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

PROJETO MIPS – ENTREGA 5

THIAGO ALVES DE ARAUJO

2016019787

Sumário

1 Tabela de figuras

2 Mux 2:1 5 bits

Agora, vamos desenvolver um multiplexador de 2 entradas, porém, diferente do anterior, cada entrada dessa possui 5 bits.

2.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
00000_11111_0_11111
11111_00000_1_11111
11111_00000_0_00000
00000_11111_1_00000
```

Figura 1 Mux 2 5bit - Goldem vectors

2.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
`timescale 1ns/100ps

module mux2_tb;

int counter, errors, aux_error;
logic clk,rst;
logic [4:0]d1;
logic [4:0]d0;
logic s;
logic [4:0]y,y_esperado;
logic [15:0]vectors[4];

mux2 dut(.d1(d1), .d0(d0), .s(s), .y(y));

initial begin
   $display("Iniciando Testbench");
   $display("|   D0    |    D1   | S  |   Y     |  Y_esperado  |");
   $display("|---------|---------|----|---------|--------------|");
   $readmemb("mux2_tv.tv",vectors);
   counter=0; errors=0;
   rst = 1;
   #12;
   rst = 0;
end

always begin
    clk=1; #7;
    clk=0; #5;
end

always @(posedge clk)
   if(~rst)
   begin
      //Atribui valores do vetor nas entradas do DUT e nos valores esperados
      {d1, d0, s ,y_esperado} = vectors[counter];
   end

always @(negedge clk)   //Sempre (que o clock descer)
      if(~rst)
      begin
         aux_error = errors;

      assert (y === y_esperado)

      else  errors = errors + 1;

      if(aux_error === errors)
         $display("|  %b  |  %b  |  %b  |  %b  |  %b  | OK",    d0, d1, s, y, y_esperado);
      else
         $display("|  %b  |  %b  |  %b  |  %b  |  %b  | ERROR", d0, d1, s, y, y_esperado);

         counter++;  //Incrementa contador dos vertores de teste

         if(counter == $size(vectors)) //Quando os vetores de teste acabarem
         begin
            $display("Testes Efetuados  = %0d", counter);
            $display("Erros Encontrados = %0d", errors);
            #10
            $stop;
         end

      end

endmodule
```

Figura 2 Mux 2 5bit - TestBench

2.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Temos duas entradas de 5 bits (d0 e d1), uma entrada de controle S e uma saída de 5 bits (y).

```
1  module mux2(input logic [4:0]d1, input logic [4:0]d0, input logic s, output tri [4:0]y);
2
3   logic ns;
4
5   tristate t0(d0, ns, y);
6   tristate t1(d1, s, y);
7   inversor inv(s, ns);
8
9  endmodule
```

Figura 3 Mux 2 5bit - Modelo Duv

2.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
# |    D0    |    D1    |  S  |    Y     |  Y_esperado  |
# |----------|----------|-----|----------|--------------|
# |  11111   |  00000   |  0  |  11111   |  11111   | OK
# |  00000   |  11111   |  1  |  11111   |  11111   | OK
# |  00000   |  11111   |  0  |  00000   |  00000   | OK
# |  11111   |  00000   |  1  |  00000   |  00000   | OK
# Testes Efetuados  = 4
# Erros Encontrados = 0
```

Figura 4 Mux 2 5bit - Simulação RTL

Também podemos observar a visualização RTL do modulo criado.

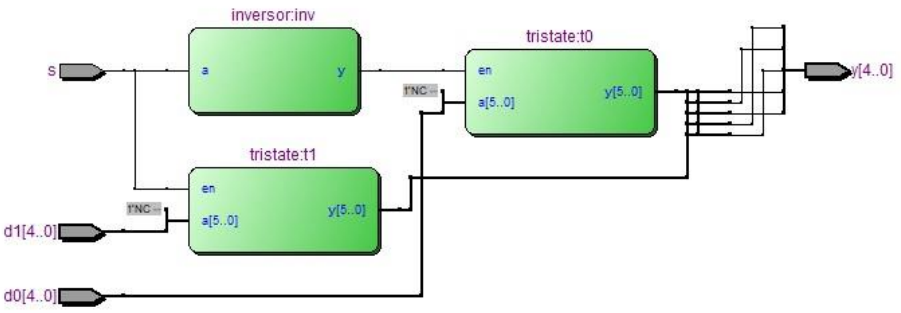


Figura 5 Mux 2 5bit - RTL view

2.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 8. Como podemos ver, 4 erros foram encontrados.

```
# Iniciando Testbench
# |   D0    |    D1   |  S  |   Y     |  Y_esperado  |
# |---------|---------|-----|---------|--------------|
# |  11111  |  00000  |  0  |  xxxxx  |  11111  | ERROR
# |  00000  |  11111  |  1  |  10111  |  11111  | ERROR
# |  00000  |  11111  |  0  |  11111  |  00000  | ERROR
# |  11111  |  00000  |  1  |  01000  |  00000  | ERROR
# Testes Efetuados  = 4
# Erros Encontrados = 4
```

Figura 6 Mux 2 5bit – Gate Level clk 8

Subindo o clock para 9, o modulo consegue se comportar normalmente.

```
# Iniciando Testbench
# |   D0    |    D1   |  S  |   Y     |  Y_esperado  |
# |---------|---------|-----|---------|--------------|
# |  11111  |  00000  |  0  |  11111  |  11111  | OK
# |  00000  |  11111  |  1  |  11111  |  11111  | OK
# |  00000  |  11111  |  0  |  00000  |  00000  | OK
# |  11111  |  00000  |  1  |  00000  |  00000  | OK
# Testes Efetuados  = 4
# Erros Encontrados = 0
```

Figura 7 Mux 2 5bit – Gate Level clk 9

Abaixo podemos ver os sinais de entrada e saída.

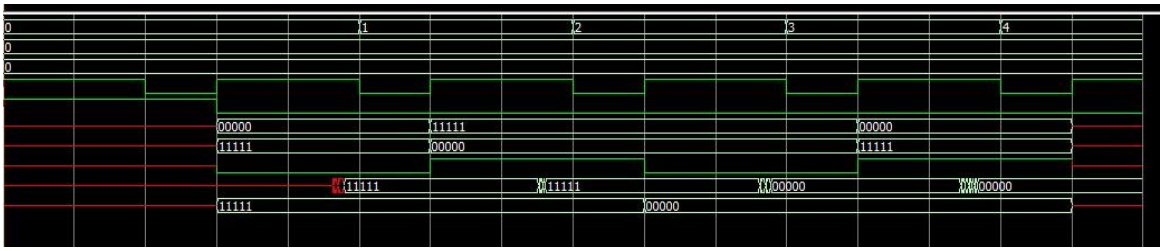


Figura 8 Mux 2 5bit – Sinais de entrada e saida

3 Mux 8:1

Agora vamos desenvolver um multiplexador de 8:1. Ele é muito parecido com o mux4 desenvolvido anteriormente, porém com algumas mudanças.

3.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
00000001_000_1
00000010_001_1
00000100_010_1
00001000_011_1
00010000_100_1
00100000_101_1
01000000_110_1
10000000_111_1
11111110_000_0
11111101_001_0
11111011_010_0
11110111_011_0
11101111_100_0
11011111_101_0
10111111_110_0
01111111_111_0
```

