INTRODUÇÃO A VHDL

Prof. Raimes Moraes

INTRODUÇÃO A VHDL

Objetivo:

Apresentar a linguagem VHDL para a configuração de PLDs através de exemplos

VHDL

- *VHSIC Hardware Description Language*
  - *VHSIC: Very High Speed Integrated Circuit*

- Linguagem com padrão industrial para descrever, modelar e gerar circuitos digitais **(IEEE 1076-1993).**

- Surgiu a partir da década de 80 (DDEUA) para documentação de projetos.

VHDL

- Vantagens:
  - **linguagem para projeto e simulação;**
  - **portabilidade entre plataformas e componentes;**
  - **mais eficiente que projetos esquemáticos e booleanos (não necessita de documentação explicativa);**
- Desvantagens:
  - **qualidade do projeto gerado varia com o compilador;**
  - **inibe criatividade do projetista;**

Por quê VHDL?

- HDL: Abel, Verilog, AHDL, VHDL.
  - **Padrão Depto. Defesa Americano, IEEE, ANSI.**
  - **VHDL é a mais utilizada (mais de 50 % do mercado)**
  - **Utilizado como ferramenta de modelagem não só em eletrônica, mas em outras áreas de conhecimento (mecânica, hidráulica).**
  - **Utilização em ASIC (*Application Specific Integrated Circuit*)**

VHDL

Versões da linguagem:

- **1987 – Padrão IEEE 1076**
- **1994 – Revisão: Padrão IEEE 1076 '93**
- **2000 - Revisão: Padrão IEEE 1076 2000**
- **2000 - Revisão: Padrão IEEE 1076 2002**
- **2007 - *VHDL Procedural Language Application Interface* (IEEE 1076c 2007): simulador VHDL deve permitir interação com linguagem de programação (C++)**
- **2009 - Revisão: Padrão IEEE 1076 2008**

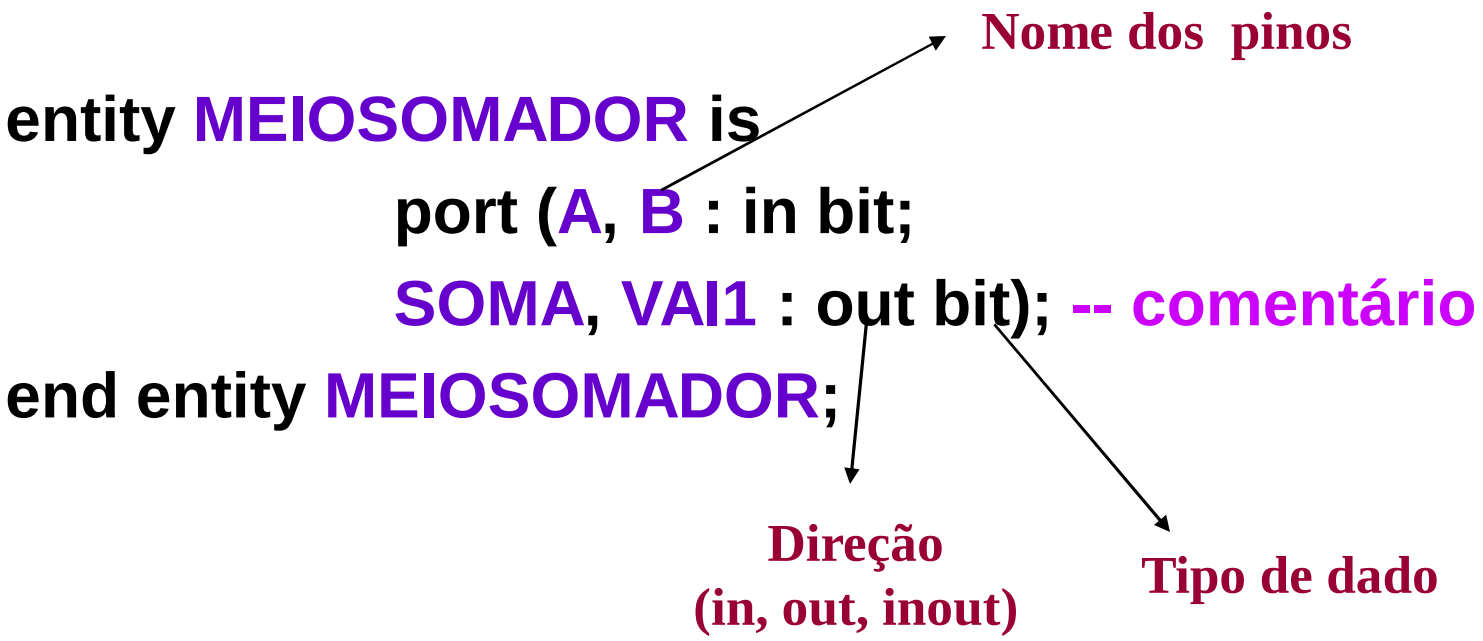
(http://www.doulos.com/knowhow/vhdl_designers_guide/vhdl_2008/)

VHDL

- Módulos da Linguagem
  - **ENTITY**
  - **ARCHITECTURE**
  - **CONFIGURATION**
  - **PACKAGE**
  - **PACKAGE BODY**

Módulo *Entity*

- Descreve os pinos de entrada e saída do dispositivo projetado.



Módulo *Architecture*

- Define a função executada pelo circuito.

```
architecture DFLOW  of  MEIOSOMADOR is
begin
                SOMA <= A xor B;
                VAI1   <= A and B;
end architecture;
```

Módulo *Configuration*

Uma ***entity*** pode possuir mais que uma ***architecture***: Para mesmos pinos de entrada e saída, podem ser associadas diferentes funções a serem executadas.

Configuration é utilizada pra especificar uma dada arquitetura à entidade. Altera recomenda que seja utilizada uma única arquitetura para cada entidade.

```
Configuration EXEMPLO of  MEIOSOMADOR is
                    for DFLOW end for;
end configuration;
```

Operadores

	Prioridade						
Outros		**	**abs**	**not**			
Multiplicação		*	/	**mod** A-B*N	**rem** A-(A/B)*B		
Sinal		+ (sign)	– (sign)				
Adição		+	–	&			
Relacionais		=	/=	<	<=	>	>=
Lógicos		**and**	**or**	**nand**	**nor**	**xor**	**xnor**

MOD: resultado tem sinal de B; REM: result. tem o sinal de A.

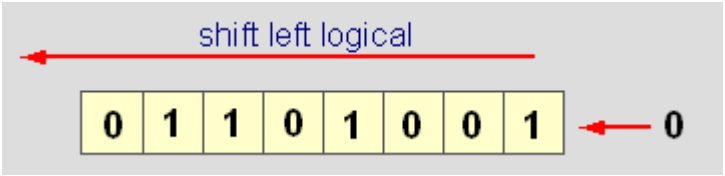
7 mod (-4)=7-4*2 = -1; (-7) rem (4) = -7- (-1)*4 = -3

& (concatenação): "0110" = "01"& "10";

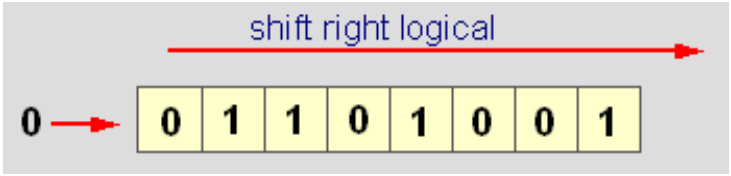
OBS: operadores aritméticos: Aplicam-se a valores inteiros.
operadores lógicos: Aplicam-se a bits.

Operadores de Deslocamento

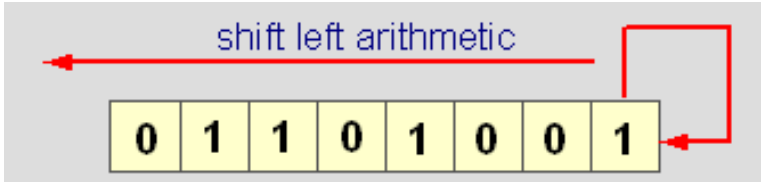
"01101001" sll 1 = "11010010"



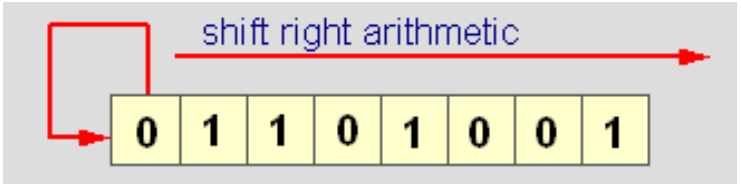
"01101001" srl 1 = "00110100"



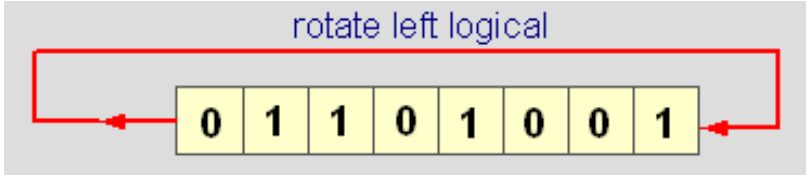
"01101001" sla 1 = "11010011"



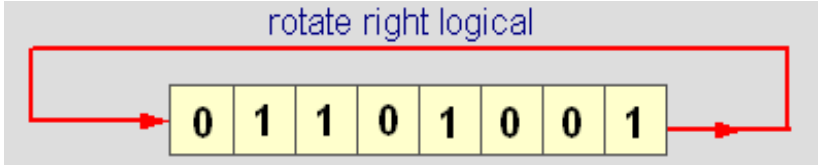
"01101001" sra 1 = "00110100"



"01101001" rol 1 = "11010010"



"01101001' ror 1 = "10110100"



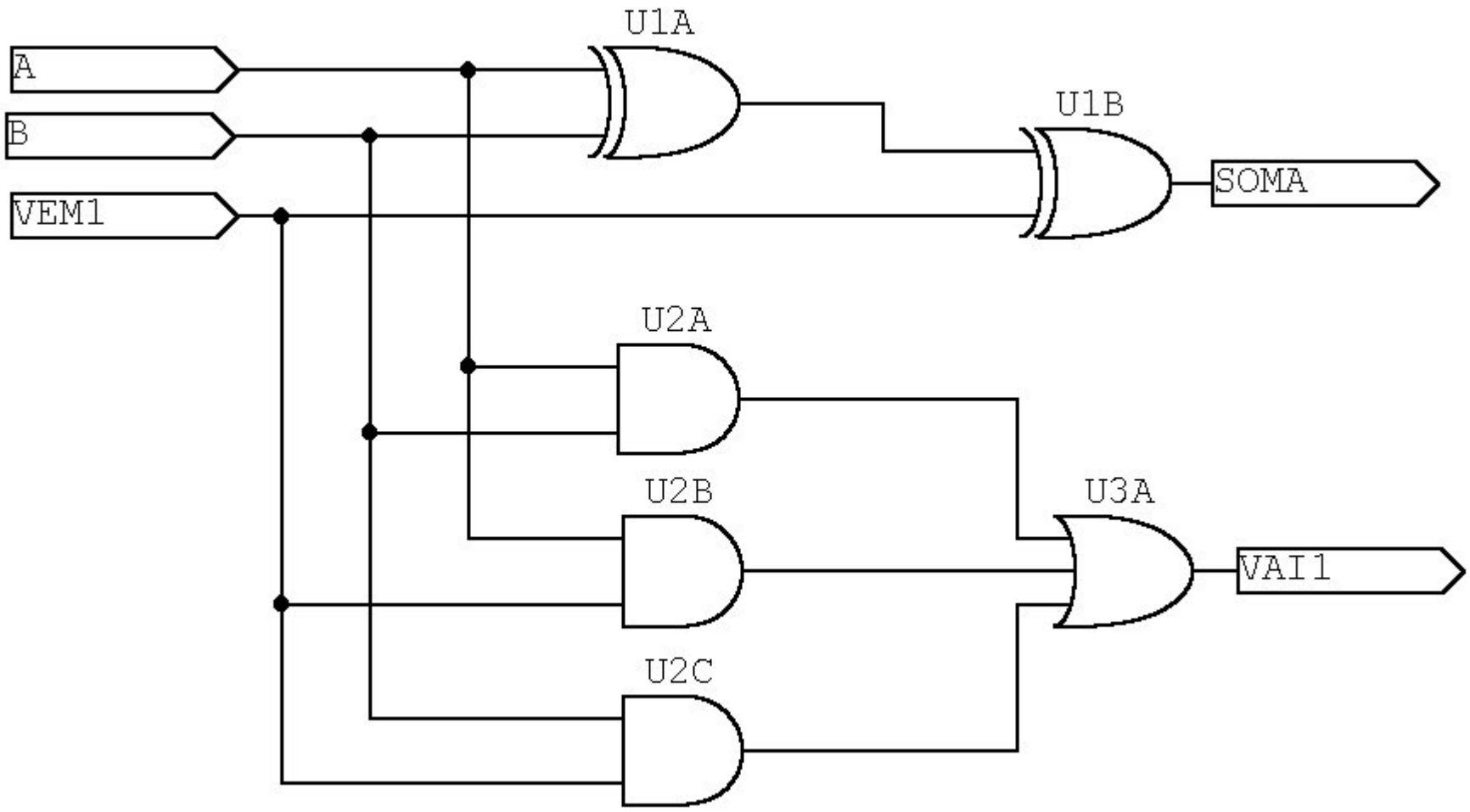
Tipos de Arquitetura

Existem diferentes modelos para declarar a lógica a ser sintetizada no PLD:

- ***Dataflow***
- ***Structural***
- ***Behavioral***
- ***Mix***

Tipos de Arquitetura: ***Dataflow***

- Ex: Somador Completo



Tipos de Arquitetura: *Dataflow*

– **As atribuições de valores ocorrem simultaneamente.**

```
entity SOMADOR is
  port (A, B, VEM1 : in bit; SOMA, VAI1 : out bit);
end entity SOMADOR;

architecture ASOMA  of  SOMADOR is
begin

SOMA <=  (A xor B) xor VEM1;
VAI1 <= (A and B) or (B and VEM1) or (A and VEM1);

end architecture;
```

Tipos de Arquitetura: *Dataflow*

```
architecture ARQ of E is

begin
        Y <= A xor C;
        Z <=  A and B;

end architecture;
```

Alteração do sinal 'A' faz com que as 2 atribuições sejam executadas paralelamente.

Alteração do sinal 'C' faz com apenas a 1ª. atribuição seja executada.

Tipos de Arquitetura: *Dataflow*

- Atribuição Condicional: **Implementar multiplexador 3 para 1**

```
entity Mux3to1 is
                port (  A, B, C, S0, S1: in bit;
                        Z        : out bit );
end entity Mux3to1;

architecture DFLOW  of Mux3to1 is
Begin

        Z <= A  when S0 = '0' and S1 = '0' else
             B  when S0 = '1' and S1 = '0' else
             C;

end architecture;
```

Tipos de Arquitetura: ***Dataflow***

- Atribuição Condicional

<optional_label>: <target> <= <value> when <condition> else
<value> when <condition> else
<value> when <condition> else
...
<value>;

Tipos de Arquitetura: ***Dataflow***

- Atribuição de Sinal Selecionado: **Realizar operação de or ou and entre as entradas A e B conforme sinal de seleção S.**

```
entity Mux2to1 is
                port (  A, B, S : in   bit;
                        Z       : out bit);
end entity Mux2to1;

architecture DFLOW  of Mux2to1 is
Begin
        with S select
                                Z <= A or    B  when '0' ,
                                     A and  B   when '1' ;
end architecture;
```

Tipos de Arquitetura: ***Dataflow***

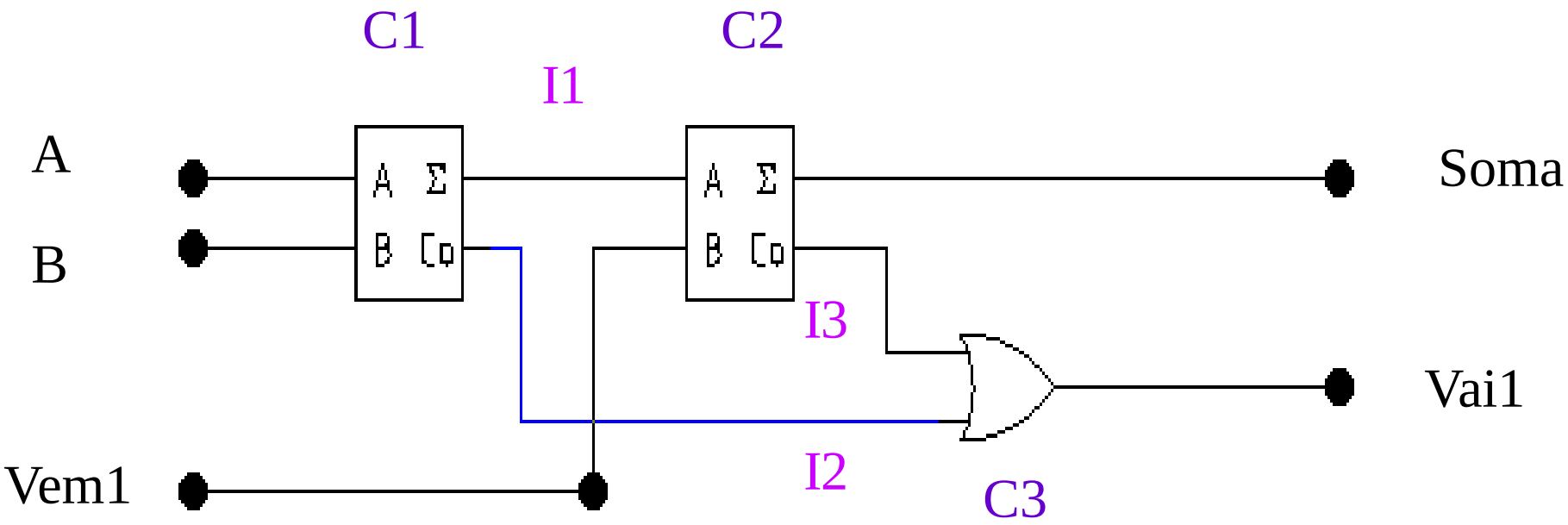
- Atribuição de Sinal Selecionado

<optional_label>: with <expression> select

<target> <= <value> when <choices>
<value> when <choices>
<value> when <choices>
...
<value> when others;

Tipos de Arquitetura: ***Structural***

- **Descreve circuito como componentes interconectados.**
- **Mesma tarefa do bdf na interligação de componentes.**



