Relatório - ULA

João Victor Chamarelli, William Petterle Pfaltzgraff, Paulo Cesar de Jesus Ricci Barbosa

[1]DEL - Sistemas Digitais - EEL480

Prof. Pedro Cruz Caminha

1. Introdução

Esse relatório aborda o projeto referente ao trabalho prático 1 de Sistemas Digitais, que consiste em montar uma unidade lógico-aritmética (ULA) de 4 bits e 8 operações usando VHDL, a ser implementada numa placa Spartan 3. É obrigatório que 2 dessas operações sejam soma e subtração a complemento de 2, com as demais sendo de livre escolha. Na secção 2 deste relatório, apresentamos as operações escolhidas e seus códigos, além de seu funcionamento, além do 'código-geral' da ULA, que permite seu funcionamento como instruído. Na secção 3, concluímos esse relatório, com reflexões que tivemos ao longo da elaboração do projeto e considerações finais. Esse documento conta com referências bibliogfráficas ao final.

2. Decisões de projeto

Para o usuário interagir com a placa, optamos por:

1. Utilizar os switches da placa para a leitura dos inputs do usuário em forma de vetor
2. Utilizar um botão onde o usuário, ao apertá-lo, traduza para a placa que ela deve armazenar o valor lido dos switches como o primeiro vetor
3. Utilizar um segundo botão onde o usuário, ao apertá-lo, traduza para a placa que ela deve armazenar o valor lido dos switches como o segundo vetor
4. Utilizar um terceiro botão onde o usuário, ao apertá-lo, traduza para a placa que ela deve armazenar o valor lido dos switches como o código da primeira operação a ser realizada
5. Utilizar um quarto botão onde o usuário, ao apertá-lo, traduza para a placa que ele deseja resetar o estado da placa para ele poder passar novamente o valor do primeiro vetor. O usuário tem a livre escolha de poder pressionar este botão a qualquer momento.

Com isto em mente, implementamos todo o projeto com a premissa de que gostaríamos de utilizar um clock automático, onde os LEDs tivessem comportamento independente da necessidade do usuário dizer que quer transitar para o próximo estado, então os LEDs são atualizados com valores em intervalos regulares tempo, mostrando ao usuário as operações que a máquina está realizando naquele instante.

3. Mapeamento da placa (pinagem) e funcionamento

3.1. Mapeamento

Para que possamos fazer a ULA funcionar, necessitamos especificar os locais na placa que realizarão certas funções. Nessa secção, iremos listar quais locais usamos e para que.

1. *LEDs*: usamos as LEDs para indicar o resultado da operação que a ULA realiza e as flags que ela registra. Seguindo as especificações do trabalho, as LEDs 7, 6, 5 e 4 (W21, Y22, V20, V19) registram as flags da operação, enquanto as LEDs 3, 2, 1 e 0 (U19, U20, T19, R20) registram a saída da operação.
2. *Switches*: a placa conta com 4 switches lógicos (T9, U8, U10, V8), que usamos para registrar as entradas A e B, além da operação a ser realizada.
3. *Buttons*: utilizamos 4 botões para a montagem dessa unidade, que foram os botões cardeais (NORTH/T14, EAST/T16, SOUTH/T15, WEST/U15). Esses botões registram a transição dos estados na máquina de estados que gerencia o recebimento dos inputs para esta ULA. O botão NORTH registra a entrada A, o botão EAST, a entrada B, o botão SOUTH, o código correspondente à operação inicial a ser realizada, e o botão WEST funciona como um reset assíncrono.

