UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

EEL 480 – Laboratório de Sistemas Digitais
1º Trabalho – 2020/2

Frederico Januario Lisbôa	120059054
Mayara Azevedo Aragão	115060969

Unidade Lógico-Aritmética

Sumário

1. Introdução

Este relatório tem como objetivo explicar as decisões tomadas na construção de uma ULA, como proposto para o primeiro trabalho prático da disciplina de Sistemas Digitais.

2. Objetivos

Este trabalho tem como objetivo desenvolver uma Unidade-Lógico-Aritmética (ULA) que opere sobre dois números de 4 bits, com implementação de 8 operações, incluindo, operações lógicas e aritméticas.

Além disso, o sistema possui interface para o teste da ALU, realizado através da placa de desenvolvimento virtual Altera DE-2-115, no Labsland.

O sistema deverá realizar a carga dos dois operandos X e Y, simultaneamente por meio das chaves da placa, que a cada 1 segundo deve alterar a operação corrente e mostrar o valor dos operandos e o resultado da operação correspondente no display de 7 segmentos da placa.

3. Módulos escolhidos

Foi decidido implementar, além das operações lógicas definidas como obrigatórias (*AND*, *OR* e *NOT*), a operação lógica *XOR,* e para as operações aritméticas, a soma, subtração, multiplicação e complemento de 2, considerando os 4 bits de entrada de cada operando.

3.1. Seletor de Operação

Para selecionar qual operação deve ser executada pela Unidade Lógico Aritmética, foi implementado no arquivo *myula.vhd* um 'switch' que seleciona a operação escolhida de acordo com o valor de um seletor.

Como solicitado, esse seletor deve mudar de valor a cada segundo, para que uma nova operação seja executada com os valores dos operandos. Para que fosse implementado um clock de um segundo, fizemos um módulo chamado *myclock*:

3.1.1. Clock de 1 segundo

Utilizando um clock de 50MHz no Labsland, é necessário fazer uma transformação para que o período do clock seja igual a 1 segundo. Para isso, criamos um sinal *clock_out* de saída, que inicialmente tem valor '0' e que após 49999999 iterações, recebe o valor negado dele mesmo(*NOT clock_out*).

Dessa forma, o período do sinal de saída *clock_out* será de 1 segundo como solicitado.

3.1.2. Contador de 3 bits

Agora, para implementar um seletor que altere de valor a cada segundo, e que possa adquirir os valores inteiros de 0 a 7 (para cobrir as 8 operações solicitadas no projeto), implementamos um módulo para um contador de 3 bits.

O contador de 3 bits, recebe como entrada o *clock_out* de saída do módulo de clock de 1 segundo descrito acima, e adiciona uma unidade a uma saída de 3 bits a cada subida de clock.

A saída do contador, posteriormente, será importada como entrada no seletor da *ULA*, e será exibida no display de 7 segmentos *HEX4* da placa Altera DE-2-115, para visualizar a operação escolhida facilmente .

3.2. Unidade Lógica - *AND*

Quando o seletor armazena o valor '*000*' a operação feita é *X AND Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi implementada utilizando um loop de quatro iterações, fazendo a operação *AND* bit a bit.

3.3. Unidade Lógica - *OR*

Quando o seletor armazena o valor '*001*' a operação feita é *X OR Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi implementada utilizando um loop de quatro iterações, fazendo a operação *OR* bit a bit.

3.4. Unidade Lógica - *NOT*

Quando o seletor armazena o valor '*010*' a operação feita é NOT *X*.

Como o operando X possui 4 bits, a operação em VHDL foi implementada utilizando um loop de quatro iterações, fazendo a operação *NOT* bit a bit.

3.5. Unidade Lógica - *XOR*

Quando o seletor armazena o valor '*011*' a operação feita é *X XOR Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi implementada utilizando um loop de quatro iterações, fazendo a operação *XOR* bit a bit.

3.6. Unidade Aritmética - *SOMADOR*

Quando o seletor armazena o valor '*100*' a operação feita é *X + Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi pensada para ser realizada de forma encapsulada, onde a mesma iteração feita para soma de 1 bit, seja sequenciada para soma de 4.

Para a construção dessa unidade, foi necessário criar uma variável de suporte C, de 4 bits, para armazenar o valor do carry nas operações sequenciais.

Utilizando a lógica e materiais fornecidos, sobre a aplicação e desenvolvimento do *full-adder* e utilização do *carry* na parte teórica da matéria, bem como de materiais referenciados na bibliografia.

