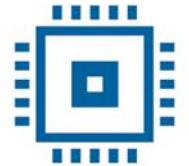




Universitatea Transilvania din Brașov

Facultatea de Inginerie Electrică și Știința Calculatoarelor

Departamentul de Automatică și Informatică Aplicată



Proiect CLP

Echipa 2

Tema 8

Profesor coordonator: Stefan Gheorghe

Studenti: Coroiu Dana-Cornelia

David Alexandra-Bianca

Grabovenco Bogdan-Iulian

Ivașcu Adrian-Mihai

CUPRINS

| | |
|--|----|
| <i>Cerința proiectului</i> | 3 |
| DESCRIEREA SISTEMULUI DE FUNCȚIONARE | 7 |
| <i>Divizor de frecvență</i> | 8 |
| MAS | 17 |
| <i>Selectie display</i> | 22 |
| <i>Modul control lumini (#0; #1; #2; #3; #4)</i> | 25 |

Cerința proiectului

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

- Se considera o intersecție de patru drumuri, noteate N, E, S, V. Pe direcțiile N-S si E- V 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 tranzitia intre intrări va exista un moment in care toate semafoarele vor avea activa culoarea roșu timp de 1 secunda.
- Circuitul are următoarele intrări:
 - o Ceas (clk_i): 10 MHz
 - o Reset (reset_n_i): activ in starea 0 logic
 - o Intretinere_semafor (service_i): activ in starea 1 logic.

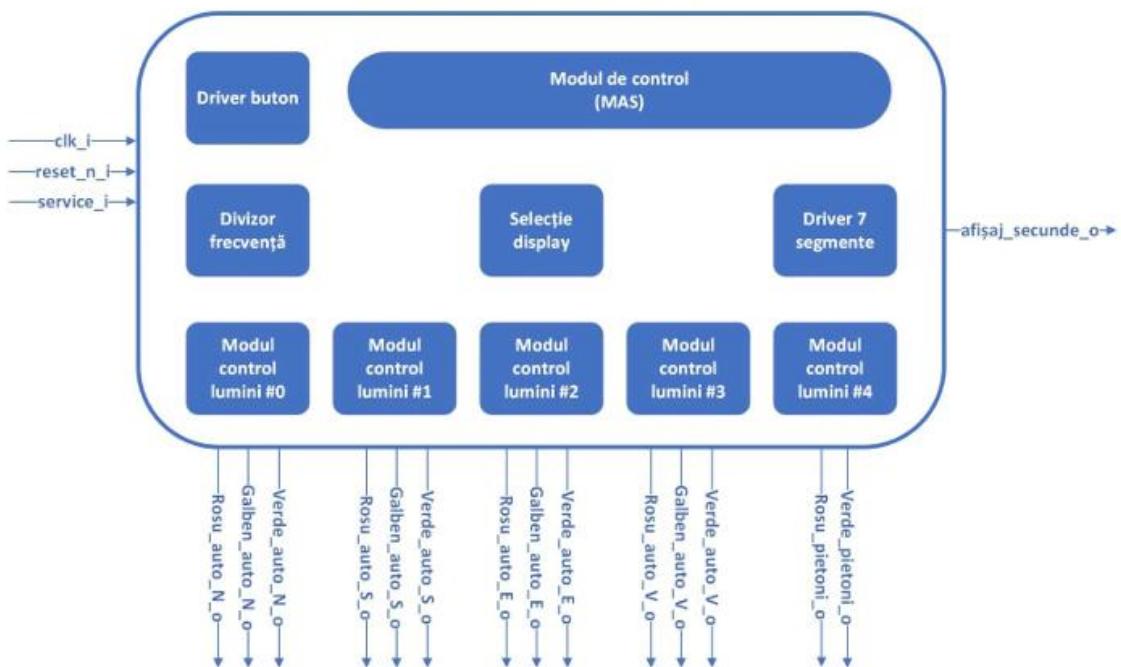
- Circuitul are următoarele ieșiri, toate active in starea 1 logic:
 - Rosu_auto_N_o, Galben_auto_N_o, Verde_auto_N_o
 - Rosu_auto_E_o, Galben_auto_E_o, Verde_auto_E_o
 - Rosu_auto_S_o, Galben_auto_S_o, Verde_auto_S_o
 - Rosu_auto_V_o, Galben_auto_V_o, Verde_auto_V_o
 - Rosu_pietoni_o, Verde_pietoni_o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

Diagramă semafor



1. Date de proiectare:

| Nr. Proiect | N Verde | S Verde | E Verde | V Verde | Pietoni Verde | Pietoni Intermitent Verde | Secventa semafoarelor |
|-------------|---------|---------|---------|---------|---------------|---------------------------|-----------------------|
| 8 | 24 | 28 | 29 | 25 | 12 | 6 | S-N-E-V |

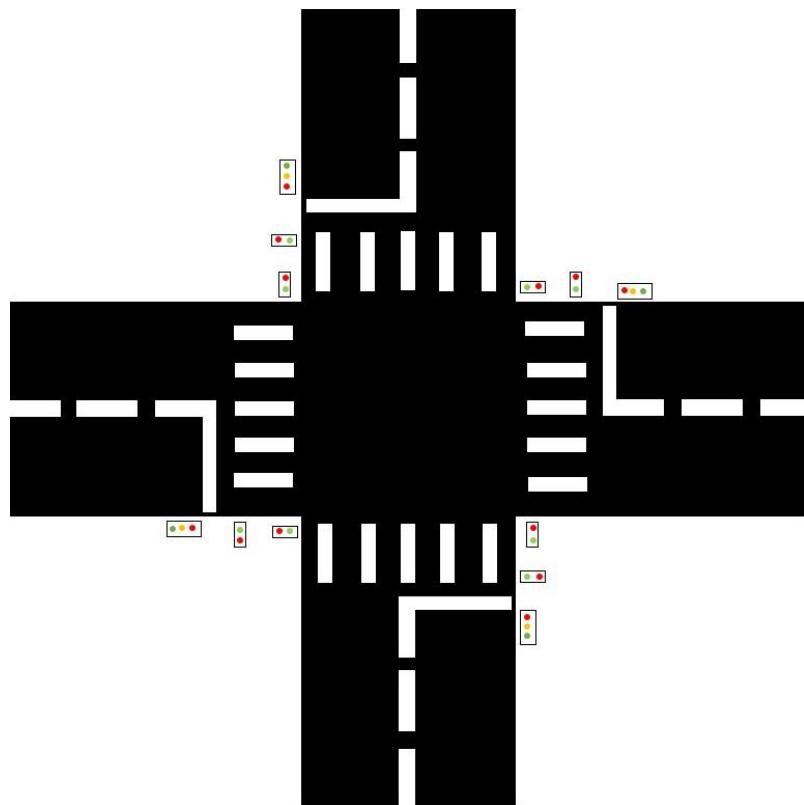
2. Introducere

În mediul urban dezvoltarea și extinderea amenajării spațiului rutier sunt limitate de construcțiile și infrastructura deja existente. Odată cu creșterea spectaculoasă a traficului rutier datorită dezvoltării activităților urbane se încearcă soluții de decongestionare a traficului prin două direcții: printr-o folosire judicioasă a spațiului rutier precum și printr-un control și monitorizare eficiente.

Un sistem de gestiune a traficului poate fi un ansamblu de programe care asigură elaborarea soluțiilor pentru comanda optimă a semafoarelor, în raport cu fluctuațiile de trafic.

În cazul unei intersecții independente, se poate aplica comutarea aprinderii semnalelor luminoase la intervale fixe (sistemele cu automatizare clasică), sau programe de reglare adaptivă în raport cu fluctuațiile de trafic instantaneu obținut din măsurători.

În cele ce urmează vom prezenta o intersecție semaforizată, cu patru drumuri, care urmărește coordonatele S-N-E-V și câte o trecere de pietoni pe fiecare direcție. Semafoarele rutiere au trei culori: roșu, galben și verde în timp ce semafoarele pietonale au două: roșu și verde.



Figură 1. Schema unei intersecții semaforizate

DESCRIEREA SISTEMULUI DE FUNCȚIONARE

Pentru a implementa un astfel de sistem, se va utiliza un automat secvențial sincron care să gestioneze ciclurile semafoarelor. Vom utiliza un contor care să numere secundele, pentru a ști când se schimbă culorile și când se trece la următorul ciclu. Mai întâi, putem implementa un modul care să gestioneze semafoarele rutiere. Acesta va fi format dintr-un 1 automat secvențial sincron care să ruleze ciclul de semaforizare în ordinea S-N-E-V. Automatul are 5 semafoare cu cate 3 stări: roșu, verde și galben /respectiv pentru pietoni:rosu, verde și verde intermitent.

Pentru semafoarele rutiere:

În starea „roșu”, toate semafoarele rutiere sunt închise și circulația este oprită. În starea „galben”, semafoarele rutiere sunt încă închise, dar conducătorii auto sunt avertizați că semaforul se va schimba în curând. În starea „verde”, semaforul corespunzător direcției respective este deschis, iar celelalte sunt închise.

Pentru pietoni:

În starea „roșu”, semaforul pentru pietoni este închis, iar trecătorii trebuie să aștepte. În starea „verde”, semaforul este deschis complet, iar trecătorii pot traversa în siguranță. În starea „verde-intermitent”, semaforul este deschis intermitent, avertizând trecătorii că trebuie să se grăbească.

Pentru a asigura funcționarea sistemului, vom utiliza o serie de module de care să coordoneze ordinea semafoarelor și să genereze semnalele de control necesare pentru fiecare semafor în parte.

Divizor de frecvență

Pentru a se putea folosi orice circuit digital care urmărește să interacționeze cu oameni, trebuie mai întâi proiectat un divizor de frecvență. Încetinirea frecvenței cu care un circuit digital lucrează sau întârzierea tactului după care se face sincronizarea datelor circuitului digital este necesară când se folosesc funcționalități temporale. Astfel pentru a ajunge să se folosească unitatea de timp de o secundă, unitate perceptibilă de către om, trebuie să întârziem tactul ceasului după care se face sincronizarea, spre exemplu, la o frecvență de 100 MHz sunt sincronizate datele la o perioadă de 10 ns(nanosecunde), pentru a putea însă folosi doar o secundă se va proceda în felul următor:

La fiecare front crescător de ceas se va incrementa cu ajutorul unui circuit de counter valoarea unui contor până când contorul va atinge limita dată de frecvență, revenind la exemplul anterior, când contorul înregistrează 100 de milioane de incrementări, atunci contorul va fi numărat o secundă.

Pentru îndeplinirea celei de-a doua cerințe, însemnând atingerea unei frecvențe de 0.5 Hz, se va incrementa primul contor, iar când primul contor emite 2 semnale, atunci va da semnal și cel de-al doilea contor.

Driver de buton

Driverul de buton este folosit pentru a trece tot circuitul într-o stare de service. Acest aspect a fost simulațat folosind 3 tipuri de apăsări: scurtă, medie și lungă, fiecare având un anumit număr de ceasuri alocate care indică cât timp operatorul ține butonul apăsat.

