UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

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

Frederico Januario Lisbôa 120059054
Mayara Azevedo Aragão 115060969

Calculadora
BCD

Sumário

1. Introdução

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

2. Objetivos

Este trabalho tem como objetivo desenvolver uma calculadora que realiza a soma ou multiplicação de 2 números naturais arbitrários A e B de até 4 algarismos, exclusivamente por operações algarismo-a-algarismo do tipo BCD.

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 A e B, identificados pelas chaves da placa, ao apertar um botão, os valores de A e B devem aparecer nos displays de 7 segmentos da placa para melhor visualização da operação executada.

3. Módulos escolhidos

Como solicitado, os módulos de soma e multiplicação foram implementados em BCD. Para solucionar o problema dado, implementamos alguns módulos para serem importados dentro do módulo principal da nossa calculadora. A seguir, os módulos e suas respectivas descrições.

3.1. Soma Algarismo

Este módulo, possui como entrada 3 sinais, todos com 4 bits ao todo. A implementação consiste em fazer a soma de dois algarismos em BCD, ou seja, considerando que caso a operação resulte em um valor maior que 9, deve ser armazenada em algarismo de unidade, e algarismo da dezena.

O resultado final, faz uma comparação entre o resultado da soma dos sinais de A, B e CarryIn, caso menor que 10, a saída permanece a mesma (representa a unidade) e o carryOut representará o 0 (deneza).

Para o caso onde o resultado da soma é maior que 10 e menor que 20 (para a implementação da soma não resultará em valores maiores que 20, mas para a multiplicação pode resultar), o carryOut recebe 1 e a saída tem seu valor subtraído de 10.

No caso onde o resultado da soma é maior que 20 e menor que 27, (utilizado na multiplicação, como explicaremos a seguir), o CarryOut recebe 2 e a saída é subtraída de 20.

Lembrando que esses valores são dados em STD_LOGIC_VECTOR(3 downto 0), e utilizamos esta técnica para manter no padrão BCD.

3.2. Soma

Para implementar a soma da calculadora (este módulo será importado no arquivo Top-Level-Entity), temos como entrada os valores de A e B fornecidos pela chaves da placa e representados em bcd. Este módulo importa o módulo anterior, de **Soma Algarismo**, e o executa quatro vezes.

Para cada algarismo em BCD dado de A e B, o CarryOut a da soma dos algarismos da unidade de A e B, entram como CarryIn na soma dos algarismos da dezena de A e B, e assim sucessivamente. Até que por fim, o resultado é dado pela saída S, e o algarismo mais significativo representado pelo CarryOut.

3.3. Multiplica Algarismo

Para implementar uma soma entre dois algarismos em BCD, fizemos este módulo que tem como entrada um algarismo de A e um algarismo de B para a multiplicação, que é feita utilizando a multiplicação da biblioteca

IEEE.NUMERIC_STD, e após a multiplicação dos algarismos dados, fazemos uma divisão por 10, para retornar a dezena do resultado em D, e o resto da divisão por 10, retornamos a unidade U, para ser utilizado pelo módulo a seguir.

3.4. Multiplica Linha

Neste módulo, importamos os módulos já implementados de **Multiplicação de algarismo** e **soma de algarismo**. Neste módulo, fizemos a multiplicação de um número A com 4 algarismos BCD, por um algarismo de B.

Foi decidido implementar desse modo, levando em consideração a forma que fazemos a multiplicação manualmente, ou seja, multiplicamos o primeiro número por cada algarismo do segundo número.

No primeiro passo, fazemos a multiplicação, de cada algarismo de A, com o algarismo de B (entradas). Após realizar cada multiplicação, implementamos a soma dos algarismos dados.

Considerando que a unidade do resultado da multiplicação do algarismo menos significativo de A com o algarismo de B, será a unidade da saída. Já a unidade da segunda multiplicação (dezena de A com algarismo de B), será a soma dos algarismos da dezena da primeira multiplicação com a unidade da segunda multiplicação e o CarryOut da primeira multiplicação entra como CarryIn na segunda multiplicação. Este processo é repetido sucessivamente, e a saída resultante contém 5 algarismos BCD, ou seja, é representada por um STD_LOGIC_VECTOR(19 downto 0).

3.5. Soma 32 bits

Esta soma foi implementada, espelhada na **soma** (3.2) de 4 algarismos, contudo, para entrada, fornecemos um número em BCD com 8 algarismos.

Foi projetada para realizarmos as somas dos resultados do módulo **Multiplicação Linha** deslocados de n algarismos em BCD para esquerda, sendo n o índice do algarismo de B em BCD. Ou seja, para o resultado da multiplicação de A pela unidade de B, apenas acrescentamos 3 zeros à esquerda. Para a multiplicação de A pela dezena de B, acrescentamos no resultado da multiplicação da linha, 2 zeros a esquerda e 1 zero a direita, e assim sucessivamente.

