UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CONCEPÇÃO ESTRUTURADA DE CIRCUITOS INTEGRADOS

PROJETO MIPS – ENTREGA 1

THIAGO ALVES DE ARAUJO

2016019787

Sumário

1 Tabela de figuras

2 Floopr

Abaixo podemos ver o diagrama de blocos de um Flop com reset assincrono. É valido notar que a imagem ilustra um floopr com 4 bits de entrada/saída de dados, porém o modulo que vai ser criado aqui é de apenas 1 bit de entrada/saída de dados.

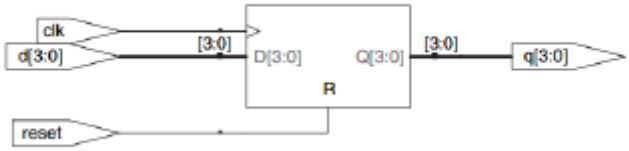

Figura 1 Flopr - Diagrama de blocos

2.1 Goldem model

Abaixo segue o trecho do código utilizado para gerar os *goldem vectors*

```
//Determina a saída de acordo com as entradas
for(int i = 0; i < tamanho; i++){
    //Se reset = 1, y = 0
    if(tabelaVerdade[i][1] == 1){
        tabelaVerdade[i][3] = 0;
    }
    //Se clk = 0, y = estado anterior
    if(tabelaVerdade[i][0] == 0){
        tabelaVerdade[i][3] = tabelaVerdade[i-1][3];
    }
    //Se clk = 1 e reset = 0, y = d;
    if(tabelaVerdade[i][0] == 1 && tabelaVerdade[i][1] == 0){
        tabelaVerdade[i][3] = tabelaVerdade[i][2];
    }
}
```

Figura 2 Flopr - Goldem Model

2.2 Goldem vector

A partir do *goldem model* criado anteriormente, geramos o seguinte vetor de testes. Da esquerda para a direita temos clock, reset, dado e saída. Cada entrada/saída possui apenas um bit de tamanho.

```
0_1_0_0
1_1_0_0
0_1_1_0
1_1_1_0
0_0_1_0
1_0_1_1
0_0_0_1
1_0_0_0
```

Figura 3 Flopr - Goldem vectors

2.3 Testbench

Abaixo podemos ver o código em SystemVerilog responsável por testar o modulo flopr. Ao iniciar damos um pulso no reset para que a saída saia do estado indeterminado “X”. Para o flopr, o tempo mínimo de clock encontrado necessário para o modulo funcionar perfeitamente foi de 5000ps.

```
1   `timescale 1ns/100ps
2   module flopr_tb;
3
4   int counter, errors, aux_error;
5   logic clk,rst;
6   logic clk2,rst2;
7   logic d;
8   logic q, q_esperado;
9   logic [3:0]vectors[8];
10  flopr dut(clk2, rst2, d, q);
11
12  initial begin
13     $display("Iniciando Testbench");
14     $display("| CLK | RST | D | Q |");
15     $display("---------------");
16     $readmemb("flopr_tv.tv",vectors);
17     counter=0; errors=0;
18     rst = 1;
19     #10;
20     rst = 0;
21  end
22
23  always
24     begin
25     clk=1; #5;
26     clk=0; #5;
27     end
28
29  always @(posedge clk)
30     if(~rst)
31     begin
32        {clk2,rst2,d,q_esperado} = vectors[counter];
33     end
34
35  always @(negedge clk)   //Sempre (que o clock descer)
36        if(~rst)
37        begin
38           aux_error = errors;
```

```
39          assert (q === q_esperado)
40       else
41       begin
42          $display("%d linha , saida = %b, (%b esperado)",counter+1, q, q_esperado);
43
44          errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado
45          end
46       if(aux_error === errors)
47          $display("| %b | %b | %b | %b | OK", clk2, rst2, d, q);
48       else
49          $display("| %b | %b | %b | %b | ERROR", clk2, rst2, d, q);
50
51          counter++;  //Incrementa contador dos vertores de teste
52
53          if(counter == $size(vectors)) //Quando os vetores de teste acabarem
54          begin
55             $display("Testes Efetuados  = %0d", counter);
56             $display("Erros Encontrados = %0d", errors);
57             #15
58             $stop;
59          end
60
61       end
62
63    endmodule
64
```