Driver de buton, dar mai exact butonul care este o componentă fizică, iar apăsarea acestuia nu poate fi făcută în condiții ideale, însemnând datorita faptului că butonul este un element mecanic, în momentul apăsării, vor apărea inevitabil fluctuații sau spike-uri care pot produce funcționări eroante ale circuitului. Problema aceasta este rezolvată prin detectarea frontului și înregistrarea semnalului generat cu întârziere, de regulă, se folosesc 2 bistabile a căror ieșiri sunt legate la o poartă SI, dar ieșirea celui de-al doilea bistabil trebuie negată, se folosește ieșirea sa negată. Astfel sunt evitate zonele unde se produc spike-uri mecanice.

Implementare

Implementarea s-a făcut ajutorul ModelSim-Intel FPGA, iar secțiunea de cod a fost scrisă cu editorul de text Notepad++. În continuare vor fi atașate programele pentru fiecare modul. Atât divizorul de frecvență, cât și driverul de buton sunt descrise după următoarele figuri.

Divizor de frecvență

```
1  module ctu #(parameter NrBiti = 27, parameter Limita = 'h3B9AC9FF)
2  //2^27 - 1 = 'hFFFFFF (maximumul posibil cu 27 de biti)
3  //Limita = 10^8 - 1 = 'h3B9AC9FF
4  //10^8 = h5F5E100
5  (
6    input clk_i,
7    input rst_i,
8    input enable_i,
9    input count_up_i,
10
11   output reg [NrBiti-1:0] data_o,
12   output reg overflow_o,
13   output reg overflow2sec_o,
14   output reg [1:0] check_ovf_o
15 );
16
17  always @(posedge clk_i or negedge rst_i) //data_o
18 begin
19   if (!rst_i) data_o <= 'b0; //cand rst_i = 0, data_o se reseteaza
20   else if (!enable_i) data_o <= data_o;
21   else if (count_up_i) data_o <= data_o + 1;
22   else data_o <= data_o;//daca nu se declara explicit, compilatorul o va face automat in locul nostru
23 end
24
```

Fig 1.5.1
Counter (I)

```
24
25  always @(posedge clk_i or negedge rst_i) //overflow_o
26 begin
27   if (!rst_i) overflow_o = 'b0;//cazul rst_i = 0, overflow_o se reseteaza
28   else if ( (data_o == Limita) & (enable_i & count_up_i) )
29 begin
30   overflow_o <= 'b1;
31   data_o <= 'b0;
32 end
33   else overflow_o <= 'b0;//isi perpetueaza valoarea daca overflow_o <= overflow_o;
34 end
35
36  always @(posedge clk_i or negedge rst_i) //overflow2sec_o
37 begin
38   if (!rst_i) overflow2sec_o <= 'b0;
39   else if (check_ovf_o == 'b10) overflow2sec_o <= 'b1;
40   else overflow2sec_o <= 'b0;
41 end
42
43  always @(posedge clk_i or negedge rst_i) //check_ovf_o
44 begin
45   if(!rst_i) check_ovf_o <= 'b0;
46   else if(overflow_o) check_ovf_o <= check_ovf_o + 'b1;
47   else if(check_ovf_o == 'b10) check_ovf_o <= 'b0;
48   else check_ovf_o <= check_ovf_o;
49 end
50
51 endmodule
```

Fig 1.5.2
Counter (II)

Scenariul de counter unde sunt inițializate semnalele de intrare.

```
1 module scn_ctu #(parameter Limita = 'b10000)//la 16 fronturi vine un reset
2 (
3     output reg enable_o,
4     output reg count_up_o
5 );
6
7     initial
8     begin
9         enable_o <= 'b1;
10        count_up_o <= 'b1;
11    end
12
13 endmodule
```

Fig 1.5.3
Scenariul de counter

Driver de buton

Pentru driverul de buton am folosit 4 bistabile de tip D legate în serie. Detectarea apăsării butonului se face cu ajutorul ieșirii negate ale celui de-al patrulea bistabil și ieșirea adevărată a celui de-al treilea bistabil, ieșirile celor două se aduc într-o poartă AND, astfel se evită spike-urile butonului, împiecând un comportament neașteptat. În figuri se va ataşa în ordine modulele care formează driverul de buton.

```
1 module and2
2 (
3     input in1_i,
4     input in2_i,
5
6     output reg out1_o
7 );
8
9     assign out1_o = in1_i & in2_i;
10
11 endmodule
```

Fig 1.6.1
Poarta AND

```

1 | module dff |
2 | (
3 |   input clk_i,
4 |   input rst_i,
5 |   input d_i,
6 |
7 |   output reg q_o,
8 |   output reg qn_o
9 | );
10|
11|   always @ (posedge clk_i or negedge rst_i)
12| begin
13|   q_o <= d_i;
14|   qn_o <= !d_i;
15| end
16|
17| endmodule

```

Fig 1.6.2
Bistabilul D

```

1 | module drv_btn|
2 | (
3 |   input clk_i,
4 |   input rst_i,
5 |   input btn_i,
6 |
7 |   output srv_o
8 | );
9 |
10|   dff DFF_inst_1
11|   (
12|     .clk_i(clk_i),
13|     .rst_i(rst_i),
14|     .d_i(btn_i),
15|
16|     .q_o(q1),
17|     .qn_o()
18|   );
19|   dff DFF_inst_2
20|   (
21|     .clk_i(clk_i),
22|     .rst_i(rst_i),
23|     .d_i(q1),
24|
25|     .q_o(q2),
26|     .qn_o()
27|   );
28|   dff DFF_inst_3
29|   (
30|     .clk_i(clk_i),
31|     .rst_i(rst_i),
32|     .d_i(q2),
33|
34|     .q_o(q3),

```

Fig 1.6.3
Driver de buton (I)

```

34   .q_o(q3),
35   .qn_o()
36 );
37   dff DFF_inst_4
38 (
39     .clk_i(clk_i),
40     .rst_i(rst_i),
41     .d_i(q3),
42
43     .q_o(),
44     .qn_o(qn4)
45 );
46   and2 AND_inst_1
47 (
48     .in1_i(q3),
49     .in2_i(qn4),
50
51     .out1_o(srv_o)
52 );
53
54 endmodule

```

Fig 1.6.4
Driver de buton (II)

```

1  module btn#(parameter CLK = 1) //CLK ceasuri pe 1
2  (
3    input clk_i,
4    output reg btn_o
5  );
6
7  initial
8  begin
9    btn_o <= 'b0; repeat(20) @(posedge clk_i); //20 de ceasuri pe 0
10   //se incepe simularea btn dupa trecerea a 10 ceasuri
11   btn_o <= 'b1; repeat(CLK) @(posedge clk_i); //CLK ceasuri pe 1
12   btn_o <= 'b0; //revenire la stare initiala
13 end
14
15 endmodule

```

Fig 1.6.5
Scenariul asociat driverului de buton

Testbench

Testbench-ul este un alt mod cu care putem simula performanțele DUT-ului obținut. DUT este acronimul lui "device under test". Testbench-ul generează semnale de intrare, monitorizează răspunsurile, și verifică dacă DUT-ul se comportă conform specificațiilor. Aceasta poate include stimulări, verificatori și module de verificare automată a rezultatelor, facilitând astfel identificarea și diagnosticarea problemelor în designul hardware. În figurile de mai jos se va introduce codul testbench-ului.

```
1  module tb();
2
3    localparam SP = 5; //semiperioada ceasului
4    localparam CLK1 = 10; //apasare normală = 10 ceasuri
5    localparam CLK2 = 20; //apasare lungă = 20 ceasuri
6    localparam CLK3 = 5; //apasare scurtă = 5 ceasuri
7    localparam NrBiti = 27;
8    localparam NrBiti1 = 4;
9    localparam Limita = 'h3B9AC9FF;//10^8
10   localparam Limita1 = 'b1111;//15 in decimal - 16 unitati
11
12  wire btn1;
13  wire btn2;
14  wire btn3;
15  wire clk;
16  wire rst;
17  wire srv1;
18  wire srv2;
19  wire srv3;
20  wire enbl;
21  wire count_up;
22  wire [NrBiti1-1:0] dt;
23  wire ovf_1sec;
24  wire ovf_2sec;
25  wire [1:0] chk;
26
27  //ceasul de 100MHz
28  clock #(SP) CLOCK_inst_1
29  (
30    .clk_o(clk),
31    .rst_o(rst)
32  );
33
34  //scenariul 1 de btn, btn este 1 timp de 10 de ceasuri
35  btn #(CLK1) BTN_inst_1
36  (
37    .clk_i(clk),
38    .btn_o(btn1)
39  );
40  drv_btn DRV_BTN_inst_1(
41    .clk_i(clk),
42    .rst_i(rst),
43    .btn_i(btn1),
44
45    .srv_o(srv1)
46  );
47
48  //scenariul 2 de btn, btn este 1 timp de 20 de ceasuri
49  btn #(CLK2) BTN_inst_2
50  (
51    .clk_i(clk),
```

Fig 1.4.7

Scenariul cu service

```

51      .clk_i(clk),
52      .btn_o(btn2)
53  );
54  drv_btn DRV_BTN_inst_2(
55      .clk_i(clk),
56      .rst_i(rst),
57      .btn_i(btn2),
58
59      .srv_o(srv2)
60  );
61
62  //scenariul 3 de btn, btn este 1 timp de 5 de ceasuri
63  btn #(CLK3) BTN_inst_3
64  (
65      .clk_i(clk),
66      .btn_o(btn3)
67  );
68  drv_btn DRV_BTN_inst_3(
69      .clk_i(clk),
70      .rst_i(rst),
71      .btn_i(btn2),
72
73      .srv_o(srv3)
74  );
75
76  //instantiere modul contor pentru de o secunda si doua secunde la 16 ceasuri
77  ctu #(NrBiti1, Limita1) CTU_inst_2
78  (
79      .clk_i(clk),
80      .rst_i(rst),
81      .enable_i(enbl),
82      .count_up_i(count_up),
83
84      .data_o(dt),
85      .overflow_o.ovf_1sec,
86      .overflow2sec_o.ovf_2sec,
87      .check_ovf_o(chk)
88  );
89  scn_ctu #(Limita1) SCN_CCU_inst_2
90  (
91      .enable_o(enbl),
92      .count_up_o(count_up)
93  );
94
95  endmodule

```

Fig 1.7.2
Testbench (II)

Rezultate

În această secțiune vor fi introduse simulările și se vor explica semnalele prezente în diagrama. O mențiune importantă este că pentru o vizibilitate mai mare s-a folosit o frecvență mult mai mică pentru a ilustra semnalul de o secundă și de 2 secunde mai clar. Tot în aceeași simulare este introdusă și partea pentru driverul de buton.

