Sistem de monitorizare si avertizare pentru cutremure

Ghinea Valentin-Andrei
Electronica, Telecomunicatii si Tehnologia Informatiei
Anul II, Grupa 424E
mai 2025

Cuprins

Introducere **4**

1 Principiul de functionare **4**

1.1 Stabilirea porturilor si semnalelor 5

1.2 Circuitul logic combinational 6

1.3 Circuit logic secvential 7

1.4 Procesul secvential / FSM 8

2 Implementare Software **9**

2.1 Implementare CLC 9

2.2 Implementare CLS 9

2.3 Functii utilizate in FSM 10

3 Implementare Hardware **11**

Concluzii **13**

Anexa **14**

Listă de figuri

1.1 Schema bloc a sistemului 4

1.2 Graf asociat CLS 7

1.3 Graf asociat FSM 8

3.1 Simulare - cutremur LOW 11

3.2 Simulare - cutremur MEDIU 12

3.3 Simulare - alarma 12

Listă de tabele

1.1 PORTD - INPUT 5

1.2 PORTB - OUTPUT 5

1.3 CLC 6

Introducere

In prezenta lucrare mi-am propus sa proiectez din punct de vedere SW si HW principiile de functioare ale unui sistem de avertizare pentru cutremure.

Am utilizat atat principiul de CLC cat si cel de CLS pentru a realiza intr-un final un proces secvential. Sistemul astfel realizat, va trebui sa raspunda la semnalele primite de la un modul cu senzor pentru cutremure si sa atentioneze vizual si sonor utilizatorul.

Sistemul este responsabil cu monitorizarea permanenta a datelor primite de la modulul cu senzor pentru cutremure. In momentul in care sunt detectate cutremure periculoase, sistemul declanseaza alarma de atentionare care ramane activa pana cand pericolul dispare.

Am realizat proiectarea software in limbaj C specific microcontrolerului ATmega164p, 10 Mhz, cu intreruperi periodice la fiecare 20 ms. Pentru proiectarea hardware am utilizat SimulIDE, program special conceput pentru simularea aplicatiilor cu microcontrolere.

1 Principiul de functionare

Pentru realizarea sistemului este nevoie de stabilirea intrarilor si iesirilor din microcontroler. Voi folosi apoi, semnalele de intrare pentru a proiecta un circuit logic combinational care va interpreta datele de la modulul cu senzor.

Voi folosi si un circuit logic secvential care va stabili daca un semnal generat de prezenta unui cutremur este suficient de indelungat pentru a decalnsa alarma de avertizare. In functie de nivelul cutremurului receptat de senzorul extern, sistemul va stabili ce tip de mesaj si avertizare sa actioneze. Toate acestea sunt detaliate in cele ce urmeaza.

Totodata, utilizatorul va avea acces la un "reset" manual, pornirea asincrona a sistemului si posibilitatea de a testa individual sistemul de alarma.

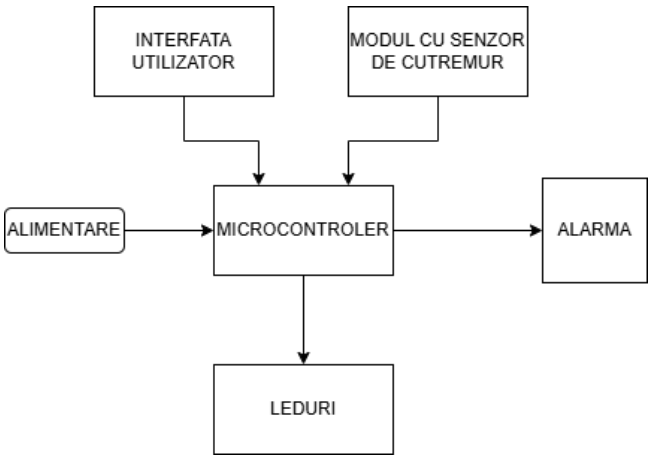

Figura 1.1: Schema bloc a sistemului

1.1 Stabilirea porturilor si semnalelor

Folosesc un microcontroler ATMega164p. Acesta are 4 porturi A, B, C, D, fiecare cu 8 biti (pini IN/OUT). Aleg pentru intrare (IN) portul D si pentru iesire (OUT) portul B.

