UNIVERSIDADE FEDERAL DE ITAJUBÁ – UNIFEI

Laura Pellizari Pereira - 2019018756

Itajubá – MG
2020.1

Projeto de circuitos para interface com ULA

Relatório técnico de projeto apresentado como requisito para obtenção de aprovação na disciplina ELTD12 - Laboratório de Eletrônica Digital II na Universidade Federal de Itajubá.

Prof. Dr. Gabriel Fanelli

Sumário

1 INTRODUÇÃO

A unidade lógica e aritmética é um circuito digital que é responsável por operações de adição e booleana, podendo ser utilizada microcontroladores ou disponibilizada em forma de circuito integrado.

Dessa forma, este relatório demonstra o desenvolvimento de um projeto de implementação de um circuito digital para interface com uma ULA, a partir do software Quartus II, assim como simulações e anotações de projeto.

2. DESENVOLVIMENTO DO PROJETO

O projeto tem como objetivo a implementação em linguagem Verilog (linguagem de descrição de hardware) do seguinte circuito:

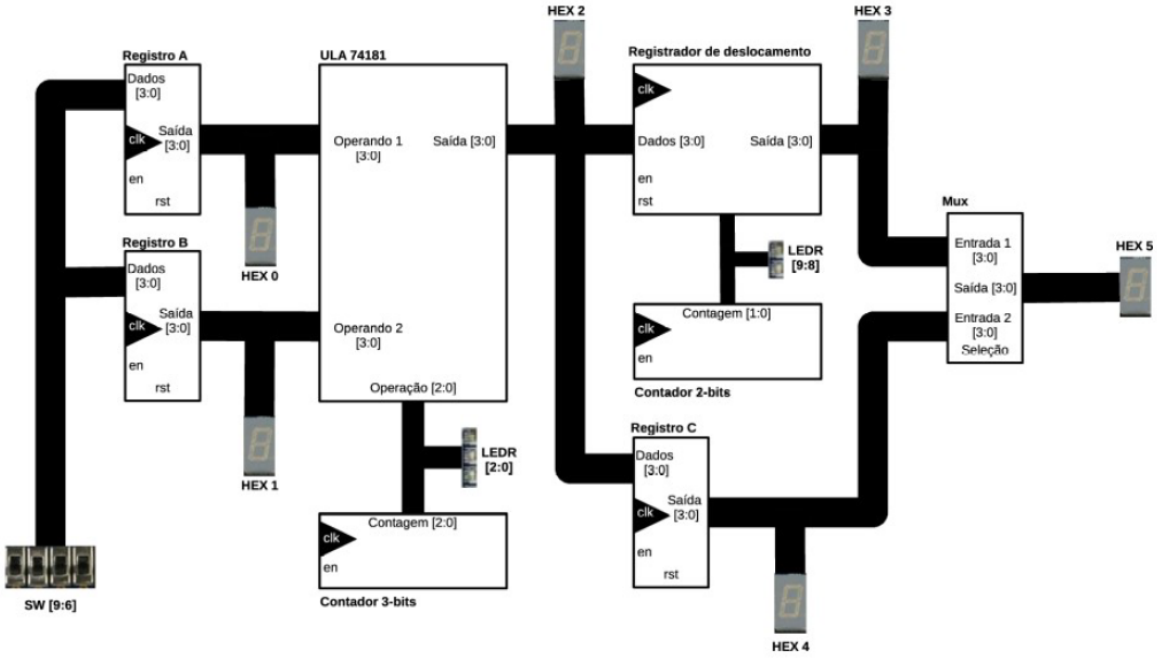


Figura 1 - Esquema do projeto a ser desenvolvido

Primeiramente, implementou – se os módulos individuais dos componentes principais do projeto, sendo assim, partiu – se da simulação dos arquivos: registrador.v, contador_2B.V, contador_3b.v, ula.v e reg_desl.v, assim como seus respectivos arquivos de testbench.

2.1 Módulo registrador.v

Para o módulo do registrador.v foi desenvolvimento o seguinte código:

```
1   module registrador (rst, clk, en, d, q);
2
3   input clk, rst, en;
4   input [3:0] d;
5   output reg [3:0] q;
6
7   always @ (posedge clk, posedge rst)
8   begin
9     if (rst)
10    begin
11      q <= 4'd0;
12    end
13    else if (en)
14    begin
15      q <= d;
16    end
17  end
18
19  endmodule
```

Figura 2 - código do registrador

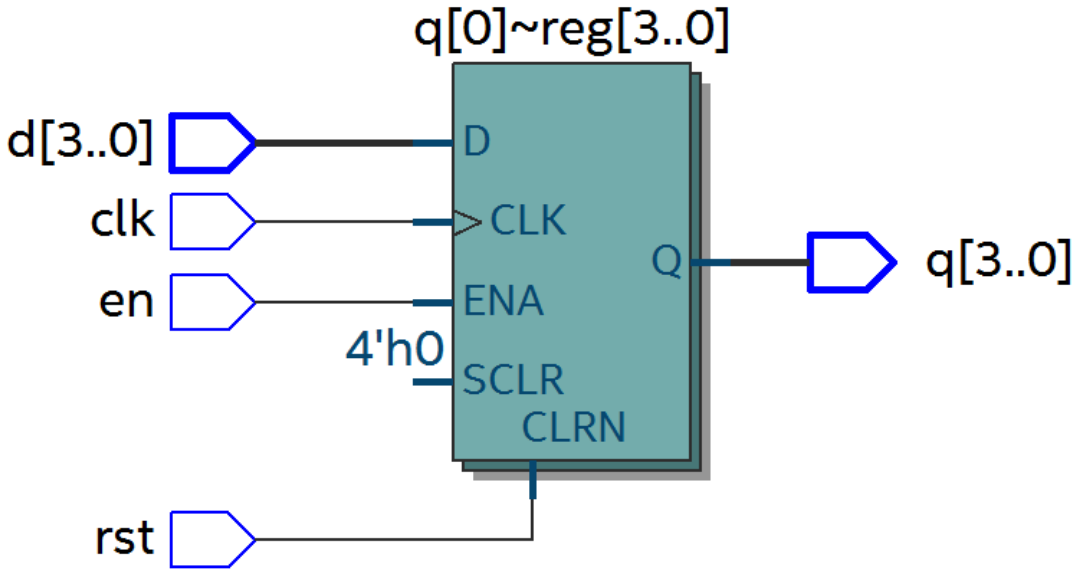


Figura 3 - RTL Registrador

Em que observa - se a operação na borda de subida do clock e presença de reset assíncrono.

A seguir observa o resultado obtido com a simulação por meio do testbench.

