



Universidade Federal  
do Rio de Janeiro  
Escola Politécnica

EEL480 - Laboratório de Sistemas Digitais  
Prof. Luís Henrique Maciel

## ULA em VHDL

Christian Marques de Oliveira Silva  
**DRE:** 117.214.742

Rio de Janeiro  
2021

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Desenvolvimento</b>	<b>2</b>
2.1	Entradas e Saídas implementadas . . . . .	2
2.2	Forma de projeção . . . . .	3
2.3	Módulos . . . . .	4
2.3.1	Full-Adder . . . . .	4
2.3.2	Full-Adder de 4 bits . . . . .	5
2.3.3	Somador e Subtrator em C2 . . . . .	7
2.3.4	Contador . . . . .	9
2.3.5	Contador dos Operandos . . . . .	11
2.3.6	Display . . . . .	12
2.3.7	ULA . . . . .	14
2.3.8	Interface com o LabsLand . . . . .	16
<b>3</b>	<b>Conclusões</b>	<b>19</b>
<b>4</b>	<b>Funcionamento no Labsland</b>	<b>19</b>
4.1	A + B (em C2) . . . . .	19
4.2	A - B (em C2) . . . . .	20
4.3	Not A . . . . .	21
4.4	Not B . . . . .	21
4.5	A and B . . . . .	22
4.6	A or B . . . . .	22
4.7	A xor B . . . . .	23
4.8	A xnor B . . . . .	23
<b>5</b>	<b>Referências</b>	<b>24</b>

# 1 Introdução

O objetivo deste projeto foi produzir uma representação de uma Unidade Lógica e Aritmética (ULA) descrita em VHDL<sup>1</sup>. Este, simplificadamente se comporta como uma ULA que possui, de forma resumida, 2 operandos de entrada, um sinal de controle (que define a operação), o resultado, flags de sinalização e um sinal de sincronização. Abaixo segue uma visualização em blocos mais detalhada:

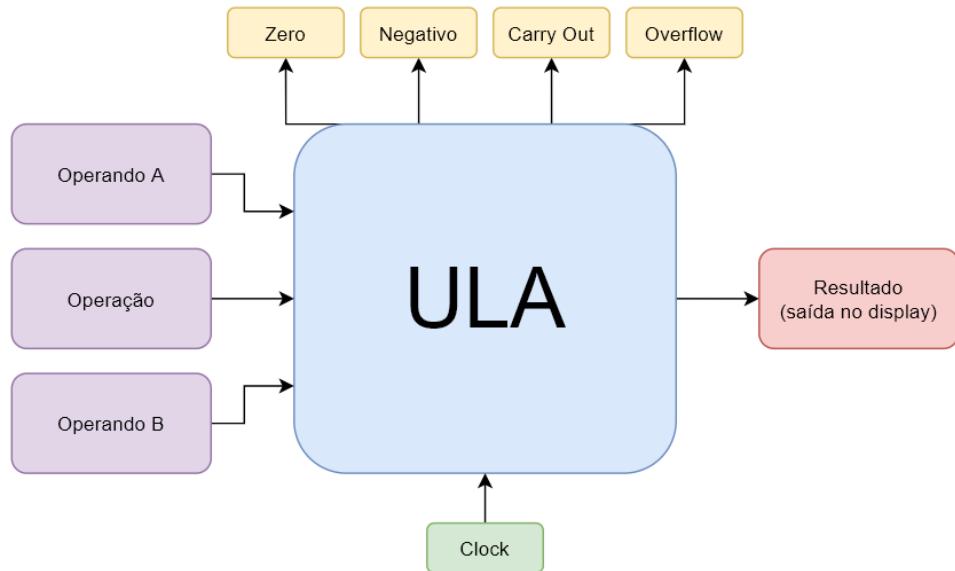


Figura 1: Diagrama de entradas e saídas inicial da ULA

- **Operandos:** Entradas dos operandos A e B, os quais possuem 4 bits e estão em C2<sup>2</sup>, assim, sua representação vai de -8 (1000) a +7 (0111);
- **Operação:** Entrada de controle, o qual possui 3 bits (permitindo 8 operações) para definir qual operação deve ser realizada entre o operando A e o operando B;
- **Resultado:** Saída de 7 bits (para o display de 7 segmentos) que contém o resultado da operação realizada;
- **Flags -** Saídas que indicam discretamente se a determinada situação ocorreu ou não;
- **Sinal de sincronização:** Entrada para sincronizar a atualização dos processos internos.

## 2 Desenvolvimento

### 2.1 Entradas e Saídas implementadas

Abaixo segue uma representação da hierarquia de componentes, em que o bloco que está no início da seta envia os sinais para o bloco que está no fim da mesma com o propósito de obter sua saída.

<sup>1</sup>Very High Speed Integrated Circuit

<sup>2</sup>Complemento de 2

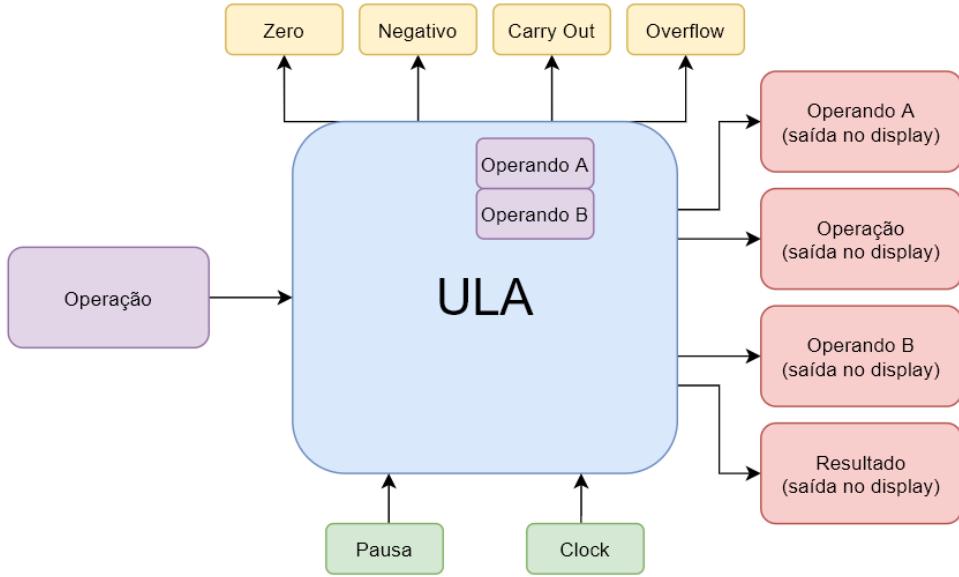


Figura 2: Diagrama de entradas e saídas da ULA implementada

Algumas alterações do modelo tradicional de uma ULA foram feitos para os testes práticos como pontuado a seguir:

- **Operandos:** Eles são gerados internamente por um contador que vai do valor mínimo (-8) ao máximo (+7);
- **Operação:** Os códigos que definem qual operação é realizada segue na tabela abaixo: Código<sub>bb</sub>

Código <sub>decimal</sub>	Código <sub>binário</sub>	Ação
0	000	$A + B$ (em C2)
1	001	$A - B$ (em C2)
2	010	Not $A$
3	011	Not $B$
4	100	$A$ and $B$
5	101	$A$ or $B$
6	110	$A$ xor $B$
7	111	$A$ xnor $B$

Tabela 1: Tabela dos códigos das operações na ULA

- **Saída no display:** Visualização dos dois operandos atuais ( $A$  e  $B$ ), o número decimal da operação e o resultado obtido;
- **Pausa:** Congela os operandos no mesmo valor. Isso ajuda nos testes práticos do LabsLand para fazer a prova real das operações.

## 2.2 Forma de projeção

A implementação no Quartus<sup>3</sup> foi realizada com um pensamento abstrato dos componentes a serem utilizados, não levando em conta (em sua maioria) os componentes físicos

<sup>3</sup>Software utilizado para a criação do código e teste do mesmo pela funcionalidade de formas de onda.

que existem em CI's<sup>4</sup> comerciais. Essa forma de abstração de funcionamento permitiu sua rápida criação.

Abaixo segue uma representação da hierarquia de componentes, em que o bloco que está no início da seta envia os sinais para o bloco que está no fim da mesma com o propósito de obter sua saída.

