Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Automatică și Tehnologia Informației

PROIECT CLP

Intersecție semaforizată
Echipa 4

Ciorîță Adrian

Chiriac Valentin

Ciovică Marius

Chelaru Alexandru

Profesor Coordonator: Savu Alexandru

Brașov 2023-2024

Contents

1. Prezentarea proiectului

Sa se proiecteze, folosind circuite logice programabile, un circuit care sa controleze semafoarele dintr-o intersecție.

- Se considera o intersecție de patru drumuri, notate N, E, S, V. Pe direcțiile N-S si EV exista atât trafic rutier cat si trafic pietonal.
- Semafoarele rutiere au trei culori (roșu, galben si verde), iar semafoarele pentru pietoni au doua culori (roșu si verde).
- Semafoarele vor permite intrarea in intersecție a autovehiculelor, in orice direcție, prin rotație, pentru fiecare dintre intrări, in ordinea: **C7**
- Exista o secvență in care toate semafoarele rutiere au activa culoarea roșie, iar cele pietonale culoarea verde

- Temporizarea este următoarea:
• N: Roșu, Galben: 2 secunde, Verde **C1** secunde, Roșu ;
• S: Roșu, Galben: 2 secunde, Verde **C2** secunde, Roșu;
• E: Roșu, Galben: 2 secunde, Verde **C3** secunde, Roșu;
• V: Roșu, Galben: 2 secunde, Verde **C4** secunde, Roșu;
• Pietoni: Roșu, **C5** secunde verde, **C6** secunde verde intermitent (0.5 Hz), Roșu
• Roșu: Cat timp oricare din celelalte semafoare este activ (galben sau verde). La tranziția intre intrări va exista un moment in care toate semafoarele vor avea activa culoarea roșu timp de 1 secunda.

• Circuitul are următoarele intrări:
  - Ceas (clk_i): 100 MHz
  - Reset (reset_n_i): activ in starea 0 logic
  - Intretinere_semafor (service_i): activ in starea 1 logic.

• Circuitul are următoarele ieșiri, toate active in starea 1 logic:
  - Rosu_auto_N_o, Galben_auto_N_o, Verde_auto_N_o
  - Rosu_auto_E_o, Galben_auto_E_o, Verde_auto_E_o
  - Rosu_auto_S_o, Galben_auto_S_o, Verde_auto_S_o
  - Rosu_auto_V_o, Galben_auto_V_o, Verde_auto_V_o
  - Rosu_pietoni_o, Verde_pietoni_o
  - afisaj_secunde_o

Cerințe de proiectare:

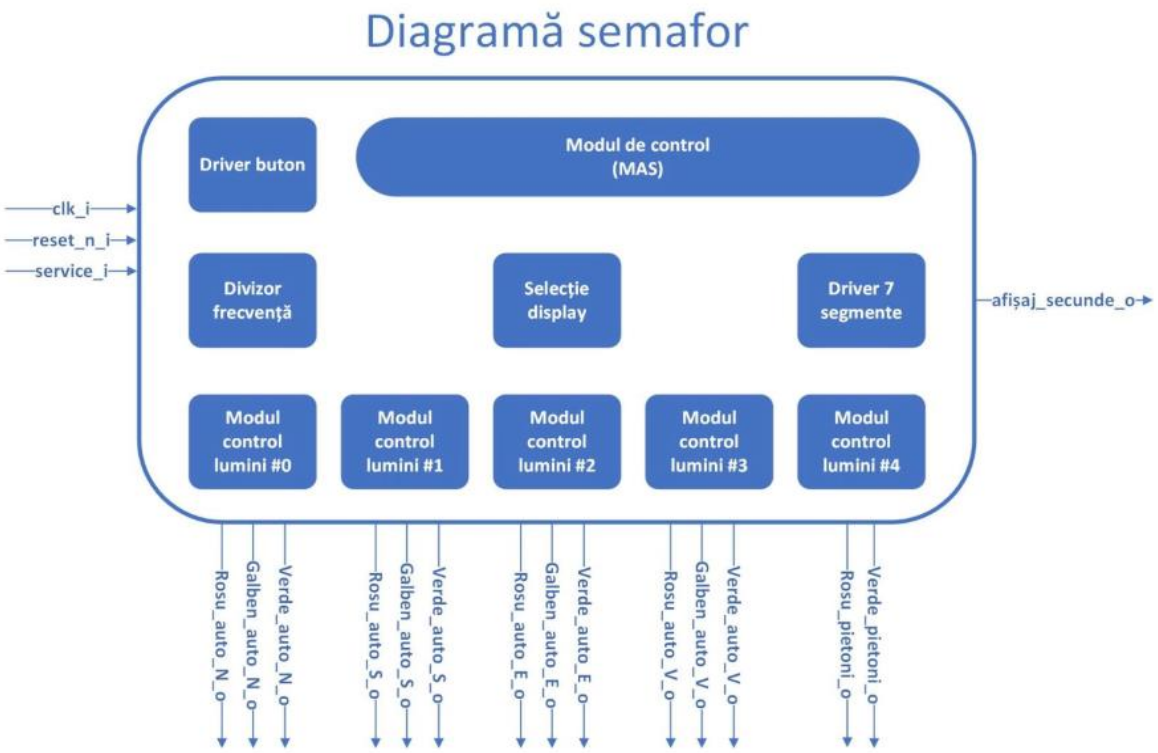
În rezolvarea temei proiectului se vor aborda următoarele probleme:
• Realizarea specificațiilor circuitului proiectat. Se va desena o schemă logică de nivel înalt, pentru fiecare bloc funcțional (numărătoare, sumatoare, (de)multiplexoare,

(de)codificatoare, porți logice, registre, bistabile), precum si pentru interconectarea blocurilor funcționale. Pentru fiecare bloc funcțional se va defini un plan de testare care să conțină minimum 3 teste funcționale.

• Se va realiza modelarea HDL (Verilog) a circuitului pe baza specificațiilor definite anterior.

• Se va realiza un mediu de testare care va permite demonstrarea funcționării circuitului,
implementând testele definite în planul de testare.

• Se va realiza implementarea pe un circuit logic programabil de tip FPGA (toate etapele, pana la generarea fișierului pentru programarea dispozitivului).



1. Datele de proiectare

Nr. proiect	N	S	E	V	Pietoni constant	Pietoni intermitent	Secvență Semafor
4	23	25	28	15	7	8	S-E-V-N

Observație: Valorile notate pentru cele 4 direcții și pietoni reprezintă numărul de secunde cât timp este aprinsă lumina verde.

2. Introducere

În mediul urban, unde spațiul este limitat și creșterea semnificativă a traficului rutier determinată de dezvoltarea urbane, gestionarea eficientă a fluxului de vehicule și a pietonilor este un desiderat major. În această context, dezvoltarea și expansiunea rețelei rutiere sunt adesea constrânte de prezența clădirilor și infrastructurii existente, ceea ce face necesară găsirea de soluții inovatoare pentru fluidizarea traficului.

Un astfel de sistem poate fi compus din programe care elaborează soluții pentru controlul optim al semafoarelor, în funcție de fluctuațiile de trafic. De pildă, un sistem de automatizare a semafoarelor poate fi programat să schimbe timpul de lucru al semafoarelor în funcție de ora zilei, ziua săptămânii sau evenimentele speciale. De asemenea, programele de reglare adaptivă pot să se ajusteze în funcție de fluctuațiile de trafic măsurate în timp real.

Sistemele de inteligent pentru gestionarea traficului includ senzori și sisteme de comunicație pentru a monitoriza și controla traficul. Aceste sisteme pot fi conectate la rețelele Wi-Fi sau 3G/4G pentru a transmite datele colectate la centralele de control și monitorizare. De asemenea, sistemele de inteligent pot fi integrate cu sistemele de navigare pentru a furniza informații utile călătorilor despre ruta cea mai scurtă și despre condițiile de trafic.

Un sistem de gestionare a traficului poate fi compus din componente multiple, cum ar fi:

* Semafoare rutiere cu trei culori: roșu, galben și verde;
* Semafoare pentrupietoni cu două culori: roșu și verde;
* Sistemul de automatizare a semafoarelor;
* Programe de reglare adaptivă;
* Sisteme de monitorizare și control a traficului;
* Senzori pentru măsurarea fluxului de vehicule și pietoni;
* Sistemul de comunicație pentru a transmita datele colectate.
* Sistemul de inteligent pentru gestionarea traficului;
* Sistemul de navigare pentru furnizarea informațiilor călătorilor;
* Centralele de control și monitorizare pentru a procesa datele colectate.

Un astfel de sistem poate fi utilizat pentru a îmbunătăți fluidizarea traficului și siguranța în zona intersecției. De asemenea, poate fi utilizat pentru a reduce timpul pierdut în trafic și pentru a îmbunătăți calitatea vieții cetățenilor.

