Laboratório de Eletrônica Digital
SEL-EESC-USP

EESC · USP

Introdução a VHDL

Aula 3

Professora Luiza Maria Romeiro Codá

ARCHITECTURE

- **Descrição por fluxo de dados (*Data-Flow*):**
  Descreve o que o sistema deve fazer utilizando expressões lógicas.

- **Descrição estrutural:**
  Descreve como é o hardware em termos de interconexão de componentes.

- **Descrição comportamental:**
  Descreve o que o sistema deve fazer de forma abstrata.

ARCHITECTURE – Comportamental

Descreve o comportamento ou funcionalidade do circuito de forma abstrata usando o comando PROCESS.

O Comando PROCESS cria uma região de código sequencial e permite a aplicação de instruções sequenciais.

Formato:

```
PROCESS(<Lista de Sensibilidade>)
BEGIN
--Comandos sequenciais do processo
 END PROCESS;
```

```
<nome_opcional>: PROCESS (lista de sensibilidade)
BEGIN
 --Comandos sequenciais do processo
END PROCESS <nome_opcional>;
```

Lista de sensibilidade: deve constar os sinais que devem alterar a saída do circuito, e é composta de todos os sinais de entrada para os circuitos combinatórios. Sinais com inicialização assíncrona devem constar obrigatoriamente na lista. Sinais síncronos não necessariamente.

Ex: Para os registradores assíncronos, a lista seria composta do clock e do reset; e para os registradores síncronos, do clock.

ARCHITECTURE – Comportamental

```
PROCESS(sensibilidade_sinal_1, sinal_2...)
BEGIN
    -- Comandos sequenciais do processo
END PROCESS;
```

- O Comando PROCESS permite a aplicação de instruções sequenciais (por exemplo, WHEN-ELSE e WITH-SELECT não são permitidos).
- Trecho entre BEGIN e END é executado sequencialmente, (a ordem importa).
- O bloco do processo é considerado como um comando único.
- Durante a simulação, o processo é disparado quando há alteração em algum sinal/variável da lista de sensibilidade.
- Diversos processos podem ser definidos numa mesma arquitetura.
- O processo como um todo é executado concorrentemente como as demais declarações ou outros processos.
- Numa sequência de atribuições ao mesmo sinal, prevalece o valor da última atribuição dentro do processo.

Comandos em VHDL – Sequenciais

"IF-THEN- END IF" e "IF-THEN-ELSE- END IF"

- Este comando permite a execução condicional de um ou mais comandos sequenciais.

- O comando **IF** inicia a lista de condições, e pode ser seguido do comando **ELSIF**, contendo também, condições a serem verificadas. Se nenhuma das condições forem verdadeiras e existir uma cláusula **ELSE**, o conjunto de comandos que segue será executado.

- Em uma cadeia de **IF ELSEs**, as condições são dispostas em uma prioridade onde o primeiro **IF** define a condição de maior prioridade.

Comandos em VHDL – Sequenciais

"IF-THEN- END IF"

```
IF condicao THEN
        -- Comandos sequenciais e/ou atribuições
END IF;
```

Será executado o que estiver dentro do bloco se a condição for verdadeira.

EXEMPLO:

```
IF A /= B THEN  -- se A é diferente de B então saída = B
    saida <= B;
END IF;
```

Comandos em VHDL – Sequenciais

"IF-THEN-ELSE- END IF"

```
IF condicao THEN
        -- Comandos sequenciais e/ou atribuições
ELSE
        -- Comandos sequenciais e/ou atribuições
END IF;
```

Se a condição for verdadeira será executado o que estiver dentro de THEN caso contrário será executado o que estiver dentro de ELSE

```
IF A = B THEN
        saida <= '0';
ELSE
        saida <= '1';
END IF;
```

Comandos em VHDL – Sequenciais

IF-THEN-ELSIF-ELSE-END IF

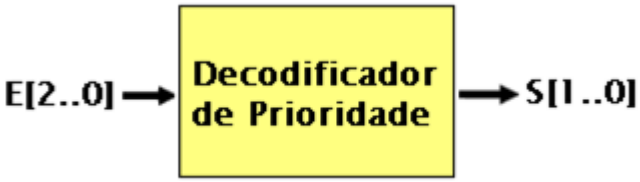
```
IF condicao_1 THEN
        -- Comandos sequenciais e/ou atribuições
ELSIF condicao_2 THEN
        -- Comandos sequenciais e/ou atribuições
ELSE
        -- Comandos sequenciais e/ou atribuições
END IF;
```

EXEMPLO:

```
IF A = B THEN
        saida0 <=´’0’;
ELSIF B < C THEN
        saida1 <= ‘0’;
ELSE -- quando B for maior ou igual a C e diferente de A
        saida1 <= ‘1’;
END IF;
```

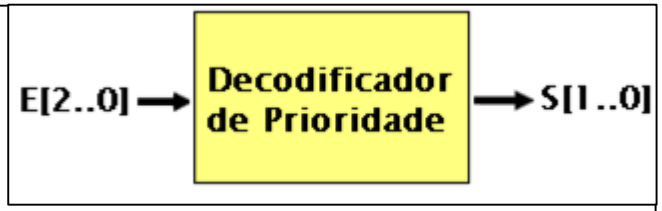
Exemplo de arquitetura de um decodificador de prioridade, com descrição comportamental utilizando estrutura: "IF-THEN-ELSE-END IF"

Nº entradas	E[2]	E[1]	E[0]	S[1]	S[0]	Nº saídas
0	0	0	0	0	0	0
1	0	0	1	0	1	1
2	0	1	0	1	0	2
3	0	1	1	1	1	2
4	1	0	0	1	1	3
5	1	0	1	1	1	3
6	1	1	0	1	1	3
7	1	1	1	1	1	3



- **se** a entrada é maior ou igual a 4 **então** S = 3
- **se** a entrada é maior ou igual a 2 e menor ou igual a 3 **então** S = 2
- **se** a entrada é igual a 1 **então** S= 1
- **se** a entrada é igual a 0 **então** S= 0