Tipos de Arquitetura: *Dataflow*

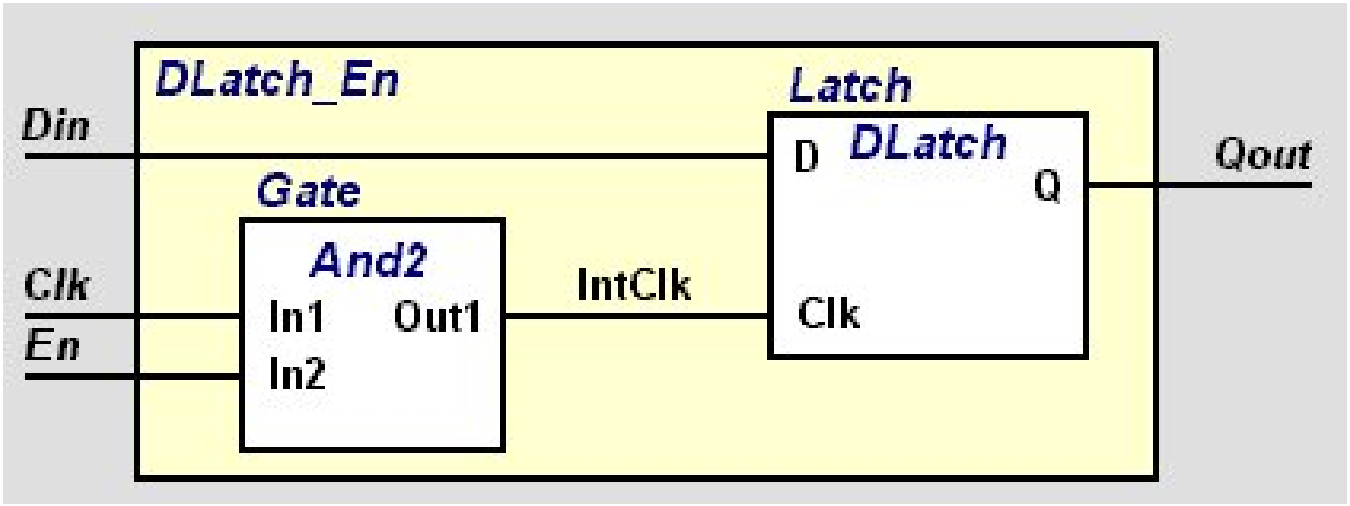
- Exercício:
  - **Criar uma porta OU com entidade denominada ORGATE, Entradas A e B; Saída denominada Z.**

```
entity ORGATE is
                port (
                        A, B    : in bit;
                        Z       : out bit
                        );
end entity ORGATE;

architecture DFLOW  of  ORGATE is
begin
                Z <= A or B;
end architecture;
```

Tipos de Arquitetura: *Structural*

```
entity SCOMPLETO  is
                port (  A, B, VEM1        : in bit; SOMA, VAI1        : out bit);
end entity  SCOMPLETO;

architecture ESTRT of SCOMPLETO is
  signal I1, I2, I3 : bit;
  component MEIOSOMADOR
                port(   A, B                    : in bit;  SOMA, VAI1       : out bit);
  end component;
  component ORGATE
                  port( A, B       :  in bit;              Z          :  out bit);
  end component;
Begin

  C1: MEIOSOMADOR port map(A,B,I1,I2);
  C2: MEIOSOMADOR port map(I1,VEM1, SOMA, I3);
  C3: ORGATE port map(I3,I2, VAI1);
end architecture;
```

Tipos de Arquitetura: ***Structural***




```
BEGIN
        C1 : mand2    PORT MAP (  in1     => clk,
                                  in2     => en,
                                  out1    => intclk);

        C2 : mlatch   PORT MAP (intclk,din,qout);
END;
```

Associação Declarada

Associação por posição

Tipos de Arquitetura: *Structural*

```
ENTITY root IS
        PORT(           din, en, clk    : IN    bit;
                        qout    : OUT   bit );
END ENTITY root;

ARCHITECTURE struct OF root IS

        COMPONENT mand2   --  componente criado em um outro projeto
                PORT (          in1,in2  : in bit;
                                out1     : out bit);
        END COMPONENT;

        COMPONENT mlatch --  componente criado em um outro projeto
                PORT (          clkd, d  : in bit;
                                q        : out bit);
        END COMPONENT;

        signal intclk: bit; --  incluir  agora o código do slide anterior
```

Tipos de Arquitetura: *Behavioral*

Relação entre sinais de entrada e saída do circuito é descrita como conjunto de comandos seqüenciais. O início e fim destes comandos seqüenciais são indicados pela diretiva processo.

```
entity mux41 is
    port   (A,B,C,D : in bit; S  : in bit_vector(1 downto 0) ; Z : out bit);
end entity mux41;
architecture firstif  of mux41 is
    begin
        process (A, B, C, D, S)
        begin
                if      (S = "00")          then          Z <= A;
                elsif  (S = "01")          then          Z <= B;
                elsif  (S = "10")          then          Z <= C;
                else                                     Z <= D;
                end if;
        end process;
end architecture;
```

Tipos de Arquitetura: ***Behavioral***

Diretiva *Process*

[nome:] PROCESS [(lista de sensibilidade)]

[parte declarativa]

BEGIN

corpo do procedimento

executado seqüencialmente

END PROCESS [nome];

A lista de sensibilidade informa os sinais cuja alteração de nível lógico dispara a execução do processo.

Tipos de Arquitetura: *Behavioral*

```
architecture BH of PRCEX is
begin
  P1 : process(A,B)
      begin
          VAI1<= A and B;
      end process P1;
  P2 : process(A,B)
      begin
          SOMA  <=  A xor B;
      end process P2;
end architecture;
```

Alteração do sinal 'A' faz com que os 2 processos sejam executados paralelamente

Tipos de Arquitetura: *Dataflow* Equivalente

```
architecture BH of PRCEX is
begin
    VAI1 <= A and B;
     SOMA   <=  A xor B;
end architecture;
```

Tipos de Arquitetura: ***Behavioral*** => ***IF-THEN***

```
IF <condition1> THEN
        {sequence of statement(s)}
ELSIF <condition2> THEN
        {sequence of statement(s)}
        .
        .
ELSE
        {sequence of statement(s)}
END IF;
```

Exercício

Modificar o exemplo anterior para que o multiplexador passe a ter cinco sinais de entrada. Caso a entrada de seleção apresente um valor maior que 4, a saída do mesmo deve ficar em '0' .

Exercício

Modificar o exemplo anterior para que o multiplexador passe a ter cinco sinais de entrada. A saída do mesmo deve ficar em estado de alta impedância caso a entrada de seleção apresente um valor não permitido.

Obs: BIT => **'0'** e **'1' ;** necessário mudar o tipo dos sinais de entrada para STD_LOGIC.

Tipos de Dados VHDL

- **C → Char, Integer, Float, Double**

  Ex. ➔ char EXEMPLO;

- **VHDL → Bit ('0'** ou **'1') ;**

  Ex. » signal CLR : bit;

  Std_logic;

  Ex. » signal CLR : std_logic;

Tipos de Dados VHDL

```
Type Std_logic is ( -- ieee library

        'U',        -- Unitialized
        'X'         -- Forcing Unknow,
        '0',        -- Forcing 0
        '1',        -- Forcing 1
        'Z' ,       -- High Impedance
        'W',        -- Weak Unknow
        'L',        -- Weak 0
        'H'         -- Weak 1
        '-');       -- Do not care
```

'U', 'X', '-','W', 'L', 'H' → Não utilizados em síntese

Bibliotecas

- **Standard** (Bit, Boolean, Integer, Real, Time, etc);
- **IEEE**
  - » **Std_logic_1164** (Std_logic Types e sua funções)
  - » **Std_logic_arith** (Funções Aritméticas)
  - » **Std_logic_signed** (Funções Aritméticas com sinal)
  - » **Std_logic_unsigned** (Funções Aritméticas sem sinal)
- **Work** (Biblioteca do programador. Diretório de trabalho)

Tipos de Arquitetura: *Behavioral* => *IF-THEN*

```
library ieee;
use ieee.std_logic_1164.all;
entity mux3s is
 port (A,B,C,D,E : in std_logic; S  : in std_logic_vector(2 downto 0) ;
                  Z : out std_logic);
end entity mux3s;
architecture firstif  of mux3s is
          begin
                    process (A, B, C, D, E, S)
                    begin
                              if      (S = "000")  then          Z <= A;
                              elsif  (S = "001") then          Z <= B;
                              elsif  (S = "010") then          Z <= C;
                              elsif  (S = "100") then          Z <= D;
                              elsif  (S = "101") then          Z <= E;
                              else                             Z <='Z';
                              end if;
                    end process;
end architecture;
```

Diretivas *Library* **e** *Use*

- Library: **nome simbólico de diretório contendo arquivos de outro projeto previamente compilado. Associação realizada pelo compilador.**

  LIBRARY <Name> ;

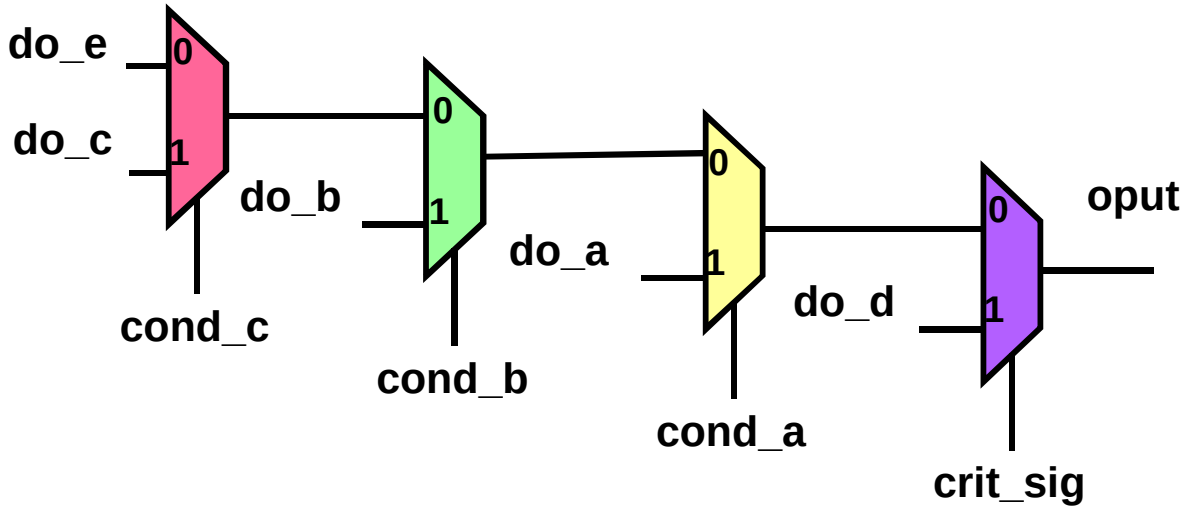
- Use: **permite o compilador localizar componentes específicos da biblioteca (diretório) para compilação**.

  Use Lib_name.Pack_name.Object;

Tipos de Arquitetura: *Behavioral* => *IF-THEN*

- Implementação de multiplexador como decodificador de prioridade (Xilinx):
  - Atraso aumenta com o laço elsif
  - Utilizar entradas mais críticas primeiro

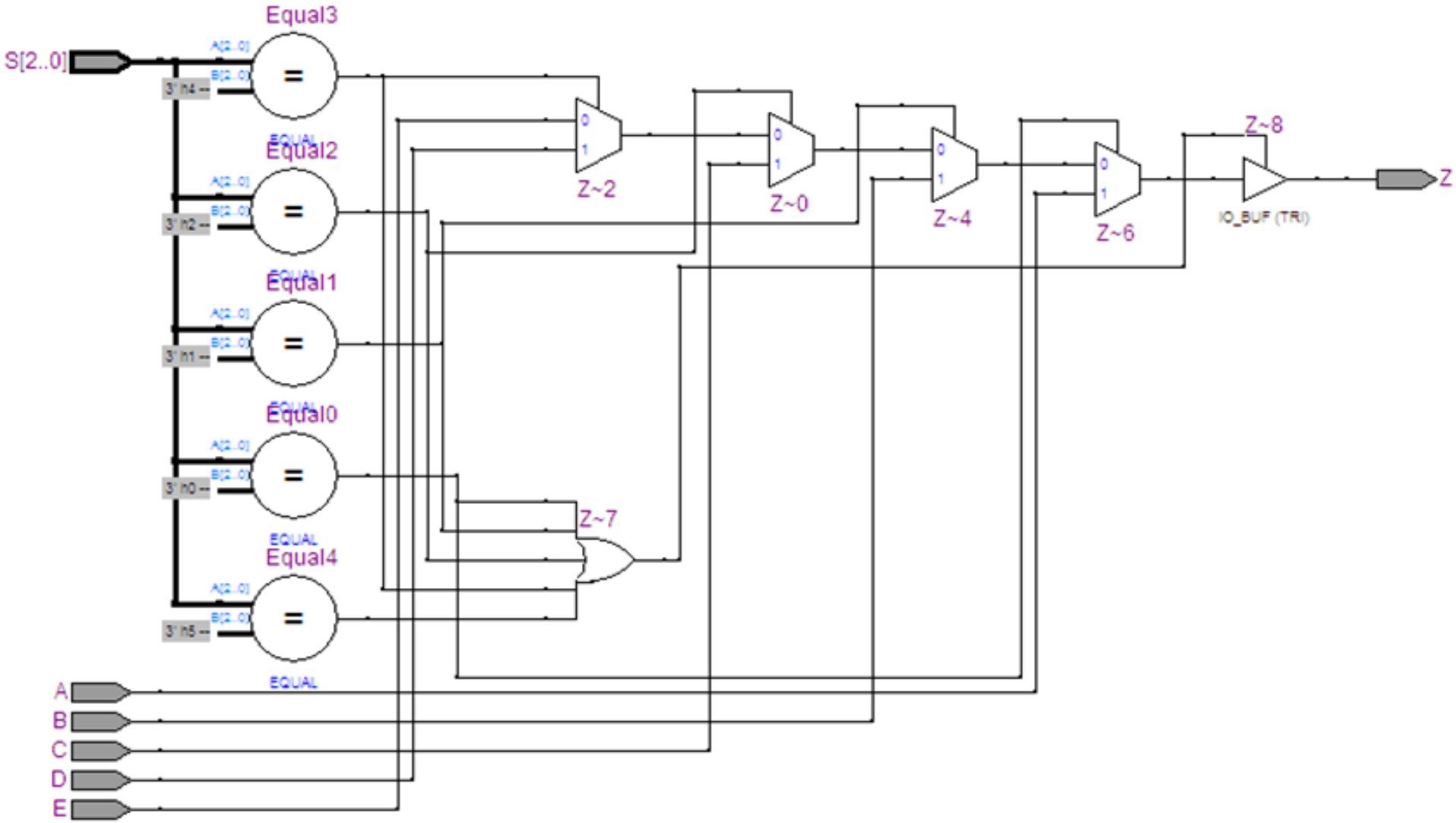
```
IF (crit_sig) THEN oput <= do_d ;

 ELSIF cond_a THEN oput <= do_a;

  ELSIF cond_b THEN oput <= do_b;

  ELSIF cond_c THEN oput <= do_c;

  ELSE oput <= do_e;

END IF;
```



Melhor utilizar Case para mux (combinacional)

Exemplo Anterior

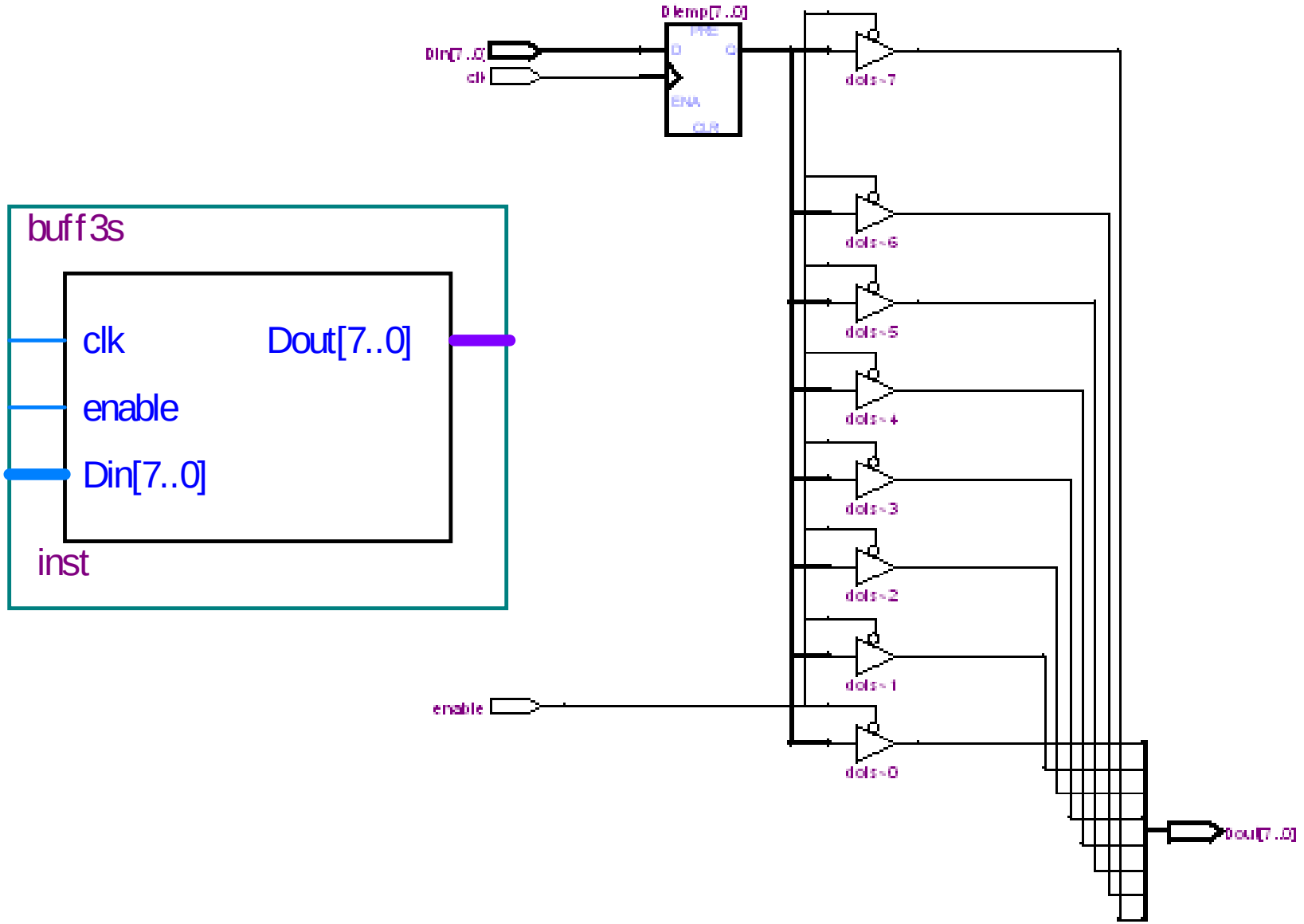
Tools => Netlit Viewer => RTL Viewer



Exercício

Projete um buffer (Tipo 74373) com oito entradas de sinais de dados, oito saídas de sinais de dados, uma entrada de sinal de clock e uma entrada de habilitação. Os dados presentes no barramento de entrada do buffer, quando da subida do sinal de clock, devem estar presentes na saída se a entrada de habilitação estiver em nível lógico baixo. Caso a entrada de habilitação esteja em nível lógico alto, a saída deve estar em estado de alta impedância.