3.6. Multiplicação

Este módulo importa os módulos **Multiplicação Linha** e **Soma 32 bits**. Primeiramente, são feitas quatro operações de Multiplicação Linha, de A por

cada algarismo BCD dado em B. O resultado é convertido em 8 algarismos BCD (32 bits) de acordo com o índice de B, como descrito no módulo anterior.

O segundo passo é fazer a soma dois a dois dos resultados convertidos, que após 3 somas deve retornar o resultado esperado. Para as somas dois a dois realizadas nesta etapa, o CarryIn é sempre nulo.

3.7. Calculadora

Para a Top-Level-Entity, importamos os módulos principais do nosso projeto, que são a **Soma** e a **Multiplicação**. Para selecionar qual operação deve ser executada pela nossa calculadora, o sinal de entrada OP, define a operação escolhida. Caso OP = 0, operação é soma, caso contrário (OP =1) operação é de multiplicação entre os valores dados de A e B.

Aplicamos os componentes importados em sinais auxiliares, portanto, caso a operação seja de soma, o resultado é concatenado com 3 zeros à esquerda, com o carryOut da soma, e com a saída da soma, totalizando 8 algarismos de saída em BCD (32bits). Caso a operação seja de multiplicação, a saída da calculadora é diretamente a saída da multiplicação, já com 8 algarismos em BCD.

3.8. Decodificador de 7 segmentos

Para que o resultado seja traduzido para um display de 7 segmentos como solicitado no Labsland, 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 BCD.

Como nossa saída da **calculadora** possui 32 bits, fizemos a decodificação dos bits de saída em 8 displays e a representação dos números de entrada A e B em 4 displays cada quando o botão *KEY(2)* não está pressionado e quando selecionando, o resultado da operação aparece nos displays.

4. Implementação

4.1. Quartus

4.1.1. Módulo *somaAlgarismo.vhd*

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

entity somaAlgarismo is
    Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cout : out STD_LOGIC_VECTOR (3 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(3 DOWNTO 0));
end SomaAlgarismo;

architecture estrutura of SomaAlgarismo is

    -- criando sinais auxiliares para fazer o tratamento dos dados para bcd!
    signal S_bcd : std_logic_vector(4 DOWNTO 0);
    signal S_aux : std_logic_vector(4 DOWNTO 0);

    signal A_aux : std_logic_vector(4 DOWNTO 0);
    signal B_aux : std_logic_vector(4 DOWNTO 0);
    signal cin_aux : std_logic_vector(4 DOWNTO 0);
    signal cout_aux : std_logic_vector(4 DOWNTO 0);

    begin

      --concatenando variaveis

      A_aux <= '0' & A;
      B_aux <= '0' & B;
      cin_aux <= '0' & cin;
      S_aux <= A_aux + B_aux + cin_aux; -- fazendo a soma, mas em formato binario

      --utilizando biblioteca unsigned

      AjusteBCD: process(S_aux)
         begin
            if S_aux < 10 then -- se soma<10, nao é necessario fazer ajuste

               S_bcd <= S_aux;
               cout_aux <= "00000";

            elsif S_aux < 20 then -- se 10<soma<20, adicionar 1 ao cout e diminuir 10 da soma!

               S_bcd <= S_aux - "01010";
               cout_aux <= "00001";

            else -- se 20<soma<27, adicionar 2 ao cout e diminuir 20 da soma! ( para o caso de A, B e Cin MAX)
               S_bcd <= S_aux - "10100";
               cout_aux <= "00010";

            end if;
         end process;


      S <= S_bcd(3 DOWNTO 0);
      cout <= cout_aux(3 DOWNTO 0);

end estrutura;
```

4.1.2. Módulo *soma.vhd*

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

entity Soma is
    Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(15 DOWNTO 0);
          cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cout : out STD_LOGIC_VECTOR(3 DOWNTO 0));
end Soma;

architecture estrutura of Soma is

    component somaAlgarismo is
    Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cout : out STD_LOGIC_VECTOR (3 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    signal cout0: std_logic_vector(3 DOWNTO 0);
    signal cout1: std_logic_vector(3 DOWNTO 0);
    signal cout2: std_logic_vector(3 DOWNTO 0);

    begin
       Soma0: SomaAlgarismo Port Map(A => A(3 downto 0),
                                     B => B(3 downto 0),
                                     cin => cin,
                                     cout => cout0,
                                     S => S(3 downto 0));

       Soma1: SomaAlgarismo Port Map(A => A(7 downto 4),
                                     B => B(7 downto 4),
                                     cin => cout0,
                                     cout => cout1,
                                     S => S(7 downto 4));

       Soma2: SomaAlgarismo Port Map(A => A(11 downto 8),
                                     B => B(11 downto 8),
                                     cin => cout1,
                                     cout => cout2,
                                     S => S(11 downto 8));

       Soma3: SomaAlgarismo Port Map(A => A(15 downto 12),
                                     B => B(15 downto 12),
                                     cin => cout2,
                                     cout => cout,
                                     S => S(15 downto 12));


end estrutura;
```