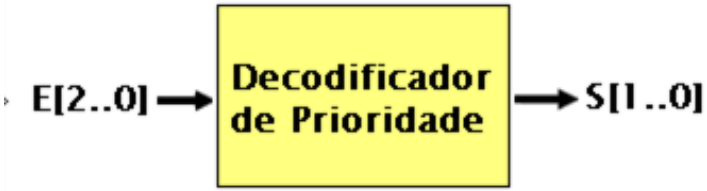
Exemplo de arquitetura de um decodificador de prioridade, com descrição comportamental utilizando estrutura: ”IF-THEN-ELSE-END IF”



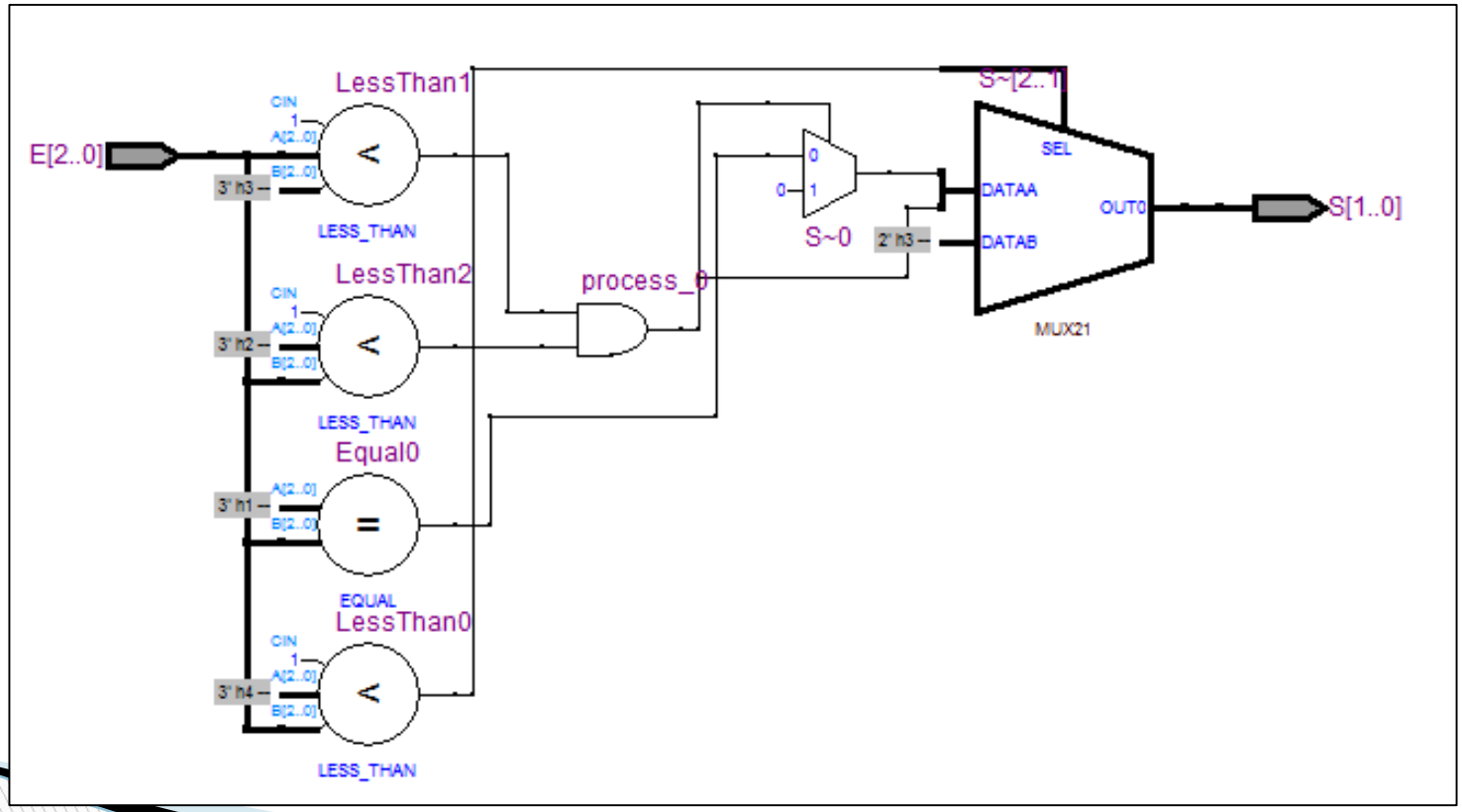

```
ENTITY dec_prior1 IS
  PORT(E : IN  BIT_VECTOR(2 DOWNTO 0);
       S : OUT BIT_VECTOR(1 DOWNTO 0));
END dec_prior1;

ARCHITECTURE comportamental OF dec_prior1 IS
BEGIN      --início da arquitetura
  PROCESS ( E ) ------------------------> Lista de sensibilidade
   BEGIN       -- início do process
    IF E >= "100" THEN  -- se a entrada é maior  ou igual a 4 então  S = 3
      S <=  "11“ ;
   ELSE -- se a entrada é maior ou igual  a  2 e menor ou igual a 3, então  S = 2
     IF   ( E <= "011")  AND ( E  >= "010")  THEN
              S <=  "10“ ;
      ELSE
         IF   E = "001"  THEN –-se a entrada é igual a 1 a saida recebe 1
            S <=  "01“ ;
         ELSE
            S <= "00"; –-se a entrada é igual a 0 a saida recebe 0
         END IF;
      END IF;
    END IF;
  END PROCESS;
END  comportamental;
```

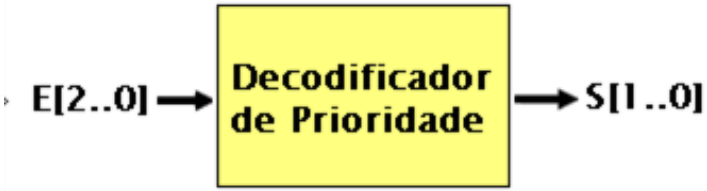
Exemplo de arquitetura do de um decodificador de prioridade, com descrição comportamental (“IF-THEN-ELSE END IF”)