Circuito Equivalente



Tipos de Arquitetura: *Behavioral => IF-THEN*

```
library ieee;
Use ieee.std_logic_1164.all;

      entity buff3s is
      port       (                    clk, enable :  in std_logic;
                                      Din  :  in std_logic_vector(7 downto 0);
                                      Dout :  out std_logic_vector(7 downto 0));
end entity buff3s;

architecture buffif  of buff3s is
signal Dtemp : std_logic_vector(7 downto 0);
      begin
            um: process (clk)
                  begin
                        if     (clk'event and clk  = '1') then Dtemp <= Din;
                        end if;
            end process um;

            dois: process (enable,Dtemp)
                  begin
                        if (enable = '0')  then Dout <= Dtemp;
                        else Dout <= (others=>'Z');
                        end if;
            end process dois;
end architecture;
```

Atributo 'Event

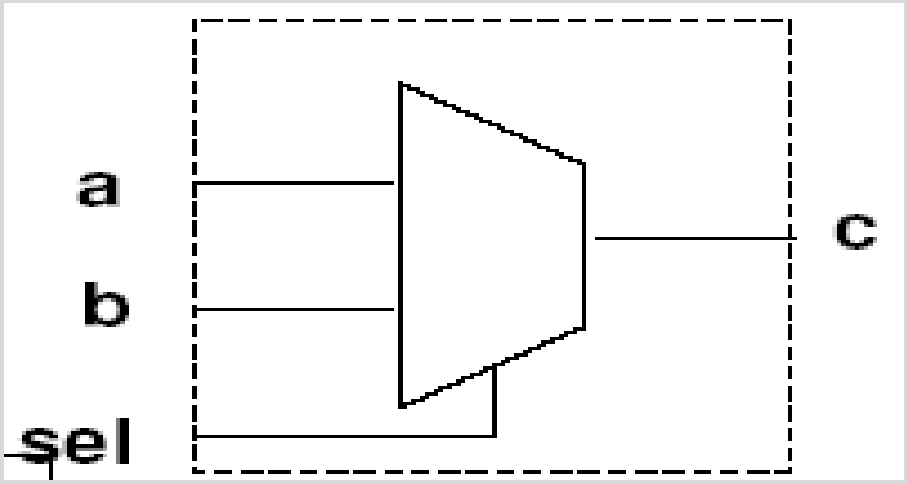
- Atributo de sinal que retorna um valor verdadeiro ('1') quando há uma mudança de valor no sinal ao qual se encontra associado.

std_logic_1164 package

SIGNAL'event and SIGNAL = '1'; rising_edge (SIGNAL)
SIGNAL'event and SIGNAL = '0'; falling_edge (SIGNAL)

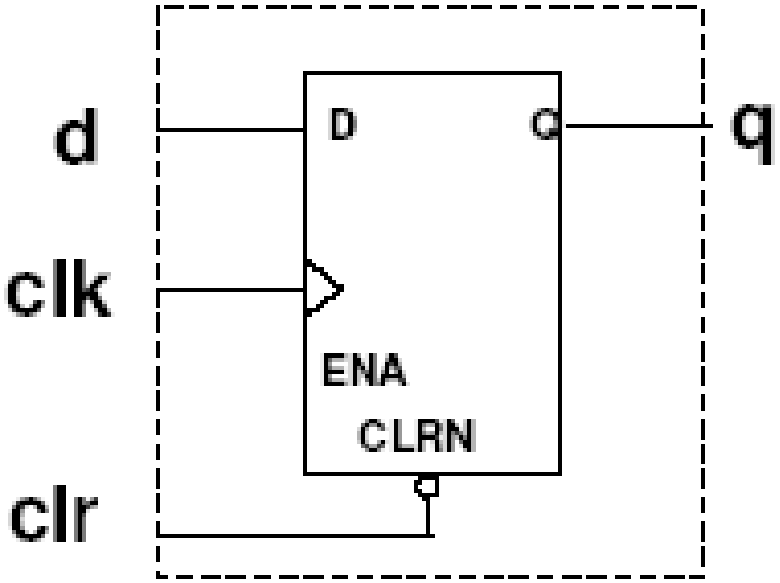
OBS:

Processo Combinacional



X

Sequencial




```
PROCESS(a, b, sel)
```

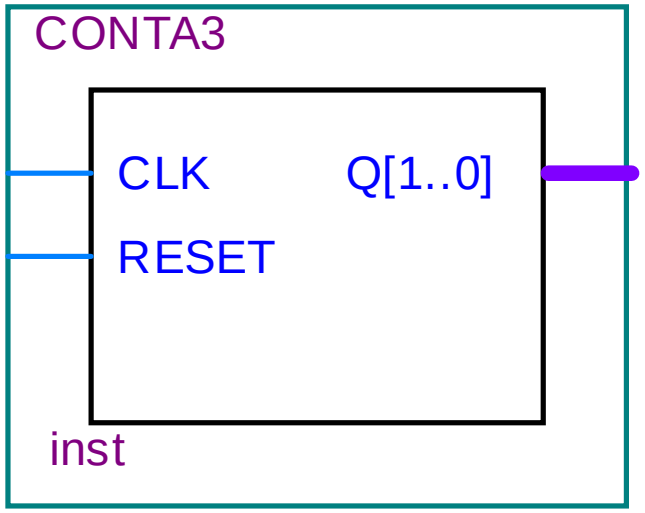
Lista de sensibilidade inclui todos os sinais

```
PROCESS(clr, clk)
```

Lista de sensibilidade inclui *clock* e sinais de controle

Exercício

Projete um contador de 0 a 3 que possua entrada para sinal de *reset* que zere a contagem.



Exercício

```
Library IEEE;
USE IEEE.Std_Logic_1164.all;
USE IEEE.std_logic_unsigned.all;

entity CONTA3 is
port ( CLK, RESET : in STD_LOGIC;
          Q     : out STD_LOGIC_VECTOR (1 DOWNTO 0 ));
end entity CONTA3;

architecture AR3 of CONTA3 is
begin
  process (CLK,RESET)
   variable CONTA : STD_LOGIC_VECTOR (1 DOWNTO 0 );
  begin
     IF (RESET = '0')                THEN      CONTA := "00";
     ELSIF CLK'event and CLK = '1'  then  CONTA := CONTA +1;
    END IF;
      Q <= CONTA;
  end process;

end architecture;
```

Operator Overload: Múltipla definição para diferentes sinais de entrada

```
package std_logic_unsigned is

  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC_VECTOR; R: INTEGER) return STD_LOGIC_VECTOR;
  function "+"(L: INTEGER; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return STD_LOGIC_VECTOR;
  function "+"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;

  function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR; R: INTEGER) return STD_LOGIC_VECTOR;
  function "-"(L: INTEGER; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC_VECTOR; R: STD_LOGIC) return STD_LOGIC_VECTOR;
  function "-"(L: STD_LOGIC; R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
```

Classes de Dados

- **C → constantes e variáveis.**

- **VHDL → constantes, variáveis, sinais.**

  - **Ex.** `constant MAX_SIZE : integer := 10;`
    `variable COUNT : integer range 0 to 8;`
    `signal INT5 : bit;`

Classes de Dados

- **Constant**
  - # **declaradas em Package, Entity, Architecture e Process**

- **Variable**
  - # **declaradas em Process e Function.**

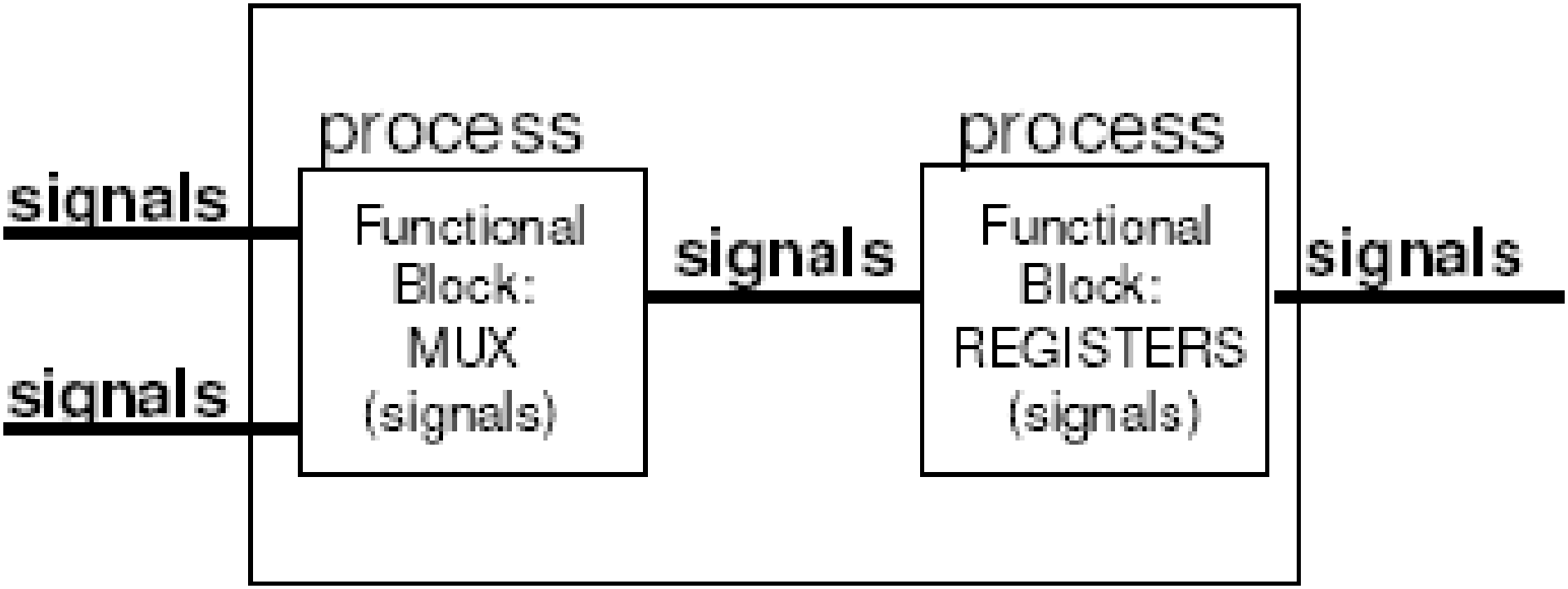
- **Signal**
  - # **declaradas em Package, Entity ou Architecture**

Variable: **realiza armazenagem local;**
conhecida apenas dentro do processo;
a atribuição de valores ocorre imediatamente;
retém valores atribuídos entre execuções do processo;

Signal : **corresponde a interconexão entre circuitos/processos**;
assume último valor atribuído antes de sair do processo;
compartilha valores atribuídos entre processos.

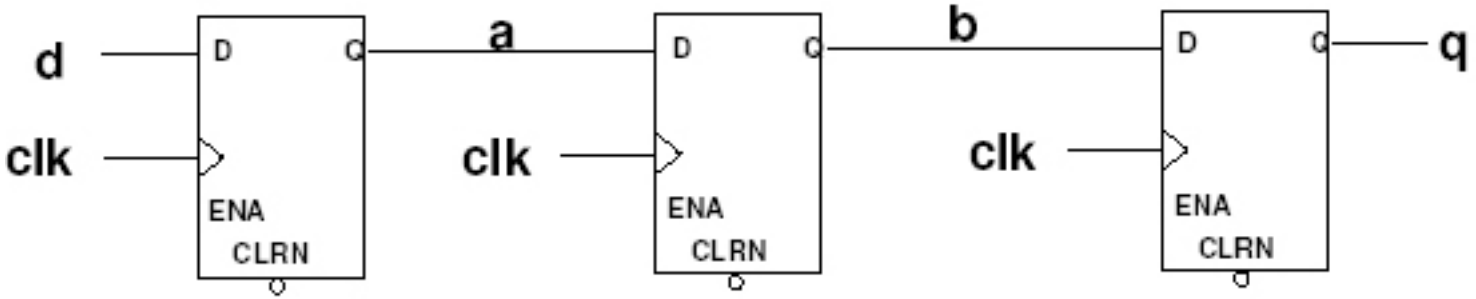
Sinal => Síntese de Hardware

Corresponde a conexão física entre circuitos para arquitetura estrutural.



Sinal => Síntese de Hardware

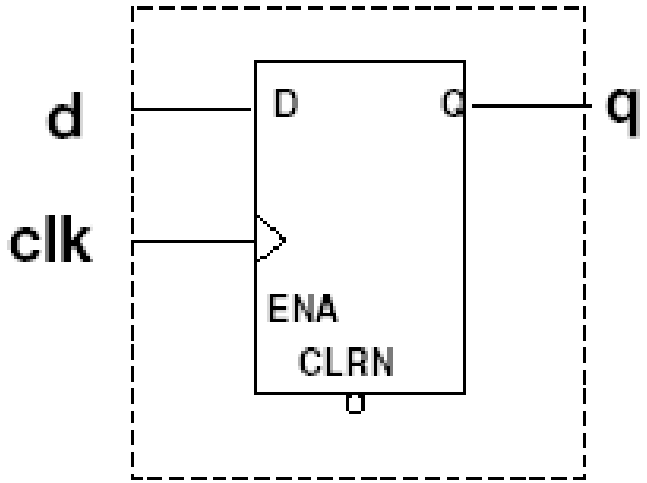
```
PROCESS (clk)
BEGIN
        IF rising_edge(clk) THEN
                a <= d;
                b <= a;
                q <= b;
        END IF;
END PROCESS;
```



Variável => Armazenamento Local

```
PROCESS (clk)
VARIABLE a, b : STD_LOGIC;
BEGIN
        IF rising_edge(clk) THEN
                a := d;
                b := a;
                q <= b;
        END IF;
END PROCESS;
```

Uso de variável



Tipos de Arquitetura: *Behavioral* => *CASE*

```
entity mux31 is
    port   (A,B,C: in bit; S  : in bit_vector(3 downto 0) ; Z : out bit);
end entity mux31;
architecture firstcase  of mux31 is
    begin
    process (A,B,C,S)
          begin
         case S is
                    when X"0"              =>    Z <= B;
                    when X"5"              =>    Z <= C;
                    when X"7" | X"9"       =>    Z <= A;  -- valor 7 ou 9
                    when others            =>    null;       --  faz nada
          end case;
          end process;
end architecture;
```

Tipos de Arquitetura: *Behavioral* => *CASE*

```
CASE {expression} IS
        WHEN <condition1> =>
                {sequence of statements}
        WHEN <condition2> =>
                {sequence of statements}
                .
                .
        WHEN OTHERS =>   -- (optional)
                {sequence of statements}
        END CASE;
```

Exercício

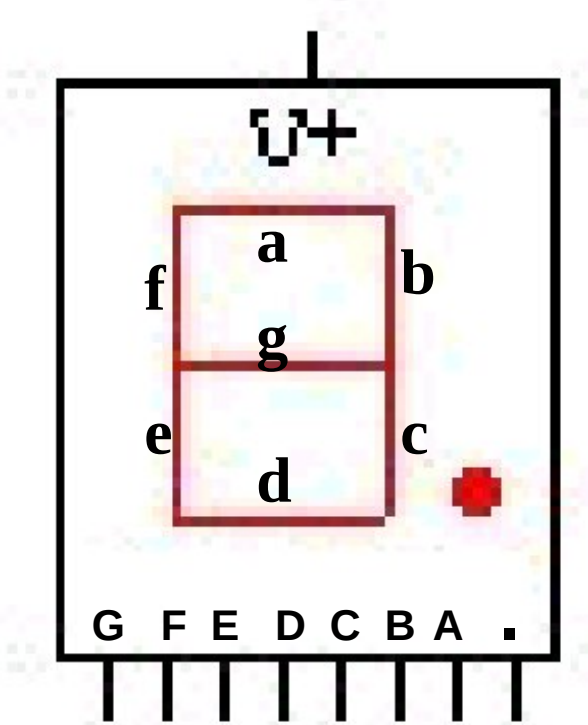
Modificar exemplo anterior para que os valores de seleção não definidos gerem estado de alta impedância.

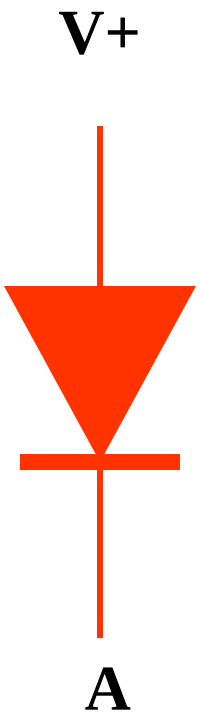
Tipos de Arquitetura: *Behavioral* => *CASE*

```
library ieee;
Use ieee.std_logic_1164.all;

entity muxz is
        port      (A,B,C: in std_logic; S  : in std_logic_vector(3 downto 0) ;
                                        Z : out std_logic);
end entity muxz;

architecture firstcase  of muxz is
        begin
         process (A,B,C,S)
          begin
            case S is
                when "0000"              =>    Z <= B;
                when "0101"              =>    Z <= C;
                when "0111" | "1001"     =>    Z <= A;    -- valor X"7" | X"9"
                when others              =>    Z <='Z';    --  tri-state
             end case;
        end process;
end architecture;
```

Exercício

Projete um conversor HEX - 7 segmentos (ânodo comum) utilizando a declaração case para tal tarefa.