D7	D6	D5	D4	D3	D2	D1	D0
START	TST1	TST2	RST	HIGH	MED	LOW	-

Tabela 1.1: PORTD - INPUT

START - pornirea sistemului de monitorizare

TST1 - testare alarma cutremur (testeaza alarma sonora si vizuala)

TST2 - testare indicatori de nivel

RST - reset manual

HIGH - receptie cutremur de intensitate ridicata

MED - receptie cutremur de intensitate medie

LOW - receptie cutremur de intensitate mica

START, TST, RST - active in "0" si sunt actionate de butoane

HIGH, MED, LOW - active in "1" si sunt generate de modulul extern al senzorului

B7	B6	B5	B4	B3	B2	B1	B0
MON	LVL2	LVL1	LVL0	CMD	AVR	BUZZ	INT

Tabela 1.2: PORTB - OUTPUT

MON - indicator sistem activ

LVL2-LVL0 - led rosu (HIGH), led galben (MEDIU), led verde (LOW)

CMD - comanda de afisare a mesajului de avertizare

AVR - led mov de avertizare - prezenta cutremur MEDIU

BUZZ - comanda BUZZER - avertizare sonora

INT - comanda led rosu intermitent - alarma

MON, LVL, CMD, AVR, BUZZ si INT sunt active in "0".

1.2 Circuitul logic combinational

Rolul acestui circuit este de a stabili in functie de valorile de pe HIGH, MED si LOW ce leduri sunt activate si daca va aprinde afisajul textului de avertizare.

Pentru un semnal LOW se va aprinde doar ledul verde - indicator pentru nivelul scazut al cutremurului.

Pentru un semnal MED se vor aprinde ledul galben - indicator pentru nivelul mediu al cutremurului, ledul de avertizare mov si afisajul textului " Cutremur".

Pentru un semnal HIGH se vor aprinde toate ledurile de avertizare inclusiv cel rosu - indicatorul nivelului ridicat al cutremurului (momentan nu se aprinde alarma, aceasta este activata separat de CLS).

HIGH	MED	LOW	B6	B5	B4	B3	B2	semnificatie
0	0	0	1	1	1	1	1	absenta cutremur
0	0	1	1	1	0	1	1	cutremur LOW
0	1	0	1	0	1	0	0	cutremur MED
0	1	1	1	1	1	1	1	X
1	0	0	0	1	1	0	0	cutremur HIGH
1	0	1	1	1	1	1	1	X
1	1	0	1	1	1	1	1	X
1	1	1	1	1	1	1	1	X

Tabela 1.3: CLC

Pentru a extrage valorile de intrare folosesc masca : (in & 0x0E) » 1. Unde "in" = PIND.

Tinand cont de faptul ca B7 (Output 7) este mereu "0" (sistem activ) si B1 = B0 = 1 (alarma nu se activeaza din CLC) tabela pentru iesirile CLC este urmatoarea:

char TAB[8] = {0x7F, 0x6F, 0x53, 0x7F, 0x33, 0x7F, 0x7F, 0x7F};

1.3 Circuit logic secvential

Scopul circuitului logic secvential este de a stabili daca activeaza alarma si avertizarile vizuale.

Daca este receptat un semnal de tip HIGH pe pinul de intrare D3 mai mult de 3 secunde, acesta va declansa alarma (semnale vizuale + buzzer).

Pentru a determina cu exactitate cat timp este prezent un semnal HIGH continuu voi folosi intreruperile periodice (o data la fiecare 20 ms). Folosesc un parametru care este incrementat o data la fiecare 20 ms x 150 si astfel, cand acesta ajunge la valoarea 3 inseamna ca au trecut 3 secunde de la declansarea numaratorii.

