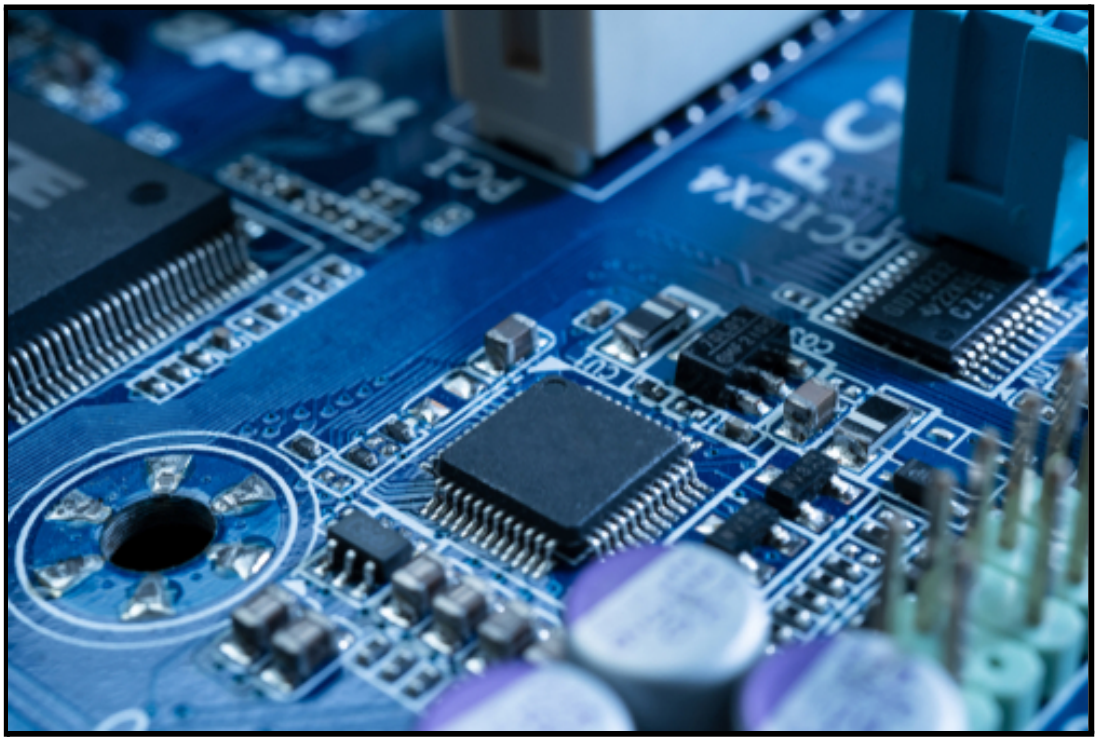
Laboratório de Sistemas Digitais - UFRJ



Trabalho Prático 1 (ULA)

Aluno: Paulo Victor Innocencio / DRE: 116213599

Aluno: Vinicius Cavalcante Silva / DRE: 120013422

Disciplina: Sistemas Digitais EEL480 - 2021/2

Professores: João Baptista de Oliveira e Souza Filho

Introdução

O objetivo deste trabalho é implementar uma Unidade Lógica Aritmética (ULA) utilizando a linguagem de descrição de hardware VHDL, que é responsável por descrever o comportamento e estrutura de um sistema digital.

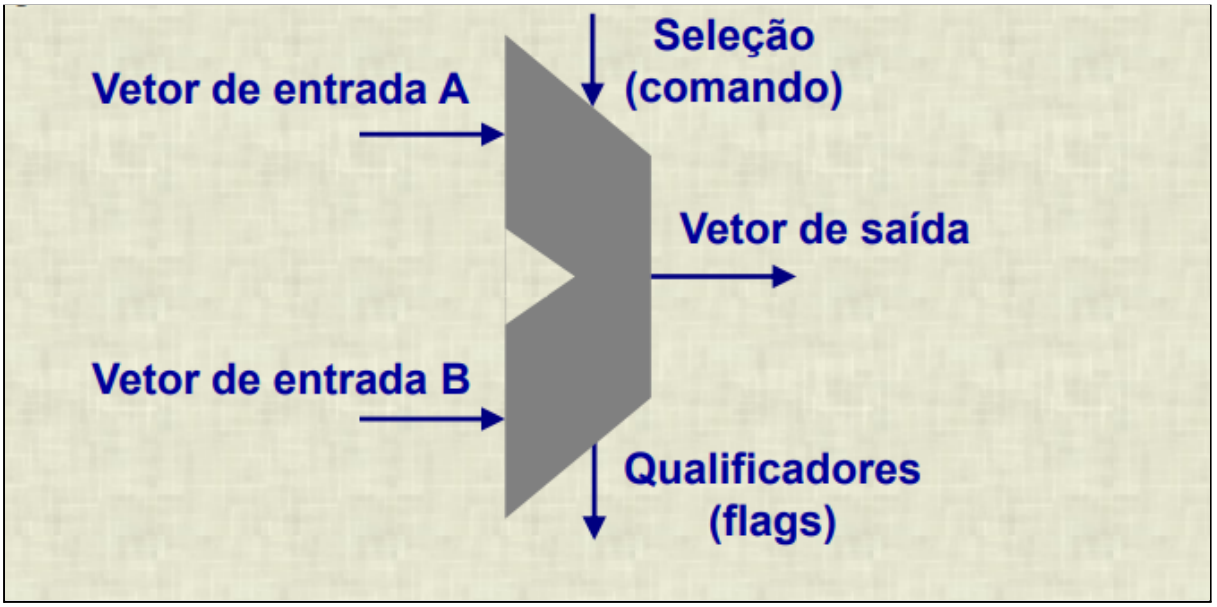


Figura 1: Representação da Unidade Lógico Aritmética (*ULA*)

Para o desenvolvimento da ULA, utilizaremos uma ALTERA DE-2-115 disponível online pela labsland.

Funcionalidades

A Unidade Lógica Aritmética implementada opera com 2 entradas de 4 bits cada uma e realiza um total de 8 operações. A seleção das operações ocorre seguinte forma:

Seleção	Operação
000	X and Y
001	X or Y
010	not X
011	X xor Y
100	X + Y
101	X - Y
110	X * Y
111	complemento de 2 (X)

As operações são divididas em 4 lógicas (and, or, not, xor) e 4 aritméticas (soma, subtração, multiplicação e complemento de 2).

No desenvolvimento de cada componente o único pacote utilizado foi o *std_logic_1164* da biblioteca *IEEE*, para a representação de cada bit. A justificativa para tal é que essa biblioteca fornece modelagem de sinais mais realistas, que se comportam melhor no sistema, diminuindo falhas.

A definição da lógica do componente é feita após a declaração da *entity* (entidade) onde é definido as entradas e as saídas do módulo. Na *architecture* são definidas as relações entre entradas e saídas juntamente com a lógica do módulo. Em alguns componentes existe a definição de *signal* que possue funcionamento análogo ao de um fio e conecta partes diferentes do sistema.

Antes de demonstrar os resultados por meio de diagramas de ondas, iremos explicar o componente descrito. Os códigos fontes de cada componentes estarão anexadas no apêndice deste relatório

Considerando que $X_3X_2X_1X_0$, $Y_3Y_2Y_1Y_0$ e $S_3S_2S_1S_0$ são vetores de 4 bits e que X e Y são de entrada e S de saída.

- **And (F = 000)**

  Esta operação é feita dentro de um laço que vai percorrer todos os bits de X e Y e realizar a operação de AND, entre bits de mesma posição.

  $X_0\ and\ Y_0,\ X_1\ and\ Y_1,\ X_2\ and\ Y_2,\ X_3\ and\ Y_3$

  O resultado dessa operação vai ser também um vetor de 4 bits com ‘1’ somente nas posições em que tanto X_n quanto Y_n são iguais a ‘1’ e ‘0’ nas demais. Segue tabela verdade:

X	Y	S
0	0	0
0	1	0
1	0	0
1	1	1

- **Or (F = 001)**

  Esta operação é feita dentro de um laço que vai percorrer todos os bits de X e Y e realizar a operação de OR, entre bits de mesma posição.

$X_0\ or\ Y_0,\ X_1\ or\ Y_1,\ X_2\ or\ Y_2,\ X_3\ or\ Y_3$

O resultado dessa operação vai ser também um vetor de 4 bits com ‘1’ somente nas posições em que X_n ou Y_n são iguais a ‘1’ e ‘0’ nas demais.

Segue tabela verdade:

X	Y	S
0	0	0
0	1	1
1	0	1
1	1	1

- **Not (F = 010)**

  Esta operação é feita dentro de um laço que vai percorrer todos os bits do vetor X e realizar a operação de inverter o bit de cada posição.

  $not(X_0),\ not(X_1),\ not(X_2),\ not(\ X_3)$

  O resultado dessa operação vai ser também um vetor de 4 bits com os bits invertidos. Segue tabela verdade:

X	S
0	1
1	0

- **Xor (F = 011)**

  Esta operação é feita dentro de um laço que vai percorrer * realizar a operação de XOR, entre bits de mesma posição.

  $X_0\ xor\ Y_0,\ X_1\ xor\ Y_1,\ X_2\ xor\ Y_2,\ X_3\ xor\ Y_3$

O resultado dessa operação vai ser também um vetor de 4 bits com ‘1’ somente nas posições em que somente um entre X_n e Y_n é igual a ‘1’ e ‘0’ nas demais. Segue tabela verdade:

X	Y	S
0	0	0
0	1	1
1	0	1
1	1	0

- **Full Adder**

Este é um componente que realiza a soma de 1 bit e faz a propagação do carry out e também tem entrada para carry in. Ele foi bastante utilizado no desenvolvimento dos demais componentes.