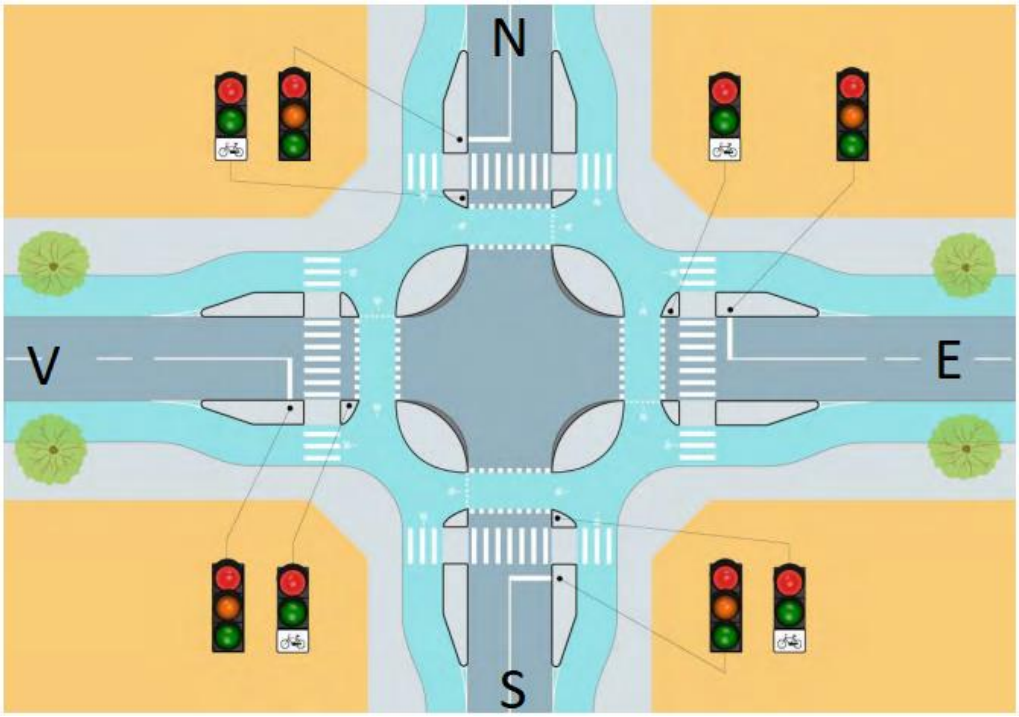


Fig.1. Intersecția

3. Divizorul de frecvență

Utilizarea unui divizor de frecvență în cadrul intersecțiilor semaforizate este esențială pentru gestionarea eficientă a semnalelor de trafic și pentru asigurarea unei funcționări corecte și sincronizate a semafoarelor. Iată câteva motive detaliate pentru care se utilizează un divizor de frecvență în acest context:

1. **Sincronizarea semafoarelor**: În intersecțiile semaforizate, este important ca toate semafoarele să fie sincronizate pentru a preveni conflictele și pentru a optimiza fluxul de trafic. Divizorii de frecvență pot ajuta la generarea unor semnale de temporizare care să coordoneze fazele semafoarelor, asigurându-se că luminile schimbă culorile într-o secvență precisă și sincronizată.
2. **Controlul perioadelor de timp**: Semafoarele trebuie să opereze în cicluri precise, unde fiecare culoare (verde, galben, roșu) este afișată pentru un anumit interval de timp. Divizorii de frecvență sunt utilizați pentru a crea semnale de ceas cu perioade exacte, care determină durata fiecărei faze a semaforului.
3. **Reducerea complexității circuitului**: Utilizarea unui singur oscilator de înaltă frecvență împreună cu divizori de frecvență permite generarea mai multor semnale de timp diferite necesare pentru controlul semafoarelor, fără a fi nevoie de multiple surse de semnal de frecvență diferită. Aceasta simplifică designul și reduce costurile sistemului.

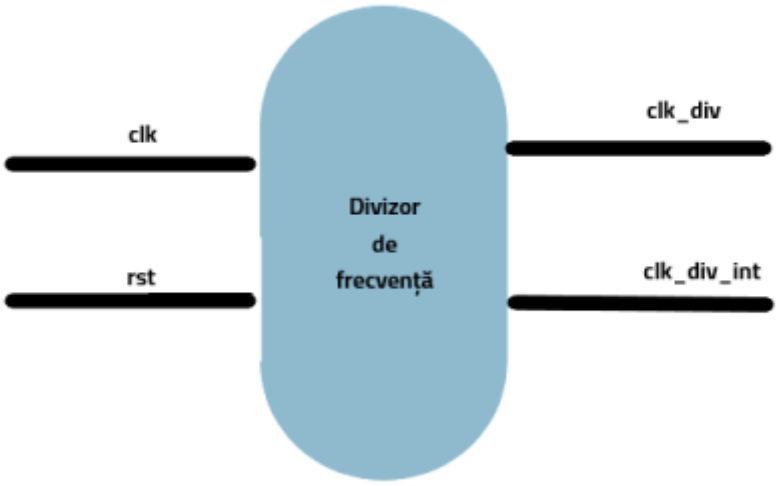


Fig.2. Divizorul de frecvență

Semnalele de intrare și de ieșire din acest modul sunt prezentate în tabelul următor:

Semnale de intrare		**Semnale de ieșire**	
clk	Semnalul de ceas	clk_div	Semnalul ce generează 1 secundă
rst	Semnalul de reset	clk_div_int	Semnalul ce generează 2 secunde

În continuare sunt prezentate modulele aferente divizorului de frecvență.

clk_divider.v

```
module clk_divider(
    input clk,
    input rst,
    output reg clk_div, //1s
    output reg clk_div_int  //2s

);

reg [27:0] counter;
reg [27:0] counter_int;

always @(posedge clk or negedge rst)
begin
    if(~rst) begin
        counter <=0;
        counter_int <=0;
        clk_div_int <=0;
```

```
    end
    else begin
        if(counter_int == 19-1)
        begin
            clk_div_int <= ~clk_div_int;
            counter_int <=0;
        end

        if(counter == 10-1)
        begin
            counter <= 0;
            clk_div <= 0;
        end
        else
        begin
            counter_int <= counter_int+1;
            counter <= counter+1;
            clk_div <= 1;
        end
    end
end
endmodule
```

testbecnh.v

```
module testbench ();

wire clk;
wire rst;

clk_divider CLK_DIV_INST (
    .clk(clk),
    .rst(rst),
    .clk_div(clk_div),
    .clk_div_int(clk_div_int)
);

gen_clk CLK_INST (
    .clk(clk),
    .rst(rst)
);

endmodule
```

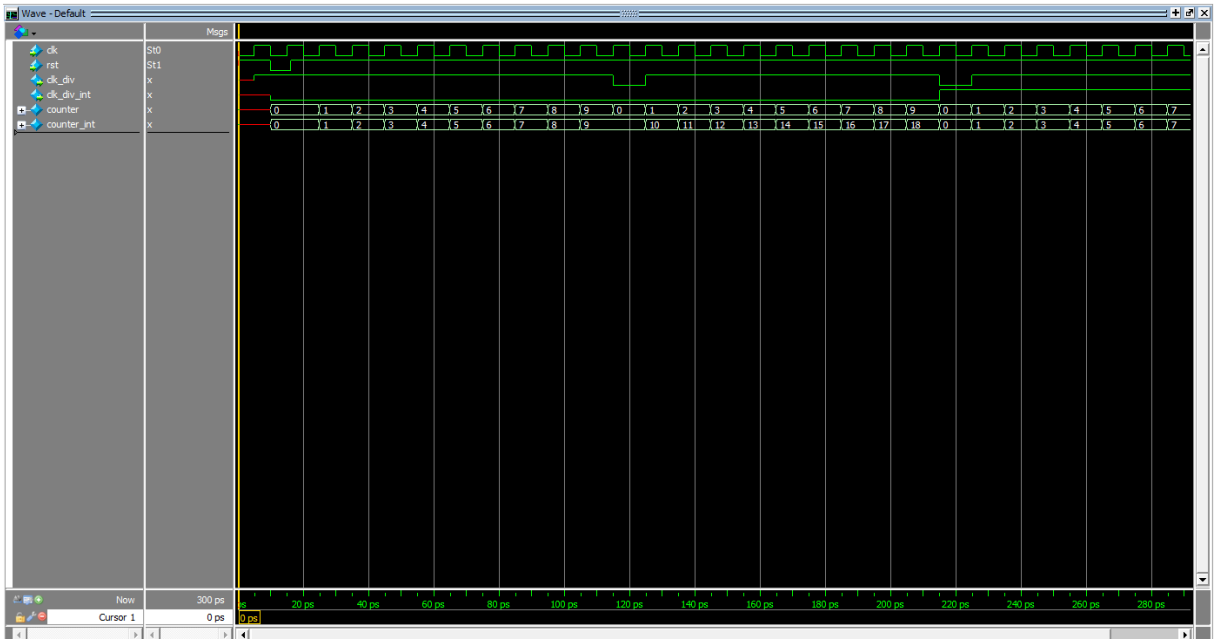


Fig.3. Funcționalitatea divizorului de frecvență

4. Driver de buton

Un diver de buton este un circuit care interfațează un buton mecanic cu un circuit digital sau electronic, asigurând că semnalul generat de apăsarea butonului este adecvat pentru a fi procesat de circuitele următoare. Iată câteva motive pentru care este necesar un diver de buton:

1. **Adaptarea semnalelor**: Butoanele mecanice generează semnale care pot fi prea slabe sau prea zgomotoase pentru a fi interpretate corect de un microcontroler sau alt circuit logic. Diverul de buton amplifică și curăță aceste semnale, făcându-le compatibile cu restul circuitului.
2. **Protecția circuitelor**: Diverul poate include protecții împotriva supratensiunilor sau scurtcircuitelor care ar putea apărea la apăsarea butonului, protejând astfel circuitele sensibile.
3. **Compatibilitate cu tensiuni diferite**: Diverul poate adapta tensiunea semnalului de la buton la nivelul necesar pentru a fi interpretat corect de circuitul digital. De exemplu, un buton care operează la 12V poate fi interfățat cu un microcontroler care operează la 5V.

Un circuit de debounce este esențial pentru a elimina efectele nedorite cauzate de natura mecanică a butoanelor, cum ar fi multiplele tranziții rapide între stările de ON și OFF la fiecare apăsare. Aceste tranziții nedorite, cunoscute sub numele de "bouncing", pot cauza probleme în aplicațiile digitale, deoarece un singur buton apăsat poate fi interpretat ca multiple apăsări. Iată de ce avem nevoie de un circuit de debounce:

1. **Eliminarea zgomotului mecanic**: Când un buton este apăsat sau eliberat, contactele mecanice interne pot sări (bounce) între stările de contact închis și deschis de mai multe ori într-un timp foarte scurt (milisecunde). Acest fenomen poate duce la multiple semnale ON/OFF nedorite. Circuitul de debounce asigură că doar un singur semnal clar este trimis către circuitul de procesare.
2. **Precizie în detectarea apăsărilor de buton**: Asigură că fiecare apăsare și eliberare a butonului este detectată corect și unică, prevenind interpretările eronate și asigurând o funcționare corectă a sistemului.
3. **Stabilitate și fiabilitate**: Asigură că sistemele care depind de apăsările de buton, cum ar fi meniurile de control, tastaturile și alte interfețe utilizator, funcționează fără erori datorate bouncing-ului.