4.1.3. Módulo *multiplicaAlgarismo.vhd*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity multiplicaAlgarismo is
    Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          D : out STD_LOGIC_VECTOR(3 DOWNTO 0); -- Dezena
          U : out STD_LOGIC_VECTOR(3 DOWNTO 0)); --Unidade
end multiplicaAlgarismo;

architecture estrutura of multiplicaAlgarismo is

    signal mult : unsigned (7 downto 0) := "00000000";

    -- separar em dezenas e unidades
    signal dez : unsigned (7 downto 0);
    signal uni : unsigned (7 downto 0);

    signal s_d : STD_LOGIC_VECTOR (7 downto 0);
    signal s_u : STD_LOGIC_VECTOR(7 downto 0);

    begin
       mult <= unsigned(A) * unsigned(B);

       dez <= mult / 10;
       uni <= mult mod 10; --mod

       s_d <= STD_LOGIC_VECTOR(dez);
       s_u <= STD_LOGIC_VECTOR(uni);

       --convertendo o resultado para bcd:

       D <= s_d(3 DOWNTO 0);
       U <= s_u(3 DOWNTO 0);

end estrutura;
```

4.1.4. Módulo *multiplicaLinha.vhd*

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.all;

entity multiplicaLinha is
    Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR( 3 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(19 DOWNTO 0));
end multiplicaLinha;

architecture estrutura of multiplicaLinha is

    component multiplicaAlgarismo is
       Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
             B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
             D : out STD_LOGIC_VECTOR(3 DOWNTO 0);
             U : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    component somaAlgarismo is
       Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
             B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
             cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
             cout : out STD_LOGIC_VECTOR(3 DOWNTO 0);
             S : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    --Sinais para armazenar cada resultado da multiplicacao por Algarismo
    signal D0 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal D1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal D2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal D3 : STD_LOGIC_VECTOR(3 DOWNTO 0);

    signal U0 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal U1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal U2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal U3 : STD_LOGIC_VECTOR(3 DOWNTO 0);

    --Sinais Para armazenar os carryout auxiliares da soma da unidade com a dezena do resultado anterior
    signal C0 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal C1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal C2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal C3 : STD_LOGIC_VECTOR(3 DOWNTO 0);

    --Sinais para armazenar os algarismos em bcd da saida

    signal S0 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal S1 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal S2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal S3 : STD_LOGIC_VECTOR(3 DOWNTO 0);
    signal S4 : STD_LOGIC_VECTOR(3 DOWNTO 0);

    begin

    --primeiramente fazer as multiplicacoes individuais de cada algarimo de A pelo algarismo de B.

    Multiplicao0: multiplicaAlgarismo Port Map ( A => A(3 DOWNTO 0),
                                                 B => B,
                                                 D => D0,
                                                 U => U0);

    Multiplicao1: multiplicaAlgarismo Port Map ( A => A(7 DOWNTO 4),
                                                 B => B,
                                                 D => D1,
                                                 U => U1);

    Multiplicao2: multiplicaAlgarismo Port Map ( A => A(11 DOWNTO 8),
                                                 B => B,
                                                 D => D2,
                                                 U => U2);

    Multiplicao3: multiplicaAlgarismo Port Map ( A => A(15 DOWNTO 12),
                                                 B => B,
                                                 D => D3,
                                                 U => U3);


    --fazer a soma da unidade de Mx encontrado com a dezena de Mx-1

    S0 <= U0;

    Soma0: somaAlgarismo Port Map(A => D0,
                                  B => U1,
                                  cin => "0000",
                                  cout => C0,
                                  S => S1);

    Soma1: somaAlgarismo Port Map(A => D1,
                                  B => U2,
                                  cin => C0,
                                  cout => C1,
                                  S => S2);

    Soma2: somaAlgarismo Port Map(A => D2,
                                  B => U3,
                                  cin => C1,
                                  cout => C2,
                                  S => S3);
```

```
101
102     Soma3: somaAlgarismo Port Map(A => D3,
103                                   B => "0000",
104                                   cin => C2,
105                                   cout => C3,
106                                   S => S4);
107     S(3 DOWNTO 0) <= S0;
108     S(7 DOWNTO 4) <= S1;
109     S(11 DOWNTO 8) <= S2;
110     S(15 DOWNTO 12) <= S3;
111     S(19 DOWNTO 16) <= S4;
112
113 end estrutura;
114
```