Figura 9 Mux 8 - Goldem vectors

3.2 Testbench

Abaixo Podemos ver o testbench utilizado para similar o modulo.

```
1  `timescale 1ns/100ps
2
3  module mux8_tb;
4
5  int counter, errors, aux_error;
6  logic clk,rst;
7  logic [7:0]d;
8  logic [2:0]s;
9  logic y,y_esperado;
10 logic [11:0]vectors[16];
11
12 mux8 dut(.d(d), .s(s), .y(y));
13
14 initial begin
15    $display("Iniciando Testbench");
16    $display("|      D     |   S   |  Y  |  Y_esperado  |");
17    $display("|------------|-------|-----|--------------|");
18    $readmemb("mux8_tv.tv",vectors);
19    counter=0; errors=0;
20    rst = 1;
21    #16;
22    rst = 0;
23 end
24
25 always begin
26     clk=1; #11;
27     clk=0; #5;
28 end
29
30 always @(posedge clk)
31    if(~rst)
32    begin
33       //Atribui valores do vetor nas entradas do DUT e nos valores esperados
34       {d,s,y_esperado} = vectors[counter];
35    end
36
37 always @(negedge clk)   //Sempre (que o clock descer)
38       if(~rst)
39       begin
40          aux_error = errors;
41
42       assert (y === y_esperado)
43
44       else  errors = errors + 1;
45
46       if(aux_error === errors)
47          $display("|  %b  |  %b  |  %b  |      %b       | OK",  d, s, y, y_esperado);
48       else
49          $display("|  %b  |  %b  |  %b  |      %b       | ERROR", d, s, y, y_esperado);
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             #10
58             $stop;
59          end
60
61       end
62
63 endmodule
64
```

Figura 10 Mux 8 - Testbench

3.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Temos uma entrada de dados de 8 bits (d), uma entrada de controle de 2 bits (s) e uma saída de 1 bit (y).

```
1   module mux8(input logic [7:0]d, input logic [2:0]s, output logic y);
2
3      logic [1:0]y0;
4
5      mux4 m4_0(d[3:0], s[1:0], y0[0]);
6      mux4 m4_1(d[7:4], s[1:0], y0[1]);
7
8      mux2 m2_0(y0[1:0], s[2], y);
9
10  endmodule
```

Figura 11 Mux 8 - Modelo Duv

3.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
# |      D     |   S   |  Y  |  Y_esperado  |
# |------------|-------|-----|--------------|
# |  00000001  |  000  |  1  |      1       | OK
# |  00000010  |  001  |  1  |      1       | OK
# |  00000100  |  010  |  1  |      1       | OK
# |  00001000  |  011  |  1  |      1       | OK
# |  00010000  |  100  |  1  |      1       | OK
# |  00100000  |  101  |  1  |      1       | OK
# |  01000000  |  110  |  1  |      1       | OK
# |  10000000  |  111  |  1  |      1       | OK
# |  11111110  |  000  |  0  |      0       | OK
# |  11111101  |  001  |  0  |      0       | OK
# |  11111011  |  010  |  0  |      0       | OK
# |  11110111  |  011  |  0  |      0       | OK
# |  11101111  |  100  |  0  |      0       | OK
# |  11011111  |  101  |  0  |      0       | OK
# |  10111111  |  110  |  0  |      0       | OK
# |  01111111  |  111  |  0  |      0       | OK
# Testes Efetuados  = 16
# Erros Encontrados = 0
```

Figura 12 Mux 8 - Simulação RTL

Também podemos observar a visualização RTL do modulo criado.

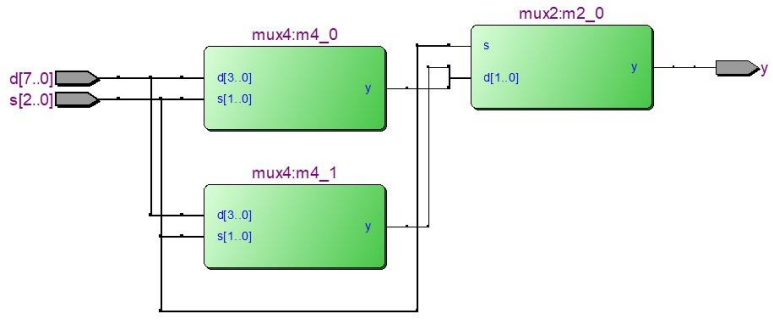


Figura 13 Mux 8 - RTL view

3.5 Simulação *Gate Level*

Iniciamos a simulação gate level com o clock em 9 e subimos até 11 onde podemos observar que os erros somem e o modulo consegue se comportar normalmente.

```
# Iniciando Testbench
# |      D      |   S   |  Y  |  Y_esperado  |
# |-------------|-------|-----|--------------|
# |  00000001   |  000  |  1  |       1      | OK
# |  00000010   |  001  |  1  |       1      | OK
# |  00000100   |  010  |  1  |       1      | OK
# |  00001000   |  011  |  1  |       1      | OK
# |  00010000   |  100  |  1  |       1      | OK
# |  00100000   |  101  |  1  |       1      | OK
# |  01000000   |  110  |  1  |       1      | OK
# |  10000000   |  111  |  1  |       1      | OK
# |  11111110   |  000  |  0  |       0      | OK
# |  11111101   |  001  |  0  |       0      | OK
# |  11111011   |  010  |  0  |       0      | OK
# |  11110111   |  011  |  0  |       0      | OK
# |  11101111   |  100  |  0  |       0      | OK
# |  11011111   |  101  |  0  |       0      | OK
# |  10111111   |  110  |  1  |       0      | ERROR
# |  01111111   |  111  |  0  |       0      | OK
# Testes Efetuados  = 16
# Erros Encontrados = 1
```

Figura 14 Mux 8 – Gate Level clk 9

```
# Iniciando Testbench
# |      D      |   S   |  Y  |  Y_esperado  |
# |-------------|-------|-----|--------------|
# |  00000001   |  000  |  x  |       1      | ERROR
# |  00000010   |  001  |  1  |       1      | OK
# |  00000100   |  010  |  1  |       1      | OK
# |  00001000   |  011  |  1  |       1      | OK
# |  00010000   |  100  |  0  |       1      | ERROR
# |  00100000   |  101  |  1  |       1      | OK
# |  01000000   |  110  |  1  |       1      | OK
# |  10000000   |  111  |  1  |       1      | OK
# |  11111110   |  000  |  1  |       0      | ERROR
# |  11111101   |  001  |  0  |       0      | OK
# |  11111011   |  010  |  1  |       0      | ERROR
# |  11110111   |  011  |  1  |       0      | ERROR
# |  11101111   |  100  |  0  |       0      | OK
# |  11011111   |  101  |  1  |       0      | ERROR
# |  10111111   |  110  |  1  |       0      | ERROR
# |  01111111   |  111  |  0  |       0      | OK
# Testes Efetuados  = 16
# Erros Encontrados = 7
```

Figura 15 Mux 8 – Gate Level clk 10