Circuitul de debounce poate fi implementat folosind componente hardware (cum ar fi condensatoare și rezistoare) sau printr-o soluție software, în care microcontrolerul este programat să ignore tranzițiile rapide care apar într-un interval scurt de timp. În acest caz, am ales să folosim un circuit de debounce proiectat software și alcătuit din două bistabile de tip D, o poartă AND și un contor.

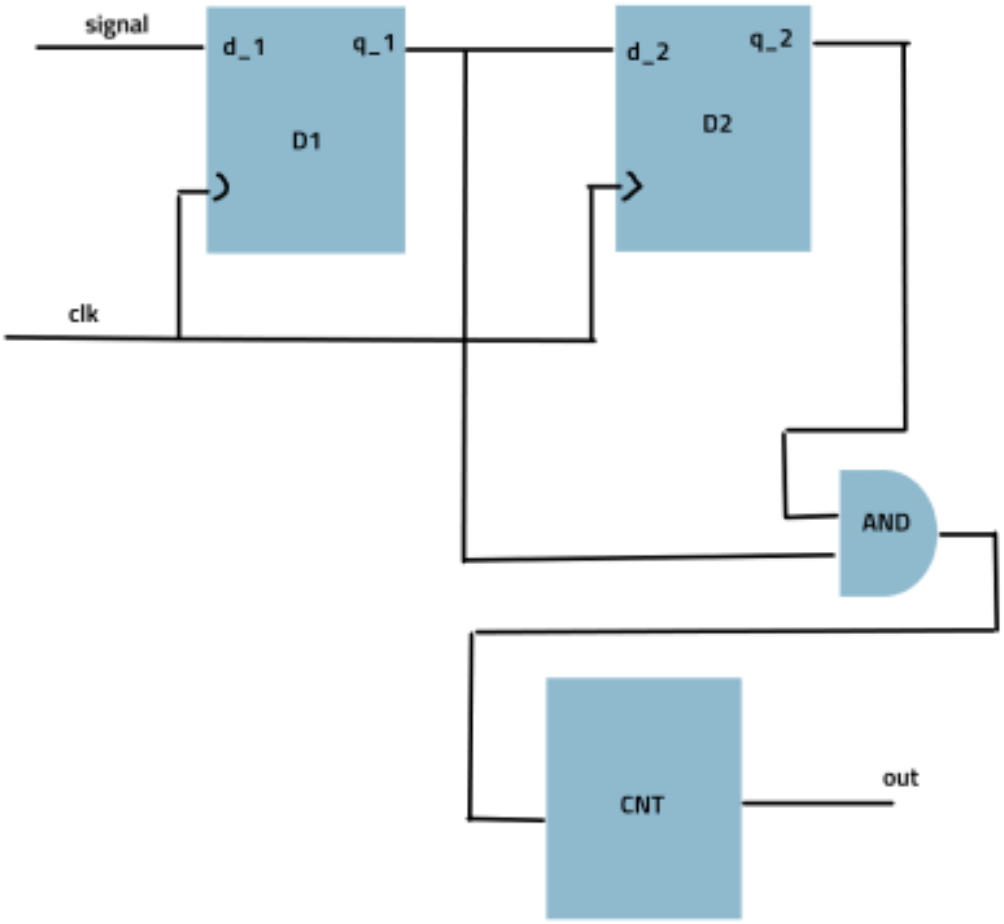


Fig.4. Schema hardware a modulului de debounce

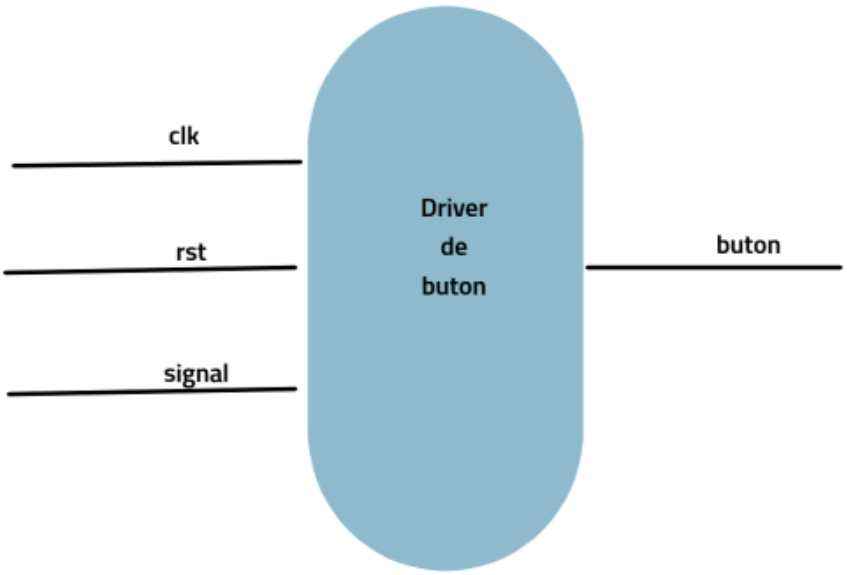


Fig.5. Driver de buton

Semnalele de intrare și de ieșire din acest modul sunt prezentate în tabelul următor:

Semnale de intrare		**Semnale de ieșire**	
clk	Semnalul de ceas	buton	Pulsul generat la detectarea unei apăsări de buton
rst	Semnalul de reset		
signal	Semnalul generat de apăsarea de buton la intrarea în circuit		

În continuare sunt prezentate modulele aferente driverului de buton:

bistabil1.v

```
module bistabil1(
    input      clk,
    input      rst,
    input      signal,
    output reg bistabile
);


 reg q_2;
 reg q_1;
```

```
always @(posedge clk or negedge rst) begin
   if(~rst) begin
      q_1 <= 0;
      q_2 <= 0;

   end
   else begin
       q_2 <= q_1;
       q_1 <= signal;
   end

   bistabile <= (q_1 & q_2);
end

endmodule
```

counter.v

```
module counter(
   input clk,
   input rst,
   input bistabile,
   output reg buton
);

reg[8:0] counter;
reg flag;

always @(posedge clk or negedge rst) begin

   if(~rst)
   begin
      counter <= 5'b00000;
      buton <= 0;
      flag <= 0;
   end
   else
    begin
      if(bistabile)
         counter <= counter + 1;
      else
      begin
         counter <=0;
         flag <= 0;
      end
      if(counter == 5'b11110)
      begin
```

```
            flag <= 1;
            buton <= 1;
            #20
            buton <= 0;
        end
        if(~bistabile)
         counter <= 5'b00000;
    end
end

endmodule
```

gen_input.v

```
module gen_input (
    input wire clk,
    output reg signal
);

//simulare apasare lunga de buton

//initial begin
//    signal <= 1;
//end


//simulare de apasari repetate
//pentru a schimba modul celor doua simulari vom decomenta initial-begin
//si vom comenta structura always

reg [5:0] counter; // Contor pentru numărul de perioade ale semnalului

always @(posedge clk) begin
    if (counter < 2) begin
        signal <= 1'b0;
        counter <= counter + 1;
    end else if (counter < 7) begin
        signal <= 1'b1;
        counter <= counter + 1;
    end else if (counter < 42) begin
        signal <= 1'b1;
        counter <= counter + 1;
    end else if (counter < 47) begin
        signal <= 1'b0;
        counter <= counter + 1;
    end else begin
        counter <= 0; // Resetează contorul
    end
```

```
end
endmodule
```

testbench.v

```
module testbench();

wire clk;
wire rst;
wire bistabile;
wire buton;
wire signal;

bistabil1 BIST1_INST(
    .clk(clk),
    .rst(rst),
    .signal(signal),
    .bistabile(bistabile)
);

counter COUNER_INST(
    .clk(clk),
    .rst(rst),
    .bistabile(bistabile),
    .buton(buton)
);

gen_clk GEN_INST(
    .clk(clk),
    .rst(rst)
);

gen_input INPUT_INST(
    .clk(clk),
    .signal(signal)
);
endmodule
```