3.2. Funcionamento

Para o funcionamento da ULA, criamos uma máquina de estados que gerencia as interações do usuário com os botões cardeais. Inicialmente, há uma variável de nome ”verifica”, que é iniciada em 0, e define o estado dessa máquina. Ao selecionarmos o input A usando os switches e apertarmos o botão North (ButtonN), a máquina armazena o valor de A e ”verifica”passa a valer 1. No estado 1, a máquina aguarda o recebimento do input B, e, ao selecionarmos seu valor nos switches e apertarmos o botão correspondente (ButtonE), o valor de B é armazenado e a máquina transita para o estado 2, onde será selecionada a primeira operação a ser mostrada (primeira operação pois a ULA foi configurada a mostrar todas as operações que ela é capaz de realizar de maneira cíclica. O valor a ser inserido nessa etapa corresponde ao código correspondente à primeira operação a ser realizada pela ULA, antes desta ciclar pelas demais. Por exemplo, deseja-se efetuar uma adição, cujo código é 0000. Logo, coloca-se 0000 como o valor de entrada nesse estado. Cada uma dessas operações tem sua própria máquina de estados interna, que controla o funcionamento do display sequencialmente. Nesse momento, o clock do sistema começa a funcionar - ele é interno a essas máquinas de estados, e conta de, aproximadamente, 2 em 2 segundos. Essa máquina de estado interna mostra, no display, os códigos referentes aos estados dela e seu valor correspondente na ’máquina de estados principal’. Além disso, exclusivamente quando a operação for de soma ou subtração, há um estado que mostra as flags dessa operação (Carry/Borrow, Oveflow, Negativo, Zero).

3.3. Códigos das operações

A ULA possui os seguintes códigos armazenados correspondentes às operações:

1. Código 0000: Operação de soma entre os dois vetores armazenados
2. Código 0001: Operação de subtração entre os dois vetores armazenados
3. Código 0010: Operação AND bit a bit entre os dois vetores armazenados
4. Código 0011: Operação OR bit a bit entre os dois vetores armazenados
5. Código 0100: Operação NOR bit a bit entre os dois vetores armazenados
6. Código 0101: Operação NAND bit a bit entre os dois vetores armazenados
7. Código 0110: Operação XOR bit a bit entre os dois vetores armazenados
8. Código 0111: Operação NOT bit a bit do segundo vetor passado

Caso o usuário passe um código que não corresponda à nenhum código das operações salvas na ULA, a ULA iniciará automaticamente no estado 0000.

4. Implementação

4.1. Soma à complemento de 2

Esse módulo realiza a soma à complemento de 2 entre dois vetores de 4 bits. Para isso, codamos unidades de meio somadores (Half-Adders), que juntos formam uma unidade de soma completa para 1 bit (Full adder). Por se tratar de um circuito expansível, 4 dessas unidades atuam para realizar a soma entre vetores de 4 bits.

4.1.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0000 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor da soma
5	0100	Flags zero, negativo, overflow, carry, nesta ordem

4.1.2. Código

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

entity SUM_2_Vectors_4_Bits is
port (VECTOR_A, VECTOR_B: in std_logic_vector(3 downto 0);
  Cin : in std_logic;
  SUM_RESULT: out std_logic_vector (3 downto 0);
  Cout, V: out std_logic);
end SUM_2_Vectors_4_Bits;

architecture fouradder_structure of SUM_2_Vectors_4_Bits is
  signal COUT_FA0, COUT_FA1, COUT_FA2, COUT_FA3: std_logic;
  component Full_Adder
    port(INPUT_A, INPUT_B, CARRY_IN: in std_logic;
      SUM, CARRY_OUT: out std_logic);
  end component;

  begin
  FA0: Full_Adder port map (VECTOR_A(0), VECTOR_B(0), '0',
     SUM_RESULT(0), COUT_FA0);
  FA1: Full_Adder port map (VECTOR_A(1), VECTOR_B(1),
     COUT_FA0, SUM_RESULT(1), COUT_FA1);
  FA2: Full_Adder port map (VECTOR_A(2), VECTOR_B(2),
     COUT_FA1, SUM_RESULT(2), COUT_FA2);
  FA3: Full_Adder port map (VECTOR_A(3), VECTOR_B(3),
     COUT_FA2, SUM_RESULT(3), COUT_FA3);
  V <= COUT_FA2 xor COUT_FA3;
```