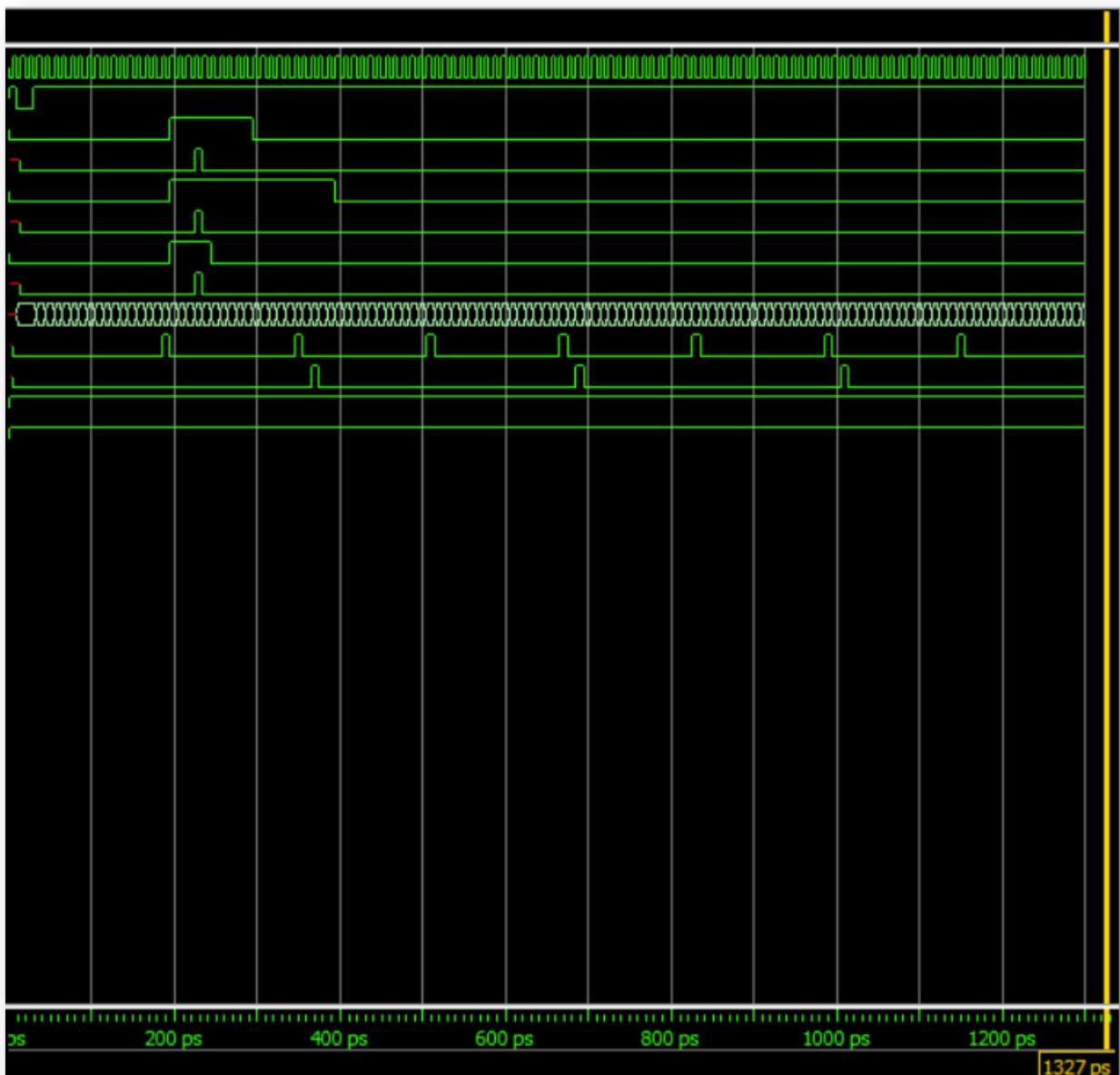
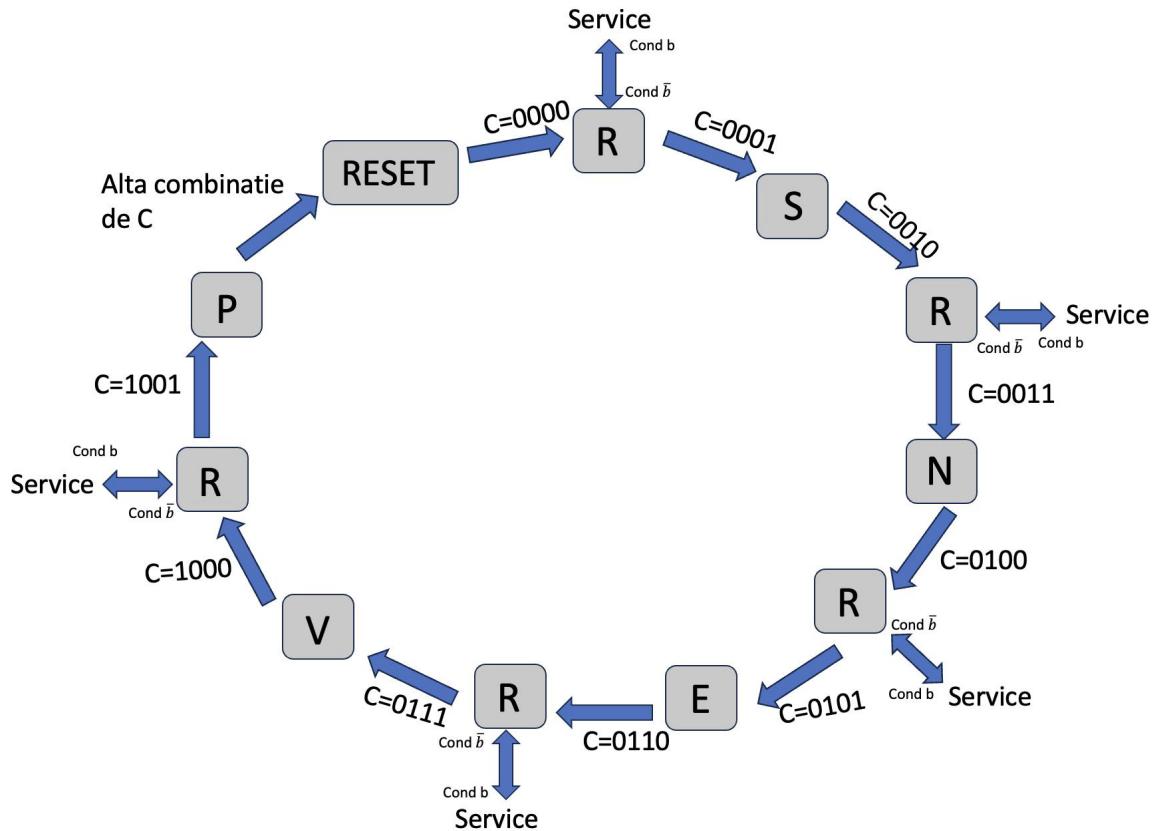


Fig 1.8.1
Diagrama de semnale

Semnalele vor fi trecute în odine, de sus în jos:

1. Semnalul de ceas/clock cu frecvență de 100MHz, s-a folosit o semiperioadă de 5 unități de timp, 5 ns, unitatea de timp default în Verilog este ns
2. Semnalul de reset, care trebuie configurat minim o dată pentru a atribui valori semnalelor care depind de el
3. Testul de buton unde apăsarea ține 10 ceasuri (apăsare medie)
4. Întârzierea semnalului de service – starea de service 1
5. Testul de buton unde apăsarea ține 20 ceasuri (apăsare lungă)
6. Întârzierea semnalului de service – starea de service 2
7. Testul de buton unde apăsarea ține 5 ceasuri(apăsare scurtă)
8. Întârzierea semnalului de service – starea de service 3
9. Semnalul contorizat care crește unitar cu fiecare front crescător de ceas
10. Semnalul de o secundă
11. Semnalul de 2 secunde
12. Semnalul de count-up, care permite incrementarea
13. Semnalul de enable, care permite utilizarea circuitului

MAS



$$\text{Cond } b = (0 \parallel 2 \parallel 4 \parallel 6 \parallel 8) \text{ \&\& BP}$$

$$\text{Cond } \bar{b} = (0 \parallel 2 \parallel 4 \parallel 6 \parallel 8) \text{ \&\& } \overline{\text{BP}}$$

Am decis ca in modelul de MAS sa facem urmatoarele:

- Controlul stării actuale
- Adaugarea stării de service

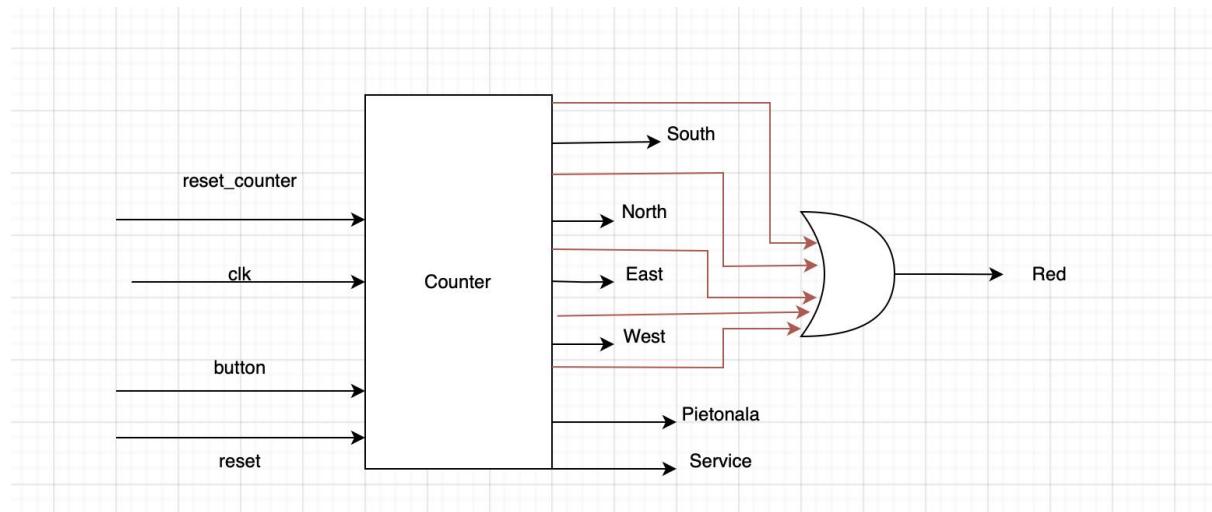
Un contor este utilizat pentru controlul stării actuale a unui semafor, având ca intrări un semnal de ceas (clk), un buton, și semnale de reset pentru contor și semafor. Starea actuală a semaforului este reprezentată de un număr între 0 și 15, care corespunde diferitelor stări ale semaforului: nord (N), sud (S), est (E), vest (V), pietoni (P), roșu, și starea de service.

Semaforul funcționează ciclic, trecând prin stările definite într-o secvență predeterminată, iar starea de reset asigură revenirea la starea inițială după un anumit număr de iterații sau în cazul unei situații specifice.

În proiectul nostru, vom simplifica numărul de stări ale semaforului eliminând starea galbenă, deoarece aceasta este implicit inclusă în stările de nord (N), sud (S), est (E) și vest (V), reducând astfel complexitatea proiectului.

Pentru funcționarea în modul de service, va fi necesară activarea butonului în timp ce semaforul se află în starea roșie. Această stare va fi menținută până când butonul nu mai este apăsat.

Pentru a asigura corectitudinea funcționării semaforului, vom acorda prioritate stării de service, iar celelalte stări vor fi considerate stări secundare, funcționând într-un mod secvențial de comutare.