Ele realiza a soma entre o X, Y e o Carry in com portas XOR, ficando $X\ xor\ Y\ xor\ Carry\ in$. O Carry out é propagado quando o resultado da soma anterior é maior que 1_{10}.

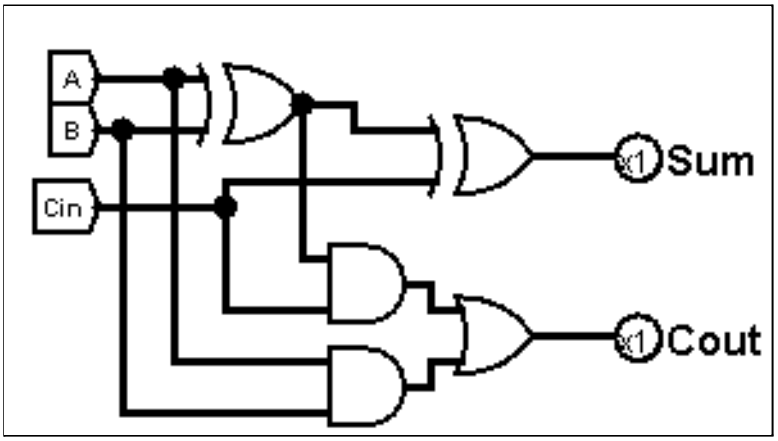


Figura 2: Simulação de um somador completo.

X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- **Adder de 4 bits (F = 100)**

  O somador de 4 bits faz uso de 4 somadores de um bit. Em cada um é feito a soma dos bits da posição de X e Y. O Carry Out de um somador fica ligado ao Carry In de outro somador.

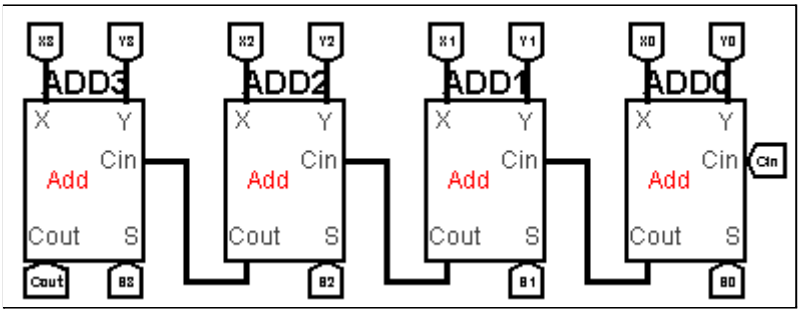


Figura 3: Simulação de um somador de 4 bits.

- **Complemento de 2 (F = 111)**

  O complemento de 2 é útil para representar números negativos e fazer subtrações também. O complemento de 2 de um número é o vetor de bits negado somado mais 1.

  Portanto para descrever esse módulo foi utilizado o somador de 1 bit que recebe como entrada Y_n xor '1' (que vai retornar o inverso de Y_n) e '0' e no Carry in do primeiro somador colocamos '1' para realizar a adição de um no vetor de bits invertido. O Carry out final é ignorado.

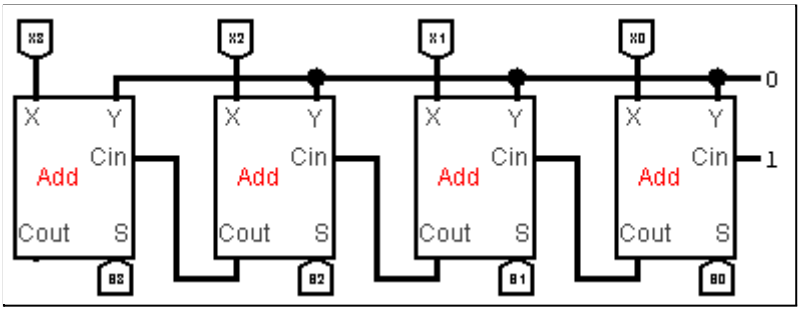


Figura 4: Simulação de um circuito de complemento de 2.

Binário	Comp. de 2
0000	0
0001	1
0010	2
0011	3
0100	4

0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

- **Subtrator de 4 bits (F = 101)**

  Este componente também utiliza o somador de 1 bit para realizar a subtração entre os números utilizando o complemento de 2 de Y.

  Primeiro fazemos o complemento de 2 de Y e somamos com X, o resultado obtido está em complemento 2.

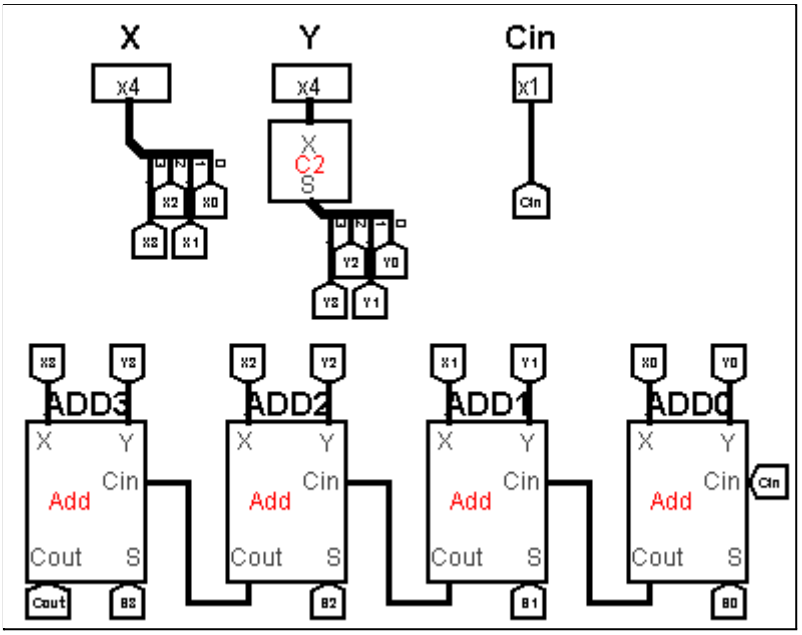


Figura 5: Simulação de um subtrator de 4 bit.s

- **Multiplicador de 4 bit (F = 110)**

  Este componente utiliza o somador de 1 bit para realizar as contas. A primeira parte é a multiplicação em si que se comporta como uma porta AND na qual o resultado só vai ser ‘1’ se as duas entradas forem ‘1’, parte semelhante a multiplicação decimal em que multiplicar por 0 é igual a 0.

  A multiplicação é feita entre todos os bits dos vetores X e Y, um por um e o resultado é somado entre si, algoritmo análogo ao de multiplicação decimal.

  Nesse caso o vetor do resultado S tem 8 bits.

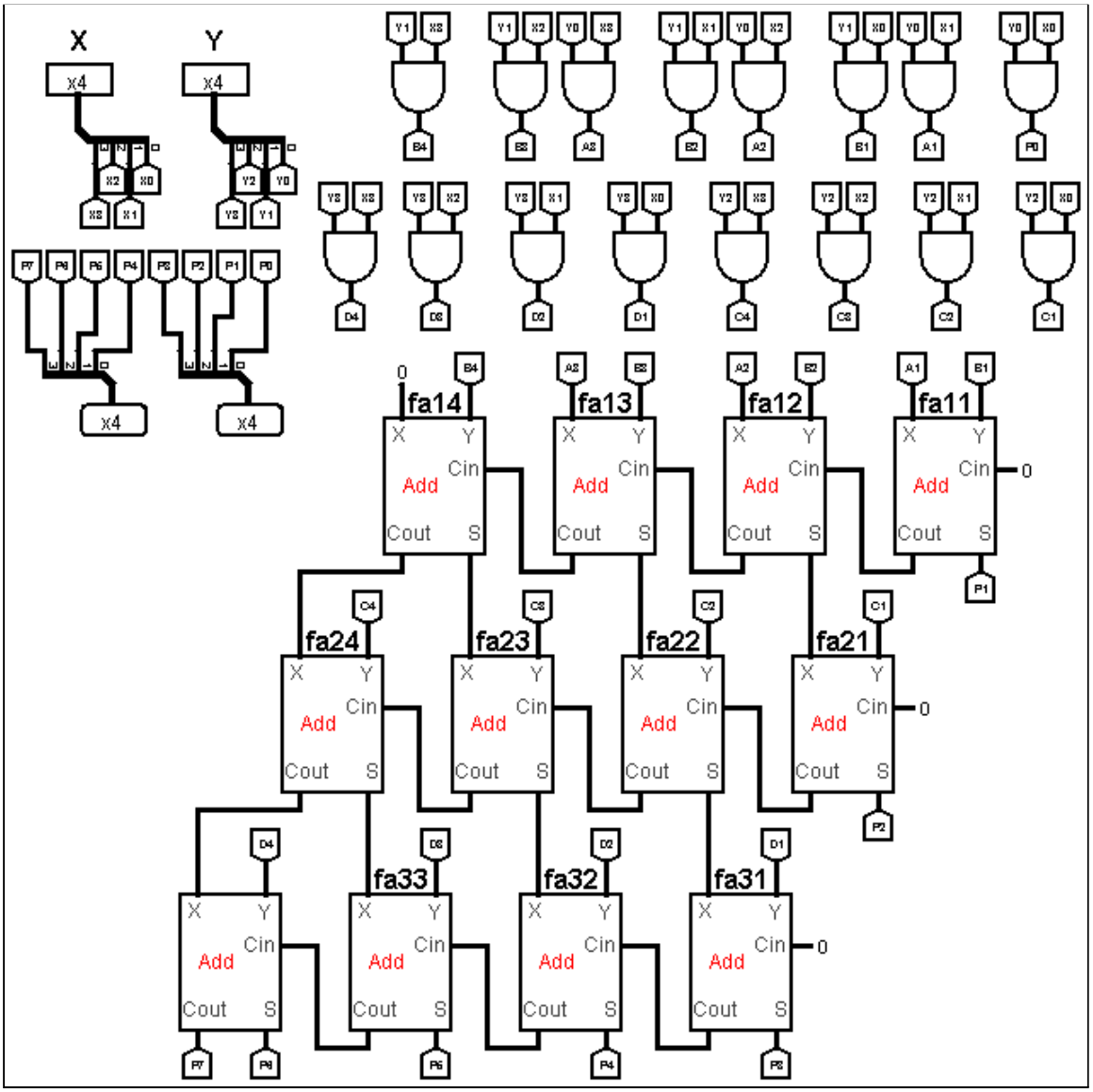


Figura 6: Simulação de um circuito multiplicador de 4 bits

- **ULA (Unidade Lógica Aritmética)**

  É um módulo que serve para unir todos os 8 componentes que realizam operações e que seleciona a operação a partir da entrada de função de 3 bits.