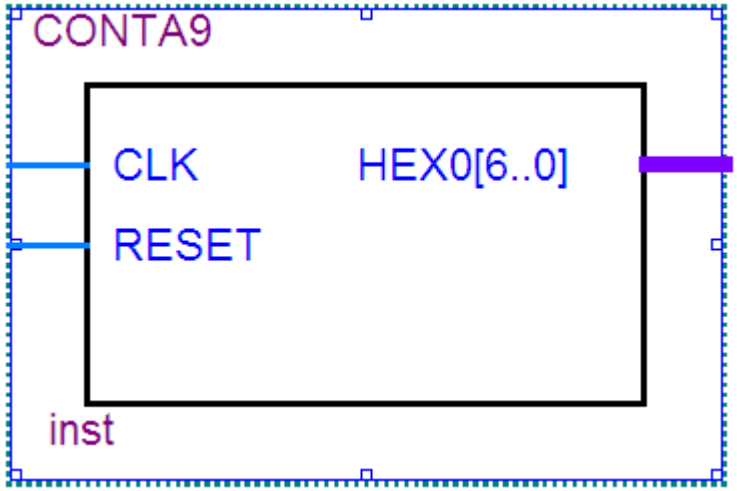


Tipos de Arquitetura: *Behavioral* => *CASE*

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 ENTITY seteseg IS
   PORT   (DSPY        : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           HEX0        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END ENTITY seteseg;
 ARCHITECTURE mostra OF seteseg IS
 BEGIN
 PROCESS (DSPY)
   BEGIN
           CASE DSPY IS                                  --    GFEDCBA
                   WHEN "0001"          => HEX0   <= "1111001";
                   WHEN "0010"          => HEX0   <= "0100100";
                   WHEN "0011"          => HEX0   <= "0110000";
                   WHEN "0100"          => HEX0   <= "0011001";
                   WHEN "0101"          => HEX0   <= "0010010";
                   WHEN "0110"          => HEX0   <= "0000010";
                   WHEN "0111"          => HEX0   <= "1111000";
                   WHEN "1000"          => HEX0   <= "0000000";
                   WHEN "1001"          => HEX0   <= "0010000";
                   WHEN "1010"          => HEX0   <= "0001000";
                   WHEN "1011"          => HEX0   <= "0000011";
                   WHEN "1100"          => HEX0   <= "1000110";
                   WHEN "1101"          => HEX0   <= "0100001";
                   WHEN "1110"          => HEX0   <= "0000110";
                   WHEN "1111"          => HEX0   <= "0001110";
                   WHEN OTHERS          => HEX0   <= "1000000";
           END CASE;
  END PROCESS;
 END ARCHITECTURE;
```

Exercício

Projete um contador de 0 a 9 que mostre o resultado no display de sete segmentos. Basear-se nos exemplos anteriores (Conta3 e Seteseg).



Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]

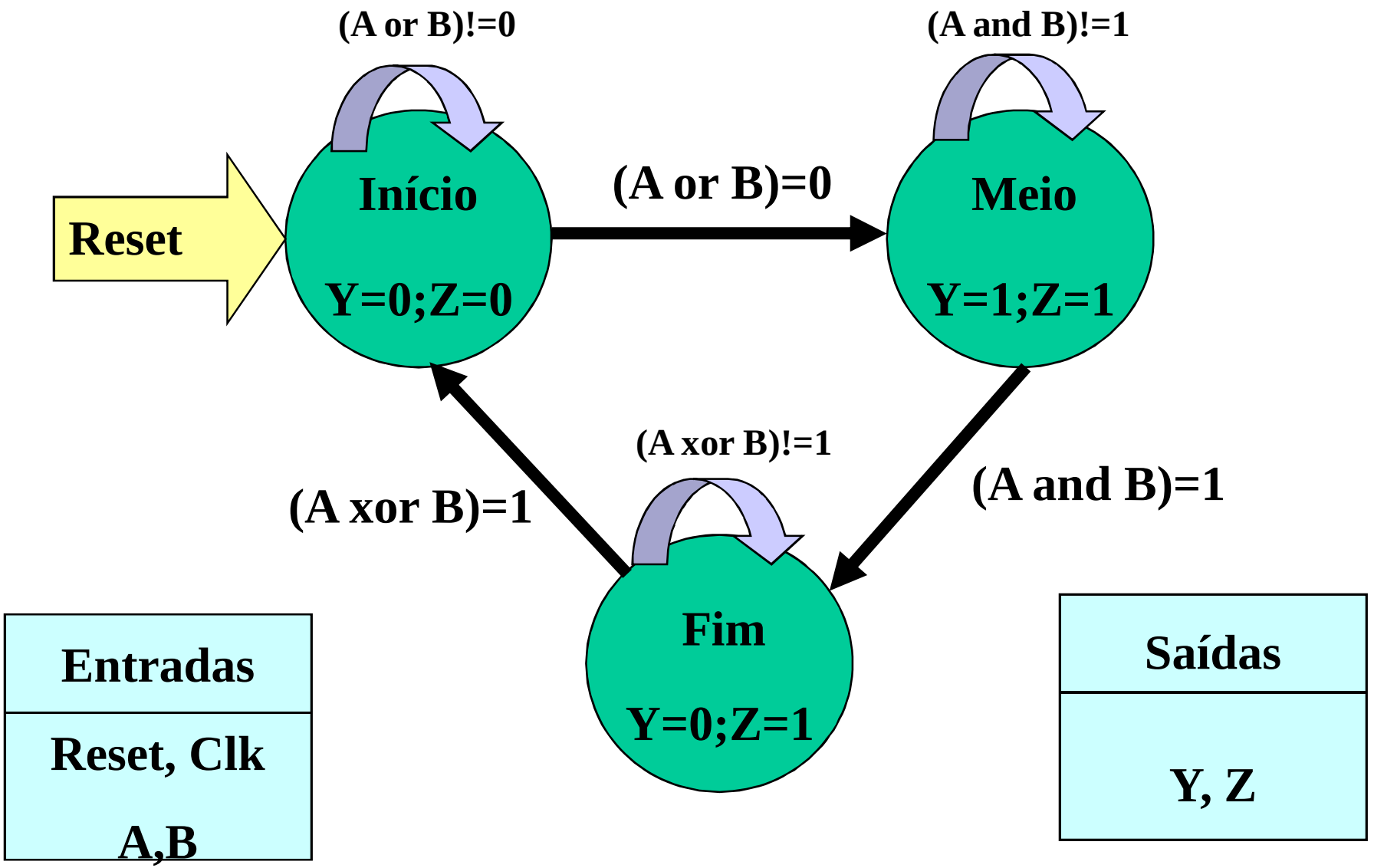
Signal Name	FPGA Pin No.
HEX0[0]	PIN_AF10
HEX0[1]	PIN_AB12
HEX0[2]	PIN_AC12
HEX0[3]	PIN_AD11
HEX0[4]	PIN_AE11
HEX0[5]	PIN_V14
HEX0[6]	PIN_V13

```
Library IEEE;
USE IEEE.Std_Logic_1164.all;
USE IEEE.std_logic_unsigned.all;

entity CONTA9 is
port ( CLK  , RESET : in bit;
                HEX0        : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
end entity CONTA9;
```

```
architecture AR3 of CONTA9 is
        signal DSPY        :   integer range  0 to 9;
begin
  process (CLK,RESET)
   variable CONTA9 : integer range  0 to 9;
  begin
        IF (RESET = '0')   THEN     CONTA9 := 0;
        ELSIF CLK'event and CLK= '1'  then
        if (CONTA9 < 9) then CONTA9 := CONTA9+1;
                          else CONTA9 := 0;
        end if;
   END IF;
         DSPY <= CONTA9;
  end process;
```

```
  PROCESS (DSPY)
   BEGIN
        CASE DSPY IS                                    --   GFEDCBA
                WHEN 1          => HEX0         <= "1111001";
                WHEN 2          => HEX0         <= "0100100";
                WHEN 3          => HEX0         <= "0110000";
                WHEN 4          => HEX0         <= "0011001";
                WHEN 5          => HEX0         <= "0010010";
                WHEN 6          => HEX0         <= "0000010";
                WHEN 7          => HEX0         <= "1111000";
                WHEN 8          => HEX0         <= "0000000";
                WHEN 9          => HEX0         <= "0010000";
                WHEN OTHERS     => HEX0         <= "1000000";
        END CASE;
  END PROCESS;
end architecture;
```

Exercício – Máquina de Estado



Tipos de Arquitetura: *Behavioral / DataFlow*

```
ENTITY mealy IS
   PORT(      reset, clk, A, B      : IN  BIT;
              Y, Z:   OUT BIT );

END ENTITY mealy;


ARCHITECTURE state_machine OF  mealy IS
   TYPE tipo_estado  IS (inicio, meio, fim);
   SIGNAL estado, proximo_estado: tipo_estado;
BEGIN
  comeco: PROCESS (clk, reset)
              BEGIN
               IF reset= '0' THEN    estado <= inicio;
              ELSIF clk'EVENT AND clk='1' THEN estado <= proximo_estado ;
              END IF;
  END PROCESS comeco;
```

Tipos de Arquitetura – *Behavioral / DataFlow*

```
maquina: process (A,B,estado)
  begin
   proximo_estado  <=   estado;
   case estado is
   when inicio => if (A or B)  = '0' then   proximo_estado  <= meio;
                  end if ;
   when meio =>   if (A and B ) = '1'  then proximo_estado  <= fim;
                   end if ;
   when fim =>    if (A xor B) = '1'  then proximo_estado  <= inicio;
                  end if ;
   end case ;
end process maquina;
    Y <='1'  when estado= meio else '0' ; -- atribuição condicional concomitante
    Z <='1'  when estado=meio or estado=fim else '0' ;
END  ARCHITECTURE;
```

Alguns Tipos de dados VHDL

- **ESCALAR: permite o uso de operadores relacionais**. $1^0 < 2^0$.

1 - Integer: Pré-definido pela linguagem para a faixa de pelo menos –(2 ^{31} –1) to 2 ^{31} –1. Sub-tipo:

type I_BYTE is range 0 to 256; **-- criando tipo próprio**

2 - Enumeration: caracteres ordenados de forma crescente pelo programador.

type BIT is (**'0'** , **'1'**); **-- pré-definido. '0' < '1'**

type BOOLEAN is (FALSE, TRUE); **--case insensitive**

Alguns Tipos de dados VHDL :

- **ARRAY: coleção de elementos de mesmo tipo.**

Ex. » Type VETOR is array (0 to 256) of BIT;

Pré-definido:

String ➔ cadeia de caracteres

Ex. » variable palavra: string(1 to 6);
palavra := "janela";

Exercício

Acrescentar à máquina de estado acima, display de sete segmentos para realizar a indicação dos estados (Início = 0; Meio = 1; Fim = 2). Utilizar *push buttom* como entrada de *clock*, *switches* para as entradas A e B, leds para as saídas Y e Z.

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]

Signal Name	FPGA Pin No.	Description
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]

Exercício

Criar um multiplexador (8 para 1), utilizando 3 entradas de seleção.

Tipos de Arquitetura – ***Behavioral***

```
library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;

entity muxl is
        port (A : in std_logic_vector(7 downto 0); Sel  : in unsigned (2
                                                                downto 0);
        Z : out std_logic);
        constant N : integer :=7;
end entity muxl;

architecture firstloop  of muxl is
begin
            process (A, SEL)
             variable I :integer range 0 to N;
             begin
                        I := conv_integer(SEL);
                        Z <= A(I);
            end process;
end architecture;
```

Std_Logic_Arith Package

- Recursos
  - » **Converte dados para/de INTEGER.**
  - » **Operações aritméticas em ARRAYS.**
  - » **Comparações de valores de ARRAYS**.
- Sintaxe

```
LIBRARY IEEE;
USE    IEEE.Std_Logic_Arith.ALL
```

Biblioteca Aritmética

Definição de tipos

type UNSIGNED is array (natural range <>) of std_logic; --

type SIGNED is array (natural range <>) of std_logic;

-- interpretado como complemento de 2

-- Msbit é bit de sinal da Array.

Definição de operadores

+ , - , * e outros

Funções de Conversão

function CONV_INTEGER(ARG: UNSIGNED) return INTEGER;

function CONV_STD_LOGIC_VECTOR(ARG: INTEGER; SIZE: INTEGER) return STD_LOGIC_VECTOR;

Controle de Fluxo Seqüencial :

- **LOOP** (loop infinito)

```
loop
        -- faz alguma coisa;
end loop;
```

- **FOR...IN...LOOP/END LOOP** (loop iterativo)

```
for VARIAVEL in 0 to 6 loop
        -- faz alguma coisa;
end loop;
```

- **While ... LOOP** (loop condicional)

```
while VARIAVEL < 0 loop
        -- faz alguma coisa;
end loop;
```

EXIT ➔ salta para fora do loop mais interno.

NEXT ➔ força nova iteração saltando comandos seguintes.

Exercício

Criar um componente utilizando FOR LOOP/END LOOP para determinar o nro de bits iguais a um de um vetor de 8 bits.

Tipos de Arquitetura – *Behavioral* => *FOR*

```
LIBRARY IEEE;
        USE IEEE.std_logic_1164.all;
        USE IEEE.std_logic_unsigned.all;
        USE IEEE.std_logic_arith.all;

ENTITY bc IS
PORT (  invec: in std_logic_vector(7 downto 0);
        outvec: out std_logic_vector(3 downto 0));
END ENTITY bc;
ARCHITECTURE rtl OF bc IS
BEGIN
PROCESS(invec)
        VARIABLE count: std_logic_vector(3 downto 0);
BEGIN
        Count:=(others=>'0');
                FOR i IN invec'right TO Invec'left LOOP
                        IF (invec(i)/='0') THEN  count:=count+1;
                        END IF;
                END LOOP;
        outvec<=count;
END PROCESS;
END ARCHITECTURE rtl;
```

Array Attributes ➡ retorna valores relativos a abrangência da array.

- left
- right
- high
- low
- length
- range
- reverse_range

Exemplo: variable MY_VECTOR : BIT_VECTOR(5 downto -5);

Attribute Expression	Value
MY_VECTOR'left	5
MY_VECTOR'right	– 5
MY_VECTOR'high	5
MY_VECTOR'low	– 5
MY_VECTOR'length	11
MY_VECTOR'range	(5 downto –5)
MY_VECTOR'reverse_range	(–5 to 5)

Exercício

Criar um Codificador de Prioridade utilizando LOOP/END LOOP. Este componente deve receber um nível lógico alto em um pino de entrada (de um total de oito) e gerar código BCD em 3 pinos de saída correspondente a entrada mais significativa que esteja nível lógico alto.

Tipos de Arquitetura – *Behavioral* => *LOOP*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity nextex is
        port    ( INP  : in std_logic_vector(7 downto 0);   Z : out std_logic_vector(2 downto 0));
end entity nextex;
architecture firstloop  of nextex is
subtype mine is integer range 0 to 7;
begin
        process(INP)
        variable I: mine;
        begin
                I:=7;
        first:  loop
                        exit when   INP(I) /= '0' or I = 0;
                        I:=I-1;
                end loop;
                Z <= conv_std_logic_vector(I,3);
        end process;
end architecture;
```

Tipos de Arquitetura – *Behavioral* => *WHILE*

```
library ieee;
Use ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
entity nextex is
        port        (INP : in std_logic_vector(7 downto 0);  Z : out std_logic_vector(2 downto 0));
end entity nextex;
architecture firstloop  of nextex is
subtype mine is integer range 0 to 7;
begin
        process (sel)
        variable I: mine;
        begin
                I:=7;
                while I > 0 loop
                        exit when INP(I)/= '0';
                        I:=I-1;
                 end loop;
                Z <= conv_std_logic_vector(I,3);
        end process;
end architecture firstloop;
```

Diretivas *Package* e *Use*

- **Package** permite o compartilhamento de definições/funções úteis em diferentes projetos.
- **Package Body** descreve o funcionamento de estruturas compartilhadas pelo package.
- **Use** inclui as definições de Package na compilação.

Diretivas *Package* e *Use*

```
PACKAGE definicoes IS

-- Subtypes  ⇨  subconjunto de um tipo previamente definido

    SUBTYPE  Vetordebit        IS BIT_VECTOR;
    SUBTYPE  BitVector2        IS BIT_VECTOR( 1 DOWNTO 0);
    SUBTYPE  BitVector3        IS BIT_VECTOR( 2 DOWNTO 0);
    SUBTYPE  ATE3      IS INTEGER RANGE 0 TO 3;
    TYPE       MagnComp        IS (G,I,P);

 END PACKAGE definicoes;
```

Diretivas *Package* e *Use*

```
USE work.definicoes.all;

ENTITY comparador IS
   PORT(     x,y   : IN  ATE3;                -- inputs
                z   : OUT MagnComp);   -- output
  END ENTITY comparador;

  ARCHITECTURE behave OF comparador IS
  BEGIN
     PROCESS (x,y)
           BEGIN
                  IF        (x > y)     THEN          z <= G;
                            ELSIF (x = y) THEN        z <= I;
                            ELSE                      z <= P;
                  END IF;
    END PROCESS;
  END ARCHITECTURE;
```

Uso de Funções

```
package EX_FUNCAO is
function INV_VETOR(DATA: in BIT_VECTOR) return BIT_VECTOR;
constant N : integer :=3;
end package EX_FUNCAO;
package body EX_FUNCAO is
function INV_VETOR(DATA: in BIT_VECTOR) return BIT_VECTOR is
        VARIABLE resultado:BIT_VECTOR(DATA'reverse_range);
        begin
                for I in DATA'range loop
                        resultado(I):=DATA(I);
                end loop;
                return resultado;
        end INV_VETOR;
end package body EX_FUNCAO;
```

Uso de Funções

```
use work.EX_FUNCAO.all;


entity exfunc is
        port(   B: in BIT_VECTOR(N-1 downto 0);
                Z: out BIT_VECTOR(N-1 downto 0));
end entity exfunc;


architecture tstfunc of exfunc is
        begin
                Z <= INV_VETOR(B);
end architecture;
```

Exercício

Criar (utilizando função) um componente que aceita 3 vetores de 3 bits (um por subida de clock) e ao receber o quarto pulso de clock, coloca em sua saída o maior vetor entre os 3 recebidos.

Uso de Funções

```
library ieee;
Use ieee.std_logic_1164.all;
package PKEXFUN is
constant N: integer:=2;      constant M: integer:=2;      subtype vetor is std_logic_vector(N downto 0);
type matrix is array(0 to M)  of vetor;                   function MR_VETOR(DATA: in matrix) return vetor;
end package PKEXFUN;
package body PKEXFUN is
function MR_VETOR(DATA: in matrix) return vetor is
        VARIABLE resultado : vetor;
        begin
                resultado := (others=> '0');
                for I in DATA'range loop
                        if DATA(I) > resultado  then
                                resultado := DATA(I);
                        end if;
                end loop;
                return resultado;
        end MR_VETOR;
end package body PKEXFUN;
```

Uso de Funções

```
use work.pkexfun.all;
entity tfun is
        port        ( clk: in bit;    DATAIN  : in vetor;   Z : out vetor);
end entity tfun;
architecture teste of tfun is
begin
        process(clk)
                variable I: integer;
                variable DATA: matrix;
        begin
                if (clk'event and clk = ‘0') then
                    if (I <= M) then DATA(I) :=  DATAIN;  I:=I+1;
                        else I:=0;          Z <= MR_VETOR(DATA);
                    end if;
                end if;
        end process;
end architecture;
```

Subprograma: *Function e Procedure*

Definidos em package body ou localmente, em *entity* ou *process*.
Permite criar tarefas a serem incorporados em diferentes projetos.

Procedure

Não retorna valor;
Pode modificar valor em pinos de saída.

Function

Retorna valor;
Não modifica valor em pinos de saída.