Scenarii de functionare:

Functionare normală

```

Ln# 1 module MAS(
2     input wire reset_counter,
3     input wire button,
4     input wire reset,
5     input wire clk,
6     output reg [2:0] stimulus
7 );
8
9     reg [3:0] staresemafor;
10    reg [3:0] staresemaforvitor;
11    reg [3:0] staresemaforant;
12
13    always @ (posedge clk or negedge reset) begin
14        if (reset) begin staresemafor = 4'b0000;
15        staresemaforvitor = 4'b0000;
16        staresemaforant = 4'b0000;
17        end
18        else staresemafor <= staresemaforvitor;
19    end
20
21
22    always @ (*) begin
23        case (staresemafor)
24            4'b0000: if (button)
25                begin
26                    staresemaforvitor <= 4'b1111;
27                    staresemaforant <= staresemafor;
28                end
29                else if (reset_counter) staresemaforvitor <= 4'b0001;
30            4'b0001: if (reset_counter) staresemaforvitor <= 4'b0100;
31            4'b0100: if (button)
32                begin
33                    staresemaforvitor <= 4'b1111;
34                    staresemaforant <= staresemafor;
35                end
36        endcase
37    end
38
39    always @ (*) begin
40        case (staresemaforvitor)
41            4'b0001: if (reset_counter) staresemaforvitor <= 4'b0011;
42            4'b0011: if (reset_counter) staresemaforvitor <= 4'b0010;
43            4'b0010: if (button)
44                begin
45                    staresemaforvitor <= 4'b1111;
46                    staresemaforant <= staresemafor;
47                end
48                else if (reset_counter) staresemaforvitor <= 4'b0111;
49            4'b0111: if (reset_counter) staresemaforvitor <= 4'b0100;
50            4'b0100: if (button)
51                begin
52                    staresemaforvitor <= 4'b1111;
53                    staresemaforant <= staresemafor;
54                end
55                else if (reset_counter) staresemaforvitor <= 4'b1011;
56            4'b1011: if (reset_counter) staresemaforvitor <= 4'b1000;
57            4'b1000: if (button)
58                begin
59                    staresemaforvitor <= 4'b1111;
60                    staresemaforant <= staresemafor;
61                end
62        endcase
63    end
64
65    always @ (*) begin
66        case (staresemafor)
67            4'b0000: stimulus <= 3'b101; // red
68            4'b0001: stimulus <= 3'b000; // south
69            4'b0100: stimulus <= 3'b101; // red
70            4'b0101: stimulus <= 3'b000; // north
71            4'b0110: stimulus <= 3'b101; // east
72            4'b0111: stimulus <= 3'b001; // west
73            4'b1000: stimulus <= 3'b101; // red
74            4'b1001: stimulus <= 3'b111; // pietonal
75            4'b1011: stimulus <= 3'b111; // service
76            4'b1100: stimulus <= 3'b000; // default
77        endcase
78    end
79
80 endmodule

```

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.

1. Declarații de porturi:

- **reset_counter**, **button**, **reset**, și **clk** sunt intrări.
- **stimulus** este ieșirea, reprezentând starea semaforului sub formă de un vector de 3 biți.

2. Declarații interne:

- **staresemafor**, **staresemaforviitor**, și **staresemaforant** sunt registre interne utilizate pentru a urmări starea curentă a semaforului, starea viitoare și starea anterioară.

3. Actualizarea stării semaforului:

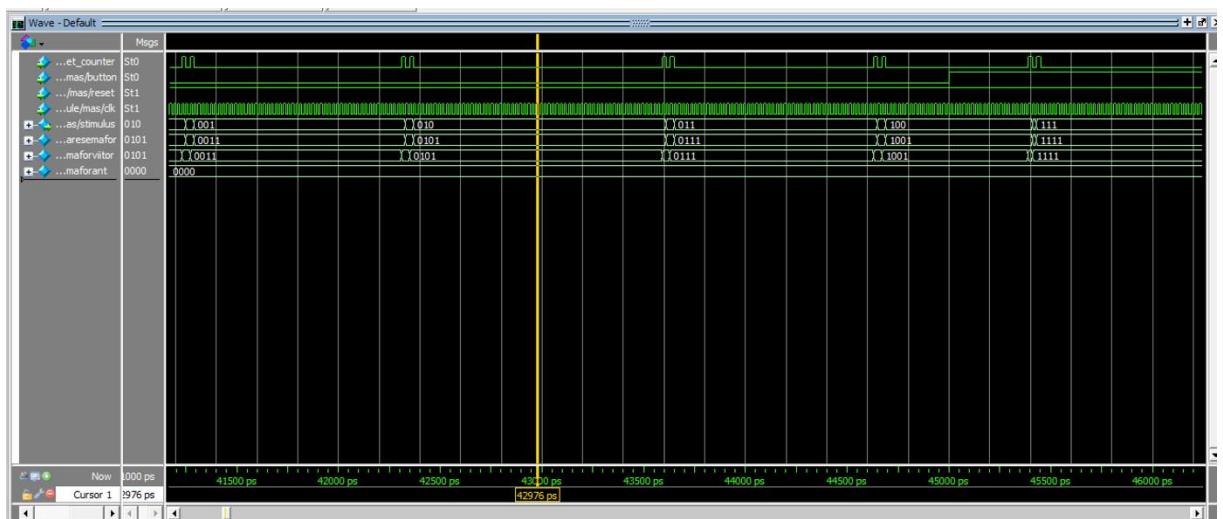
- Blocul **always** actualizează starea semaforului pe frontul ascendent al semnalului de ceas (**clk**) sau pe frontul descendente al semnalului **reset**.
- Dacă semnalul **reset** este activat, starea semaforului și cele două stări asociate (**staresemaforviitor** și **staresemaforant**) sunt resetate la **4'b0000**.

4. Determinarea stării viitoare a semaforului:

- Blocul **always** determină starea viitoare a semaforului în funcție de starea curentă și intrările primite (**button** și **reset_counter**).
- Se folosește o structură **case** pentru a determina starea viitoare a semaforului în funcție de starea curentă și de evenimentele survenite, cum ar fi apăsarea butonului sau trecerea unui anumit număr de cicluri de ceas.

5. Așezarea valorii de ieșire **stimulus**:

- Blocul **always** actualizează **stimulus** în funcție de starea semaforului.
- Fiecare stare a semaforului este asociată cu o configurație specifică a luminilor semaforului, reflectată în valorile asignate pentru **stimulus**.

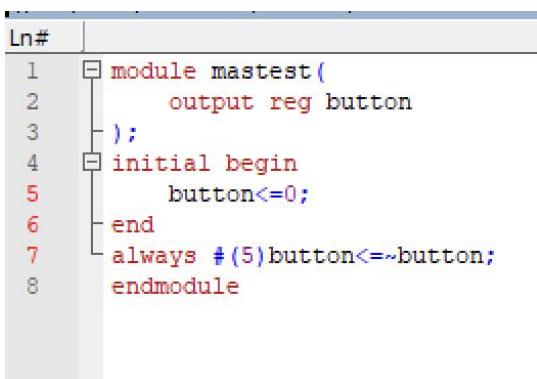


Figură.1.Rezultat simulare MA

Testare

```
1  module testmas()
2    wire reset_counter,
3    wire button,
4    wire reset,
5    wire clk,
6    reg [2:0] stimulus,
7    reg [2:0]stimulustest,
8    wire resettest,
9    wire reset_countertest
10   );
11   generator_clock #(10) gen
12   (
13     .clk_o(clk),
14     .rst_o (reset),
15     .enable_i(enable_i),
16     .count_up_i(count_up_i)
17   );
18   mastest buttonceva(
19     .button(button)
20   );
21   testmastest mastestmas(
22     .reset_counter(reset_countertest),
23     .reset(resettest)
24   );
25   MAS mas(
26     .clk(clk),
27     .reset(rst_o),
28     .reset_counter(reset_counter),
29     .stimulus(stimulus),
30     .button(button)
31   );
32   MAS mastest(
33     .clk(clk),
34     .reset(resettest),
35     .reset_counter(reset_countertest),
36     .stimulus(stimulustest),
37     .button(0)
38   );
39 endmodule
```

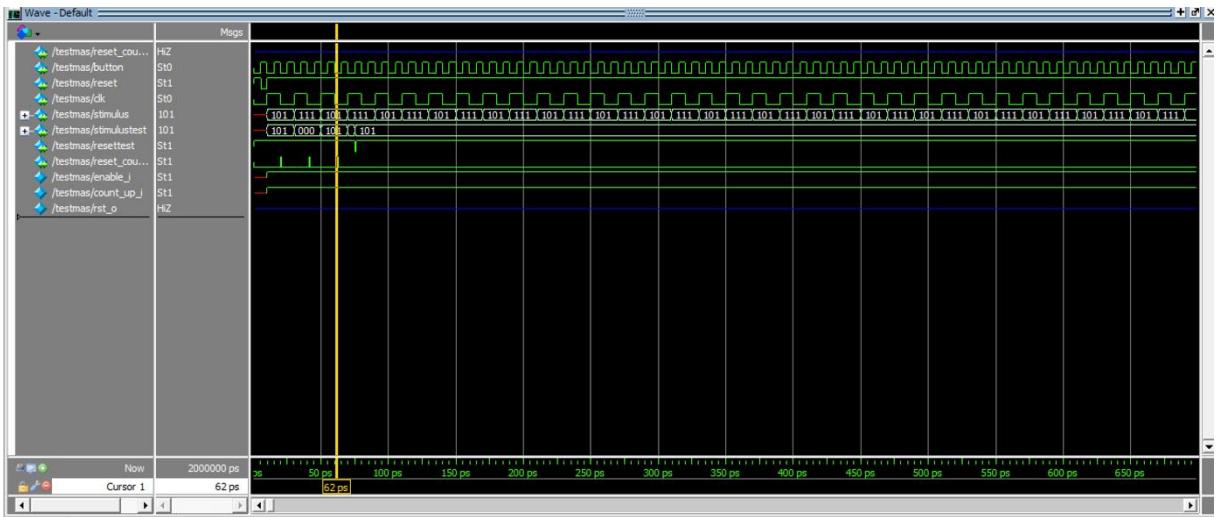
Modulul **testmas** integrează și testează funcționalitatea modulului **MAS** prin intermediul diferitelor instanțe ale acestuia și a sub-modulelor asociate, precum și prin generarea unui semnal de ceas și a unui semnal de buton utilizate în cadrul simulării sau testării.



The screenshot shows a code editor with a table header 'Ln#' and a single column of Verilog code. The code defines a module named 'mastest' with the following structure:

```
1  module mastest(
2    output reg button
3  );
4  initial begin
5    button<=0;
6  end
7  always #(5)button<=~button;
8 endmodule
```

Acest cod Verilog definește un modul numit **mastest**, care este un simulator de semnal de buton.