Tem como entrada dois vetores X e Y de 4 bits, um vetor F de 3 bits que define a função e como saída possui um vetor S de 8 bits com o resultado da operação e um bit do borrow out. Para que o tamanho da saída das operações sejam compatíveis, aquelas que têm 4 bits são concatenadas com “0000” no MSB.

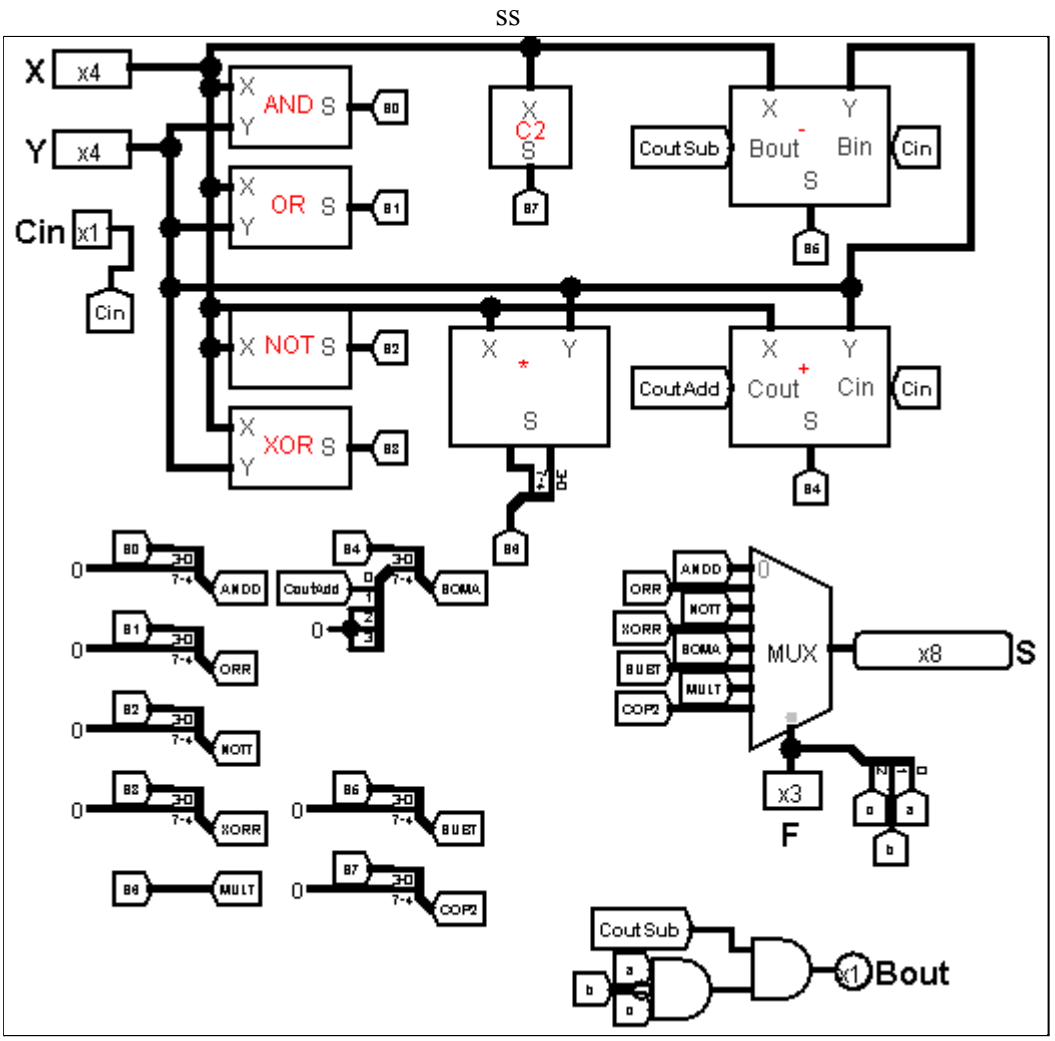


Figura 7: Simulação de uma ULA de 8 operações.

- **Interface**

É um componente que vai se conectar com a ULA para registrar os vetores de entrada e de saída, de acordo com a saída que ocorre juntamente com o clock.

A interface tem 5 entradas, uma para o clock, um de reset o sistema para recomeçar as operações, um que seleciona os número e da inicio as operações e mais dois vetores de 4 bits que representam as entradas X e Y. Possui também 5 saídas, um vetor de 8 bits para o resultado da operação, um para propagar o barrow out, um vetor de 3 bits que representa a função do momento e mais dois vetores de 4 bits que representa o número de entrada que estará no display de 7 segmentos.

Utiliza como componente a ULA e monta uma lógica para que o sistema reinicie toda vez que o botão de reset seja apertado, que os número sejam

definidos quando o botão de seleção é apertado e posteriormente ele começe a realizar operações em sequência, até que o sistema seja resetado.

- **Decodificador de 7 segmentos**

  É um componente que decodifica um número binário para BCD no formato que o número é mostrado no display de 7 segmentos.

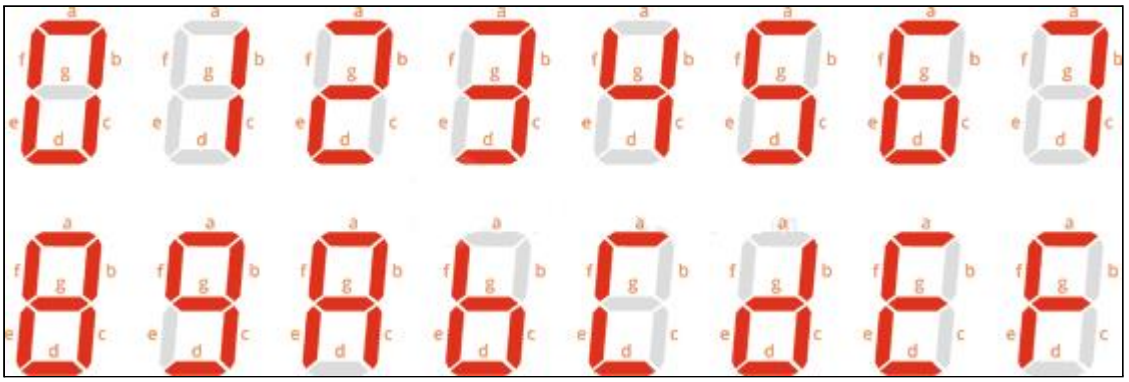


Figura 8: Decodificação de binário para BCD em um display de 7 segmentos. Disponível em: https://bit.ly/3f3ixIQ

- **Divisor de frequência**

  É o responsável por fazer com que o clock da placa do laboratório online de 50MHz passe para um período de 1 segundo. O novo clock ocorre a cada $25 \cdot 10^6$ ciclos do clock de 50MHz.

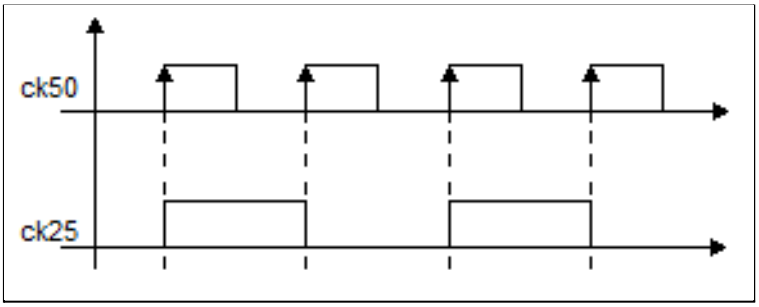


Figura 9: Ideia de como o divisor de frequência funciona. Disponível em: https://bit.ly/3HNkndg

- **Placa**

  O componente top level, é o que reúne todos os componentes desenvolvidos e consegue interagir com a FPGA online. Os nomes das entradas e saídas já estão substituídos pelos nomes dos pinos da placa.

  Possui como entradas, o clock de 50 MHz, um par de botão de pressão que representa o reset (botão (0)) e o sel (botão (1)) e também switches que irão ler os números de entradas, o vetor X é definido nos botões 17, 16, 15 e 14 e o vetor Y é definido nos botões 12, 11, 10 e 9.

  As saídas são 3 leds vermelhos que piscam de acordo com a função do momento, um led verde que acende caso o resultado da subtração seja negativo (borrow out negado) e outros 8 displays de 7 segmentos, 2 para

representar o número X em decimal, 2 para representar o número Y em decimal e 4 outros displays que representaram as saídas. Nas saídas das operações lógicas o resultado é dado em binário é nas operações aritméticas o resultado é dado em hexadecimal.

Após definir os números nos switchs e apertar o botão SEL as operações começam a ocorre e vão mudando a cada um segundo (definido pelo clock). Para mudar os número deve-se alterar os switchs como desejado, apertar reset e depois SEL.