```
  Cout <= COUT_FA3;
end fouradder_structure;
```

4.2. Subtração à complemento de 2

Esse módulo realiza a subtração à complemento de 2 entre dois vetores de 4 bits. Para isso, usamos a mesma lógica usada no módulo do somador, mas, para que a subtração fosse realizada, transformamos o segundo input em seu complemento a 2. Assim, realizamos a operação A + (-B).

4.2.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0001 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor da subtração
5	0100	Flags zero, negativo, overflow, borrow, nesta ordem

4.2.2. Código

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

entity DIFF_2_Vectors_4_Bits is
   Port ( VECTOR_A : in STD_LOGIC_VECTOR (3 downto 0);
          VECTOR_B : in STD_LOGIC_VECTOR (3 downto 0);
          Cin_2 : in STD_LOGIC;
          DIFF_RESULT : out STD_LOGIC_VECTOR (3 downto 0);
          Borrow : out STD_LOGIC;
          V : out STD_LOGIC);
end DIFF_2_Vectors_4_Bits;

architecture Behavioral of DIFF_2_Vectors_4_Bits is
  signal NOT_VECTOR_B : std_logic_vector (3 downto 0);
  component NOT_Vector_4_Bits
       Port (input_vector : in STD_LOGIC_VECTOR (3 downto 0);
            output_vector : out STD_LOGIC_VECTOR (3 downto 0)
            );
  end component;

  component SUM_2_Vectors_4_Bits
       Port (VECTOR_A, VECTOR_B: in std_logic_vector(3 downto
          0);
          Cin : in std_logic;
          SUM_RESULT: out std_logic_vector (3 downto 0);
```

```
        Cout, V: out std_logic);
 end component;


 begin
   NOT_B: NOT_Vector_4_Bits port map (VECTOR_B,
      NOT_VECTOR_B);
   DIFF : SUM_2_Vectors_4_Bits port map(VECTOR_A,
      NOT_VECTOR_B, Cin_2, DIFF_RESULT, Borrow, V);


end Behavioral;
```

4.3. AND

Esse módulo realiza a operação AND bit-a-bit entre os dois vetores de entrada.

4.3.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0010 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor do AND bit a bit

4.3.2. Código

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


entity AND_2_Vectors_4_Bits is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        C : out STD_LOGIC_VECTOR (3 downto 0));
end AND_2_Vectors_4_Bits;

architecture Behavioral of AND_2_Vectors_4_Bits is

begin
 C <= A and B;

end Behavioral;
```

4.4. OR

Esse módulo realiza a operação OR bit-a-bit entre os dois vetores de entrada.

4.4.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0011 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor do OR bit a bit

4.4.2. Código

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

entity OR_2_Vectors_4_Bits is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C : out STD_LOGIC_VECTOR (3 downto 0));
end OR_2_Vectors_4_Bits;


architecture Behavioral of OR_2_Vectors_4_Bits is

 begin
      C <= A or B;

 end Behavioral;
```

4.5. NOR

Esse módulo realiza a operação NOR bit-a-bit entre os dois vetores de entrada.

4.5.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0100 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor do NOR bit a bit

4.5.2. Código

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

```
entity NOR_2_Vectors_4_Bits is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C : out STD_LOGIC_VECTOR (3 downto 0));
end NOR_2_Vectors_4_Bits;


architecture Behavioral of NOR_2_Vectors_4_Bits is

 begin
      C <= NOT (A or B);

 end Behavioral;
```

4.6. NAND

Esse módulo realiza a operação NAND bit-a-bit entre os dois vetores de entrada.