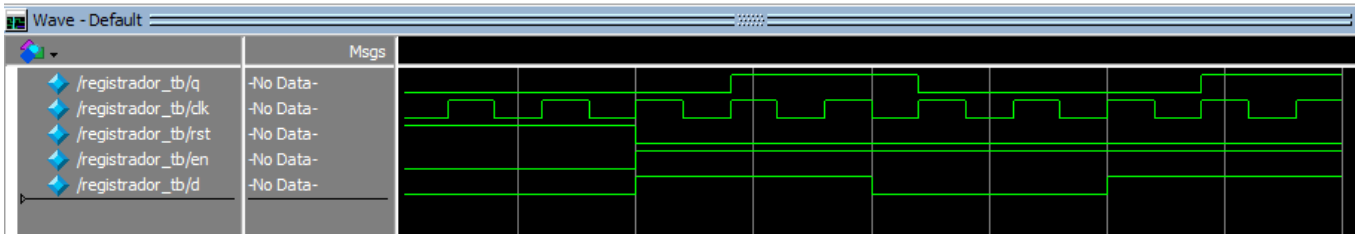


Figura 4 - Testebench do registrador

É possível perceber coerência com o esperado, pois como programado para obter mudanças na borda de subida do clock, a saída q repete o comportamento da entrada d.

2.2 Módulo contador_2b.v e módulo contador_3b.v

Para os módulos dos contadores o raciocínio utilizado foi o mesmo, a variar apenas a quantidade de bits em questão. As imagens abaixo demonstram o código e a visualização de cada contador.

```
1   module contador_2b (clk, rst, en, out, led0, led1);
2
3   input rst, clk, en;
4   output reg [1:0] out;
5   output led0, led1;
6
7   assign led0 = out[0], led1 = out[1];
8
9
10  always @ (posedge clk, posedge rst)
11  begin
12    if (rst)
13    begin
14      out <= 2'b0;
15    end
16     else if (en)
17     begin
18       out <= out + 1'b1;
19     end
20
21  end
22
23  endmodule
```

Figura 5 - Código do contador de 2 bits

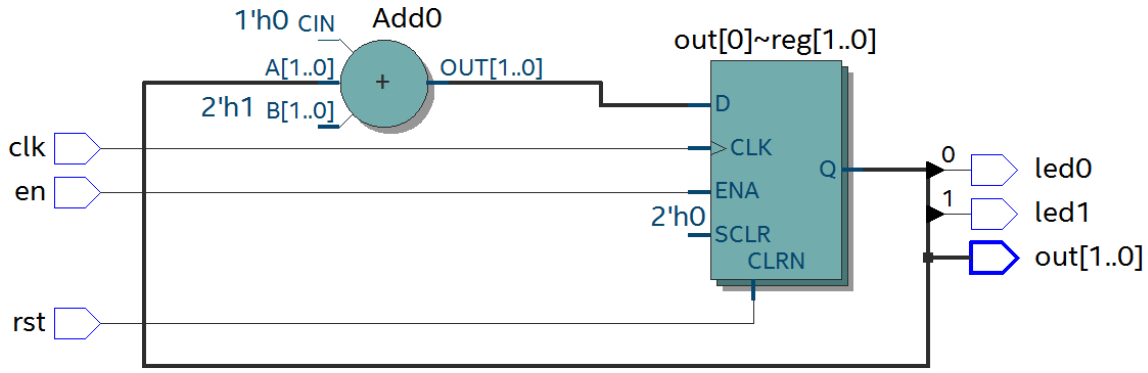


Figura 6 - RTL contador de 2 bits

```
1   module contador_3b (clk, rst, en, out, led0, led1, led2);
2
3   input rst, clk, en;
4   output reg [2:0] out;
5   output led0, led1, led2;
6
7   assign led0 = out[0], led1 = out[1], led2 = out[2];
8
9
10  always @ (posedge clk, posedge rst)
11  begin
12    if (rst)
13    begin
14      out <= 3'b0;
15    end
16     else if (en)
17     begin
18       out <= out + 1'b1;
19     end
20  end
21
22  endmodule
```

Figura 7 - Código do contador de 3 bits

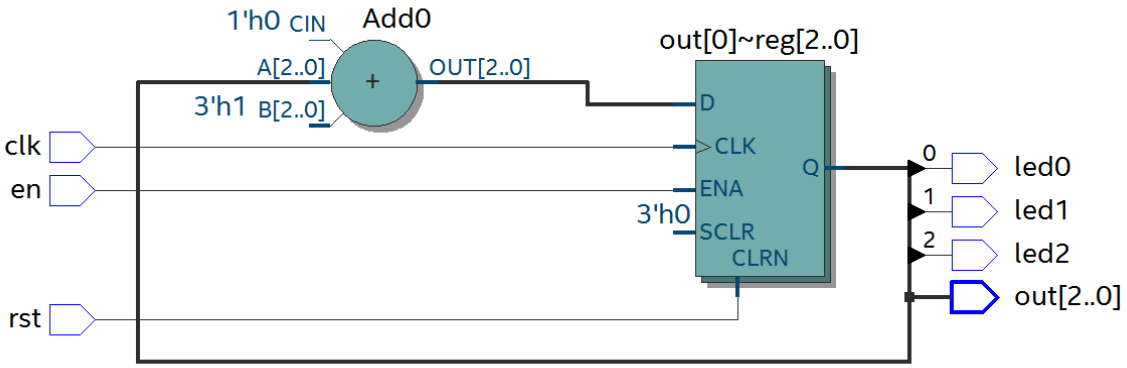


Figura 8 - RTL Contador de 3 bits

O código foi baseado na ideia de variação na borda de subida do clock, reset assíncrono e enquanto o enable estiver acionado, a cada borda de subida do clock, é adicionado 1 bit de contagem ao resultado de saída do circuito.

A seguir, os resultados obtidos com a simulação do testebench de cada um dos arquivos.

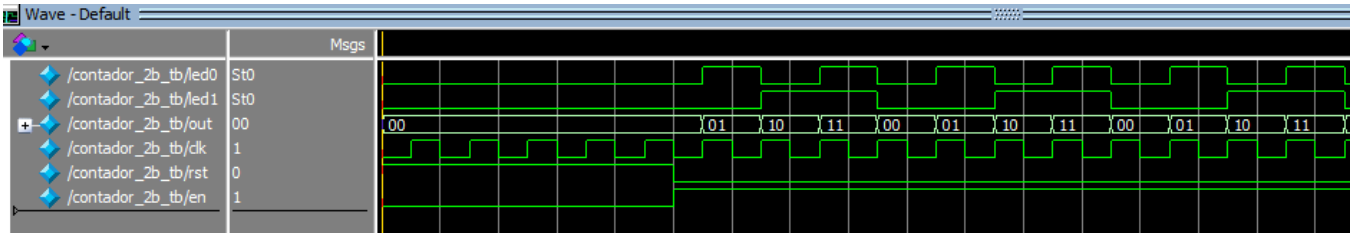


Figura 9 - Resultado do testebench do contador de 2 bits

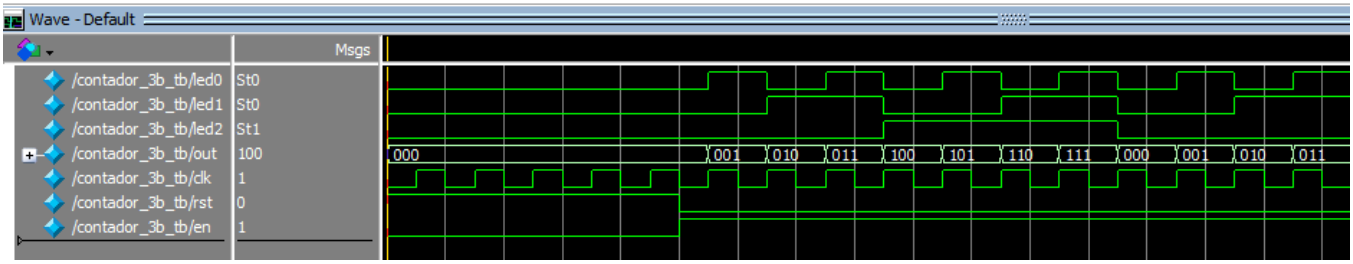


Figura 10 - Resultado do testebench do contador de 3 bits

A simulação demonstrou os resultados esperados: contagem crescente até 3 para o contador de 2 bits e até 7 para o contador de 3 bits, que pode ser observada tanto no vetor count quanto nos leds.