Figură.2. Rezultat simulare test MAS

Selectie display

La selecția display-ului ne-am înțeles să se facă următoarele:

- Alegerea semnalelor output în funcție de intrarea de la MAS;
- Calcularea fiecărui timp de output;
- Semnalul de ready pentru MAS;
- Afisarea timpului în secunde;

Alegerea semnalelor output în funcție de intrarea de la MAS se va realiza printr-un decodator 3 la 8, care va avea ca intrări starea actuală, clock-ul și semnalul reset și va scoate 7 ieșiri separate:

1. North - Semnalul în care se activează semaforul de nord;
2. South - Semnalul în care se activează semaforul de sud;
3. East - Semnalul în care se activează semaforul de est;
4. West - Semnalul în care se activează semaforul de vest;
5. Pietonal - Semnalul în care se activează semaforul de pietoni;
6. Red - Semnalul în care se activează semaforul de roșu (cel de 1 secundă);
7. Service - Semnalul în care se activează semaforul de service;

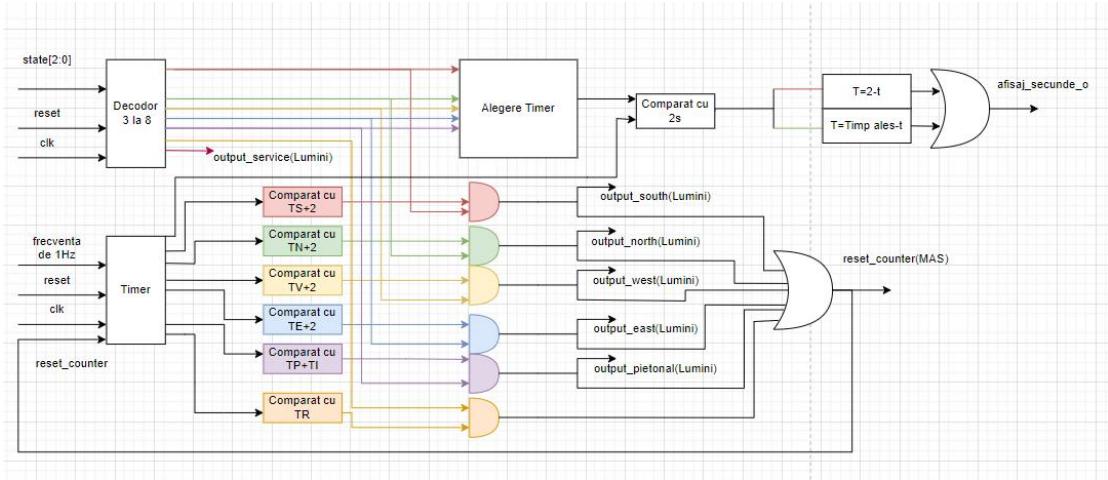
Primele 6 ieșiri vor fi comparate cu un timer, care va arăta cât de mult trebuie afișate, în timp ce starea de service este nedeterminată.

Pentru a calcula fiecare timp de output, am creat un timer universal, care are ca intrări un semnal cu o frecvență de 1 Hz, resetul, clock-ul și un semnal de reset_counter. Acest timer va compara în 6 instanțe diferite timpul: North, South, East, West, Pietonal și Red. Resetarea timer-ului (reset_counter) se face în momentul în care comparația outputului selectat devine falsă (timpul nostru actual este mai mare decât timpul ales +2).

Am ales valoarea TS/N/V/E +2 pentru a adăuga timpul necesar celor 2 secunde de lumină galbenă, care vine imediat înainte de valorile de verde, simplificând astfel schema, eliminând stările neimportante. Același lucru putem spune și despre valoarea TP+TI, reprezentând valorile timpului de verde simplu și verde intermitent.

Semnalul de ready pentru MAS este obținut din valoarea de reset a timer-ului (reset_counter), deoarece în momentul în care acesta se activează, știm că timpul necesar pentru galben și verde a trecut.

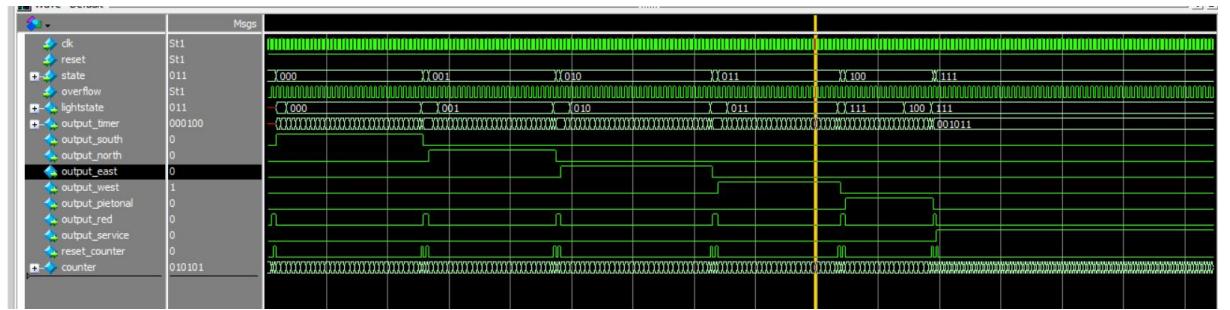
Afișarea timpului în secunde se face printr-un block numit "Alegere timer", care are ca intrare una dintre valorile de la decodator și, în funcție de aceasta, alege ce timer să folosească în continuare. Timer-ul ales va include mai întâi cele 2 secunde de galben, iar apoi timpul necesar pentru fiecare variantă de verde (cu 2 secunde în plus).



Figură 7.1. Schema funcțională a selecției display

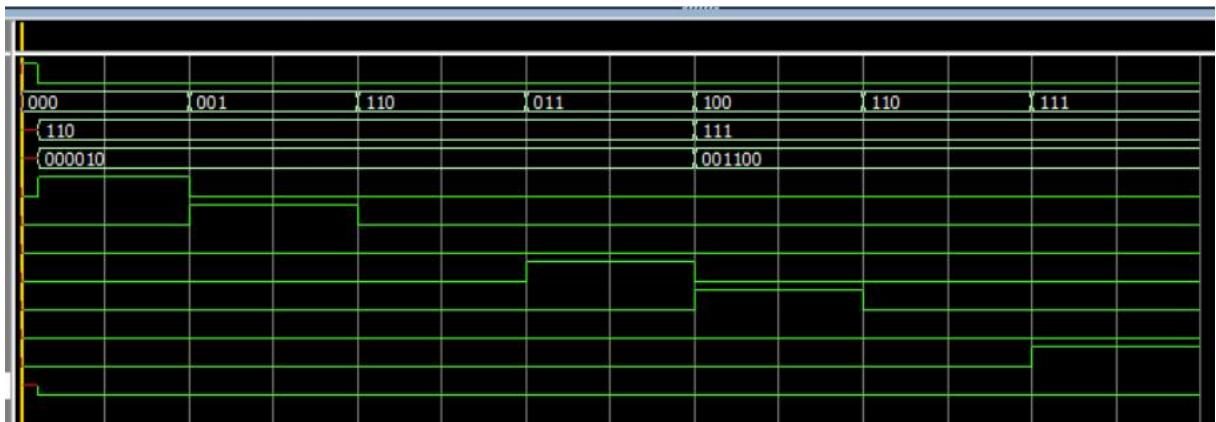
Scenarii de funcționare:

Funcționare simplă:



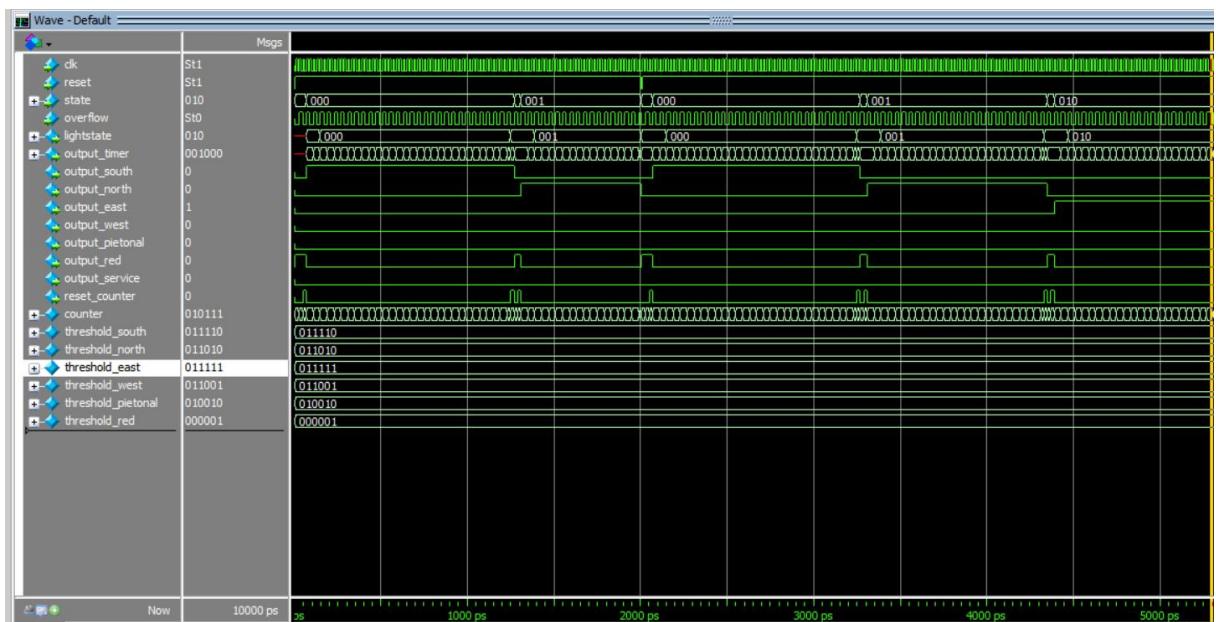
Remarcam functionarea corectă a proiectului

Functionare cand una din etape nu functioneaza:



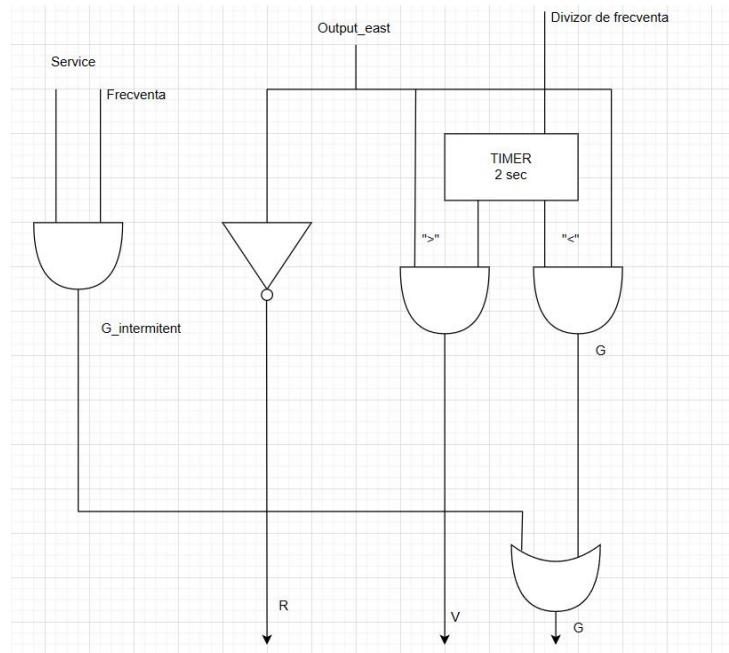
Remarcam functionarea selectiei nu depinde de "cursivitatea" semnalelor acesta putand fi aprinse in orice ordinea , cate 1