```
# Iniciando Testbench
# |      D      |   S   |  Y  |  Y_esperado  |
# |-------------|-------|-----|--------------|
# |  00000001   |  000  |  1  |       1      | OK
# |  00000010   |  001  |  1  |       1      | OK
# |  00000100   |  010  |  1  |       1      | OK
# |  00001000   |  011  |  1  |       1      | OK
# |  00010000   |  100  |  1  |       1      | OK
# |  00100000   |  101  |  1  |       1      | OK
# |  01000000   |  110  |  1  |       1      | OK
# |  10000000   |  111  |  1  |       1      | OK
# |  11111110   |  000  |  0  |       0      | OK
# |  11111101   |  001  |  0  |       0      | OK
# |  11111011   |  010  |  0  |       0      | OK
# |  11110111   |  011  |  0  |       0      | OK
# |  11101111   |  100  |  0  |       0      | OK
# |  11011111   |  101  |  0  |       0      | OK
# |  10111111   |  110  |  0  |       0      | OK
# |  01111111   |  111  |  0  |       0      | OK
# Testes Efetuados  = 16
# Erros Encontrados = 0
```

Figura 16 Mux 8 – Gate Level clk 11

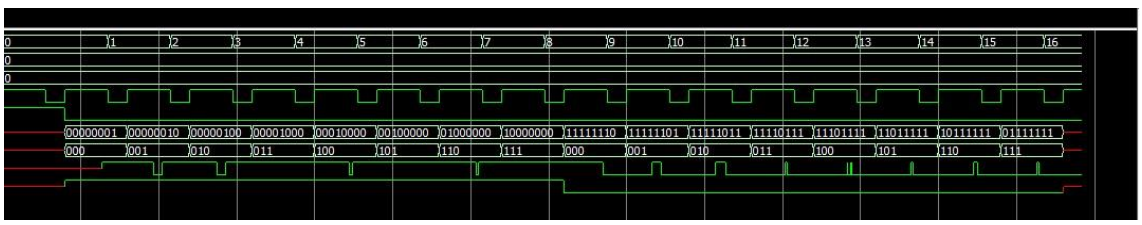


Figura 17 Mux 8 - – Sinais de entrada e saida

4 Mux 2:1 32 bits

Agora vamos estender as entradas de saídas do mux2 (criado anteriormente) para 32 bits. As mudanças são muito poucas e podem ser observadas principalmente no testbench e nos goldem vectors

4.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
00000000000000000000000000000000_
11111111111111111111111111111111_0_
11111111111111111111111111111111
11111111111111111111111111111111_
00000000000000000000000000000000_1_
11111111111111111111111111111111
11111111111111111111111111111111_
00000000000000000000000000000000_0_
00000000000000000000000000000000
00000000000000000000000000000000_
11111111111111111111111111111111_1_
00000000000000000000000000000000
```

Figura 18 Mux 2 32 bits - Goldem vectors

4.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
1  `timescale 1ns/100ps
2
3  module mux2_tb;
4
5  int counter, errors, aux_error;
6  logic clk,rst;
7  logic [31:0]d1;
8  logic [31:0]d0;
9  logic s;
10 logic [31:0]y,y_esperado;
11 logic [96:0]vectors[4];
12
13 mux2 dut(.d1(d1), .d0(d0), .s(s), .y(y));
14
15 initial begin
16    $display("Iniciando Testbench");
17    $display("|                D0                |                D1                | S  |
18    $display("|----------------------------------|----------------------------------|----|----------
19    $readmemb("mux2_tv.tv",vectors);
20    counter=0; errors=0;
21    rst = 1;
22    #13;
23    rst = 0;
24 end
25
26 always begin
27     clk=1; #5;
28     clk=0; #5;
29 end
30
31 always @(posedge clk)
32    if(~rst)
33    begin
34       //Atribui valores do vetor nas entradas do DUT e nos valores esperados
35       {d1, d0, 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
45          else  errors = errors + 1;
46
47          if(aux_error === errors)
48             $display("|  %b  |  %b  |  %b  |  %b  |      %b       | OK",    d0, d1, s, y, y_esperado);
49          else
50             $display("|  %b  |  %b  |  %b  |  %b  |      %b       | ERROR", d0, d1, s, y, y_esperado);
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                #10
59                $stop;
60             end
61
62          end
63
64       endmodule
```

Figura 19 Mux 2 32 bits - Testbench

4.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Ele tem a lógica exatamente igual ao mux2 anterior, porém a entrada possui 32 bits, assim como a saída

```
1   module mux2(input logic [31:0]d1, input logic [31:0]d0, input logic s, output tri [31:0]y);
2
3    logic ns;
4
5    tristate t0(d0, ns, y);
6    tristate t1(d1, s, y);
7    inversor inv(s, ns);
8
9   endmodule
10
```

Figura 20 Mux 2 32 - Modelu duv

4.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
#  |                D0                |                D1                | S |                Y                 |            Y_esperado            |
# |----------------------------------|----------------------------------|---|----------------------------------|----------------------------------|
#  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 0 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
#  | 00000000000000000000000000000000 | 11111111111111111111111111111111 | 1 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
#  | 00000000000000000000000000000000 | 11111111111111111111111111111111 | 0 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
#  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 1 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# Testes Efetuados  = 4
# Erros Encontrados = 0
```

Figura 21 Mux 2 32 - Simulação RTL

Também podemos observar a visualização RTL do modulo criado.