Prin urmare:

In starea Q = 0:

- daca D3 este 1 se trece in starea Q = 1 (if((in & 0x08) != 0) Q = 1;) - daca D3 este 0 atunci Q ramane 0

In starea Q = 1:

- daca D3 este 0 se trece in Q = 0, increment = 0

- daca D3 este in continuare 1 atunci increment++

- daca increment >= 3 se trece in starea Q = 2, altfel Q = 1

In starea Q = 2:

- este activa functia alarma() - in aceasta functie se va declansa alarma

- daca D3 = 0 se revine la starea Q = 0 si increment = 0 (reluarea monitorizarii)

Codul asociat este descris in capitolul Implementare Software.

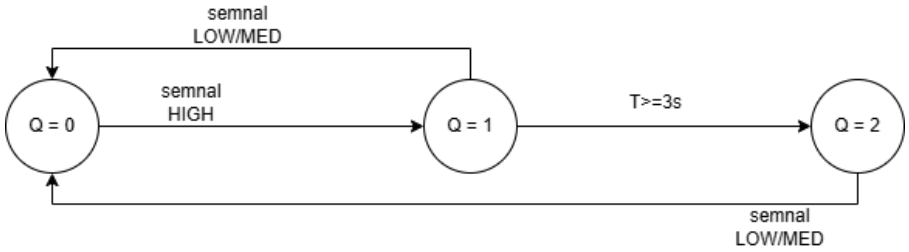

Figura 1.2: Graf asociat CLS

1.4 Procesul secvential / FSM

Am proiectat sistemul astfel incat acesta sa monitorizeze in permanenta intrarile corespunzatoare gradelor de intensitate ale cutremurelor.

Utilizatorul porneste sistemul apasand pe butonul D7 (START). Butoanele sunt active in "0" si functioneaza astfel: semnalul este valabil doar atunci cand utilizatorul nu mai actioneaza butonul. Din punct de vedere SW acest lucru este implementat. Graful de stari nu prevede pasul intermediar pentru a nu ingreuna schema (exceptie RST/).

Utilizatorul poate actiona astfel:

- testIndicatori() <=> S = 1 si se aprind indicatorii de nivel cutremur

- testAlarma() <=> S = 5 si se testeaza si buzzerul + ledul intermitent

- monitorizare() <=> S = 2 => interpretarea datelor de intrare

Aceste functii corespund starilor respective si sunt detaliate in capitolul Implementare Software.

