam pe led 1 Blink

       if (Tasta_apasata == 99)
        {
          PORTA=0x01; //debug
        }
       else if( ((select_regiune == 0) && (Tasta_apasata < 3 )) || ((select_regiune == 1) &&
(Tasta_apasata == 3 )))
        // regiunea locala are 3 banci cu id de la 0 la 2
        // regiunea international are o banca revolut cu id 3 daca am selectat regiunea
international doar banca cu id 3 este valida
        {
           S=id_client;                // trecem in urmatoarea stare pentru citit id_client
           select_banca=Tasta_apasata; // memoram banca selectata
           asteapta=1;
        }
        else
        {
             // alta tasta nu este valida aprindem ledurile pentru 3 secunde
             // nu mai citim tastatura 3 secunde
             PORTA=0xff;
             blocare=1;
             S=id_rezidenta; //resetam starea la id_rezidenta
        }
```

```
     break;


     case id_client:  // stare citire id_client
       if (Tasta_apasata == 99)
        {
          PORTA=0x02; // debug
        }
        else if( ((select_banca == 0) && (Tasta_apasata < 4 )) || ((select_banca == 1) && (Tasta_apasata < 6 )) ||
            ((select_banca == 2) && (Tasta_apasata < 3 )) || ((select_banca == 3) && (Tasta_apasata < 7 )) )
        // pentru unele banci avem mai putini clienti pentru Revolut 7 clienti de la 0 la 6
        {
           select_client=Tasta_apasata; // memoram id_client
           cnt_pin=0;    // contor pozitie cifra pin [0,1,2,3]
           S=in_pin;     // trecem in starea citire Pin
           asteapta=1;
           PORTA=0x08;   // debug
         }
        else
        {
             // alta tasta nu este valida aprindem ledurile pentru 3 secunde
             // nu mai citim tastatura 3 secunde
             PORTA=0xff;
             blocare=1;
             S=id_rezidenta; //resetam starea la id_rezidenta
        }
     break;


     case in_pin:   // stare citim pin si verificam din baza de date in cazul nostru o matrice multidimensionala


       if (Tasta_apasata == 99)
        {
        }
        else if(Tasta_apasata == 10)
        {
         asteapta=1;
         S=id_rezidenta;  //resetam starea la id_rezidenta
        }
       else             if(             (Tasta_apasata             <             10)             &&
(client_pin[select_regiune][select_banca][select_client][cnt_pin] == Tasta_apasata) )
```

```
       // doar taste de la 0 la 9 sunt valide si verificam daca tasta apasata este corecta cu cifra
salvata in DB matricea pentru PIN
       {
        asteapta=1;
        //PORTA=cnt_pin;  // debug cifra pin
        cnt_pin++;        // incrementam contor pozitie cifra pin pin
        LED3=~LED3 & 0x04;   // Semnalam aceasta stare pe LED3 BLINK
        PORTA=(0x00 | LED3);
        if (cnt_pin == 4) // pin-ul este corect trecem in starea pentru operatii
            {
             S=operatii;
             LED1=0; LED2=0x0; LED3=0;  LED4=0;
             }
       }
       else
       {  // daca o tasta nu corespunde cu pin din DB
          // nu mai citim tastatura 3 secunde
          PORTA=0xff;
          blocare=1;
          cnt_pin=0;              // reluam verificarea PIN-ului prin reinitializare contor pozitie
pe 0
          S=in_pin;  //resetam starea la id_rezidenta
       }
     break;


     case operatii:  // starea pentru efectuare operatii
        if (Tasta_apasata == 99)
        {
         }
         else if(Tasta_apasata == 9)
         {
            PORTA=0x01;//LED1=~LED1 & 0x01;    //comanda pentru consultare balanta consta in apsarea
Tasta9
            asteapta=1;
         }
         else if(Tasta_apasata == 0)
         {
            PORTA=0x02; //LED2=~LED2 & 0x02;   //comanda pentru consultare balanta consta in apsarea
Tasta0
            asteapta=1;
        }
         else if(Tasta_apasata == 10)
```

```
         {
            PORTA=0x04; //LED3=~LED3 & 0x04;   //comanda pentru emitere chitanta consta in apsarea
TastaE
            asteapta=1;
         }
         else if(Tasta_apasata == 11)
         {
            PORTA=0x08; //LED4=~LED4 & 0x08;     //comanda pentru retragere card (reset) consta in
apsarea TastaD
            S=id_rezidenta;
            asteapta=1;
         }
         else
         {
          PORTA=0xff;
          blocare=1;
          S=id_rezidenta;
          //in acest caz nu trebuie sa revenuim la starea id_rezidenta daca am apasat un buton
gresit
          return;
          }
        //PORTA=(0x00 | LED1 | LED2 | LED3 | LED4 );
     break;

    default:
     break;
   }
}



// Declare your global variables here

void main(void)
{

int a,b,c,d;

 int client1[4]={1,2,3,4};
 int client2[4]={2,0.0,3};
 int client3[4]={2,0,2,4};
 int client4[4]={1,5,9,4};
 int client5[4]={6,4,7,8};
```