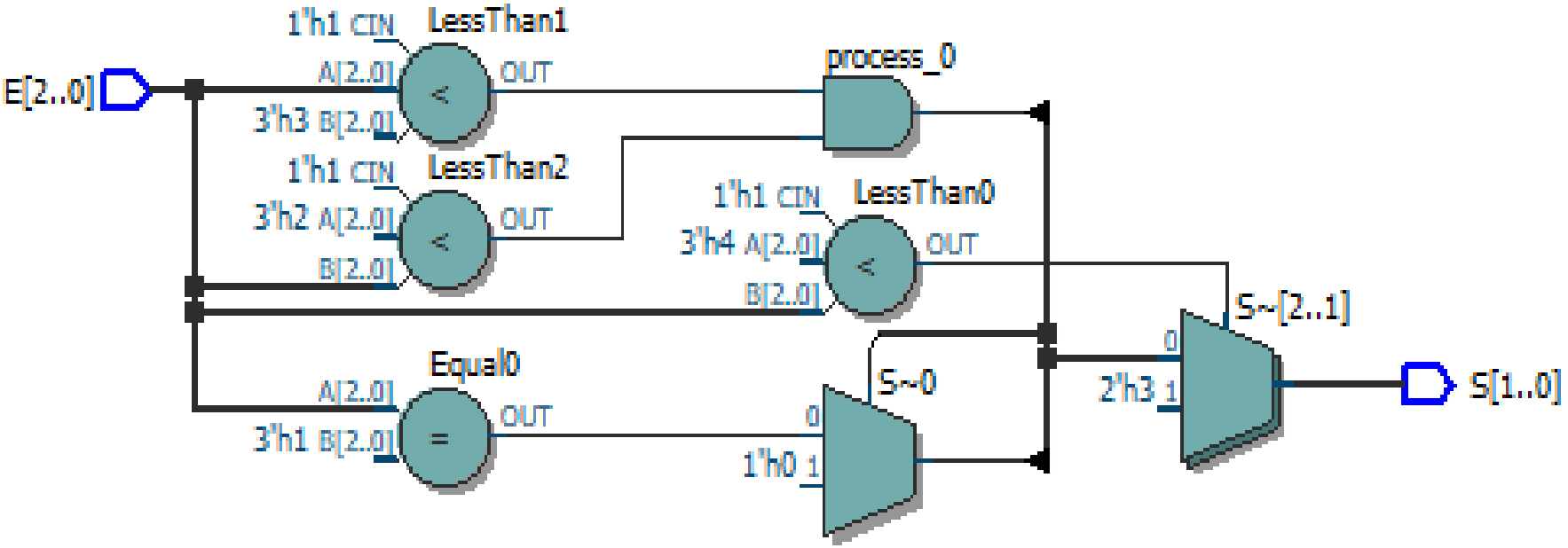
- Circuito gerado : versão Quartus 12.0



Exemplo de arquitetura do de um decodificador de prioridade, com descrição comportamental (“IF-THEN-ELSE END IF”)

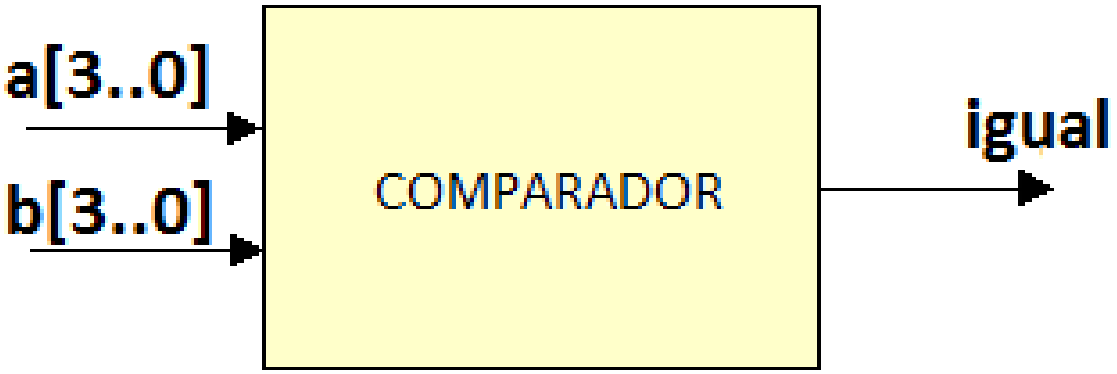


- Circuito gerado: versão Quartus 15.0



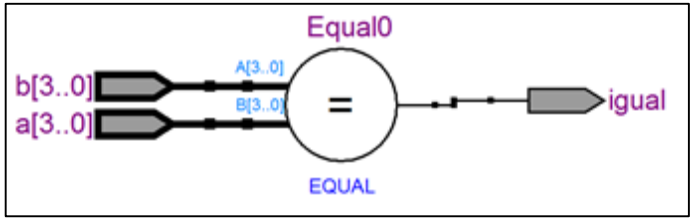
Prática nº4
Comparador de Igualdade – Descrição Comportamental

Fazer o projeto em VHDL, gerar RTL e simular:

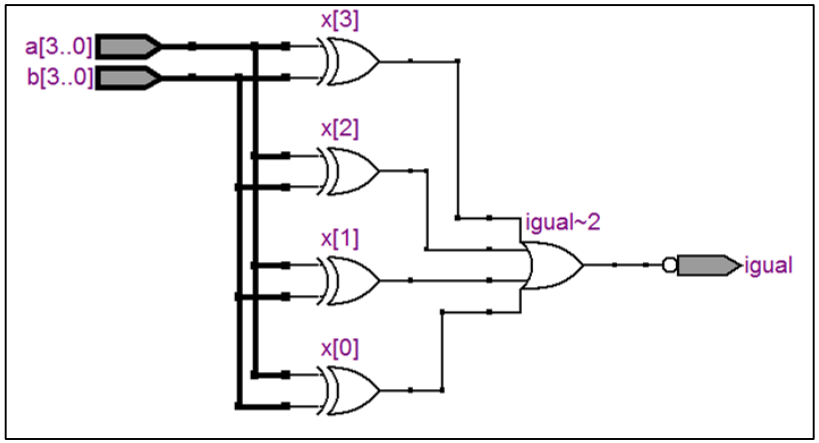


Circuitos gerados para o Comparador de Igualdade pelas diferentes descrições:

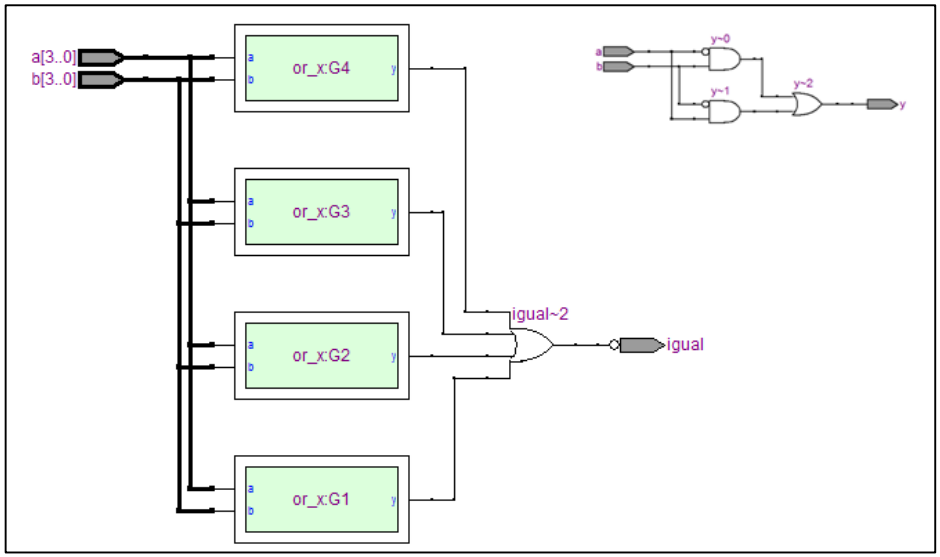
Comportamental (IF-THEN-ELSE) e Fluxo de Dados (WHEN-ELSE)



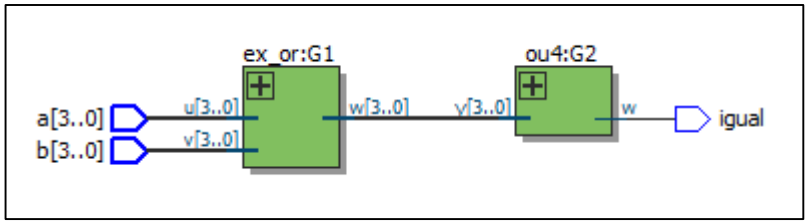
Fluxo de Dados – Expr. Lógicas



Estrutural



Estrutural com porta_xor4 e porta_or4 como componentes



Comandos em VHDL – Sequenciais
CASE-WHEN

Seleciona a execução que ocorrerá de uma lista de alternativas. É utilizado basicamente para decodificação.

```
CASE <expressão> IS
        WHEN <condição_1>          => <comando_a>;
        WHEN <condição_2>          => <comando_b>; <comando_c>;
        WHEN <condição_3> | <condição_4 > => <comando_d>;
        WHEN <condição_5> TO <condição_7> => <comando_e>;
<comando_f>;
END CASE;
```

Comandos em VHDL – Sequenciais
CASE-WHEN

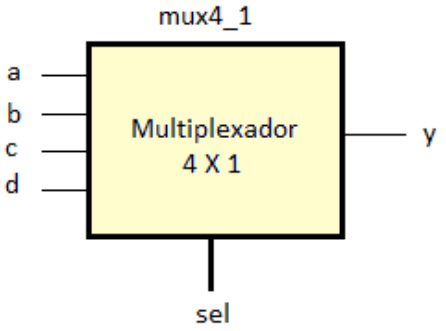
Alguns estados de entrada do seletor diferentes levando ao mesmo valor de saída:

```
CASE seletor IS  -- Existindo apenas 4 estados do seletor e para
                 -- alguns estados da entrada levam à mesma saída
        WHEN estado1_seletor =>
                saída <= saida_1;
        WHEN estado2_seletor =>
                saída <= saída_2;
        WHEN OTHERS  =>
                saída <= saída_3;
END CASE;
```

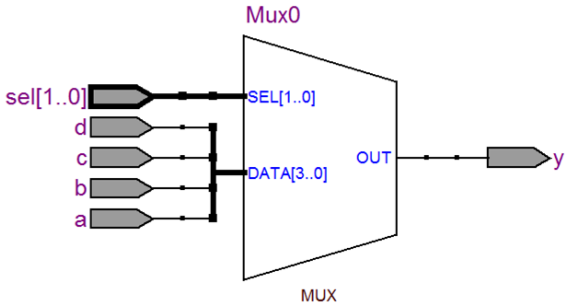
Exemplo de arquitetura de um multiplexador 4X1, com descrição comportamental utilizando estrutura: ”CASE WHEN”

```
ENTITY mux4_1 IS
        PORT ( a, b, c, d : IN BIT;
               sel        :IN BIT_VECTOR(1 DOWNTO 0);
               y          : OUT BIT);
END mux4_1;

ARCHITECTURE comportamental OF mux4_1 IS
BEGIN
   PROCESS ( sel,a,b,c,d ) —lista de sensibilidade
   BEGIN
      CASE sel IS
         WHEN "00" => y <= a;
         WHEN "01" => y <= b;
         WHEN "10" => y <= c;
         WHEN "11" => y <= d;
      END CASE;
   END PROCESS;
END comportamental;
```



seleção		Y
0	0	a
0	1	b
1	0	c
1	1	d



Circuito sintetizado

Comandos em VHDL
Comparação entre WHEN-ELSE e IF-ELSE

Ambos os comandos levam em conta a prioridade das condições de seleção. Permitem a omissão de possibilidades, e a primeira condição válida detectada no conjunto de condições especificadas é a escolhida.

WHEN-ELSE: Usada em regiões de **código concorrente**. A operação executada é a transferência de um valor para um único sinal.

IF-ELSE: Usada em regiões de **código sequencial**. É mais flexível, pois permite a execução de múltiplos comandos sequenciais.

Comandos em VHDL

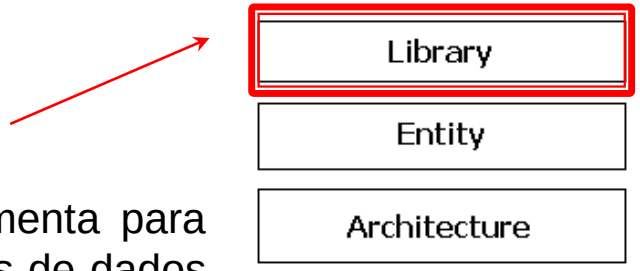
Comparação entre WITH-SELECT e CASE-WHEN

Têm como similaridade o fato de que todos os valores possíveis da expressão de seleção devem ser apresentados.