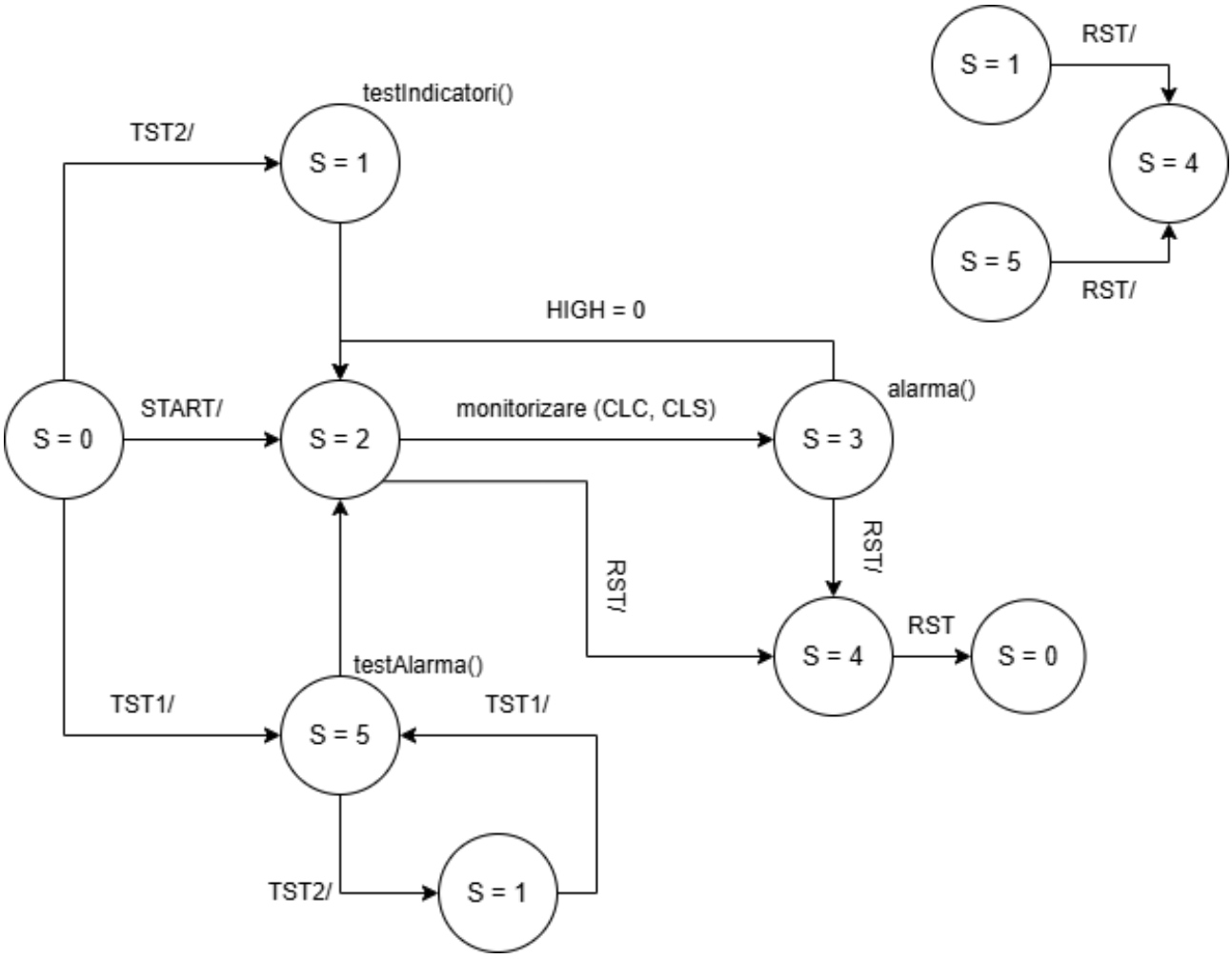

Figura 1.3: Graf asociat FSM

2 Implementare Software

2.1 Implementare CLC

Implementarea CLC-ului se face utilizand metoda OUT = TAB[IN].

```
void clc(void)
{
    char index;
    index = (in & 0x0E) >> 1;
    out = TAB[index];
}
```

Unde:

char TAB[8] = {0x7F, 0x6F, 0x53, 0x7F, 0x33, 0x7F, 0x7F, 0x7F};

in = PIND;

2.2 Implementare CLS

Folosesc metoda switch - case pentru a implementa CLS-ul mai usor deoarece se face testul asupra unei variabile de tip "timer" strans legata de clock-ul intern al microcontrolerului.

```
void cls(void){
  switch(Q){
     case 0:
            if((in & 0x08) != 0) Q = 1;
     break;
     case 1:
            if((in & 0x08) == 0){ Q = 0; increment = 0;}
            if(timer == 0) increment = increment + 1;
            if(increment >= 3) Q = 2;
     break;
     case 2:
            alarma();
            if((in & 0x08) == 0){ Q = 0; S = 2; increment = 0;}
     break;
  }
  }
```

Daca in starea 0 este receptat un impuls HIGH, se trece in starea 1 unde sunt testati bitul pentru nivelul HIGH si variabila "increment".

```
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x3C;
timer = (timer + 1)%50;
timer_inter = (timer_inter + 1)%50;//pentru led intermitent

}
```

O data la fiecare 20 ms timerul creste si se reseteaza dupa 20 X 50 (1 s). Astfel, cand (timer == 0) => increment = increment + 1. Daca trec 3 secunde se trece in starea 2, starea de alarma.