Resultados

- **And (4 bits)**

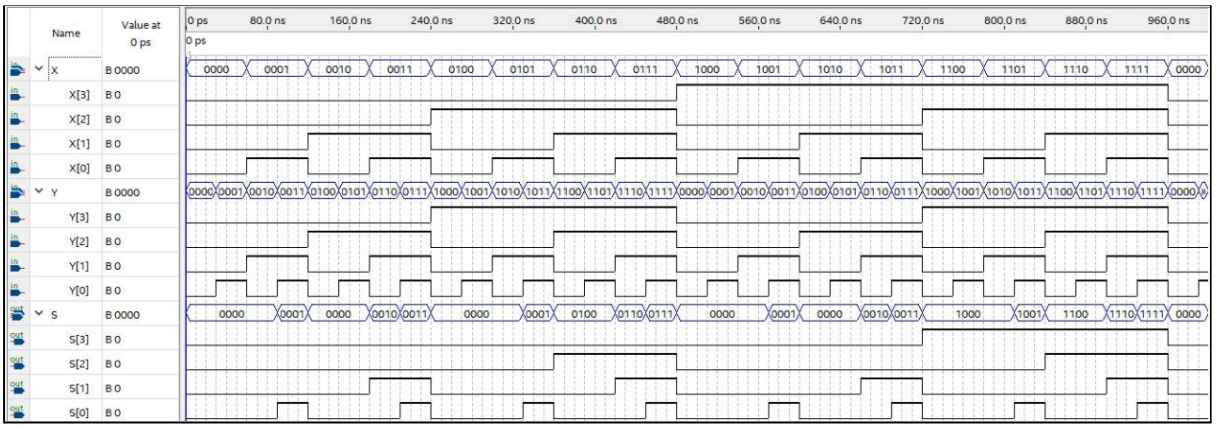


Figura 10: Funcionamento do *And* (4 bits)

- **Or (4 bits)**

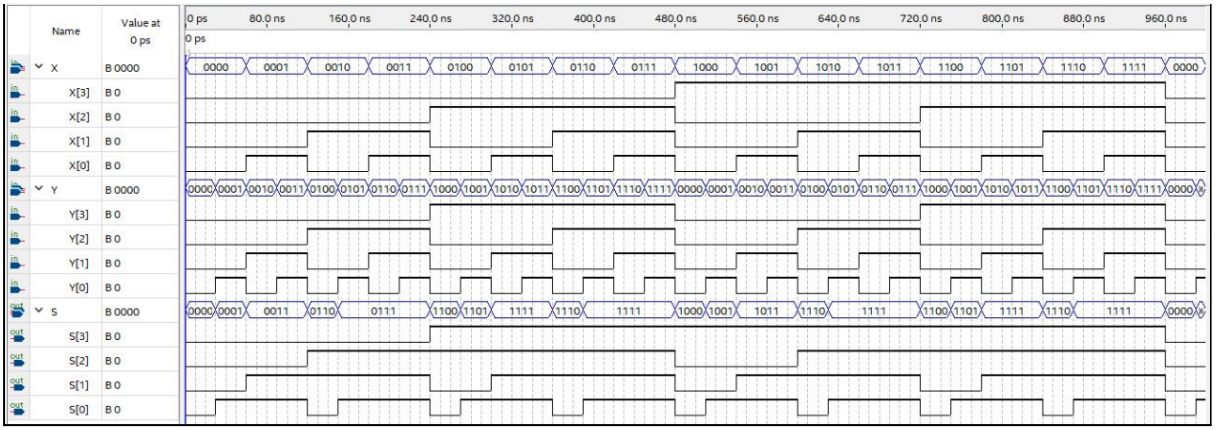


Figura 11: Funcionamento do *Or* (4 bits)

- **Not (4 bits)**

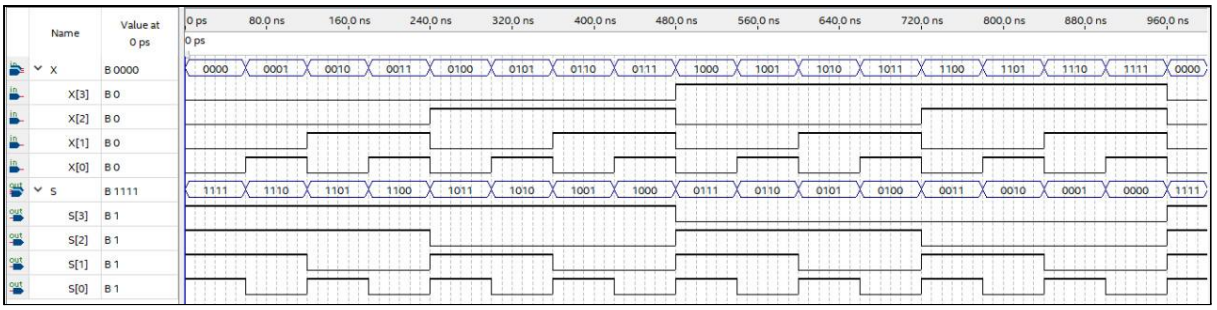


Figura 12: Funcionamento do *Not* (4 bits)

- **Xor (4 bits)**

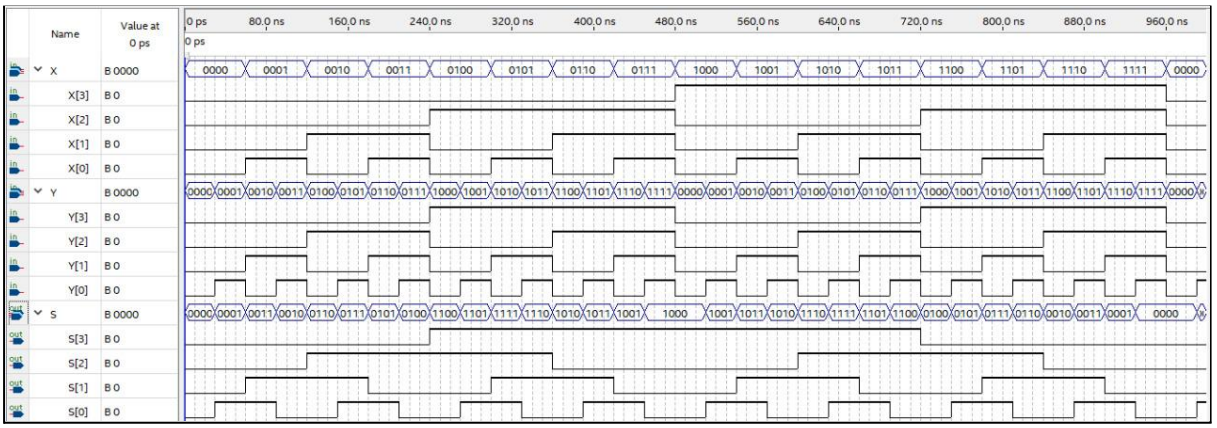


Figura 13: Funcionamento do *Xor* (4 bits)

- **Full Adder (4 bits)**

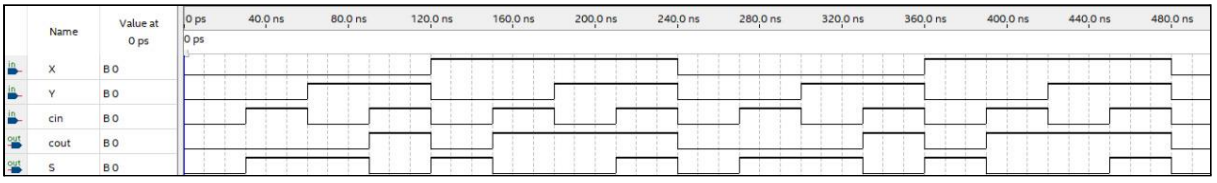


Figura 14: Funcionamento do *Full Adder* (4 bits)

- **Adder (4 bits)**

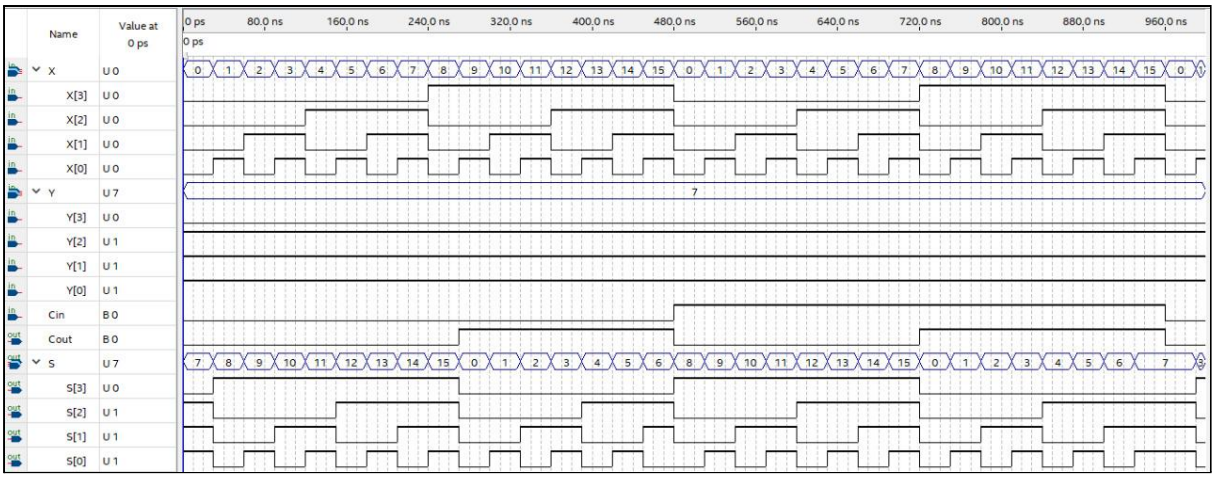


Figura 15: Funcionamento do *Adder* (4 bits)