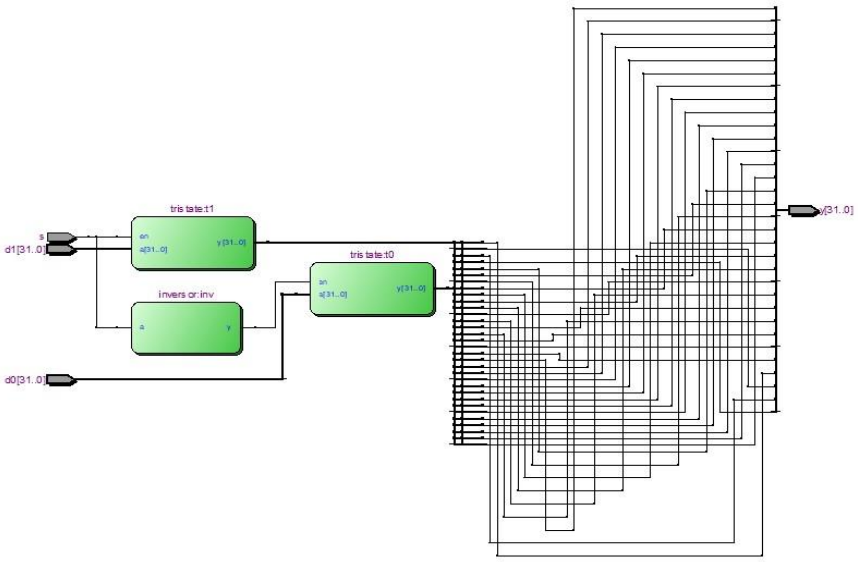


Figura 22 Mux 2 32 - RTL view

4.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 8. Como podemos ver, 4 erros foram encontrados.

```
# Iniciando Testbench
# |                D0                |                D1                | S |            Y                |            Y_esperado             |
# |----------------------------------|----------------------------------|---|-----------------------------|-----------------------------------|
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 0 |  1111111xxx111111x11x11x111111111 |  11111111111111111111111111111111 | ERROR
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 1 |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 0 |  00000000111000000100100100000000 |  00000000000000000000000000000000 | ERROR
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 1 |  00000001010000000000001000000000 |  00000000000000000000000000000000 | ERROR
# Testes Efetuados  = 4
# Erros Encontrados = 3
```

Figura 23 Mux 2 32 - clk 8

```
# Iniciando Testbench
# |                D0                |                D1                | S |            Y                |            Y_esperado             |
# |----------------------------------|----------------------------------|---|-----------------------------|-----------------------------------|
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 0 |  xlxxxxlxxxxxlxxxxxxxxxxxxxxxxxx |  11111111111111111111111111111111 | ERROR
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 1 |  11011111111111111011111111111111 |  11111111111111111111111111111111 | ERROR
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 0 |  10111011111101111111111111111111 |  00000000000000000000000000000000 | ERROR
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 1 |  00101010101001010110000000011111 |  00000000000000000000000000000000 | ERROR
# Testes Efetuados  = 4
# Erros Encontrados = 4
```

Figura 24 Mux 2 32 - clk 9

```
# Iniciando Testbench
# |                D0                |                D1                | S |            Y                |            Y_esperado             |
# |----------------------------------|----------------------------------|---|-----------------------------|-----------------------------------|
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 0 |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 1 |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# |  00000000000000000000000000000000 |  11111111111111111111111111111111 | 0 |  00000000000000000000000000000000 |  00000000000000000000000000000000 | OK
# |  11111111111111111111111111111111 |  00000000000000000000000000000000 | 1 |  00000000000000000000000000000000 |  00000000000000000000000000000000 | OK
# Testes Efetuados  = 4
# Erros Encontrados = 0
```

Figura 25 Mux 2 32 - clk 10

Abaixo podemos ver os sinais de entrada e saída.

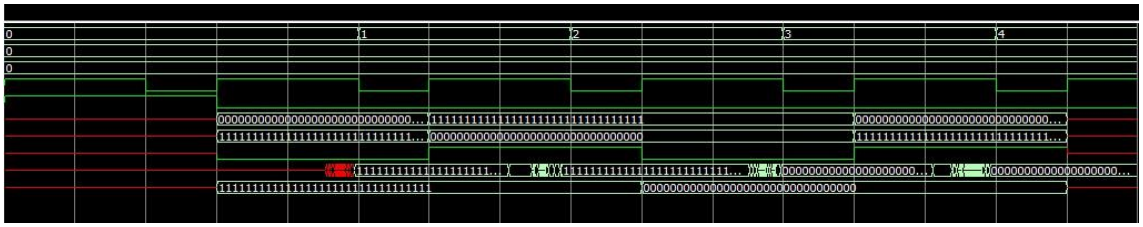


Figura 26 Mux 2 32– Sinais de entrada e saida

5 Mux 4:1 32 bits

Agora vamos estender as entradas de saídas do mux4 (criado anteriormente) para 32 bits. As mudanças são muito poucas e podem ser observadas principalmente no testbench e nos goldem vectors.

5.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
11111111111111111111111111111111_00_
11111111111111111111111111111111
00000000000000000000000000000000_
00000000000000000000000000000000_
11111111111111111111111111111111_
00000000000000000000000000000000_01_
11111111111111111111111111111111
00000000000000000000000000000000_
11111111111111111111111111111111_
00000000000000000000000000000000_
00000000000000000000000000000000_10_
11111111111111111111111111111111
11111111111111111111111111111111_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_11_
11111111111111111111111111111111
11111111111111111111111111111111_
11111111111111111111111111111111_
11111111111111111111111111111111_
00000000000000000000000000000000_00_
00000000000000000000000000000000
11111111111111111111111111111111_
11111111111111111111111111111111_
00000000000000000000000000000000_
11111111111111111111111111111111_01_
00000000000000000000000000000000
11111111111111111111111111111111_
00000000000000000000000000000000_
11111111111111111111111111111111_
11111111111111111111111111111111_10_
00000000000000000000000000000000
00000000000000000000000000000000_
11111111111111111111111111111111_
11111111111111111111111111111111_
11111111111111111111111111111111_11_
00000000000000000000000000000000
```

Figura 27 Mux 4 32 bits - Goldem vectors

5.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
`timescale 1ns/100ps

module mux4_tb;

int counter, errors, aux_error;
logic clk,rst;
logic [31:0]d0;
logic [31:0]d1;
logic [31:0]d2;
logic [31:0]d3;
logic [1:0]s;
logic [31:0]y,y_esperado;
logic [161:0]vectors[8];

mux4 dut(.d3(d3), .d2(d2), .d1(d1), .d0(d0), .s(s), .y(y));

initial begin
   $display("Iniciando Testbench");
   $display("    |                D                 |  S   |                Y                 |            Y_esperado            |");
   $display("    |----------------------------------|------|----------------------------------|----------------------------------|");

   $readmemb("mux4_tv.tv",vectors);
   counter=0; errors=0;
   rst = 1;
   #15;
   rst = 0;
end

always begin
   clk=1; #10;
   clk=0; #5;
end

always @(posedge clk)   //Sempre (que o clock subir)
   if(~rst)
   begin
      {d3, d2, d1, d0, s, y_esperado} = vectors[counter];
   end

always @(negedge clk)   //Sempre (que o clock descer)
      if(~rst)
      begin
         aux_error = errors;

      assert (y === y_esperado)

      else  errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado

      if(aux_error === errors)begin
         $display("D0 |  %b  |  %b  |  %b  |  %b  | OK",   d0, s, y, y_esperado);
         $display("D1 |  %b  |",   d1);
         $display("D2 |  %b  |",   d2);
         $display("D3 |  %b  |",   d3);
         $display(" ");
      end
      else begin
         $display("D0 |  %b  |  %b  |  %b  |  %b  | ERRO", d0, s, y, y_esperado);
         $display("D1 |  %b  |",   d1);
         $display("D2 |  %b  |",   d2);
         $display(" ");
      end
         counter++;  //Incrementa contador dos vertores de teste

         if(counter == $size(vectors)) //Quando os vetores de teste acabarem
         begin
            $display("Testes Efetuados  = %0d", counter);
            $display("Erros Encontrados = %0d", errors);
            #15
            $stop;
         end

      end

endmodule
```