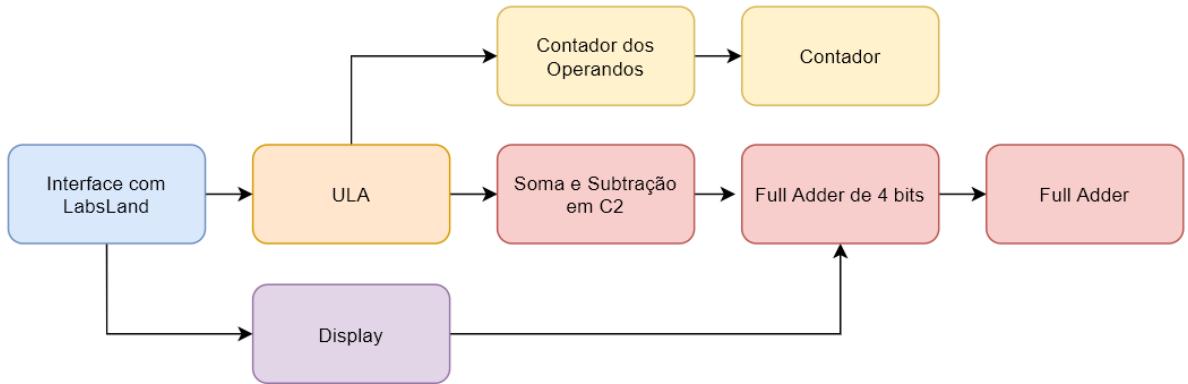


Figura 3: Diagrama geral de componentes

## 2.3 Módulos

Os módulos serão abordados do menor nível de hierarquia até o maior para facilitar a compreensão, uma vez que os mesmos não dependem de nenhum componente para a compreensão de seu funcionamento.

### 2.3.1 Full-Adder

#### 2.3.1.1 Descrição

Sua função é simplificar as duas primeiras operações da ULA como mencionado na tabela 1, pois, para uma soma ou subtração de números binários com N bits basta um Full-Adder de N bits. Deste modo, tendo posse desse componente genérico é possível criar o Full-Adder de N bits (nesse caso N=4) para que sejam possíveis essas operações. Abaixo segue um diagrama de entradas e saídas do módulo:



Figura 4: Diagrama de entradas e saídas do Módulo Full-Adder

As expressões foram obtidas tendo como base a teoria ensinada da matéria em questão, em que, para um Full-Adder de 1 bit, a saída é  $S = A \text{ xor } B \text{ xor } C_{in}$  e o carry propagado é  $C_{out} = (A \text{ and } C_{in}) \text{ or } (B \text{ and } C_{in}) \text{ or } (A \text{ and } B)$ .

---

<sup>4</sup>Circuitos Integrados

### 2.3.1.2 Código

É o único módulo inteiramente implementado com expressões lógicas.

```

11  -- Bibliotecas
12  library IEEE;
13  use IEEE.std_logic_1164.all;
14
15
16
17
18  -- Entidade Principal
19
20  entity FA is
21    port(
22      A, B, C_in : in  std_logic;
23      S, C_out   : out std_logic);
24  end FA;
25
26
27
28
29
30  -- Arquitetura Principal
31
32  architecture hardware of FA is
33  begin
34
35    S    <= A xor B xor C_in;
36    C_out <= (A and C_in) or (B and C_in) or (A and B);
37
38  end hardware;
39

```

Figura 5: Código em VHDL do Full-Adder (*FA.vhd*)

### 2.3.1.3 Simulação

Foi criada uma simulação somente compilando o arquivo '*FA.vhd*' e obteve-se o seguinte resultado:

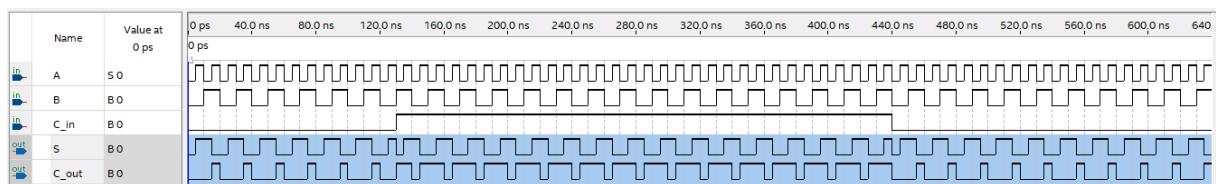


Figura 6: Simulação WF do módulo Full-Adder

Na simulação acima foi utilizado para entrada A um intervalo de **10ns** e para B de **20ns**, permitindo, assim, variar todas as possibilidades para as entradas (A e B). Já o  $C_{in}$  foi setado em 1 em um trecho da linha do tempo. E como pode ser visto, o módulo se comportou da forma correta tanto no  $S$  como no  $C_{out}$ .

## 2.3.2 Full-Adder de 4 bits

### 2.3.2.1 Descrição

É o módulo/componente intermediário entre o Full-Adder simples e o somador/subtrator em C2 implementado. Assim, conectada cada unidade criada para formar uma cadeia de Full-Adders, onde o  $C_{out}$  do anterior conecta no  $C_{in}$  do atual.



Figura 7: Diagrama de entradas e saídas do Módulo Full-Adder de 4 bits

### 2.3.2.2 Código

Basicamente faz uso de "port map's" para criar componentes de Full-Adder de 1 bit e, além do retorno do  $C_{out}$ , retorna o  $C_{out-1}$  para a identificação do **overflow**.

```

11  -- Bibliotecas
12  library IEEE;
13  use IEEE.std_logic_1164.all;
14
15
16
17  -- Entidade Principal
18  entity FA_4_bits is
19  port(
20      x,y : in  std_logic_vector(3 downto 0);
21      cin : in  std_logic;
22      cout : out std_logic;
23      cout_1: out std_logic;
24      z : out  std_logic_vector(3 downto 0));
25  end FA_4_bits;
26
27
28
29
30
31
32
33  -- Arquitetura Principal
34  architecture hardware of FA_4_bits is
35
36      -- Definicao do componente
37      component FA is
38          port(
39              A, B, C_in : in  std_logic;
40              S, C_out   : out std_logic);
41      end component;
42
43
44
45
46      signal carry : std_logic_vector (3 downto 0);
47
48  begin
49
50      a0: FA port map(x(0), y(0), cin, z(0), carry(0));
51      a1: FA port map(x(1), y(1), carry(0), z(1), carry(1));
52      a2: FA port map(x(2), y(2), carry(1), z(2), carry(2));
53      a3: FA port map(x(3), y(3), carry(2), z(3), carry(3));
54
55      -- Carry's out
56      cout_1 <= carry(2); -- carry out do penultimo bit
57      cout    <= carry(3); -- carry out do ultimo bit
58
59  end hardware;
60

```

Figura 8: Código em VHDL do Full-Adder de 4 bits (*FA\_4\_bits.vhd*)

### 2.3.2.3 Simulação

Foi criada uma simulação compilando os arquivos '*FA\_4\_bits.vhd*' e '*FA.vhd*', obtendo-se o seguinte resultado:

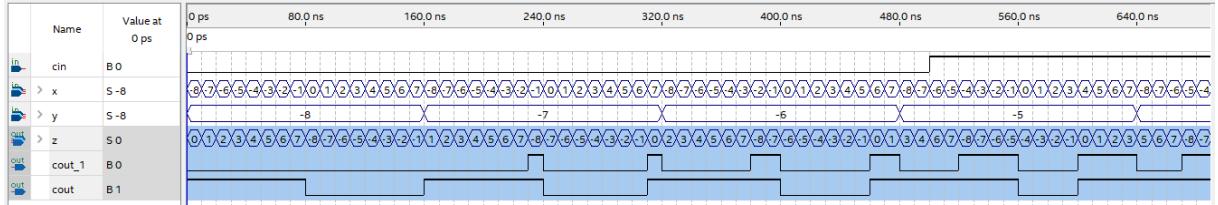


Figura 9: Simulação WF do módulo Full-Adder de 4 bits

Na simulação acima foram utilizados 2 contadores iniciando em -8. Para entrada X com intervalo de **10ns** e para Y de **160ns**, permitindo, assim, variar todas as possibilidades X antes de incrementar Y. Já o  $C_{in}$  foi setado em 1 em um trecho da linha do tempo. E como pode ser visto, o módulo se comportou da forma correta tanto no **Z** em  $C_{out}$  e  $C_{out-1}$ .

### 2.3.3 Somador e Subtrator em C2

#### 2.3.3.1 Descrição

O objetivo deste módulo é compreender as duas operações mais complexas da ULA, chamando o módulo de Full-Adder de 4 bits para facilitar sua função. Ainda, ele possui uma entrada para informar se a operação entre o operando A e B é soma ou subtração.

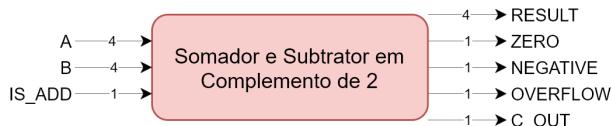


Figura 10: Diagrama de entradas e saídas do Módulo Somador e Subtrator em C2

#### 2.3.3.2 Código

A implementação desse módulo utiliza bastante a descrição discreta, se baseando nas expressões lógicas vistas na teoria desta disciplina. Ainda, além do resultado, esse módulo retorna 4 flags:

- **ZERO**: Indica se a saída é realmente zero para facilitar algum possível uso. Para isso, compara todos os bits de saída e verifica se não houve overflow;
- **NEGATIVE**: Indica se a saída é realmente um número negativo para facilitar algum possível uso. Para isso, compara o MSB<sup>5</sup> de saída e verifica se não houve overflow;
- **OVERFLOW**: Indica se a saída passou do limite de 4 bits em C2 (-8 a +7). Para isso, utilizando a teoria vista, compara o  $C_{out}$  com o  $C_{out-1}$  na expressão  $OVERFLOW = C_{out} \text{ xor } C_{out-1}$ ;
- **$C_{out}$** : Dependendo da operação, esse termo pode ser barrado<sup>6</sup>. Para isso, utilizando a teoria vista, se for **soma**  $C_{out} = C'_{out}$  e sendo **subtração**  $C_{out} = \overline{C'_{out}}$ .

