UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Lukas Leitão
Fernando Girão

Experiência I

Trabalho apresentado à disciplina de Laboratório de Sistemas Digitais turma EL3 do curso de Engenharia Eletrônica e de Computação da UFRJ

Professor J.B.O Souza Filho

Rio de Janeiro
Outubro de 2018

Sumário

1 Introdução

Desenvolvemos uma ULA capaz de gerar 8 operações para duas entradas de 4 bits quaisquer. O usuário insere os dois números e, após selecionar a operação que deseja, o resultado é exibido através de leds localizados na FPGA. As operações que fazem parte da ULA são: adição(000), subtração(001), multiplicação(010), incremento de 1(011), xor(100), or(101), and(110) e inversão(111). Os bits entre parênteses são os bits de seleção para a escolha da operação.

2 Implementação

2.1 ULA

A ULA foi projetada levando-se em conta os componentes construídos abaixo e o *Multiplexador*, que é um conjunto 3 bits que variam de 000 à 111 para representar qual operação deve ser escolhida. Sendo assim, temos que o projeto da ULA se resume a:

1. Declaração das entradas e saídas
2. Declaração dos componentes utilizados
3. Declaração dos sinais utilizados
4. Port map dos componentes que irão executar a operação
5. Seleção da operação através do MUX

   De 1, temos 2 entradas que são os dois vetores de 4 bits e uma saída que executa a operação escolhida pelo usuário. No *Port* incluímos um MUX para selecionar as operação de (000 à 111) que foram especificadas na seção de *Resultados*.
   Após isso, fizemos a importação de todos os componentes projetados(2) e que serão explicados nas seções abaixo.
   Em 3, criamos diversos sinais de saída referentes à saída de cada componentes que foi importado no código da ULA. Sendo assim, poderemos manipulá-los de tal forma que, em 4 iremos escolher essas saídas de acordo com a escolha do usuário.

2.2 Componentes da ULA

2.2.1 Somador

O projeto do somador de quatro bits consiste no uso de quatro somadores simples de apenas dois bits. Utiliza-se o comando "port map"para cada somador, que é a correspondência entrada/saída do circuito com módulos somadores de dois bits existentes. Assim, de acordo com "*Apêndice*− 5.4" temos a implementação de cada somador do somador completo.

O somador possui as entradas A, B que são os dois vetores de 4 bits, Cin que é o $Carry - in$ e três saídas S, $Cout$ e OV que representam, respectivamente, a saída da soma entre A, B e Cin, o $Carry - out$ e o $Overflow$. Para fins de ilustração, o somador funciona da seguinte forma:

$$A + B + Cin = S + Cout \tag{1}$$

$$OV = C3 \oplus C2 \tag{2}$$

onde C3 = Cout e C2 estão ilustrados abaixo:

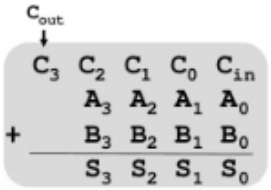

Figura 1: Somador

Para que pudéssemos utilizar os somadores de 2 bits, utilizamos o código *component* que importa o bloco somador de 2 bits.(Apêndice 5.3 - Figura 15)

2.2.2 Subtrator

A composição de subtrator é muito parecida com a do somador. Aqui, utilizamos o conceito de soma em *complemento de* 2 , na qual a segunda entrada é invertida e somada à "1". Assim, a subtração é dada da seguinte forma:

$$SUB = A + \bar{B} + 1 \tag{3}$$

A descrição do subtrator começa a partir de seu $Port$ (Apêndice 5.2 - Figura 16), onde A, B representam as entradas de quatro bits cada, S representa a saída, e $Borrow$ é o valor invertido de Cout no portmap do somador de quatro bits. E, por fim, o OV que representa o $Overflow$.

Já na arquitetura do código (Apêndice 5.2 - Figura 16), criamos uma variável do tipo *Signal* para receber o valor invertido de B. Além disso, há o $Cout$ que servirá de $Borrow$.
Também importamos os componente *inverter* para que pudéssemos gerar $\bar{B}$. Por fim, utilizou-se o componente $fourBitsAdder$.

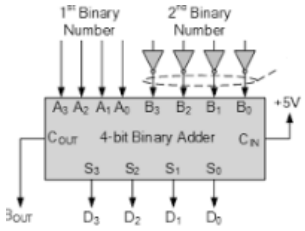

Figura 2: Subtrator

A partir do *begin*, o funcionamento se dá da seguinte forma:
1- Port map do inversor, para que B se torne $\bar{B}$
2- Port map do somador, para que A soma de A com $\bar{B}$ e $'1'$ seja feita
3- Inversão do $Cout$ para que ele se torne o $Borrow$

2.2.3 Multiplicador

O projeto do multiplicador tem duas entradas de 4 bits e uma saída de 4 bits também. O resultado verdadeiro possui como saída um resultado de 8 bits, porém, para efeito do projeto na FPGA, só iremos retornar os 4 bits menos significativos. Sua implementação necessita de 3 componentes de somadores de 8 bits.

Os somadores de 8 bits tem os pinos iguais a de um somador de 4 bits, porém, as entradas dos números "A"e "B"e o resultado da operação "S"possuem 8 bits. A implementação do somador de 8 bits necessita de dois somadores de 4 bits, utilizando as entradas do somador de 8 bits e as saídas do de 4 bits como mostra a figura abaixo:

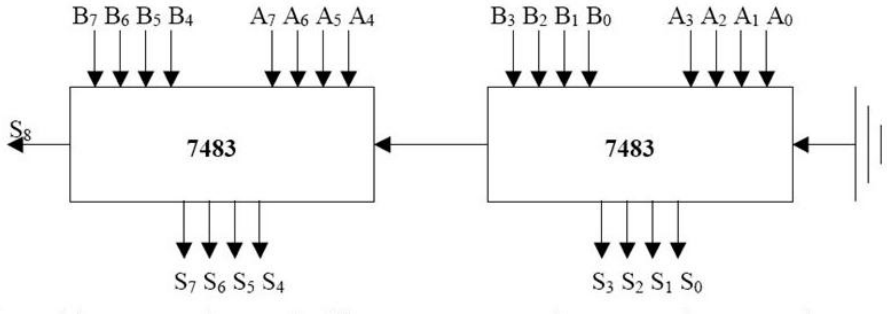

Figura 3: Somador 8 bits

Conhecendo os sinais usados para a implementação do multiplicador no código (Apêndice 5.8 - Figura 22) e os somadores de 8 bits, agora é possível continuar a explicação da implementação.