WITH-SELECT: Usada em regiões de **código concorrente**. Transfere um valor para um único sinal.

CASE-WHEN: Usada em regiões de **código sequencial**. Permite a execução de múltiplos comandos sequenciais.

LIBRARY



Bibliotecas ou Library : São diretórios criados pela ferramenta para compilação e simulação , no qual existem definições de tipos de dados e unidades de projetos compilados . Quando deseja-se utilizar uma biblioteca em um projeto, antes da entidade é referenciada a biblioteca através da cláusula (LIBRARY (nome);), e em seguida o pacote utilizado (partes da biblioteca), através da cláusula (USE nome_do_pacote.ALL;)

Ex:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
```

Observações:

1.Caso não seja declarada a biblioteca, é utilizada a biblioteca do "*WORK* " e "STANDARD" da ferramenta de síntese.

2. evitar usar biblioteca não padronizada pois podem levar a falta de portabilidade do código em diferentes ferramentas de sintese e simulação.

LIBRARY IEEE (padronizadas)

Pacotes:

- IEEE.std_logic_1164 : contempla somente operações lógicas

  Definição do tipo STD_LOGIC

  Declaração: LIBRARY ieee;

  USE ieee. std_logic_1164.ALL;

- IEEE.numeric_std : permite operações aritméticas em sinais do tipo signed (ou unsigned)

  Declaração: LIBRARY ieee;

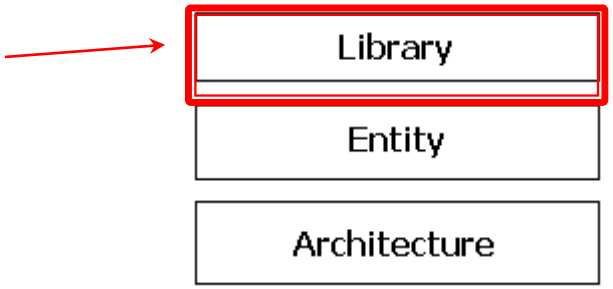
  USE ieee.numeric_std.ALL;

- IEEE.std_logic_arith : funções de conversão de tipos

  Declaração: LIBRARY ieee;

  USE ieee.std_logic_arith.ALL;

LIBRARY IEEE (padronizadas)



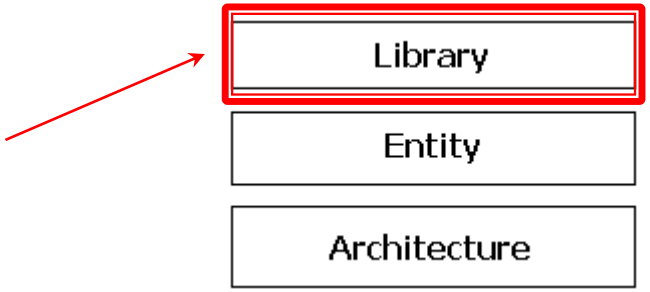
Pacotes (continuação):

- IEEE.std_logic_unsigned: permitem operações com dados std_logic_vector do tipo unsigned (somar vetores apenas com operador +)

- IEEE.std_logic_signed : permitem operações com dados std_logic_vector do tipo signed.

```
LIBRARY ieee;
USE ieee.std_logic_unsigned.ALL; --Se for unsigned
ou
USE ieee.std_logic_signed.ALL; -- No caso de signed
```

LIBRARY



Pacotes mais utilizados da biblioteca IEEE:

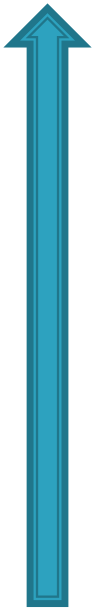
LIBRARY IEEE;	Usada para incluir a biblioteca IEEE;
USE IEEE.std_logic_1164.ALL;	Define o tipo STD_LOGIC (VECTOR), usado em descrições mais fiéis a um circuito real.
USE IEEE.std_logic_arith.ALL;	Define os tipos SIGNED e UNSIGNED.
USE IEEE.std_logic_signed.ALL;	Permite tratar sinais do tipo STD_LOGIC_VECTOR como inteiros com sinal (define funções aritméticas, de comparação, etc).
USE IEEE.std_logic_unsigned.ALL;	Permite tratar sinais do tipo STD_LOGIC_VECTOR como inteiros sem sinal (define funções aritméticas, de comparação, etc).
USE IEEE.numeric_std.ALL;	Define os tipos UNSIGNED e SIGNED como matriz de std_logic

LIBRARY - Tipos mais utilizados

Biblioteca IEEE: Tipos STD_LOGIC e STD_LOGIC_VECTOR

Cláusula: LIBRARY ieee
USE ieee.std_logic_1164.ALL;

Precedência



Valor		Estado Lógico	
U		Não Inicializado	
X		Desconhecido Forte	
0	1	Nível Baixo Forte	Nível Alto Forte
W		Desconhecido Fraco	
L	H	Nível Baixo Fraco	Nível Alto Fraco
Z		Alta Impedância	
–		Não Importa	

Valores "STD_LOGIC"	U	X	0	1	Z	W	L	H	–
Valores codificados	X	X	O	1	Z	X	0	1	X

OBS: A biblioteca WORK é inclusa automaticamente no projeto VHDL.

Exemplo de atribuição corretas

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
```

```
a <= b(5); -- atribuição de bit de um sinal tipo BIT_VECTOR a um sinal tipo BIT
b(0) <= a; -- atribuição de um sinal tipo BIT a um bit de um sinal do tipo BIT_VECTOR
c <= d(5); -- atribuição de bit de um sinal tipo STD_LOGIC_VECTOR a um sinal tipo STD_LOGIC
d(0) <= c; -- atribuição de um sinal tipo STD_LOGIC a um bit de um sinal do tipo STD_LOGIC_VECTOR
```