Exercício

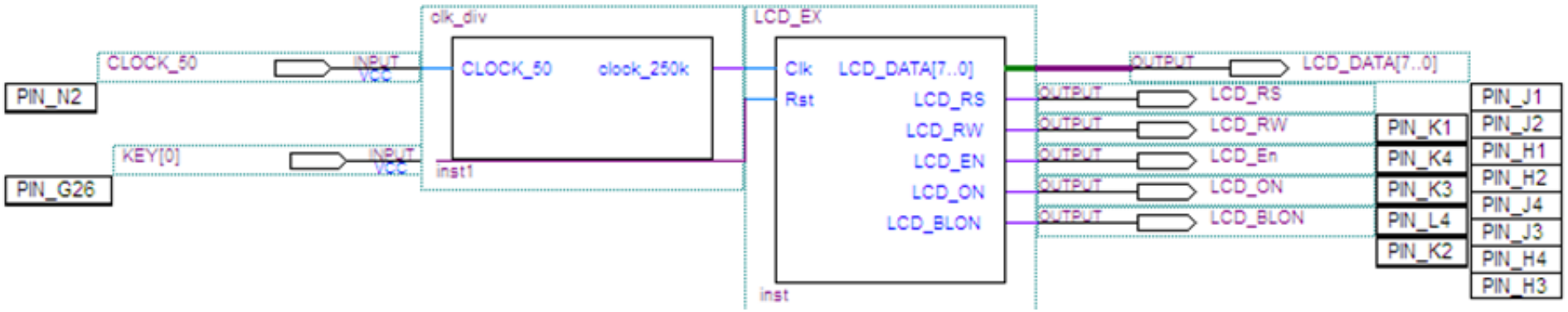
Criar um componente que aceita 6 opcodes (mor, mand,mxor,meq,mg,ms). O componente deve receber ainda dois operandos de 4 bits e realizar operações lógicas (or, and, xor) e comparação (igual, maior, menor) de acordo com o opcode. A saída lógica ou o resultado da comparação devem ser apresentadas em saídas distintas (sai1, comp1).

```
library IEEE;
use IEEE.std_logic_1164.all;
package tiposeprocedure is
constant N:integer:=3;   type OPCODE is (mor, mand, mxor, meq,mg,ms);
procedure executa (signal A,B: in Std_logic_vector(N downto 0);signal  instr : in OPCODE;
                                        signal saida:out Std_logic_vector(N downto 0); signal  COMP : out boolean);
end tiposeprocedure;
package body tiposeprocedure  is
procedure executa (signal A,B: in std_logic_vector(N downto 0); signal instr : in OPCODE;
                                        signal Saida:out Std_logic_vector(N downto 0); signal COMP : out boolean) is
            begin
            case instr is
             when mor =>          saida <=  A or   B;              comp  <=    false;
             when mand =>         saida <=  A and B;               comp  <=    false;
             when mxor =>         saida <=  A xor  B;              comp  <=    false;
             when meq  =>         saida <=  (others =>'0');        comp  <=    A = B;
            when mg =>            saida <=  (others =>'0');        comp  <=    A > B;
             when ms =>           saida <=  (others =>'0');        comp  <=    A < B;
            end case;
end procedure executa;
end package body tiposeprocedure;
```

```
library IEEE;
use IEEE.std_logic_1164.all;

use WORK.tiposeprocedure.ALL;

entity tprd is
port (A1,B1: in Std_logic_vector(N downto 0); inst1 : in OPCODE;
                                    sai1:out Std_logic_vector(N downto 0); COMP1 : out
boolean);

end entity tprd;

architecture EXAMPLE of tprd is
begin
executa(A1,B1,inst1,sai1,comp1);
end architecture;
```

Exercício - LCD

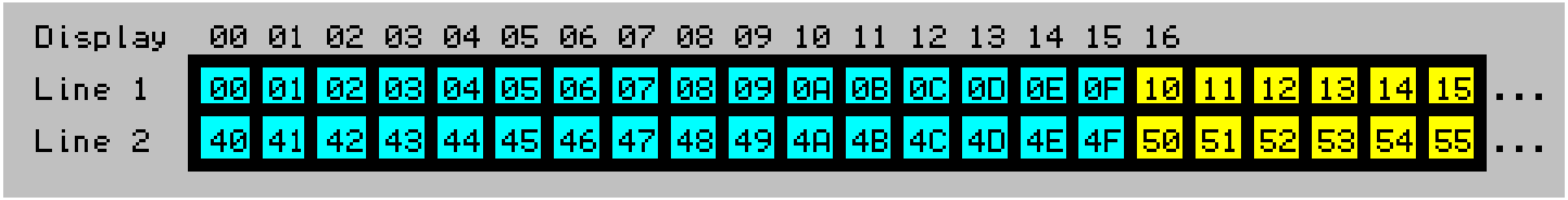
Criar módulo para escrever mensagem no display de cristal líquido do módulo LCD



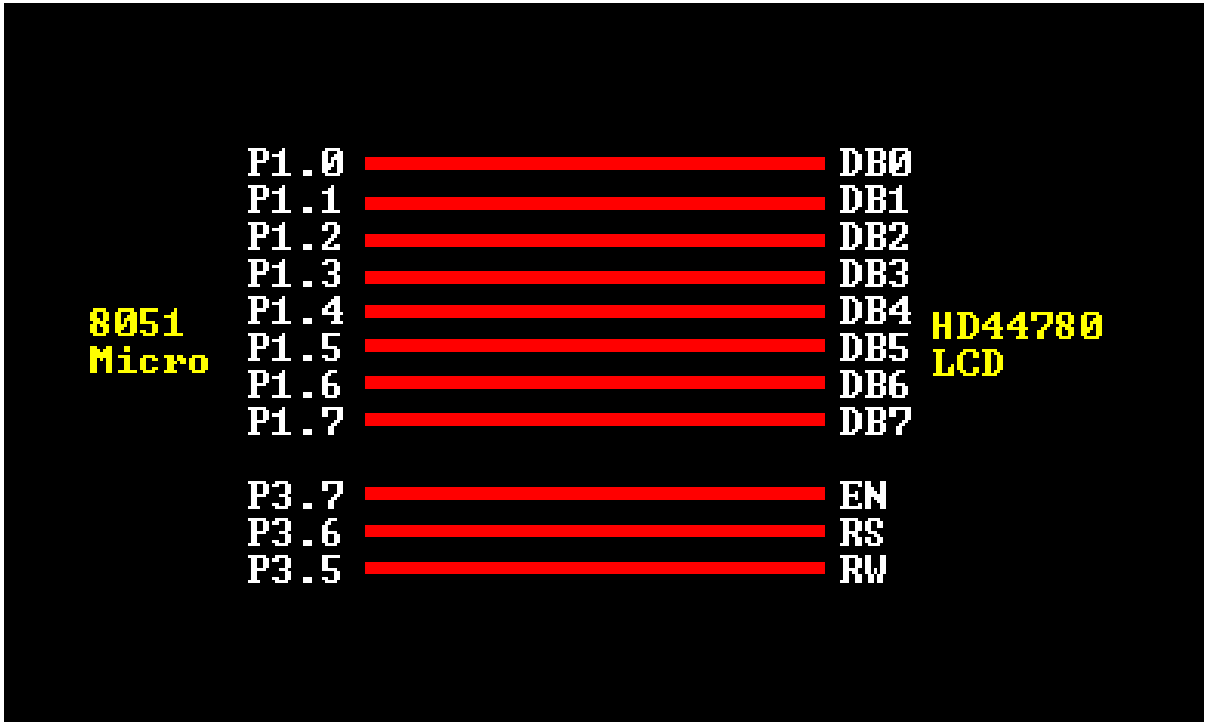
Assignments => Import Assignments => DE2_pin_assignments.csv

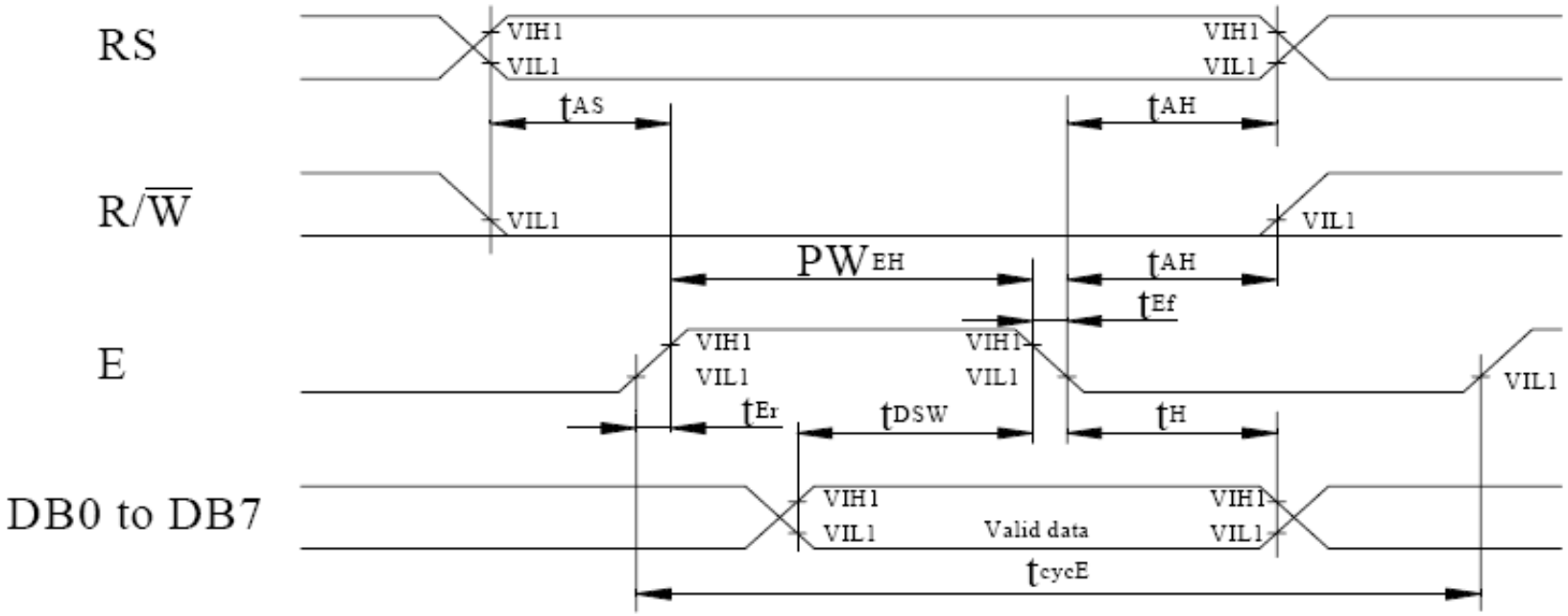
Endereço da Área Visível

LCD de 2 linhas



Conexão e procedimento de escrita





Signal Name	FPGA Pin No.	Description
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF

```
-- ----------------------------
   Entity LCD_EX is
-- ----------------------------
  port(     Clk                 : in std_logic;              -- System Clock de 250kHz
            Rst                 : in std_logic;

            -- LCD signals
      LCD_DATA    : inout std_logic_vector(7 downto 0);  -- LCD data is bidirectional bus...
      LCD_RS                  : out   std_logic;              -- LCD register select
      LCD_RW        : out   std_logic;                        -- LCD Read / nWrite
      LCD_EN        : out   std_logic;                        -- LCD Enable
      LCD_ON        : out   std_logic;
      LCD_BLON    : out   std_logic
      );
end LCD_EX;
```

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;
ENTITY clk_div IS
        PORT
        (       CLOCK_50                                : IN    STD_LOGIC;
                clock_250k                              : OUT   STD_LOGIC);
END clk_div;
ARCHITECTURE a OF clk_div IS
BEGIN
        PROCESS
        variable count: integer range 0 to 200;
        BEGIN
                WAIT UNTIL CLOCK_50'EVENT and CLOCK_50 = '1';
                        IF count < 99 THEN count := count + 1; clock_250k <= '0';
                        ELSif (count = 200) then count := 0; clock_250k <= '0';
                        else    count := count + 1; clock_250k <= '1';
                        END IF;
        END PROCESS;
END a;
```

```
package COMANDOS is
--  17 PALAVRAS DE 9 BITS
      constant NRO_BITS:  INTEGER := 9;          constant NRO_PALV:  INTEGER :=  17;
      type ROM is array (0 to NRO_PALV-1) of std_logic_vector (NRO_BITS-1 downto 0);
            constant TABELA:  ROM := ROM'(
--
                        "000111000",  -- comando    Function Set  8 bits   - 2 linhas -  5x8
                        "000001110",  -- comando    Display On    Cursor On
                        "000000110",  -- comando    Escreve dado => desloca cursor direita
                        "000000001",  -- comando    Clear Display
                        "010000000",  -- comando    DDRAM Address
                        "100100000",
                        "100100000",
                        "101010000",
                        .....
```

```
when Initial =>
                                        Dado   := Tabela(I);
                                        State  <= Initial;
                                        LCD_RS <= DADO(8);
                                        LCD_DATA  <= DADO(7 downto 0);
                                        LCD_EN <= '0';
        case Count is

                when 1   =>    LCD_EN <= '1';            Count := Count+1;
                when 12 =>       -- aguarda 48 us para clock de 250kHz

               if    (I = 3 )then  Count := Count+1; -- identifica comando clear display
                           elsif (I = 16) then State  <= Stop;
                           else  Count := 0; I:=I+1;
                end if;
                  when  500 =>   Count  := 0; I:=I+1;  -- aguarda 2 ms para clear display
                  when others =>                Count := Count+1;
               end case;
```

Controle de Fluxo Seqüencial :

- WAIT

  O comando WAIT suspende a execução de um processo até que uma borda de subida ou borda de descida de um sinal seja detectado. Não deve aparecer em processos que tenham lista de sensitividade. Exemplos de declaração:

wait until signal = value ;

wait until signal'event and signal = value;

wait until not signal'stable and signal = value ;

Declarações equivalentes para Wait:

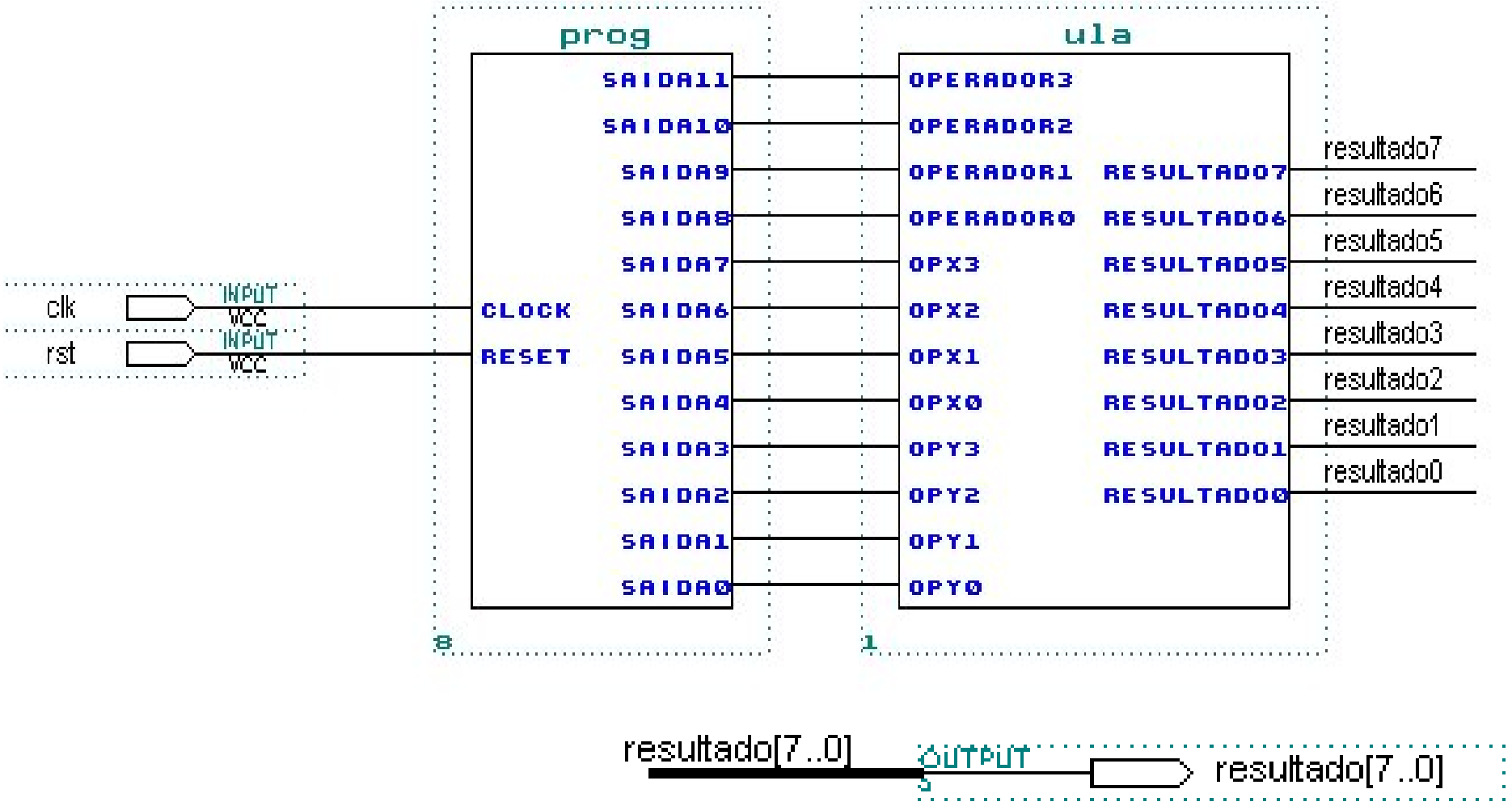
wait until CLK = '1';

wait until CLK'event and CLK = '1';

wait until not CLK'stable and CLK = '1';

Exercício

Criar uma memória de programa e um processador que execute um conjunto de 6 instruções



Exercício - ROM

Criar uma memória ROM que coloque na saída um vetor de 12 bits a cada pulso de clock. Ao chegar no final do último dado da memória, o próximo pulso de clock deve apresentar na saída, o primeiro dado da tabela.

Os 4 bits mais significativos da ROM devem conter o código de uma instrução, os próximos 4 bits devem ser o operando X e os últimos 4 bits o operando Y.

As instruções a serem executadas são ADD (0000) , MUL (0001), SUB (0010) , AND (0011), OR (0100), FATORIAL de X(0101).

Exercício – ROM – Parte I