Figura 4 Flopr - Testbench

2.4 Modelo duv

Para o modulo flopr temos como entrada clock, reset e dado e como saída apenas o Q. Como se trata de um flopr assíncrono, o reset também vai ser parâmetro para o *aways* que irá mudar o valor da saída caso o ele mude (independente do clock). Abaixo podemos ver o comportamento completo do módulo.

```
1  module flopr(input logic clk, input logic reset, input logic d, output logic q);
2
3  //Reset Assincrono
4  always_ff @(posedge clk, posedge reset)
5     if (reset) q <= 1'b0;
6     else q <= d;
7  endmodule
8
```

Figura 5 Flopr - Modelo duv

2.5 Simulação

Após iniciarmos a simulação e testarmos os tempos mínimos de clock para o modulo funcionar sem erros, chegamos aos seguintes resultados:

```
# Iniciando Testbench
# | CLK | RST | D | Q |
# ---------------
# | 0 | 1 | 0 | 0 | OK
# | 1 | 1 | 0 | 0 | OK
# | 0 | 1 | 1 | 0 | OK
# | 1 | 1 | 1 | 0 | OK
# | 0 | 0 | 1 | 0 | OK
# | 1 | 0 | 1 | 1 | OK
# | 0 | 0 | 0 | 1 | OK
# | 1 | 0 | 0 | 0 | OK
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 6 Flopr - Transcript da simulação RTL

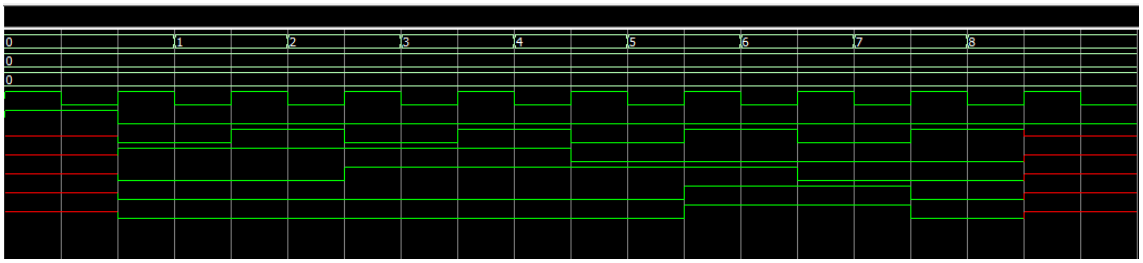

Figura 7 Flopr - Sinais de entrada e saída

3 Flopenr

Abaixo podemos ver o diagrama de blocos de um Flop com reset assincrono e eneble. É valido notar que a imagem ilustra um flopenr com 4 bits de entrada/saída da dados, porém o modulo que vai ser criado aqui é de apenas 1 bit de entrada/saída de dados.

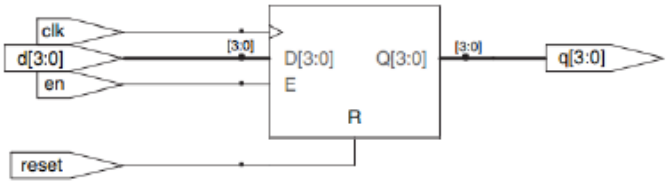

Figura 8 Flopenr - Diagrama de blocos

3.1 Goldem model

Abaixo segue o trecho do código utilizado para gerar os *goldem vectors*

```
//Determina a saída de acordo com as entradas
for(int i = 0; i < tamanho; i++){
    //Se reset = 1, y = 0
    if(tabelaVerdade[i][1] == 1){
        tabelaVerdade[i][4] = 0;
    }
     //Se en = 1, y = d;
    if(tabelaVerdade[i][2] == 1){
        tabelaVerdade[i][4] = tabelaVerdade[i][3];
    }
}
```

Figura 9 Flopenr - Goldem Model

3.2 Goldem vector

A partir do *goldem model* criado anteriormente, geramos o seguinte vetor de testes. Da esquerda para a direita temos clock, reset, enable, dado e saída. Cada entrada/saída possui apenas um bit de tamanho.