2.3 Módulo ULA.v

Para a construção do módulo ULA.v foi visitado o datasheet disponibilizado nos requisitos de projeto. Dessa forma, associado as informações iniciais de apenas 8 operações para a ULA, sendo elas não inclusas operações de soma, foram escolhidas as seguintes operações para a ULA:

0 F = A

1 F = A’

2 F = B

3 F = B’

4 F = A * B

5 F = A’ * B

6 F = A * B’

7 F = A’ * B’

O código, a visualização RTL e o resultado do testebench são demonstrados abaixo.

```
 1   module ULA (op_a, op_b, sel_ULA, out);
 2
 3   input [3:0] op_a, op_b;
 4   input [2:0] sel_ULA;
 5   output reg [3:0] out;
 6
 7   reg [3:0] out_ULA;
 8
 9   always @(*)
10   begin
11      case (sel_ULA)
12        3'b000: begin out_ULA <= op_a;          end
13        3'b001: begin out_ULA <= ~op_a;         end
14        3'b010: begin out_ULA <= op_b;          end
15        3'b011: begin out_ULA <= ~op_b;         end
16        3'b100: begin out_ULA <= op_a & op_b;   end
17        3'b101: begin out_ULA <= ~op_a & op_b;  end
18        3'b110: begin out_ULA <= op_a & ~op_b;  end
19        3'b111: begin out_ULA <= ~op_a & ~op_b; end
20       default: begin out_ULA <= out_ULA;  end
21      endcase
22
23     out <= out_ULA;
24   end
25
26   endmodule
```

Figura 11 - Código da ULA

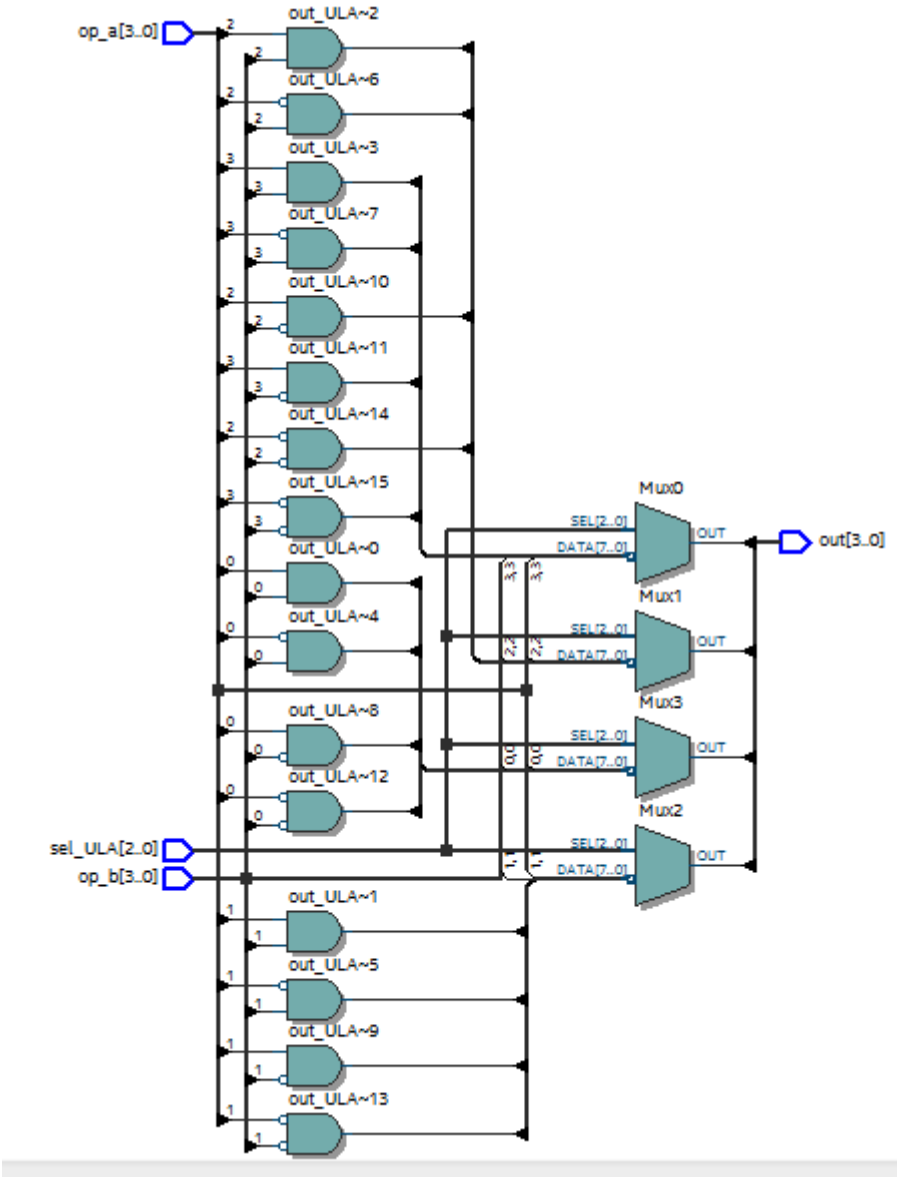


Figura 12 - RTL ULA

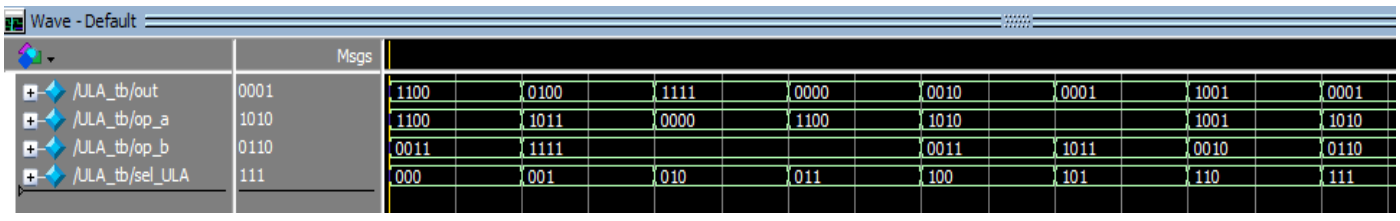


Figura 13 - Resultado do Testebench da ULA

O resultado obtido é satisfatório, pois todas as condições ocorrem como esperado, obedecendo todas as seleções da ULA.

2.4 Módulo reg_desl.v

O módulo do registrador de deslocamento foi baseado no módulo do registrador, com a inclusão de um case para obedecer à tabela abaixo:

Palavra	Descrição
00	Carregar os dados no registrador.
01	Deslocamento à direita com preenchimento de zeros.
10	Deslocamento à esquerda com preenchimento de zeros.
11	Deslocamento à direita mantendo o bit mais significativo.