<sup>5</sup>Bit Mais Significativo - neste caso, como a saída tem 4 bits seria o **S3** na sequência:  $S = S_3 S_2 S_1 S_0$ .

<sup>6</sup>É o valor binário inverso. Como só possui um bit, se for 1, o barrado é zero e vice-versa

```

11  -- Bibliotecas
12  library IEEE;
13  use IEEE.std_logic_1164.all;
14
15
16
17  -- Entidade Principal
18  entity Add_Sub_C2 is
19  port(
20      A,B           : in  std_logic_vector(3 downto 0);    -- Operandos de 4 bits
21      IS_ADD        : in  std_logic;                      -- Definição de soma ou subtração
22      ZERO,NEGATIVE,OVERFLOW : out std_logic;            -- Flags de indicação de zero, valor negativo e overflow, respectivamente
23      C_OUT         : out std_logic;                      -- Flag de carry out
24      RESULT        : out std_logic_vector(3 downto 0));   -- Resultado de 4 bits dos 2 operandos
25
26
27 end Add_Sub_C2;
28
29
30
31
32
33  -- Arquitetura Principal
34  architecture hardware of Add_Sub_C2 is
35
36  component FA_4_bits is
37  port(
38      x,y           : in  std_logic_vector(3 downto 0);
39      cin           : in  std_logic;
40      cout          : out std_logic;
41      cout_1        : out std_logic;
42      z             : out std_logic_vector(3 downto 0));
43  end component;
44
45
46  signal operatorB,res  : std_logic_vector(3 downto 0);
47  signal signalRES     : std_logic := res(3);
48  signal c_in          : std_logic;
49  signal c_out_0        : std_logic;                    -- Carry out do último bit e carry out do bit anterior
50  signal c_out_1        : std_logic;                    -- overflow
51  signal over          : std_logic;                   -- overflow
52
53 begin
54
55  -- Prepara o segundo operando para a operação (inverte caso seja uma subtração)
56  with IS_ADD select operatorB <=
57      B when '1'; -- Soma
58      not B when '0'; -- Subtração
59
60  -- Prepara o carry in para a operação (1 caso seja uma subtração)
61  c_in <= not
62
63  Full_Adder:                               .in, c_out_0, c_out_1, res);
64
65  -- Caso seja uma adição, a saída e o cout;
66  -- Caso seja uma subtração, a saída é cout barrado (not cout)
67  c_out <= ((not IS_ADD) and c_out_0) or (IS_ADD and (not c_out_0));
68
69  -- caso o carry do bit atual e do anterior sejam diferentes, houve overflow
70  over <= c_out_0 xor c_out_1;
71
72  -- Se não houve overflow e o bit mais significativo do resultado for 1, o número é negativo
73  -- 1000 (-8) a 1111 (-1)
74  NEGATIVE <= signalRES and (not over);
75
76  -- Se não houve overflow e todos os bits de resultado são 0, a saída realmente é zero
77  ZERO <= (not over) and (not res(3)) and (not res(2)) and (not res(1)) and (not res(0));
78
79  -- Passagem dos sinais para as saídas
80  OVERFLOW <= over;
81  RESULT <= res;
82
83 end hardware;

```

Figura 11: Código em VHDL do Somador e Subtrator em C2 (*Add\_Sub\_C2.vhd*)

### 2.3.3.3 Simulação

Foi criada uma simulação compilando os arquivos '*Add\_Sub\_C2.vhd*', '*FA\_4\_bits.vhd*' e '*FA.vhd*', obtendo-se o seguinte resultado:

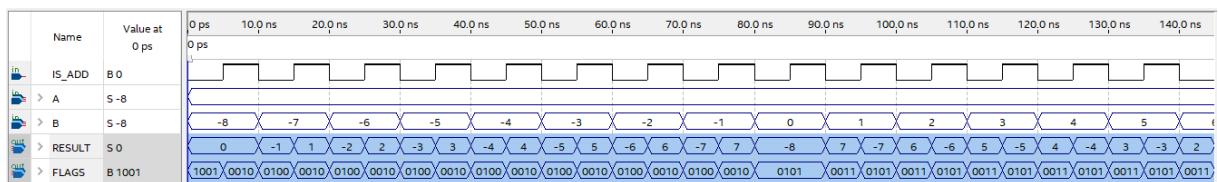


Figura 12: Simulação WF do módulo Somador e Subtrator em C2

Na simulação acima foram utilizados 2 contadores iniciando em -8. Para entrada A com intervalo de **160ns** e para B de **10ns**, permitindo, assim, variar todas as possibilidades B antes de incrementar A. Já o *IS\_ADD* esteve variando entre 0 e 1 em cada incremento de B, logo, foi possível detectar cada operação sendo somada ou subtraída. Ainda, as FLAGs de saída foram agrupadas para facilitar a leitura a ordem correta de leitura sendo *ZER*, *NEG*, *OVER*, *C<sub>OUT</sub>*. E como pode ser visto, o módulo se comportou da forma correta tanto no **S** como em **C<sub>out</sub>**.

### 2.3.4 Contador

#### 2.3.4.1 Descrição

O propósito deste módulo é reduzir o clock de 50MHz de entrada para que as operações ocorram no display do LabsLand com intervalo de 1s – 2s para ser perceptível. Logo, o mesmo pode também ser chamado de um divisor de clock, uma vez que sua única função é diminuir a frequência de atualização dos estados de mudança.



Figura 13: Diagrama de entradas e saídas do Módulo Contador

Outro fator interessante implementado foi o botão de *pause*, pois nos testes do LabsLand tornou-se muito prático validar se as operações estavam sendo feitas corretamente.

#### 2.3.4.2 Código

O código de divisor de clock implementado baseou-se na vídeo-aula de *"LED BLINK EM VHDL — Curso de FPGA #014"* do Wagner Rambo 2021, em que há um *contador geral do clock* e um *contador interno* para incrementar o valor de saída. O botão de **pause** está como um "filtro", só permitindo os incrementos caso esteja em nível lógico baixo.

Outro ponto importante foi criação deste módulo como entidade genérica. Isso foi feito para que facilite o aumento e diminuição de frequência de incremento dos operandos A e B, uma vez que, para passar por todas as possibilidades, um deles precisa estar 16 vezes mais rápido que o outro. Desta forma, o argumento recebido *times* serve para multiplicar o valor máximo calculado, visando aumentar esse incremento no operando mais lento.

Ainda, agindo diretamente no incremento do operando, na própria definição da variável *counter\_output*, o operando é setado para começar com o -8 em C2. Além de haver uma situação de contorno para quando o valor do registrador é o maior possível a nível dos bits (1111) e executa-se um incremento. Nesta situação o registrador é zerado (0000).

A obtenção do tempo entre o incremento (variável *maximum*) dos operandos foi feito da seguinte forma:

$$\begin{aligned}
 Freq &= 50Mhz \rightarrow T_{clk} = T_{prescaler} = 2 \times 10^{-8}s \\
 T_{objetivo} &= 2s \\
 maximum &= \frac{T_{objetivo}}{T_{prescaler}} \\
 maximum &= \frac{2s}{2 \times 10^{-8}s} \therefore maximum = 1 \times 10^8
 \end{aligned}$$

Já para a obtenção do tamanho ( $N$ ) do registrador do prescaler (o qual não pode resetar antes de atingir o valor máximo) ocorreu através do seguinte raciocínio para utilizar  $N=31$ :

$$\begin{aligned}
 maximum &= 1 \times 10^8 \\
 2^N \geq maximum \rightarrow \log_2(2^N) \geq \log_2(maximum) \rightarrow N \geq \log_2(maximum) \\
 \log_2(maximum) &= \log_2(1 \times 10^8) = 29.9 \\
 &\therefore \\
 N &\geq 29.9 \approx 30
 \end{aligned}$$

```

13  -- Bibliotecas
14  library IEEE;
15  use IEEE.std_logic_1164.all;
16  use IEEE.numeric_std.all;
17  use IEEE.std_logic_unsigned.all;
18
19
20
21
22  -- Entidade Principal
23
24  entity counter is
25    generic(times : integer := 1); -- Vezes em que o limite máximo será multiplicado para possibilitar um incremento
26    port(
27      clk       : in  std_logic;           -- Clock de entrada do sistema
28      pause     : in  std_logic;           -- Evita que o contador aumente (estagnando os 2 operandos)
29      q        : out std_logic_vector(3 downto 0)); -- Operando de Saída
30  end counter;
31
32
33
34
35  -- Arquitetura Principal
36
37  architecture hardware of counter is
38
39  begin
40
41    counter_process : process(clk)
42      variable counter_output : std_logic_vector(3 downto 0) := "1000"; -- Contador da saída (-8 a 7)
43      variable prescaler: std_logic_vector(30 downto 0) := (others=>'0'); -- Contador p/ redução da atualização de saída (-1 seg)
44      variable maximum : integer := 100000000; -- Contador p/ redução da atualização de saída (-1 seg)
45
46    begin
47
48      if (rising_edge(clk) and (pause /= '1')) then -- Toda vez que houver um pulso de clock e não estiver no modo pausa
49        prescaler := prescaler + 1;
50
51      if (prescaler = (maximum * times)) then
52        prescaler := (others=>'0');
53
54        if (counter_output = "1111") then counter_output := "0000"; -- Zera o contador quando chegar no limite
55        else counter_output := counter_output + 1; -- Incrementa o contador a cada pulso de clock
56        end if;
57
58      end if;
59
60      -- Atualiza a saída com o valor do contador
61      q <= counter_output;
62
63    end process;
64
65  end hardware;
66
67
68