Daca HIGH nu mai este activ, se reseteaza CLS (revine in starea 0).

2.3 Functii utilizate in FSM

Implementarea functiilor utilizate in starile FSM-ului.

```
void monitorizare(void)
{
    clc();
    cls();
}
void testIndicatori(void)
{
    out = 0x83; //activez semnale vizuale
}
    Se aprind indicatorii de nivel al cutremurului.

void testAlarma(void)
{
    out = 0x10; //testez alarma
}
    Se aprind buzzerul si ledul rosu.
```

```
void alarma(void) (
{
    out = 0x30;
    if (timer_inter < 25)
        out |= (1 << 0);
    else
        out &= ~(1 << 0);
    S = 3;
}
```

Se aprind buzzerul, afisajul de mesaj si ledul rosu intermitent. Pe principiul timerului din CLS, se aprinde intermitent ledul rosu (OUT 1) cu frecventa = 1 Hz.

3 Implementare Hardware

Pentru proiectarea hardware am utilizat mediul SimulIDE. Respectivul program permite incarcarea unui fisier de tip ".hex" generat automat de compilatorul CodeVisionAVR in microcontrolerul ATMega164p.

Senzorul si intrarile sunt simulate cu ajutorul unui modul cu 8 switchuri. Intrarile corespunzatoare senzorului sunt active in "1", iar D7, D6, D5, D4 sunt folosite asemenea butoanelor.