```
-- EXEMPLO DE CRIACAO DE ROM PARA ENVIAR COMANDOS A ALU
--  UTILIZADO PELO PROGRAMA PROG.VHD
package COMANDOS is
--  7 PALAVRAS DE 12 BITS CONTENDO INTRUCOES (4 BITS) E OPERANDOS (4 BITS CD.)
      constant NRO_BITS:  INTEGER := 12;       constant NRO_PALV:  INTEGER :=  7;
      subtype  INSTRDADO  is  BIT_VECTOR    (NRO_BITS-1 downto 0);
      subtype  NRO_COMAN  is  INTEGER range  0 to NRO_PALV-1;
      type ROM is array (0 to NRO_PALV-1) of INSTRDADO;
                constant TABELA:  ROM := ROM'(
--                              INSTR  X   Y                    MNEMONICO
                              "000000100011",  -- instrucao         ADD 2,3
                              "000100110011",  -- instrucao         MUL 3,3
                              "001000110010",  -- instrucao         SUB 3,2
                              "001100110010",  -- instrucao         AND 3,2
                              "010000110010",  -- instrucao         OR  3,2
                              O"2477",        -- B"010 100 111 111", -- instrucao              FATORIAL X
                              X"3AF"          --"0011_1010_1111"     -- instrucao              AND A,F
                                 );

end package COMANDOS;
```

Exercício – ROM – Parte II

```
-- EXEMPLO DE MODULO DE COMANDO PARA LER ROM;  ENVIA COMANDOS PARA A ALU
use work.COMANDOS.all;
entity PROG is
        port(   CLOCK, RESET    :       in      BIT;
                SAIDA           :       out     INSTRDADO );
end entity  PROG;
architecture BEHAVIOR of PROG is
signal ENDERECO: NRO_COMAN;
begin
    process  -- Time stepping process
    begin
    wait until CLOCK'event and CLOCK ='1';
       if    RESET = '1'   then ENDERECO <= NRO_COMAN'low;  --  testa se reset em nivel alto e  inicia
          elsif ENDERECO = NRO_COMAN'high then ENDERECO <= NRO_COMAN'low; -- testa se final e  re-
inicia                  else     ENDERECO <= ENDERECO + 1;     end if;     -- proximo comando
     end process;
     process (ENDERECO)
            begin       SAIDA <= TABELA(ENDERECO);                      end process;
end architecture  ;
```

Exercício - CPU

Criar um processador que interprete as instruções geradas pela ROM e execute as instruções solicitadas

Exercício – CPU – Parte I

```
-- -- exemplo da implementacao de processador.
-- ilustra utilizacao de funcoes em biblioteca.
-- faz 5 operacoes, sendo especificadas pelo operador (4 bits).
-- aceita 2 operandos de 4 bits (opx e opy). Resultado de 8 bits.
-- Devido ao fato de se implementar operacoes logicas nesta CPU
-- fez-se necessario fazer a saida em std_logic e aplicar as conversoes abaixo.

library IEEE;
use IEEE.std_logic_1164.all;
USE ieee.std_logic_arith.all;

package DATA_TYPES is
constant        n: Natural := 4;
constant        k: Natural := 8;
subtype         DTYPE is SIGNED (n-1 downto 0);
subtype         ITYPE is STD_LOGIC_VECTOR(n-1 downto 0);
subtype         OTYPE is STD_LOGIC_VECTOR(k-1 downto 0);
end package DATA_TYPES;
```

Exercício – CPU – Parte II

```
library IEEE;
use IEEE.std_logic_1164.all;
USE ieee.std_logic_arith.all;
use work.data_types.all;

ENTITY CPU IS
PORT( operador : IN ITYPE;   opx,opy : IN DTYPE;
        resultado : OUT OTYPE );
END ENTITY CPU;

ARCHITECTURE resolve OF CPU IS
BEGIN
PROCESS (OPERADOR, opx, opy)
        VARIABLE auxiliar : DTYPE; -- insercao de variavei de tamanho n
        VARIABLE AUXLGA : ITYPE;
        VARIABLE AUXLGB : ITYPE;
        VARIABLE AUXi,fator : INTEGER;
BEGIN
```

Exercício – CPU – Parte III

```
BEGIN
case operador is
        when "0000" =>  AUXILIAR :=  (OPX + OPY);                    -- ADD X,Y
                        AUXLGA :=    conv_STD_LOGIC_VECTOR(auxiliar,4);
                                IF AUXILIAR(3) = '1'  THEN
                                                resultado <= "1111" & AUXLGA;
                                ELSE
                                                resultado <= "0000" & AUXLGA;
                                END IF;

        when "0001" =>  resultado <= OPX * OPY;                      -- MUL X,Y

        when "0010" =>  AUXILIAR :=  (OPX - OPY);                    -- SUB X,Y
                        AUXLGA    :=    conv_STD_LOGIC_VECTOR(auxiliar,4);

                        IF AUXILIAR(3) = '1' THEN resultado <= "1111" & AUXLGA;
                        ELSE                      resultado <= "0000" & AUXLGA;
                        END IF;
```

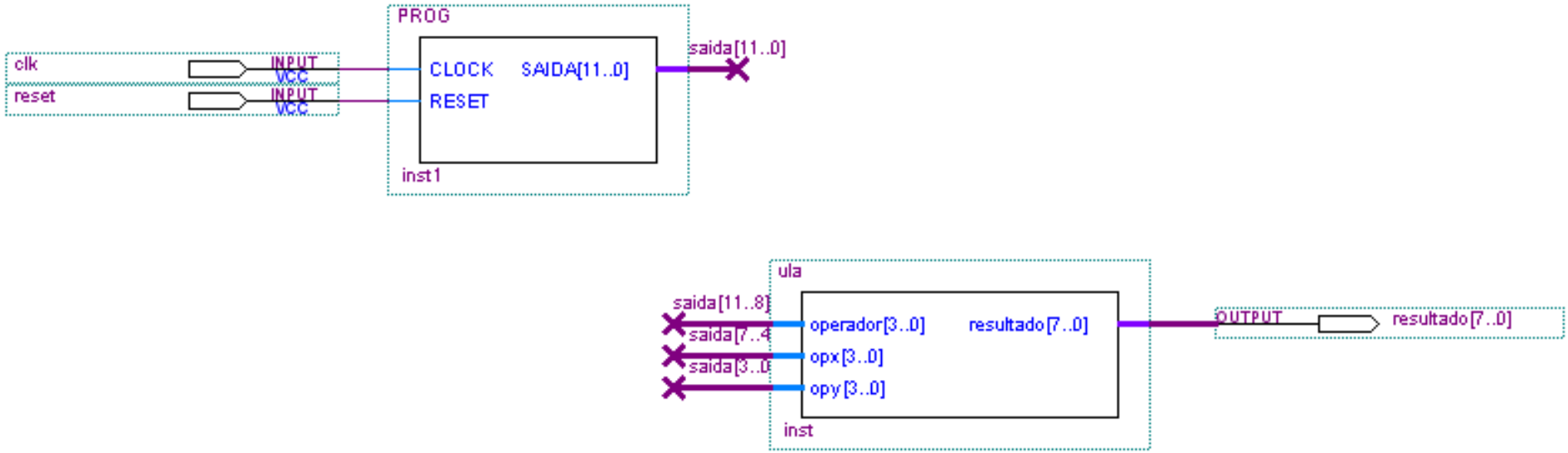
Exercício – CPU – Parte IV

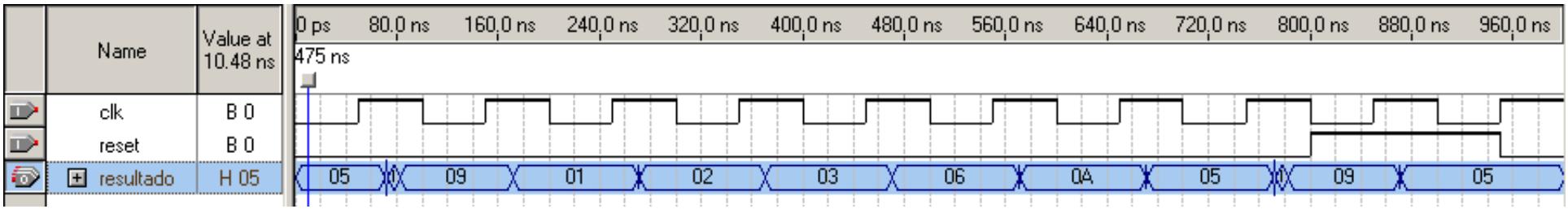
```
when "0011" =>                                          -- AND X,Y

        AUXLGA   :=  conv_STD_LOGIC_VECTOR(OPX,4);
        AUXLGB   :=  conv_STD_LOGIC_VECTOR(OPY,4);
        AUXLGA   :=  AUXLGA and AUXLGB;
        resultado <= "0000" & AUXLGA;


when "0100" =>                                          -- OR  X,Y

        AUXLGA   :=  conv_STD_LOGIC_VECTOR(OPX,4);
        AUXLGB   :=  conv_STD_LOGIC_VECTOR(OPY,4);
        AUXLGA   :=  AUXLGA or AUXLGB;
        resultado <= "0000" & AUXLGA;
```

Exercício – CPU – Parte IV

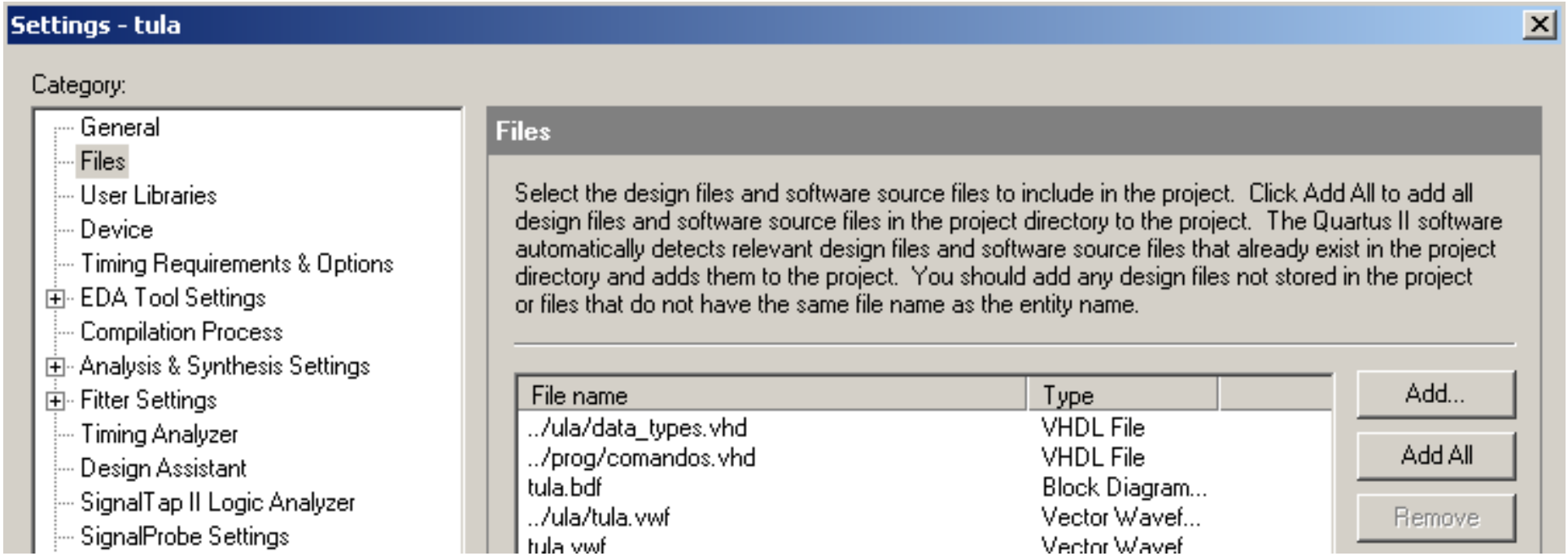
```
when "0101" =>                                          -- FATORIAL X

                        auxi   :=  conv_INTEGER(OPX);

                        fator  :=1;

                        for    j in 2 to 15 loop

                                exit when j>auxi;
                                fator := fator*j;

                        end loop;

                        resultado   <=  conv_STD_LOGIC_VECTOR(fator,8);


            when others  => null;

        end case;