Figura 28 Mux 4 32 - Testbench

5.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Ele tem a lógica exatamente igual ao mux2 anterior, porém a entrada possui 32 bits, assim como a saída

```
module mux4(input logic [31:0]d3, input logic[31:0]d2, input logic[31:0]d1,
            input logic[31:0]d0, input logic [1:0]s, output logic [31:0]y);

   logic [31:0]a0;
   logic [31:0]a1;

   mux2 m0(d1, d0, s[0], a0);
   mux2 m1(d3, d2, s[0], a1);
   mux2 m2(a1, a0, s[1], y);

endmodule
```

Figura 29 Mux 4 32 - Modelo duv

5.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# D0 |  00000000000000000000000000000000  |  10  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
# D3 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  11  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  00000000000000000000000000000000  |  00  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  01  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  10  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  00000000000000000000000000000000  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  11  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
# D3 |  00000000000000000000000000000000  |
#
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 30 Mux 4 32 - Simulação RTL

Também podemos observar a visualização RTL do modulo criado.

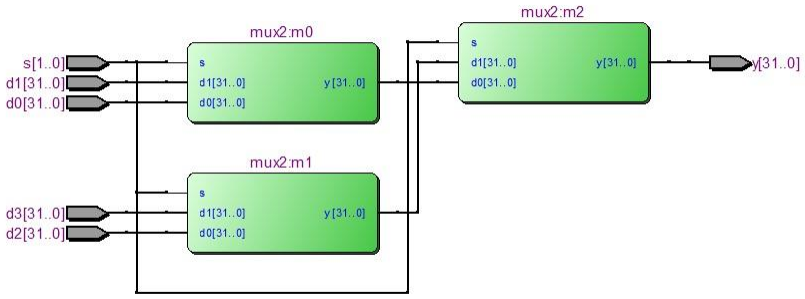


Figura 31 Mux 4 32 – RTL view

5.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 10. Como podemos ver, 8 erros foram encontrados.

```
# Iniciando Testbench
#     |                D                   |  S   |                Y                   |             Y_esperado             |
#     |------------------------------------|------|------------------------------------|------------------------------------|
# D0 |  11111111111111111111111111111111  |  00  |  11x111111x11x111xx11111111111111  |  11111111111111111111111111111111  | ERRO
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  01  |  11011111111111111111111111111111  |  11111111111111111111111111111111  | ERRO
# D1 |  11111111111111111111111111111111  |
# D2 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  10  |  11011111101101110111111111111111  |  11111111111111111111111111111111  | ERRO
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
#
# D0 |  00000000000000000000000000000000  |  11  |  11011111111111111111111111111111  |  11111111111111111111111111111111  | ERRO
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  00  |  00100000010010001100000000000000  |  00000000000000000000000000000000  | ERRO
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  01  |  00100000000000000000000000000000  |  00000000000000000000000000000000  | ERRO
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  10  |  00100000010010001100000000000000  |  00000000000000000000000000000000  | ERRO
# D1 |  11111111111111111111111111111111  |
# D2 |  00000000000000000000000000000000  |
#
# D0 |  11111111111111111111111111111111  |  11  |  00100000000000000100000000000000  |  00000000000000000000000000000000  | ERRO
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
#
# Testes Efetuados  = 8
# Erros Encontrados = 8
```

Figura 32 Mux 4 32 - clk 10

Subindo o clock para 11, o modulo funcionou perfeitamente.

```
# D0 |  00000000000000000000000000000000  |  10  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
# D3 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  11  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  00000000000000000000000000000000  |  00  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  01  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  11111111111111111111111111111111  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  10  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  00000000000000000000000000000000  |
# D3 |  11111111111111111111111111111111  |
#
# D0 |  11111111111111111111111111111111  |  11  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# D1 |  11111111111111111111111111111111  |
# D2 |  11111111111111111111111111111111  |
# D3 |  00000000000000000000000000000000  |
#
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 33 Mux 4 32 - clk 11

Abaixo podemos ver os sinais de entrada e saída.

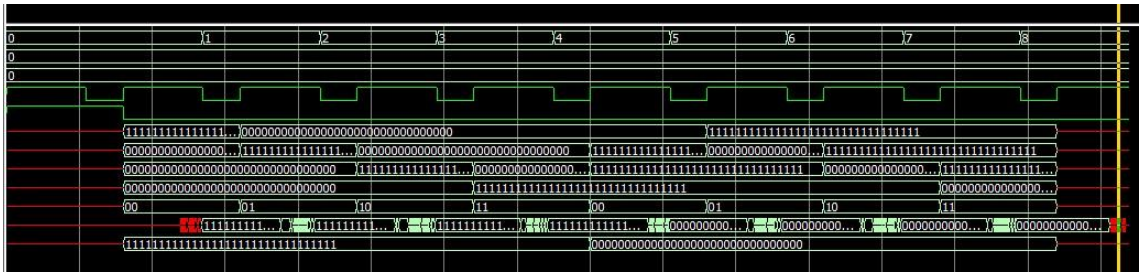


Figura 34 Mux 4 32 - Sinais de entrada e saída.

6 Mux 8:1 32 bits

Agora vamos estender as entradas de saídas do mux4 (criado anteriormente) para 32 bits. As mudanças são muito poucas e podem ser observadas principalmente no testbench e nos goldem vectors.

6.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
11111111111111111111111111111111_000_
11111111111111111111111111111111

00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
11111111111111111111111111111111_
00000000000000000000000000000000_001_
11111111111111111111111111111111

00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
00000000000000000000000000000000_
11111111111111111111111111111111_
00000000000000000000000000000000_
00000000000000000000000000000000_010_
11111111111111111111111111111111
```

Figura 35 Mux 8 32 bits - Goldem vectors

6.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
`timescale 1ns/100ps

module mux8_tb;

int counter, errors, aux_error;
logic clk,rst;
logic [31:0]d0;
logic [31:0]d1;
logic [31:0]d2;
logic [31:0]d3;
logic [31:0]d4;
logic [31:0]d5;
logic [31:0]d6;
logic [31:0]d7;
logic [2:0]s;
logic [31:0]y,y_esperado;

logic [290:0]vectors[8];

mux8 dut(.d7(d7), .d6(d6), .d5(d5), .d4(d4), .d3(d3), .d2(d2), .d1(d1), .d0(d0), .s(s), .y(y));

initial begin
    $display("Iniciando Testbench");
    $display("   |                D                   |   S   |                Y                  |            Y_esperado            |");
    $display("   |------------------------------------|-------|-----------------------------------|----------------------------------|");
    $readmemb("mux8_tv.tv",vectors);
    counter=0; errors=0;
    rst = 1;
    #40;
    rst = 0;
end

always begin
    clk=1; #30;
    clk=0; #10;
end

always @(posedge clk)
    if(~rst)
    begin
        //Atribui valores do vetor nas entradas do DUT e nos valores esperados
        {d7, d6, d5, d4, d3, d2, d1, d0, s, y_esperado} = vectors[counter];
    end