Exemplo de atribuição incorretas

```
a <= c;   -- tipos diferentes: BIT x STD_LOGIC
b <= d;   -- tipos diferentes: BIT_VECTOR x STD_LOGIC_VECTOR
e <= b;   -- tipos diferentes: INTEGER x BIT_VECTOR
e <= d;   --tipos diferentes: INTEGER x STD_LOGIC_VECTOR
```

Tipos em VHDL (continuação) :

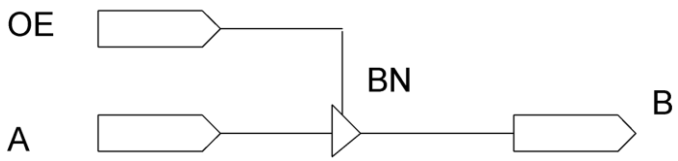
CONVERSÃO ENTRE TIPOS:

DESCRIÇÃO VHDL	*FUNÇÃO*	Biblioteca
Inteiro <= conv_integer (vetor);	Converte um vetor em inteiro	std_logic_arith.
Vetor <= std_logic_vector (to_unsigned(inteiro, nºbits));	Converte um inteiro em vetor	numeric_std
Inteiro<=to_integer(unsigned(vetor_std_logic_vector))	Converte um vetor em inteiro	numeric_std
Inteiro<=to_integer(signed(vetor_std_logic_vector))	Converte um vetor em inteiro	numeric_std
-- Dentro de um process <variável_tipo_std_ulogic> := To_StdUlogic(<variável_tipo_bit>);	Converte bit em std_logic	std_logic_1164.
-- Dentro de um process <variável_tipo_std_logic_vector>:= to_stdLogicVector(<variável_tipo_bit_vector>);	Converte std_logic_vector em bit_vector	std_logic_1164

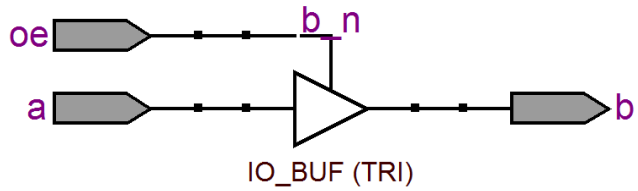
Observação: Não é permitida a transferência de valores entre objetos de tipos diferentes.

programa completo: CIRCUITO TRI-STATE

```
LIBRARY ieee; --  cláusula da biblioteca
USE ieee.std_logic_1164.ALL;  --  cláusula USE
ENTITY tri_state IS -- Declaração da entidade
  PORT ( a, oe : in std_logic;
                b      : out std_logic);
END tri_state;
--  declaração da arquitetura
ARCHITECTURE  a OF tri_state IS
       SIGNAL  b_n : std_logic;
BEGIN
    PROCESS (a,oe)
        BEGIN
            IF (oe = '1' ) then
                b_n <= a;
            ELSE
                    b_n <= 'Z' ;
            end if;
    END PROCESS;
            b <= b_n;
END  a ;
```

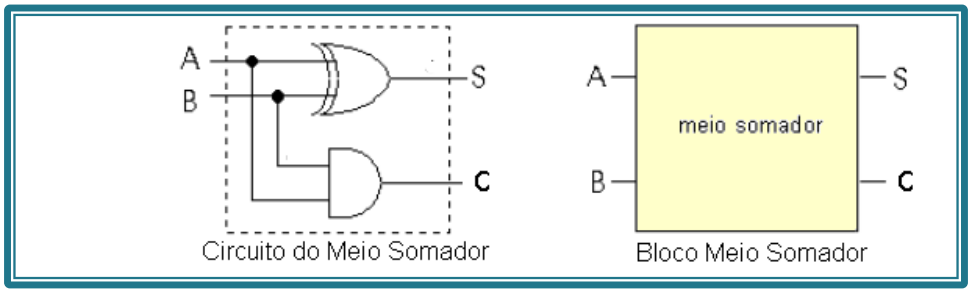


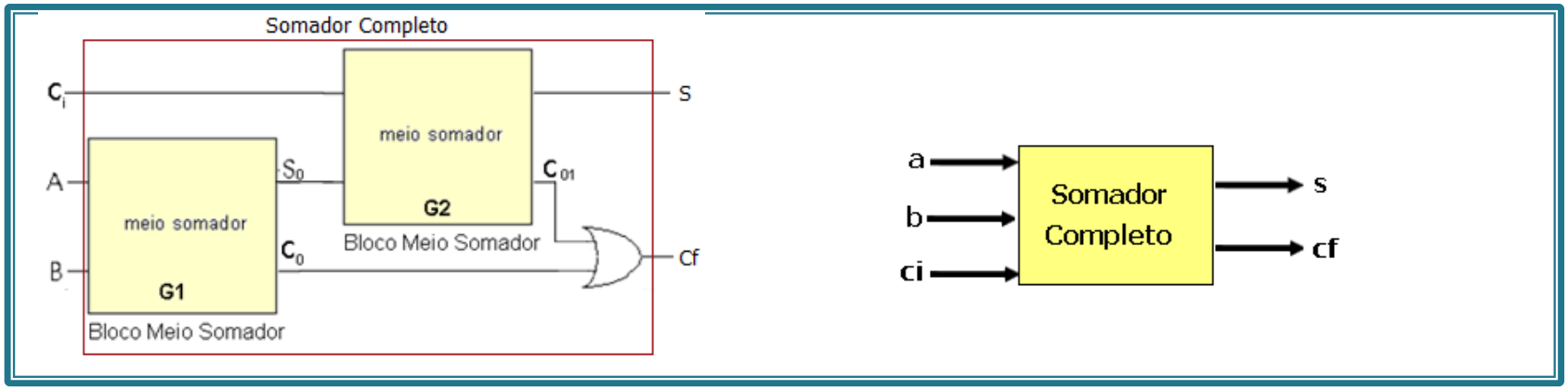
Circuito sintetizado:



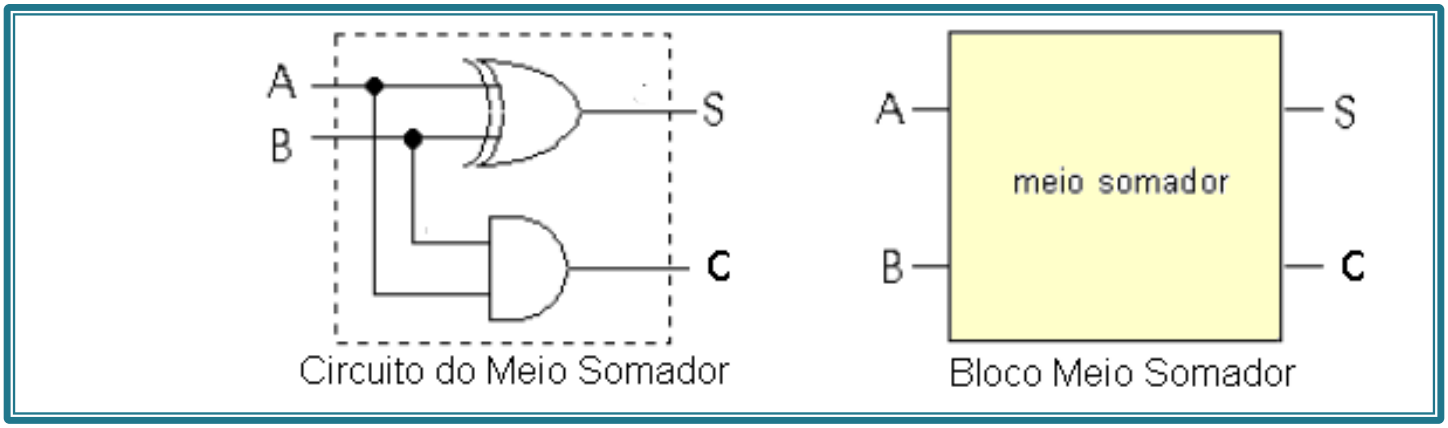
Junção de Projetos

Criar um projeto de um meio somador em uma pasta, em seguida criar outro projeto de um somador completo utilizando o projerto do meio somador como componente em pastas separadas. Utilize a biblioteca ieee.std_logic_1164, ou seja, o tipo de sinais deve ser STD_LOGIC e de vetores STD_LOGIC_VECTOR





Projeto do meio somador em VHDL



Projeto do bloco meio somador:

Iniciar um novo projeto meio_somador o qual estará numa pasta chamada meio_somador. Compilar. Fechar projeto (File Close Project)

s = a ⊕b

cf = a.b

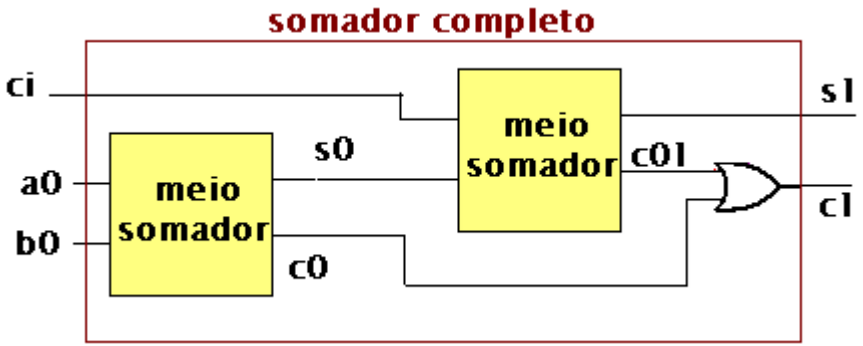
a	b	s	cf
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

```
-- nome da entidade mesmo nome do arquivo(do projeto)
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY meio_somador IS
  PORT( a,b :IN  STD_LOGIC;
              s,cf :OUT STD_LOGIC);
END meio_somador;
ARCHITECTURE a OF meio_somador IS
BEGIN
    s  <= a XOR b;
    cf <= a AND b;
END a;
```

Exemplo de Arquitetura de um Somador Completo – Descrição Estrutural: Usando Meio-Somadores

Um somador completo pode ser implementado utilizando blocos meio-somadores:

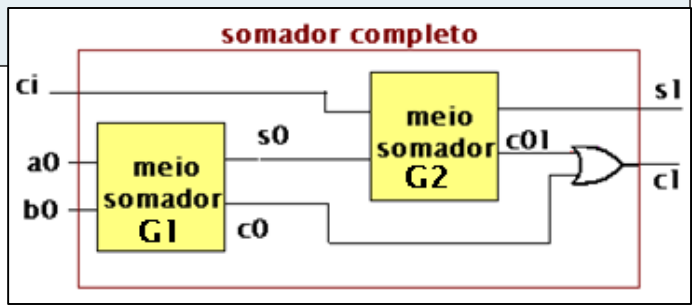
- ➔ Inicie um outro projeto através de New Project Wizard e crie uma pasta somador_completo e um arquivo VHDL somador_completo.
- ➔ Nesta pasta somador_completo, faça uma cópia da pasta meio_somador.
- ➔ Escreva a descrição do somador_completo utilizando o comando PORT MAP considerando o projeto Meio_somador como componente. Para isso, acrescente o projeto meio_somador em:
- ➔ Project ⟶ Add/Remove Files in Project...
- ➔ Uma janela de busca de projeto vai se abrir. Procurar a pasta meio_somador (dentro da pasta somador_completo e clicar em **add** e **Apply.**
- ➔ Desta forma o projeto meio_somador está adicionado à pasta do projeto somador_completo. E o arquivo VHDL do meio_somador pode ser aberto e até modificado. Ambos projetos serão compilados juntos.
- ➔ Faça uma cópia da entidade do meio somador e a declare no projeto somador_completo como um componente



Descrição estrutural do somador completo usando blocos meio somadores:

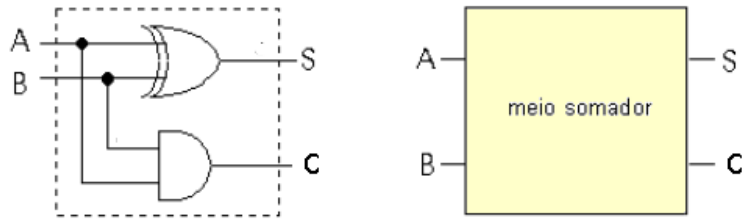
Observação: a descrição VHDL do projeto meio_somador consta na pasta separada desse projeto e não na descrição do somador_completo