Os sinais soma0, soma1, soma2 e soma3 possuindo 4 bits representam respectivamente os resultados dos ANDs realizados entre os bits B0, B1, B2, B3 com todos os 4 bits do número A. Como é ilustrado na figura abaixo:

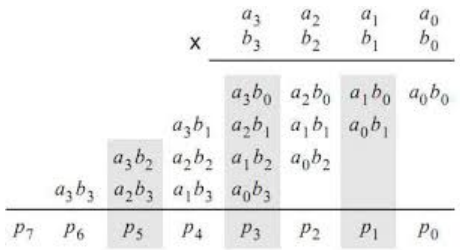

Figura 4: Multiplicando

Existem 4 sinais chamados "parte_soma_final"possuindo 8 bits que representam da terceira linha a sexta linha da figura acima. A "parte1_soma_final" representa os ANDs dos B0 com o A inserindo 0's onde não há esses ANDS (3ª linha Fig 4), a "parte2_soma_final"representa os ANDs dos B1 com o A também inserindo 0's onde não há esses ANDS (4ª linha Fig 4) e assim por diante.

Os 3 somadores do 8 bits foram chamados de "somaBit01", "somaBit23"e "resultado_0123". Os números postos nos nomes dos 2 primeiros somadores acima se dá ao fato de que suas 2 entradas receberam "parteX_soma_final", onde esse X será substituído por um dos dois números em seu nome em cada entrada e seus *carry in* estarão ligados ao terra. Suas saídas OV e Cout não serão conectadas a nada. A saída em cada um desses dois somadores terá o nome de "resultado_parcial01"e "resul-

tado_parcial02"respectivamente. E assim, estas duas saídas serão as entradas do somador "resultado_0123"e esta saída será o resultado da multiplicação. Porém, como só queremos os 4 bits menos significativos do resultado da multiplicação, o sinal de saída deste último somador será chamado de "saida_real"pois possui 8 bits e a saída "S"do multiplicador será os 4 bits menos significativos deste sinal.

2.2.4 Incrementador

O incrementador funciona de forma bem simples: dada uma entrada qualquer de quatro bits, ele adiciona o valor 0001 à essa entrada. Para que o código ficasse sucinto, decidimos utilizar novamente o somador.
A análise pode ser feita começando pelo $Port$, no qual há a entrada A e a saída S que é dada por:

$$S = A + 1 \tag{4}$$

(Apêndice 5.7 - Figura 21)
Em seguida, adicionamos um somador 4 bits como component e inserimos as entradas A e 0000, pois o valor 0001 que incrementa será adicionado através do Cin. É por esse motivo que, no terceiro campo do $Port - map$ do incrementador há o valor $'1'$.

2.2.5 XOR

Dada duas entradas A e B, este bloco irá executar a seguinte operação:

$$S = A \oplus B \tag{5}$$

Sendo assim, há a necessidade de apenas duas entradas e uma saída no na $Entidade\ Porta - Xor$.(Apêndice 5.13 - Figura 27)
A partir do *begin* o código é executado através de um loop for para que seja feita a operação bit a bit. Teremos ao final do processo, então, o valor da saída S. Esta saída será igual a 1 apenas quando os 2 bits em questão forem diferentes.

2.2.6 OR

Dada duas entradas A e B, este bloco irá executar a seguinte operação:

$$S_n = A_n + B_n \tag{6}$$

, com n variando entre 0 e 3.
Ou seja: a operação OR é feita bit a bit.
Caso apenas um dos bits seja igual a 1, o resultado da operação será 1.

2.2.7 AND

Dada duas entradas A e B, este bloco irá executar a seguinte operação:

$$S_I = A_I(and)B_I \tag{7}$$

, com I variando entre 0 e 3.
A operação fará com que caso apenas 1 dos bits seja 0, o resultado será 0.
(Apêndice 5.5 - Figura 19)

2.2.8 NOT

Dada duas entradas A e B, este bloco irá executar a seguinte operação:

$$S = \bar{A} \tag{8}$$

Ou seja, caso o bit seja 0, muda para 1 e caso o bit seja 1, muda para 0.
(Apêndice 5.9 - Figura 23)

2.3 Contador

O contador foi implementado com as entradas load, dados, clock, clear e com 1 saída retornando a contagem. Como a FPGA funciona numa frequência de $50Mhz$, o contador precisa contar até 5.10^7 para representar 1 segundo de contagem. Porém, esse contador conta até 6 segundos para que a cada 2 segundos a FPGA mostre o número A, depois o B, e em seguida o resultado da operação da ULA totalizando 6 segundos. Então, o contador terá módulo 30.10^7.

O processo de contagem é sensível ao clock e ao reset. Dentro do processo há uma variável chamada "contando"que vai de 30.10^7 a 0, e esta variável recebe 0 quando o $reset$ = '1' ou recebe o que está na entrada do $data$ quando $load$ = '1' junto com um pulso de clock. Quando $reset$ e o $load$ foram iguais a '0', a saída do contador será acrescentada em 1 em cada pulso de clock até que chegue ao máximo da variável "contando". Após chegar no seu máximo que é 30.10^7, a variável receberá 0 e continuará acrescentando de 1 em 1 até o máximo novamente.
(Apêndice 5.6 - Figura 20)

2.4 Gerador de sinais

Abaixo está representada a máquina de estados do gerador de sinais. Pensamos em duas etapas do processo como um todo: uma para fazer todos os cálculos através da ula e outra para exibir os valores inseridos e o resultado referentes à eles.