- **Subtrator (4 bits)**

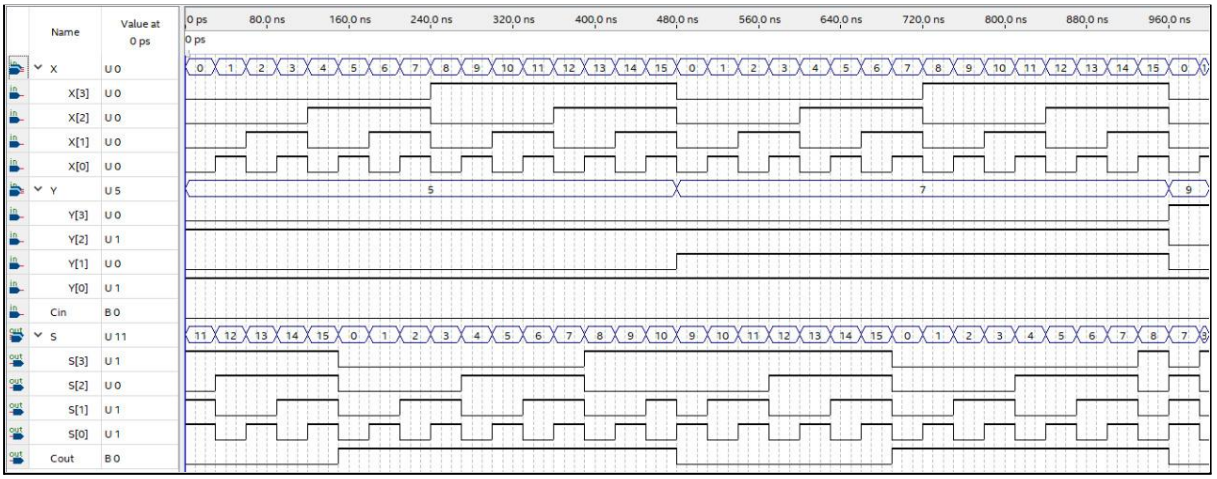


Figura 16: Funcionamento do *Subtrator* (4 bits)

- **Multiplicador (4 bits)**

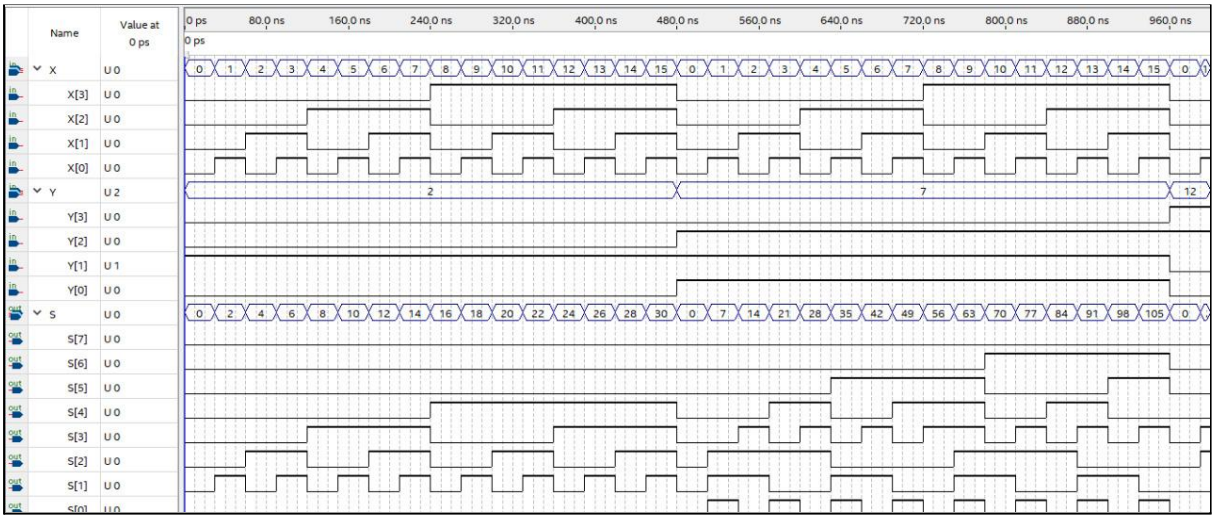


Figura 17: Funcionamento do *Multiplicador* (4 bits)

- **Complemento de 2 (4 bits)**

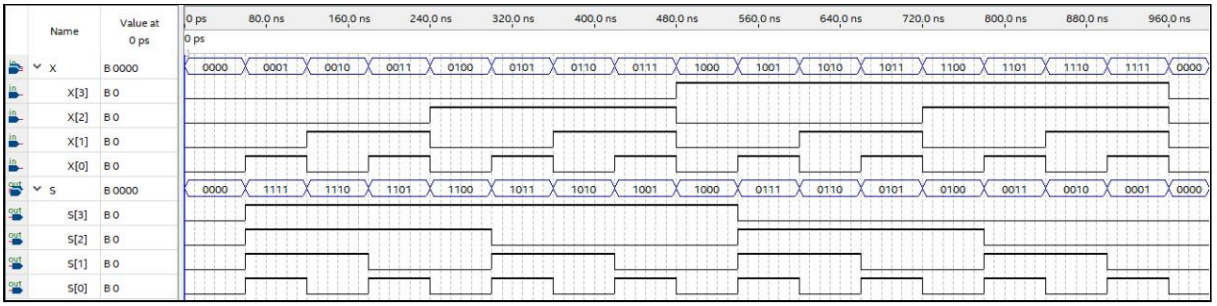


Figura 18: Funcionamento do *Complemento de 2* (4 bits)