3.7. Unidade Aritmética - *SUBTRATOR*

Quando o seletor armazena o valor '*101*' a operação feita é *X - Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi pensada para ser realizada de forma encapsulada, onde a mesma iteração feita para subtração de 1 bit, seja sequenciada para subtração de 4 bits.

Se o resultado da subtração for negativo, ou seja, o operando *X* for menor que o operando *Y*, será feito o COMPLEMENTO DE 2 do resultado. Nesse caso, o *CarryOut* vai receber '1' indicando uma *flag* de sinal negativo e a saída da ULA, vai retornar o valor correspondente em positivo para ser passado para o display.

Foi feito utilizando a lógica e materiais fornecidos, sobre a aplicação e desenvolvimento do *full-adder* junto ao conceito de *borrow* na parte teórica da matéria, bem como de materiais referenciados na bibliografia.

3.8. Unidade Aritmética - *MULTIPLICADOR*

Quando o seletor armazena o valor '*110*' a operação feita é *X * Y*.

Como os operandos possuem 4 bits cada, a operação em VHDL foi pensada para ser realizada de forma encapsulada, utilizando o conceito do SOMADOR, além de deslocar o vetor *X* para a direita utilizando vetores auxiliares B, e realizar operação *AND* do vetor *X* com o bit do operando Y multiplicado na iteração da vez .

Caso o bit de *Y* a ser multiplicado (que possui ordem *n*) seja '0', será feito um *AND* de cada bit de *X* com '0', resultando em *'0000'* para a soma da vez. Por outro lado, caso o bit de Y seja '1', será somado o vetor *X* ao resultado, contudo, deslocado de *n* posições, a considerar o peso correspondente do bit de *Y* multiplicado da vez.

Utilizando a lógica e materiais fornecidos, sobre a aplicação e desenvolvimento do *full-adder* junto ao conceito de *carry* na parte teórica da matéria, bem como de materiais referenciados na bibliografia.

3.9. Unidade Aritmética - *COMPLEMENTO DE 2*

Quando o seletor armazena o valor '*111'* a operação feita é *C2(X), ou seja,* o complemento de 2 de *X*.

Como o operando X possui 4 bits, a operação em VHDL foi pensada de modo a analisar se o valor de X é positivo ou negativo. Caso seja positivo (bit mais significativo de X tem valor *'0'*), o valor é decodificado para um Display de 7 segmentos.

No caso do valor ser negativo (bit mais significativo de X tem valor *'1'*), é feito um *NOT* para todos os bits de *X*. Feito isso, somamos o *NOT X* a *'0001'* utilizando o módulo do *SOMADOR,* e o resultado é decodificado em um display de 7 segmentos.

O *CarryOut* sempre recebe o valor do bit mais significativo de *X*, ou seja, X(3), pois representa o sinal do resultado.

A lógica foi baseada nos materiais fornecidos na parte teórica da matéria, bem como de materiais referenciados na bibliografia.

3.10. Decodificador de 7 segmentos

Para que o resultado seja traduzido para um display de 7 segmentos como solicitado, foi implementado um módulo de decodificador que faz a representação de um vetor de 4 bits em um display de 7 segmentos com representação numérica em hexadecimal.

Como nossa saída da *ULA* possui 8 bits, fizemos a decodificação dos 4 bits mais significativos da saída em um display *HEX3*, e a decodificação dos 4 bits menos significativos da saída em um segundo display *HEX2*.

Além disso, para facilitar a visualização das chaves de entrada *X* e *Y* no Labsland, fizemos a decodificação dos operandos nos displays *HEX0* para o vetor de bits *X*, e *HEX1* para o vetor de bits *Y,* além da decodificação do seletor no display *HEX4*, como citado anteriormente.

4. Implementação

4.1. Quartus

4.1.1. Módulo *mysomador.vhd*

```
1  library ieee;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY mysomador IS
5
6     GENERIC (N: INTEGER := 4); -- Numero de bits
7
8     PORT (Cin : IN STD_LOGIC;
9           X, Y : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
10          S : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
11          Cout : OUT STD_LOGIC ) ;
12 END mysomador;
13
14
15 ARCHITECTURE teste OF mysomador IS
16    --SIGNAL C : STD_LOGIC_VECTOR(4 DOWNTO 0) ;
17 BEGIN
18
19    PROCESS(X, Y, Cin)
20       VARIABLE C : STD_LOGIC_VECTOR(N DOWNTO 0) ;
21    BEGIN
22       C(0) := Cin;
23       for i in 0 to N-1 loop
24
25          S(i) <= X(i) XOR Y(i) XOR C(i) ;
26          C(i+1) := (X(i) AND Y(i)) OR (X(i) AND C(i)) OR (Y(i) AND C(i)) ;
27       end loop;
28
29       Cout <= C(N);
30    END PROCESS;
31 END teste;
```