4.1.5. Módulo *soma32.vhd*

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity Soma32 is
 5     Port (A : in STD_LOGIC_VECTOR(31 DOWNTO 0);
 6           B : in STD_LOGIC_VECTOR(31 DOWNTO 0);
 7           S : out STD_LOGIC_VECTOR(31 DOWNTO 0);
 8           cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
 9           cout : out STD_LOGIC_VECTOR(3 DOWNTO 0));
10  end Soma32;
11
12  architecture estrutura of Soma32 is
13
14     component somaAlgarismo is
15     Port (A : in STD_LOGIC_VECTOR(3 DOWNTO 0);
16           B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
17           cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
18           cout : out STD_LOGIC_VECTOR (3 DOWNTO 0);
19           S : out STD_LOGIC_VECTOR(3 DOWNTO 0));
20     end component;
21
22     signal cout0: std_logic_vector(3 DOWNTO 0);
23     signal cout1: std_logic_vector(3 DOWNTO 0);
24     signal cout2: std_logic_vector(3 DOWNTO 0);
25     signal cout3: std_logic_vector(3 DOWNTO 0);
26     signal cout4: std_logic_vector(3 DOWNTO 0);
27     signal cout5: std_logic_vector(3 DOWNTO 0);
28     signal cout6: std_logic_vector(3 DOWNTO 0);
29
30     begin
31        Soma0: SomaAlgarismo Port Map(A => A(3 downto 0),
32                                      B => B(3 downto 0),
33                                      cin => cin,
34                                      cout => cout0,
35                                      S => S(3 downto 0));
36
37        Soma1: SomaAlgarismo Port Map(A => A(7 downto 4),
38                                      B => B(7 downto 4),
39                                      cin => cout0,
40                                      cout => cout1,
41                                      S => S(7 downto 4));
42
43        Soma2: SomaAlgarismo Port Map(A => A(11 downto 8),
44                                      B => B(11 downto 8),
45                                      cin => cout1,
46                                      cout => cout2,
47                                      S => S(11 downto 8));
48
49        Soma3: SomaAlgarismo Port Map(A => A(15 downto 12),
50                                      B => B(15 downto 12),
51                                      cin => cout2,
52                                      cout => cout3,
53                                      S => S(15 downto 12));
```

```
55        Soma4: SomaAlgarismo Port Map(A => A(19 downto 16),
56                                      B => B(19 downto 16),
57                                      cin => cout3,
58                                      cout => cout4,
59                                      S => S(19 downto 16));
60
61        Soma5: SomaAlgarismo Port Map(A => A(23 downto 20),
62                                      B => B(23 downto 20),
63                                      cin => cout4,
64                                      cout => cout5,
65                                      S => S(23 downto 20));
66
67        Soma6: SomaAlgarismo Port Map(A => A(27 downto 24),
68                                      B => B(27 downto 24),
69                                      cin => cout5,
70                                      cout => cout6,
71                                      S => S(27 downto 24));
72
73        Soma7: SomaAlgarismo Port Map(A => A(31 downto 28),
74                                      B => B(31 downto 28),
75                                      cin => cout6,
76                                      cout => cout,
77                                      S => S(31 downto 28));
78
79  end estrutura;
80
```

4.1.6. Módulo multiplicacao.*vhd*

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Multiplicacao is
5       Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
6             B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
7             S : out STD_LOGIC_VECTOR(31 DOWNTO 0));
8   end Multiplicacao;
9
10  architecture estrutura of Multiplicacao is
11
12      component multiplicaLinha is
13      Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
14            B : in STD_LOGIC_VECTOR(3 DOWNTO 0);
15            S : out STD_LOGIC_VECTOR(19 DOWNTO 0));
16      end component;
17
18      component Soma32 is
19      Port (A : in STD_LOGIC_VECTOR(31 DOWNTO 0);
20            B : in STD_LOGIC_VECTOR(31 DOWNTO 0);
21            S : out STD_LOGIC_VECTOR(31 DOWNTO 0);
22            cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
23            cout : out STD_LOGIC_VECTOR (3 DOWNTO 0));
24      end component;
25
26      signal M0 : STD_LOGIC_VECTOR(19 DOWNTO 0);
27      signal M1 : STD_LOGIC_VECTOR(19 DOWNTO 0);
28      signal M2 : STD_LOGIC_VECTOR(19 DOWNTO 0);
29      signal M3 : STD_LOGIC_VECTOR(19 DOWNTO 0);
30
31      signal S0 : STD_LOGIC_VECTOR(31 DOWNTO 0);
32      signal S1 : STD_LOGIC_VECTOR(31 DOWNTO 0);
33      signal S2 : STD_LOGIC_VECTOR(31 DOWNTO 0);
34      signal S3 : STD_LOGIC_VECTOR(31 DOWNTO 0);
35
36      signal SS1 : STD_LOGIC_VECTOR(31 DOWNTO 0); -- Saidas da soma dois a dois
37      signal SS2 : STD_LOGIC_VECTOR(31 DOWNTO 0);
38
39      signal CS1 : STD_LOGIC_VECTOR(3 DOWNTO 0); -- carrys
40      signal CS2 : STD_LOGIC_VECTOR(3 DOWNTO 0);
41      signal CS3 : STD_LOGIC_VECTOR(3 DOWNTO 0);
42
43      begin
44
45      Mult_0 : multiplicaLinha
46      Port Map(A => A,
47               B => B(3 downto 0),
48               S => M0);
49
50      Mult_1 : multiplicaLinha
51      Port Map(A => A,
52               B => B(7 downto 4),
53               S => M1);
54
55      Mult_2 : multiplicaLinha
56      Port Map(A => A,
57               B => B(11 downto 8),
58               S => M2);
59
60      Mult_3 : multiplicaLinha
61      Port Map(A => A,
62               B => B(15 downto 12),
63               S => M3);
64
65      --ajustando as saidas da multiplicacao para 32 bits para fazer a soma:
66
67      S0 <= "000000000000" & M0;
68      S1 <= "00000000" & M1 & "0000";
69      S2 <= "0000" & M2 & "00000000";
70      S3 <= M3 & "000000000000";
71
72      Soma0 : Soma32
73      Port Map(A => S0,
74               B => S1,
75               S => SS1,
76               cin => "0000",
77               cout => CS1);
78
79      Soma1 : Soma32
80      Port Map(A => S2,
81               B => S3,
82               S => SS2,
83               cin => "0000",
84               cout => CS2);
85
86      Soma2 : Soma32
87      Port Map(A => SS1,
88               B => SS2,
89               S => S, --SS1 + SS2
90               cin => "0000",
91               cout => CS3);
92
93
94  end estrutura;
```