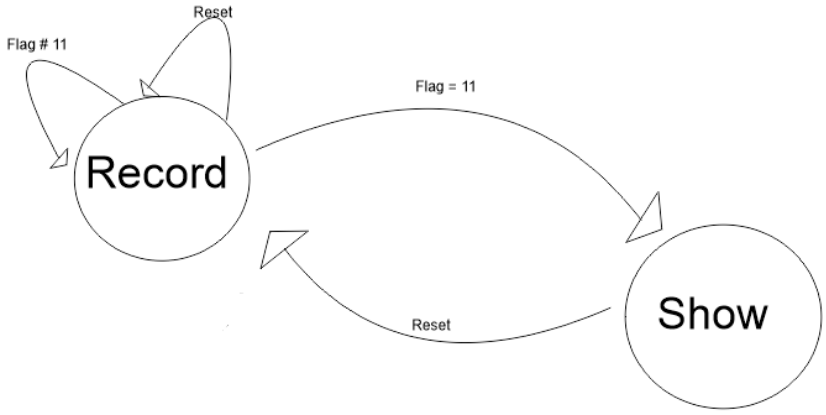

Figura 5: Máquina de estados

Primeiramente, é preciso descrever sobre as variáveis, sinais e componentes que utilizamos para compor o gerador de sinais e assim, futuramente, mostraremos como funciona a máquina de estados do mesmo.
O componente do gerador possui o *clk* e o *rst*, que são, respectivamente, o clock da FPGA e o reset. Temos, em seguida, as entradas para os dois números binários de 4 bits que serão inseridos através da $EntradaUla$, que representa fisicamente os switches. A partir daí, os atributos restantes são os leds $ledA$ e $ledB$, que indicam visualmente os valores inseridores pelo usuário e $ledsToShowInAndOut$

que tem como utilidade mostrar os bits A e B inseridos e seu respectivo resultado de acordo com a operação escolhida.(Apêndice 5.1 - Figura 15)
No tocando aos componentes inseridos para que o gerador pudesse funcionar, precisamos inserir o componente *contador*, pois o período do clock é muito pequeno para que o usuário tenha tempo de ver as entradas e saídas do sistema. Logo, o contador funciona como o mencionado em 2.2 justamente para que o usuário possa visualizar o que foi explicado acima.
Em seguida, inserimos o compoente ULA, para que as operações fossem executadas e mostradas de acordo com a seleção do usuário. (Apêndice 5.1 - Figura 15)
Além dos componentes acima, precisamos criar valores do tipo "*signal*" para que o gerador pudesse funcionar utilizando os componentes inseridos. Temos os seguintes sinais criados:

(a) flag

(b) entradaXTempUla

(c) entradaYTempUla

(d) Cout

(e) OV

(f) saidaContador

(g) saidaUla

O sinal $flag$ é utilizado para sinalizar ao estado atual que é preciso mudar para o próximo estado.
As entradas em 2 e 3 servem para receber os valores da $EntradaUla$, uma vez que esta última deve receber duas entradas. Logo, devemos salvá-las.
4 e 5 servem para armazenar os valores do $Carryout$ e do $Overflow$, respectivamente.
6 será utilizado no bloco que controla a quantidade de segundos em que as entradas e a saída aparecerão para o usuário.
7 armazena o resultado da operação de acordo com a escolhida pelo usuário(Apêndice 5.1 - Figura 15)
Ao iniciar o $begin$ do código, decidimos fazer "$port-maps$" para os componentes da Ula e do contador para que eles pudessem fazer os cálculos de acordo com os sinais estabelecidos acima e devolvê-los para o gerador.
Após isso, criamos um processo referente à máquina de estado para que ela dependa do clock e do reset. (Apêndice 5.1 - Figura 15)

Em seguida, dentro da máquina de estados, zeramos os sinais que foram mencionais acima caso o $reset$ seja verdadeiro, ou seja, o usuário quis voltar para o estado inicial. (Apêndice 5.1 - Figura 15)
Caso isso não aconteça, verifica-se os possíveis tipos de estados:

(a) state_record

(b) state_show

Quando o estado atual for $state_record$, ou seja, o estado de gravação dos valores calculados, verificaremos:

(a) Se o botão referente à entrada X (primeira entrada) foi apertado

(b) Se o botão referente à entrada Y (segunda entrada) foi apertado

(c) Se o flag que representa a mudança de estado é verdadeiro

Referente à 1, caso o botão referente à X seja pressionado, $flag(0)$ receberá 1, que significa a inserção do primeiro número
Em 2, caso o botão referente à Y seja pressionado, $flag(1)$ receberá, que significa a inserção do segundo número.
Caso as condições acima ocorram, o vetor $flag$ valerá 11, que indicará a mudança para o estado de exibição através de:

$$estado <= state\_show \tag{9}$$

Caso a nenhuma condição acima ocorra, o estado permanecerá o mesmo através de:

$$estado <= state\_record \tag{10}$$

Satisfeita a condição $flag == 11$, temos que o próximo estado será o estado de exibição. Neste estado, exibimos a entrada X por dois segundos $10 * 10^7$, a entrada Y por dois segundos e, por fim, o resultado até que o contador chegue em seu valor limite especificado em 2.2
Para cada condição representada em (Apêndice 5.1 - Figura 15), $ledA$ e $ledB$ serão acesos para demonstrar que durante o tempo atual, a entrada X ou Y está sendo exibida. Já na última condição desse estado, mostramos o resultado de acordo com a operação escolhida. O estado de exibição continua em operação até que seja inserido o $reset$ e volt para o estado de gravação.
É importante observarmos que o usuário pode mudar as operações da ula mesmo no estado de exibição, uma vez que continuamente a ULA recebe os valores $entradaUla(2downto0)$ (que variam de 000 à 111) através da da seleção os switches.(Apêndice 5.1 - Figura 15)