    END PROCESS;
END ARCHITECTURE;
```

Exercício – CPU – BDF e Simulação

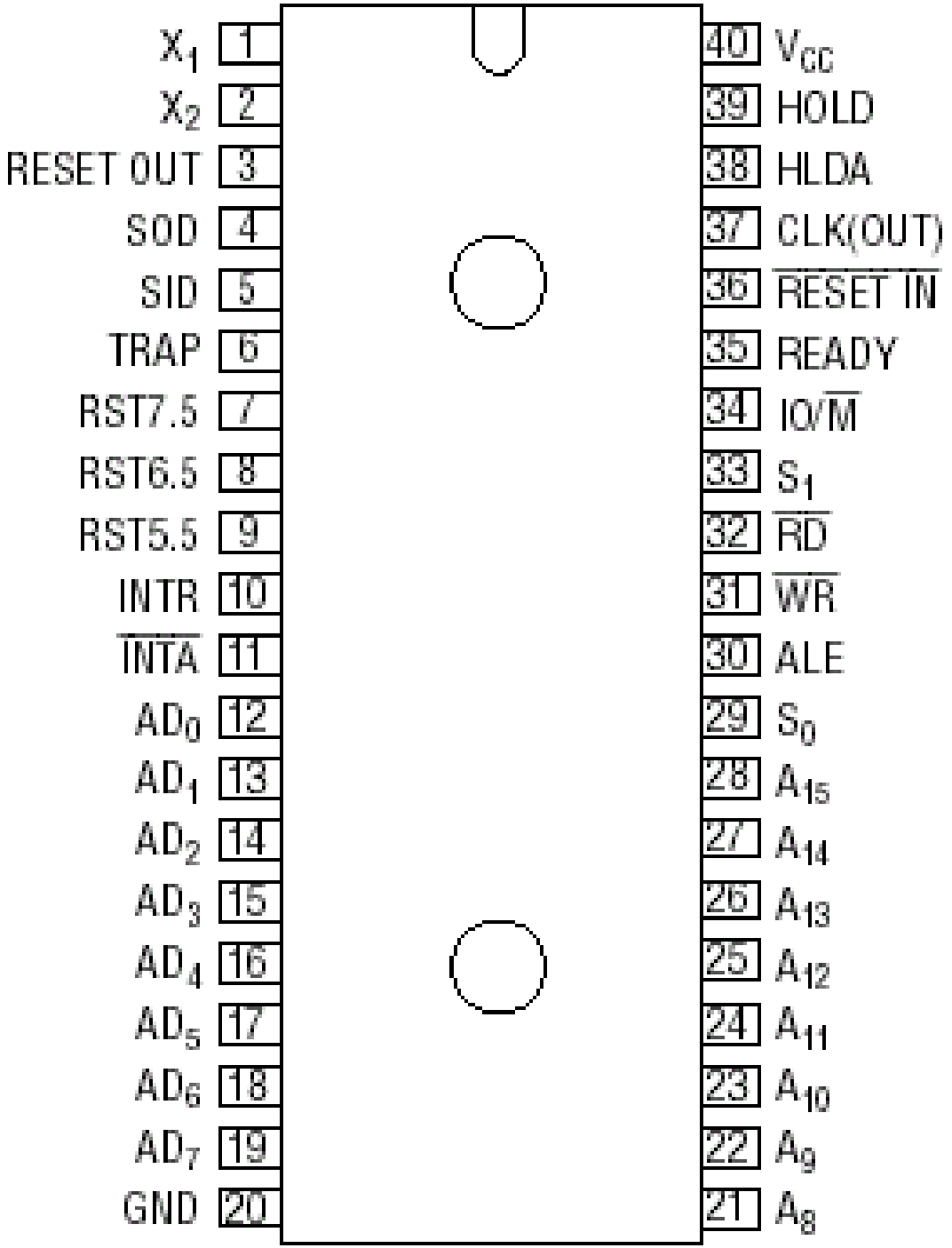


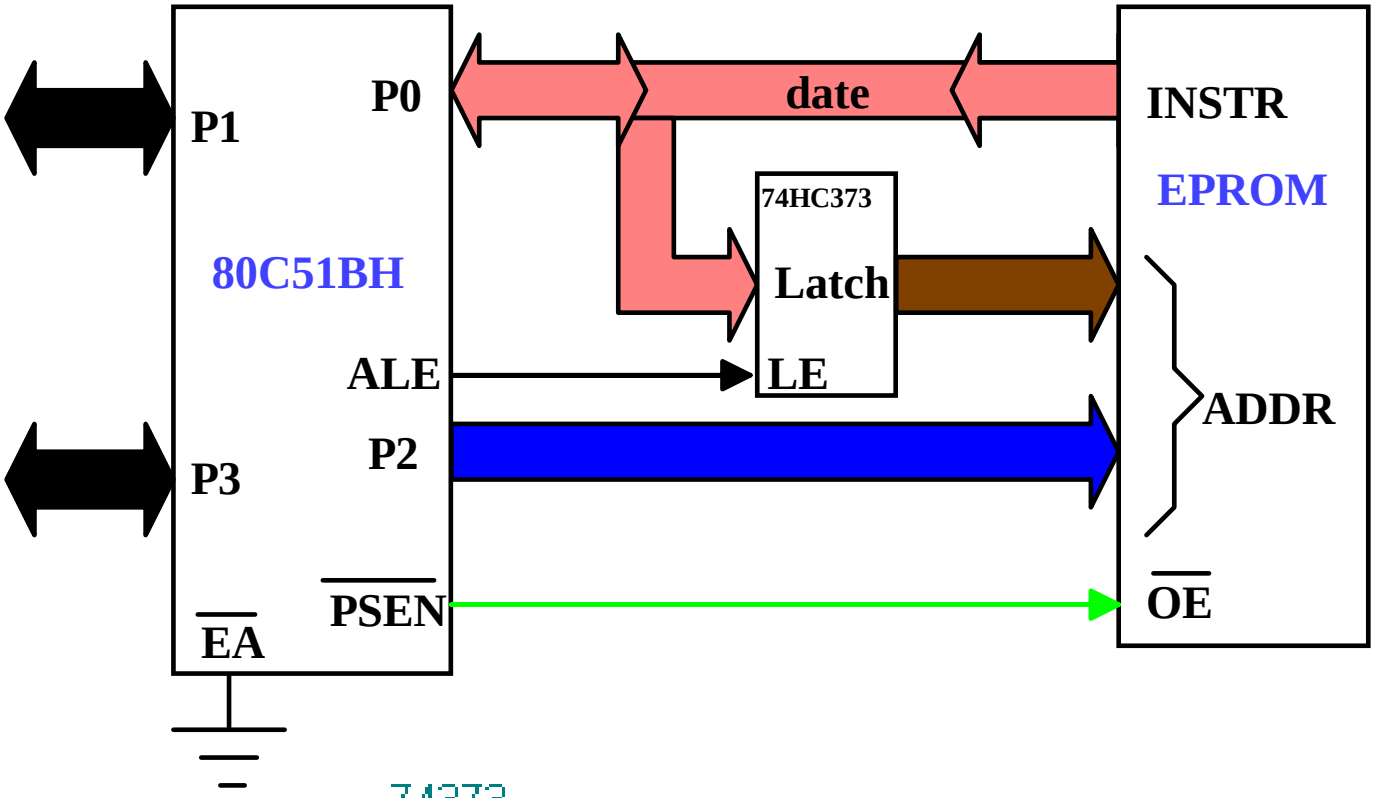


Exercício – CPU – Observação

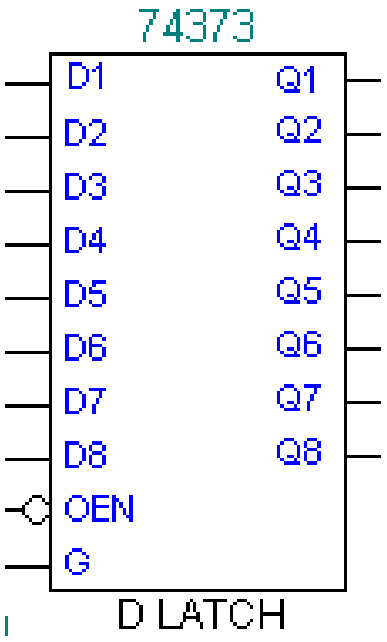


Microprocessador 8085



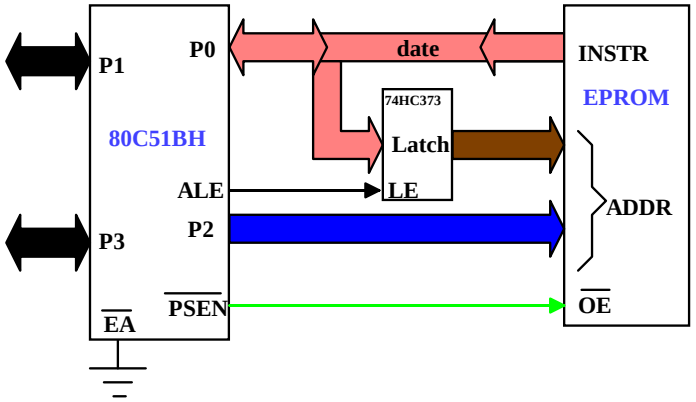


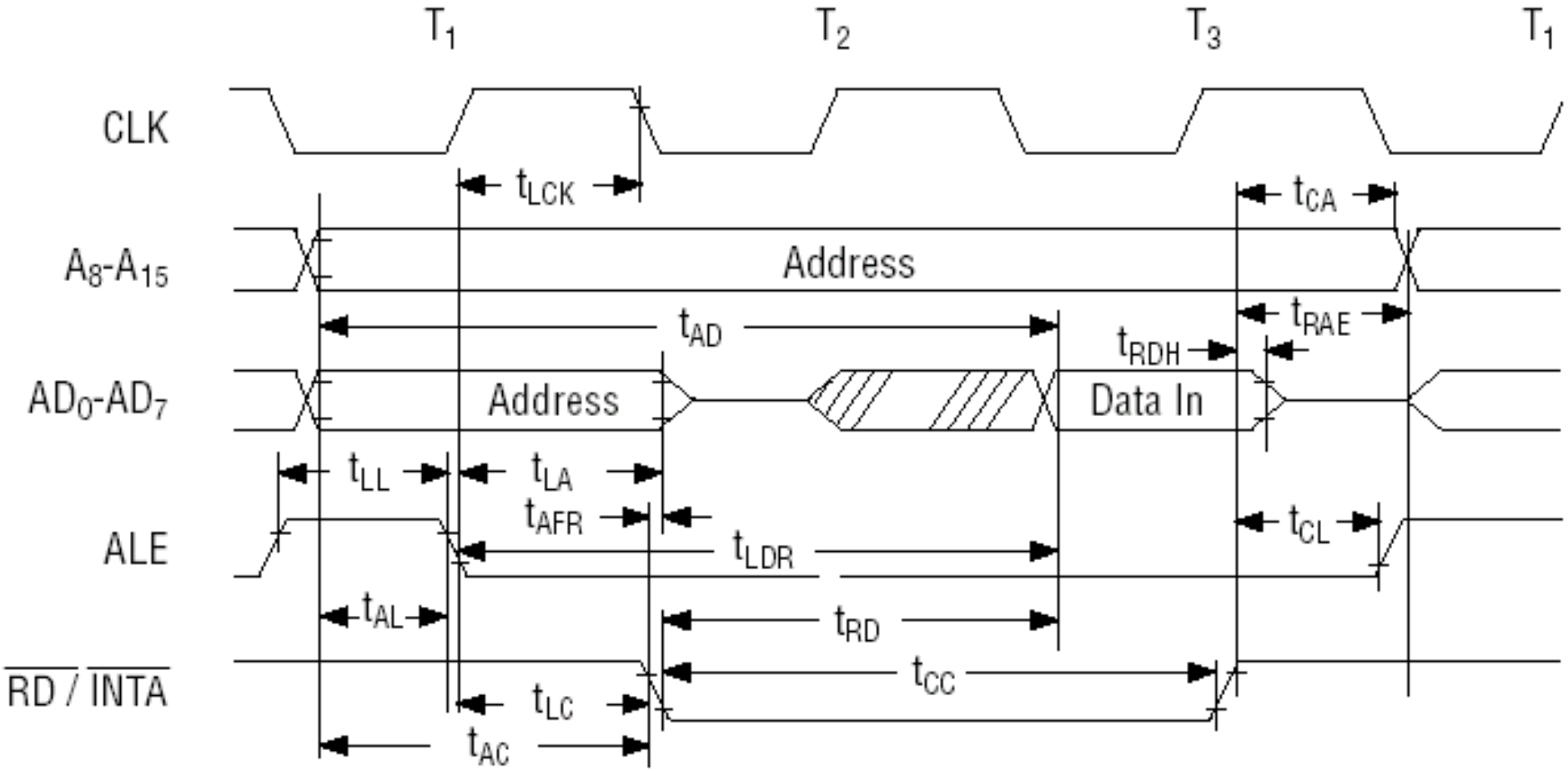
Interface do Microprocessador Com Memória



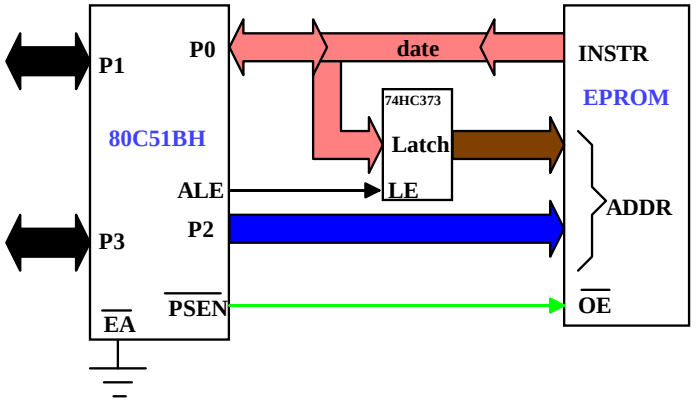
Inputs			Outputs
OEN	G	D	Q
H	X	X	Z
L	X	X	X
L	H	L	L
L	H	H	H
L	L	X	Q_0

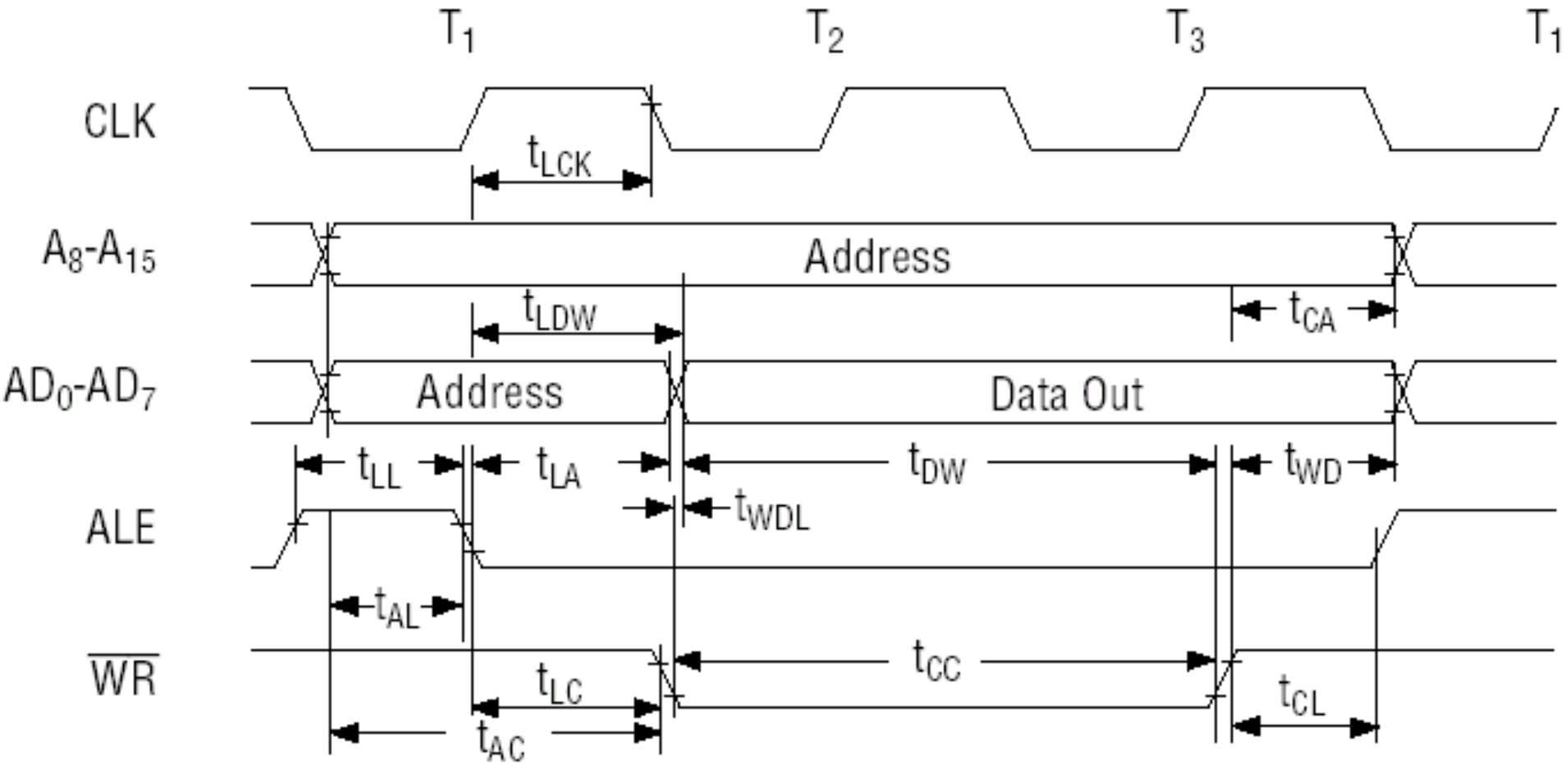
Exemplo de Acesso à Memória
Leitura



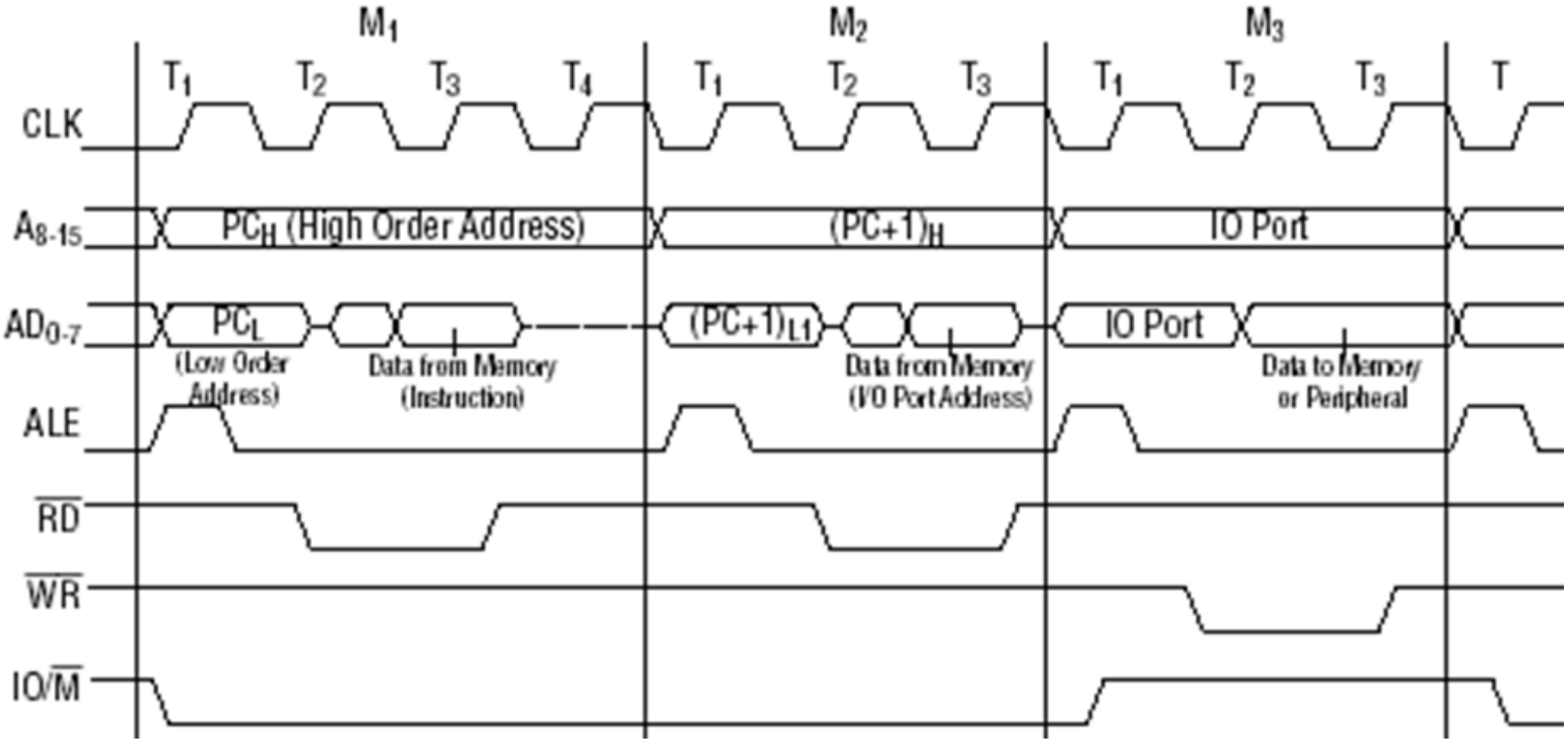


Exemplo de Acesso à Memória Escrita





Exemplo – CPU - 8085



2 Ciclos de leitura da memória de programa e um ciclo de escrita em dispositivo de I/O pelo 8085

Exemplo – CPU – 8085

Criar um subconjunto do 8085 que execute instruções de leitura e escrita (MOV A,M; MOV M,A e JMP) seguindo os padrões de acesso ao barramento.

Exemplo – CPU - 8085

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY m8085 IS
PORT(       iclk, reset          : IN STD_LOGIC;
            ADRH                 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0 );
            ADRL                 : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0 );
            WR,RD                : OUT STD_LOGIC;
            ALE                  : BUFFER STD_LOGIC);
END ENTITY m8085;

ARCHITECTURE micro OF m8085 IS
TYPE        estados    IS (T1, T2, T3, T4);
TYPE        tipo_acao  IS (fetch, MOV_AM, MOV_MA, JMP);
SIGNAL      instrucao                    : tipo_acao;
SIGNAL      IR,ACC,H,L                   : STD_LOGIC_VECTOR(7 DOWNTO 0 );
SIGNAL      estado                       : estados;
SIGNAL      clock                        : STD_LOGIC;
```

Exemplo – CPU - 8085

```
SIGNAL    flag_JMP                    : integer range 0 to 6;
SIGNAL    PC                          : STD_LOGIC_VECTOR( 15 DOWNTO 0 );
SIGNAL    LSB_ADD, MSB_ADD  : STD_LOGIC_VECTOR( 7 DOWNTO 0 );

BEGIN

PROCESS (iclk, reset)

BEGIN
        IF reset = '1' THEN
                        clock <= '1'; ALE <= '1'; RD <= '1';WR <= '1';
                        ADRL <=(others => '0'); ADRH <= (others => '0');
                        H <= X"20"; L <= X"00";
                        flag_JMP <= 0;

        ELSIF rising_edge(iclk) THEN
                        clock <= not clock;
```

```
CASE estado IS

    WHEN T1 =>  ALE <= '0';
                case instrucao is
                    when fetch => ADRH <= PC(15 downto 8);
                                  RD <= clock; WR <= '1';
                                  flag_JMP <= 0;
                                   if (ALE = '1') then ADRL <= PC(7 downto 0);
                                  else   ADRL <= (others => 'Z');                end if;

                   when MOV_AM => ADRH <= H; ADRL <= (others => 'Z');
                                  RD <= clock; WR <= '1';

                   when MOV_MA  =>     ADRH <= H;  ADRL <= (others => 'Z');
                                       RD <= '1'; WR <= clock;

                   when JMP =>         RD  <= clock; ADRL <= (others => 'Z');

                        if  (flag_JMP < 3)  then   ADRH <= PC(15 downto 8);  end if;

                  end case;
```

```
WHEN T2 => ALE <= '0';
            case instrucao is
                        when JMP  => RD <= '0'; WR <= '1'; ADRL <= (others => 'Z');
                                                           ADRH <= PC(15 downto 8);

                        when fetch  => RD <= '0'; WR <= '1'; IR <= ADRL;
                                       ADRL <= (others => 'Z'); ADRH <= PC(15 downto 8);

                         when  MOV_AM  => RD <= '0'; WR <= '1'; ACC <= ADRL;
                                                    ADRL <= (others => 'Z');  ADRH <= H;

                         when  MOV_MA => RD <= '1';WR <= '0';
                                                 ADRL <= ACC;    ADRH <= H;

                        end case;
```

```
WHEN T3 =>  ADRH <= PC(15 downto 8);
            RD <=  '1';
            case instrucao is
                when fetch   =>  WR <= '1'; ALE <= '0';
                when JMP   =>  WR <= '1'; ALE <=   not clock;
                               flag_JMP <= flag_JMP+1;
                               if    (flag_JMP = 0)  then LSB_ADD <=  ADRL;
                                                          ADRL <= PC(7 downto 0);

                               elsif (flag_JMP = 2)      then MSB_ADD <=  ADRL;
                                                              ADRH <= MSB_ADD;
                                                              ADRL <= LSB_ADD;

                               elsif (flag_JMP > 2)      then ADRH <= MSB_ADD;
                                                              ADRL <= LSB_ADD;
                               end if;
                when MOV_AM =>  WR <= '1'; ALE <=   not clock;
                                ADRL <= PC(7 downto 0);
                when MOV_MA => WR <= '1'; ADRL <= ACC;
                               ALE <=   not clock;
                               ADRL <= PC(7 downto 0);
            end case;
```