4.6.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0101 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor do NAND bit a bit

4.6.2. Código

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

entity NAND_2_Vectors_4_Bits is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C : out STD_LOGIC_VECTOR (3 downto 0));
end NAND_2_Vectors_4_Bits;

architecture Behavioral of NAND_2_Vectors_4_Bits is

begin
 C <= NOT (A and B);

end Behavioral;
```

4.7. XOR

Nesse módulo, fazemos a operação XOR, bit-a-bit, entre os dois vetores.

4.7.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0110 (código da operação da ULA)
2	0001	Input A
3	0010	Input B
4	0011	Valor do XOR bit a bit

4.7.2. Código

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

entity XOR_2_Vectors_4_Bits is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          C : out STD_LOGIC_VECTOR (3 downto 0));
end XOR_2_Vectors_4_Bits;

architecture Behavioral of XOR_2_Vectors_4_Bits is

begin
C <= a xor b;

end Behavioral;
```

4.8. NOT B

Nesse módulo, fazemos a inversão do valor do vetor, usando a função NOT.

4.8.1. Microestados

Nesta operação definimos microestados que serão mostrados sequencialmente ao usuário:

Microestado	Grupo LEDs à esq	Grupo LEDs à dir
1	0000	0111 (código da operação da ULA)
2	0001	Input B
3	0010	NOT B

4.8.2. Código

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

entity NOT_Vector_4_Bits is
   Port ( input_vector : in STD_LOGIC_VECTOR (3 downto 0);
        output_vector : out STD_LOGIC_VECTOR (3 downto 0));
end NOT_Vector_4_Bits;

architecture Behavioral of NOT_Vector_4_Bits is

begin
  output_vector <= NOT input_vector;