- **ULA**

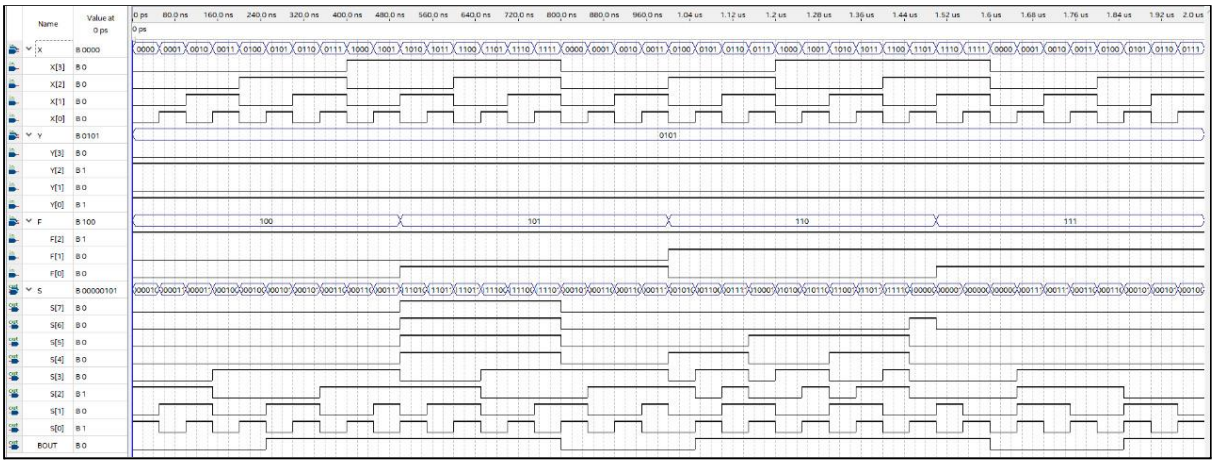


Figura 19: Funcionamento da ULA

- **Interface**

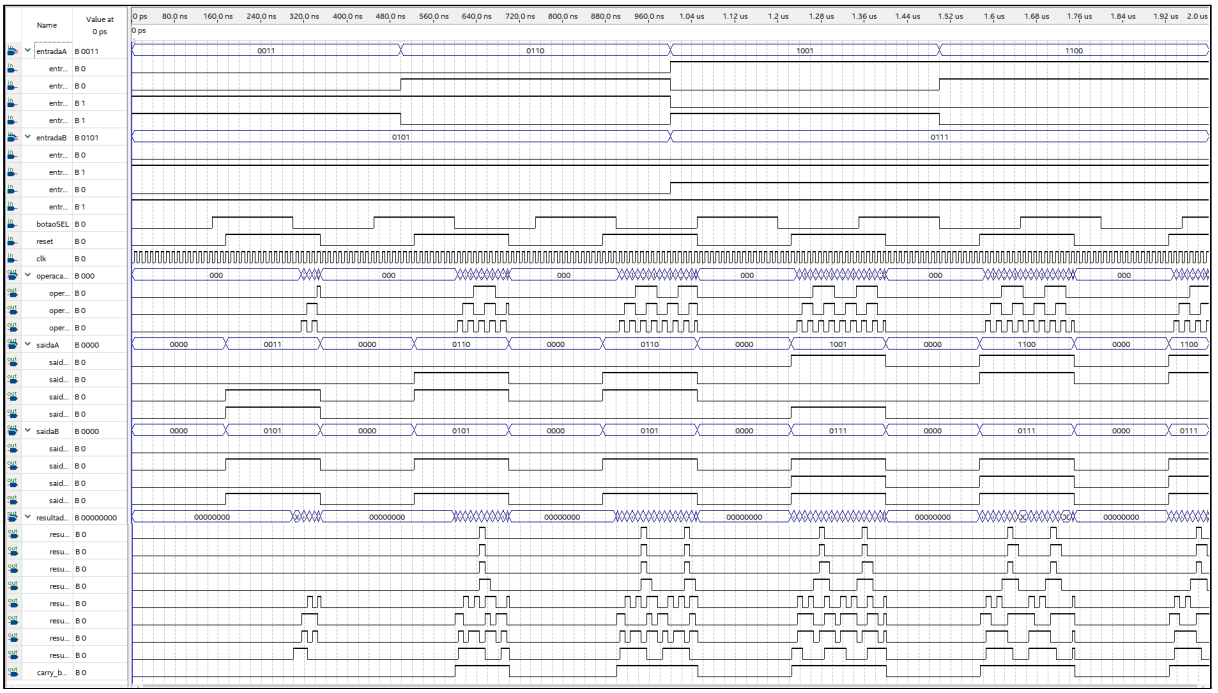


Figura 20: Funcionamento da Interface

- **Decodificador 7 segmentos**

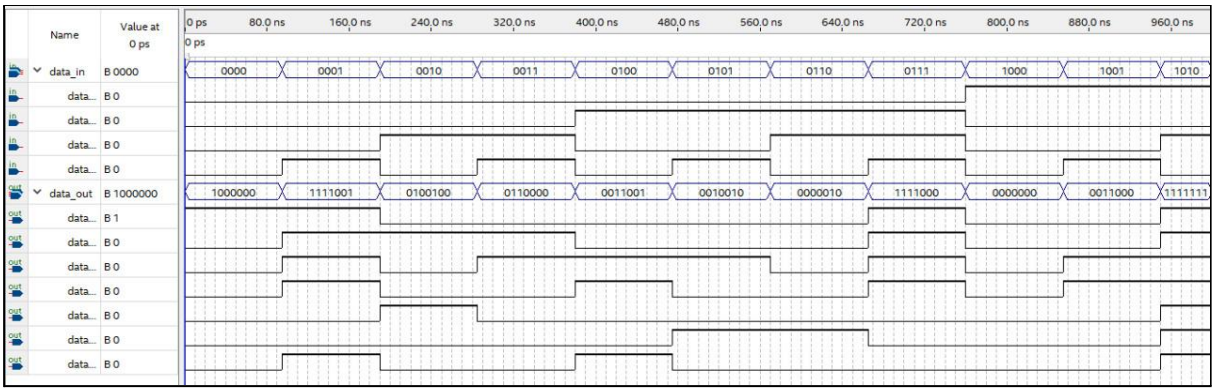


Figura 21: Funcionamento da Decodificador 7 segmentos

- **Divisor de frequência**

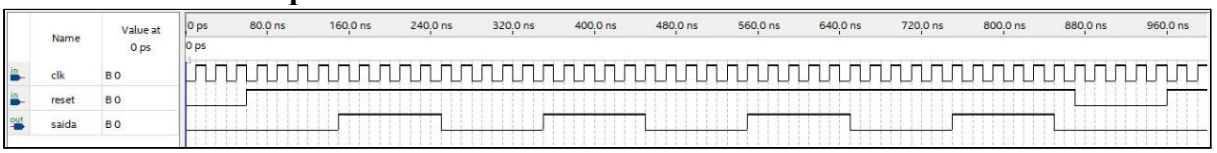


Figura 22: Funcionamento do Divisor de frequência

- **Placa**

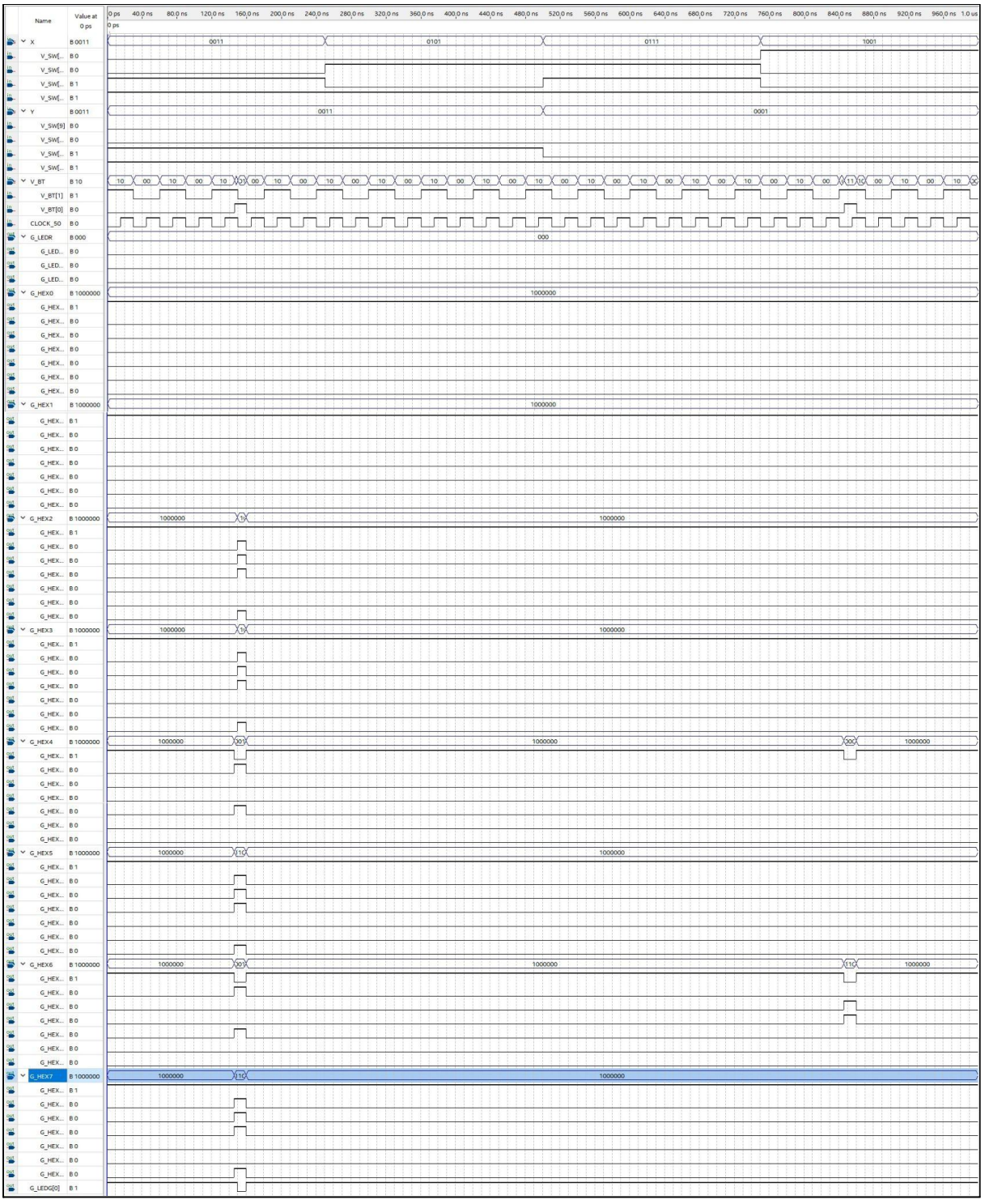


Figura 23: Funcionamento da Placa

Conclusões

No decorrer deste relatório foi descrito as partes de desenvolvimento de uma Unidade Lógica Aritmética e sua implementação em VHDL. O trabalho mostrou como as partes separadas foram utilizadas para conseguir montar um sistema único e coeso no final, mostrando como funciona bem a ideia de divisão do trabalho, resolvendo por partes.

Apêndice

- **And**

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

entity and_4b is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end and_4b;

architecture and_4b_behav of and_4b is
    begin
        andd: for i in 0 to 3 generate
            S(i) <=  X(i) and Y(i);

        end generate andd;
end and_4b_behav;
```

Apêndice 1: Implementação em VHDL do *And* (4 bits)

- **Or**

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

entity or_4b is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end or_4b;

architecture or_4b_behav of or_4b is
    begin
        orr:  for i in 0 to 3 generate
            S(i) <=  X(i) or Y(i);

        end generate orr;
end or_4b_behav;
```

Apêndice 2: Implementação em VHDL do *OR* (4 bits)

- **Not**

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

entity not_4b is port
    (
        X    : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end not_4b;

architecture not_4b_behav of not_4b is
    begin
        nott:   for i in 0 to 3 generate
            S(i) <= not(X(i));

        end generate nott;
end not_4b_behav;
```

Apêndice 3: Implementação em VHDL do *Not* (4 bits)

- **Xor**

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

entity xor_4b is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end xor_4b;

architecture xor_4b_behav of xor_4b is
    begin
        xorr: for i in 0 to 3 generate
            S(i) <=  X(i) xor Y(i);

        end generate xorr;
end xor_4b_behav;
```

Apêndice 4: Implementação em VHDL do *Xor* (4 bits)

- **Fulladder**

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

entity fulladder is port
    (
        X, Y : in  std_logic;
        Cin  : in  std_logic;
        S    : out std_logic;
        Cout : out std_logic
    );
end fulladder;

architecture fa_behav of fulladder is
    begin
        S    <= (X xor Y) xor cin;
        Cout <= (X and Y) or ((X xor Y) and Cin);

end fa_behav;
```

Apêndice 5: Implementação em VHDL do *Full Adder* (4 bits)

- **Somador de 4 bits**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity adder4bit is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        Cin  : in  std_logic;
        Cout : out std_logic;
        S    : out std_logic_vector(3 downto 0)
    );
end adder4bit;

architecture adder_behav of adder4bit is

    component fulladder port
        (
            X, Y : in  std_logic;
            Cin  : in  std_logic;
            S    : out std_logic;
            Cout : out std_logic
        );
    end component;

    signal sinal : std_logic_vector(2 downto 0);

    begin
        fa1: fulladder port map(X(0), Y(0),     Cin , S(0),  sinal(0));
        fa2: fulladder port map(X(1), Y(1), sinal(0), S(1),  sinal(1));
        fa3: fulladder port map(X(2), Y(2), sinal(1), S(2),  sinal(2));
        fa4: fulladder port map(X(3), Y(3), sinal(2), S(3),      Cout);

end adder_behav;
```

Apêndice 6: Implementação em VHDL do *Adder* (4 bits)