3 Resultados

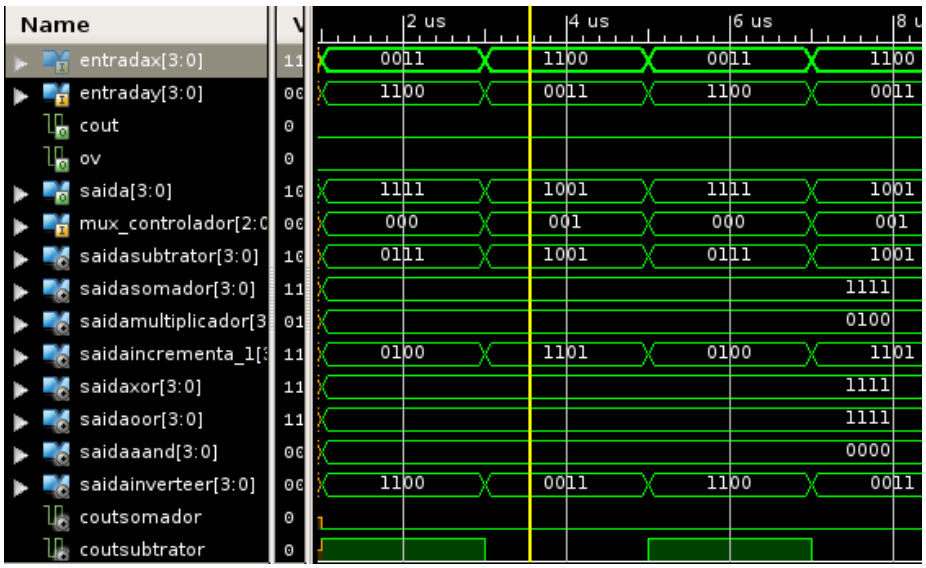

Figura 6: Simulação geral da ULA

Operações - Selecionadores

- Adição - 000

- Subtração - 001
- Multiplicação - 010
- Incremento de 1 - 011
- XOR - 100
- OR - 101
- AND - 110
- Inversão - 111

3.1 Adição

Selecionador: 000

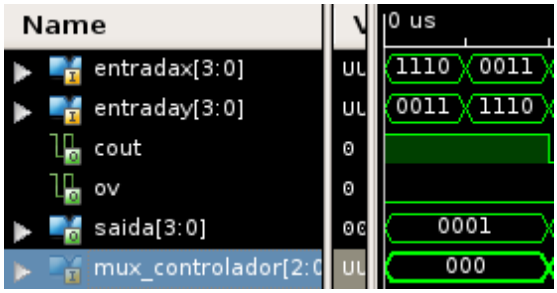

Figura 7: Adição

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, após certo intervalo, elas são invertidas. Observamos que a alternância das entradas não altera a saída e que há um *carry out* = 1 em todo o intervalo. O OV não é utlizado

Saída = 1110 + 0011 = 0001 com Cout = 1

Resultado da operação foi como esperado.

3.2 Subtração

Selecionador: 001
Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, após certo intervalo, elas são invertidas. Observamos que a alternância das entradas altera a saída e que há um *carry out* = 0 na metade do intervalo e *carry out* = 1 na outra metade. Esse *carry out* ,no caso, simboliza o *borrow* do subtrator. o OV não é utilizado.

Saída = 1110 - 0011 = 1011 com Borrow = 0

Saída = 0011 - 1110 = 0101 com Borrow = 1

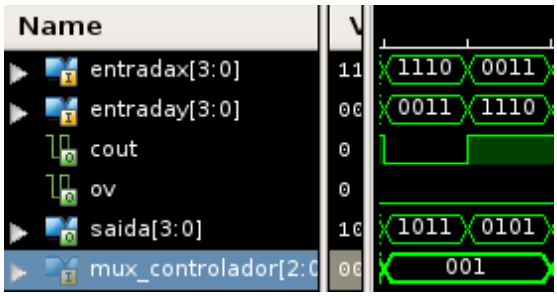

Figura 8: Subtração

Resultado da operação foi como esperado.

3.3 Multiplicação

Selecionador: 010

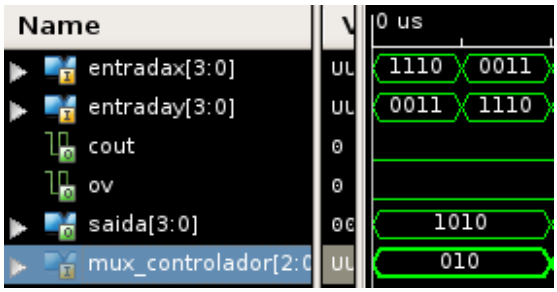

Figura 9: Multiplicação

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, após certo intervalo, elas são invertidas. Observamos que a alternância das entradas não altera a saída e que há um *carry out* = 0 pois ele não é utlizado nessa operação, assim como o OV

Saída = 1110 x 0011 = 00101010

Como só queremos os 4 bits menos significativos, o resultado da operação foi como esperado.

3.4 Incremento de 1

Selecionador: 011
Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, e após certo intervalo, elas são invertidas. Observamos

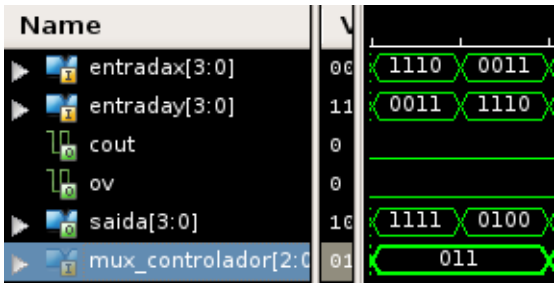

Figura 10: Incrementa de 1 a entradax

que a alternância das entradas altera a saída pois a operação só utiliza a entradax e a incrementa 1, e que há um *carry out* = 0 em todo o intervalo pois ele não é utilizado assim como o OV não é utlizado.

Saída = entradax +1
Entrada = 1110 , Saída = 1111
Entrada = 0011 , Saída = 0100

Resultado da operação foi como esperado.

3.5 XOR

Selecionador: 100

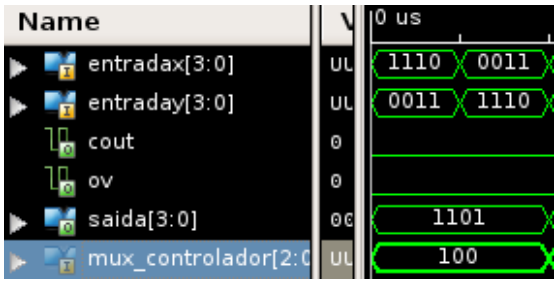

Figura 11: XOR

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, e após certo intervalo, elas são invertidas. Observamos que a alternância das entradas não altera a saída, e que há um *carry out* = 0 em todo o intervalo pois ele não é utilizado assim como o OV não é utlizado.

Saída = entradax XOR entraday
Saída = 1101

Resultado da operação foi como esperado.