end Behavioral;
```

4.9. ULA

Nesse módulo, juntamos todos os componentes que mostramos acima num só módulo/código, montando a ULA.

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

entity maquinaDeEstados is
  Port ( input_vector : in STD_LOGIC_VECTOR (3 downto 0);
    CLK : in STD_LOGIC;
    ButtonN : in STD_LOGIC; -- Armazena o A
    ButtonE : in STD_LOGIC; -- Armazena o B
    ButtonS : in STD_LOGIC; -- Armazena a primeira operacao
    rst : in STD_LOGIC;
    Outesquerda : out STD_LOGIC_VECTOR (3 downto 0);
    OutDireita : out STD_LOGIC_VECTOR (3 downto 0));
end maquinaDeEstados;

architecture Behavioral of maquinaDeEstados is

    component SUM_2_Vectors_4_Bits
      Port (VECTOR_A, VECTOR_B: in std_logic_vector(3 downto 0);
        Cin : in std_logic;
        SUM_RESULT: out std_logic_vector (3 downto 0);
        Cout, V: out std_logic);
    end component;

    component DIFF_2_Vectors_4_Bits
      Port (VECTOR_A, VECTOR_B: in std_logic_vector(3 downto 0);
        Cin_2 : in std_logic;
        DIFF_RESULT: out std_logic_vector (3 downto 0);
        Borrow, V: out std_logic);
    end component;
```

```
component AND_2_Vectors_4_Bits
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component OR_2_Vectors_4_Bits
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component NOR_2_Vectors_4_Bits
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component NAND_2_Vectors_4_Bits
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component XOR_2_Vectors_4_Bits
  Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
    B : in STD_LOGIC_VECTOR (3 downto 0);
    C : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component NOT_Vector_4_Bits
  Port ( input_vector : in STD_LOGIC_VECTOR (3 downto 0);
    output_vector : out STD_LOGIC_VECTOR (3 downto 0));
end component;

--> Componentes da mquina de estados
signal verifica: integer range 0 to 5 := 0;
signal Selecionador_Fases : integer range 0 to 3 := 0;-----
signal Contador : integer range 0 to 100000000 :=0; ------
   contar 2 segundos
-- signal continuar : integer range 0 to 9 := 9;
signal A : std_logic_vector (3 downto 0); ----- valor do
   primeiro numero
signal B : std_logic_vector (3 downto 0); ----- valor do
   segundo numero
signal ULA_Op_Code : std_logic_vector (3 downto 0); -----
   valor correspondente ao cdigo da operao inicial da ULA
```

```
-- Sadas Somador
signal SOMA : std_logic_vector (3 downto 0); --------
   resultado da soma
signal cout_Soma : std_logic;
signal overflow_soma : std_logic;

-- Sadas Subtrator
signal DIF : std_logic_vector (3 downto 0); --------
   resultado da soma
signal borrow_diff : std_logic;
signal overflow_diff : std_logic;

-- Sada AND dos 2 vetores
signal ANDE : std_logic_vector (3 downto 0);------resultado
   and de A e B

-- Sada OR dos 2 vetores
signal ORE : std_logic_vector (3 downto 0);------resultado
   or de A e B

-- Sada NOR dos 2 vetores
signal NORE : std_logic_vector (3 downto 0);------resultado
   nor de A e B

-- Sada NAND dos 2 vetores
signal NANDE : std_logic_vector (3 downto
   0);------resultado nand de A e B

-- Sada XOR dos 2 vetores
signal XORE : std_logic_vector (3 downto 0);------resultado
   xor de A e B

-- Sada NOT do segundo vetor
signal NOT_B : std_logic_vector (3 downto
   0);------resultado not de B

begin
AND_2 : AND_2_Vectors_4_Bits port map(A,B,ANDE);
SUM : SUM_2_Vectors_4_Bits port map(A, B,'0',SOMA,
   cout_Soma, overflow_soma);
DIFF : DIFF_2_Vectors_4_Bits port map(A, B, '1', DIF,
   borrow_diff, overflow_diff);
OR_2 : OR_2_Vectors_4_Bits port map(A,B,ORE);
NOR_2 : NOR_2_Vectors_4_Bits port map(A,B,NORE);
NAND_2 : NAND_2_Vectors_4_Bits port map(A,B,NANDE);
XOR_2 : XOR_2_Vectors_4_Bits port map(A,B,XORE);
NOT_B : NOT_Vector_4_Bits port map(B, NOT_B);

process(Clk,rst)
```

```
begin
    if (rst = '1') then
        -- Resetar todos os sinais
        Selecionador_Fases <= 0;
        OutEsquerda<="0000";
        OutDireita<="0000";
        -- continuar <= 0;
        Contador <= 0;
        verifica <= 0;
    elsif rising_edge(CLK) then
        -- Lidar com o recebimento dos inputs
        if (buttonN = '1' and Selecionador_Fases = 0) then
            A <= input_vector;
            OutEsquerda<=input_vector;
            Selecionador_Fases <= 1;
        elsif (buttonE = '1' and Selecionador_Fases = 1)
           then
            B <= input_vector;
            OutDireita<=input_vector;
            Selecionador_Fases <= 2;
        elsif (buttonS = '1' and Selecionador_Fases = 2)
           then
            ULA_Op_Code<= input_vector;
            OutEsquerda<= "0000";
            OutDireita<=input_vector;
            Selecionador_Fases <= 3;
        end if;

        if Selecionador_Fases = 3 then
            case ULA_Op_Code is
                when "0000" => -- ULA realiza operao de
                   soma
                    if (verifica = 0) then -- Mquina de
                       estados da operao de soma no
                       estado 0
                        -- Mostrar cdigo da operao da ULA
                        if Contador < 100000000 then
                            Contador <= Contador + 1;
                            OutEsquerda <= "0000";
                            OutDireita <= ULA_Op_Code;
                        else
                            -- Ajustar pra transitar a
                               mquina de estados da