Functia de Reset



Remarcam functionarea reset în momentul 2000ps în care incepe complet de la 0

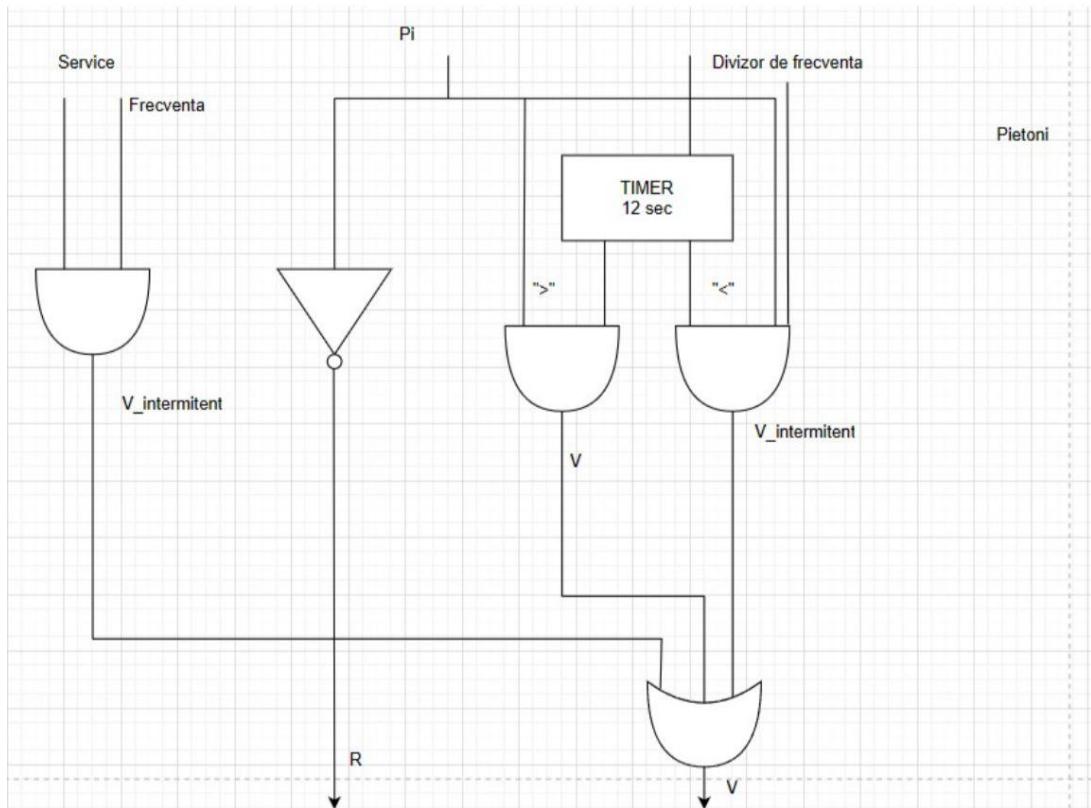
Modul control lumini (#0; #1; #2; #3; #4)



Figură 2. Schema modul de lumini pentru autoturisme

În schema de mai sus am reprezentat funcționarea modulului de lumini pentru autoturisme. Acesta are 3 ieșiri: roșu, galben și verde. Cu ajutorul unui timer vom număra câte secunde trec de când semaforul este în starea curentă. Dacă acesta la ieșire numără mai puțin de 2 secunde, atunci semaforul va arăta culoarea galben. În caz contrar, dacă acesta numără mai mult de 2 secunde, se va aprinde verde. Pe lângă acestea, se observă și intrarea pentru modul de service, în care semaforul va să afișeze galben intermitent. În cazul în care nu se afișează nici verde și nici

galben, semaforul se va seta pe roșu.



Figură 3. Schema modul de lumini pentru pietoni

În schema de mai sus am reprezentat funcționarea modulului de lumini pentru pietoni. Acesta are 2 ieșiri: roșu și verde. Cu ajutorul unui timer vom număra câte secunde trec de când semaforul este în starea curentă. Dacă acesta la ieșire numără mai puțin de 12 secunde, atunci semaforul va arăta culoarea verde intermitent. În caz contrar, dacă acesta numără mai mult de 2 secunde, se va aprinde verde.

Pe lângă acestea, se observă și intrarea pentru modul de service, în care semaforul va să afișeze verde intermitent. În cazul în care nu se afișează nici verde și nici galben, semaforul se va seta pe roșu.

8.1.Cod

TopModule.v (reprezintă modulul de unde simulăm submodulele, ceea ce face legătura între ele)

```

e TopModule.v |]
module TopModule (
    input wire clk,
    input sec,
    input sec2,
    input enable_i,
    input rst_o,
    input count_up_i,
    input [2:0] stimulus,
    input [2:0] lightstate,
    input [2:0] current_state,
    input reset_counter,
    input reg [5:0] output_timer,
    input reg output_south,
    input reg output_north,
    input reg output_east,
    input reg output_west,
    input reg output_pietonal,
    input reg output_red,
    input reg output_service,
    input reg button,
    input reg Rosu_auto_S_o,
    input reg Galben_auto_S_o,
    input reg Verde_auto_S_o
);
generator_clock#(10) gen
(
    .clk_o(clk),
    .rst_o(rst_o),
    .enable_i(enable_i),
    .count_up_i(count_up_i)
);

|SelectieDispaly selectiedisplay (
    .clk(cik),
    .state(stimulus),
    .overflow(sec),
    .reset(rst_o),
    .current_state(current_state),
    .lightstate(lightstate),
    .reset_counter(reset_counter),
    .output_south(output_south),
    .output_north(output_north),
    .output_east(output_east),
    .output_west(output_west),
    .output_pietonal(output_pietonal),
    .output_red(output_red),
    .output_timer(output_timer),
    .output_service(output_service)
);
|SouthLights south(
    .clk(cik),
    .rst(rst_o),
    .service(output_service),
    .overflow(sec),
    .overflow2sec(sec2),
    .output_south(output_south),
    .Rosu_auto_S_o(Rosu_auto_S_o),
    .Galben_auto_S_o(Galben_auto_S_o),
    .Verde_auto_S_o(Verde_auto_S_o)
);
|NorthLights north(
    .clk(cik),
    .rst(rst_o),
    .service(output_service),
    .overflow(sec),
    .overflow2sec(sec2),
    .output_north(output_north),
    .Rosu_auto_N_o(Rosu_auto_N_o),
    .Galben_auto_N_o(Galben_auto_N_o),
    .Verde_auto_N_o(Verde_auto_N_o)
);
|EastLights east(
    .clk(cik),
    .rst(rst_o),
    .service(output_service),
    .overflow(sec),
    .overflow2sec(sec2),
    .output_east(output_east),
    .Rosu_auto_E_o(Rosu_auto_E_o),
    .Galben_auto_E_o(Galben_auto_E_o),
    .Verde_auto_E_o(Verde_auto_E_o)
);
|WestLights west(
    .clk(cik),
    .rst(rst_o),
    .service(output_service),
    .overflow(sec),
    .overflow2sec(sec2),
    .output_west(output_west),
    .Rosu_auto_W_o(Rosu_auto_W_o),
    .Galben_auto_W_o(Galben_auto_W_o),
    .Verde_auto_W_o(Verde_auto_W_o)
);
|PietonalLights pietonal(
    .clk(cik),
    .rst(rst_o),
    .service(output_service),
    .overflow(sec),
    .overflow2sec(sec2),
    .output_pietonal(output_pietonal),
    .Rosu_auto_P_o(Rosu_auto_P_o),
    .Galben_auto_P_o(Galben_auto_P_o),
    .Verde_auto_P_o(Verde_auto_P_o)
);
endmodule

```

Conform codului de mai sus se observă că, controlul semafoarelor din sud se face cu ajutorul submodulului [SouthLights south](#).

Astfel remarcăm porturile:

- Clk: semnal de ceas
- Rst: semnal de resetare
- Overflow: semnal de overflow
- Overflow2sec: semnalul de overflow pentru două secunde
- Service: semnal de service

Ieșiri pentru controlul semafoarelor:

- Output_south (pentru modulul de sud)

- Rosu_auto_S_o
- Galben_auto_S_o
- Verde_auto_S_o
-

Similar și pentru celelalte submodule.

În urmatoarele figuri este reprezentat codul pentru modulul de control pentru sud.

În celelalte cazuri: nord,sud,est,vest si pietonal este asemanător.

```
module SouthLights(
    input clk,
    input rst,
    input output_south,
    input service,
    input overflow,
    input overflow2sec,
    output reg Rosu_auto_S_o,
    output reg Galben_auto_S_o,
    output reg Verde_auto_S_o
);
reg [6:0] count;
reg IntermittentG ;
reg SimpluG ;
always @(posedge overflow)
count <= count + 1;

always @(*) begin

    if(!rst)count<=0;
    else if (service) begin
        count<=0;
        if(overflow2sec) begin
            Rosu_auto_S_o <= 0;
            IntermittentG <= 1;
            Verde_auto_S_o <= 0;
        end
        else begin
            Rosu_auto_S_o <= 0;
            IntermittentG <= 0;
            Verde_auto_S_o <= 0;
        end
    end
    else
        IntermittentG <= 0;
        if (output_south) begin
            if (count <= 1) begin
                Rosu_auto_S_o <= 0;
                SimpluG <= 1;
                Verde_auto_S_o <= 0;
            end
            else begin
                Rosu_auto_S_o <= 0;
                SimpluG <= 0;
                Verde_auto_S_o <= 1;
            end
        end
    end
end

else begin
    Rosu_auto_S_o <= 0;
    SimpluG <= 0;
    Verde_auto_S_o <= 1;
end
end else begin
    Rosu_auto_S_o <= 1;
    SimpluG <= 0;
    Verde_auto_S_o <= 0;
    count <= 0;
end
end

always @(*) begin
    if(IntermittentG|SimpluG)
        Galben_auto_S_o=1;
    else Galben_auto_S_o=0;
end
endmodule
```

Porturi ale modulului `SouthLights`

-Intrări:

- `clk`: Semnal de ceas.
- `rst`: Semnal de resetare.
- `output_south`: Semnal pentru activarea semaforului din sud.
- `service`: Semnal de serviciu (pentru mod de întreținere).
- `overflow`: Semnal de overflow care indică trecerea unui anumit timp (o secundă).

- `overflow2sec`: Semnal de overflow care indică trecerea a două secunde.

Ieșiri:

- `Rosu_auto_S_o`: ieșire pentru semnalul roșu.
- `Galben_auto_S_o`: ieșire pentru semnalul galben.
- `Verde_auto_S_o`: ieșire pentru semnalul verde.

Registre și variabile interne:

- `count`: Registru pentru numărarea timpului.
- `IntermitentG`: Variabilă pentru starea intermitentă a galbenului.
- `SimpluG`: Variabilă pentru starea simplă a galbenului.