4.1.2. Módulo *mymulty.vhd*

```
library ieee;
use ieee.std_logic_1164.all;

entity mymulty is

   GENERIC (N: INTEGER := 4); -- Numero de bits
   port (
        X: in  std_logic_vector (N-1 downto 0);
        Y: in  std_logic_vector (N-1 downto 0);
        S: out std_logic_vector ((2*N)-1 downto 0) );

end entity mymulty;

architecture teste of mymulty is
    component mysomador
        port (
            X, Y:   in  std_logic_vector (3 downto 0);
            Cin:    in  std_logic;
            S:      out std_logic_vector (3 downto 0);
           Cout:    out std_logic );
    end component;

-- Termos Gi para AND de X com y(i+1):
    signal G0, G1, G2:  std_logic_vector (3 downto 0);

-- Termos Bj -> serão sinais recebidos pelo modulo somador e
-- deslocados para proxima soma(B0 tem os bits de AND com y0)
    signal B0, B1, B2:  std_logic_vector (3 downto 0);

begin

    G0 <= (X(3) and Y(1), X(2) and Y(1), X(1) and Y(1), X(0) and Y(1));
    G1 <= (X(3) and Y(2), X(2) and Y(2), X(1) and Y(2), X(0) and Y(2));
    G2 <= (X(3) and Y(3), X(2) and Y(3), X(1) and Y(3), X(0) and Y(3));
    B0 <=  ('0',          X(3) and Y(0), X(2) and Y(0), X(1) and Y(0));

soma_1:
    mysomador port map ( X => G0, Y => B0, Cin => '0', Cout => B1(3),
                         S(3) => B1(2), S(2) => B1(1), S(1) => B1(0), S(0) => S(1) );

soma_2:
    mysomador port map ( X => G1, Y => B1, Cin => '0', Cout => B2(3),
                         S(3) => B2(2), S(2) => B2(1), S(1) => B2(0), S(0) => S(2) );

soma_3:
    mysomador port map ( X => G2, Y => B2, Cin => '0', Cout => S(7),
                         S => S(6 downto 3) );  -- ultima soma deslocada para a esquerda em 3 casas

    S(0) <= X(0) and Y(0);
end architecture teste;
```

4.1.3. Módulo *myula.vhd*

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY myula IS

   GENERIC (N: INTEGER := 4); -- Numero de bits

   PORT (Selector : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
         X, Y : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
         S : OUT STD_LOGIC_VECTOR((2*N)-1 DOWNTO 0) ;
         Cout : OUT STD_LOGIC ) ;

END myula;
```

```
ARCHITECTURE teste OF myula IS

    signal Saida_Multy: STD_LOGIC_VECTOR((2*N)-1 DOWNTO 0) ;

    component mymulty
              port(
                 X: in  std_logic_vector (N-1 downto 0);
                 Y: in  std_logic_vector (N-1 downto 0);
                 S: out std_logic_vector ((2*N)-1 downto 0));
    end component;
```

```
26      component mysomador
27          port (
28              X, Y:       in  std_logic_vector (3 downto 0);
29              Cin:    in  std_logic;
30              S:      out std_logic_vector (3 downto 0);
31             Cout:    out std_logic
32          );
33       end component;
34
35
36      SIGNAL S_Aux: STD_LOGIC_VECTOR(N-1 DOWNTO 0) ; --para complemento de 2 na subtracao
37
38  BEGIN
39
40      multiplicacao: mymulty port map ( X => X, Y => Y, S => Saida_Multy );
41
42      my_case : process (Selector, X, Y) is
43
44      --Variaveis auxiliares para a ULA:
45         VARIABLE C : STD_LOGIC_VECTOR(N DOWNTO 0) ;
46         VARIABLE Comp2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
47         VARIABLE NOTX : STD_LOGIC_VECTOR(3 DOWNTO 0);
48         VARIABLE NOTY : STD_LOGIC_VECTOR(3 DOWNTO 0);
49         VARIABLE NOTS : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
50
51      begin
52
53         CASE Selector IS
54  ------------------------------------------------------------------------------
55          -- AND
56             when "000" =>
57
58                for i in 0 to N-1 loop
59                   S(i) <= X(i) AND Y(i);
60                end loop;
61
62                Cout <= '0';
63
64                for j in N to (2*N)-1 loop
65                   S(j) <= '0'; --completando os bits de saida mais significativos com 0
66                end loop;
67
68
69  ------------------------------------------------------------------------------
70             -- OR
71             when "001" =>
72
73                for i in 0 to N-1 loop
74                   S(i) <= X(i) OR Y(i);
75                end loop;
76
77                Cout <= '0';
78
79                for j in N to (2*N)-1 loop
80                   S(j) <= '0'; --completando os bits de saida mais significativos com 0
81                end loop;
82
83  ------------------------------------------------------------------------------
84              -- NOT
85              when "010" =>
86
87                 for i in 0 to N-1 loop
88
89                    S(i) <= NOT X(i) ;
90
91                 end loop;
92
93                 Cout <= '0';
94
95                 for j in N to (2*N)-1 loop
96                    S(j) <= '0'; --completando os bits de saida mais significativos com 0
97                 end loop;
98
99
```