4.1.7. Top Level Entity *mycalculadora.vhd*

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

entity mycalculadora is

    Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          OP : in STD_LOGIC; -- 0(soma) , 1(multiplicacao)
          S : out STD_LOGIC_VECTOR(31 DOWNTO 0));

end mycalculadora ;

architecture estrutura of mycalculadora is

    component Soma is
    Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(15 DOWNTO 0);
          cin : in STD_LOGIC_VECTOR(3 DOWNTO 0);
          cout : out STD_LOGIC_VECTOR(3 DOWNTO 0));
    end component;

    component Multiplicacao is
    Port (A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          S : out STD_LOGIC_VECTOR(31 DOWNTO 0));
    end component;

    signal cout0 : STD_LOGIC_VECTOR(3 DOWNTO 0);

    signal S_soma : STD_LOGIC_VECTOR(15 DOWNTO 0);
    signal S_mult : STD_LOGIC_VECTOR(31 DOWNTO 0);

    begin

       Somador: Soma
       Port Map(A => A,
                B => B,
                cin => "0000",
                cout => cout0,
                S => S_soma);

       |
       Multiplicador: Multiplicacao
       Port Map(A => A,
                B => B,
                S => S_mult);


       Operacao: process (S_soma, S_mult, OP)
       begin
          if OP = '0' then
             S <= "000000000000"  & cout0 & S_soma;
          else
             S <= S_mult;
          end if;
       end process;