always @(negedge clk)   //Sempre (que o clock descer)
        if(~rst)
        begin
            aux_error = errors;

        assert (y === y_esperado)

        else  errors = errors + 1;

        if(aux_error === errors)begin
            $display("D0 |  %b  |  %b  |  %b  |  %b  | OK",   d0, s, y, y_esperado);
            $display("D1 |  %b  |",   d1);
            $display("D2 |  %b  |",   d2);
            $display("D3 |  %b  |",   d3);
            $display("D4 |  %b  |",   d4);
            $display("D5 |  %b  |",   d5);
            $display("D6 |  %b  |",   d6);
            $display("D7 |  %b  |",   d7);
            $display(" ");
        end
        else begin
            $display("D0 |  %b  |  %b  |  %b  |  %b  | ERRO", d0, s, y, y_esperado);
            $display("D1 |  %b  |",   d1);
            $display("D2 |  %b  |",   d2);
            $display("D3 |  %b  |",   d3);
            $display("D4 |  %b  |",   d4);
            $display("D5 |  %b  |",   d5);
            $display("D6 |  %b  |",   d6);
            $display("D7 |  %b  |",   d7);
            $display(" ");
        end
```

Figura 36 Testbench

6.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Temos sete entradas de 32 bits, uma entrada de controle S de 3 bits e uma saída de 1 bits (y).

```
module mux8(input logic [31:0]d7, input logic [31:0]d6, input logic [31:0]d5, input logic [31:0]d4, input logic [31:0]d3,
            input logic [31:0]d2, input logic [31:0]d1, input logic [31:0]d0, input logic [2:0]s  , output logic [31:0]y);

    logic [31:0]y0;
    logic [31:0]y1;

    mux4 m4_0(d3, d2, d1, d0, s[1:0], y0);
    mux4 m4_1(d7, d6, d5, d4, s[1:0], y1);

    mux2 m2_0(y1, y0, s[2], y);

endmodule
```

Figura 37 Modelo duv

6.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
#    |                D                  |   S   |                Y                 |            Y_esperado           |
#    |-----------------------------------|-------|----------------------------------|---------------------------------|
# D0 |  11111111111111111111111111111111 |  000  |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# D1 |  00000000000000000000000000000000 |
# D2 |  00000000000000000000000000000000 |
# D3 |  00000000000000000000000000000000 |
# D4 |  00000000000000000000000000000000 |
# D5 |  00000000000000000000000000000000 |
# D6 |  00000000000000000000000000000000 |
# D7 |  00000000000000000000000000000000 |
#
# D0 |  00000000000000000000000000000000 |  001  |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# D1 |  11111111111111111111111111111111 |
# D2 |  00000000000000000000000000000000 |
# D3 |  00000000000000000000000000000000 |
# D4 |  00000000000000000000000000000000 |
# D5 |  00000000000000000000000000000000 |
# D6 |  00000000000000000000000000000000 |
# D7 |  00000000000000000000000000000000 |
#
# D0 |  00000000000000000000000000000000 |  010  |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# D1 |  00000000000000000000000000000000 |
# D2 |  11111111111111111111111111111111 |
# D3 |  00000000000000000000000000000000 |
# D4 |  00000000000000000000000000000000 |
# D5 |  00000000000000000000000000000000 |
# D6 |  00000000000000000000000000000000 |
# D7 |  00000000000000000000000000000000 |
```

```
# Testes Efetuados   = 8
# Erros Encontrados = 0
```

Também podemos observar a visualização RTL do modulo criado.

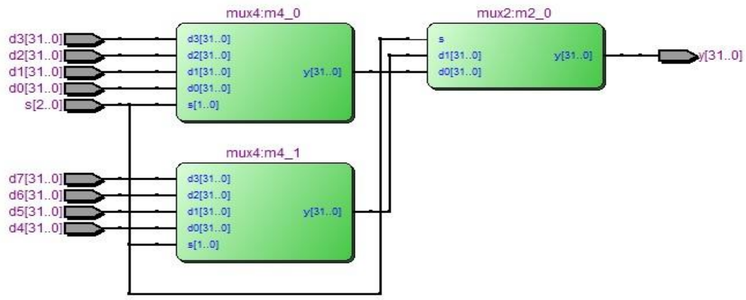


Figura 38 Rtl view

6.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 8. Como podemos ver, 4 erros foram encontrados.

```
#
# D0 |  00000000000000000000000000000000 |  111  |  11111111111111111111111111111111 |  11111111111111111111111111111111 | OK
# D1 |  00000000000000000000000000000000 |
# D2 |  00000000000000000000000000000000 |
# D3 |  00000000000000000000000000000000 |
# D4 |  00000000000000000000000000000000 |
# D5 |  00000000000000000000000000000000 |
# D6 |  00000000000000000000000000000000 |
# D7 |  11111111111111111111111111111111 |
#
# Testes Efetuados  = 8
# Erros Encontrados = 1
```

Figura 39 clk 8

```
# D0 |  00000000000000000000000000000000  |  110  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
# D3 |  00000000000000000000000000000000  |
# D4 |  00000000000000000000000000000000  |
# D5 |  00000000000000000000000000000000  |
# D6 |  11111111111111111111111111111111  |
# D7 |  00000000000000000000000000000000  |
#
# D0 |  00000000000000000000000000000000  |  111  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# D1 |  00000000000000000000000000000000  |
# D2 |  00000000000000000000000000000000  |
# D3 |  00000000000000000000000000000000  |
# D4 |  00000000000000000000000000000000  |
# D5 |  00000000000000000000000000000000  |
# D6 |  00000000000000000000000000000000  |
# D7 |  11111111111111111111111111111111  |
#
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 40clk 9

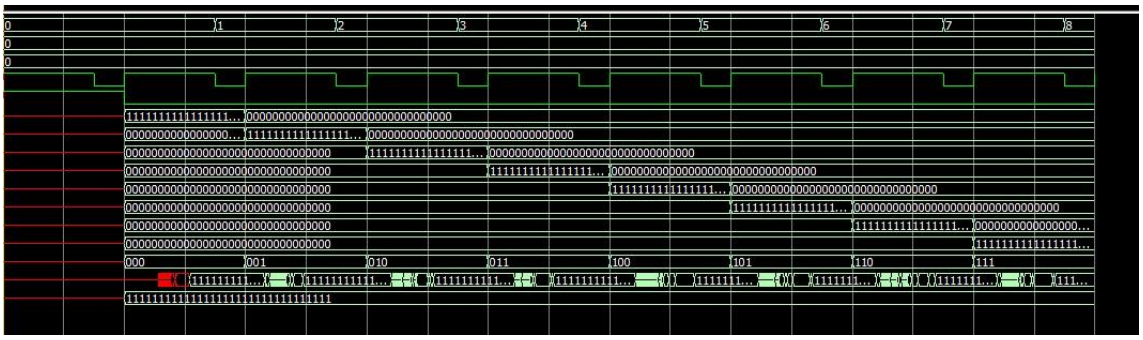


Figura 41 Sinias de entrada e saida

7 Flopr 32 bits

Agora vamos estender as entradas de saídas do flopr (criado anteriormente) para 32 bits. As mudanças são muito poucas e podem ser observadas principalmente no testbench e nos goldem vectors.

7.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
0_1_00000000000000000000000000000000_
00000000000000000000000000000000
1_1_00000000000000000000000000000000_
00000000000000000000000000000000
0_1_11111111111111111111111111111111_
00000000000000000000000000000000
1_1_11111111111111111111111111111111_
00000000000000000000000000000000
0_0_11111111111111111111111111111111_
00000000000000000000000000000000
1_0_11111111111111111111111111111111_
11111111111111111111111111111111
0_0_00000000000000000000000000000000_
11111111111111111111111111111111
1_0_00000000000000000000000000000000_
00000000000000000000000000000000
```