3.6 OR

Selecionador: 101

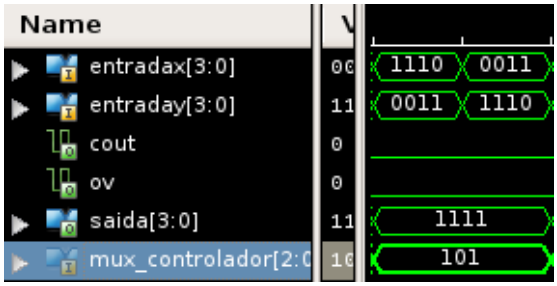

Figura 12: OR

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, e após certo intervalo, elas são invertidas. Observamos que a alternância das entradas não altera a saída, e que há um *carry out* = 0 em todo o intervalo pois ele não é utilizado assim como o OV não é utlizado.

Saída = entradax OR entraday
Saída = 1111

Resultado da operação foi como esperado.

3.7 AND

Selecionador: 110

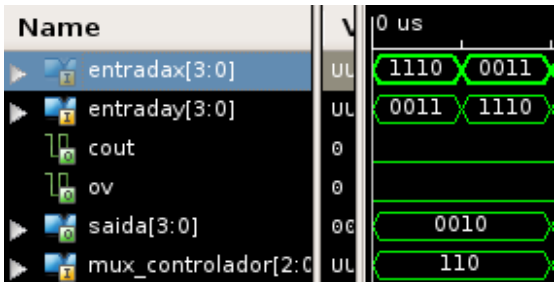

Figura 13: AND

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, e após certo intervalo, elas são invertidas. Observamos que a alternância das entradas não altera a saída, e que há um *carry out* = 0 em todo o intervalo pois ele não é utilizado assim como o OV não é utlizado.

Saída = entradax AND entraday
Saída = 0010

Resultado da operação foi como esperado.

3.8 Inversão

Selecionador: 111

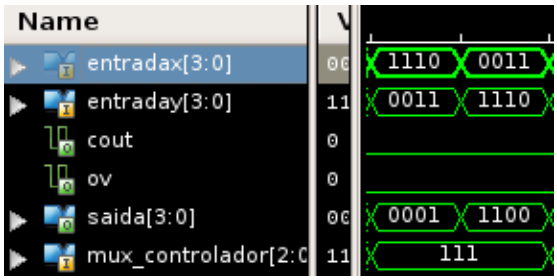

Figura 14: Inverte a entradax

Nesta imagem temos primeiramente a primeira entrada com 1110 e a segunda com 0011, e após certo intervalo, elas são invertidas. Observamos que a alternância das entradas altera a saída pois a operação só utiliza a entradax e a inverte, e que há um *carry out* = 0 em todo o intervalo pois ele não é utilizado assim como o OV não é utlizado.

Saída = entradax invertida
Entrada = 1110 , Saída = 0001
Entrada = 0011 , Saída = 1100

Resultado da operação foi como esperado.

4 Conclusão

O projeto está funcionando perfeitamente, porém, há muitas saídas e entradas inspiradas nos componentes reais que não foram utilizadas, podendo, portanto, não terem feito parte do projeto.

5 Apêndice

5.1 Gerador De Sinais

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity GeradorDeSinais is
7      port (clk, rst: in bit; -- rst serÃ¡ um botÃ£o
8            botaoEntradaDoNumeroX, botaoEntradaDoNumeroY: in std_logic;
9            entradaUla: in std_logic_vector (3 downto 0);
10           ledA, ledB, led_result : out std_logic;
11           ledsToShowInAndOut : out std_logic_vector (3 downto 0));
12
13  end GeradorDeSinais;
14
15  architecture hardware of GeradorDeSinais is
16
17     component contador
18     port ( load : in bit;
19            clk  : in bit;
20            rst  : in bit;
21            data : in integer range 300000000 downto 0;
22            saida: out integer range 300000000 downto 0);
23
24     end component;
25
26     component ULA
27      Port ( entradaX : in  STD_LOGIC_VECTOR (3 downto 0);
28             entradaY : in  STD_LOGIC_VECTOR (3 downto 0);
29             Cout : out  STD_LOGIC;
30             OV : out  STD_LOGIC;
31             SAIDA : out  STD_LOGIC_VECTOR (3 downto 0);
32             MUX_CONTROLADOR : in  STD_LOGIC_VECTOR (2 downto 0));
```

```
33      end component;
34
35      type state is (state_record,state_show);
36      signal estado: state;
37
38      signal flag: std_logic_vector (1 downto 0);
39      signal entradaXTempUla: STD_LOGIC_VECTOR (3 downto 0);
40      signal entradaYTempUla: STD_LOGIC_VECTOR (3 downto 0);
41      signal Cout: STD_LOGIC;
42      signal OV: STD_LOGIC;
43
44      signal saidaContador: integer range 300000000 downto 0;
45      signal saidaULA: std_logic_vector (3 downto 0);
46
47
48
49  begin
50      ProcessamentoUla: ULA port map (EntradaXTempUla, EntradaYTempUla, Cout, OV,
    saidaULA, entradaUla(2 downto 0));
51      contador1: contador port map ( '0', clk, rst, 0, saidaContador);
52
53      maquinaDeEstado: process(clk, rst)
54      begin
55          if rst = '1' then
56              EntradaXTempUla <= "0000";
```

```
57              EntradaYTempUla <= "0000";
58              ledsToShowInAndOut <= "0000";
59              led_result <= '0';
60              ledA <= '0';
61              ledB <='0';
62              estado <= state_record;
63              flag <= "00";
64
65          elsif (clk'event and clk = '1') then
66              case estado is
67                  when state_record =>
68
69                      if botaoEntradaDoNumeroX = '1'   then
70                          EntradaXTempUla <= entradaUla;
71
72                          flag(0) <= '1';
73
74                      elsif botaoEntradaDoNumeroY = '1' then
75                          EntradaYTempUla <= entradaUla;
76                          ledB <= '1';
77                          flag(1) <= '1';
78
79
80                      elsif (flag = "11") then
81                          ledA <='0';
82                          ledB <='0';
83                          estado <= state_show;
```