```
0_1_0_0_0
1_1_0_0_0
0_1_1_0_0
1_1_1_0_0
0_1_0_1_0
1_1_0_1_0
0_1_1_1_0
1_1_1_1_0
0_0_0_0_0
1_0_0_0_0
0_0_1_0_0
1_0_1_0_0
0_0_0_1_0
1_0_0_1_0
0_0_1_1_0
1_0_1_1_1
```

Figura 10 Flopenr - Goldem vectors

3.3 Testbench

Abaixo podemos ver o código em SystemVerilog responsável por testar o modulo flopenr. Ao iniciar damos um pulso no reset para que a saída saia do estado indeterminado “X”. Para o flopenr, o tempo mínimo de clock encontrado necessário para o modulo funcionar perfeitamente foi de 6000ps.

```
1  `timescale 1ns/100ps
2  module flopenr_tb;
3
4  int counter, errors, aux_error;
5  logic clk,rst;
6  logic clk2,rst2;
7  logic d;
8  logic en;
9  logic q, q_esperado;
10 logic [4:0]vectors[16];
11 flopenr dut(clk2, rst2, en, d, q);
12
13 initial begin
14    $display("Iniciando Testbench");
15    $display("| CLK | RST | EN | D | Q |");
16    $display("---------------");
17    $readmemb("flopenr_tv.tv",vectors);
18    counter=0; errors=0;
19    rst = 1;
20    #12;
21    rst = 0;
22 end
23
24 always
25    begin
26    clk=1; #6;
27    clk=0; #6;
28    end
29
30 always @(posedge clk)
31    if(~rst)
32    begin
33       {clk2,rst2,en,d,q_esperado} = vectors[counter];
34    end
35
36 always @(negedge clk)   //Sempre (que o clock descer)
37       if(~rst)
38       begin
39          aux_error = errors;
40          assert (q === q_esperado)
41       else
42       begin
43          $display("%d linha , sa?da = %b, (%b esperado)",counter+1, q, q_esperado);
44
45          errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado
46          end
47       if(aux_error === errors)
48          $display("| %b | %b | %b | %b | %b | OK", clk2, rst2, en, d, q);
49       else
50          $display("| %b | %b | %b | %b | %b | ERROR", clk2, rst2, en, d, q);
51
52          counter++;  //Incrementa contador dos vertores de teste
53
54          if(counter == $size(vectors)) //Quando os vetores de teste acabarem
55          begin
56             $display("Testes Efetuados  = %0d", counter);
57             $display("Erros Encontrados = %0d", errors);
58             #15
59             $stop;
60          end
61
62       end
63
64 endmodule
```

Figura 11 Flopenr - Testbench

3.4 Modelo duv

Para o modulo flopenr temos como entrada clock, reset, enable, dado e como saída apenas o Q. Como se trata de um flopenr assíncrono, o reset também vai ser parâmetro para o *aways* que irá mudar o valor da saída caso o ele mude (independente do clock). Abaixo podemos ver o comportamento completo do módulo.

```
1  module flopenr(input logic clk, input logic reset, input logic en, input logic d, output logic q);
2
3  //Reset Assincrono
4  always @(posedge clk, posedge reset)
5     if (reset) q <= 1'b0;
6     else if (en) q <= d;
7  endmodule
8
```

Figura 12 Flopenr - Modelo duv

3.5 Simulação

Após iniciarmos a simulação e testarmos os tempos mínimos de clock para o modulo funcionar sem erros, chegamos aos seguintes resultados:

```
# Iniciando Testbench
# | CLK | RST | EN | D | Q |
# ---------------
# | 0 | 1 | 0 | 0 | 0 | OK
# | 1 | 1 | 0 | 0 | 0 | OK
# | 0 | 1 | 1 | 0 | 0 | OK
# | 1 | 1 | 1 | 0 | 0 | OK
# | 0 | 1 | 0 | 1 | 0 | OK
# | 1 | 1 | 0 | 1 | 0 | OK
# | 0 | 1 | 1 | 1 | 0 | OK
# | 1 | 1 | 1 | 1 | 0 | OK
# | 0 | 0 | 0 | 0 | 0 | OK
# | 1 | 0 | 0 | 0 | 0 | OK
# | 0 | 0 | 1 | 0 | 0 | OK
# | 1 | 0 | 1 | 0 | 0 | OK
# | 0 | 0 | 0 | 1 | 0 | OK
# | 1 | 0 | 0 | 1 | 0 | OK
# | 0 | 0 | 1 | 1 | 0 | OK
# | 1 | 0 | 1 | 1 | 1 | OK
# Testes Efetuados  = 16
# Erros Encontrados = 0
```

Figura 13 Flopenr - Transcript da simulação RTL

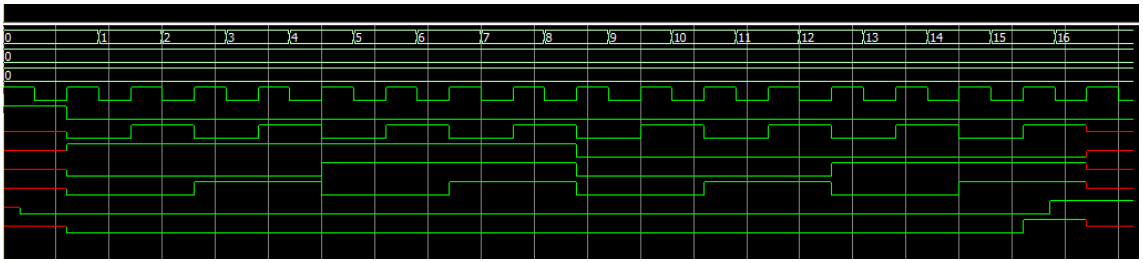

Figura 14 Flopenr - Sinais de entrada e saída

4 Mux 2:1

Abaixo podemos ver o diagrama de blocos de um multiplexador de duas entradas. É valido notar que a imagem ilustra um mux 2:1 com 4 bits em cada entrada (d0 e d1) e 4

bits na saída Y, porém o modulo que vai ser criado aqui é de apenas 1 bit para as entradas/saída.

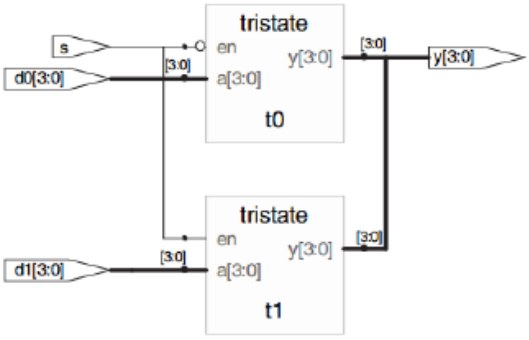

Figura 15 Mux2 - Diagrama de blocos

4.1 Goldem model

Abaixo segue o trecho do código utilizado para gerar os *goldem vectors.*

```
//Cria uma tabela com todas as possibilidades
for(int i = 0; i < tamanho; i++){
    aux = i%2;
    tabelaVerdade[i][2] = aux;
    aux = (i/2)%2;
    tabelaVerdade[i][1] = aux;
    aux = (i/4)%2;
    tabelaVerdade[i][0] = aux;
}

//Determina a saída de acordo com as entradas
for(int i = 0; i < tamanho; i++){
    //Se s=0, y=d0
    if(tabelaVerdade[i][2] == 0){
        tabelaVerdade[i][3] = tabelaVerdade[i][0];
    }
    //Se s = 1, y=d1
    else if(tabelaVerdade[i][2] == 1){
        tabelaVerdade[i][3] = tabelaVerdade[i][1];
    }
}
```

Figura 16 Mux2 - Goldem model

4.2 Goldem vector

A partir do *goldem model* criado anteriormente, geramos o seguinte vetor de testes. Da esquerda para a direita temos d0, d1, s, e y. Cada entrada/saída possui apenas um bit de tamanho.

```
0_0_0_0
0_0_0_0
1_0_0_1
0_1_0_0
1_1_0_1
0_0_1_0
1_0_1_0
0_1_1_1
1_1_1_1
```

Figura 17 Mux2 - Goldem vectors

4.3 Testbench

Abaixo podemos ver o código em SystemVerilog responsável por testar o modulo mux2. O tempo mínimo de clock encontrado necessário para o modulo funcionar perfeitamente foi de 10000ps.