```
100 ----------------------------------------------------------------------------------
101         -- XOR
102         when "011" =>
103
104            for i in 0 to N-1 loop
105               S(i) <= X(i) XOR Y(i) ;
106            end loop;
107
108            Cout <= '0';
109
110            for j in N to (2*N)-1 loop
111               S(j) <= '0'; --completando os bits de saida mais significativos com 0
112            end loop;
113
114
115 ----------------------------------------------------------------------------------
116         -- SOMADOR
117         when "100" =>
118
119               -- Setando o CarryIn em 0 para a soma do fulladder
120
121               C(0) := '0';
122
123               for i in 0 to N-1 loop
124                  S(i) <= X(i) XOR Y(i) XOR C(i) ;
125                  C(i+1) := (X(i) AND Y(i)) OR (X(i) AND C(i)) OR (Y(i) AND C(i)) ;
126               end loop;
127
128               Cout <= '0'; -- Para sinal sempre positivo!
129               S(N) <= C(N); --atribuindo o Carry para o S(4), pois saida da ULA tem 8 bits
130
131               for j in N+1 to (2*N)-1 loop
132                  S(j) <= '0'; --completando os bits de saida mais significativos com 0
133               end loop;
134
135 ----------------------------------------------------------------------------------
136         -- SUBTRATOR
137         when "101" =>
138
139               C(0) := '0';
140
141               for i in 0 to N-1 loop
142                  NOTX(i) := NOT X(i);
143
144                  S_Aux(i) <= X(i) XOR Y(i) XOR C(i) ;
145                  C(i+1) := (NOTX(i) AND Y(i)) OR (NOTX(i) AND C(i)) OR (Y(i) AND C(i)) ;
146               end loop;
147
148               -- Aplicando agora complemento de 2 no resultado,
149               -- caso negativo, para tratar numa flag de sinal
150
151               Cout <= C(N); --flag de sinal
152
153               if(C(N) = '1') then
154
155                  Comp2(0) := '1';
156                  Comp2(1) := '0';
157                  Comp2(2) := '0';
158                  Comp2(3) := '0';
159
160                  for i in 0 to N-1 loop
161                     NOTS(i) := NOT S_Aux(i);
162
163                     S(i) <= NOTS(i) XOR Comp2(i) XOR C(i) ;
164                     C(i+1) := (NOTS(i) AND Comp2(i)) OR (NOTS(i) AND C(i)) OR (Comp2(i) AND C(i)) ;
165                  end loop;
166
167               else
168                  for i in 0 to N-1 loop
169                     S(i) <= S_Aux(i);
170                  end loop;
171               end if;
172
173               for j in N to (2*N)-1 loop
174                  S(j) <= '0'; --completando os bits de saida mais significativos com 0
175               end loop;
176
178 ----------------------------------------------------------------------------------
179          -- MULTIPLICAÇÃO
180
181          when "110" =>
182
183             for i in 0 to (2*N)-1 loop
184                S(i) <= Saida_Multy(i); -- saida recebe resultado do componente mymulty
185             end loop;
186
187             Cout <= '0'; --sinal sempre positivos
188
189
```

