1. Cerința

Analiza, proiectare și implementarea unui divizor de frecvență, precum și cea a unui driver de buton, cu aplicare într-o intersecție semaforizată.

1.1. Proiectare

1.2. Divizor de frecvență

Pentru a se putea folosi orice circuit digital care urmărește să intereacționeze 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 circutului 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 periodă 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.

1.3. Driver de buton

Driverul de buton este folosit pentru a trece tot circuitul într-o stare de service. Acest aspect a fost simulat 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 apasărea acestuia nu poate fi făcută în codiții 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 sune legate la o poartă ȘI, dar ieșirea celui de-al doilea bistabil trebuie negată, se folosește ieșirea sa negată. Astfel sunt evitate zonele unde se produc spike-uri mecanice.

1.4. 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.

1.5. Divizor de frecvență

```
module ctu #(parameter NrBiti = 27, parameter Limita = 'h3B9AC9FF)
//2^27 - 1 = 'h7FFFFF (maximuml posibil cu 27 de biti)
 //Limita = 10^8 - 1 = 'h3B9AC9FF
//10^8 = h5F5E100
(
input clk_i,
input rst_i,
input enable_i,
input count_up_i,

output reg [NrBiti-1:0] data_o,
output reg overflow_o,
output reg overflow2sec_o,
output reg [1:0] check_ovf_o
);

always @(posedge clk_i or negedge rst_i) //data_o
begin
if (!rst_i) data_o <= 'b0; //cand rst_i = 0, data_o se reseteaza
else if (!enable_i) data_o <= data_o;
else if (count_up_i) data_o <= data_o + 1;
else data_o <= data_o;//daca nu se declara explicit, compilatorul o va face automat in locul nostru
end
```

Fig 1.5.1
Counter (I)

```

always @(posedge clk_i or negedge rst_i) //overflow_o
begin
if (!rst_i) overflow_o = 'b0;//cazul rst_i = 0, overflow_o se reseteaza
else if ( (data_o == Limita) & (enable_i & count_up_i) )
begin
overflow_o <= 'b1;
data_o <= 'b0;
end
else overflow_o <= 'b0;//isi perpetueaza valoarea daca overflow_o <= overflow_o;
end

always @(posedge clk_i or negedge rst_i) //overflow2sec_o
begin
if (!rst_i) overflow2sec_o <= 'b0;
else if (check_ovf_o == 'b10) overflow2sec_o <= 'b1;
else overflow2sec_o <= 'b0;
end

always @(posedge clk_i or negedge rst_i) //check_ovf_o
begin
if(!rst_i) check_ovf_o <= 'b0;
else if(overflow_o) check_ovf_o <= check_ovf_o + 'b1;
else if(check_ovf_o == 'b10) check_ovf_o <= 'b0;
else check_ovf_o <= check_ovf_o;
end

endmodule
```

Fig 1.5.2
Counter (II)

Scenariul de counter unde sunt inițializate semnalele de intrare.

```
module scn_ctu #(parameter Limita = 'b10000)//la 16 fronturi vine un reset
(
output reg enable_o,
output reg count_up_o
);

initial
begin
enable_o <= 'b1;
count_up_o <= 'b1;
end

endmodule
```

Fig 1.5.3
Scenariul de counter

1.6. 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 atașa în ordine modulele care formează driverul de buton.

```
module and2
(
input in1_i,
input in2_i,

output reg out1_o
);

assign out1_o = in1_i & in2_i;

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

1.7. Testbench

Testbench-ul este unealta 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. Acesta 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.

```
module tb();

localparam SP = 5; //semiperioada ceasului
localparam CLK1 = 10; //apasare normala = 10 ceasuri
localparam CLK2 = 20; //apasare lunga = 20 ceasuri
localparam CLK3 = 5; //apasare scurta = 5 ceasuri
localparam NrBiti = 27;
localparam NrBiti1 = 4;
localparam Limita = 'h3B9AC9FF;//10^8
localparam Limita1 = 'b1111;//15 in decimal - 16 unitati

wire btn1;
wire btn2;
wire btn3;
wire clk;
wire rst;
wire srv1;
wire srv2;
wire srv3;
wire enbl;
wire count_up;
wire [NrBiti1-1:0] dt;
wire ovf_1sec;
wire ovf_2sec;
wire [1:0] chk;

//ceasul de 100MHZ
clock #(SP) CLOCK_inst_1
(
.clk_o(clk),
.rst_o(rst)
);

//scenariul 1 de btn, btn este 1 timp de 10 de ceasuri
btn #(CLK1) BTN_inst_1
(
.clk_i(clk),
.btn_o(btn1)
);
drv_btn DRV_BTN_inst_1(
.clk_i(clk),
.rst_i(rst),
.btn_i(btn1),