Figura 14 - Comandos do Registrador de Deslocamento

Utilizou – se a técnica de concatenação, observada no código desenvolvido a seguir:

```
1   module reg_desloc (rst, clk, en, d, out, sel);
2
3   input rst, clk, en;
4   input [3:0] d;
5   input [1:0] sel;
6   output reg [3:0] out;
7
8   reg [3:0] q;
9
10  always @ (posedge clk, posedge rst)
11  begin
12    if (rst)
13    begin
14      q <= 4'b0;
15    end
16    else if (en)
17     begin
18       case ({sel})
19        2'b00: begin q <= d;                end
20        2'b01: begin q <= {1'b0, d[3:1]}; end
21        2'b10: begin q <= {d[3:1], 1'b0}; end
22        2'b11: begin q <= {d[3], d[3:1]}; end
23       default begin q <= q;                end
24        endcase
25     end
26     out <= q;
27  end
28
29  endmodule
```

Figura 15 - Código do Registrador de Deslocamento

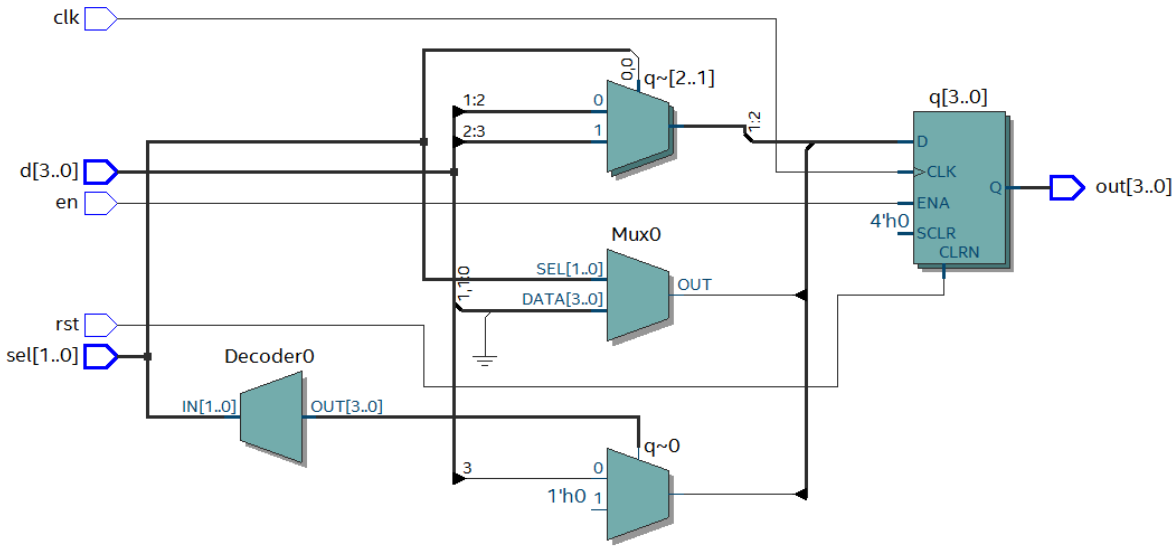


Figura 16 - RTL Registrador de deslocamento

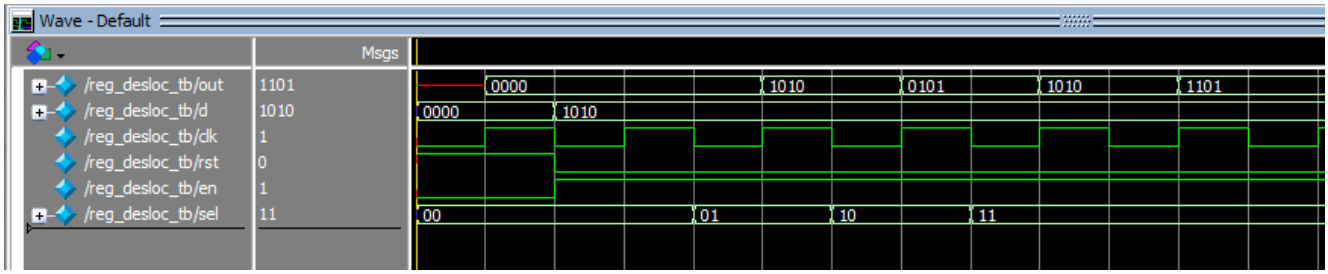


Figura 17 - Resultado do Testebench Registrador de deslocamento

O resultado obtido no testebench é coerente ao esperado, exceto pelo atraso de uma borda de subida de clock. As concatenações ocorrem como esperado.

2.5 Módulo sistema.v

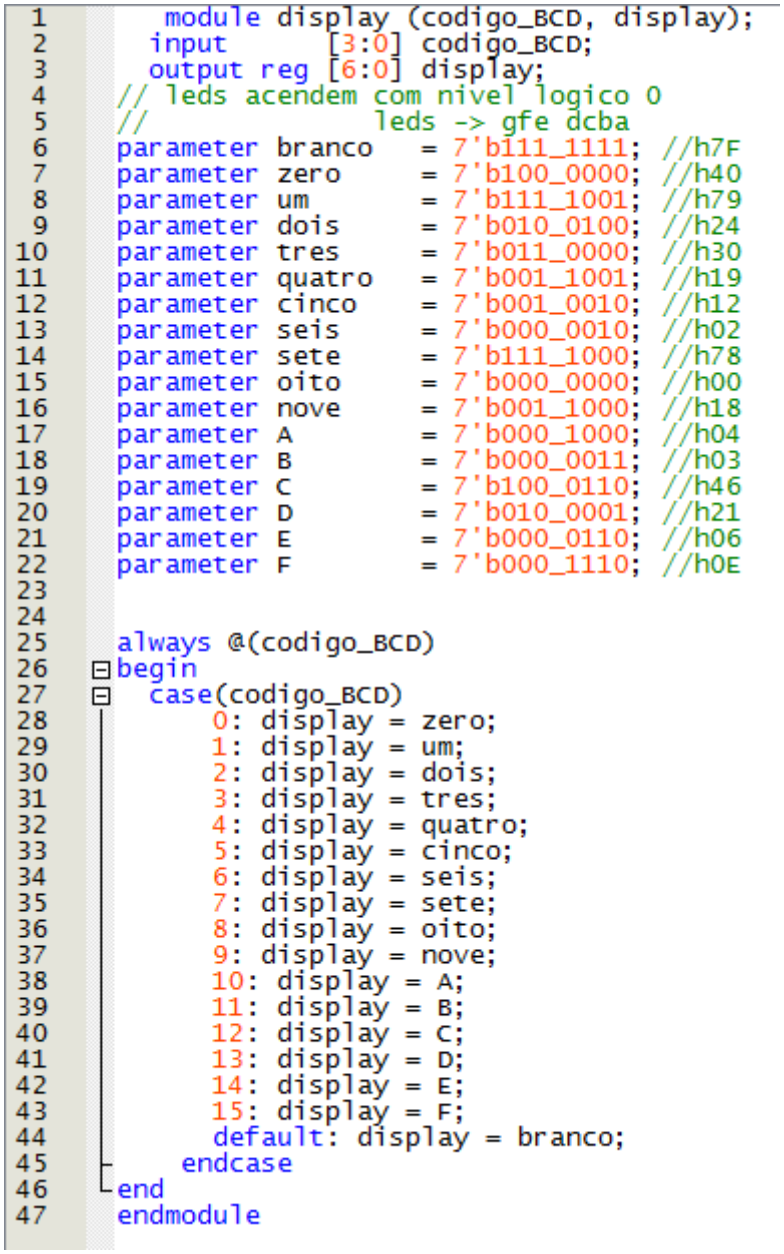
A construção do módulo do sistema deu – se a partir da união dos módulos apresentados antes.