                               operao
                            verifica <= 1;
                            Contador <= 0;
                        end if;
                    elsif (verifica = 1) then -- Mquina
                       de estados da operao de soma no
```

```
    estado 1
      -- Mostrar input A
      if Contador < 100000000 then
           Contador <= Contador + 1;
           OutEsquerda <= "0001";
           OutDireita <= A;
      else
           -- Ajustar pra transitar a
              mquina de estados da
              operao
           verifica <= 2;
           Contador <= 0;
      end if;
elsif (verifica = 2) then -- Mquina
    de estados da operao de soma no
    estado 2
      -- Mostrar input B
      if Contador < 100000000 then
           Contador <= Contador + 1;
           OutEsquerda <= "0010";
           OutDireita <= B;
      else
           -- Ajustar pra transitar a
              mquina de estados da
              operao
           verifica <= 3;
           Contador <= 0;
      end if;
elsif (verifica = 3) then -- Mquina
    de estados da operao de soma no
    estado 3
      -- Mostrar resultado da operao
      if Contador < 100000000 then
           Contador <= Contador + 1;
           OutEsquerda <= "0011";
           OutDireita <= SOMA;
      else
           -- Ajustar pra transitar a
              mquina de estados da
              operao
           verifica <= 4;
           Contador <= 0;
      end if;
elsif (verifica = 4) then -- Mquina
    de estados da operao de soma no
    estado 4
      -- Mostrar flags
      if Contador < 100000000 then
           Contador <= Contador + 1;
```

```
                    OutEsquerda <= "0100";
                    OutDireita(3)<= not
                       (overflow_soma or
                       SOMA(3) or SOMA(2) or
                       SOMA(1) or SOMA(0)); --
                       flag zero
                    OutDireita(2)<= (SOMA(3)
                       and (not overflow_soma))
                       or ((not SOMA(3)) and
                       cout_Soma and
                       overflow_soma); -- flag
                       negativo
                    OutDireita(1)<=
                       overflow_soma; -- flag
                       overflow
                    OutDireita(0)<= cout_Soma;
                       -- flag carry
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0001";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0001" => -- ULA realiza operao de
           subtrao
            if (verifica = 0) then -- Mquina de
               estados da operao de subtrao no
               estado 0
                -- Mostrar cdigo da operao da ULA
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0000";
                    OutDireita <= ULA_Op_Code;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 1;
                    Contador <= 0;
                end if;
            elsif (verifica = 1) then -- Mquina
               de estados da operao de subtrao
               no estado 1
```

```
        -- Mostrar input A
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0001";
            OutDireita <= A;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 2;
            Contador <= 0;
        end if;
    elsif (verifica = 2) then -- Mquina
       de estados da operao de subtrao
       no estado 2
        -- Mostrar input B
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0010";
            OutDireita <= B;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 3;
            Contador <= 0;
        end if;
    elsif (verifica = 3) then -- Mquina
       de estados da operao de subtrao
       no estado 3
        -- Mostrar resultado da operao
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0011";
            OutDireita <= DIF;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 4;
            Contador <= 0;
        end if;
    elsif (verifica = 4) then -- Mquina
       de estados da operao de subtrao
       no estado 4
        -- Mostrar flags
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0100";
```

```
                    OutDireita(3)<= not
                       (overflow_diff or DIF(3)
                       or DIF(2) or DIF(1) or
                       DIF(0)); -- flag zero
                    OutDireita(2)<= (DIF(3) and
                       (not overflow_diff)) or
                       ((not DIF(3)) and
                       borrow_diff and
                       overflow_diff); -- flag
                       negativo
                    OutDireita(1)<=
                       overflow_diff; -- flag
                       overflow
                    OutDireita(0)<=
                       borrow_diff; -- flag
                       carry
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0010";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0010" => -- ULA realiza operao de
           AND dos 2 vetores
            if (verifica = 0) then -- Mquina de
               estados da operao AND no estado 0
                -- Mostrar cdigo da operao da ULA
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0000";
                    OutDireita <= ULA_Op_Code;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 1;
                    Contador <= 0;
                end if;
            elsif (verifica = 1) then -- Mquina
               de estados da operao AND no