```

Figura 14: Código em VHDL do Contador (*counter.vhd*)

### 2.3.4.3 Simulação

Foi criada uma simulação compilando somente o arquivo '*counter.vhd*', obtendo-se o seguinte resultado:

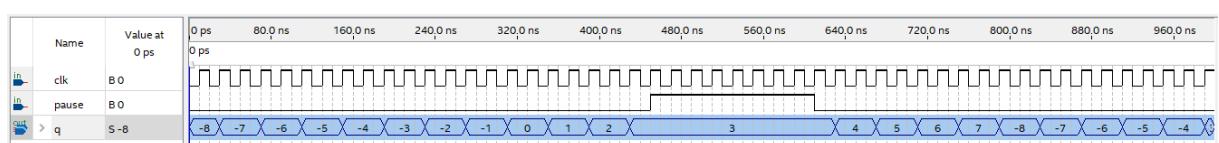


Figura 15: Simulação WF do módulo Contador

Na simulação acima foi inserido um sinal de frequência de 50 MHz na entrada *clk* e na entrada de *pause* foi inserido 1 em um trecho da linha do tempo. E como pode ser visto, o módulo se comportou da forma correta para a saída *q*, incrementando o valor (que começa em -8) e pausando/retomando a contagem nos momentos adequados.

### 2.3.5 Contador dos Operandos

#### 2.3.5.1 Descrição

A sua função é bem simples uma vez que ele cria 2 componentes de contador, um para o operando A e outro para o B. Como o A irá ficar no display à esquerda, o tempo de atualização do mesmo precisa ser 16 vezes maior do que o operando B.

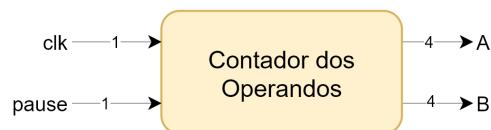


Figura 16: Diagrama de entradas e saídas do Módulo Contador dos Operandos

#### 2.3.5.2 Código

A título de implementação bastou criar os 2 componentes genéricos passando os multiplicadores do tempo de estouro do *prescaler*, sendo  $times_A=16$  (para ser seu incremento ser mais lento) e  $times_B=1$ .

```

13  |
14  |   |-- Bibliotecas
15  |   library IEEE;
16  |   use IEEE.std_logic_1164.all;
17  |
18  |
19  |
20  |   |-- Entidade Principal
21  |   entity counter_2_numbers is
22  |       port(
23  |           clk      : in  std_logic;
24  |           pause    : in  std_logic;
25  |           A, B    : out std_logic_vector(3 downto 0));
26  |       end port;
27  |   end entity;
28  |
29  |
30  |
31  |
32  |
33  |   |-- Arquitetura Principal
34  |   architecture hardware of counter_2_numbers is
35  |       -- Definicao do componente
36  |       component counter
37  |           generic(times : integer := 1); -- Vezes em que o limite máximo será multiplicado para possibilitar um incremento
38  |           port(
39  |               clk      : in  std_logic;          -- Clock de entrada do sistema
40  |               pause    : in  std_logic;          -- Evita que o contador aumente (estagnando os 2 operandos)
41  |               q        : out std_logic_vector(3 downto 0)); -- Operando de Saída
42  |           end component;
43  |
44  |       begin
45  |           A_counter: counter generic map(16)    port map(clk, pause, A);
46  |           B_counter: counter generic map(1)     port map(clk, pause, B);
47  |       end begin;
48  |
49  |
50  |
51  |
52  |

```

Figura 17: Código em VHDL do Contador de Operandos (*counter\_2\_numbers.vhd*)

### 2.3.5.3 Simulação

Foi criada uma simulação compilando os arquivos '*counter\_2\_numbers.vhd*' e '*counter.vhd*', obtendo-se o seguinte resultado:

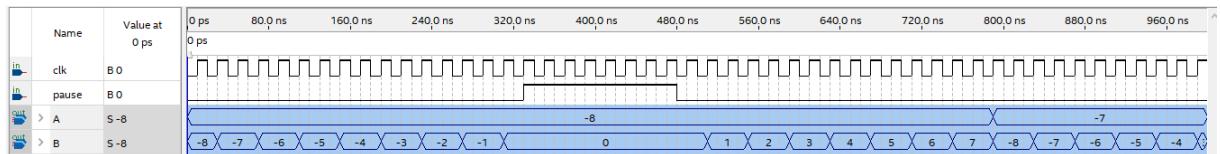


Figura 18: Simulação WF do módulo Contador de Operandos

Na simulação acima foi inserido um sinal de frequência de 50 MHz na entrada *clk* e na entrada de *pause* foi inserido 1 em um trecho da linha do tempo. E como pode ser visto, o módulo se comportou da forma correta para as saídas **A** e **B**, incrementando os valores (que começam em -8), com **A** sendo 16x mais lento do que **B** e pausando/retomando a contagem nos momentos adequados.

### 2.3.6 Display

#### 2.3.6.1 Descrição

O objetivo deste módulo é facilitar a visualização na tela dos números exibidos, tanto os operandos, como a operação quanto o resultado. Outra ação realizada foi a extração do módulo de um número caso o mesmo seja negativo (Ex.:  $[-6 \rightarrow 1010]$  em módulo fica  $+6 \rightarrow 0110$ ).



Figura 19: Diagrama de entradas e saídas do Módulo Display

#### 2.3.6.2 Código

Como os displays do LabsLand 2021 possuem Anodo Comum<sup>7</sup>, o nível lógico de ativação é baixo (0).

Para a extração do módulo do número foi criado um componente do Full-Adder de 4 bits para que haja uma subtração do valor +7 (0111) e uma inversão dos bits para obter o valor correto do módulo (Ex.:  $[-6 \rightarrow 1010]$  e  $[+7 \rightarrow 0111]$ , passando pelo FA\_4bits temos  $[+1 \rightarrow 0001]$ , no entanto, se invertendo os bits e desconsiderando o MSB temos:  $[+6 \rightarrow 110]$ ).

<sup>7</sup>Todos os pinos positivos dos segmentos conectados pelo positivo

```

11  -- Bibliotecas
12  library IEEE;
13  use IEEE.std_logic_1164.all;
14  use IEEE.numeric_std.all;
15  use IEEE.std_logic_unsigned.all;
16
17
18
19
20  -- Entidade Principal
21
22  entity display is
23    port(
24      VALUE_IN    : in   std_logic_vector(3 downto 0);
25      DISPLAY_OUT : out  std_logic_vector(6 downto 0));
26  end display;
27
28
29
30
31
32  -- Arquitetura Principal
33
34  architecture hardware of display is
35
36
37  -- Definição de componentes
38
39  component FA_4_bits is
40    port (
41      x,y    : in   std_logic_vector(3 downto 0);
42      cin   : in   std_logic;
43      cout  : out  std_logic;
44      z     : out  std_logic_vector(3 downto 0));
45  end component;
46
47
48
49  -- Definição de variáveis
50
51  signal VALUE_IN_sub : std_logic_vector(3 downto 0);
52  signal VALUE_IN_proc : std_logic_vector(3 downto 0);
53  signal less_one_in_FA : std_logic_vector(3 downto 0) := "0111";
54
55
56
57 begin
58
59  FA: FA_4_bits port map(VALUE_IN, less_one_in_FA, '0', open, VALUE_IN_sub); -- Subtrai 1
60
61  process(VALUE_IN)
62    variable NEG : std_logic := VALUE_IN(3); -- variável que identifica se o valor é negativo
63
64  begin
65    -- Caso o valor seja o número '8', que é um caso especial ou seja um número positivo
66    if (VALUE_IN = "1000" or NEG = '0') then
67      VALUE_IN_proc <= VALUE_IN;
68
69    -- Caso o valor seja um número negativo
70    else
71      -- Realiza o inverso do complemento de 2 para obter a magnitude do valor
72      VALUE_IN_proc <= not VALUE_IN_sub; -- Inverte todos os bits
73      VALUE_IN_proc(3) <= '0'; -- Faz uma 'máscara' para só obter de 0 a 7
74
75    end if;
76
77  case VALUE_IN_proc is
78    -- 0123456
79    when "0000" => DISPLAY_OUT <= "1000000"; -- "0"
80    when "0001" => DISPLAY_OUT <= "1111001"; -- "1"
81    when "0010" => DISPLAY_OUT <= "0100100"; -- "2"
82    when "0011" => DISPLAY_OUT <= "0110000"; -- "3"
83    when "0100" => DISPLAY_OUT <= "0011001"; -- "4"
84    when "0101" => DISPLAY_OUT <= "0010010"; -- "5"
85    when "0110" => DISPLAY_OUT <= "0000010"; -- "6"
86    when "0111" => DISPLAY_OUT <= "1111000"; -- "7"
87    when "1000" => DISPLAY_OUT <= "0000000"; -- "8"
88    when others => DISPLAY_OUT <= "1111111"; -- "apagado"
89  end case;
90
91  end process;
92
93 end hardware;

```

Figura 20: Código em VHDL do Display (*display.vhd*)

### 2.3.7 ULA

#### 2.3.7.1 Descrição

Assim como seu nome, é o coração do sistema. Responsável por gerenciar qual é a saída de acordo com a entrada de seleção, dividindo em 2 partes: saída em C2 e binária. Sempre são gerados os resultados das operações em C2, mas a saída deste módulo é definida pela configuração da entrada de 3 bits de seleção.