```
 int client6[4]={5,4,6,9};
 int client7[4]={1,1,1,1};
 int client8[4]={2,2,2,2};
 int client9[4]={7,5,6,3};
 int client10[4]={9,8,7,4};
 int client11[4]={5,1,3,5};
 int client12[4]={8,7,5,3};
 int client13[4]={3,2,5,6};
 int client14[4]={7,3,6,5};
 int client15[4]={8,7,4,8};
 int client16[4]={2,4,5,6};
 int client17[4]={9,8,5,3};
 int client18[4]={1,4,6,7};
 int client19[4]={5,6,3,2};
 int client20[4]={6,2,5,8};

//pentru fiecare client atribuim un vector cu cele 4 cifre ale pinului
// 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=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRA=(1<<DDA7) | (1<<DDA6) | (1<<DDA5) | (1<<DDA4) | (1<<DDA3) | (1<<DDA2) | (1<<DDA1) | (1<<DDA0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTA=(0<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) | (0<<PORTA2) |
(0<<PORTA1) | (0<<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=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) |
(0<<PORTB1) | (0<<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=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) |
(0<<PORTD1) | (0<<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")

//regiune[0/1] banca[0/1/2/3] clinet[0/1/2/3/4/5/6] pin[1234]
```

```
for(a=0;a<2;a++)      //contor pentru regiuni 0 national si 1 international selectabil si cu tasta 0
sau 1

 for(b=0;b<4;b++)     //contor pentru banci id de la 0 la 3 selectabil cu tasta de la 0 la 3

  for(c=0;c<7;c++)    //contor pentru clineti de la 0 la 7

   for(d=0;d<4;d++)   //contor pentru pin initial toti clientii ii initializam


        if((a == 0) && (b == 0) && (c == 0) && (d < 4))

        {

            client_pin[a][b][c][d]=client1[d];  // scriem pin cifra cu cifra deoarece in C nu se pot
initializa linii sau coloane intregi

        }

        if((a == 0) && (b == 0) && (c == 1) && (d < 4))

        {

            client_pin[a][b][c][d]=client2[d];

        }

        if((a == 0) && (b == 0) && (c == 2) && (d < 4))

        {

            client_pin[a][b][c][d]=client3[d];

        }

        if((a == 0) && (b == 0) && (c == 3) && (d < 4))

        {

            client_pin[a][b][c][d]=client4[d];

        }

        if((a == 0) && (b == 1) && (c == 0) && (d < 4))

        {

            client_pin[a][b][c][d]=client5[d];

        }

        if((a == 0) && (b == 1) && (c == 1) && (d < 4))

        {

            client_pin[a][b][c][d]=client6[d];

        }

        if((a == 0) && (b == 1) && (c == 2) && (d < 4))

        {

            client_pin[a][b][c][d]=client7[d];

        }

        if((a == 0) && (b == 1) && (c == 3) && (d < 4))

        {

            client_pin[a][b][c][d]=client8[d];

        }

        if((a == 0) && (b == 1) && (c == 4) && (d < 4))

        {

            client_pin[a][b][c][d]=client9[d];
```

```
}
if((a == 0) && (b == 1) && (c == 5) && (d < 4))
{
    client_pin[a][b][c][d]=client10[d];
}
if((a == 0) && (b == 2) && (c == 0) && (d < 4))
{
    client_pin[a][b][c][d]=client11[d];
}
if((a == 0) && (b == 2) && (c == 1) && (d < 4))
{
    client_pin[a][b][c][d]=client12[d];
}
if((a == 0) && (b == 2) && (c == 2) && (d < 4))
{
    client_pin[a][b][c][d]=client13[d];
}
if((a == 1) && (b == 3) && (c == 0) && (d < 4))
{
    client_pin[a][b][c][d]=client14[d];
}
if((a == 1) && (b == 3) && (c == 1) && (d < 4))
{
    client_pin[a][b][c][d]=client15[d];
}
if((a == 1) && (b == 3) && (c == 2) && (d < 4))
{
    client_pin[a][b][c][d]=client16[d];
}
if((a == 1) && (b == 3) && (c == 3) && (d < 4))
{
    client_pin[a][b][c][d]=client17[d];
}
if((a == 1) && (b == 3) && (c == 4) && (d < 4))
{
    client_pin[a][b][c][d]=client18[d];
}
if((a == 1) && (b == 3) && (c == 5) && (d < 4))
{
    client_pin[a][b][c][d]=client19[d];
}
```