               estado 1
                -- Mostrar input A
                if Contador < 100000000 then
```

```
                    Contador <= Contador + 1;
                    OutEsquerda <= "0001";
                    OutDireita <= A;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 2;
                    Contador <= 0;
                end if;
            elsif (verifica = 2) then -- Mquina
               de estados da operao AND no
               estado 2
                -- Mostrar input B
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0010";
                    OutDireita <= B;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 3;
                    Contador <= 0;
                end if;
            elsif (verifica = 3) then -- Mquina
               de estados da operao AND no
               estado 3
                -- Mostrar resultado da operao
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0011";
                    OutDireita <= ANDE;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0011";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0011" => -- ULA realiza operao de
           OR dos 2 vetores
            if (verifica = 0) then -- Mquina de
               estados da operao OR no estado 0
```

```
        -- Mostrar cdigo da operao da ULA
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0000";
            OutDireita <= ULA_Op_Code;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 1;
            Contador <= 0;
        end if;
    elsif (verifica = 1) then -- Mquina
       de estados da operao OR no estado
       1
        -- Mostrar input A
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0001";
            OutDireita <= A;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 2;
            Contador <= 0;
        end if;
    elsif (verifica = 2) then -- Mquina
       de estados da operao OR no estado
       2
        -- Mostrar input B
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0010";
            OutDireita <= B;
        else
            -- Ajustar pra transitar a
               mquina de estados da
               operao
            verifica <= 3;
            Contador <= 0;
        end if;
    elsif (verifica = 3) then -- Mquina
       de estados da operao OR no estado
       3
        -- Mostrar resultado da operao
        if Contador < 100000000 then
            Contador <= Contador + 1;
            OutEsquerda <= "0011";
```

```
                    OutDireita <= ORE;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0100";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0100" => -- ULA realiza operao de
           NOR dos 2 vetores
            if (verifica = 0) then -- Mquina de
               estados da operao NOR no estado 0
                -- Mostrar cdigo da operao da ULA
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0000";
                    OutDireita <= ULA_Op_Code;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 1;
                    Contador <= 0;
                end if;
            elsif (verifica = 1) then -- Mquina
               de estados da operao NOR no
               estado 1
                -- Mostrar input A
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0001";
                    OutDireita <= A;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 2;
                    Contador <= 0;
                end if;
            elsif (verifica = 2) then -- Mquina
               de estados da operao NOR no
               estado 2
                -- Mostrar input B
                if Contador < 100000000 then
```

```
                    Contador <= Contador + 1;
                    OutEsquerda <= "0010";
                    OutDireita <= B;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 3;
                    Contador <= 0;
                end if;
            elsif (verifica = 3) then -- Mquina
               de estados da operao NOR no
               estado 3
                -- Mostrar resultado da operao
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0011";
                    OutDireita <= NORE;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0101";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0101" => -- ULA realiza operao de
           NAND dos 2 vetores
            if (verifica = 0) then -- Mquina de
               estados da operao NAND no estado 0
                -- Mostrar cdigo da operao da ULA
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0000";
                    OutDireita <= ULA_Op_Code;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 1;
                    Contador <= 0;
                end if;
            elsif (verifica = 1) then -- Mquina
               de estados da operao NAND no
               estado 1
```

```
                -- Mostrar input A
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0001";
                    OutDireita <= A;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 2;
                    Contador <= 0;
                end if;
            elsif (verifica = 2) then -- Mquina
               de estados da operao NAND no
               estado 2
                -- Mostrar input B
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0010";
                    OutDireita <= B;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 3;
                    Contador <= 0;
                end if;