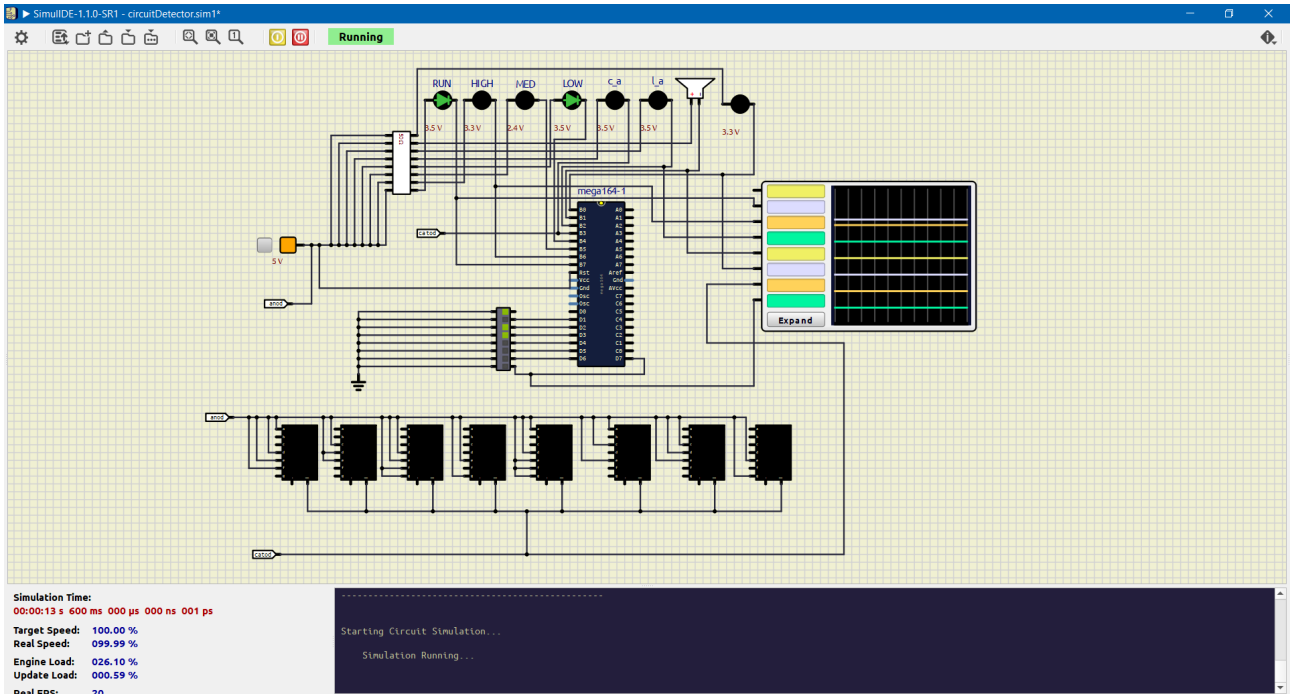

Figura 3.1: Simulare - cutremur LOW

Fiecare led este comandat de pinul corespunzator in functie de situatie (active in "0").

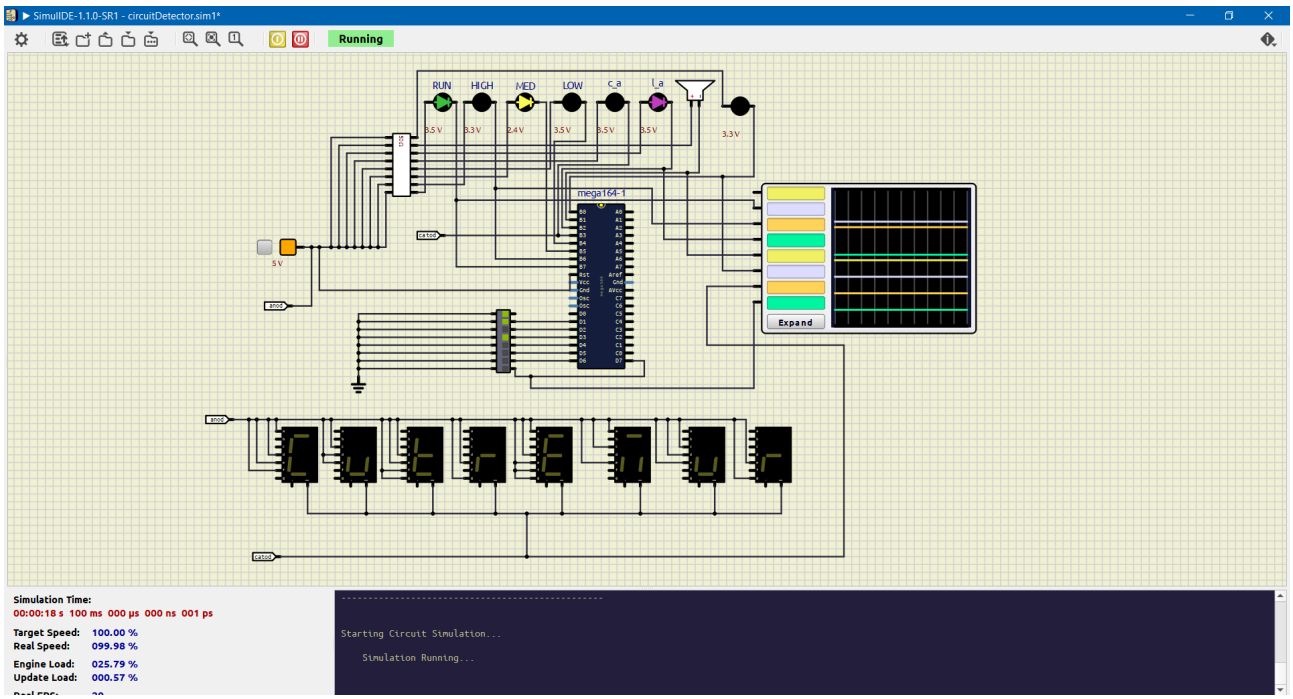

Figura 3.2: Simulare - cutremur MEDIU

Se poate observa ca ledul galben corespunzator este aprins si se afiseaza mesajul de avertizare: "Cutremur".