```
190 ----------------------------------------------------------------------------
191         -- COMPLEMENTO DE 2
192         when others =>
193
194         -- Setando o CarryIn em 0 para a soma do fulladder
195
196               Cout <= X(3); -- Sinal do numero passado em Complemento de 2
197
198               if(X(3) = '1') then
199
200                  Comp2(0) := '1';
201                  Comp2(1) := '0';
202                  Comp2(2) := '0';
203                  Comp2(3) := '0';
204
205                  C(0) := '0';
206
207                  for i in 0 to N-1 loop
208                     NOTX(i) := NOT X(i);
209
210                     S(i) <= NOTX(i) XOR Comp2(i) XOR C(i) ;
211                     C(i+1) := (NOTX(i) AND Comp2(i)) OR (NOTX(i) AND C(i)) OR (Comp2(i) AND C(i)) ;
212                  end loop;
213
214                  --Cout <= C(N);
215               else
216                  for i in 0 to N-1 loop
217                     S(i) <= X(i);
218                  end loop;
219
220               end if;
221
222               for j in N to (2*N)-1 loop
223                  S(j) <= '0';
224               end loop;
225
226      END CASE;
227   end process my_case;
228
229 END teste;
```

4.1.4. Módulo *myclock.vhd*

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY myclock is
5  PORT (
6           clk_in, reset : in std_logic_vector(0 downto 0);
7           clk_out: out std_logic_vector(0 downto 0)
8       );
9  END myclock;
10
```

```
11 ARCHITECTURE Teste of myclock is
12    signal novo_clk: std_logic_vector(0 downto 0);
13    signal contador : integer range 0 to 49999999 := 0;
14
15 BEGIN
16
17    myfrequency: process (reset, clk_in)
18    BEGIN
19
20       if (reset = "1") then
21          novo_clk <= "0";
22          contador <= 0;
23       elsif rising_edge(clk_in(0)) then --Subida do clock
24
25          if (contador = 49999999) then --(50000000) - 1
26             novo_clk <= NOT(novo_clk);
27             contador <= 0;
28          else
29             contador <= contador + 1;
30          end if;
31       end if;
32
33    END process;
34
35    clk_out <= novo_clk;
36
37 END Teste;
```

4.1.5. Módulo *mycontador3bits.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY mycontador3bits IS
PORT (   clk: in std_logic; -- clock input
         reset: in std_logic; -- reset input
         contador: out STD_LOGIC_VECTOR(2 downto 0) -- output 3bits
      );

END mycontador3bits;

ARCHITECTURE Teste OF mycontador3bits IS
   SIGNAL contador_up: std_logic_vector(2 downto 0);

BEGIN

   PROCESS(clk)
   BEGIN

      if(rising_edge(clk)) then
         if(reset='1') then
            contador_up <= "000";
         else
            if(contador_up = "000") then contador_up <= "001";
            elsif (contador_up = "001") then contador_up <= "010";
            elsif (contador_up = "010") then contador_up <= "011";
            elsif (contador_up = "011") then contador_up <= "100";
            elsif (contador_up = "100") then contador_up <= "101";
            elsif (contador_up = "101") then contador_up <= "110";
            elsif (contador_up = "110") then contador_up <= "111";
            elsif (contador_up = "111") then contador_up <= "000";
            end if;
         end if;
      end if;

   END PROCESS;
   contador <= contador_up;
END Teste;
```

4.1.6. Módulo *mydecodificador7seg.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

ENTITY mydecodificador7seg is
   GENERIC (N: INTEGER := 4); -- Numero de bits

   PORT (
         Sinal : in std_logic_vector(N-1 downto 0);
         Display: out std_logic_vector(6 downto 0)
         );

END mydecodificador7seg;
```

```
ARCHITECTURE Teste of mydecodificador7seg is

BEGIN
    --anodo

    process(Sinal)
    begin
       case Sinal is
       when x"0" => Display<="1000000";
       when x"1" => Display<="1111001";
       when x"2" => Display<="0100100";
       when x"3" => Display<="0110000";
       when x"4" => Display<="0011001";
       when x"5" => Display<="0010010";
       when x"6" => Display<="0000010";
       when x"7" => Display<="1111000";
       when x"8" => Display<="0000000";
       when x"9" => Display<="0010000";
       when x"a" => Display<="0001000"; -- hex(a)
       when x"b" => Display<="0000011"; -- hex(b)
       when x"c" => Display<="1000110"; -- hex(c)
       when x"d" => Display<="0100001"; -- hex(d)
       when x"e" => Display<="0000110"; -- hex(e)
       when x"f" => Display<="0001110"; -- hex(f)

       when others=> Display<="1111111";

       end case;
    end process;


END Teste;
```

4.1.7. Top Level Entity *myproject.vhd*

Para implementar a Top Level Entity e testar no quartus, o programa não suporta o clock de 50MHzs, somente no Labsland para teste utilizando a placa e componentes. Portanto, aqui no quartus, para fins de teste, utilizamos como clock o próprio sinal de entrada std_logic *CLOCK_50.*