end estrutura;
```

4.2. Labsland

No Labsland, devemos interpretar o chaveamento da placa para que os números A e B da entrada sejam lidos. Isto porque cada número, vai precisar de 16 chaves para leitura do valor passado(*SW(15 downto 0)*), após um clique no botão *KEY(0)*, os valores em A são carregados e um clique em *KEY(1)* carrega os valores das chaves em B. A operação escolhida foi implementada na chave (SW(17)).

Além disso, será necessário utilizar um decodificador para display BCD para mostrar numericamente os valores de A, B e o resultado da operação. Para isto, utilizamos os 8 displays da placa, sendo que quando o botão virtual *KEY(2)* não estiver pressionado, serão

exibidos os valores de A (HEX7 a HEX4) e B(HEX3 a HEX0), e quando for pressionado, será exibido o resultado da operação correspondente.

4.2.1. Top Level Entity *myproject.vhd*:

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

entity myproject is
    Port (SW: in std_logic_vector(17 downto 0) ;
          KEY: in std_logic_vector(3 downto 0); -- default 1/ press 0
          HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7: out std_logic_vector(6 downto 0));

end myproject ;

architecture estrutura of myproject is

    component mycalculadora is
    Port (  A : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          B : in STD_LOGIC_VECTOR(15 DOWNTO 0);
          OP : in STD_LOGIC; -- 0(soma) , 1(multiplicacao)
          S : out STD_LOGIC_VECTOR(31 DOWNTO 0));
    end component;

    component mydecodificadorBCD is
       PORT (Sinal : in std_logic_vector(3 downto 0);
             Display: out std_logic_vector(6 downto 0));
    end component;

    signal Saida: STD_LOGIC_VECTOR(31 DOWNTO 0) ;
    signal A: STD_LOGIC_VECTOR(15 DOWNTO 0) ;
    signal B: STD_LOGIC_VECTOR(15 DOWNTO 0) ;

    signal A_HEX4: std_logic_vector(6 downto 0);
    signal A_HEX5: std_logic_vector(6 downto 0);
    signal A_HEX6: std_logic_vector(6 downto 0);
    signal A_HEX7: std_logic_vector(6 downto 0);

    signal B_HEX0: std_logic_vector(6 downto 0);
    signal B_HEX1: std_logic_vector(6 downto 0);
    signal B_HEX2: std_logic_vector(6 downto 0);
    signal B_HEX3: std_logic_vector(6 downto 0);

    signal S_HEX0: std_logic_vector(6 downto 0);
    signal S_HEX1: std_logic_vector(6 downto 0);
    signal S_HEX2: std_logic_vector(6 downto 0);
    signal S_HEX3: std_logic_vector(6 downto 0);
    signal S_HEX4: std_logic_vector(6 downto 0);
    signal S_HEX5: std_logic_vector(6 downto 0);
    signal S_HEX6: std_logic_vector(6 downto 0);
    signal S_HEX7: std_logic_vector(6 downto 0);


    begin
```

```
    Chaveamento: process (SW(17 downto 0), KEY(3 downto 0))
    begin
       if KEY(0) = '0' then
          A <= SW(15 downto 0);
        end if;

        if KEY(1) = '0' then
          B <= SW(15 downto 0);
       end if;

    end process;

    Calcular: mycalculadora
    Port Map(   A => A,
             B => B,
             OP => SW(17),
             S =>  Saida);

    dec0: mydecodificadorBCD port map(A(3 downto 0),A_HEX4);
    dec1: mydecodificadorBCD port map(A(7 downto 4),A_HEX5);
    dec2: mydecodificadorBCD port map(A(11 downto 8),A_HEX6);
    dec3: mydecodificadorBCD port map(A(15 downto 12),A_HEX7);

    dec4: mydecodificadorBCD port map(B(3 downto 0),B_HEX0);
    dec5: mydecodificadorBCD port map(B(7 downto 4),B_HEX1);
    dec6: mydecodificadorBCD port map(B(11 downto 8),B_HEX2);
    dec7: mydecodificadorBCD port map(B(15 downto 12),B_HEX3);

```

```
79          dec8: mydecodificadorBCD port map(Saida(3 downto 0),S_HEX0);
80          dec9: mydecodificadorBCD port map(Saida(7 downto 4),S_HEX1);
81          dec10: mydecodificadorBCD port map(Saida(11 downto 8),S_HEX2);
82          dec11: mydecodificadorBCD port map(Saida(15 downto 12),S_HEX3);
83          dec12: mydecodificadorBCD port map(Saida(19 downto 16),S_HEX4);
84          dec13: mydecodificadorBCD port map(Saida(23 downto 20),S_HEX5);
85          dec14: mydecodificadorBCD port map(Saida(27 downto 24),S_HEX6);
86          dec15: mydecodificadorBCD port map(Saida(31 downto 28),S_HEX7);
87
88
89          EscolherDisplay: process (KEY(2))
90          begin
91             if KEY(2) = '1' then
92                HEX0 <= B_HEX0;
93                HEX1 <= B_HEX1;
94                HEX2 <= B_HEX2;
95                HEX3 <= B_HEX3;
96
97                HEX4 <= A_HEX4;
98                HEX5 <= A_HEX5;
99                HEX6 <= A_HEX6;
100               HEX7 <= A_HEX7;
101            end if;
102
103            if KEY(2) = '0' then
104               HEX0 <= S_HEX0;
105               HEX1 <= S_HEX1;
106               HEX2 <= S_HEX2;
107               HEX3 <= S_HEX3;
108               HEX4 <= S_HEX4;
109               HEX5 <= S_HEX5;
110               HEX6 <= S_HEX6;
111               HEX7 <= S_HEX7;
112            end if;
113         end process;
114
115
116   end estrutura;
```