Figura 42 Flopr 32 bits - Goldem vectors

7.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
`timescale 1ns/100ps

module floprr32_tb;

int counter, errors, aux_error;
logic clk,rst;
logic clk2,rst2;
logic [31:0]d;
logic [31:0]q, q_esperado;
logic [65:0]vectors[8];

floprr32 dut(.clk(clk2), .reset(rst2), .d(d), .q(q));

initial begin
   $display("Iniciando Testbench");
   $display("| CLK | RST |              D                   |                Q                 |            Q_esperado            |");
   $display("|-----|-----|----------------------------------|----------------------------------|----------------------------------|");
   $readmemb("floprr32_tv.tv",vectors);
   counter=0; errors=0;
   rst = 1;
   #15;
   rst = 0;
end

always begin

   clk=1; #10;
   clk=0; #5;

end

always @(posedge clk)
   if(~rst)
   begin
      {clk2,rst2,d,q_esperado} = vectors[counter];
   end

always @(negedge clk)   //Sempre (que o clock descer)
      if(~rst)
      begin
         aux_error = errors;
         assert (q === q_esperado)
      else
         errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado

      if(aux_error === errors)
         $display("|  %b    |  %b    |  %b  |  %b  |  %b  | OK", clk2, rst2, d, q, q_esperado);
      else
         $display("|  %b    |  %b    |  %b  |  %b  |  %b  | ERRO", clk2, rst2, d, q, q_esperado);

         counter++;  //Incrementa contador dos vertores de teste

         if(counter == $size(vectors)) //Quando os vetores de teste acabarem
         begin
            $display("Testes Efetuados  = %0d", counter);
            $display("Erros Encontrados = %0d", errors);
            #15
            $stop;
         end

      end

endmodule
```

Figura 43flopr 32 - Testbench

7.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Temos clk, reset, uma entrada de dados de 32 bits e uma saída de 32 bits (q).

```
module floprr32(input logic clk, input logic reset, input logic [31:0]d, output logic [31:0]q);

//Reset Assincrono
always_ff @(posedge clk, posedge reset)

   if (reset) q <= 32'b0;
   else q <= d;

endmodule

```

Figura 44 flopr - modelo duv

7.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
# | CLK | RST |                D                 |                Q                 |           Q_esperado             |
# |-----|-----|----------------------------------|----------------------------------|----------------------------------|
# |  0  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  0  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  0  | 11111111111111111111111111111111 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
# |  0  |  0  | 00000000000000000000000000000000 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
# |  1  |  0  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 45 flopr - Simulação rtl

Também podemos observar a visualização RTL do modulo criado.

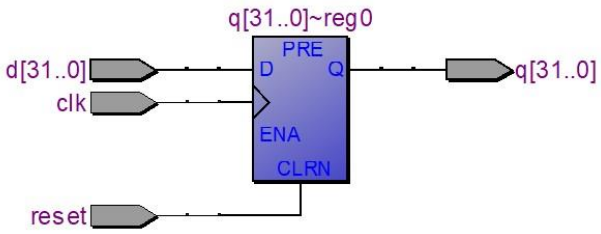


Figura 46 flopr - rtl view

7.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 9. Como podemos ver, 4 erros foram encontrados.

```
# Iniciando Testbench
# | CLK | RST |                D                 |                Q                 |           Q_esperado             |
# |-----|-----|----------------------------------|----------------------------------|----------------------------------|
# |  0  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  0  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  0  | 11111111111111111111111111111111 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
# |  0  |  0  | 00000000000000000000000000000000 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
# |  1  |  0  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# Testes Efetuados  = 8
# Erros Encontrados = 0
```

Figura 47 clk 9

```
# Iniciando Testbench
# | CLK | RST |                D                 |                Q                 |           Q_esperado             |
# |-----|-----|----------------------------------|----------------------------------|----------------------------------|
# |  0  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 00000000000000000000000000000000 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  1  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  0  |  0  | 11111111111111111111111111111111 | 00000000000000000000000000000000 | 00000000000000000000000000000000 | OK
# |  1  |  0  | 11111111111111111111111111111111 | 11111111111111111111101111111111 | 11111111111111111111111111111111 | ERRO
# |  0  |  0  | 00000000000000000000000000000000 | 11111111111111111111111111111111 | 11111111111111111111111111111111 | OK
# |  1  |  0  | 00000000000000000000000000000000 | 00000000000000000000010000000000 | 00000000000000000000000000000000 | ERRO
# Testes Efetuados  = 8
# Erros Encontrados = 2
```

Figura 48clk 10

Abaixo podemos ver os sinais de entrada e saída.

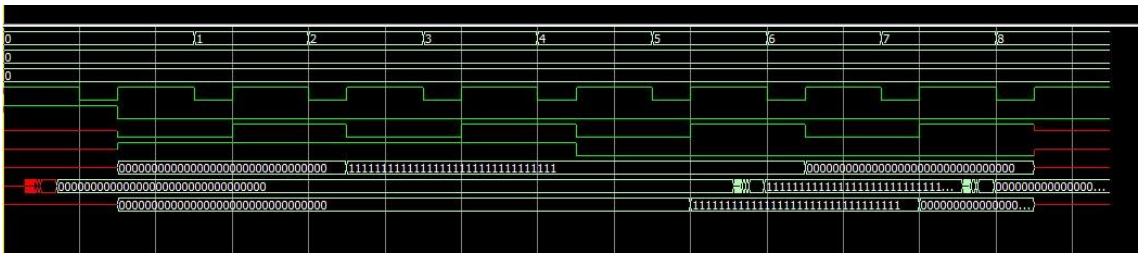


Figura 49 flopr - Sinais de entrada e saida

8 Flopenr 32 bits

Agora vamos estender as entradas de saídas do flopenr (criado anteriormente) para 32 bits. As mudanças são muito poucas e podem ser observadas principalmente no testbench e nos goldem vectors

8.1 Goldem vector

Abaixo podemos ver os *goldem vectors* gerados a partir do *goldem model*