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY myproject IS

   GENERIC (N: INTEGER := 4); -- Numero de bits

   PORT (
        CLOCK_50: in std_logic; --Recebe o clock da FPGA
        V_SW: in std_logic_vector(8 downto 0); --Reset_Seletor, X3,X2,X1,X0,Y3,Y2,Y1,Y0
        HEX0, HEX1: out std_logic_vector (6 downto 0); --em ordem, X,Y.
        HEX2, HEX3: out std_logic_vector (6 downto 0); --Em ordem, o resultado da ULA em 2 Displays,
        LEDG: out std_logic --flags em ordem: ov, cout, zero, sinal
        );
END myproject;
```

```
ARCHITECTURE estrutura OF myproject IS

   component myula
         port(
               Selector : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
               X : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
               Y : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
               S : OUT STD_LOGIC_VECTOR((2*N)-1 DOWNTO 0) ;
               Cout : OUT STD_LOGIC
            );
   end component;
```

```
29    component myclock
30       port(
31             clk_in, reset : in std_logic;
32             clk_out: out std_logic
33          );
34    end component myclock;
35
36    component mycontador3bits
37       port (
38          clk: in std_logic; -- clock input
39          reset: in std_logic; -- reset input
40          contador: out std_logic_vector(2 downto 0) -- output 3-bit contador para seletor
41       );
42    end component mycontador3bits;
```

```
44    component mydecodificador7seg --Precisaremos de 4 desses, um para cada display.
45       port (
46          Sinal : in std_logic_vector(3 downto 0);
47          Display : out std_logic_vector(6 downto 0)
48       );
49    end component mydecodificador7seg;
```

```
50
51    --Declação de sinais:
52
53    SIGNAL clk1seg: std_logic;
54    SIGNAL contadorSeletor: std_logic_vector(2 downto 0);
55    SIGNAL saidaUla: std_logic_vector(7 downto 0);
56    SIGNAL CarryOutUla: std_logic;
57
58  BEGIN
59
60    clock50MHz: myclock port map(CLOCK_50,'0',clk1seg); --clk_in, reset, clk_out.
61
62    --contador3bits: mycontador3bits port map(clk1seg, V_SW(8), contadorSeletor); -> No labsland
63    contador3bits: mycontador3bits port map(CLOCK_50, V_SW(8), contadorSeletor);
64
65    ula: myula port map (
66       contadorSeletor,
67       V_SW(7 downto 4),
68       V_SW(3 downto 0),
69       saidaUla,
70       CarryOutUla
71       );
72
73
74    dec0: mydecodificador7seg port map(V_SW(7 downto 4),HEX0); --V_SW, HEX0.
75    dec1: mydecodificador7seg port map(V_SW(3 downto 0),HEX1);
76
77    dec2: mydecodificador7seg port map(saidaUla(3 downto 0),HEX2);
78    dec3: mydecodificador7seg port map(saidaUla(7 downto 4),HEX3);
79
80    LEDG <= CarryOutUla;
81
82  END estrutura;
```

4.2. Labsland

No Labsland, será necessário passar um clock de *50MHz* e utilizar a saída do clock de 1 segundo do componente myclock, para a entrada do contador, para que seja feito um incremento de uma unidade a cada segundo, selecionando então uma nova operação na *ULA*.

Logo, o mapeamento da entrada no contador será diferente no Quartus do que para teste no Labsland. Além disso, importamos o sinal do contador de 3 bits para um display de 7 segmentos, para mostrar numericamente qual operação está sendo feita.

4.2.1. Top Level Entity *myprojectLABSLAND.vhd* Labsland:

```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;

ENTITY myprojectLABSLAND IS

    GENERIC (N: INTEGER := 4); -- Numero de bits

    PORT (
          CLOCK_50: in std_logic; --Recebe o clock da FPGA
          V_SW: in std_logic_vector(8 downto 0); --Reset_Seletor, X3,X2,X1,X0,Y3,Y2,Y1,Y0
          HEX0, HEX1: out std_logic_vector (6 downto 0); --Representacao de X,Y em displays.
          HEX2, HEX3: out std_logic_vector (6 downto 0); --Resultado da ULA EM 2 Displays
          HEX4: out std_logic_vector (6 downto 0);-- Display pro Seletor.
          LEDG: out std_logic_vector(3 downto 0)  --LEDG(3) flag de sinal
          );