```
84                  else
85                     estado <= state_record;
86                  end if;
87
88                        )
89
90              when state_show =>
91
92                   if saidaContador  < 100000000 then
93                        ledsToShowInAndOut <= EntradaXTempUla;
94                        ledA <= '1';
95                        ledB <= '0';
96                        led_result <= '0';
97                   elsif ((saidaContador >= 100000000) and (saidaContador < 200000000))
     then
98                        ledsToShowInAndOut <= EntradaYTempUla;
99                        ledA <= '0';
100                       ledB <= '1';
101                       led_result <= '0';
102                  elsif saidaContador >= 200000000  then
103                       ledsToShowInAndOut <= saidaULA;
104                       ledA <= '0';
105                       ledB <= '0';
106                       led_result <= '1';
107                   estado <= state_show;
```

Figura 15: Gerador de sinais

5.2 Subtrator

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity fourBitsSubtractor is
7       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
8              B : in  STD_LOGIC_VECTOR (3 downto 0);
9              S : out  STD_LOGIC_VECTOR (3 downto 0);
10             Borrow : out  STD_LOGIC;
11             OV : out std_logic);
12  end fourBitsSubtractor;
13
14  architecture Behavioral of fourBitsSubtractor is
15     signal Binvertido: std_logic_vector(3 downto 0);
16     signal Coutt: std_logic;
17
18     component inverter
19        port (X: in STD_LOGIC_VECTOR (3 downto 0);
20              Xbar: out STD_LOGIC_VECTOR (3 downto 0) );
21     end component;
22
23     component fourBitsAdder
24        port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
25             B : in  STD_LOGIC_VECTOR (3 downto 0);
26             Cin : in  STD_LOGIC;
27             S : out  STD_LOGIC_VECTOR (3 downto 0);
28             Cout : out  std_logic;
29             OV : out std_logic   );
30
31     end component;
32
33
34  begin
35
36     Inversor: inverter port map (B,Binvertido);
37     SUB: fourBitsAdder port map (A,Binvertido,'1',S, Coutt, OV);
38     Borrow <= not Coutt;
39
40  end Behavioral;
```

Figura 16: Subtrator

5.3 Somador 2 bits

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity fullAdder is
7       Port ( A : in  STD_LOGIC;
8              B : in  STD_LOGIC;
9              Cin : in  STD_LOGIC;
10             S : out  STD_LOGIC;
11             Cout : out  STD_LOGIC);
12  end fullAdder;
13
14  architecture Behavioral of fullAdder is
15
16  begin
17   S<= A xor B xor Cin;
18   Cout <= (A and B) or ( (A or B)  and Cin);
19
20
21  end Behavioral;
22
23
```

Figura 17: Somador 2 bits

5.4 Somador 4 bits

```
1
2
3   entity fourBitsAdder is
4       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
5              B : in  STD_LOGIC_VECTOR (3 downto 0);
6              Cin : in  STD_LOGIC;
7              S : out  STD_LOGIC_VECTOR (3 downto 0);
8              Cout : out  std_logic;
9              OV : out std_logic   );
10  end fourBitsAdder;
11
12  architecture Behavioral of fourBitsAdder is
13     signal c: std_logic_vector(4 downto 1);
14
15     component fullAdder
16        port (A,B,Cin: in std_logic;
17                  S,Cout: out std_logic);
18     end component;
19
20  begin
21
22     FA0: fullAdder port map (A(0),B(0),Cin,S(0),c(1));
23     FA1: fullAdder port map (A(1),B(1),c(1),S(1),c(2));
24     FA2: fullAdder port map (A(2),B(2),c(2),S(2),c(3));
25     FA3: fullAdder port map (A(3),B(3),c(3),S(3),c(4));
26
27     OV <= c(3) xor c(4);
28     Cout<=c(4);
29
30  end Behavioral;
31
32
```

Figura 18: Somador 4 bits

5.5 Porta And

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity Porta_And is
7       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
8              B : in  STD_LOGIC_VECTOR (3 downto 0);
9              S : out  STD_LOGIC_VECTOR (3 downto 0));
10  end Porta_And;
11
12  architecture andArch of Porta_And is
13
14  begin
15
16     process(B, A)
17     begin
18        for I in 0 to 3 loop
19           S(I) <= A(I) and B(I);
20        end loop;
21     end process;
```

Figura 19: Porta And

5.6 Contador

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7   entity contador is
8      port ( load : in bit;
9             clk  : in bit;
10            rst  : in bit;
11            data : in integer range 300000000 downto 0;
12            saida: out integer range 300000000 downto 0);
13
14  end contador;
15
16  architecture hardware of contador is
17
18  begin
19     contagem: process ( clk, rst )
20        variable contando : integer range 300000000 downto 0;
21        begin
22           if (rst = '1') then
23              contando := 0;
24           elsif (clk'event and clk = '1') then
25              if (load = '1') then
```

```
26                  contando := data;
27               else
28                  if (contando >= 300000000) then
29                     contando := 0;
30                  else
31                     contando := contando + 1;
32                  end if;
33               end if;
34            end if;
35            saida <= contando;
36         end process;
37
38   end hardware;
39
40
```

Figura 20: Contador

5.7 Incrementador

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7   entity incrementador is
8
9       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
10             S : out  STD_LOGIC_VECTOR (3 downto 0));
11
12  end incrementador;
13
14  architecture incrementadorArch of incrementador is
15
16     component fourBitsAdder
17
18        Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
19               B : in  STD_LOGIC_VECTOR (3 downto 0);
20               Cin : in  STD_LOGIC;
21               S : out  STD_LOGIC_VECTOR (3 downto 0);
22               Cout : out  std_logic;
23               OV : out std_logic   );
24     end component;
25
26  begin
27
28     incrementa1: fourBitsAdder port map (A, "0000", '1', S);
29
30  end incrementadorArch;
```

Figura 21: Incrementador

5.8 Multiplicador

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7  entity multiplicaFourBits is
8      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
9             B : in  STD_LOGIC_VECTOR (3 downto 0);
10            S : out  STD_LOGIC_VECTOR (3 downto 0));
11
12
13 end multiplicaFourBits;
14
15 architecture multiplicaArch of multiplicaFourBits is
16
17    signal soma0, soma1, soma2, soma3: std_logic_vector (3 downto 0 );
18    signal parte1_soma_final, parte2_soma_final, parte3_soma_final: std_logic_vector(7 downto 0);
19    signal parte4_soma_final, resultado_parcial01, resultado_parcial02: std_logic_vector(7 downto 0);
20    signal saida_real: std_logic_vector(7 downto 0);
21
22    component eightBitsAdder
23       Port (A : in std_logic_vector ( 7 downto 0);
24             B : in std_logic_vector ( 7 downto 0);
25             Cin: in std_logic;
26             S : out  STD_LOGIC_VECTOR (7 downto 0);
27             Cout : out  std_logic);
28
29    end component;
```