```
        if((a == 1) && (b == 3) && (c == 6) && (d < 4))
        {
            client_pin[a][b][c][d]=client20[d];
        }
```

```
while (1)
      {
      // Place your code here
```

```
                                                    }
}
```

Capitolul 3

Diagrama structurală corespunzătoare codului sursă

Diagrama structurală constă în reprezentarea funcțiilor bancomatului cum se apelează între ele.

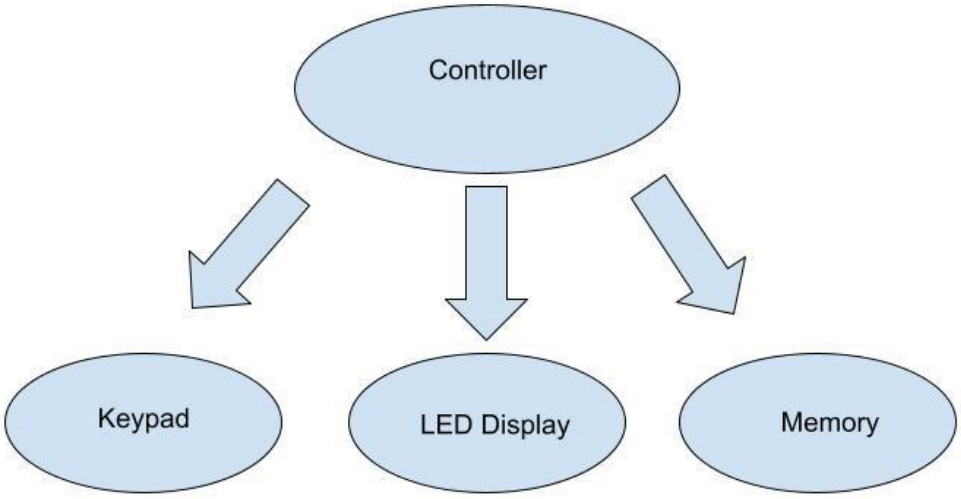

Figura 3.1

Diagrama secvențială corespunzătoare codului sursă

Diagrama secvențială constă în ordinea în care se apelează funcțiile.

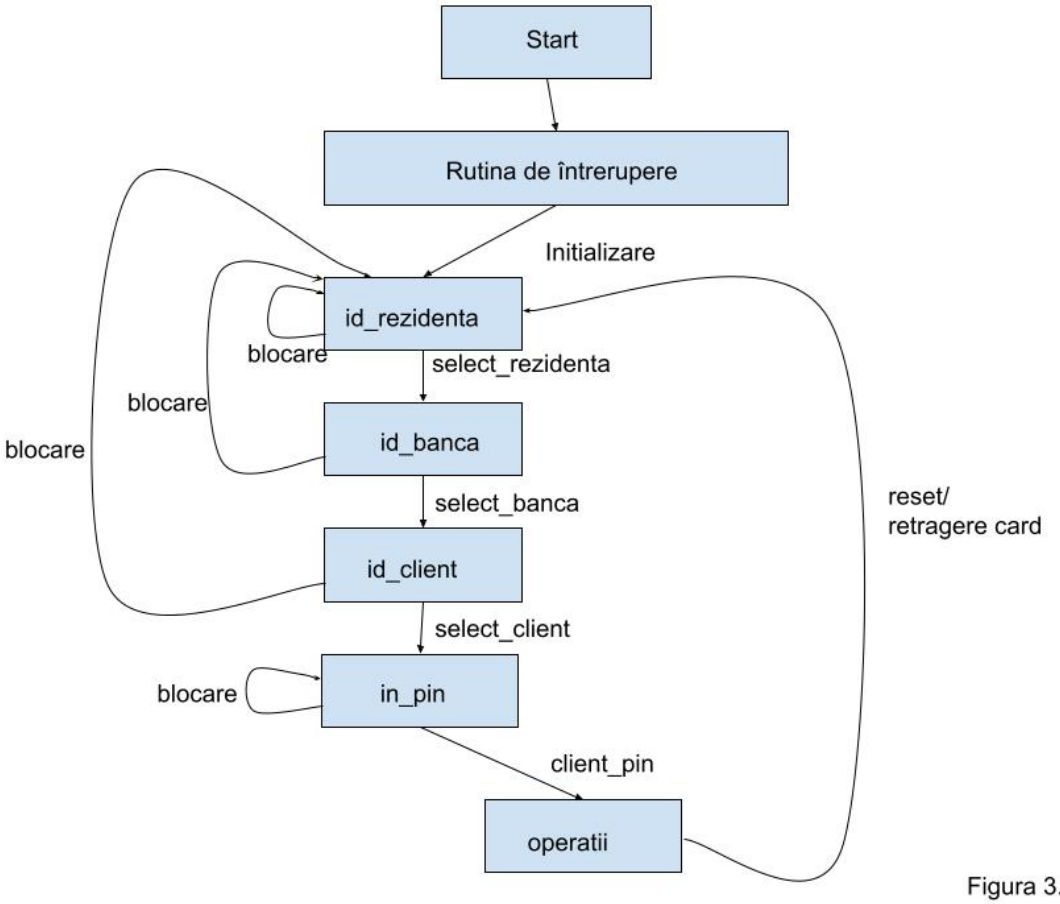

Figura 3.2

Capitolul 4