4.2.2. mydecodificadorBCD.vhd

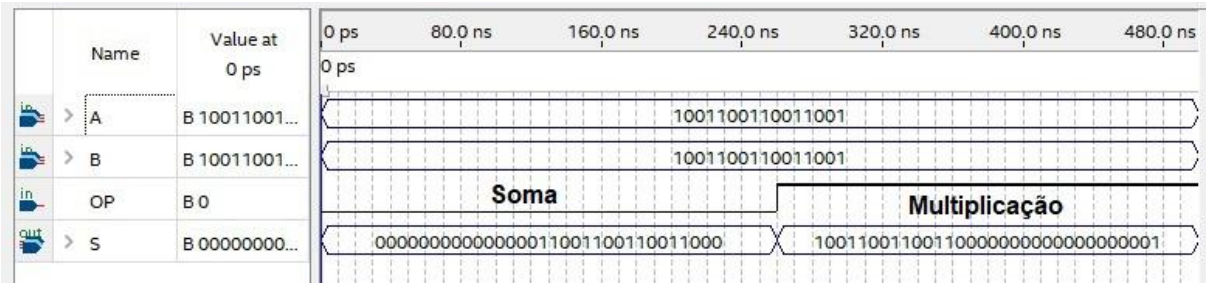
```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.numeric_std.all;
4
5   ENTITY mydecodificadorBCD is
6      GENERIC (N: INTEGER := 4); -- Numero de bits
7
8      PORT (Sinal : in std_logic_vector(N-1 downto 0);
9            Display: out std_logic_vector(6 downto 0));
10
11  END mydecodificadorBCD;
12
13  ARCHITECTURE Teste of mydecodificadorBCD is
14
15  BEGIN
16     --anodo
17
18     process(Sinal)
19     begin
20        case Sinal is
21        when x"0" => Display<="1000000";
22        when x"1" => Display<="1111001";
23        when x"2" => Display<="0100100";
24        when x"3" => Display<="0110000";
25        when x"4" => Display<="0011001";
26        when x"5" => Display<="0010010";
27        when x"6" => Display<="0000010";
28        when x"7" => Display<="1111000";
29        when x"8" => Display<="0000000";
30        when x"9" => Display<="0010000";
31        when x"a" => Display<="1000000";
32        when x"b" => Display<="1000000";
33        when x"c" => Display<="1000000";
34        when x"d" => Display<="1000000";
35        when x"e" => Display<="1000000";
36        when x"f" => Display<="1000000";
37
38        when others=> Display<="1111111";
39
40        end case;
41     end process;
42  END Teste;
```

5. Testes

5.1. Quartus

O programa foi verificado e compilado com sucesso, nas imagens abaixo estão identificadas os sinais de entrada, A e B, além da OP (Operação), utilizado para selecionar a operação corrente.

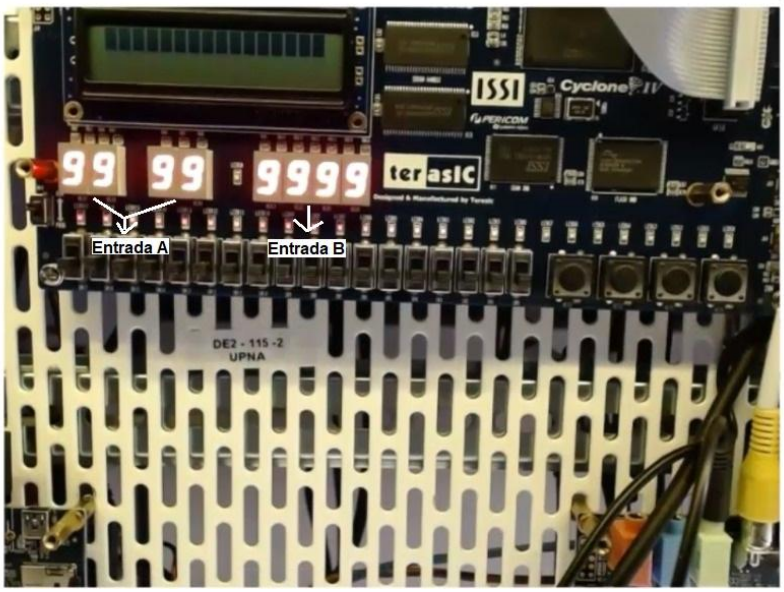
A seguir, as operações foram feitas com os valores de A=9999 e B=9999, para comprovar o funcionamento do projeto mesmo com valores limites:

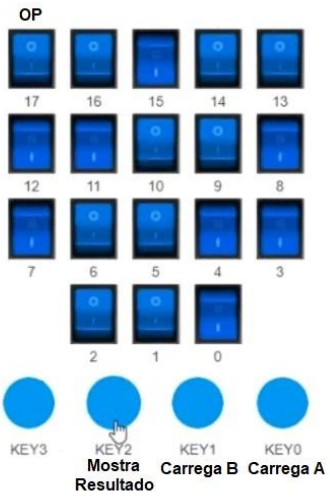


5.2. Labsland

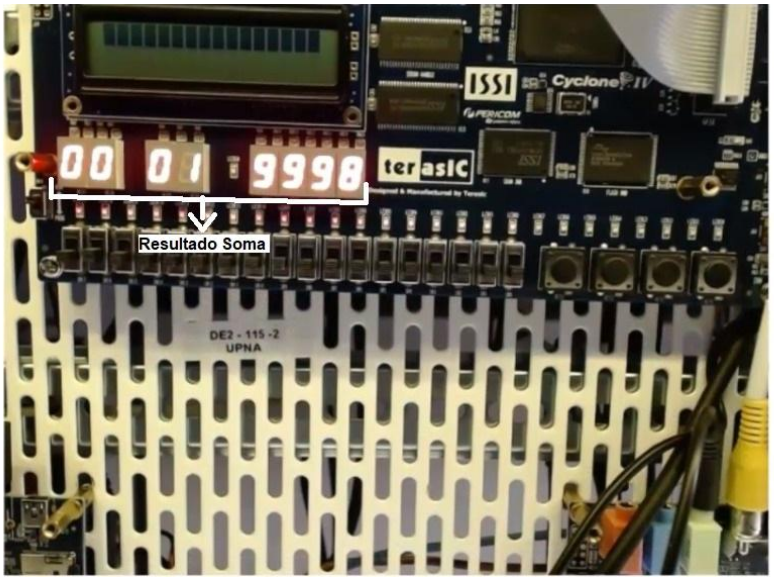
O programa foi verificado e compilado com sucesso, nas 3 imagens abaixo estão identificadas, respectivamente, os valores das entradas A e B, uma operação de soma entre A e B e por fim a operação de multiplicação de A por B, com resultados exibidos no display:

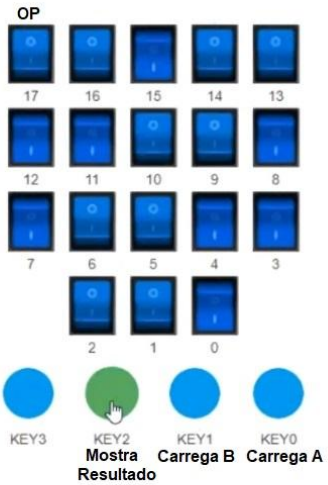
- A e B



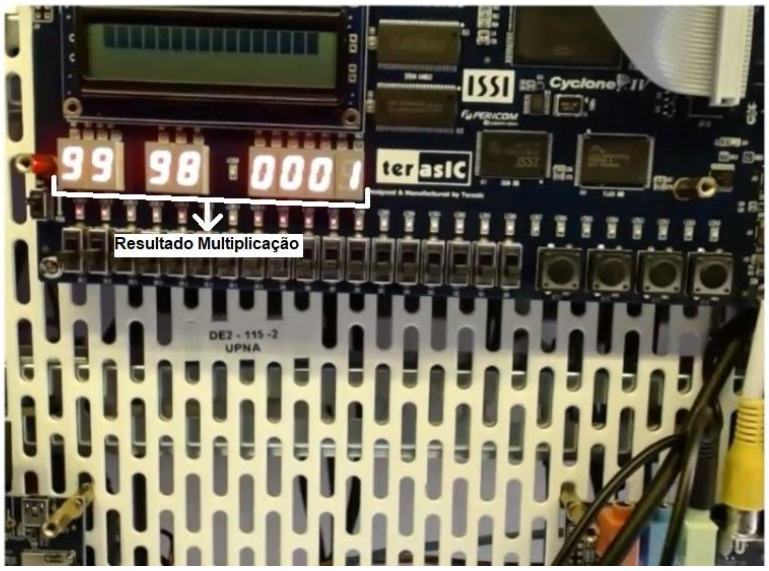


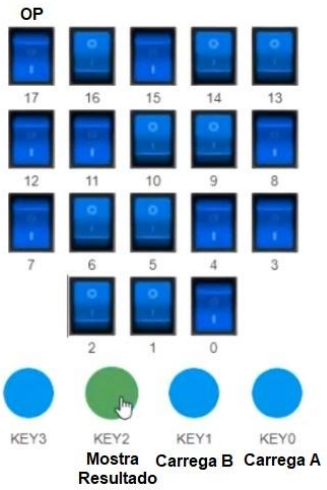
- Soma A + B





- Multiplicação A * B





6. Conclusão

O projeto implementado da Calculadora BCD funcionou como deveria nos testes realizados tanto no Labsland como no Quartus, sendo necessário algumas mudanças na Top Level Entity para testar nas diferentes plataformas utilizadas, além da implementação de um display de 7 segmentos para exibir os resultados na placa do Labsland.

O desenvolvimento deste trabalho foi mais simples comparado ao primeiro trabalho prático da disciplina, considerando que implementamos lógicas semelhantes às que utilizamos no dia a dia para cálculos no sistema decimal, enquanto que para realizar as

mesmas operações utilizando sistema binário, envolve outra lógica para obter resultados semelhantes.

Estarmos familiarizados com o ambiente de desenvolvimento, tanto no Quartus como no Labsland, foi um fator decisivo na nossa confiança em desenvolver o projeto, além disso, utilizar a biblioteca IEEE.NUMERIC_STD auxiliou bastante no desenvolvimento do projeto em BCD.