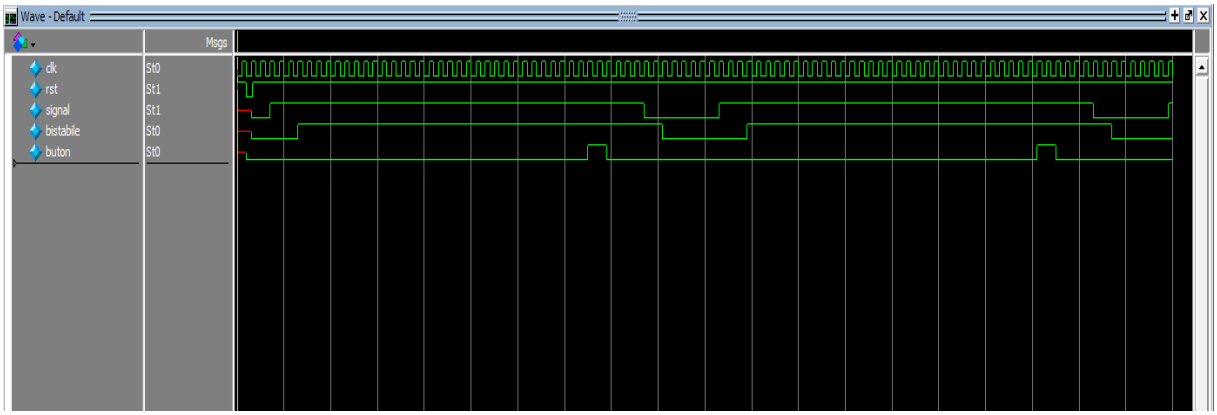


Fig.6. Simulare- apăsări repetate ale butonului

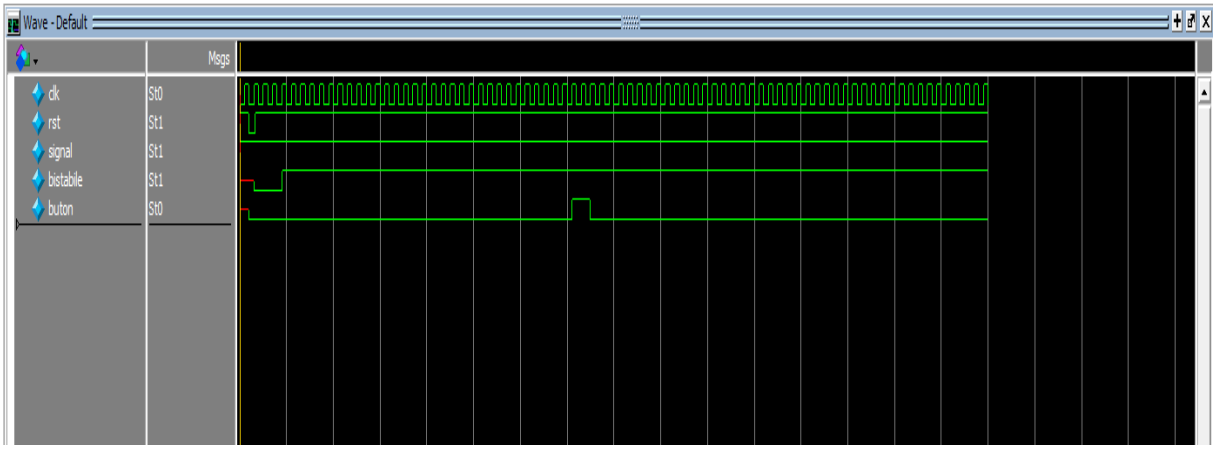


Fig.7. Simulare- apăsare lungă a butonului

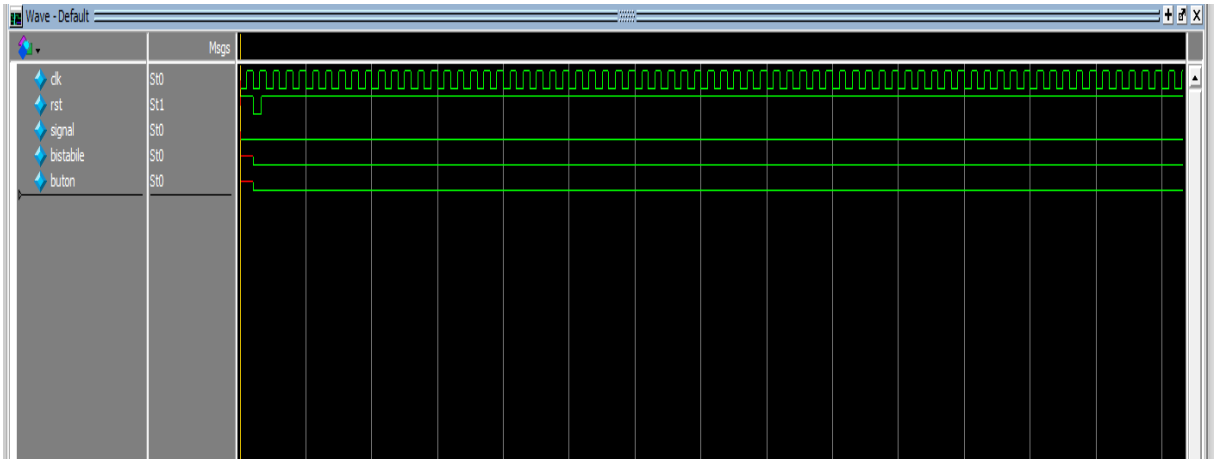


Fig.8. Simulare- buton neapăsat

5. Modulul de MAS

Modul MAS implementează un semafor programabil care poate trece prin mai multe stări în funcție de intrările primite și semnalul de ceas, și furnizează o ieșire corespunzătoare stării curente a semaforului

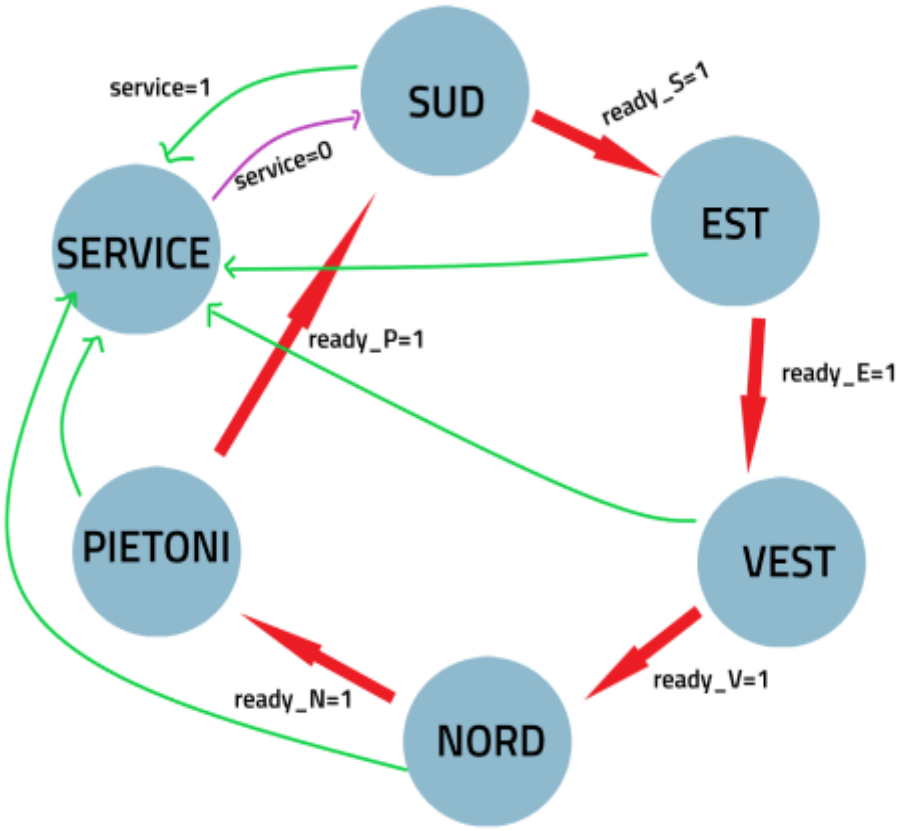


Fig.9. Diagramă de funcționare MAS

Pentru a simplifica diagrama de functionare pentru MAS, am introdus starea de rosu și galben în fiecare modul de lumini. Astfel, în loc de o diagramă cu 14 stătri am obținut o diagramă cu 6 stări.

Starea actuală a semaforului este reprezentată de un număr între 0 și 15, care corespunde diferitelor stări ale semaforului: nord (N), sud (S), est (E), vest (V), pietoni (P) și starea de service(SERVICE).

Semaforul funcționează ciclic, trecând prin stările definite într-o secvență predeterminată, apăsarea butonului de reset asigurând revenirea la starea inițială, iar apăsarea butonului de service asigurând trecerea și ieșirea din această stare.

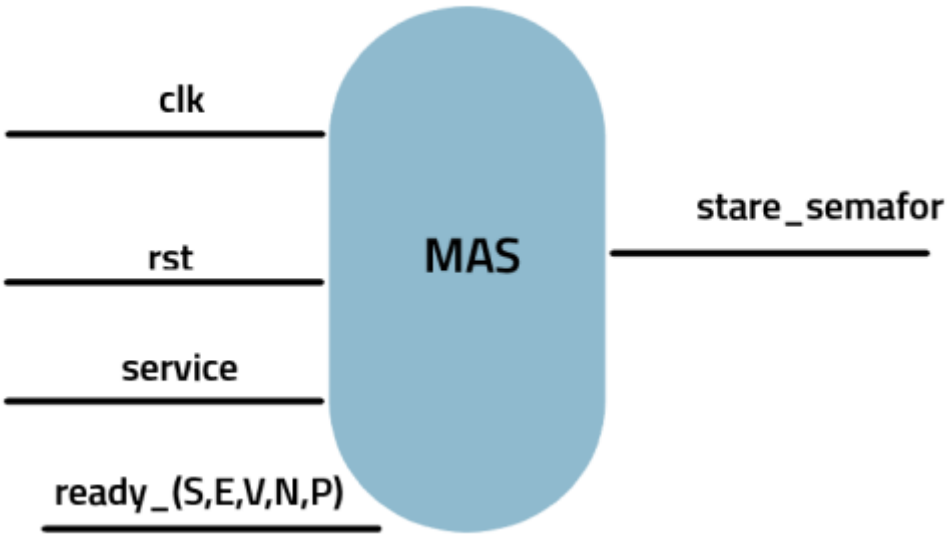


Fig.10. Modulul de MAS

Semnalele de intrare și de ieșire din acest modul sunt prezentate în tabelul următor:

Semnale de intrare		**Semnale de ieșire**	
clk	Semnalul de ceas	stare_semafor	Starea în care se află modulul de MAS în momentul actual
rst	Semnalul de reset		
service	Semnalul generat de apăsarea butonului de service		
ready_(S,E,V,N,P)	Semnalele generate de fiecare modul de lumini după ce s-a terminat secvența acestuia		