```
30
31
32  begin
33
34     process(B, A)
35     begin
36        for I in 0 to 3 loop
37           soma0(I) <= B(0) and A(I);
38
39           soma1(I) <= B(1) and A(I);
40
41           soma2(I) <= B(2) and A(I);
42
43           soma3(I) <= B(3) and A(I);
44        end loop;
45     end process;
46
47     parte1_soma_final <= "0000" & soma0;
48     parte2_soma_final <= "000" & soma1 & '0';
49     parte3_soma_final <= "00" & soma2 & "00";
50     parte4_soma_final <= '0' & soma3 & "000";
51
52     somaBit01: eightBitsAdder port map (parte1_soma_final,parte2_soma_final,'0',resultado_parcial01);
53     somaBit23: eightBitsAdder port map (parte3_soma_final,parte4_soma_final,'0',resultado_parcial02);
54     resultado_0123: eightBitsAdder port map (resultado_parcial01,resultado_parcial02,'0',saida_real);
55
56     S <= saida_real(3 downto 0);
57
58  end multiplicaArch;
59
```

Figura 22: Multiplicador

5.9 Inversora(NOT)

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity Inverter is
7       Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
8              Xbar : out  STD_LOGIC_VECTOR (3 downto 0));
9   end Inverter;
10
11  architecture Behavioral of Inverter is
12
13  begin
14
15          Xbar(0) <=  not X(0);
16          Xbar(1) <=  not X(1);
17          Xbar(2) <=  not X(2);
18          Xbar(3) <=  not X(3);
19
20
21
22  end Behavioral;
```

Figura 23: Not

5.10 OR

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7   entity Porta_Or is
8       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
9              B : in  STD_LOGIC_VECTOR (3 downto 0);
10             S : out  STD_LOGIC_VECTOR (3 downto 0));
11  end Porta_Or;
12
13  architecture OrArch of Porta_Or is
14
15  begin
16
17     process(B, A)
18     begin
19        for I in 0 to 3 loop
20           S(I) <= A(I) or B(I);
21        end loop;
22     end process;
```

Figura 24: OR

5.11 Somador de 8 bits

```
1
2
3   entity eightBitsAdder is
4      Port (   A : in std_logic_vector ( 7 downto 0);
5               B : in std_logic_vector ( 7 downto 0);
6               Cin: in std_logic;
7               S : out  STD_LOGIC_VECTOR (7 downto 0);
8               Cout : out  std_logic);
9
10  end eightBitsAdder;
11
12  architecture eightBitsArch of eightBitsAdder is
13     signal cout_signal: std_logic_vector(2 downto 1);
14
15     component fourBitsAdder
16        port (A,B: in std_logic_vector (3 downto 0);
17              Cin: in std_logic;
18              S  : out std_logic_vector (3 downto 0);
19              Cout: out std_logic;
20              OV : out std_logic );
21
22
23
24     end component;
```

```
26  begin
27
28     fourBitsAdder01: fourBitsAdder port map
29     (A(3 downto 0), B(3 downto 0), Cin, S(3 downto 0), cout_signal(1));
30
31     fourBitsAdder02: fourBitsAdder port map
32     (A(7 downto 4), B(7 downto 4), cout_signal(1), S(7 downto 4), cout_signal(2));
33
34     Cout <= cout_signal(2);
35
36  end eightBitsArch;
37
38
```

Figura 25: Somador de 8 bits

5.12 ULA

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6
7   entity ULA is
8       Port ( entradaX : in  STD_LOGIC_VECTOR (3 downto 0);
9              entradaY : in  STD_LOGIC_VECTOR (3 downto 0);
10             Cout : out  STD_LOGIC;
11             OV : out  STD_LOGIC;
12             SAIDA : out  STD_LOGIC_VECTOR (3 downto 0);
13             MUX_CONTROLADOR : in  STD_LOGIC_VECTOR (2 downto 0));
14  end ULA;
15
16  architecture Behavioral of ULA is
17
18     component Porta_Xor
19      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
20             B : in  STD_LOGIC_VECTOR (3 downto 0);
21             S : out  STD_LOGIC_VECTOR (3 downto 0));
22     end component;
23
24     component Porta_Or
25      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
26             B : in  STD_LOGIC_VECTOR (3 downto 0);
27             S : out  STD_LOGIC_VECTOR (3 downto 0));
28     end component;
29
30     component Porta_And
31      Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
32             B : in  STD_LOGIC_VECTOR (3 downto 0);
```

```
32              B : in  STD_LOGIC_VECTOR (3 downto 0);
33              S : out  STD_LOGIC_VECTOR (3 downto 0));
34      end component;
35
36      component Inverter
37       Port ( X : in  STD_LOGIC_VECTOR (3 downto 0);
38              Xbar : out  STD_LOGIC_VECTOR (3 downto 0));
39      end component;
40
41      component incrementador
42       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
43              S : out  STD_LOGIC_VECTOR (3 downto 0));
44      end component;
45
46      component multiplicaFourBits is
47          Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
48                 B : in  STD_LOGIC_VECTOR (3 downto 0);
49                 S : out  STD_LOGIC_VECTOR (3 downto 0));
50      end component;
51
52      component fourBitsSubtractor
53       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
54              B : in  STD_LOGIC_VECTOR (3 downto 0);
55              S : out  STD_LOGIC_VECTOR (3 downto 0);
56              Borrow : out  STD_LOGIC;
57              OV : out  STD_LOGIC);
```

```
58      end component;
59
60      component fourBitsAdder
61       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
62              B : in  STD_LOGIC_VECTOR (3 downto 0);
63              Cin : in  STD_LOGIC;
64              S : out  STD_LOGIC_VECTOR (3 downto 0);
65              Cout : out  std_logic;
66              OV : out std_logic   );
67      end component;
68
69      signal saidaSubtrator: STD_LOGIC_VECTOR (3 downto 0);
70      signal saidaSomador: STD_LOGIC_VECTOR (3 downto 0);
71      signal saidaMultiplicador: STD_LOGIC_VECTOR (3 downto 0);
72      signal saidaIncrementa_1: STD_LOGIC_VECTOR (3 downto 0);
73      signal saidaXor: STD_LOGIC_VECTOR (3 downto 0);
74      signal saidaOor: STD_LOGIC_VECTOR (3 downto 0);
75      signal saidaAand: STD_LOGIC_VECTOR (3 downto 0);
76      signal SaidaInverteer: STD_LOGIC_VECTOR (3 downto 0);
77      signal CoutSomador: STD_LOGIC;
78      signal CoutSubtrator: STD_LOGIC;
79      signal OVSomador: STD_LOGIC;
80      signal OVSubtractor: STD_LOGIC;
81
82  begin
83
84
85      Xoor:           Porta_Xor port map ( entradaX, entradaY, saidaXor);
86      Oor:            Porta_Or port map ( entradaX, entradaY, saidaOor);
87      Aand:           Porta_And port map ( entradaX, entradaY, saidaAand);
88      Inverteer:      Inverter port map ( entradaX, SaidaInverteer);
```

```
88      Inverteer:     Inverter port map ( entradaX, SaidaInverteer);
89
90
91      incrementa_1:  incrementador port map ( entradaX, saidaIncrementa_1);
92      multiplicador: multiplicaFourBits port map ( entradaX, entradaY, saidaMultiplicador
    );
93      subtrator:     fourBitsSubtractor port map ( entradaX, entradaY, saidaSubtrator,
    CoutSubtrator, OVSubtractor);
94      somador:       fourBitsAdder port map ( entradaX, entradaY, '0', saidaSomador,
    CoutSomador, OVSomador);
95
96      WITH MUX_CONTROLADOR select
97         SAIDA <= saidaSomador when "000",
98            saidaSubtrator when "001",
99            saidaMultiplicador when "010",
100           saidaIncrementa_1 when "011",
101           saidaXor when "100",
102           saidaOor when "101",
103           SaidaAand when "110",
104           SaidaInverteer when "111",
105           "0000" WHEN OTHERS;
106
107     with MUX_CONTROLADOR select
108        Cout <= CoutSomador when "000",
109           CoutSubtrator when "001",
110           '0' WHEN OTHERS;
111
```