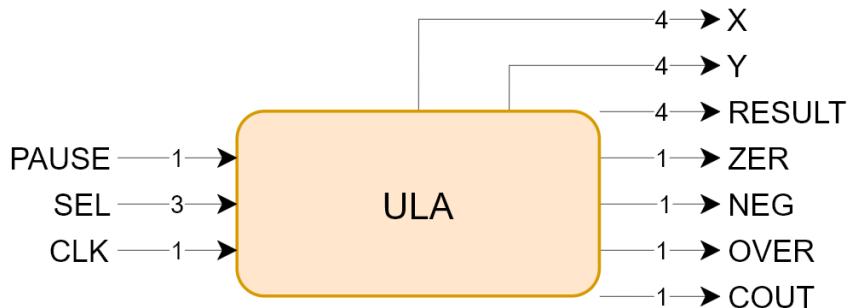


Figura 21: Diagrama de entradas e saídas do Módulo ULA

#### 2.3.7.2 Código

A **saída em C2** cria um componente *Somador e Subtrator em C2* para a operação de soma (passando IS\_ADD=1) e outro para a subtração (passando IS\_ADD=0), no qual a seleção define qual resultado será direcionado para a saída.

Já a **saída binária** inicialmente zera as flags *NEG*, *OVER* e *C<sub>OUT</sub>*, define qual operação irá para a saída e, após isso, verifica se todos os bits são zero (0000) para acionar a flag *ZER*=1.

Ainda, antes de realizar qualquer operação e a partir das entradas de *pause* e *clk*, cria-se o componente que gera os 2 operandos (A e B) internamente com o tempo de incremento correto.

```

24  -- Bibliotecas
25  library IEEE;
26  use IEEE.std_logic_1164.all;
27
28
29
30
31  -- Entidade Principal
32  entity ULA is
33  port(
34      A,B           : in  std_logic_vector(3 downto 0);    -- Operandos de 4 bits (simulação no quartus)
35      X,Y           : out std_logic_vector(3 downto 0);   -- Operandos de 4 bits
36      SEL            : in  std_logic_vector(2 downto 0);   -- Definição da seleção de operação
37      PAUSE          : in  std_logic;                   -- Para temporariamente o incremento dos operandos
38      CLK             : in  std_logic;                   -- Clock para o funcionamento do sistema
39      ZER,NEG,OVER  : out std_logic;                   -- Flags de indicação de zero, valor negativo e overflow, respectivamente
40      COUT           : out std_logic;                   -- Flag de Carry Out
41      RESULT         : out std_logic_vector(3 downto 0); -- Resultado de 4 bits dos 2 operandos
42  end ULA;
43
44
45
46
47
48
49  -- Arquitetura Principal
50  architecture hardware of ULA is
51
52  -- Definição de componentes
53
54  component Add_Sub_C2 is
55  port (
56      A,B           : in  std_logic_vector(3 downto 0);    -- Operandos de 4 bits
57      IS_ADD        : in  std_logic;                   -- Definição de soma ou subtração
58      ZERO,NEGATIVE,OVERFLOW : out std_logic;       -- Flags de indicação de zero, valor negativo e overflow, respectivamente
59      C_OUT          : out std_logic;                   -- Flag de Carry Out
60      RESULT         : out std_logic_vector(3 downto 0)); -- Resultado de 4 bits dos 2 operandos
61  end component;
62
63  component counter_2_numbers is
64  port (
65      clk            : in  std_logic;                   -- Clock de entrada do sistema
66      pause          : in  std_logic;                   -- Evita que o contador aumente (estagnando os 2 operandos)
67      A, B           : out std_logic_vector(3 downto 0)); -- Operandos de Saída
68
69
70
71  begin
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155

```

Figura 22: Código em VHDL da ULA (*ULA.vhd*)

### 2.3.7.3 Simulação

Foi criada uma simulação compilando os arquivos '*Add\_Sub\_C2.vhd*', '*FA\_4\_bits.vhd*', '*FA.vhd*', '*counter\_2\_numbers.vhd*' e '*counter.vhd*', obtendo-se o seguinte resultado:

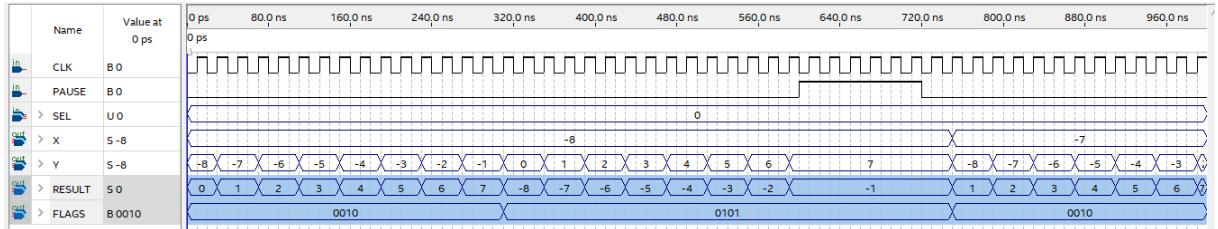


Figura 23: Simulação WF do módulo ULA

Na simulação acima foi inserido um sinal de frequência de 50 MHz na entrada *clk*, na entrada de *pause* foi inserido 1 em um trecho da linha do tempo e a seleção foi configurada para a operação de soma (*SEL=000*). E como pode ser visto, o módulo se comportou da forma correta ao gerar os operandos **X** e **Y** (incrementando seus valores, os quais começam em -8 e com **X** sendo 16x mais lento do que **Y**), além do resultado das operações e as *FLAGS* estarem condizentes, pausando/retomando a contagem nos momentos adequados.

### 2.3.8 Interface com o LabsLand

#### 2.3.8.1 Descrição

O objetivo deste módulo é encapsular todos os demais componentes já criados para interagir com o LabsLand 2021. Outra utilidade que possui é exibir os caracteres especiais que facilitam a visualização das operações em C2, como os sinais de negativo nos operandos e resultado (caso haja) e o sinal de igualdade.

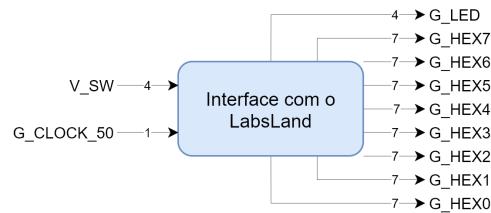


Figura 24: Diagrama de entradas e saídas do Módulo Interface com o LabsLand

A configuração de visualização nos displays foi escolhida visando facilitar ao máximo a compreensão do que está havendo no momento, como mostrado abaixo. Também vale ressaltar que os registradores  $G_{HEX}N$  vão de 7 a 0, da esquerda para a direita respectivamente. Logo, o  $G_{HEX}7$  é o **sinal do operando A** e o  $G_{HEX}0$  é o **módulo do resultado da operação**.



Figura 25: Configuração do painel de displays no LabsLand

### 2.3.8.2 Código

A implementação tornou-se bem simples com o uso de *signal's*, pois toda a lógica já estava implementada nos componentes criados neste módulo, restando somente o preenchimento correto e utilização para as saídas nas ocasiões devidas.