Testarea funcționalității sistemului

Pentru simularea sistemului o să rulăm fișierul în AVRStudio și o să punem valori pe PINC și PIND conform Tabelului 2.4, ca să putem observa cum se schimbă biții pe PORTA unde este definită ieșirea. O să mai urmărim ce valori iau variabile definite: blocare, Tasta_apasata, S, cnt_blocare și cnt_pin. Cazul pe care o să îl tratăm este primul, unde ID este 0|0|0 și pinul este 1234. Am ales acest caz pentru simplitate. Scopul este să trecem prin toate stările și să ne asigurăm că merge blocarea.

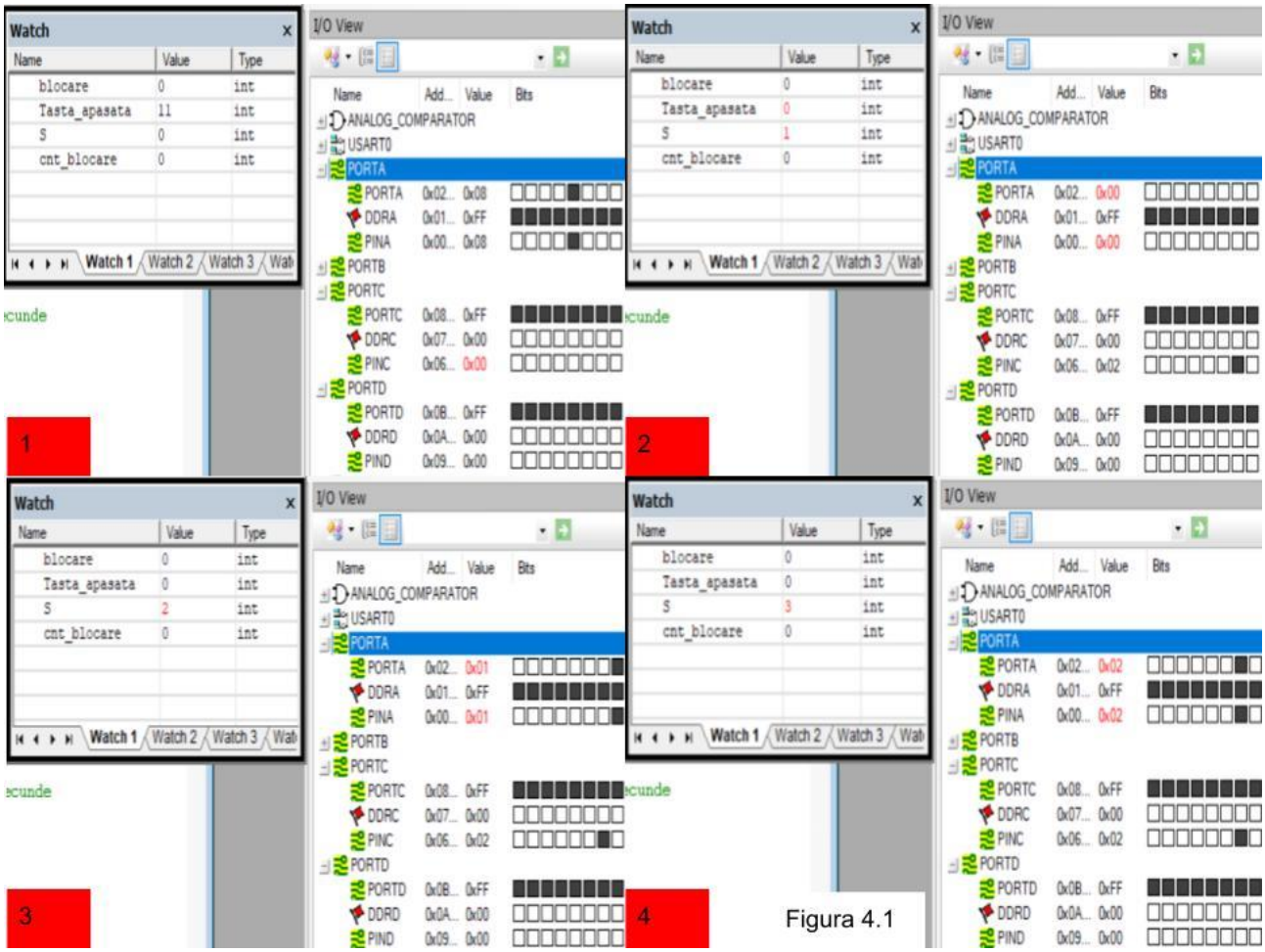

Figura 4.1

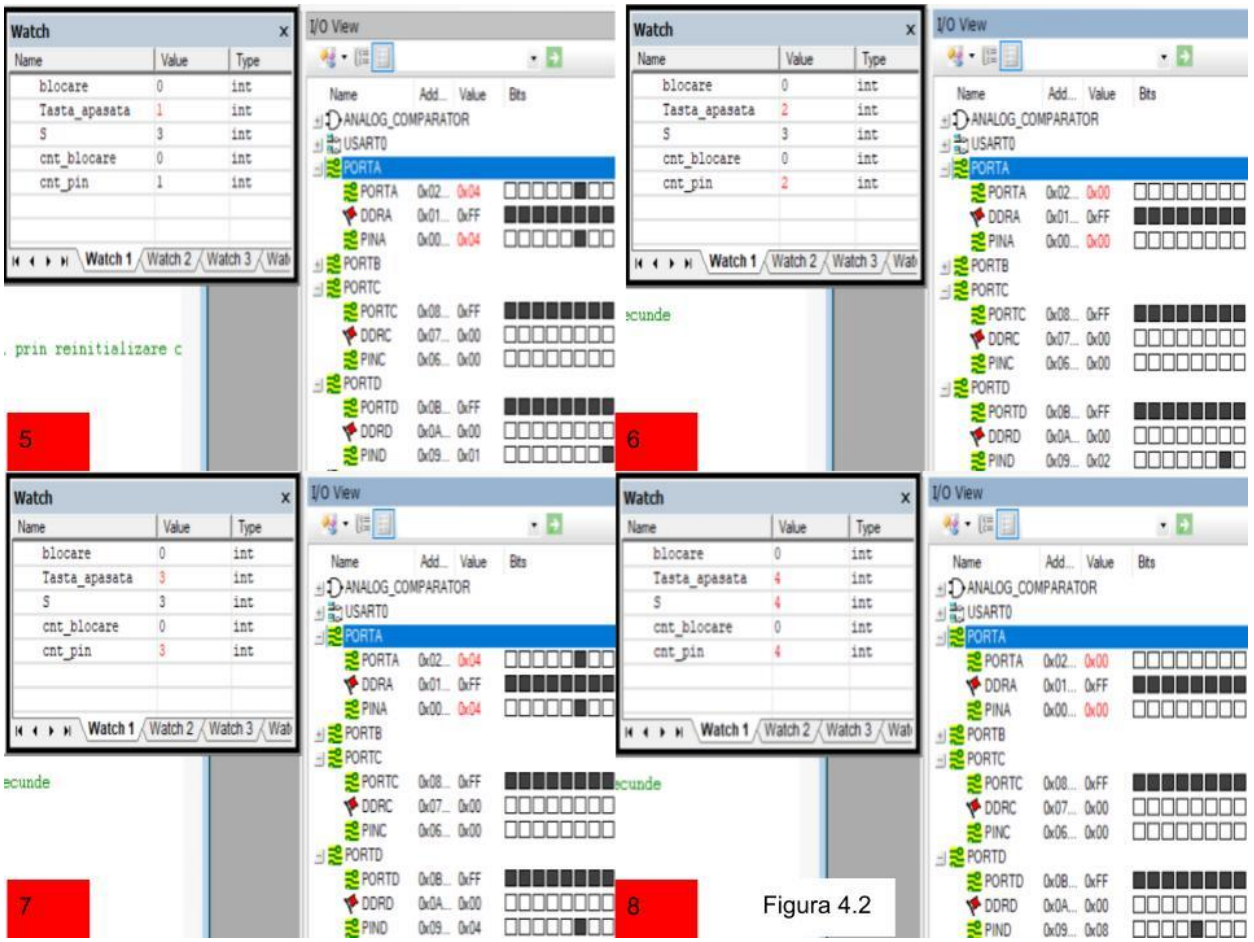

Figura 4.2

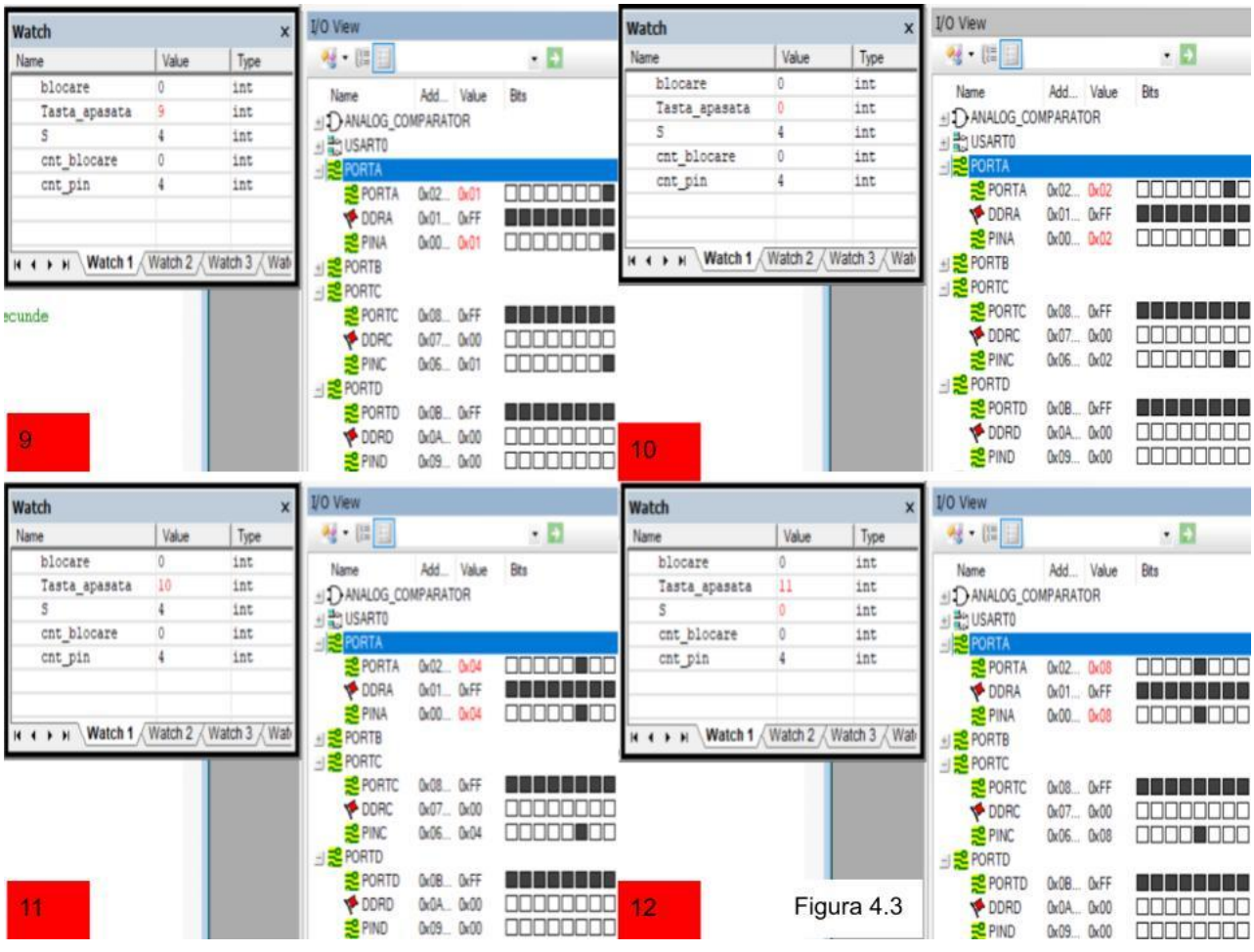