- **Subtrator de 4 bits**

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

entity sub4bit is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        Cin  : in  std_logic;   -- Borrow out
        Cout : out std_logic;   -- Borrow in
        S    : out std_logic_vector(3 downto 0)
    );
end sub4bit;

architecture sub4bit_behav of sub4bit is

    component fulladder port
        (
            X, Y : in  std_logic;
            Cin  : in  std_logic;
            S    : out std_logic;
            Cout : out std_logic
        );
    end component;

    component compl2 port
        (
            X   : in  std_logic_vector(3 downto 0);
            S    : out std_logic_vector(3 downto 0)
        );
    end component;

    signal sinal : std_logic_vector(3 downto 0);
    signal compl : std_logic_vector(3 downto 0);

    begin
        cp1: compl2    port map(Y, compl);

        fa1: fulladder port map(X(0), compl(0),     Cin , S(0),  sinal(0));
        fa2: fulladder port map(X(1), compl(1), sinal(0), S(1),  sinal(1));
        fa3: fulladder port map(X(2), compl(2), sinal(1), S(2),  sinal(2));
        fa4: fulladder port map(X(3), compl(3), sinal(2), S(3),      Cout);

end sub4bit_behav;
```

Apêndice 7: Implementação em VHDL do *Subtrator* (4 bits)

- **Multiplicador de 4 bits**

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

entity mult4bit is port
    (
        X, Y   : in  std_logic_vector(3 downto 0);
        S      : out std_logic_vector(7 downto 0)
    );
end mult4bit;

architecture mult4bit_comport of mult4bit is

    component fulladder port
        (
            X, Y : in  std_logic;
            Cin  : in  std_logic;
            S    : out std_logic;
            Cout : out std_logic
        );
    end component;

    signal s1 : std_logic_vector(3 downto 1);
    signal s2 : std_logic_vector(3 downto 1);
    signal c1 : std_logic_vector(4 downto 1);
    signal c2 : std_logic_vector(4 downto 1);
    signal c3 : std_logic_vector(3 downto 1);

    begin
        S(0) <= X(0) and Y(0);

        fa11: fulladder port map(X(1) and Y(0), X(0) and Y(1),   '0',  S(1), c1(1));
        fa12: fulladder port map(X(2) and Y(0), X(1) and Y(1), c1(1), s1(1), c1(2));
        fa13: fulladder port map(X(3) and Y(0), X(2) and Y(1), c1(2), s1(2), c1(3));
        fa14: fulladder port map(          '0', X(3) and Y(1), c1(3), s1(3), c1(4));

        fa21: fulladder port map(        s1(1), X(0) and Y(2),   '0',  S(2), c2(1));
        fa22: fulladder port map(        s1(2), X(1) and Y(2), c2(1), s2(1), c2(2));
        fa23: fulladder port map(        s1(3), X(2) and Y(2), c2(2), s2(2), c2(3));
        fa24: fulladder port map(        c1(4), X(3) and Y(2), c2(3), s2(3), c2(4));

        fa31: fulladder port map(        s2(1), X(0) and Y(3),   '0',  S(3), c3(1));
        fa32: fulladder port map(        s2(2), X(1) and Y(3), c3(1),  S(4), c3(2));
        fa33: fulladder port map(        s2(3), X(2) and Y(3), c3(2),  S(5), c3(3));
        fa34: fulladder port map(        c2(4), X(3) and Y(3), c3(3),  S(6),  S(7));

end mult4bit_comport;
```

Apêndice 8: Implementação em VHDL do *Multiplicador* (4 bits)

- **Complemento de 2**

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

entity compl2 is port
    (
        X   : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end compl2;

architecture compl2_behav of compl2 is

    component fulladder port
        (
            X, Y : in  std_logic;
            Cin  : in  std_logic;
            S    : out std_logic;
            Cout : out std_logic
        );
    end component;

    signal sinal : std_logic_vector(3 downto 0);

    begin
        fa1: fulladder port map(X(0) xor '1', '0',      '1', S(0), sinal(0));
        fa2: fulladder port map(X(1) xor '1', '0', sinal(0), S(1), sinal(1));
        fa3: fulladder port map(X(2) xor '1', '0', sinal(1), S(2), sinal(2));
        fa4: fulladder port map(X(3) xor '1', '0', sinal(2), S(3), sinal(3));

end compl2_behav;
```

Apêndice 9: Implementação em VHDL do *Complemento de 2* (4 bits)

- **ULA**

```
-- cout = 0 -> número negativo
-- cout = 1 -> número positvo

-- sel  operaçao
-- 000  X and Y
-- 001  X OR  Y
-- 010  not X
-- 011  X xor Y
-- 100  X + Y
-- 101  X - Y
-- 110  X * Y
-- 111  X comp 2

library IEEE;
use ieee.std_logic_1164.all;
--use ieee.std_logic_unsigned.all;

entity P10ula is port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        F    : in  std_logic_vector(2 downto 0);
        S    : out std_logic_vector(7 downto 0);
        BOUT : out std_logic
    );
end P10ula;

architecture rtl of P10ula is

    -- Operação and
    component P01and_4b port
        (
            X, Y : in  std_logic_vector(3 downto 0);
            S    : out std_logic_vector(3 downto 0)
        );
    end component;

    -- Operação or
    component P02or_4b port
        (
            X, Y : in  std_logic_vector(3 downto 0);
            S    : out std_logic_vector(3 downto 0)
        );
    end component;
```

Apêndice 10.1

```
-- Operação not
component P03not_4b port
    (
        X    : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end component;

-- Operação xor
component P04xor_4b port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        S    : out std_logic_vector(3 downto 0)
    );
end component;

-- Operação soma 4 bits
component P06adder4bit port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        Cin  : in  std_logic;
        Cout : out std_logic;
        S    : out std_logic_vector(3 downto 0)
    );
end component;

-- Operação subtrair 4 bits
component P07sub4bit port
    (
        X, Y : in  std_logic_vector(3 downto 0);
        Cin  : in  std_logic;
        Cout : out std_logic;
        S    : out std_logic_vector(3 downto 0)
    );
end component;

-- Operação multiplicar 4 bits
component P08mult4bit port
    (
        X, Y   : in  std_logic_vector(3 downto 0);
        S      : out std_logic_vector(7 downto 0)
    );
end component;
```

Apêndice 10.2

```
    -- Operação complemento de 2 de X
    component P09compl2 port
         (
            X   : in  std_logic_vector(3 downto 0);
            S   : out std_logic_vector(3 downto 0)
         );
    end component;

    signal r_and, r_or, r_not, r_xor, r_sum, r_sub, r_cp2   :   std_logic_vector(3 downto 0);
    signal r_mult                                           :   std_logic_vector(7 downto 0);
    signal cout, boutt                                      :   std_logic;

    begin
        andd    :   P01and_4b       port map(X, Y, r_and);
        orr     :   P02or_4b        port map(X, Y, r_or );
        nott    :   P03not_4b       port map(X,    r_not);
        xorr    :   P04xor_4b       port map(X, Y, r_xor);
        sum     :   P06adder4bit    port map(X, Y,   '0', cout , r_sum);
        sub     :   P07sub4bit      port map(X, Y,   '0', boutt, r_sub);
        mult    :   P08mult4bit     port map(X, Y, r_mult);
        cop2    :   P09compl2       port map(X,    r_cp2);
        BOUT    <=  boutt;

        process(F, r_and, r_or, r_not, r_xor, r_sum, r_sub, r_cp2, r_mult)
            begin

                case F is
                    when "000"  => S <= "0000" & r_and;                     -- X and Y
                    when "001"  => S <= "0000" & r_or;                      -- X or  Y
                    when "010"  => S <= "0000" & r_not;                     -- not X
                    when "011"  => S <= "0000" & r_xor;                     -- X xor Y
                    when "100"  => S <= "000"  & cout & r_sum;              -- X + Y
                    When "101"  => S <= boutt & boutt & boutt & boutt & r_sub;  -- X - Y
                    when "110"  => S <= r_mult;                             -- X * Y
                    when "111"  => S <= "0000" & r_cp2;                     -- cp2 X
                    when others => S <= "ZZZZZZZZ";
                end case;
        end process;

end rtl;
```

Apêndice 10.3: Implementação em VHDL da ULA

- **Interface**

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

entity P11interface is port
    (
        clk, reset, botaoSEL    : in  std_logic;                    -- botaoSEL carrega A e B simultaneamente
        entradaA, entradaB      : in  std_logic_vector(3 downto 0); -- switches
        resultadoDISPLAY        : out std_logic_vector(7 downto 0); -- display pra A e B tbm
        carry_borrowLED         : out std_logic;
        operacaoLED             : out std_logic_vector(2 downto 0);
        saidaA, saidaB          : out std_logic_vector(3 downto 0)
    );
end P11interface;

architecture behav of P11interface is

    component P10ula port
        (
            X, Y    : in  std_logic_vector(3 downto 0);
            F       : in  std_logic_vector(2 downto 0);
            S       : out std_logic_vector(7 downto 0);
            BOUT    : out std_logic
        );
    end component;

    signal A, B         : std_logic_vector(3 downto 0) := "0000";
    signal result       : std_logic_vector(7 downto 0) := "00000000";
    signal carry_borrow : std_logic := '0';
    signal operacao     : std_logic_vector(2 downto 0) := "000";

    type tipo_estado is (entrada, saida);
    signal estado       : tipo_estado := entrada;

    signal init: std_logic := '1';
```