Um arquivo referente a um multiplexador foi criado dentro do módulo sistema. O código demonstra um simples multiplexador 2x1 de 4 bits.

```
1   module  mux (A, B, sel_mux, out);
2
3   input [3:0] A, B;
4   input sel_mux;
5   output reg [3:0] out;
6
7   always @( A or B or sel_mux )
8   begin
9       if (sel_mux == 1'b0)
10      begin
11          out <= A;
12      end
13      else
14      begin
15          out <= B;
16      end
17  end
18
19  endmodule
```

Figura 18 - Código do Mux

Dentro do mesmo módulo, o arquivo disponibilizado para a decodificação do display de 7 segmentos foi modificado com o objetivo de inserir cinco palavras (A, B, C, D, E, F). A modificação do código foi baseada na informação de que os leds do display de 7 segmentos acendem com nível lógico 0. A seguir é apresentado o código modificado.



```
1   module display (codigo_BCD, display);
2     input      [3:0] codigo_BCD;
3     output reg [6:0] display;
4   // leds acendem com nivel logico 0
5   //                leds -> gfe dcba
6   parameter branco   = 7'b111_1111; //h7F
7   parameter zero     = 7'b100_0000; //h40
8   parameter um       = 7'b111_1001; //h79
9   parameter dois     = 7'b010_0100; //h24
10  parameter tres     = 7'b011_0000; //h30
11  parameter quatro   = 7'b001_1001; //h19
12  parameter cinco    = 7'b001_0010; //h12
13  parameter seis     = 7'b000_0010; //h02
14  parameter sete     = 7'b111_1000; //h78
15  parameter oito     = 7'b000_0000; //h00
16  parameter nove     = 7'b001_1000; //h18
17  parameter A        = 7'b000_1000; //h04
18  parameter B        = 7'b000_0011; //h03
19  parameter C        = 7'b100_0110; //h46
20  parameter D        = 7'b010_0001; //h21
21  parameter E        = 7'b000_0110; //h06
22  parameter F        = 7'b000_1110; //h0E
23
24
25  always @(codigo_BCD)
26  begin
27    case(codigo_BCD)
28        0: display = zero;
29        1: display = um;
30        2: display = dois;
31        3: display = tres;
32        4: display = quatro;
33        5: display = cinco;
34        6: display = seis;
35        7: display = sete;
36        8: display = oito;
37        9: display = nove;
38        10: display = A;
39        11: display = B;
40        12: display = C;
41        13: display = D;
42        14: display = E;
43        15: display = F;
44        default: display = branco;
45    endcase
46  end
47  endmodule
```

Figura 19 - Código Display modificado

O próximo passo do projeto foi a implementação de um arquivo de controle, baseado na construção de uma máquina de estados responsável pelo acionamento das entradas de enable dos registradores. O código da máquina é demonstrado abaixo.

```
module controle (clk, rst, state, out);

input clk, rst;

output reg [1:0] state;
output reg [5:0] out;

parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;

always @(state)
   begin
      case(state)
         s0:
         begin
            out <= 6'b000000;
         end
         s1:
         begin
            out <= 6'b100011;
         end
         s2:
         begin
            out <= 6'b010011;
         end
         s3:
         begin
            out <= 6'b001111;
         end
         default:
         begin
            out <= out;
         end
      endcase
   end

always @(posedge clk, posedge rst)
begin
   if(rst)
   begin
      state <= s0;
   end
   else
   begin
      case(state)
         s0:
         begin
            state <= s1;
         end
         s1:
         begin
            state <= s2;
         end
         s2:
         begin
            state <= s3;
         end
         s3:
         begin
            state <= s0;
         end
         default:
         begin
            state <= state;
         end
      endcase
   end
end
endmodule
```

Figura 20 - Código do Controle do sistema

A lógica utilizada para as saídas de cada estado foi a que cada estado é responsável pelo acionamento de um ou mais registradores. Dessa maneira, as saídas estão

distribuídas na seguinte ordem dos registradores: reg_a, reg_b, reg_c, reg_desl, contador_2b, contador_3b.

O agrupamento de cada valor do vetor de saída desse arquivo foi corretamente distribuído no arquivo sistema, como será demonstrado adiante.