Na parte dos caracteres especiais (sinal de número negativo e sinal de igualdade), sempre exibe o '=' pois até mesmo nas operações binárias é necessário, porém, os displays para indicação de número negativo só acendem o segmento central quando seu MSB é 1. Caso isso não ocorra, o display fica apagado, representando que o número é positivo.

```

24
25     -- Bibliotecas
26
27     library IEEE;
28     use IEEE.std_logic_1164.all;
29
30
31     -- Entidade Principal
32
33     entity labsland is
34         port(
35             V_SW          : in  std_logic_vector(3 downto 0); -- Entradas do sistema (Seleção e pausa)
36             G_LED         : out std_logic_vector(3 downto 0); -- Flags de indicação (zero, negativo, overflow e carry out)
37             G_CLOCK_50   : in  std_logic;                      -- Clock para o funcionamento do sistema
38             G_HEX7       : out std_logic_vector(6 downto 0); -- Sinal A
39             G_HEX6       : out std_logic_vector(6 downto 0); -- Operando A
40             G_HEX5       : out std_logic_vector(6 downto 0); -- Sinal B
41             G_HEX3       : out std_logic_vector(6 downto 0); -- Operação
42             G_HEX4       : out std_logic_vector(6 downto 0); -- Operando B
43             G_HEX2       : out std_logic_vector(6 downto 0); -- Igualdade
44             G_HEX1       : out std_logic_vector(6 downto 0); -- Sinal do Resultado
45             G_HEX0       : out std_logic_vector(6 downto 0)); -- Resultado da operação
46
47     end labsland;
48
49
50     -- Arquitetura Principal
51
52     architecture hardware of labsland is
53
54         -- Definição do componente
55         component ULA is
56             port(
57                 X,Y           : out std_logic_vector(3 downto 0); -- Operandos de 4 bits
58                 SEL          : in  std_logic_vector(2 downto 0); -- Para temporariamente o incremento dos operandos
59                 PAUSE        : in  std_logic;                      -- Clock para o funcionamento do sistema
60                 CLK          : in  std_logic;
61                 ZER,NEG,OVER : out std_logic;
62                 COUT         : out std_logic;
63                 RESULT       : out std_logic_vector(3 downto 0)); -- Resultado de 4 bits dos 2 operandos
64
65         end component;
66
67         component display is
68             port(
69                 VALUE_IN    : in  std_logic_vector(3 downto 0);
70                 DISPLAY_OUT : out std_logic_vector(6 downto 0));
71         end component;
72
73
74         -- Conversões
75         signal PAUSE,ZER,NEG,OVER,COUT           : std_logic;
76         signal SEL                           : std_logic_vector(2 downto 0);
77         signal A,B,RESULT                   : std_logic_vector(3 downto 0);
78         signal DISPLAY_A,DISPLAY_B,DISPLAY_RES,DISPLAY_OP : std_logic_vector(6 downto 0);
79         signal DISPLAY_SA,DISPLAY_SB,DISPLAY_SRES,DISPLAY_IG : std_logic_vector(6 downto 0);
80
81
82         -- Caracteres de exibição no display
83         signal display_neg      : std_logic_vector(6 downto 0) := "011111"; -- hifen '-' (Números negativos)
84         signal display_pos      : std_logic_vector(6 downto 0) := "111111"; -- vazio ''
85         signal display_eq       : std_logic_vector(6 downto 0) := "011011"; -- Igual '='
86
87     begin
88
89         -- Renomeia a seleção de operação
90         SEL <= V_SW(2 downto 0);
91         -- Renomeia o botão de pausa
92         PAUSE <= V_SW(3);
93
94         ULA_calculate: ULA port map(A,B,SEL,PAUSE,G_CLOCK_50,ZER,NEG,OVER,COUT,RESULT);
95
96         disp_A: display port map(A, DISPLAY_A);
97         disp_B: display port map(B, DISPLAY_B);
98         disp_OP: display port map('0' & SEL, DISPLAY_OP);
99         disp_RES: display port map(RESULT, DISPLAY_RES);
100
101         process(A,B,RESULT)
102             begin
103                 if (A(3) = '1') then -- Operando A é negativo
104                     DISPLAY_SA <= display_neg;
105                 else -- Operando A é positivo
106                     DISPLAY_SA <= display_pos;
107                 end if;
108
109                 if (B(3) = '1') then -- Operando B é negativo
110                     DISPLAY_SB <= display_neg;
111                 else -- Operando B é positivo
112                     DISPLAY_SB <= display_pos;
113                 end if;
114
115                 if (RESULT(3) = '1') then -- Resultado é negativo
116                     DISPLAY_SRES <= display_neg;
117                 else -- Resultado é positivo
118                     DISPLAY_SRES <= display_pos;
119                 end if;
120             end process;
121
122             -- Atualização das saídas
123             DISPLAY_IG <= display_eq;
124             G_HEX7 <= DISPLAY_SA;
125             G_HEX6 <= DISPLAY_A;
126             G_HEX5 <= DISPLAY_SB;
127             G_HEX4 <= DISPLAY_B;
128             G_HEX3 <= DISPLAY_OP;
129             G_HEX2 <= DISPLAY_IG;
130             G_HEX1 <= DISPLAY_SRES;
131             G_HEX0 <= DISPLAY_RES;
132             G_LED <= ZER & NEG & OVER & COUT;
133
134     end hardware;

```

Figura 26: Código em VHDL da Interface com o LabsLand (*labsland.vhd*)

### 3 Conclusões

O código projetado não apresentou falhas após seu desenvolvimento, cumprindo exatamente o proposto para sua confecção.

O único momento em que houve um "erro" foi, em um dos testes, quando o operando A apareceu como um número positivo sendo que o mesmo deveria ser negativo. A causa desse erro foi porque a FPGA do LabsLand 2021 estava com o segmento central do display de sinal queimado, impossibilitando a visualização do mesmo aceso. A comprovação foi feita testando em outra placa, onde o código voltou a funcionar.

Ainda sobre erros do LabsLand 2021, a documentação do mesmo informa que a ordem dos segmentos dos displays é inversa ao funcionamento real, onde a ordem do MSB ao LSB está trocada.

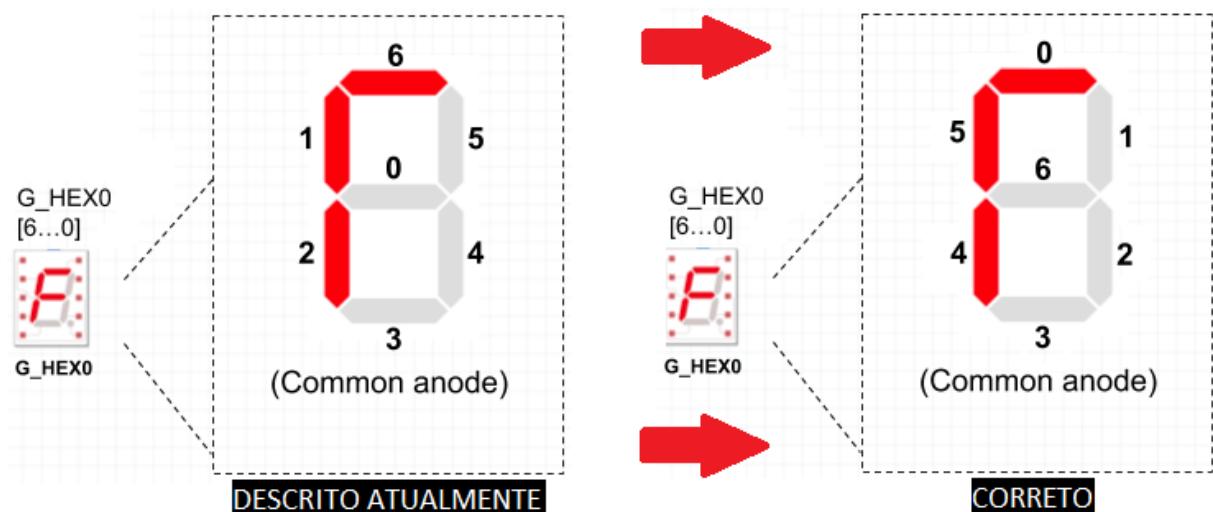


Figura 27: Correção da documentação do LabsLand 2021 para a ordem do display de 7 segmentos

A cima pode ser visto que o MSB do registrador  $G\_HEX0$  (6) é o segmento descrito como 6, no entanto, o que realmente surte efeito no MSB é o segmento descrito inicialmente como 0. E o mesmo equivale para as outras alterações. O único que se manteve foi o segmento 3. Após realizar as alterações a interação funcionou corretamente.

### 4 Funcionamento no Labsland

#### 4.1 A + B (em C2)

Abaixo, como o display  $G\_HEX3$  exibe 0, a operação de soma ocorre:  $-7 + +2 = -7 + 2 = -5$ , obtendo **NEGATIVE=1** e **C<sub>OUT</sub>=1**. O que está correto.