            elsif (verifica = 3) then -- Mquina
               de estados da operao NAND no
               estado 3
                -- Mostrar resultado da operao
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0011";
                    OutDireita <= NANDE;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0110";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0110" => -- ULA realiza operao de
           XOR dos 2 vetores
```

```
if (verifica = 0) then -- Mquina de
   estados da operao XOR no estado 0
    -- Mostrar cdigo da operao da ULA
    if Contador < 100000000 then
        Contador <= Contador + 1;
        OutEsquerda <= "0000";
        OutDireita <= ULA_Op_Code;
    else
        -- Ajustar pra transitar a
           mquina de estados da
           operao
        verifica <= 1;
        Contador <= 0;
    end if;
elsif (verifica = 1) then -- Mquina
   de estados da operao XOR no
   estado 1
    -- Mostrar input A
    if Contador < 100000000 then
        Contador <= Contador + 1;
        OutEsquerda <= "0001";
        OutDireita <= A;
    else
        -- Ajustar pra transitar a
           mquina de estados da
           operao
        verifica <= 2;
        Contador <= 0;
    end if;
elsif (verifica = 2) then -- Mquina
   de estados da operao XOR no
   estado 2
    -- Mostrar input B
    if Contador < 100000000 then
        Contador <= Contador + 1;
        OutEsquerda <= "0010";
        OutDireita <= B;
    else
        -- Ajustar pra transitar a
           mquina de estados da
           operao
        verifica <= 3;
        Contador <= 0;
    end if;
elsif (verifica = 3) then -- Mquina
   de estados da operao XOR no
   estado 3
    -- Mostrar resultado da operao
    if Contador < 100000000 then
```

```
                    Contador <= Contador + 1;
                    OutEsquerda <= "0011";
                    OutDireita <= XORE;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da ULA
                    verifica <= 0;
                    ULA_Op_Code <= "0111";
                    Contador <= 0;
                end if;
            else
                verifica <= 0; -- Default case
                   para prevenir estados
                   indefinidos
            end if;
        when "0111" => -- ULA realiza operao de
           NOT do segundo vetor
            if (verifica = 0) then -- Mquina de
               estados da operao OR no estado 0
                -- Mostrar cdigo da operao da ULA
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0000";
                    OutDireita <= ULA_Op_Code;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 1;
                    Contador <= 0;
                end if;
            elsif (verifica = 1) then -- Mquina
               de estados da operao OR no estado
               2
                -- Mostrar input B
                if Contador < 100000000 then
                    Contador <= Contador + 1;
                    OutEsquerda <= "0001";
                    OutDireita <= B;
                else
                    -- Ajustar pra transitar a
                       mquina de estados da
                       operao
                    verifica <= 2;
                    Contador <= 0;
                end if;
            elsif (verifica = 2) then -- Mquina
               de estados da operao OR no estado
               3
```

```
                                    -- Mostrar resultado da operao
                                    if Contador < 100000000 then
                                        Contador <= Contador + 1;
                                        OutEsquerda <= "0010";
                                        OutDireita <= NOT_B;
                                    else
                                        -- Ajustar pra transitar a
                                           mquina de estados da ULA
                                        verifica <= 0;
                                        ULA_Op_Code <= "0000";
                                        Contador <= 0;
                                    end if;
                                else
                                    verifica <= 0; -- Default case
                                       para prevenir estados
                                       indefinidos
                                end if;
                            when others =>
                                verifica <= 0;
                                ULA_Op_Code <= "0000";
                                Contador <= 0;
                        end case;
                    end if;
                end if;
            end process;
        end Behavioral;
```

5. Conclusão

Em síntese, esse primeiro contato prático com a disciplina nos mostrou o quão complexo é o processo realizado pelo computador para entender comandos que são básicos para seres humanos, enquanto observamos a extensa quantidade de tópicos que Sistemas Digitais aborda em seus projetos. Aqui, cito as referências desse relatório: [Albertini 2024]; [GTA/UFRJ 2024]; [Fontes on-line diversas 2024].

6. Github

O projeto está disponível aqui

Referências

Albertini, B. (2024). Tutorial de VHDL em Português.

Fontes on-line diversas (2024). Foruns on-line e sites acessados durante o desenvolvimento do projeto, i.e Stack Overflow, Wikipedia, entre outros.

GTA/UFRJ (2024). Site - Sistemas Digitais - GTA/UFRJ.