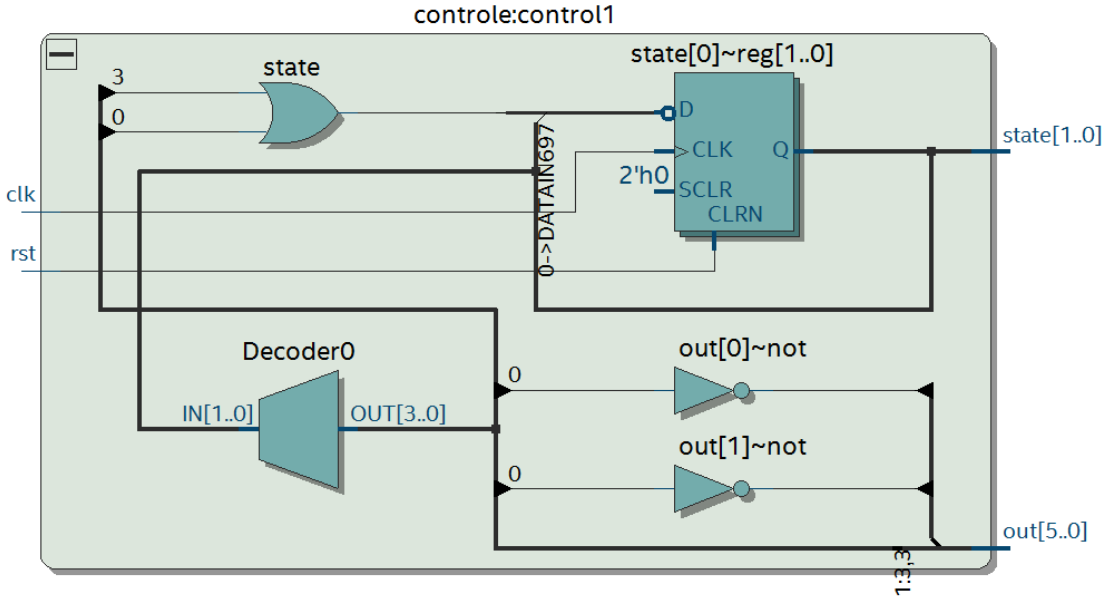


Figura 21 - RTL Controle

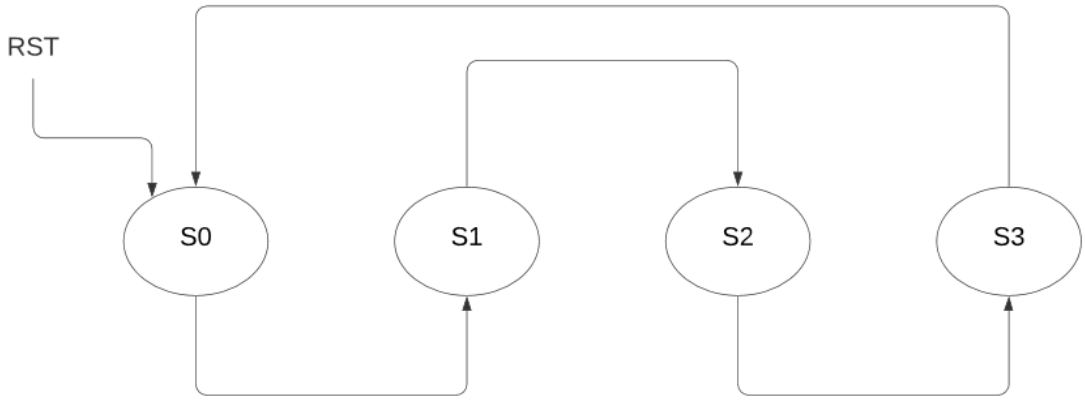


Figura 22 - Diagrama de Estados

Em seguida, os arquivos e módulos foram colocados em um arquivo apenas, denominado sistema.

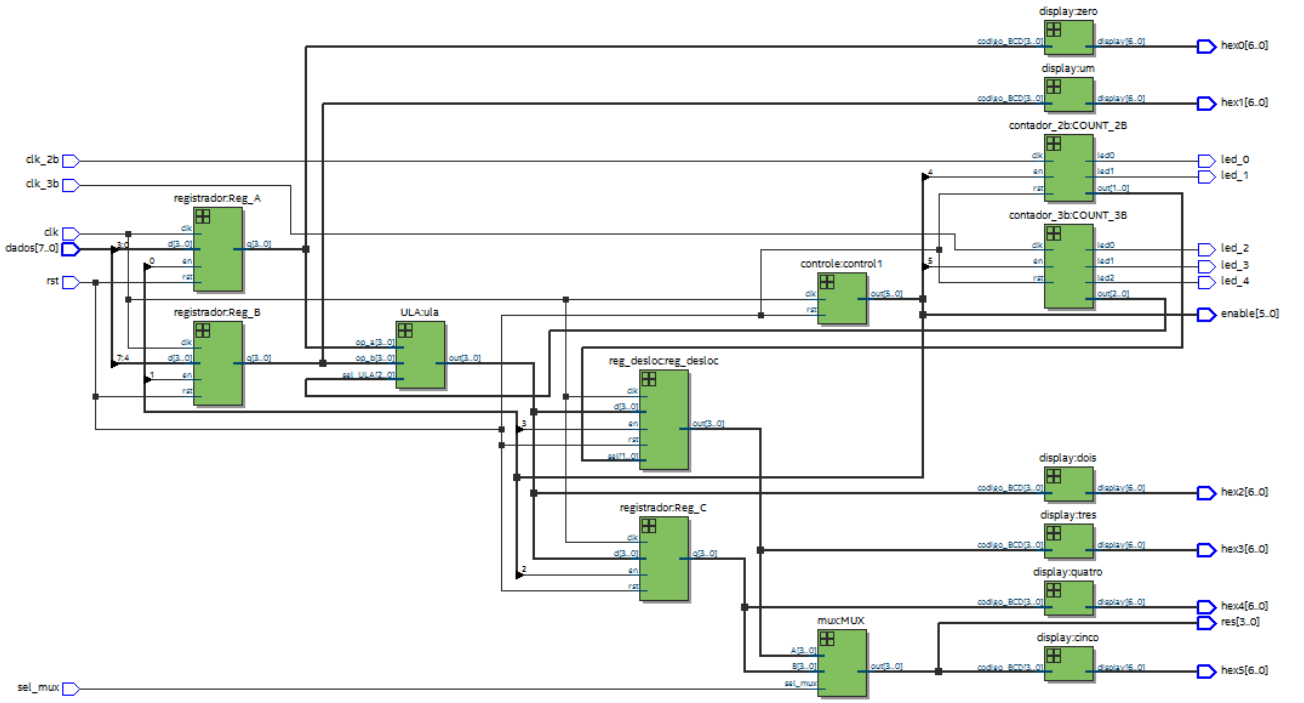
Tal arquivo é responsável por chamar de maneira estrutural todos os componentes do sistema, tais quais: Registrador A, Registrador B, Registrador C, Registrador de Deslocamento, Contador de 2 bits, Contador de 3 bits, ULA, Mux e os 6 displays de sinalização. Foram declarados fios de ligações entre as declarações.

É interessante destacar as atribuições citadas anteriormente acerca do vetor enable e seus respectivos domínios.

Abaixo segue o código do sistema e sua visualização em RTL.

```
module sistema (clk, clk_2b, clk_3b, rst, sel_mux, dados, res, hex0, hex1, hex2, hex3, hex4, hex5, enable, led_0, led_1, led_2, led_3, led_4);

input clk, rst, clk_2b, clk_3b;
input [7:0] dados;
input sel_mux;
output [5:0] enable;
output [6:0] hex0, hex1, hex2, hex3, hex4, hex5;
output led_0, led_1, led_2, led_3, led_4;
output [3:0] res;

wire [3:0] reg_a, reg_b, out_ULA, q_desl, out_reg_c;
wire [2:0] count_3b;
wire [1:0] count_2b;

// module controle (clk, rst, state, out);
controle control1 (clk, rst, estado, enable);

// module registrador (rst, clk, en, d, q);
registrador Reg_A (rst, clk, enable[0], dados[3:0], reg_a);

registrador Reg_B (rst, clk, enable[1], dados[7:4], reg_b);

registrador Reg_C (rst, clk, enable[2], out_ULA, out_reg_c);

// module contador_2b (clk, rst, en, out, led0, led1);
contador_2b COUNT_2B (clk_2b, rst, enable[4], count_2b, led_0, led_1);

// module contador_3b (clk, rst, en, out, led0, led1, led_2);
contador_3b COUNT_3B (clk_3b, rst, enable[5], count_3b, led_2, led_3, led_4);

// module ULA (op_a, op_b, sel_ULA, out)
ULA ula (reg_a, reg_b, count_3b, out_ULA);

 // module reg_desloc (rst, clk, en, d, out, sel);
reg_desloc reg_desloc (rst, clk, enable[3], out_ULA, q_desl, count_2b);

// module  mux (A, B, sel_mux, out)
mux MUX (q_desl, out_reg_c, sel_mux, res);

// module decod_display (codigo_BCD, display)
display zero (reg_a, hex0);

display um (reg_b, hex1);

display dois (out_ULA, hex2);

display tres (q_desl, hex3);

display quatro (out_reg_c, hex4);

display cinco (res, hex5);

endmodule
```

Figura 23 - Código do sistema



O resultado obtido é bastante satisfatório, pois representa de forma fiel o circuito proposto.