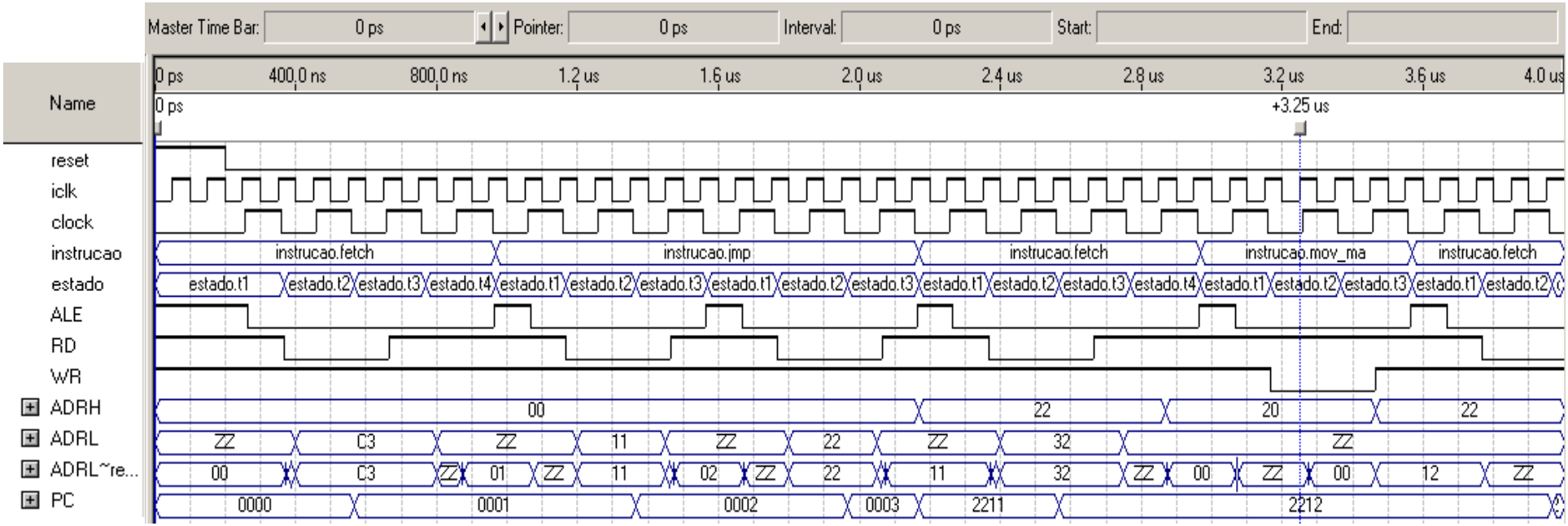
```
        WHEN T4 => ALE <=   not clock; RD <= '1'; WR <= '1';

                    case IR is
                            when X"32" | X"3A" => ADRH <= H;
                                                  ADRL <= L;

                            when  others =>        ADRL <= PC(7 downto 0);
                                                   ADRH <= PC(15 downto 8);

                    end case;

        END CASE;
   END IF;
END PROCESS;
```

```
PROCESS (CLOCK, RESET)
BEGIN
        IF reset = '1' THEN PC <= (others => '0'); estado<= T1; instrucao <= fetch;
        ELSIF clock'EVENT AND clock = '1' THEN
                CASE estado IS
                    WHEN T1 => estado <= T2;
                    WHEN T2 => estado <= T3;
                                case instrucao is
                                        when MOV_AM | MOV_MA =>  PC <= PC;
                                        when others => PC <= PC + 1;
                               end case;
                   WHEN T3 => case instrucao is
                                        when fetch => estado  <= T4;
                                        when MOV_AM => estado <= T1; instrucao <= fetch;
                                        when MOV_MA => estado <= T1; instrucao <= fetch;
                                        when JMP => estado <= T1;
                                                    if (flag_JMP >= 3) then instrucao <= fetch;
                                                            PC(15 downto 8) <=  MSB_ADD;
                                                            PC(7 downto  0) <=  LSB_ADD;
                                                     end if;
                                    end case;
```

Exemplo– CPU - 8085

```
                WHEN T4 => estado <= T1;
                        case IR IS
                                WHEN X"3A" => instrucao <= MOV_AM;
                                WHEN X"32" => instrucao <= MOV_MA;
                                WHEN X"C3" => instrucao <= JMP;
                                WHEN OTHERS => instrucao <= fetch;
                        end case;
                END CASE;
        END IF;
END PROCESS;
END ARCHITECTURE micro;
```

Exemplo – CPU - 8085



Exemplo – RAM

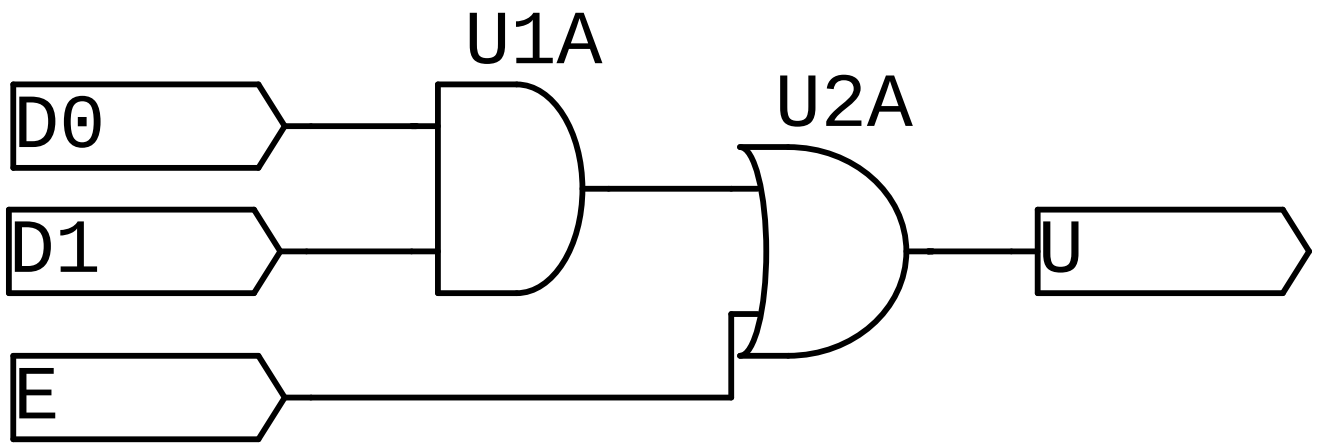
```
library IEEE;
use IEEE.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY ram IS                    -- memória assíncrona
        GENERIC     (  n: NATURAL:= 8; -- tamanho da palavra
                       p: NATURAL:=64; -- nro. de palavras
                       k: NATURAL:= 6);  -- nro. de bits de endereço
        PORT(       DATA          : INOUT  UNSIGNED(n-1 DOWNTO 0);     -- dados bus
                    ADDR          : IN     UNSIGNED(k-1 DOWNTO 0);     -- ender. bus
                    Rd,Wr,CS      : IN     BIT); -- sinais de controle
END ram;
ARCHITECTURE comportamento OF ram IS
SUBTYPE palavra IS UNSIGNED(n-1 DOWNTO 0);
TYPE vetor IS ARRAY(0 TO p-1) OF palavra;
SIGNAL memoria: vetor;
BEGIN
   PROCESS (Wr,Rd,CS,ADDR, DATA,memoria)
        BEGIN
            DATA <= (OTHERS => 'Z');
            IF (CS = '0')  THEN
                  IF (Rd = '0') THEN          DATA <= memoria(CONV_INTEGER(ADDR));
                  ELSIF (Wr = '0') THEN       memoria(CONV_INTEGER(ADDR)) <= DATA;
                  END IF;
            END IF;
     END PROCESS;
END comportamento;
```

Generics

Define constantes utilizadas no projeto. Facilita modificações do mesmo. No modo estrutural, *generics* no corpo da arquitetura prevalecem sobre valor original utilizado na criação do componente.

```
entity ANDGATE is
        generic (M :integer := 8);
        port (      A          : in bit_vector(M-1 downto 0);
                    Z          : out bit);
end entity ANDGATE;
architecture GEN of  ANDGATE is
begin
    process (A)
    variable andout:bit;
    begin
        andout:='1';
        for K in 0 to M-1 loop
        andout:=andout and A(K);
        end loop;
    Z <= andout;
    end process;
end architecture;
```

Generics



Generics

```
entity GENAND is
        port (          D           : in bit_vector(1 downto 0);
                        E           : in bit;
                        U           : out bit);

end entity GENAND;
architecture GEN2 of  GENAND is
        signal C: bit;
        component ANDGATE
                generic (M :integer :=2);
                port(A:in bit_vector(M-1 downto 0); Z:out bit);
        end component;
begin
        U1: ANDGATE port map (D,C);
        U <= C or E;
end architecture;
```

Biblioteca Altera - LPM

LPM: Library of **P**arametrized **M**odules

• Consiste-se de biblioteca de componentes da Altera otimizadas para a arquitetura dos CIs produzidos pela Altera.

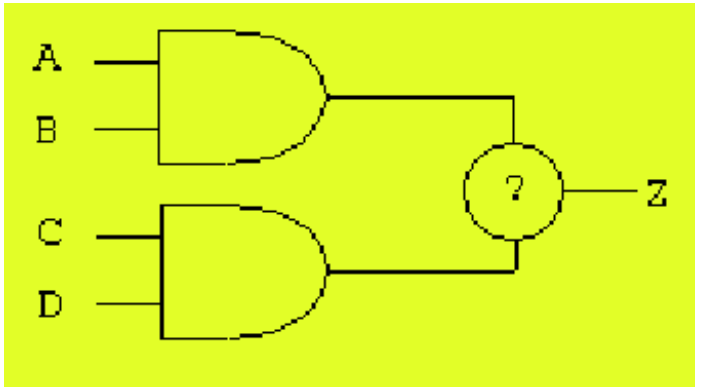
• Podem ser configuradas por meio do **MegaWizard Plug-In Manager** para gerar símbolo para arquivos bdf.

Exemplo – RAM - Megafunção

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
LIBRARY lpm;                    -- declaração da biblioteca ( help do Quartus --  Megafunctions/LPM)
USE lpm.lpm_components.ALL;
-- deve-se suprir para a escrita memwrite em alto e subida de clock   -- para leitura, endereco e memwrite em baixo
ENTITY amemory IS
   PORT(read_data                       : OUT std_logic_vector(7 DOWNTO 0);
      memory_address                    : IN std_logic_vector(2 DOWNTO 0);
      write_data                        : IN std_logic_vector(7 DOWNTO 0);
      Memwrite                          : IN std_logic;
      clock,reset                       : IN std_logic);

END ENTITY   amemory;
ARCHITECTURE behavior OF amemory IS
BEGIN
component LPM_RAM_DQ
    GENERIC MAP (           lpm_widthad     => 3,
                            lpm_outdata     => "UNREGISTERED",
                            lpm_indata      => "REGISTERED",
                            lpm_address_control => "UNREGISTERED",
                            -- Lê arquivo no formato mif para inicializar valores
                            -- lpm_file        => "memory.mif",  -- nome do arquivos com dados
                            lpm_width       => 8)
end component;
    PORT MAP (data => write_data, address => memory_address(2 DOWNTO 0),
                we => Memwrite, inclock => clock, q => read_data);
END ARCHITECTURE;
```

Data Flow Modeling

```
architecture CONCURRENT of MULTIPLE is
begin
    Z <= A and B;
    Z <= C and D;
end architecture;
```



.....

Múltiplos Drives

Mais de um processo atribui valor a mesma variável.

Qual valor a variável assume?

STD_ULOGIC X STD_LOGIC

	U	X	0	1	Z	W	L	H	-
U	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'
X	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
0	'U'	'X'	'0'	'X'	'0'	'0'	'0'	'0'	'X'
1	'U'	'X'	'X'	'1'	'1'	'1'	'1'	'1'	'X'
Z	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'X'
W	'U'	'X'	'0'	'1'	'W'	'W'	'W'	'W'	'X'
L	'U'	'X'	'0'	'1'	'L'	'W'	'L'	'W'	'X'
H	'U'	'X'	'0'	'1'	'H'	'W'	'W'	'H'	'X'
-	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'

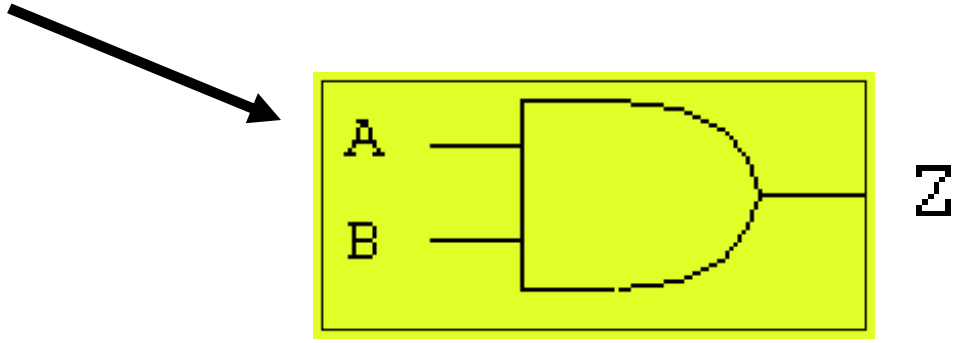
Type Std_Ulogic is (-- ieee library

'U', -- Unitialized,
'X','0','1', -- Forcing Unknow, 0, 1
'Z', -- High Impedance
'W','L','H', -- Weak Unknow, 0, 1
'-'); -- Don't Care

Existe função na biblioteca para determinar o valor final atribuído ao sinal:Std_logic.

Data Flow Modeling / Function / Resolution Function

```
use work.RES_PACK.all;

entity wiredand is
        port(A, B: in BIT; Z: out RESOLVED_BIT);
end entity wiredand;

architecture WAND_VHDL of wiredand is
        begin
                Z <= A;
                Z <= B;
end architecture;
```



Data Flow Modeling - Resolution Function

```
package RES_PACK is
function RES_FUNC(DATA: in BIT_VECTOR) return BIT;
subtype RESOLVED_BIT is RES_FUNC BIT;
end package RES_PACK ;

package body RES_PACK is
function RES_FUNC(DATA: in BIT_VECTOR) return BIT is
-- synopsys resolution_method wired_and
        begin
                for I in DATA'range loop
                        if DATA(I) = '0' then
                        return '0';
                        end if;
                end loop;
                return '1';
        end;
end package body RES_PACK;
```

- **C → Char, Integer, Float, Double**

  Ex. → char EXEMPLO;

- **VHDL → Integer, Enumeration (Bit, Boolean, Character), Array (String), Record.**

  - **Bit → '0' ou '1'**

    Ex. » variable CLR : bit;

  - **Boolean → false ou true.**

  - **Character → valor ASCII de um caractere.**

    Ex. » variable letra: character;

    letra := 'A';

Arquivos para o Exemplo de escrita em Crital Líquido

```
-- EXEMPLO DE CRIACAO DE ROM PARA ENVIAR COMANDOS A ALU
--  UTILIZADO PELO PROGRAMA LCD_EX.VHD
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
package COMANDOS is
--  17 PALAVRAS DE 9 BITS
     constant NRO_BITS:  INTEGER := 9;                constant NRO_PALV:  INTEGER :=  17;
     type ROM is array (0 to NRO_PALV-1) of std_logic_vector (NRO_BITS-1 downto 0);
               constant TABELA:  ROM := ROM'(
                              "000111000", -- comando Function Set  8 bits 2 linhas 5x8
                              "000001110", -- comando Display On    D=1 C=1 B=0
                              "000000110", -- comando Entry Mode     I/D=1 SH=1
                              "000000001", -- comando Clear Display
                              "010000000", -- comando DDRAM Address
                              "100100000",
                              "100100000",
                              "101010000",
                              "101010010",
                              "101001111",
                              "101001010",
                              "100100000",
                              "101000110",
                              "101010000",
                              "101000111",
                              "101000001",
                              "100100000"
                              );
end package COMANDOS;
```

```
-- EXEMPLO DE DIVISÃO DO CLOCK DE 50 MHz PARA GERAR CLOCK DE 250 kHz (4 us)
--  UTILIZADO PELO PROGRAMA LCD_EX.VHD

LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.all;

ENTITY clk_div IS

        PORT
        (       CLOCK_50                                        : IN    STD_LOGIC;
                clock_250k                                      : OUT   STD_LOGIC);
END clk_div;

ARCHITECTURE a OF clk_div IS
BEGIN
        PROCESS
        variable count: integer range 0 to 200;
        BEGIN

                WAIT UNTIL CLOCK_50'EVENT and CLOCK_50 = '1';

                        IF count < 99 THEN count := count + 1; clock_250k <= '0';
                        ELSif (count = 200) then count := 0; clock_250k <= '0';
                        else            count := count + 1; clock_250k <= '1';
                        END IF;

        END PROCESS;
END a;
```

```
library ieee;
use ieee.std_logic_1164.all;
 use work.comandos.all;
  -- ------------------------------
    Entity LCD_EX is
port( Clk               : in std_logic;       -- System Clock de 250kHz
      Rst               : in std_logic;
-- LCD signals
      LCD_DATA    : inout std_logic_vector(7 downto 0);          -- LCD data is a bidirectional bus...
      LCD_RS            : out   std_logic;                              -- LCD register select
      LCD_RW            : out   std_logic;                        -- LCD Read / nWrite
      LCD_EN            : out   std_logic;                        -- LCD Enable
      LCD_ON            : out   std_logic;
      LCD_BLON          : out   std_logic
      );
end LCD_EX;
-- ------------------------------
 Architecture RTL of LCD_EX is
type State_type is (Boot, Initial, Stop);
  signal State : State_type;
Begin
             LCD_RW <= '0';
             LCD_ON <= '1';
             LCD_BLON <= '1';
```

```
process (Clk, Rst)
  variable Count: integer range 0 to 7500;
  variable I: integer range 0 to 30;
  variable DADO: std_logic_vector (8 downto 0);
Begin
    if Rst = '1' then
                                    State    <= Boot;
                                    Count   := 0;
                                    LCD_EN   <= '0';
                                     LCD_RS   <= '1';
                                    LCD_DATA     <= (others=>'0');
    elsif rising_edge(Clk) then

     case State is
           when Boot =>    State            <= Boot;
                      LCD_RS    <= '1';
                       LCD_EN    <= '0';
                      Count := Count + 1;
                      I  := 0;
                      if Count = 7500 then  -- aguarda 30 ms para clock de 250kHz
                                            LCD_RS    <= '0';
                                            Count      := 0;
                                            State      <= Initial;
                      end if;
```

```
when Initial =>       Dado   := Tabela(I);
                       State  <= Initial;
                       LCD_RS <= DADO(8);
                       LCD_DATA  <= DADO(7 downto 0);
                       LCD_EN <= '0';
                      case Count is

                        when 1    =>   LCD_EN <= '1';    Count := Count+1;
                        when 12 =>      -- aguarda 48 us para clock de 250kHz
                                  if    (I = 3 )then  Count := Count+1; -- identifica comando clear display
                                  elsif (I = 16) then State  <= Stop;
                                  else  Count := 0; I:=I+1; end if;

                        when  500 =>          Count  := 0; I:=I+1; -- aguarda 2 ms para clear display

                        when others =>        Count := Count+1;
                    end case;

     when Stop =>         State  <= Stop;
end case;
    end if;
 end process;

end RTL;
```