Figura 4.3

Ultimul caz care a mai rămas de verificat este blocarea.

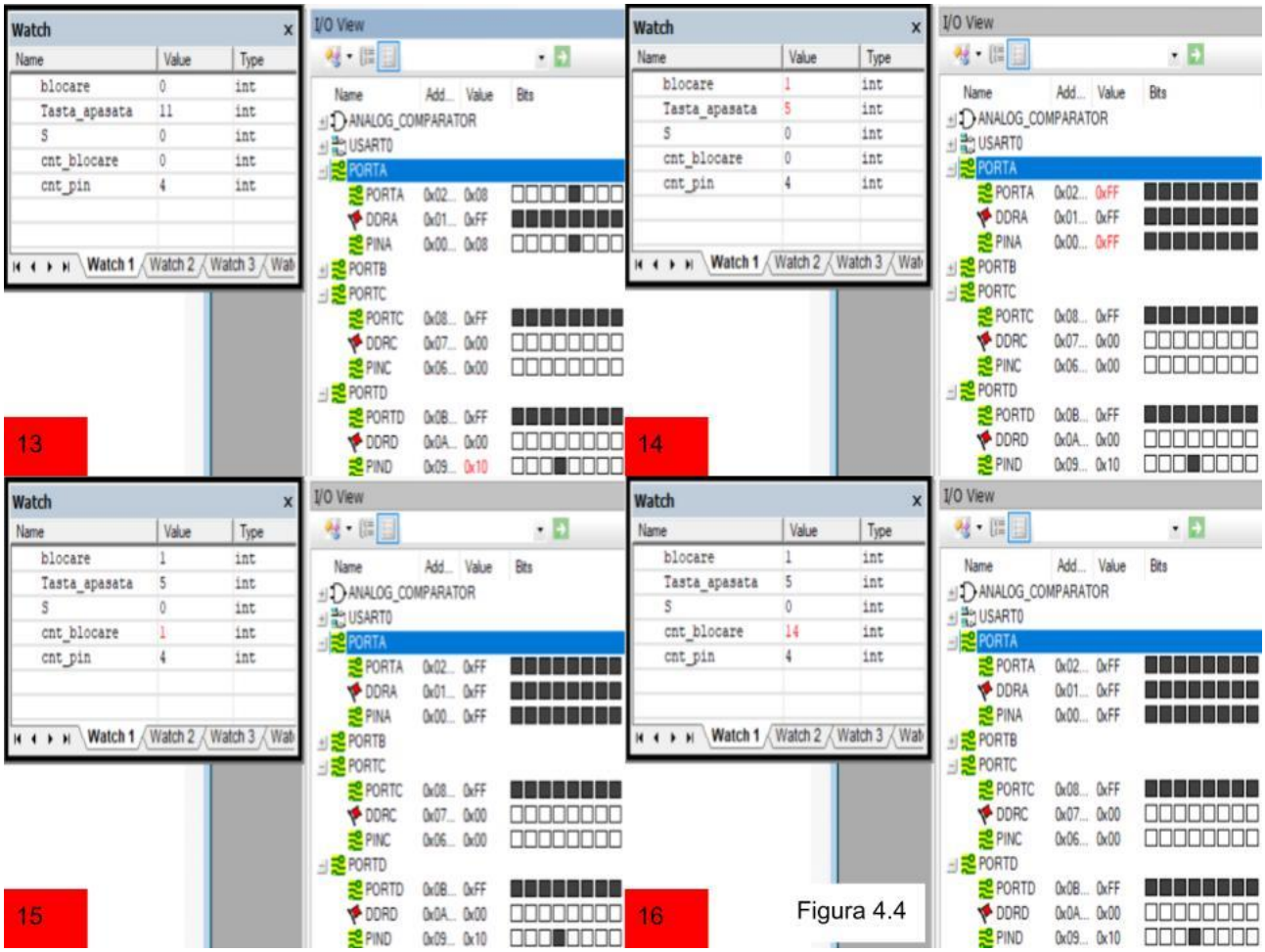

Figura 4.4

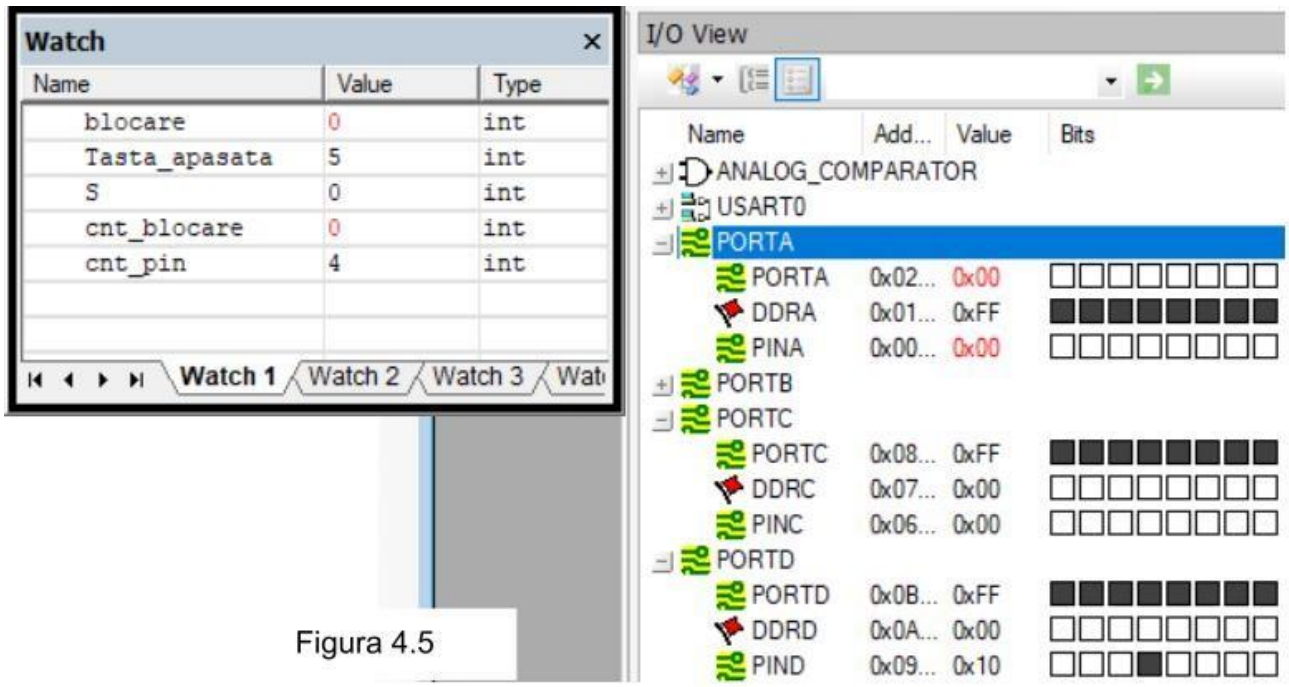

Figura 4.5

Schema și implementarea hardware

Pentru o implementare mai amplă a proiectului o să facem schema hardware în Proteus, folosind microcontroller-ul ATMega164P