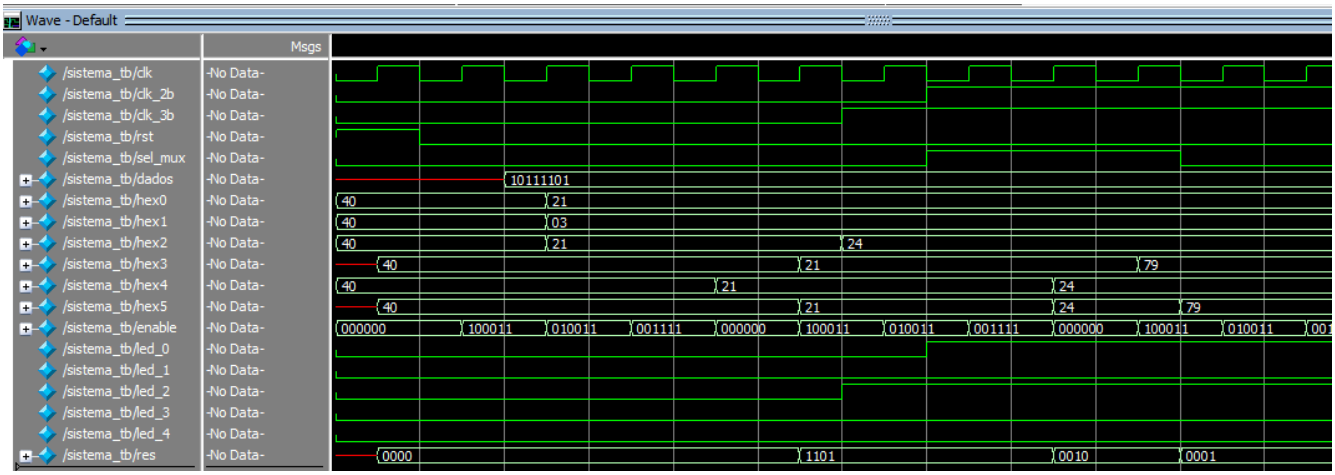


Figura 24 - Resultado do Testebench do sistema 1

O resultado do testebench obteve o resultado esperado. Podemos observar a entrada de dados no registrador A = 1101 e no registrador B = 1011. Observamos no HEX0 o valor de 21 em hexadecimal correspondente a “D” ou 1101, ou seja, o valor de saída do registrador A. O mesmo ocorre com o HEX1 que mostra o valor 03 correspondente a “B” ou 1011, saída do registrador B.

Com a seleção da ULA em 000, ela deve apenas carregar o valor de A, o que é visto no HEX2, o qual demonstra o valor hexadecimal de 21.

Como a seleção do registrador de deslocamento está em 00, ele também deve apenas ler o valor da saída da ULA, o que é visto em HEX3. O mesmo ocorre com o registrador C, visto em HEX4. Como a seleção do MUX está em 0, observamos o valor do registrador de deslocamento.

No tempo de 120 ns, podemos observar uma mudança na saída da ULA. Com a mudança da ULA para a seleção 001, esperamos como resultado a inversão dos bits de entrada do registrador A, ou seja, o inverso de 1101, que seria 0010. O que é corretamente observado pelo número 24 em hexadecimal correspondente à 0010.

Nesse ciclo, a seleção do registrador de deslocamento está alocada em 01, correspondente ao deslocamento à direita com preenchimento de zeros, isto é, o número espero é 0001, o que é observado pela saída 79 em hexadecimal.

O registrador C armazenou a saída da ULA, correspondente a 24. E como a seleção do MUX está em 1, podemos observar a saída do registrador C em HEX5. E no tempo 200 ns, com a mudança da seleção do MUX para 0, observamos a saída do registrador de deslocamento em HEX5.

2.6 PINAGEM

Para o processo de mapeamento de pinos seguiu – se as recomendações feitas na tabela a seguir.

Sinal	Componente da placa	Pino da FPGA	Descrição	*I/O standard*
Clock do sistema	MAX10_CLK1_50	PIN_P11	50 MHz clock input(Bank 3B	3.3-V LVTTL
Clock do contador de 3 bits	KEY 0	PIN_B8	Push-button[0]	3.3 V SCHMITT TRIGGER
Clock do contador de 2 bits	KEY 1	PIN_A7	Push-button[1]	3.3 V SCHMITT TRIGGER
Reset	SW0	PIN_C10	Slide Switch[0]	3.3-V LVTTL
Dados – bit[0] lsb	SW6	PIN_A13	Slide Switch[6]	3.3-V LVTTL
Dados – bit[1]	SW7	PIN_A14	Slide Switch[7]	3.3-V LVTTL
Dados – bit[2]	SW8	PIN_B14	Slide Switch[8]	3.3-V LVTTL
Dados – bit[3] msb	SW9	PIN_F15	Slide Switch[9]	3.3-V LVTTL
Habilita Reg. A	SW1	PIN_C11	Slide Switch[1]	3.3-V LVTTL
Habilita Reg. B	SW2	PIN_D12	Slide Switch[2]	3.3-V LVTTL
Habilita Reg. C	SW3	PIN_C12	Slide Switch[3]	3.3-V LVTTL
Habilita Reg. Des.	SW4	PIN_A12	Slide Switch[4]	3.3-V LVTTL
Os displays de 7-segmentos HEX[5:0] devem ser conectados conforme indicado na figura 1 através de decodificadores BCD para 7-segmentos.				
O valor dos contadores de 2 e 3-bits devem ser mostrados nos leds LDR, conforme indicado na figura 1.				

Figura 25 - Tabela de mapeamento de pinos

Exceto as chaves slide switch para a habilitação dos registradores, pois foi visto apenas no final do projeto que os mesmo eram habilitados por chaves. Dessa maneira, neste respectivo projeto, os registradores são habilitados por meio de uma máquina de estados já descrita anteriormente.

As chaves que antes correspondiam à habilitação dos registradores, agora foram substituídas por entrada de dados do registrador A. Utilizou – se também a chave Slide Switch [5] para a seleção do multiplexador.

Para a implementação dos pinos correspondentes ao display de 7 segmentos e aos leds, baseou – se no arquivo DE10-Lite_User_Manual, onde seguiu – se as seguintes tabelas:

Signal Name	FPGA Pin No.	Description	I/O Standard
LEDR0	PIN_A8	LED [0]	3.3-V LVTTL
LEDR1	PIN_A9	LED [1]	3.3-V LVTTL
LEDR2	PIN_A10	LED [2]	3.3-V LVTTL
LEDR3	PIN_B10	LED [3]	3.3-V LVTTL
LEDR4	PIN_D13	LED [4]	3.3-V LVTTL
LEDR5	PIN_C13	LED [5]	3.3-V LVTTL
LEDR6	PIN_E14	LED [6]	3.3-V LVTTL
LEDR7	PIN_D14	LED [7]	3.3-V LVTTL
LEDR8	PIN_A11	LED [8]	3.3-V LVTTL
LEDR9	PIN_B11	LED [9]	3.3-V LVTTL