În continuare sunt prezentate modulele aferente modulului de MAS:

decoder.v

```
module decoder(

    input clk_div,
    input clk_div_int,
    input clk,
    input rst,
```

```
    input button,
    input service,
    input ready_N,
    input ready_S,
    input ready_E,
    input ready_V,
    input ready_P,
    output reg[2:0] stare_semafor
);

reg[2:0] stare_urmatoare;


localparam SUD     = 3'b000;
localparam EST     = 3'b001;
localparam VEST    = 3'b010;
localparam NORD    = 3'b011;
localparam PIETONI = 3'b100;
localparam SERVICE = 3'b111;


always @(posedge clk or negedge rst) begin
    if(~rst) begin
        stare_semafor   <=3'b000;
        stare_urmatoare <=3'b000;
    end
    else
    stare_semafor <= stare_urmatoare;
end

 always @(*) begin
    case (stare_semafor)
        SUD: if (service)       stare_urmatoare <= SERVICE;
            else if (ready_S)  stare_urmatoare <= EST;
            else if (~rst)     stare_semafor <= SUD;
            else               stare_urmatoare <= SUD;
        EST: if (service)       stare_urmatoare <= SERVICE;
            else if (ready_E)  stare_urmatoare <= VEST;
            else if (~rst)     stare_semafor <= SUD;
            else               stare_urmatoare <= EST;
        VEST: if (service)      stare_urmatoare <= SERVICE;
             else if (ready_V) stare_urmatoare <= NORD;
             else if (~rst)    stare_urmatoare <= SUD;
             else              stare_urmatoare <= VEST;
        NORD: if (service)      stare_urmatoare <= SERVICE;
             else if (ready_N) stare_urmatoare <= PIETONI;
```

```
            else if (~rst)    stare_urmatoare <= SUD;
            else              stare_urmatoare <= NORD;
        PIETONI: if (service)   stare_urmatoare <= SERVICE;
            else if (ready_P) stare_urmatoare <= SUD;
            else if (~rst)    stare_urmatoare <= SUD;
            else              stare_urmatoare <= PIETONI;
        SERVICE: if (~service) stare_urmatoare <= SUD;
        default: stare_urmatoare <= SUD;
    endcase
end

endmodule
```

ready.v

```
module ready (
    input clk,
    output reg ready_s,
    output reg service_s
);

integer count = 0;
integer count1 =0;

initial begin
    service_s = 0;
    ready_s = 0;
end

always @(posedge clk) begin
    if (count < 10) begin
        // Generăm 10 margini ascendente ale semnalului de ceas
        count <= count + 1;
        ready_s <= 0; // Semnalul ready este 0 pentru cele 10 margini ascendente
    end else begin
        // După cele 10 margini ascendente, stabilim semnalul ready la 1
        ready_s <= 1;
           count <=0;
    end
end

    always @(posedge clk) begin
    if (count1 < 100) begin
        // Generăm 10 margini ascendente ale semnalului de ceas
        count1 <= count1 + 1;
         // Semnalul ready este 0 pentru cele 10 margini ascendente
    end else begin
```

```
        // După cele 10 margini ascendente, stabilim semnalul ready la 1
        service_s = ~service_s;
              count1 <=0;
    end
end

endmodule
```

testbench.v

```
module testbench();

wire clk;
wire clk_div;
reg button;
wire [2:0] stare_semafor;
wire service;
wire rst;
wire clk_div_int;
wire ready_S;
wire ready_E;
wire ready_V;
wire ready_N;
wire ready_P;

gen_clk GENERATROR_INST(
    .clk(clk),
    .rst(rst)
);


clk_divider DIVIDER_INST(
    .clk(clk),
    .rst(rst),
    .clk_div(clk_div),
    .clk_div_int(clk_div_int)
);

decoder DECODER_INST(
    .clk(clk),
    .rst(rst),
    .button(button),
    .stare_semafor(stare_semafor),
    .service(),
    .clk_div(clk_div),
```

```
    .clk_div_int(clk_div_int),
    .ready_S(ready_S),
    .ready_E(ready_E),
    .ready_V(ready_V),
    .ready_N(ready_N),
    .ready_P(ready_P)
);


SUD SUD_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .clk_div_int(clk_div_int),
    .clk_div(clk_div),
    .ready_S(ready_S),
    .verde_S(verde_S),
    .galben_S(galben_S),
    .rosu_S(rosu_S)
);

EST EST_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .clk_div_int(clk_div_int),
    .clk_div(clk_div),
    .ready_E(ready_E),
    .verde_E(verde_E),
    .galben_E(galben_E),
    .rosu_E(rosu_E)

);

VEST VEST_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .clk_div_int(clk_div_int),
    .clk_div(clk_div),
    .ready_V(ready_V),
    .verde_V(verde_V),
    .galben_V(galben_V),
    .rosu_V(rosu_V)
);
```

```
NORD NORD_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .clk_div_int(clk_div_int),
    .clk_div(clk_div),
    .ready_N(ready_N),
    .verde_N(verde_N),
    .galben_N(galben_N),
    .rosu_N(rosu_N)
);

PIETONI PIETONI_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .clk_div_int(clk_div_int),
    .clk_div(clk_div),
    .ready_P(ready_P),
    .verde_P(verde_P),
    .rosu_P(rosu_P)
);

wire [7:0] Cell0_i;
wire [7:0] Cell1_i;
wire [7:0] Cell2_i;
wire [7:0] Cell3_i;

TrafficLight TRAFFIC_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .Cell0_i(Cell0_i),
    .Cell1_i(Cell1_i),
    .Cell2_i(Cell2_i),
    .Cell3_i(Cell3_i)
);

//SegDriver drv(
//    .clk(clk),
//    .rst(rst),
//    .Cell0_i(Cell0_i),
//    .Cell1_i(8'b11111111),
//    .Cell2_i(8'b11111111),
//    .Cell3_i(8'b11111111),
//    .seg_o(seg),
```

```
//    .dp_o(dp),
//    .an_o(an)
//    );

ready READY_INST(
    .clk(clk),
    .ready_s(ready),
    .service_s(service)
);

endmodule;
```

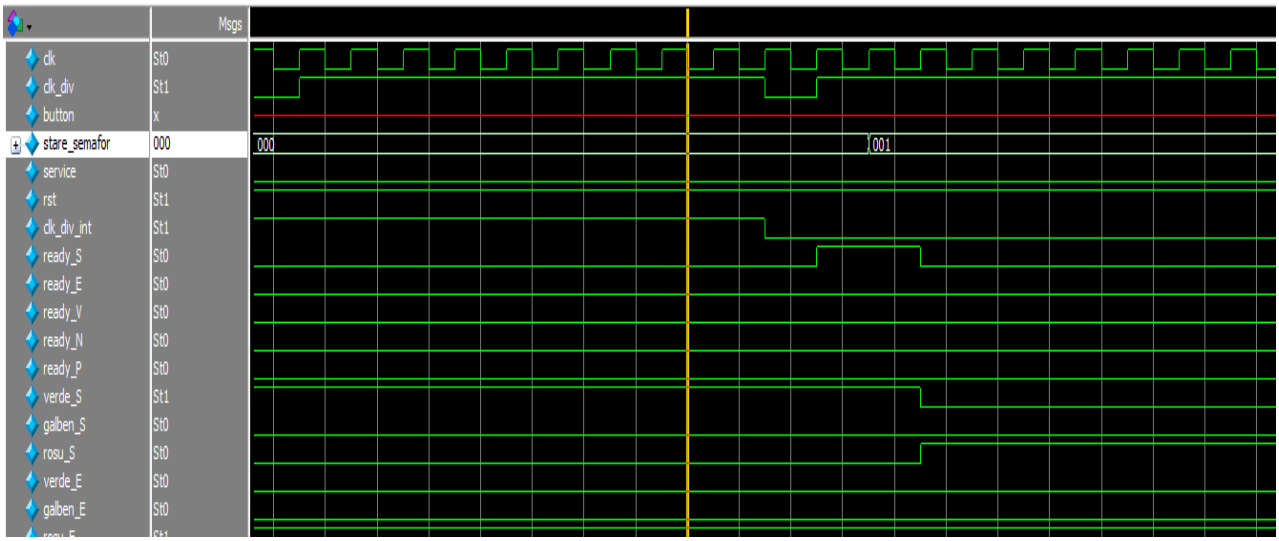


Fig.11.Simulare-Mod normal de funcționare

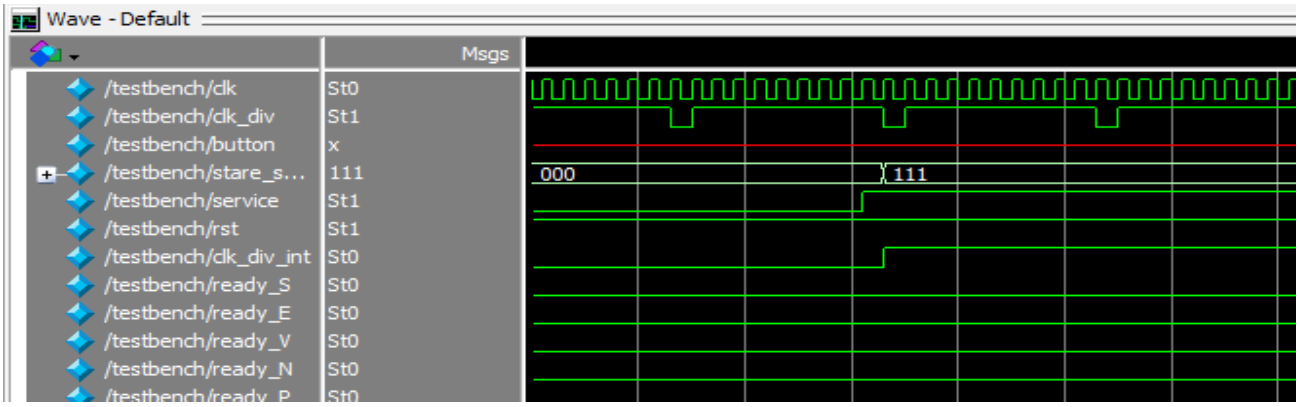


Fig.12.Simulare- Întrare în starea de service

6. Selecție display

Este un modul ce se ocupă de prelucrarea semnalelor transmise de modulul de control către modulul de lumini astfel încât să realizeze afișarea pe un afișor cu 7 segmente a stării în care se află modulul de MAS.