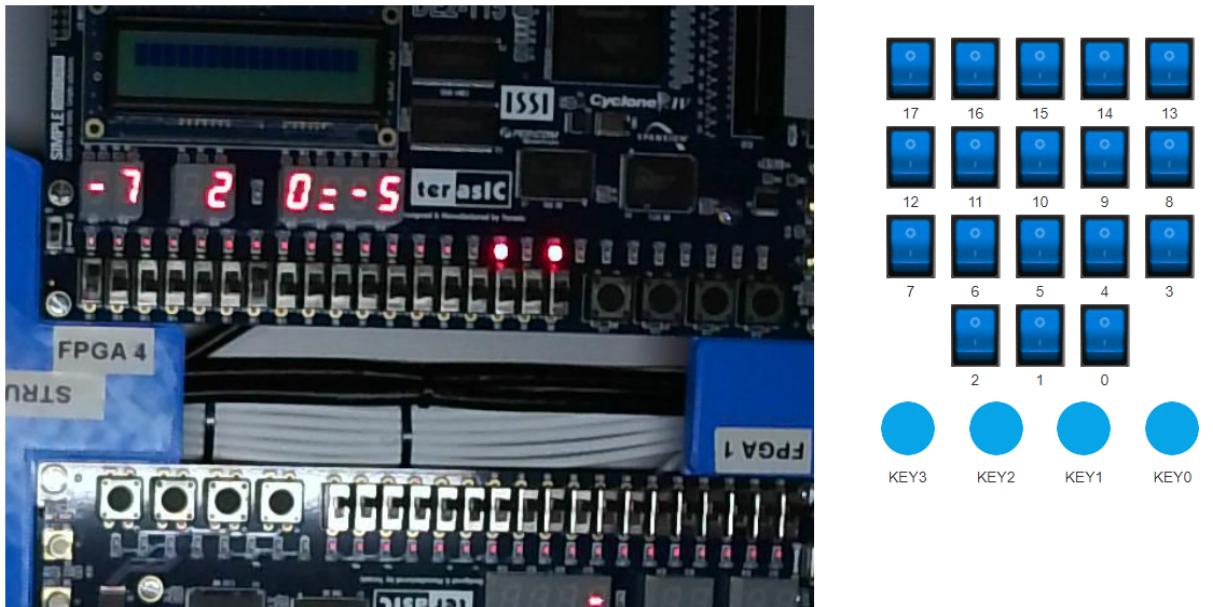


Figura 28: Soma em C2 entre A e B

#### 4.2 A - B (em C2)

Abaixo, como o display *G\_HEX3* exibe 1, a operação de subtração ocorre:  $-8 - (-7) = -8 + 7 = -1$ , obtendo **NEGATIVE=1**. O que está correto.

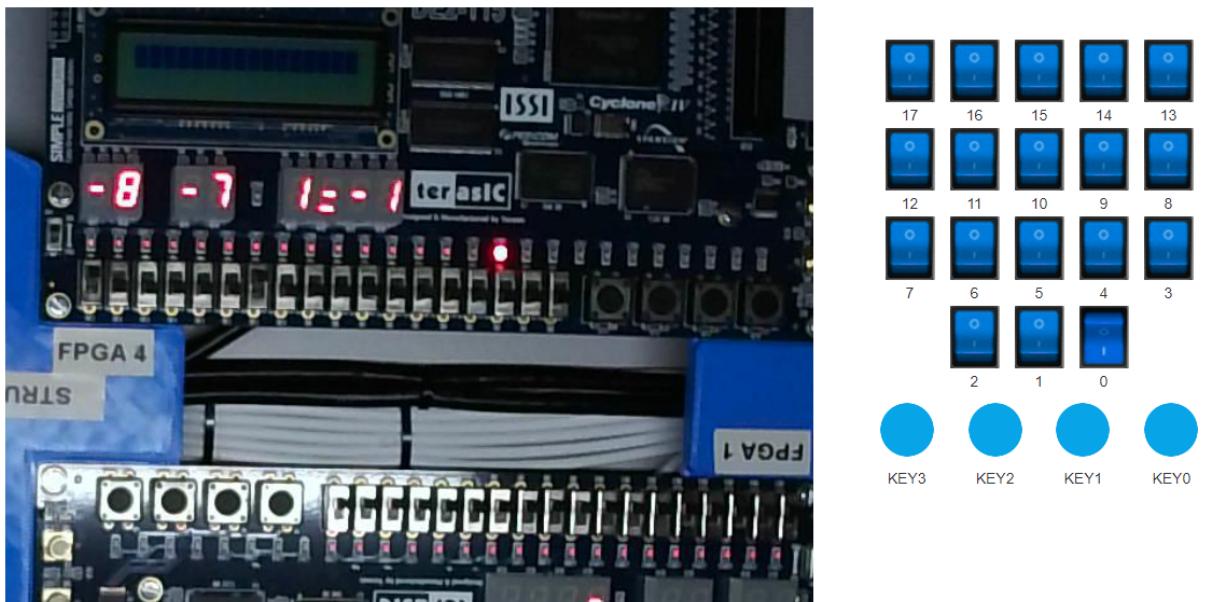


Figura 29: Subtração em C2 entre A e B

### 4.3 Not A

Abaixo, como o display *G\_HEX3* exibe 2, a operação de NOT A ocorre:  $-7 = \overline{1001} = 0110 = +6$ . O que está correto.

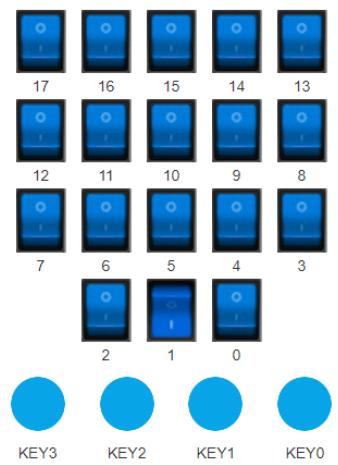
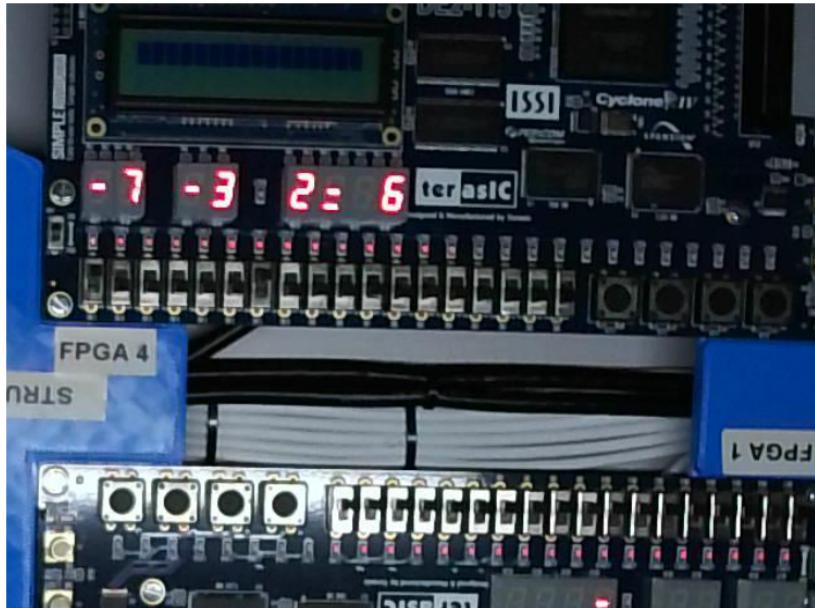


Figura 30: NOT A

### 4.4 Not B

Abaixo, como o display *G\_HEX3* exibe 3, a operação de NOT B ocorre:  $+7 = \overline{0111} = 1000 = -8$ . O que está correto.

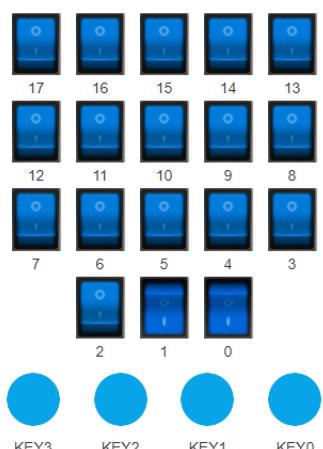


Figura 31: NOT B

## 4.5 A and B

Abaixo, como o display *G\_HEX3* exibe 4, a operação de AND ocorre:  $-5 = 1011$  e  $+6 = 0110$ , onde  $1011 \text{ and } 0110 = 0010 = +2$ . O que está correto.

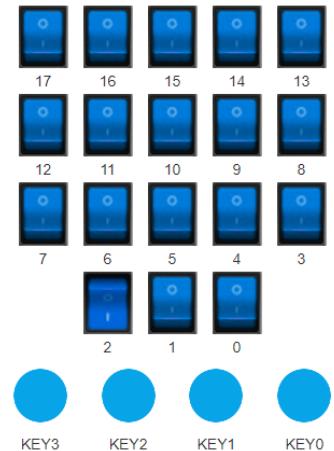
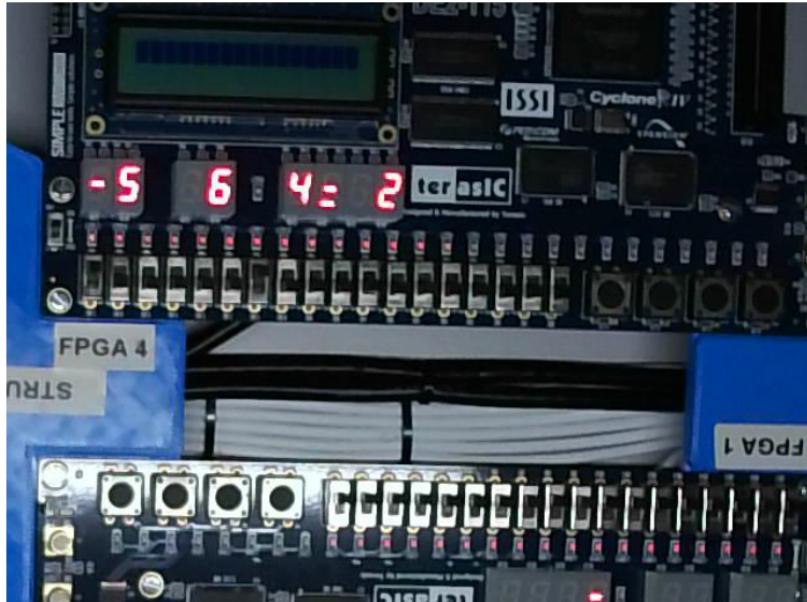


Figura 32: AND entre A e B

## 4.6 A or B

Abaixo, como o display *G\_HEX3* exibe 5, a operação de OR ocorre:  $-4 = 1100$  e  $+2 = 0010$ , onde  $1100 \text{ or } 0010 = 1110 = -2$ . O que está correto.