```
1  `timescale 1ns/100ps
2
3  module mux2_tb;
4
5  int counter, errors, aux_error;
6  logic clk,rst;
7  logic d0,d1;
8  logic s;
9  logic ns;
10 logic y,y_esperado;
11 logic [3:0]vectors[9];
12
13 mux2 dut(d0, d1, s, y);
14
15 initial begin
16    $display("Iniciando Testbench");
17    $display("| D0 | D1 | S | Y |");
18    $display("---------------");
19    $readmemb("mux2_tv.tv",vectors);
20    counter=0; errors=0;
21    rst = 1;
22    #20;
23    rst = 0;
24 end
25
26 always begin
27     clk=1; #10; //O clock em 1 durara 7ns
28    clk=0; #10; //O clock em 0 durara 5ns
29 end
30
31 always @(posedge clk)   //Sempre (que o clock subir)
32    if(~rst)
33    begin
34       //Atribui valores do vetor nas entradas do DUT e nos valores esperados
35       {d0,d1,s,y_esperado} = vectors[counter];
36    end
37
```

```
38  always @(negedge clk)   //Sempre (que o clock descer)
39        if(~rst)
40        begin
41           aux_error = errors;
42
43        assert (y === y_esperado)
44        else
45        begin
46           //Mostra mensagem de erro se a saida do DUT for diferente da saida esperada
47           $display("Error: input = %b", d0);
48           $display("%d OPCAO , output = %b, (%b expected)",s, y, y_esperado);
49
50           errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado
51           end
52        if(aux_error === errors)
53           $display("| %b | %b | %b | %b | OK", d0, d1, s, y);
54        else
55           $display("| %b | %b | %b | %b | ERROR", d0, d1, s, y);
56
57           counter++;  //Incrementa contador dos vertores de teste
58
59           if(counter == $size(vectors)) //Quando os vetores de teste acabarem
60           begin
61              $display("Testes Efetuados  = %0d", counter);
62              $display("Erros Encontrados = %0d", errors);
63              #10
64              $stop;
65           end
66
67        end
68
69  endmodule
```

Figura 18 Mux2 - TestBench

4.4 Modelo duv

Para o modulo mux2 temos como entrada d0, d1 e S e como saída apenas o Y. O mux2 é feio a partir de dois módulos tristate e um inversor, onde o primeiro tristate recebe o dado d0 e o controle S invertido (~S) e o segundo tristate recebe o dado d1 e o controle S. Abaixo podemos ver o comportamento do modulo.

```
1  module mux2(input logic d0, d1, input logic s, output tri y);
2
3   tristate t0(d0, ns, y);
4   tristate t1(d1, s, y);
5   inversor inv(s, ns);
6
7  endmodule
8
```

Figura 19 Mux2 - Modelo duv

4.5 Simulação

Após iniciarmos a simulação e testarmos os tempos mínimos de clock para o modulo funcionar sem erros, chegamos aos seguintes resultados:

```
# Iniciando Testbench
# | D0 | D1 | S | Y |
# ---------------
# | 0 | 0 | 0 | 0 | OK
# | 0 | 0 | 0 | 0 | OK
# | 1 | 0 | 0 | 1 | OK
# | 0 | 1 | 0 | 0 | OK
# | 1 | 1 | 0 | 1 | OK
# | 0 | 0 | 1 | 0 | OK
# | 1 | 0 | 1 | 0 | OK
# | 0 | 1 | 1 | 1 | OK
# | 1 | 1 | 1 | 1 | OK
# Testes Efetuados  = 9
# Erros Encontrados = 0
```

Figura 20 Mux2 - Transcript da simulação RTL

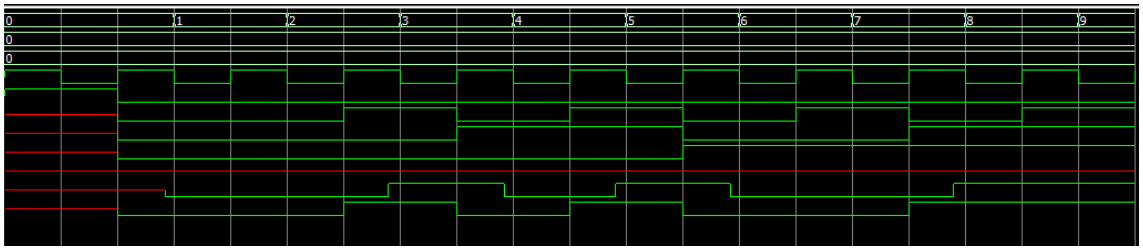

Figura 21 Mux2 - Sinais de entrada e saída

5 Mux 4:1

Abaixo podemos ver o diagrama de blocos de um multiplexador de quatro entradas. É valido notar que a imagem ilustra um mux 4:1 com 4 bits em cada entrada (d0,d1, d2 e d3) e 4 bits na saída Y, porém o modulo que vai ser criado aqui é de apenas 1 bit para as entradas/saída.

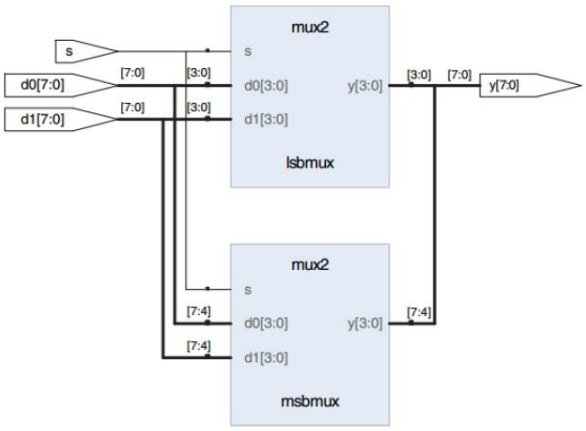

Figura 22 Mux4 - Diagrama de blocos

5.1 Goldem model

Abaixo segue o trecho do código utilizado para gerar os *goldem vectors.*