END myprojectLABSLAND;
```

```
ARCHITECTURE estrutura OF myprojectLABSLAND IS

    component myula
          port(
                Selector : IN STD_LOGIC_VECTOR(2 DOWNTO 0) ;
                X : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Y : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;

                S : OUT STD_LOGIC_VECTOR((2*N)-1 DOWNTO 0) ;
                Cout : OUT STD_LOGIC
             );
    end component;

    component myclock
       port(
             clk_in, reset : in std_logic;
             clk_out: out std_logic
          );
    end component myclock;
```

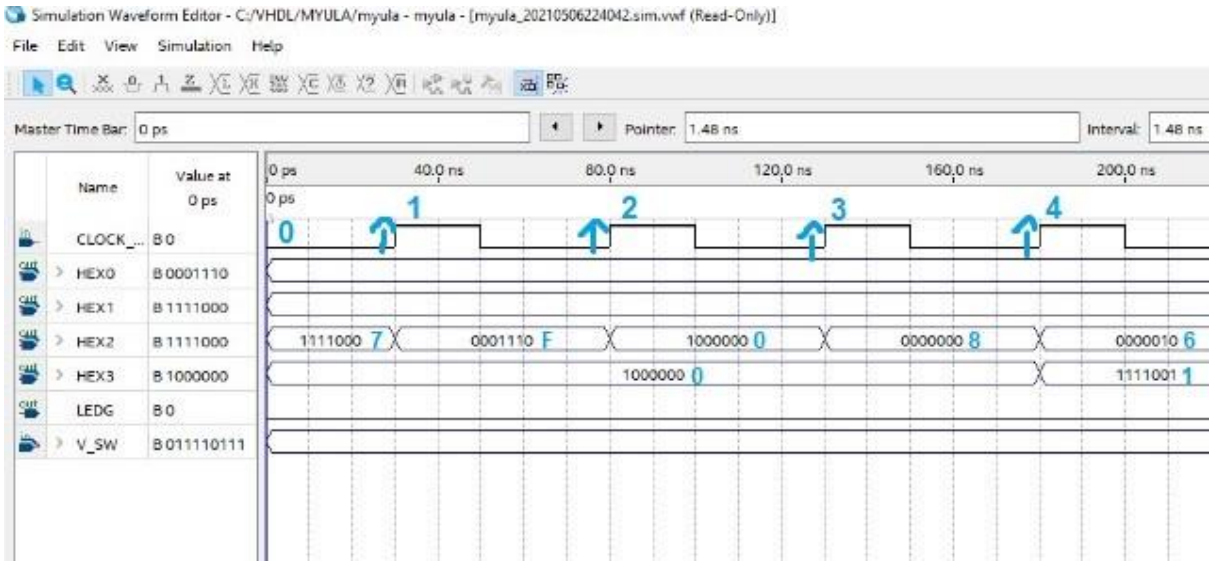
```
    component mycontador3bits
       port (
          clk: in std_logic; -- clock input
          reset: in std_logic; -- reset input
          contador: out std_logic_vector(2 downto 0) -- output 3-bit contador para seletor
       );
    end component mycontador3bits;

    component mydecodificador7seg --Precisaremos de 4 desses, um para cada display.
       port (
          Sinal : in std_logic_vector(3 downto 0);
          Display : out std_logic_vector(6 downto 0)
       );
    end component mydecodificador7seg;
```

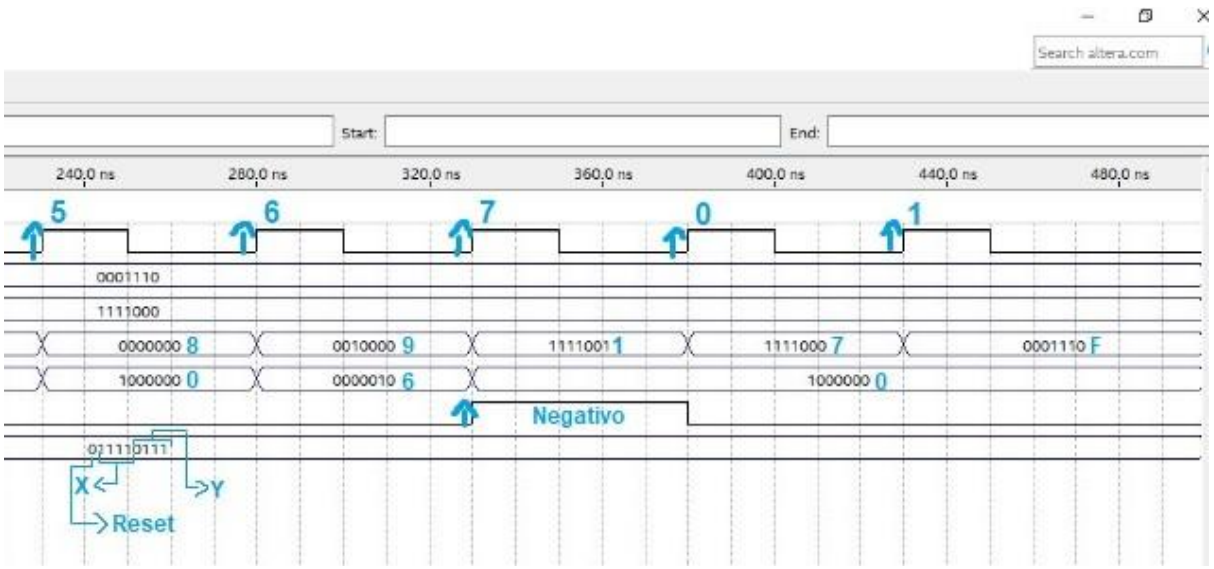
```
    --Declação de sinais:

    SIGNAL clk1seg: std_logic;
    SIGNAL contadorSeletor: std_logic_vector(2 downto 0);
    SIGNAL contadorDisplay: std_logic_vector(3 downto 0);
    SIGNAL saidaUla: std_logic_vector(7 downto 0);
    --SIGNAL CarryOutUla: std_logic;

BEGIN


    clock50MHz: myclock port map(CLOCK_50,'0',clk1seg); --clk_in, reset, clk_out.

    contador3bits: mycontador3bits port map(clk1seg, V_SW(8), contadorSeletor);

    ula: myula port map (
       contadorSeletor(2 downto 0),
       V_SW(7 downto 4),
       V_SW(3 downto 0),
       saidaUla,
       LEDG(3)
       );

    contadorDisplay(3) <= '0';
    contadorDisplay(2 downto 0) <= contadorSeletor(2 downto 0);

    dec0: mydecodificador7seg port map(V_SW(7 downto 4),HEX0); --V_SW, HEX0.
    dec1: mydecodificador7seg port map(V_SW(3 downto 0),HEX1);

    dec2: mydecodificador7seg port map(saidaUla(3 downto 0),HEX2);
    dec3: mydecodificador7seg port map(saidaUla(7 downto 4),HEX3);

    dec4: mydecodificador7seg port map( contadorDisplay,HEX4);


END estrutura;
```

5. Testes

5.1. Quartus

O programa foi verificado e compilado com sucesso, nas imagens abaixo estão identificadas as subidas de clock, que ocasionam na alteração da operação corrente(incremento do seletor), os valores de entrada em cada decodificador para os displays 7 segmentos (HEX2 e HEX3) com seus respectivos valores em Hexadecimal, o LEDG, que representa o sinal do resultado (1 = negativo e 0 = positivo), e as entradas da nossa ULA juntamente com o bit de Reset.





5.2. Labsland

O programa foi verificado e compilado com sucesso, na imagem abaixo estão identificadas as chaves de entrada e os displays de saída do nosso projeto. (Na imagem abaixo o led de sinal está apagado, representando que o resultado é positivo)

6. Conclusão

O projeto implementado da Unidade Lógica Aritmética funcionou como deveria nos testes realizados tanto no Labsland como no Quartus, sendo necessário apenas algumas mudanças na Top Level Entity para testar nas diferentes plataformas utilizadas.

7. Referências Bibliográficas

- https://www.gta.ufrj.br/ensino/EEL480/Introducao-VHDL.pdf
- https://ic.unicamp.br/~cortes/mc602/slides/VHDL/VHDL_3_MC_Circ_Arit_v1.pdf
- https://www.ics.uci.edu/~jmoorkan/vhdlref/ifs.html
- https://electronics.stackexchange.com/questions/273467/how-do-i-link-two-components-from-different-files-in-vhdl
- https://stackoverflow.com/questions/26288447/using-entities-from-another-file-in-vhdl
- https://stackoverflow.com/questions/53329810/vhdl-4-bit-multiplier-based-on-4-bit-adder
- https://www.fpga4student.com/2017/06/vhdl-code-for-counters-with-testbench.html
- https://www.allaboutcircuits.com/technical-articles/std-logic-vector-data-type-in-vhdl-code/
- https://technobyte.org/vhdl-code-synchronous-upcounter-behavioral/
- https://buzztech.in/vhdl-modelling-styles-behavioral-dataflow-structural/