Logica de incrementare a contorului:

La fiecare semnal `posedge overflow` (la fiecare overflow), `count` este incrementat cu 1:

Logica principală:

Blocul `always @(*)` descrie logica de control a semaforului în funcție de diverse condiții

- Dacă `rst` este activ (0), contorul `count` este resetat la 0.
- Dacă `service` este activ, contorul `count` este resetat la 0 și se verifică semnalul `overflow2sec`:
 - Dacă `overflow2sec` este activ, semaforul roșu este oprit, galbenul intermitent este activat și verdele este oprit.
 - În caz contrar, semaforul roșu este oprit, galbenul intermitent este oprit și verdele este oprit.
 - Dacă `output_south` este activ:
 - Dacă contorul `count` este mai mic sau egal cu 1, semaforul roșu este oprit, galbenul simplu este activat și verdele este oprit.
 - În caz contrar, semaforul roșu este oprit, galbenul simplu este oprit și verdele este activat.
 - Dacă `output_south` este oprit, semaforul roșu este activat, galbenul simplu este oprit, verdele este oprit și contorul `count` este resetat la 0.

Logica pentru semnalul galben:

Un alt bloc `always @(*)` controlează starea semnalului galben în funcție de variabilele `IntermitentG` și `SimpluG`:

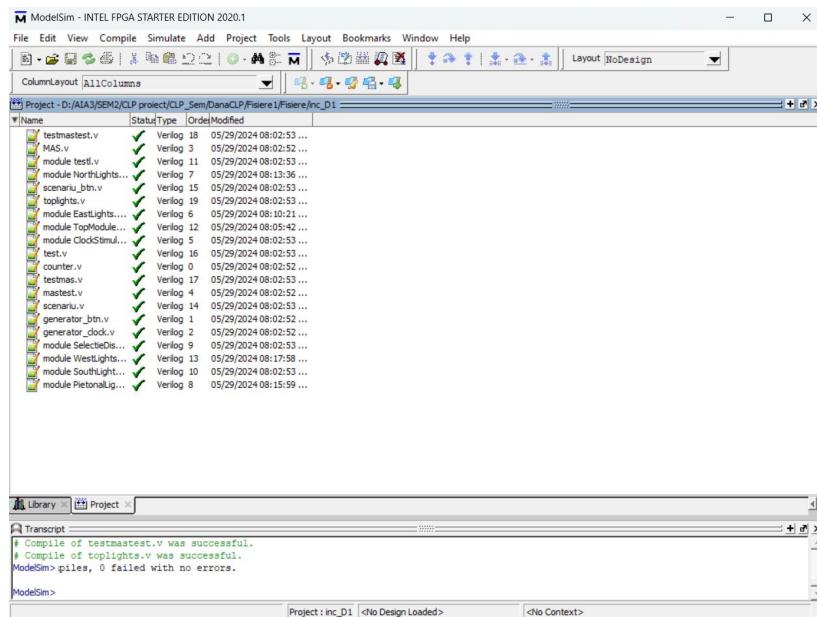
- Dacă `IntermitentG` sau `SimpluG` este activ, semaforul galben este activat.
- În caz contrar, semaforul galben este oprit.

8.2.Scenariul de simulare

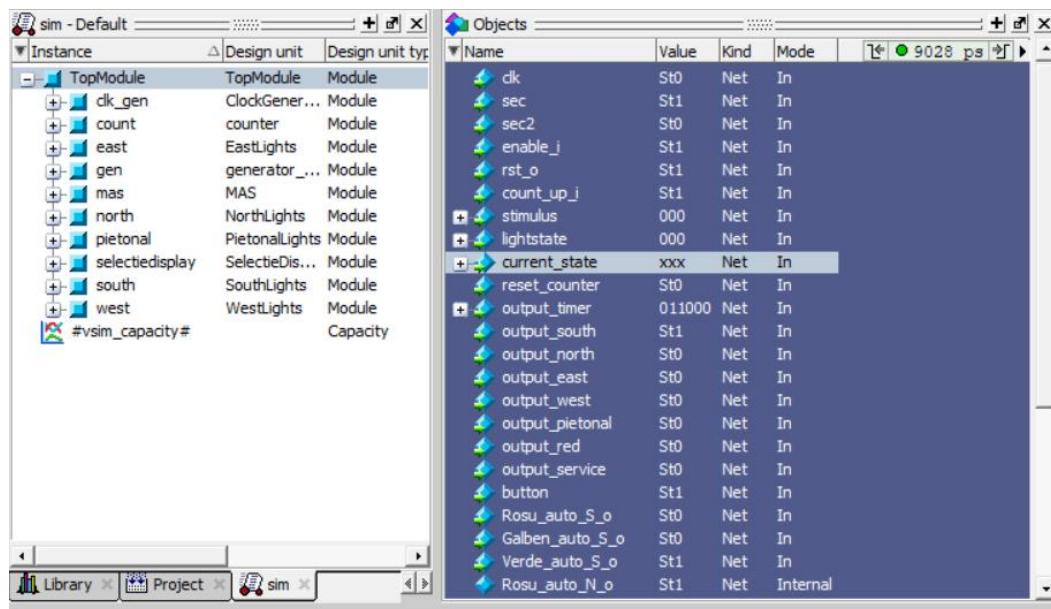
Pentru a asigura funcționalitatea acestui modul, avem nevoie să testăm cu ajutorul unor scenarii de simulare.

8.2.1.Funcționalitatea aplicației

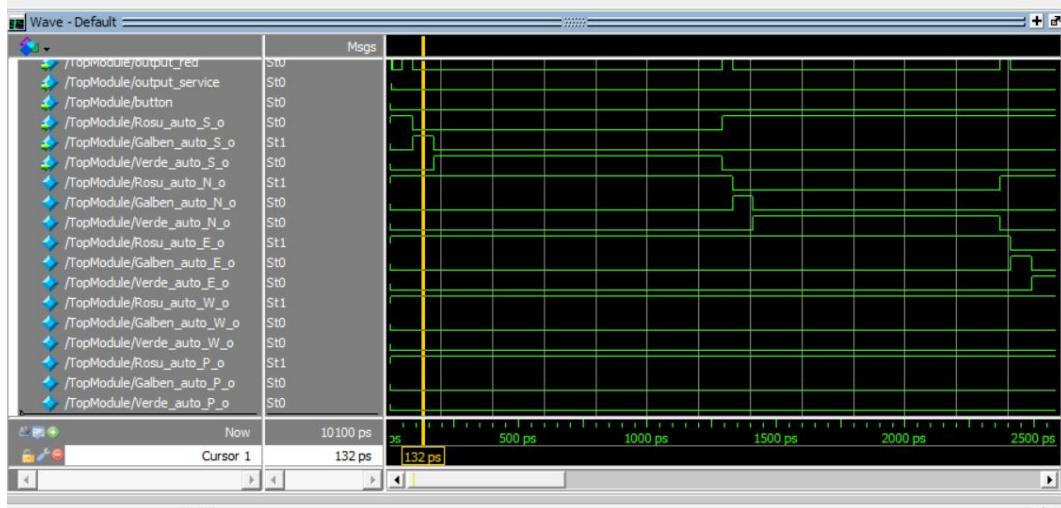
În momentul rulării aplicației, observăm că aplicația funcționează conform cerințelor de proiectare. Acest lucru este exemplificat în următoarele figuri:



Figură 4.Compilare module

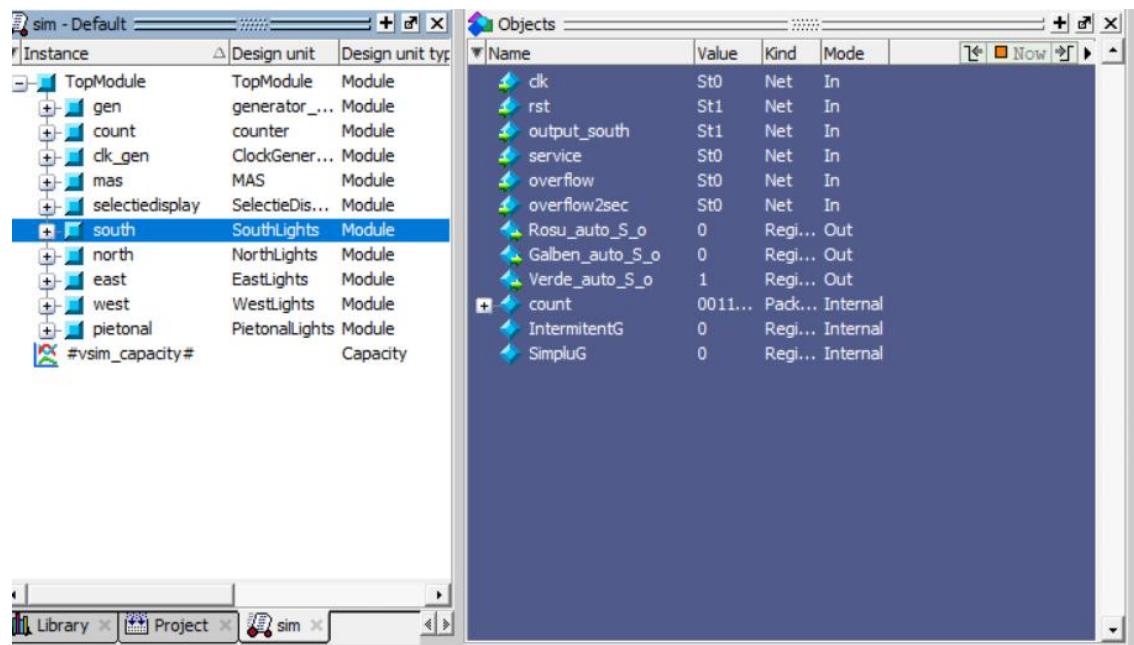


Figură 5.Simulare TopModule



Figură 6.Resultat simulare TopModule

Pentru a exemplifica funcționarea unui singur modul de lumini am ales modulul pentru sud, astfel:



Figură 7.Simulare un singur modul separat



Figură 8.Resultat simulare modul pentru luminile din sud

8.1.2. Apăsare reset

| Name | Status | Type | Order | Modified |
|----------------------|--------|---------|-------|-------------------------|
| toplights.v | ✓ | Verilog | 4 | 05/29/2024 04:16:16 ... |
| module testl.v | ✓ | Verilog | 2 | 05/29/2024 04:16:16 ... |
| generator_clock.v | ✓ | Verilog | 0 | 05/29/2024 04:16:16 ... |
| test.v | ✓ | Verilog | 3 | 05/29/2024 04:16:16 ... |
| module SouthLight... | ✓ | Verilog | 1 | 05/29/2024 04:19:17 ... |

- Blocul „**initial**” este folosit pentru a seta valorile inițiale ale semnalelor și pentru a defini comportamentul lor în timp.
- Două blocuri „**always**” sunt folosite pentru a genera semnalele de overflow cu perioade diferite.
- Generare overflow2sec_o: La fiecare 100 unități de timp, semnalul overflow2sec_o își schimbă starea (comută între 0 și 1).
- Generare overflow_o: La fiecare 25 unități de timp, semnalul overflow_o își schimbă starea.