Figura 26 – Mapeamento dos leds

Signal Name	FPGA Pin No.	Description	I/O Standard
HEX00	PIN_C14	Seven Segment Digit 0[0]	3.3-V LVTTL
HEX01	PIN_E15	Seven Segment Digit 0[1]	3.3-V LVTTL
HEX02	PIN_C15	Seven Segment Digit 0[2]	3.3-V LVTTL
HEX03	PIN_C16	Seven Segment Digit 0[3]	3.3-V LVTTL
HEX04	PIN_E16	Seven Segment Digit 0[4]	3.3-V LVTTL
HEX05	PIN_D17	Seven Segment Digit 0[5]	3.3-V LVTTL
HEX06	PIN_C17	Seven Segment Digit 0[6]	3.3-V LVTTL
HEX07	PIN_D15	Seven Segment Digit 0[7], DP	3.3-V LVTTL
HEX10	PIN_C18	Seven Segment Digit 1[0]	3.3-V LVTTL
HEX11	PIN_D18	Seven Segment Digit 1[1]	3.3-V LVTTL
HEX12	PIN_E18	Seven Segment Digit 1[2]	3.3-V LVTTL
HEX13	PIN_B16	Seven Segment Digit 1[3]	3.3-V LVTTL

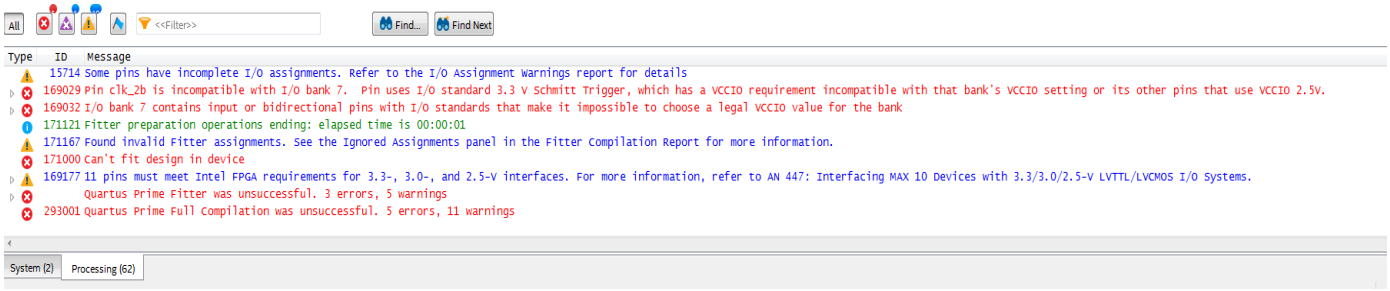
Figura 27 – Mapeamento do display de 7 segmentos 1

HEX14	PIN_A17	Seven Segment Digit 1[4]	3.3-V LVTTL
HEX15	PIN_A18	Seven Segment Digit 1[5]	3.3-V LVTTL
HEX16	PIN_B17	Seven Segment Digit 1[6]	3.3-V LVTTL
HEX17	PIN_A16	Seven Segment Digit 1[7] , DP	3.3-V LVTTL
HEX20	PIN_B20	Seven Segment Digit 2[0]	3.3-V LVTTL
HEX21	PIN_A20	Seven Segment Digit 2[1]	3.3-V LVTTL
HEX22	PIN_B19	Seven Segment Digit 2[2]	3.3-V LVTTL
HEX23	PIN_A21	Seven Segment Digit 2[3]	3.3-V LVTTL
HEX24	PIN_B21	Seven Segment Digit 2[4]	3.3-V LVTTL
HEX25	PIN_C22	Seven Segment Digit 2[5]	3.3-V LVTTL
HEX26	PIN_B22	Seven Segment Digit 2[6]	3.3-V LVTTL
HEX27	PIN_A19	Seven Segment Digit 2[7] , DP	3.3-V LVTTL
HEX30	PIN_F21	Seven Segment Digit 3[0]	3.3-V LVTTL
HEX31	PIN_E22	Seven Segment Digit 3[1]	3.3-V LVTTL
HEX32	PIN_E21	Seven Segment Digit 3[2]	3.3-V LVTTL
HEX33	PIN_C19	Seven Segment Digit 3[3]	3.3-V LVTTL
HEX34	PIN_C20	Seven Segment Digit 3[4]	3.3-V LVTTL
HEX35	PIN_D19	Seven Segment Digit 3[5]	3.3-V LVTTL
HEX36	PIN_E17	Seven Segment Digit 3[6]	3.3-V LVTTL
HEX37	PIN_D22	Seven Segment Digit 3[7] , DP	3.3-V LVTTL
HEX40	PIN_F18	Seven Segment Digit 4[0]	3.3-V LVTTL
HEX41	PIN_E20	Seven Segment Digit 4[1]	3.3-V LVTTL
HEX42	PIN_E19	Seven Segment Digit 4[2]	3.3-V LVTTL
HEX43	PIN_J18	Seven Segment Digit 4[3]	3.3-V LVTTL
HEX44	PIN_H19	Seven Segment Digit 4[4]	3.3-V LVTTL
HEX45	PIN_F19	Seven Segment Digit 4[5]	3.3-V LVTTL
HEX46	PIN_F20	Seven Segment Digit 4[6]	3.3-V LVTTL
HEX47	PIN_F17	Seven Segment Digit 4[7] , DP	3.3-V LVTTL
HEX50	PIN_J20	Seven Segment Digit 5[0]	3.3-V LVTTL
HEX51	PIN_K20	Seven Segment Digit 5[1]	3.3-V LVTTL
HEX52	PIN_L18	Seven Segment Digit 5[2]	3.3-V LVTTL
HEX53	PIN_N18	Seven Segment Digit 5[3]	3.3-V LVTTL
HEX54	PIN_M20	Seven Segment Digit 5[4]	3.3-V LVTTL
HEX55	PIN_N19	Seven Segment Digit 5[5]	3.3-V LVTTL
HEX56	PIN_N20	Seven Segment Digit 5[6]	3.3-V LVTTL
HEX57	PIN_L19	Seven Segment Digit 5[7] , DP	3.3-V LVTTL

Figura 28 - Mapeamento do display de 7 segmentos 2

O vetor enable [5:0] foi conectado nos leds 3, 4, 5, 6 e 7, onde a posição enable[0] ficou sem conexão.

É necessário destacar que ao denominar o clock do contador de 2 bits como 3.3 V Schmitt Trigger, foi obtido o seguinte erro de compilação:



Dessa forma, o clock do contador de 2 bits foi deixado em 2.5 V, para a compilação ser efetuada com sucesso.

3 CONSIDERAÇÕES FINAIS

O projeto de um circuito para interface com ULA exige do projetista uma habilidade em unir a experiência prática com os conceitos teóricos. No projeto realizado notou-se que cada módulo desempenhava uma função, contribuindo para a construção do sistema final. Observou-se também que projetar um circuito em Verilog exige além do conhecimento da sintaxe, conhecimento profundo daquilo que se deseja projetar.

O objetivo de verificar o funcionamento de cada dispositivo foi atingido visto que fez-se necessário para o projetista ter uma excelente bagagem teórica referentes a cada dispositivo.

Notou-se também que existiram falhas no projeto apresentado, como: a implementação dos dados de entradas em entradas separadas, quando no circuito proposto é apresentado apenas uma; elaboração de um testebench capaz de testar todas as possibilidades de operação de uma só vez; a habilitação dos registradores por meio de chaves.

Entretanto, o projeto pode ser considerado como uma ótima fonte de aprendizado e aplicação do conhecimento.