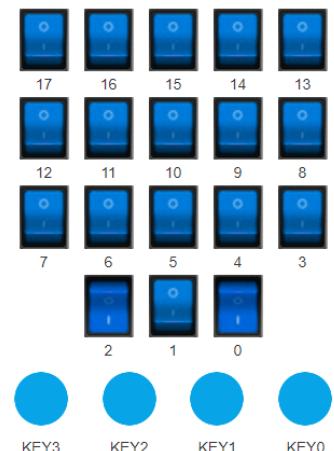
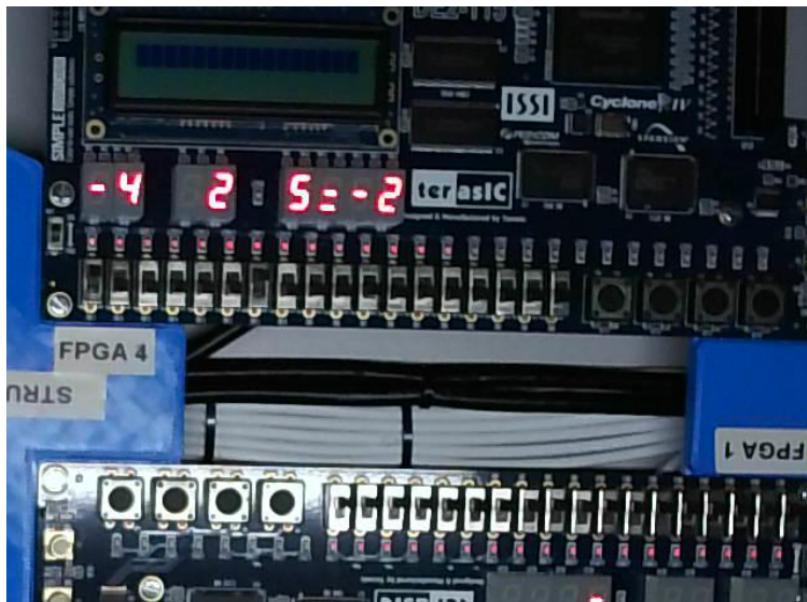


Figura 33: OR entre A e B

## 4.7 A xor B

Abaixo, como o display *G\_HEX3* exibe 6, a operação de XOR ocorre:  $-3 = 1101$  e  $-2 = 1110$ , onde  $1101 \text{ xor } 1110 = 0011 = +3$ . O que está correto.

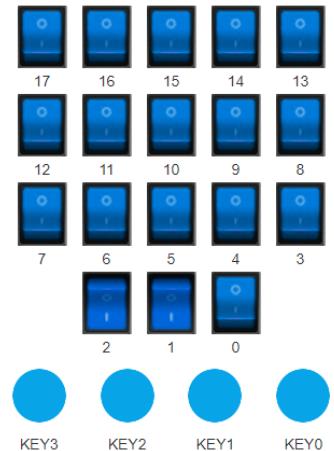
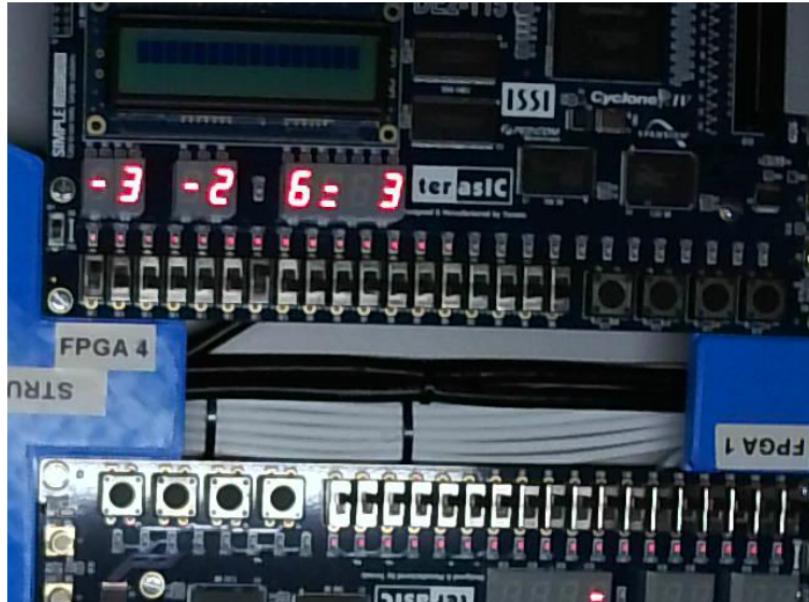


Figura 34: XOR entre A e B

## 4.8 A xnor B

Abaixo, como o display *G\_HEX3* exibe 7, a operação de XNOR ocorre:  $-8 = 1000$  e  $+3 = 0011$ , onde  $1000 \text{ xnor } 0011 = 0100 = +4$ . O que está correto.

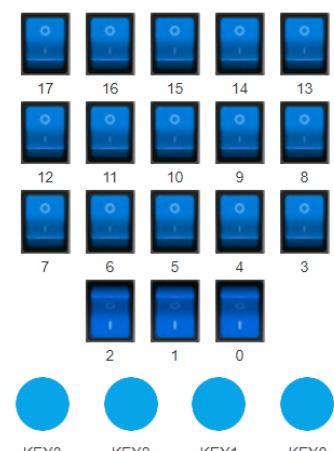
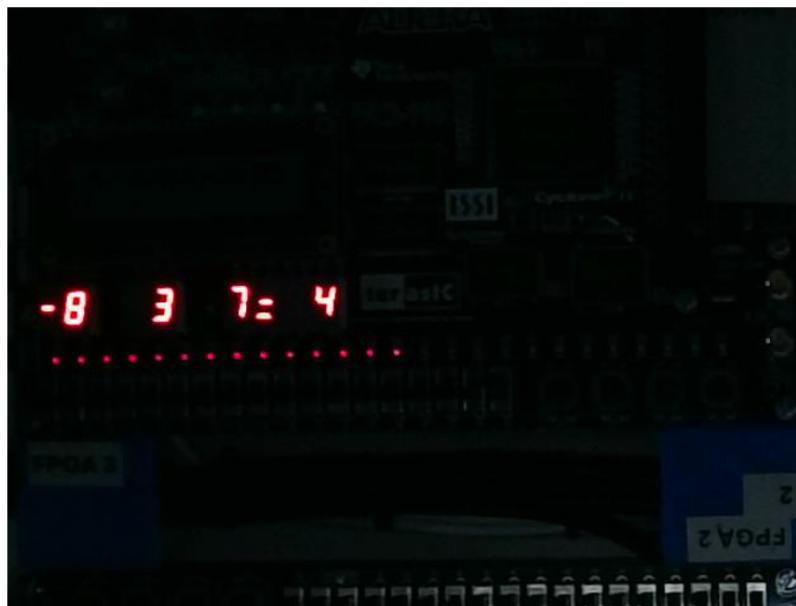


Figura 35: XNOR entre A e B

## 5 Referências

- Jose Paulo Braffman (jul. de 2021). *2021-1 - Sistemas Digitais - EEL480*. URL: <https://www.moodle.poli.ufrj.br/course/view.php?id=198> (acesso em 14/06/2021).
- LabsLand (2021). *Laboratório de FPGA's online*. URL: <https://altera-de2-115-vhdl.ide.labsland.com/> (acesso em 17/06/2021).
- Luís Henrique Maciel (jul. de 2021). *EEL480 - Lab. de Sistemas Digitais*. URL: <https://classroom.google.com/u/0/c/MzY4MDYzMzQ5MDc4> (acesso em 14/06/2021).
- Wagner Rambo (2021). *Playlist de FPGA utilizando o Quartus no Youtube*. URL: [https://www.youtube.com/playlist?list=PLZ8dBTv2\\_5HS79fVexGTtCMDUp7kjnumS](https://www.youtube.com/playlist?list=PLZ8dBTv2_5HS79fVexGTtCMDUp7kjnumS) (acesso em 14/06/2021).