Apêndice 11.1

```
begin
    alu: P10ula port map (A, B, operacao, result, carry_borrow);

    resultadoDISPLAY    <= result;
    operacaoLED         <= operacao;
    carry_borrowLED     <= carry_borrow;

    process(clk, botaoSEL, reset)
        begin
            if (reset = '0') then
                estado  <= entrada;
                A       <= "0000";
                B       <= "0000";
                saidaA  <= "0000";
                saidaB  <= "0000";
                operacao <= "000";
            elsif (estado = entrada and botaoSEL = '1') then -- definindo A e B
                saidaA <= entradaA;
                saidaB <= entradaB;
            elsif (estado = entrada and botaoSEL = '0') then -- botao manda 0 quando eh apertado
                estado  <= saida;
                A       <= entradaA;
                B       <= entradaB;
                saidaA  <= entradaA;
                saidaB  <= entradaB;
                operacao <= "000";
            elsif (estado = saida and rising_edge(clk))then
                if init = '1' then
                    estado <= entrada;
                    A <= "0000";
                    B <= "0000";
                    saidaA <= "0000";
                    saidaB <= "0000";
                    operacao <= "000";
                    init <= '0';
                elsif operacao = "000" then
                    operacao <= "001";
                elsif operacao = "001" then
                    operacao <= "010";
                elsif operacao = "010" then
                    operacao <= "011";
                elsif operacao = "011" then
                    operacao <= "100";
                elsif operacao = "100" then
                    operacao <= "101";
                elsif operacao = "101" then
                    operacao <= "110";
                elsif operacao = "110" then
```

Apêndice 11.2

```
                    operacao <= "111";
                elsif operacao = "111" then
                    operacao <= "000";
                else
                    operacao <= "000";
                end if;
            end if;
    end process;
end behav;
```

Apêndice 11.3: Implementação em VHDL da Interface

- **Decodificador 7 segmentos**

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

entity decodificador7seg is port
    (
        data_in  :  in  std_logic_vector(3 downto 0); --V_SW   : in std_logic_vector(3 downto 0);
        data_out :  out std_logic_vector(6 downto 0)  --G_HEX0 : out std_logic_vector(6 downto 0)
);
end decodificador7seg;

architecture decode of decodificador7seg is
begin
   with data_in select
        data_out <= "1000000" when "0000",--0
                    "1111001" when "0001",--1
                    "0100100" when "0010",--2
                    "0110000" when "0011",--3     --  ---0---
                    "0011001" when "0100",--4     --  |     |
                    "0010010" when "0101",--5     --  5     1
                    "0000010" when "0110",--6     --  |     |
                    "1111000" when "0111",--7     --  ---6---
                    "0000000" when "1000",--8     --  |     |
                    "0011000" when "1001",--9     --  4     2
                    --"0001000" when "1010",--a    --  |     |
                    --"0000011" when "1011",--b    --  ---3---
                    --"1000110" when "1100",--c
                    --"0100001" when "1101",--d
                    --"0000110" when "1110",--e
                    --"0001110" when "1111",--f
                    "1111111" when others;
end decode;
```

Apêndice 12 Implementação em VHDL do *Decodificador de 7 segmentos*

- **Divisor de frequência**

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

entity divisorFREQ is port
    (
        clk     : in  std_logic;
        reset   : in  std_logic;
        saida   : out std_logic
    );
end divisorFREQ;

architecture display of divisorFREQ is

    signal ck : std_logic;

    begin
        saida <= ck;

    divisor2: process (clk,reset)

        variable cont  : natural range 0 to 25000000 := 0;

        begin
            if (reset='0') then
                ck <='0';

            elsif (clk'event) and (clk='1') then
                cont := cont +1;

                if cont=25000000 then
                    ck   <= NOT ck;
                    cont := 0;
                end if;
            end if;
    end process divisor2;
end display;
```

Apêndice 13: Implementação em VHDL do *Divisor de Frequência*

- **Placa**

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

entity P14placa is port
    (
        CLOCK_50                        : in  std_logic;
        V_BT                            : in  std_logic_vector(1  downto 0);  -- V_BT(0)=reset V_BT(1)=selecao (carrega A e B simultaneamente)
        V_SW                            : in  std_logic_vector(17 downto 9);  -- switches A:17ate14 / B:12ate9
        G_HEX7, G_HEX6                  : out std_logic_vector(6  downto 0);  -- display7seg -> entradaA
        G_HEX5, G_HEX4                  : out std_logic_vector(6  downto 0);  -- display7seg -> entradaB
        G_HEX3, G_HEX2, G_HEX1, G_HEX0  : out std_logic_vector(6  downto 0);  -- display7seg -> resultado
        G_LEDG                          : out std_logic_vector(0  downto 0);  -- borrow out
        G_LEDR                          : out std_logic_vector(2  downto 0)   -- redLED mostra operaçao atual
    );
end P14placa;

architecture behav of P14placa is

    component P11interface port
        (
            clk, reset, botaoSEL : in  std_logic;                    -- botaoSEL carrega A e B simultaneamente
            entradaA, entradaB   : in  std_logic_vector(3 downto 0); -- switches
            resultadoDISPLAY     : out std_logic_vector(7 downto 0); -- display pra A e B tbm
            carry_borrowLED      : out std_logic;
            operacaoLED          : out std_logic_vector(2 downto 0);
            saidaA, saidaB       : out std_logic_vector(3 downto 0)  -- saidas para display A e display B
        );
    end component;

    component P12decodificador7seg port
        (
            data_in     : in  std_logic_vector(3 downto 0); --V_SW   : in  std_logic_vector(3 downto 0);
            data_out    : out std_logic_vector(6 downto 0)  --G_HEX0 : out std_logic_vector(6 downto 0)
        );
    end component;

    component P13divisorFREQ port
        (
            clk     : in  std_logic;
            reset   : in  std_logic;
            saida   : out std_logic
        );
    end component;
```

Apêndice 14.1

```
signal clk1seg, bout                          : std_logic;
signal op                                     : std_logic_vector(2 downto 0);
signal dpA, dpB, dispA0, dispA1, dispB0, dispB1 : std_logic_vector(3 downto 0); --
signal resultado                              : std_logic_vector(7 downto 0) := "00000000";
signal results0, results1, results2, results3 : std_logic_vector(3 downto 0);

begin

    DIVfreq: P13divisorFREQ port map(CLOCK_50, V_BT(0), clk1seg); -- entradas: clock 50 Mhz, Botao de Reset, saida: clock de 1 seg

    ITERF  : P11interface port map( clk             => clk1seg,
                                    reset           => V_BT(0),
                                    botaoSEL        => V_BT(1), -- entrada: clock de 1 seg, reset, botao de seleção
                                    entradaA        => V_SW(17 downto 14),
                                    entradaB        => V_SW(12 downto  9), -- entrada: A,B
                                    resultadoDISPLAY => resultado,
                                    carry_borrowLED  => bout,
                                    operacaoLED      => op,
                                    saidaA           => dpA,
                                    saidaB           => dpB);  -- saida: result,carry/borrowOUT,operacao

    DECOD0  : P12decodificador7seg port map(dispA0, G_HEX6);    --entrada: A, saida: display 7 seg entrada A
    DECOD1  : P12decodificador7seg port map(dispA1, G_HEX7);    --entrada: B, saida: display 7 seg entrada B
    DECOD2  : P12decodificador7seg port map(dispB0, G_HEX4);
    DECOD3  : P12decodificador7seg port map(dispB1, G_HEX5);

    DECOD4  : P12decodificador7seg port map(results0, G_HEX0);
    DECOD5  : P12decodificador7seg port map(results1, G_HEX1);
    DECOD6  : P12decodificador7seg port map(results2, G_HEX2);
    DECOD7  : P12decodificador7seg port map(results3, G_HEX3);

    G_LEDR <= op;
    G_LEDG(0) <= not(bout);

    process (resultado)
        begin
            if (op(2) = '0') then
                results0  <= "000" & resultado(0);
                results1  <= "000" & resultado(1);
                results2  <= "000" & resultado(2);
                results3  <= "000" & resultado(3);
            else
                results0  <= resultado(3 downto 0);
                results1  <= resultado(7 downto 4);
                results2  <= "0000";
                results3  <= "0000";
            end if;
    end process;
```

Apêndice 14.2

```
process (dpA)
    begin
        if (dpA = "1010") then
            dispA0  <= "0000";
            dispA1  <= "0001";
        elsif (dpA = "1011") then
            dispA0  <= "0001";
            dispA1  <= "0001";
        elsif (dpA = "1100") then
            dispA0  <= "0010";
            dispA1  <= "0001";
        elsif (dpA = "1101") then
            dispA0  <= "0011";
            dispA1  <= "0001";
        elsif (dpA = "1110") then
            dispA0  <= "0100";
            dispA1  <= "0001";
        elsif (dpA = "1111") then
            dispA0  <= "0101";
            dispA1  <= "0001";
        else
            dispA0  <= dpA;
            dispA1  <= "0000";
        end if;
end process;
```

Apêndice 14.3

```
        process (dpB)
            begin
                if (dpB = "1010") then
                    dispB0  <= "0000";
                    dispB1  <= "0001";
                elsif (dpB = "1011") then
                    dispB0  <= "0001";
                    dispB1  <= "0001";
                elsif (dpB = "1100") then
                    dispB0  <= "0010";
                    dispB1  <= "0001";
                elsif (dpB = "1101") then
                    dispB0  <= "0011";
                    dispB1  <= "0001";
                elsif (dpB = "1110") then
                    dispB0  <= "0100";
                    dispB1  <= "0001";
                elsif (dpB = "1111") then
                    dispB0  <= "0101";
                    dispB1  <= "0001";
                else
                    dispB0  <= dpB;
                    dispB1  <= "0000";
                end if;
        end process;


end behav;
```

Apêndice 14.4: Implementação em VHDL do sistema da Placa.

Bibliografia e Referências

- A imagem de capa utilizada neste documento não é de minha autoria e foi retirada da web.
- Vídeo e Slides do curso EEL480 - Sistemas Digitais