Pinul B3 (OUT 3) comanda afisajul.

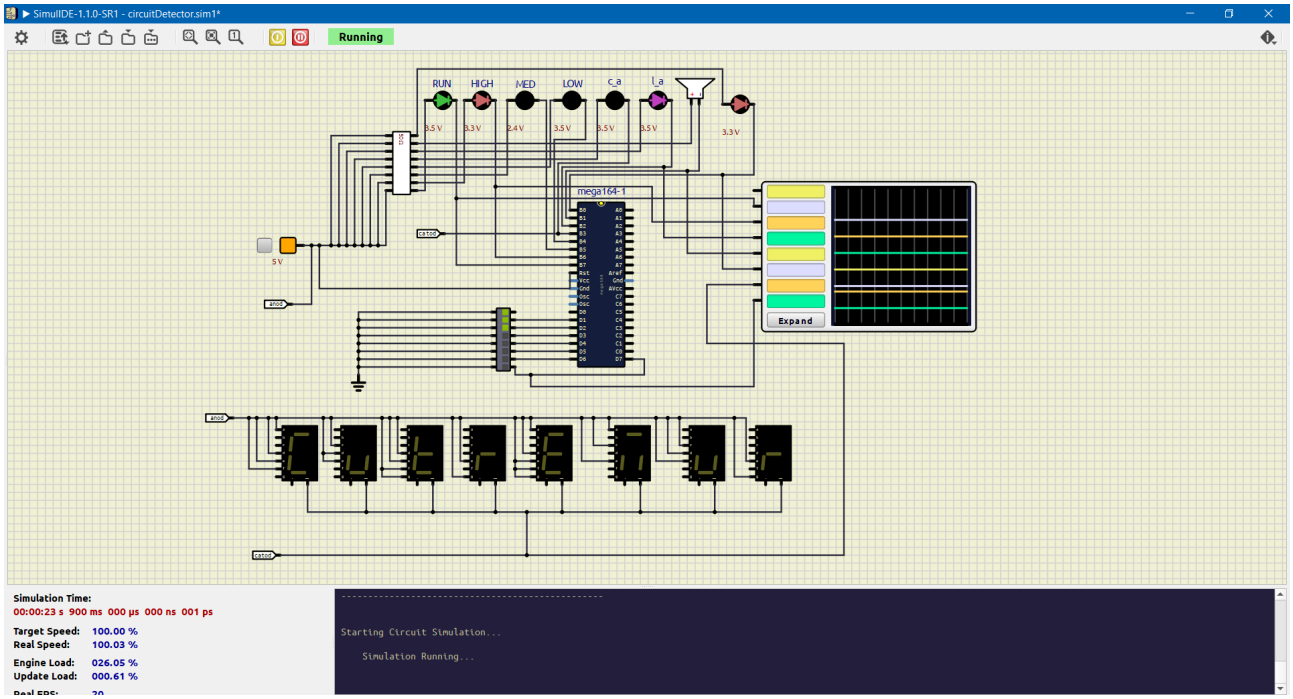

Figura 3.3: Simulare - alarma

Pentru afisarea textului "Cutremur" s-au folosit 8 afisoare pe 7 segmente. Pentru polarizarea ledurilor sa folosit un multiport rezistiv de 50 de ohmi.

Alimentarea se face de la o sursa constanta de 5V. Pentru buzzer s-a folosit un difuzor pe 350 Hz.

Concluzii

Proiectul poate fi imbunatatit prin optimizarea procesului secvential si adaugarea unui model real pentru senzorul de cutremur. In acest proiect am simulat existenta unui modul cu senzor care emite valori discrete pentru intensitatea cutremurului (conversie A-D externa), dar ar fi interesant de vazut daca se poate modifica logica de programare astfel inca microcontrolerul sa primeasca in timp real valori de la senzor si sa efectueze conversia analog-digital in interiorul sau. Pentru aceasta ar trebui efectuata o comunicatie seriala cu senzorul prin intermediul unui port dedicat.