| Module | Design Unit | Design Unit Type | Top Category |
|-----------------|---------------|------------------|--------------|
| toplights | toplights | Module | DU Instance |
| gen | generator_... | Module | DU Instance |
| testl | testl | Module | DU Instance |
| testli | testli | Module | DU Instance |
| south | SouthLights | Module | DU Instance |
| southl | SouthLights | Module | DU Instance |
| #vsim_capacity# | Capacity | Statistics | |

Figură 9. Simulare reset

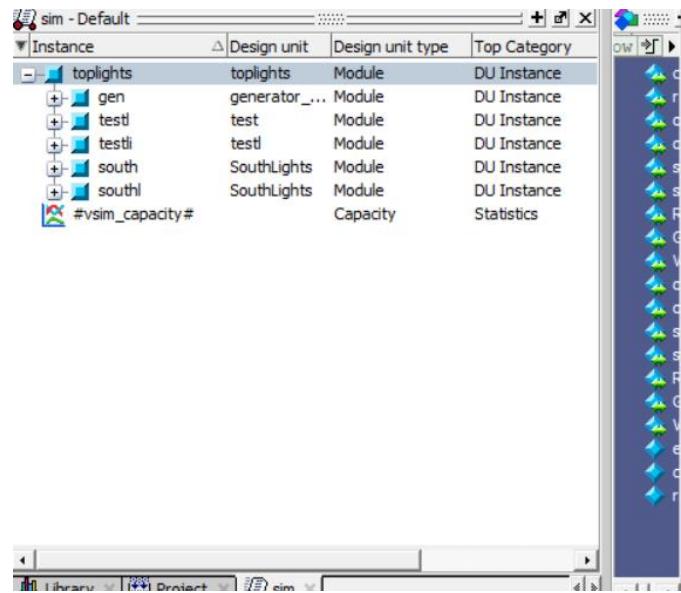


Figură 10. Rezultat pentru reset

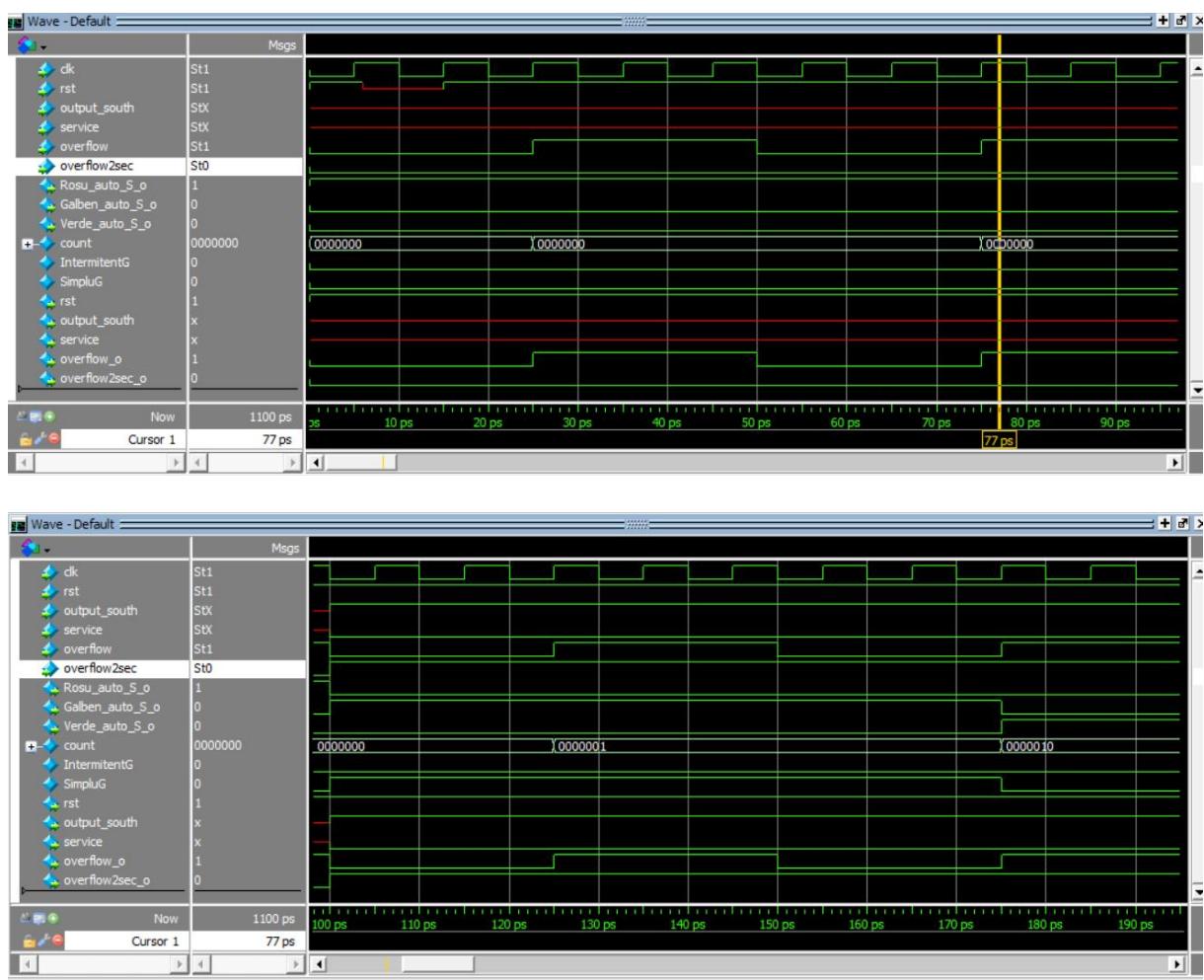
8.1.3. Funcționarea independentă

```
module toplights(
    wire clk,
    wire rst_o,
    wire output_south,
    wire output_service,
    wire sec,
    wire sec2,
    wire Rosu_auto_S_o,
    wire Galben_auto_S_o,
    wire Verde_auto_S_o,
    wire output_southl,
    wire output_servicel,
    wire sec1,
    wire sec21,
    wire Rosu_auto_S_ol,
    wire Galben_auto_S_ol,
    wire Verde_auto_S_ol
);
    SouthLights south(
        .clk(clk),
        .rst(rst_o),
        .service(output_service),
        .overflow(sec),
        .overflow2sec(sec2),
        .output_south(output_south),
        .Rosu_auto_S_o(Rosu_auto_S_o),
        .Galben_auto_S_o(Galben_auto_S_o),
        .Verde_auto_S_o(Verde_auto_S_o)
    );
    generator_clock#(5) gen
    (
        .clk_o(clk),
        .rst_o(rst_o),
        .enable_i(enable_i),
        .count_up_i(count_up_i)
    );
    test test1(
        .rst(rst_o),
        .output_south(output_south),
        .service(output_service),
        .overflow2sec_o(sec2),
        .overflow_o(sec)
    );
    test test1l(
        .rst(rst_ol),
        .output_south(output_southl),
        .service(output_servicel),
        .overflow2sec_o(sec21),
        .overflow_o(sec1)
    );
endmodule
```

- Inițializare semnale: Setează overflow_o și overflow2sec_o la 0, iar rst la 1(activ).
- După 100 unități de timp: Setează output_south la 1(activ) și service la 0 (dezactivat).
- După alte 100 unități de timp: Setează rst la 0(dezactivare reset).
- După 1 unitate de timp: Setează rst la 1(activare reset).
- După alte 500 unități de timp: Setează output_south la 0(dezactivat) și service la 1 (activ).



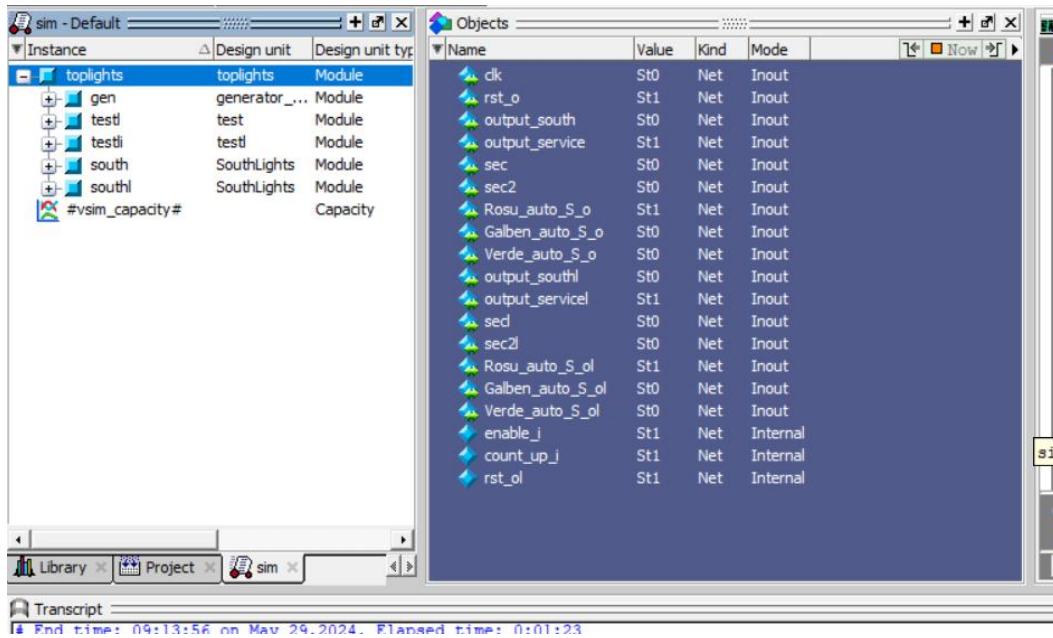
Figură 11.Verificare funcționare independentă



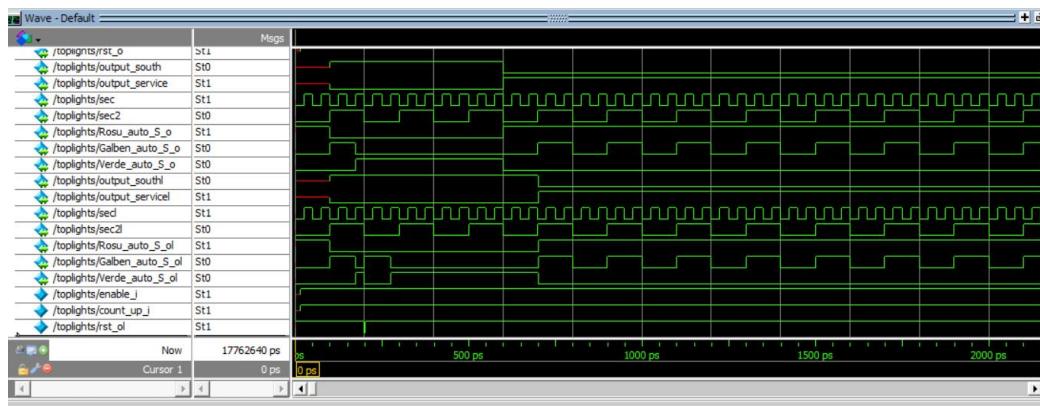
Figură 12.Simulare verificare inițială

8.1.4. Apăsare continuă a butonului de service

Am ales și această abordare pentru a vedea cum reționează luminile la apăsarea continuă a butonului de service.



Figură 13. Apăsarea continuă a butonului de service



Figură 14. Simulare apăsare continuă a butonului de service