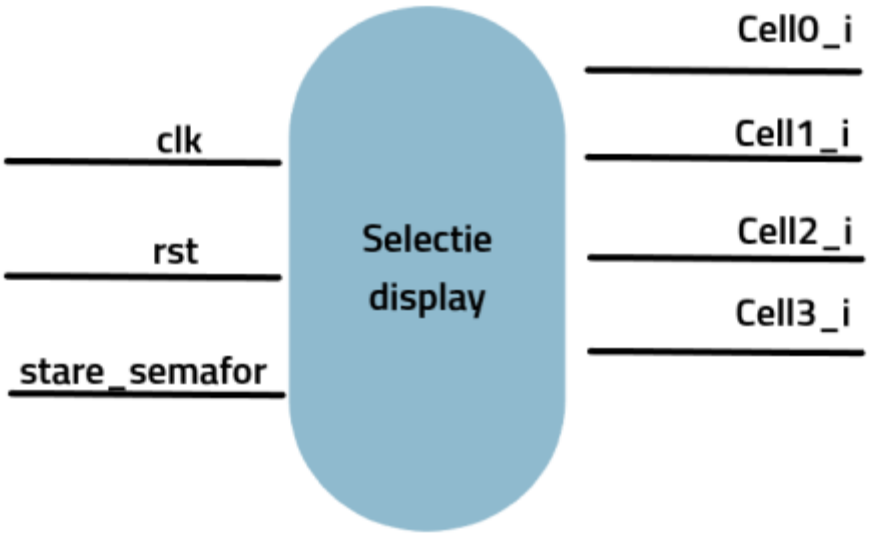


Fig.13. Modulul Selecție Display

Semnalele de intrare și de ieșire din acest modul sunt prezentate în tabelul următor:

Semnale de intrare		**Semnale de ieșire**	
clk	Semnalul de ceas	Cell[0,1,2,3]_i	Codificare binară destinată aprinderii segmentelor afișorului
rst	Semnalul de reset		
stare_semafor	Starea în care se află modulul de MAS în momentul actual		

În continuare sunt prezentate modulele aferente modulului de Selecție Display:

TrafficLight.v

```
module TrafficLight(
    input clk,
    input rst,
    input [2:0] stare_semafor,
    output reg [7:0] Cell0_i,
    output reg [7:0] Cell1_i,
    output reg [7:0] Cell2_i,
    output reg [7:0] Cell3_i
);
```

```
//logica catod comun

   always @(*) begin
      case (stare_semafor)
         3'b000: Cell0_i = 8'b11000000; // SUD        0
         3'b001: Cell0_i = 8'b11111001; // EST        1
         3'b010: Cell0_i = 8'b10100100; // VEST       2
         3'b011: Cell0_i = 8'b10110000; // NORD       3
         3'b100: Cell0_i = 8'b10011001; // PIETONI    4
         3'b111: Cell0_i = 8'b10011110; // SERVICE    6
         default: Cell0_i = 8'b11111111; // off       tot stins
      endcase

      // Celelalte celule nu sunt utilizate(sunt stinse)
      Cell1_i = 8'b11111111;
      Cell2_i = 8'b11111111;
      Cell3_i = 8'b11111111;
   end

endmodule
```

gen_stimuli.v

```
module gen_stimuli(
input clk,
input rst,
output reg [2:0]stare_semafor

);

reg [10:0] counter;

always @(posedge clk or negedge rst)begin

   if(~rst)
      counter <=0;
   else
   begin
      counter <= counter + 1;
      if(counter < 5)
      stare_semafor <= 3'b000;
      else
      stare_semafor <= 3'b001;
   end
end
```

```
endmodule
```

testbench.h

```
module testbench();

wire clk;
wire rst;
wire [2:0] stare_semafor;
wire [7:0] Cell0_i;
wire [7:0] Cell1_i;
wire [7:0] Cell2_i;
wire [7:0] Cell3_i;

TrafficLight TRAFFIC_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor),
    .Cell0_i(Cell0_i),
    .Cell1_i(Cell1_i),
    .Cell2_i(Cell2_i),
    .Cell3_i(Cell3_i)
);


gen_clk GEN_CLK_INST(
    .clk(clk),
    .rst(rst)
);

gen_stimuli STIMULI_INST(
    .clk(clk),
    .rst(rst),
    .stare_semafor(stare_semafor)
);
endmodule
```

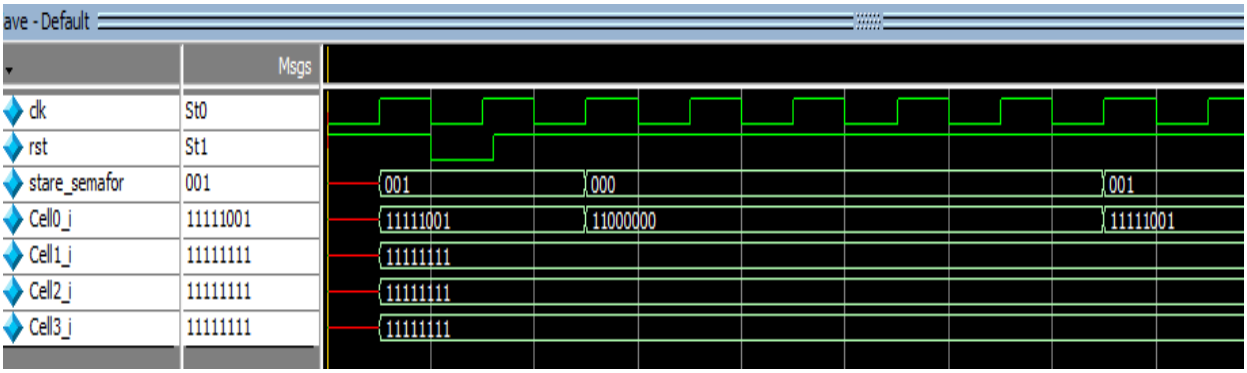


Fig.13.Simulare-Funcționalitatea modulului.

7. Modulul de lumini

Modulul de lumini este responsabil de prelucrarea semnalului de la modulul de MAS. În funcție de codificarea acestuia, sevor prelucra semnalele ce vor activa atât cele 4 direcții destinate autovehiculelor cât și cele pentru pietoni. Pentru claritatea funcționării fiecărei direcții, am definit module individuale pentru directiile N,S,E,V și pietoni.

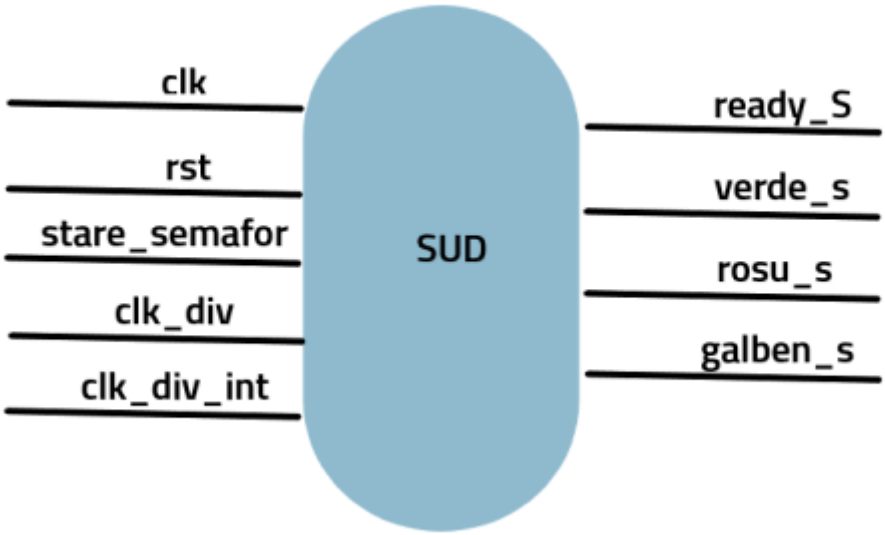


Fig.14. Modulul de lumini - SUD.