```
//Determina a saída de acordo com as entradas
for(int i = 0; i < tamanho; i++){
    //Se s0 = 0 e s1 = 0, y = d0
    if(tabelaVerdade[i][4] == 0 && tabelaVerdade[i][5] == 0){
        tabelaVerdade[i][6] = tabelaVerdade[i][0];
    }
    //Se s0 = 1 e s1 = 0, y = d1
    if(tabelaVerdade[i][4] == 1 && tabelaVerdade[i][5] == 0){
        tabelaVerdade[i][6] = tabelaVerdade[i][1];
    }
    //Se s0 = 0 e s1 = 1, y = d2
    if(tabelaVerdade[i][4] == 0 && tabelaVerdade[i][5] == 1){
        tabelaVerdade[i][6] = tabelaVerdade[i][2];
    }
    //Se s0 = 1 e s1 = 1, y = d3
    if(tabelaVerdade[i][4] == 1 && tabelaVerdade[i][5] == 1){
        tabelaVerdade[i][6] = tabelaVerdade[i][3];
    }
`
```

Figura 23 Mux4 - Goldem Model

5.2 Goldem vector

A partir do *goldem model* criado anteriormente, geramos o seguinte vetor de testes. Da esquerda para a direita temos d0, d1, d2, d3, s0, s1 e y. Cada entrada/saída possui apenas um bit de tamanho.

```
0_1_1_1_0_0_0
0_1_1_1_0_0_0
1_0_0_0_0_0_1
1_0_1_1_1_0_0
0_1_0_0_1_0_1
1_1_0_1_0_1_0
0_0_1_0_0_1_1
1_1_1_0_1_1_0
0_0_0_1_1_1_1
```

Figura 24 Mux4 - Goldem vectors

5.3 Testbench

Abaixo podemos ver o código em SystemVerilog responsável por testar o modulo mux2. O tempo mínimo de clock encontrado necessário para o modulo funcionar perfeitamente foi de 10000ps.