.srv_o(srv1)
);

//scenariul 2 de btn, btn este 1 timp de 20 de ceasuri
btn #(CLK2) BTN_inst_2
(
.clk_i(clk),
```

Fig 1.7.1
Testbench (I)

```
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_CTU_inst_2
90      (
91      .enable_o(enbl),
92      .count_up_o(count_up)
93      );
94
95      endmodule
```

Fig 1.7.2
Testbench (II)

1.8. 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.

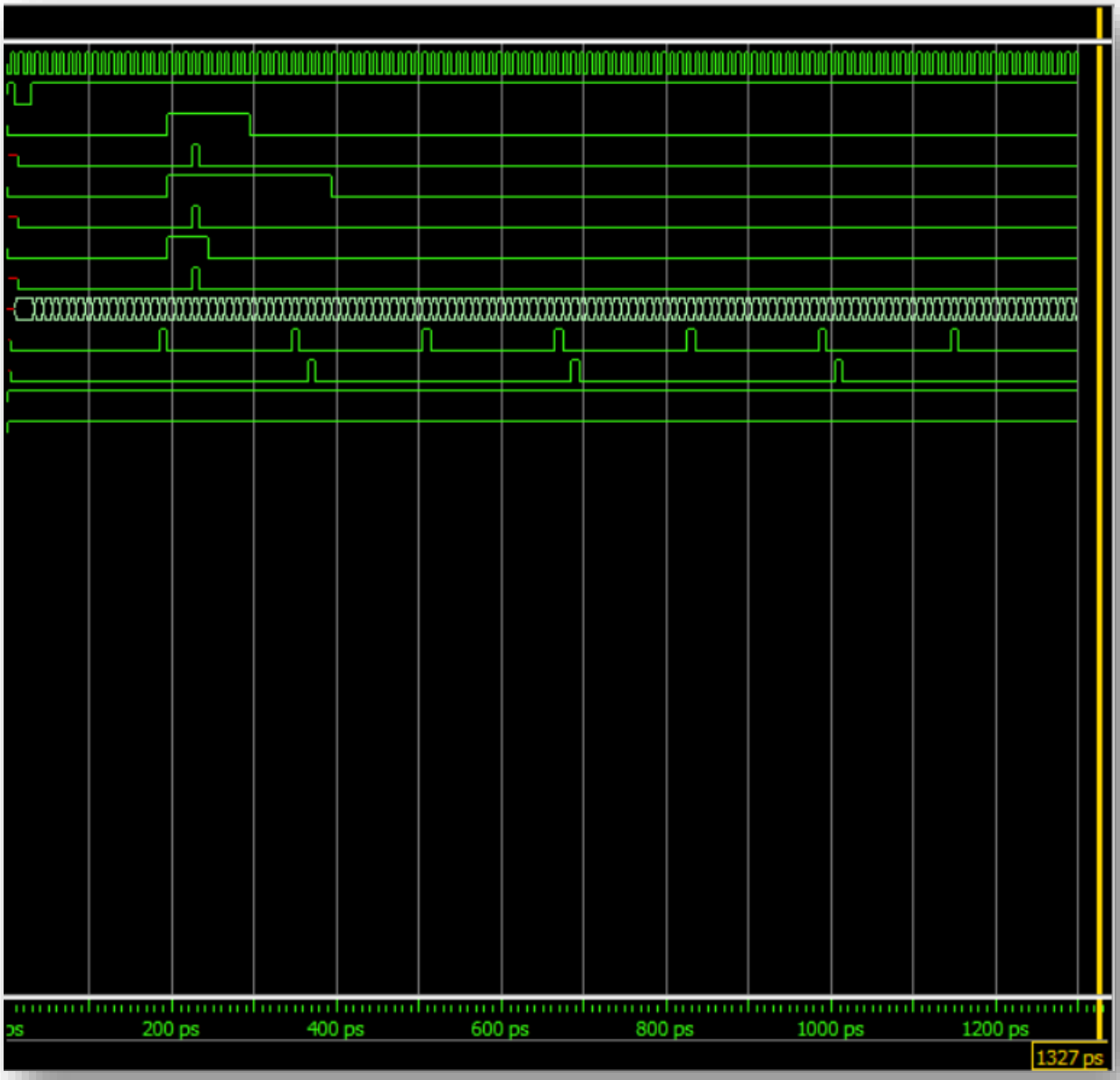

Fig 1.8.1
Diagrama de semnale

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

1. Semnalul de ceas/clock cu frecvența 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