Pe de alta parte, mai pot fi adaugate aspecte precum implementarea unui LCD in loc de acele afisoare pe 7 segmente sau utilizarea unei alarme care sa poata emite un sunet programabil, nu doar un sunet constant la 350 de Hz.

In ceea ce priveste interfata cu utilizatorul, proiectul implementeaza un protocol simplu (apasarea unor butoane pentru testari sau activare sistem), dar poate fi dezvoltat prin permiterea utilizatorului sa interactioneze sau sa modifice parametrii de lucru pentru stabilirea valorilor de referinta a intensitatii cutremurului.

Proiectul duce lipsa de gestionarea clara a eventualelor erori care nu sunt tratate explicit.

Anexa

```
#include <mega164.h>

char S; //stare PS
char Q = 0; //stare CLS
char in;
char out;
int timer = 0;//pentru cls
int timer_inter = 0;
int increment = 0;
char TAB[8] = {0x7F, 0x6F, 0x53, 0
   x7F, 0x33, 0x7F, 0x7F, 0x7F};
void stareInactiva(void);
void testIndicatori(void);
void testAlarma(void);
void monitorizare(void);
void clc(void);
void cls(void);
void alarma(void);

interrupt [TIM0_OVF] void
   timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x3C;
timer = (timer + 1)%50;
timer_inter = (timer_inter + 1)
   %50;

}

void main(void)
{
//am sters codul generat automat
   de AVR pentru lizibilitate
//este necesar totusi pentru
   rularea codului
#asm("sei")

S = 0;
Q = 0;
out = 0xFF;
while (1)
{
in = PIND;

switch(S)
{
case 0:
    if((in & 0x20) == 0) S = 1;
    if((in & 0x40) == 0) S = 5;
    if((in & 0x80) == 0) S = 2;
    stareInactiva();
break;

case 1:
    if((in & 0x20) != 0)
       testIndicatori();
    if((in & 0x80) == 0) S = 2;
    if((in & 0x40) == 0) S = 5;
    if((in & 0x10) == 0) S = 4;
break;

case 2:
    monitorizare();
    if((in & 0x10) == 0) S = 4;
break;

case 3:
    cls();
    if((in & 0x10) == 0) S = 4;
    if((in & 0x08) == 0) {S = 2;
       out = 0xFF;increment = 0; Q
        = 0;}
break;

case 4:     //reset manual
    if((in & 0x10) != 0) S = 0;

break;

case 5:
    if((in & 0x40) != 0)
       testAlarma();
```

```
    if((in & 0x20) == 0) S = 1;
    if((in & 0x80) == 0) S = 2;
    if((in & 0x10) == 0) S = 4;
break;
}

PORTB = out;
};
}

void monitorizare(void)
{
clc();
cls();
}

void testIndicatori(void)
{
out = 0x83; //activez semnale
   vizuale
}
void testAlarma(void)
{
out = 0x10; //testez alarma
}

void stareInactiva(void)
{
out = 0xFF;
}
void clc(void)
{
char index;
index = (in & 0x0E) >> 1;
out = TAB[index];
}

void alarma(void)  (
{
out = 0x30;
if (timer_inter < 25)
    out |= (1 << 0);
else
    out &= ~(1 << 0);
S = 3;
}

void cls(void)
{
switch(Q)
{
case 0:
    if((in & 0x08) != 0) Q = 1;
break;
case 1:
    if((in & 0x08) == 0){ Q = 0;
       increment = 0;}
    if(timer == 0) increment =
       increment + 1;
    if(increment >= 3) Q = 2;
break;

case 2:
    alarma();
    if((in & 0x08) == 0){ Q = 0; S
        = 2; increment = 0;}
break;
}

}
```