Figura 26: ULA

5.13 XOR

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4
5
6   entity Porta_Xor is
7       Port ( A : in  STD_LOGIC_VECTOR (3 downto 0);
8              B : in  STD_LOGIC_VECTOR (3 downto 0);
9              S : out  STD_LOGIC_VECTOR (3 downto 0));
10  end Porta_Xor;
11
12  architecture xorArch of Porta_Xor is
13
14  begin
15
16     process(B, A)
17     begin
18        for I in 0 to 3 loop
19           S(I) <= A(I) xor B(I);
20        end loop;
21     end process;
22
23
24  end xorArch;
```

Figura 27: XOR

5.14 PINAGEM

```
1
2
3   NET "clk"       LOC = "E12"  | IOSTANDARD = LVCMOS33; #| PERIOD = 20.000 ;
4   #OFFSET = IN  10.000 VALID 20.000 BEFORE "CLK_50M" ;
5   #OFFSET = OUT 20.000 AFTER "CLK_50M" ;
6
7   #NET "CLK_AUX"       LOC = "V12"  | IOSTANDARD = LVCMOS33 | PERIOD = 20.000 ;
8   #OFFSET = IN  10.000 VALID 20.000 BEFORE "CLK_AUX" ;
9   #OFFSET = OUT 20.000 AFTER "CLK_AUX" ;
10
11  #NET "CLK_SMA"       LOC = "U12"  | IOSTANDARD = LVCMOS33 | PERIOD = 20.000 ;
12  #OFFSET = IN  10.000 VALID 20.000 BEFORE "CLK_SMA" ;
13  #OFFSET = OUT 20.000 AFTER "CLK_SMA" ;
14
15  ##############################################################################
16  # Discrete Indicators (LED)
17  ##############################################################################
18
19  NET "led_result"        LOC = "R20"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW
     ;
20  NET "ledB"        LOC = "T19"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
21  NET "ledA"        LOC = "U20"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
22  #NET "LED<3>"        LOC = "U19"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
23  NET "ledsToShowInAndOut(0)"        LOC = "V19"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
    SLEW = SLOW ;
24  NET "ledsToShowInAndOut(1)"        LOC = "V20"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
    SLEW = SLOW ;
25  NET "ledsToShowInAndOut(2)"        LOC = "Y22"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
    SLEW = SLOW ;
26  NET "ledsToShowInAndOut(3)"        LOC = "W21"  | IOSTANDARD = LVCMOS33 | DRIVE = 8 |
    SLEW = SLOW ;
```

```
27
28  ##############################################################################
29  # Directional Push-Buttons (BTN)
30  ##############################################################################
31
32  NET "rst"       LOC = "T16"  | IOSTANDARD = LVCMOS33 | PULLDOWN ;
33  #NET "BTN_NORTH"     LOC = "T14"  | IOSTANDARD = LVCMOS33 | PULLDOWN ;
34  NET "botaoEntradaDoNumeroY"     LOC = "T15"  | IOSTANDARD = LVCMOS33 | PULLDOWN ;
35  NET "botaoEntradaDoNumeroX"      LOC = "U15"  | IOSTANDARD = LVCMOS33 | PULLDOWN ;
36
37  ##############################################################################
38  # Rotary Knob (ROT)
39  ##############################################################################
40
41  #NET "ROT_CENTER"    LOC = "R13"  | IOSTANDARD = LVCMOS33 | PULLDOWN ;
42  #NET "ROT_A"         LOC = "T13"  | IOSTANDARD = LVCMOS33 | PULLUP ;
43  #NET "ROT_B"         LOC = "R14"  | IOSTANDARD = LVCMOS33 | PULLUP ;
44
45  ##############################################################################
46  # Mechanical Switches (SW)
47  ##############################################################################
48
49  NET "entradaUla(0)"         LOC = "V8"   | IOSTANDARD = LVCMOS33 ;
50  NET "entradaUla(1)"         LOC = "U10"  | IOSTANDARD = LVCMOS33 ;
51  NET "entradaUla(2)"         LOC = "U8"   | IOSTANDARD = LVCMOS33 ;
52  NET "entradaUla(3)"         LOC = "T9"   | IOSTANDARD = LVCMOS33 ;
```

Figura 28: PINAGEM