urmând ca să folosim fișierul .hex al codului principal ca să îl programăm. De ținut cont, pentru ca programul să meargă trebuie să folosim după fiecare tastă apăsată o buclă de așteptare pentru a ne asigura ca atunci când o apăsăm programul să nu primească aceeași comandă de 2 ori.

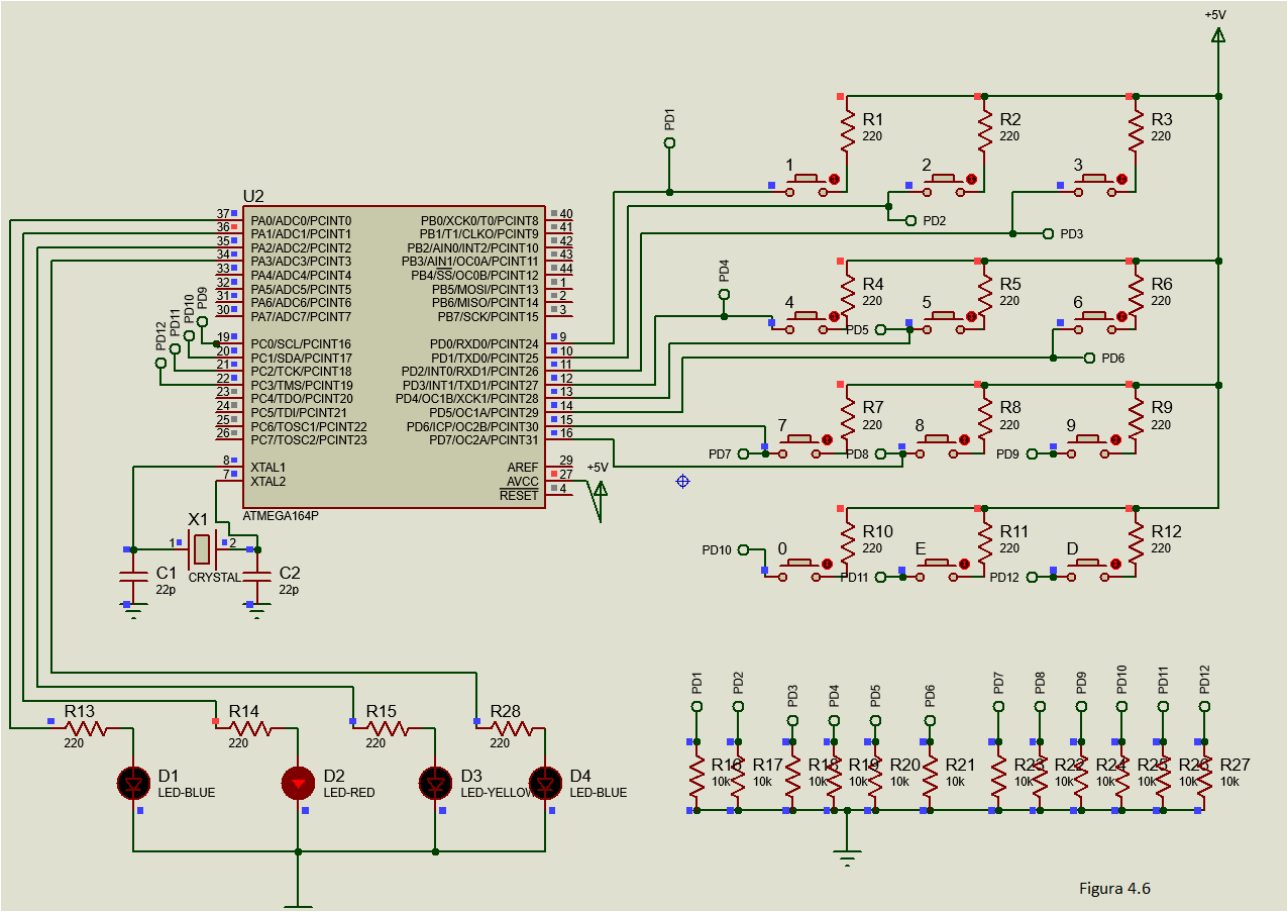

Figura 4.6