```
1   `timescale 1ns/100ps
2
3   module mux4_tb;
4
5   int counter, errors, aux_error;
6   logic clk,rst;
7   logic d0, d1, d2, d3;
8   logic s0;
9   logic s1;
10  logic y,y_esperado;
11  logic [6:0]vectors[9];
12
13  mux4 dut(d0, d1, d2, d3, s0, s1, y);
14
15  initial begin
16      $display("Iniciando Testbench");
17      $display("| D0 | D1 | D2 | D3 | S0 | S1 | Y |");
18      $display("-----------------------------------");
19      $readmemb("mux4_tv.tv",vectors);
20      counter=0; errors=0;
21      rst = 1;
22      #20;
23      rst = 0;
24  end
25
26  always begin
27      clk=1; #10; //O clock em 1 dura 7ns
28      clk=0; #10; //O clock em 0 dura 5ns
29  end
30
31  always @(posedge clk)    //Sempre (que o clock subir)
32      if(~rst)
33      begin
34          //Atribui valores do vetor nas entradas do DUT e nos valores esperados
35          {d0, d1, d2, d3, s0, s1, y_esperado} = vectors[counter];
36      end
37
38  always @(negedge clk)    //Sempre (que o clock descer)
39          if(~rst)
40          begin
41              aux_error = errors;
42
43          assert (y === y_esperado)
44          else
45          begin
46              //Mostra mensagem de erro se a saida do DUT for diferente da saida esperada
47              //$display("Error: input in position %d = %b", i, d0);
48              $display("%d OPÇÃO , output = %b, (%b expected)", s0, y, y_esperado);
49
50              errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado
51              end
52
53          if(aux_error === errors)
54              $display("| %b | %b | %b | %b | %b | %b | %b | OK", d0, d1, d2, d3, s0, s1, y);
55          else
56              $display("| %b | %b | %b | %b | %b | %b | %b | ERRO", d0, d1, d2, d3, s0, s1, y);
57
58              counter++;  //Incrementa contador dos vertores de teste
59
60              if(counter == $size(vectors)) //Quando os vetores de teste acabarem
61              begin
62                  $display("Testes Efetuados  = %0d", counter);
63                  $display("Erros Encontrados = %0d", errors);
64                  #15
65                  $stop;
66              end
67
68          end
69
70  endmodule
```

Figura 25 Mux4 - Testbench

5.4 Modelo duv

Para o modulo mux4 temos como entrada d0, d1, d2, d3, s0 s1 e como saída apenas o Y. O mux4 é feio a partir de três módulos mux2, onde o primeiro mux recebe os dados d0 e d1, o controle

s0 e sai com y0. O segundo mux recebe os dados d2 e d3, o controle s0, e sai com y1. O terceiro mux recebe em sua entrada d0 o dado proveniente da saída y0 do primeiro mux e na entrada d1 o dado vindo da saída y1 do segundo mux. Ele também recebe o controle s1 e sai com o dado Y.

```
1  module mux4(input logic d0, d1, d2, d3, input logic s0, s1, output logic y);
2
3     mux2 m1(d0, d1, s0, y0);
4     mux2 m2(d2, d3, s0, y1);
5     mux2 m3(y0, y1, s1, y);
6
7  endmodule
8
```

Figura 26 Mux4 - Modelo duv

5.5 Simulação

Após iniciarmos a simulação e testarmos os tempos mínimos de clock para o modulo funcionar sem erros, chegamos aos seguintes resultados:

```
# Iniciando Testbench
# | D0 | D1 | D2 | D3 | S0 | S1 | Y |
# -----------------------------------
# | 0 | 1 | 1 | 1 | 0 | 0 | 0 | OK
# | 0 | 1 | 1 | 1 | 0 | 0 | 0 | OK
# | 1 | 0 | 0 | 0 | 0 | 0 | 1 | OK
# | 1 | 0 | 1 | 1 | 1 | 0 | 0 | OK
# | 0 | 1 | 0 | 0 | 1 | 0 | 1 | OK
# | 1 | 1 | 0 | 1 | 0 | 1 | 0 | OK
# | 0 | 0 | 1 | 0 | 0 | 1 | 1 | OK
# | 1 | 1 | 1 | 0 | 1 | 1 | 0 | OK
# | 0 | 0 | 0 | 1 | 1 | 1 | 1 | OK
# Testes Efetuados  = 9
# Erros Encontrados = 0
```

Figura 27 Mux4 - Transcript da simulação RTL

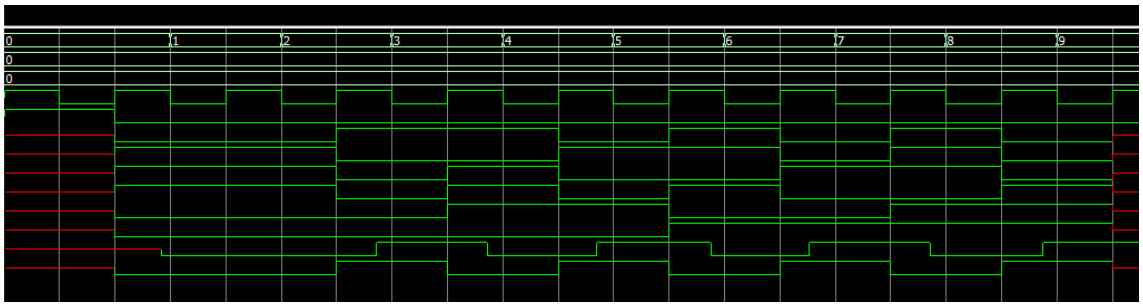

Figura 28 Mux4 - Sinais de entrada e saída