```
0_1_0_00000000000000000000000000000000_
00000000000000000000000000000000
1_1_0_00000000000000000000000000000000_
00000000000000000000000000000000
0_1_1_00000000000000000000000000000000_
00000000000000000000000000000000
1_1_1_00000000000000000000000000000000_
00000000000000000000000000000000
0_1_0_11111111111111111111111111111111_
00000000000000000000000000000000
1_1_0_11111111111111111111111111111111_
00000000000000000000000000000000
0_1_1_11111111111111111111111111111111_
00000000000000000000000000000000
1_1_1_11111111111111111111111111111111_
00000000000000000000000000000000
0_0_0_00000000000000000000000000000000_
00000000000000000000000000000000
1_0_0_00000000000000000000000000000000_
00000000000000000000000000000000
0_0_1_00000000000000000000000000000000_
00000000000000000000000000000000
1_0_1_00000000000000000000000000000000_
00000000000000000000000000000000
0_0_0_11111111111111111111111111111111_
00000000000000000000000000000000
1_0_0_11111111111111111111111111111111_
00000000000000000000000000000000
0_0_1_11111111111111111111111111111111_
00000000000000000000000000000000
1_0_1_11111111111111111111111111111111_
11111111111111111111111111111111
```

Figura 50 Flopenr 32 bits - Goldem vectors

8.2 Testbench

Abaixo podemos ver o testbench utilizado para testar o modulo.

```
`timescale 1ns/100ps

module flopenr32_tb;

int counter, errors, aux_error;
logic clk,rst;
logic clk2,rst2;
logic [31:0]d;
logic en;
logic [31:0]q, q_esperado;
logic [66:0]vectors[16];

flopenr32 dut(.clk(clk2), .reset(rst2), .en(en), .d(d), .q(q));

initial begin
    $display("Iniciando Testbench");
    $display("|  CLK  |  RST  |  EN  |                D                |                Q                |             Q_esperado           |");
    $display("|-------|-------|------|---------------------------------|---------------------------------|----------------------------------|");
    $readmemb("flopenr32_tv.tv",vectors);
    counter=0; errors=0;
    rst = 1;
    #11;
    rst = 0;
end

always begin
    clk=1; #6;
    clk=0; #5;
end

always @(posedge clk)
    if(~rst)
    begin
        {clk2,rst2,en,d,q_esperado} = vectors[counter];
    end

always @(negedge clk)   //Sempre (que o clock descer)
     if(~rst)
      begin
         aux_error = errors;
         assert (q === q_esperado)

      else errors = errors + 1; //Incrementa contador de erros a cada bit errado encontrado

      if(aux_error === errors)
         $display("|   %b   |   %b   |  %b  |  %b  |  %b  |  %b  |      %b       | OK", clk2, rst2, en, d, q, q_esperado);
      else
         $display("|   %b   |   %b   |  %b  |  %b  |  %b  |  %b  |      %b       | ERRO", clk2, rst2, en, d, q, q_esperado);

         counter++;  //Incrementa contador dos vertores de teste

         if(counter == $size(vectors)) //Quando os vetores de teste acabarem
         begin
            $display("Testes Efetuados  = %0d", counter);
            $display("Erros Encontrados = %0d", errors);
            #15
            $stop;
         end

      end

endmodule
```

Figura 51 flopenr - testbench

8.3 Modelo duv

Abaixo podemos ver o modelo duv do modulo. Temos clk, reset, en, uma entrada de dados de 32 bits e uma saída de 32 bits (q).

```
module flopenr32(input logic clk, reset, en, input logic [31:0]d, output logic [31:0]q);

//Reset Assincrono
always @(posedge clk, posedge reset)

   if (reset) q <= 32'b0;
   else if (en) q <= d;

endmodule
```

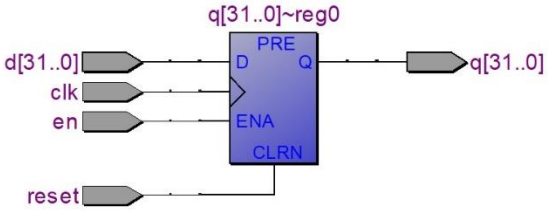
Figura 52 flopenr - duv

8.4 Simulação RTL

Abaixo podemos ver o transcript da simulação RTL. Como podemos ver, o modulo está se comportando como o esperado.

```
# Iniciando Testbench
# |  CLK  |  RST  |  EN  |                D                 |                Q                 |            Q_esperado            |
# |-------|-------|------|----------------------------------|----------------------------------|----------------------------------|
# |   0   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  11111111111111111111111111111111  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# Testes Efetuados  = 16
# Erros Encontrados = 0
```

Também podemos observar a visualização RTL do modulo criado.



8.5 Simulação Gate Level

Iniciamos a simulação gate level com o clock em 10. Como podemos ver, 1 erro foi encontrado.

```
# Iniciando Testbench
# |  CLK  |  RST  |  EN  |                D                 |                Q                 |            Q_esperado            |
# |-------|-------|------|----------------------------------|----------------------------------|----------------------------------|
# |   0   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  11111111111111111111111111111111  |  11110111111111111111111111111111  |  11111111111111111111111111111111  | ERRO
# Testes Efetuados  = 16
# Erros Encontrados = 1
```

Figura 53clk 10

```
# Iniciando Testbench
# |  CLK  |  RST  |  EN  |                D                 |                Q                 |            Q_esperado            |
# |-------|-------|------|----------------------------------|----------------------------------|----------------------------------|
# |   0   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   1   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  00000000000000000000000000000000  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  0   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   0   |   0   |  1   |  11111111111111111111111111111111  |  00000000000000000000000000000000  |  00000000000000000000000000000000  | OK
# |   1   |   0   |  1   |  11111111111111111111111111111111  |  11111111111111111111111111111111  |  11111111111111111111111111111111  | OK
# Testes Efetuados  = 16
# Erros Encontrados = 0
```

Figura 54clk 11

Abaixo podemos ver os sinais de entrada e saída.

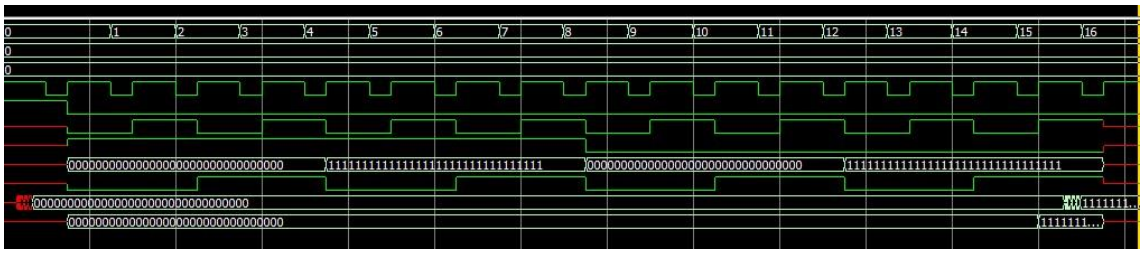


Figura 55 flopenr - Sinais de entrada e saida