Semnalele de intrare și de ieșire din acest modul sunt prezentate în tabelul următor:

Semnale de intrare		**Semnale de ieșire**	
clk	Semnalul de ceas	ready_S	Semnalul ce indică finalizarea schimbului de lumini din starea de SUD
rst	Semnalul de reset	rosu_s	Semnal ce permite aprinderea luminii roșii
stare_semafor	Starea în care se află modulul de MAS în momentul actual	verde_s	Semnal ce permite aprinderea luminii verzi
clk_div	Semnalul ce generează 1 secundă	galben_s	Semnal ce permite aprinderea luminii galbene

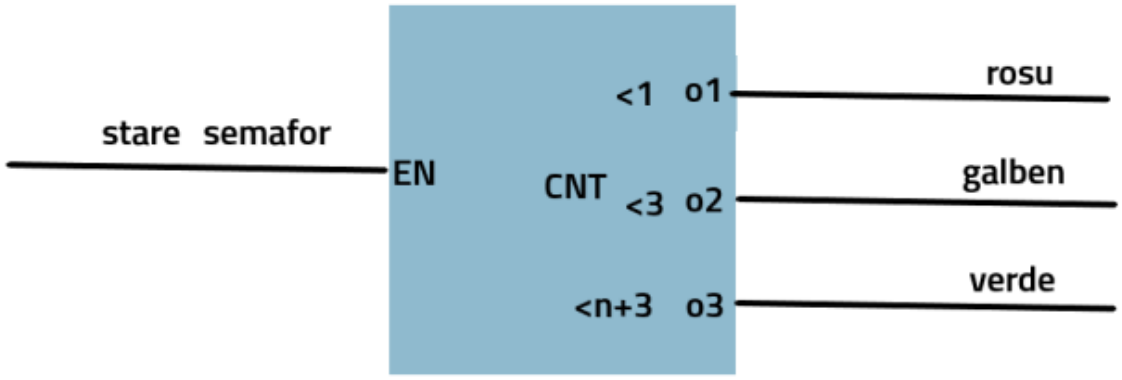


Fig.15. Modulul de lumini –Schemă funcțională

În continuare sunt prezentate modulele aferente blocului de lumini:

SUD.v

```
module SUD (
    input clk,
    input rst,
    input [2:0] stare_semafor,
    input clk_div_int,
    input clk_div,
    output reg ready_S,
    output reg verde_S,
    output reg galben_S,
    output reg rosu_S
);

reg [5:0] counter;

always @(posedge clk or negedge rst) begin
if(~rst) begin
counter <= 0;
ready_S <=0;
end
else begin
    if(stare_semafor == 3'b000)begin
        if(~clk_div) counter = counter+1;

        if(counter <1)
        begin
            rosu_S   <= 1;
            verde_S  <= 0;
            galben_S <= 0;
```

```
      end
        else if(counter <3)
            begin
              rosu_S   <= 0;
              verde_S  <= 0;
              galben_S <= 1;
            end
           else if(counter < 28)
               begin
                 rosu_S   <= 0;
                 verde_S  <= 1;
                 galben_S <= 0;
               end
               else
               begin
                  ready_S <= 1;
                  counter <= 0;
                  #20
                  ready_S <=0;
               end
   end
   else
   begin
      if(stare_semafor == 3'b111)
      begin
         ready_S <= 0;
         if(clk_div_int)
         begin
            rosu_S <=0;
            verde_S <=0;
            galben_S <= 1;
         end
         else
         begin
            galben_S <= 0;
            rosu_S <=0;
            verde_S <=0;
         end
      end
      else
      begin
         rosu_S  <= 1;
         galben_S <= 0;
         verde_S <= 0;
      end
   end
```

```
end
end

endmodule
```

EST.v

```
module EST (
    input clk,
    input rst,
    input [2:0] stare_semafor,
    input clk_div_int,
    input clk_div,
    output reg ready_E,
    output reg verde_E,
    output reg galben_E,
    output reg rosu_E
);

reg [5:0] counter;

always @(posedge clk or negedge rst) begin
if(~rst) begin
counter <= 0;
ready_E<=0;
end
else begin
    if(stare_semafor == 3'b001)begin
        if(~clk_div) counter = counter+1;

        if(counter < 1)
        begin
            rosu_E   <= 1;
            verde_E  <= 0;
            galben_E <= 0;
        end
            else if(counter <3)
                begin
                  rosu_E   <= 0;
                  verde_E  <= 0;
                  galben_E <= 1;
                end
               else if(counter < 31)
                   begin
                     rosu_E   <= 0;
```

```
                verde_E  <= 1;
                galben_E <= 0;
              end
              else
              begin
                ready_E <= 1;
                counter <= 0;
                #20
                ready_E <= 0;
              end
   end
   else
   begin
      if(stare_semafor == 3'b111)
      begin
         ready_E <= 0;
         if(clk_div_int)
         begin
            rosu_E <=0;
            verde_E <=0;
            galben_E <= 1;
         end
         else
         begin
            galben_E <= 0;
            rosu_E <=0;
            verde_E <=0;
         end
      end
      else
      begin
         rosu_E  <= 1;
         galben_E <= 0;
         verde_E <= 0;
      end
   end
end
end

endmodule
```

VEST.v

```
module VEST (
   input clk,
```

```
    input rst,
    input [2:0] stare_semafor,
    input clk_div_int,
    input clk_div,
    output reg ready_V,
    output reg verde_V,
    output reg galben_V,
    output reg rosu_V
);

reg [5:0] counter;

always @(posedge clk or negedge rst) begin
if(~rst) begin
counter <= 0;
ready_V<=0;
end
else begin
   if(stare_semafor == 3'b010)begin
      if(~clk_div) counter = counter+1;

      if(counter < 1)
      begin
         rosu_V   <= 1;
         verde_V  <= 0;
         galben_V <= 0;
      end
         else if(counter <3)
             begin
               rosu_V   <= 0;
               verde_V  <= 0;
               galben_V <= 1;
             end
            else if(counter < 18)
                begin
                  rosu_V   <= 0;
                  verde_V  <= 1;
                  galben_V <= 0;
                end
                else
                begin
                   ready_V <= 1;
                   counter <= 0;
                   #20
                   ready_V <=0;
                end
```

```
    end
    else
    begin
        if(stare_semafor == 3'b111)
        begin
            ready_V <= 0;
            if(clk_div_int)
            begin
                rosu_V <=0;
                verde_V <=0;
                galben_V <= 1;
            end
            else
            begin
                galben_V <= 0;
                rosu_V <=0;
                verde_V <=0;
            end
        end
        else
        begin
            rosu_V  <= 1;
            galben_V <= 0;
            verde_V <= 0;
        end
    end
end
end

endmodule
```

NORD.v

```
module NORD (
    input clk,
    input rst,
    input [2:0] stare_semafor,
    input clk_div_int,
    input clk_div,
    output reg ready_N,
    output reg verde_N,
    output reg galben_N,
    output reg rosu_N
);
```

```
reg [5:0] counter;

always @(posedge clk or negedge rst) begin
if(~rst) begin
counter <= 0;
ready_N<=0;
end
else begin
   if(stare_semafor == 3'b011)begin
      if(~clk_div) counter = counter+1;

      if(counter < 1)
      begin
         rosu_N   <= 1;
         verde_N  <= 0;
         galben_N <= 0;
      end
         else if(counter <3)
             begin
               rosu_N   <= 0;
               verde_N  <= 0;
               galben_N <= 1;
             end
            else if(counter < 26)
                begin
                  rosu_N   <= 0;
                  verde_N  <= 1;
                  galben_N <= 0;
                end
                else
                begin
                   ready_N <= 1;
                   counter <= 0;
                   #20
                   ready_N <= 0;
                end
   end
   else
   begin
      if(stare_semafor == 3'b111)
      begin
         ready_N <= 0;
         if(clk_div_int)
         begin
            rosu_N <=0;
            verde_N <=0;
```

```
            galben_N <= 1;
        end
        else
        begin
            galben_N <= 0;
            rosu_N <=0;
            verde_N <=0;
        end
    end
    else
    begin
        rosu_N  <= 1;
        galben_N <= 0;
        verde_N <= 0;
    end
  end
end
end

endmodule
```

PIETONI.V

```
module PIETONI (
  input clk,
  input rst,
  input [2:0] stare_semafor,
  input clk_div_int,
  input clk_div,
  output reg ready_P,
  output reg verde_P,
  output reg rosu_P
);

reg [5:0] counter;

always @(posedge clk or negedge rst) begin
if(~rst) begin
counter <= 0;
ready_P<=0;
end
else begin
  if(stare_semafor == 3'b100)begin
    if(~clk_div) counter= counter+1;
```

```
    if(counter < 1)
    begin
       rosu_P   <= 1;
       verde_P  <= 0;

    end
       else if(counter <8)
           begin
             rosu_P   <= 0;
             verde_P  <= 1;

           end
          else if(counter < 16)
              begin
             if(clk_div_int)
             begin
                rosu_P <=0;
                verde_P <=1;
             end
       else
       begin
          rosu_P <=0;
          verde_P <=0;
       end

              end
              else
              begin
                ready_P <= 1;
                counter <= 0;
                #20
                ready_P <= 0;
              end
end
else
begin
   if(stare_semafor == 3'b111)
   begin
      ready_P <= 0;
      if(clk_div_int) //2s
      begin
         rosu_P <=0;
         verde_P <=1;
      end
      else
      begin
```

```
            rosu_P <=0;
            verde_P <=0;
         end
      end
      else
      begin
         rosu_P  <= 1;
         verde_P <= 0;
      end
   end
end
end

endmodule
```

Gen_stimuli.v

```
module gen_stimuli(
   input clk,
   input rst,
   input clk_div,
   input clk_div_int,
   input ready_S,
   output reg [2:0] stare_semafor
);

localparam SUD     = 3'b000;
localparam EST     = 3'b001;
localparam VEST    = 3'b010;
localparam NORD    = 3'b011;
localparam PIETONI = 3'b100;
localparam SERVICE = 3'b111;


//scenariu comutare intre stari si reset

//always @(posedge clk or negedge rst) begin
//    if(~rst) begin
//        stare_semafor <= SUD;
//    end
//
//    if(ready_S) begin
//        stare_semafor <= EST;
//    end
//end
```

```
//scenariu service
//always @(posedge clk or negedge rst) begin
//    if(~rst) begin
//        stare_semafor <= SERVICE;
//    end
//end


//scenariu semafor pietoni
always @(posedge clk or negedge rst) begin
    if(~rst) begin
        stare_semafor <= PIETONI;
    end
end

endmodule
```

Observație! Pentru modulul de lumini al pietonilor s-a eliminat culoarea galben. Pentru starea de SERVICE s-a considerat să se afișeze verde intermittent.

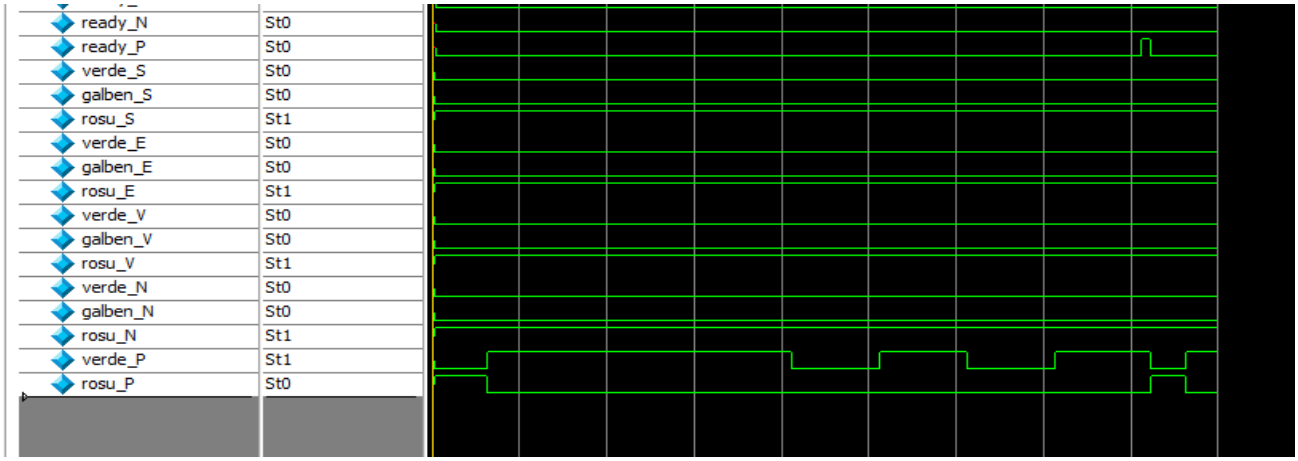


Fig.16. Simulare-Scenariu Pietoni

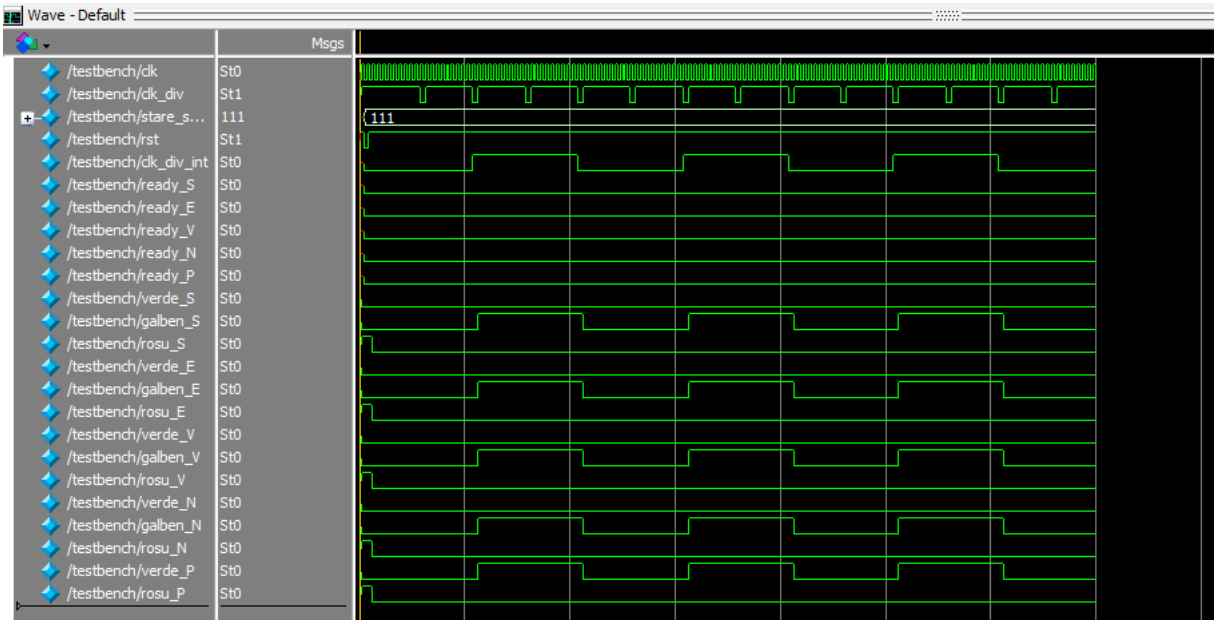


Fig.17. Simulare-Scenariu SERVICE

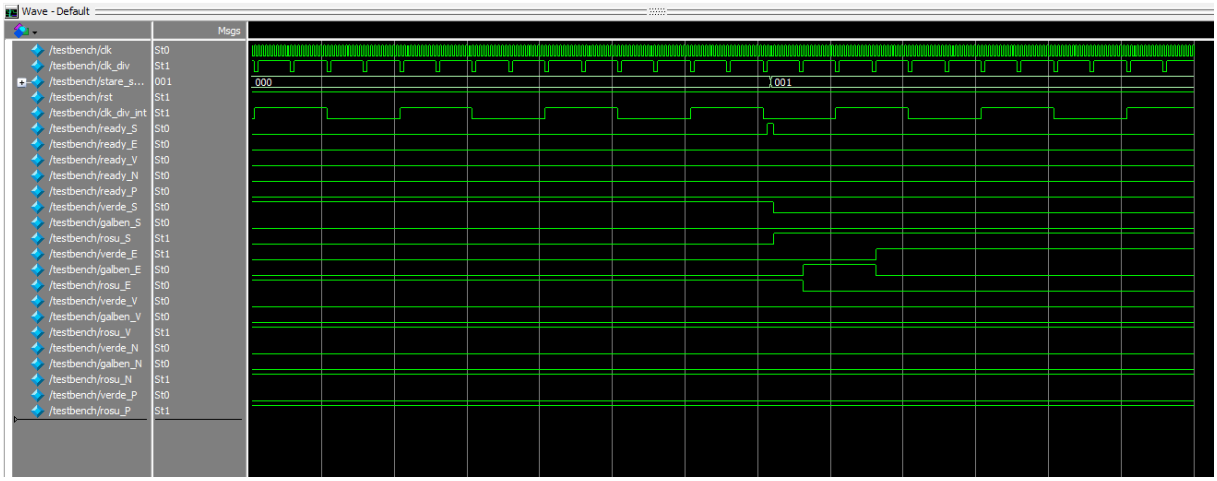


Fig.18. Simulare-Funcționare normal

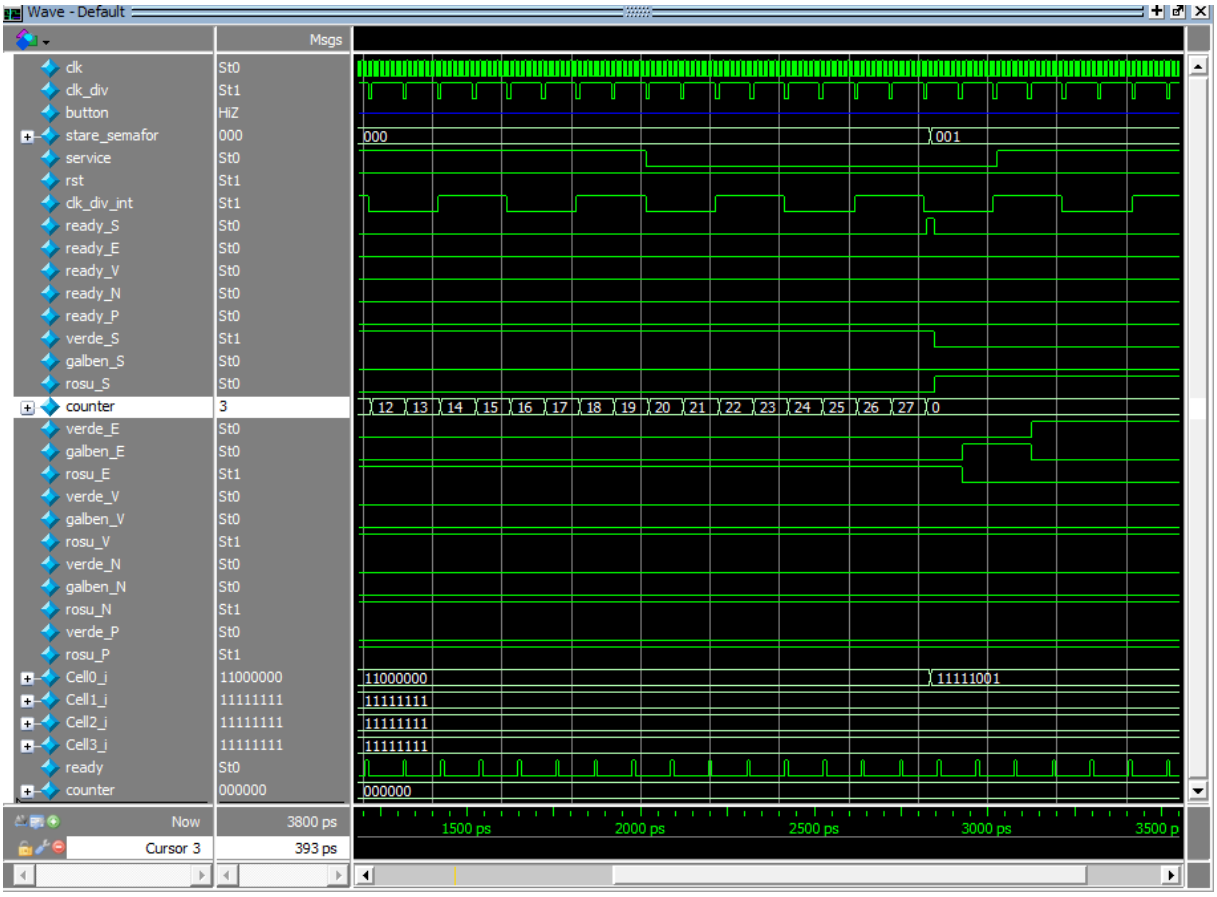


Fig.19. Simulare-Proiect întreg