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL; -- declaração da bibiloteca e pacote
-- nome da entidade principal mesmo nome do arquivo(do projeto)
ENTITY somador IS
                PORT( a0,b0,ci :IN STD_LOGIC;
                        s1,c1 :OUT STD_LOGIC);
END somador;
-- criando a arquitetura do somador completo utilizando blocos meio somadores como componentes
ARCHITECTURE estrutural OF somador IS
--declarando os sinais internos ao projeto meio-somador
    SIGNAL s0, c0, c01: STD_LOGIC;
-- declarando o projeto do  meio_somador como um componente
COMPONENT meio_somador IS
                PORT( a,b :IN STD_LOGIC;
                        s,cf :OUT STD_LOGIC);
END COMPONENT;
-- inicio da arquitetura do projeto somador
  BEGIN
            G1: meio_somador PORT MAP (a0, b0, s0, c0) ;
            G2: meio_somador PORT MAP (a =>ci,  b=>s0,  s=>s1, cf => c01) ;
            c1 <= c0 OR  c01;
 END estrutural ;
```

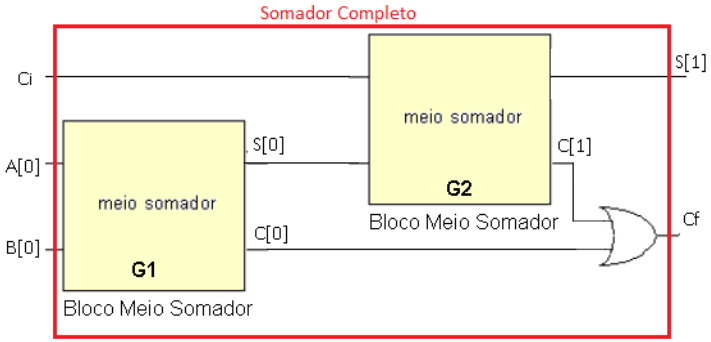


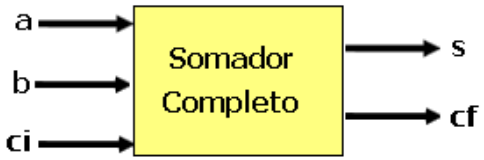
Prática nº5

5.1

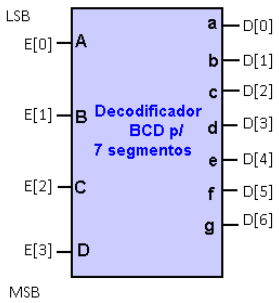


5.2

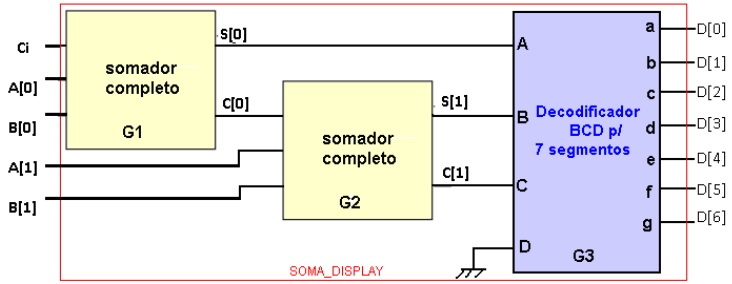




5.3



5.4



RESOLUÇÃO DAS PRÁTICAS

Exemplo de Arquitetura de um Somador Completo – Descrição por Fluxo de Dados
Usando Expressões Lógicas

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY somador IS
   PORT(ci, a, b : IN  STD_LOGIC;
        s, cf : OUT STD_LOGIC);
END somador;
ARCHITECTURE fluxo_dados OF somador IS
   BEGIN
      s <= (a XOR b XOR ci);
      cf <= (a AND b) OR (ci AND (a OR b ));
END fluxo_dados;
```

Exemplo de Arquitetura de um Somador Completo – Descrição por Fluxo de Dados
Usando Comando Concorrente WHEN-ELSE

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY somador IS
   PORT(vem_1, a, b : IN  STD_LOGIC;
                s, vai_1      : OUT STD_LOGIC);
END somador;
ARCHITECTURE fluxo_dados OF somador IS
BEGIN
                s <= '1' WHEN (a = '0' AND b = '1' AND ci = '0') ELSE
                '1' WHEN (a = '1' AND b = '0' AND ci = '0') ELSE
                '1' WHEN (a = '0' AND b = '0' AND ci = '1') ELSE
                '1' WHEN (a = '1' AND b = '1' AND ci = '1') ELSE
                '0';
                co <= '1' WHEN (a = '1' AND b = '1' AND ci = '0') ELSE
                '1' WHEN (a = '0' AND b = '1' AND ci = '1') ELSE
                '1' WHEN (a = '1' AND b = '0' AND ci = '1') ELSE
                '1' WHEN (a = '1' AND b = '1' AND ci = '1') ELSE
                '0';
END fluxo_dados;
```

RESOLUCÃO DAS PRÁTICAS

Prática nº4 Comparador com porta ex_xor e porta ou4 comco componentes- Resolução

```
ENTITY comparador IS
                    PORT ( a,b: IN BIT_VECTOR( 3 DOWNTO 0);
                           igual  : OUT BIT);
END comparador;

ARCHITECTURE a OF comparador  IS
SIGNAL x :  BIT_VECTOR( 3 DOWNTO 0);
COMPONENT ex_or IS
 PORT ( u,v : IN BIT_VECTOR(3 DOWNTO 0);
          w : OUT BIT_VECTOR(3 DOWNTO 0));
END COMPONENT;

COMPONENT ou4 IS
 PORT( y  : IN BIT_VECTOR(3 DOWNTO 0);
         w : OUT BIT);
END COMPONENT;

BEGIN
                    G1: ex_or  PORT MAP ( a, b, x);
                    G2 : ou4   PORT MAP (x, igual);

END a;
ENTITY ex_or IS
 PORT( u,v : IN BIT_VECTOR(3 DOWNTO 0);
         w : OUT BIT_VECTOR(3 DOWNTO 0) );
END ex_or;
ARCHITECTURE a OF ex_or IS
BEGIN
w <= (((NOT u) AND v) OR ((NOT u) AND v));
END a;
---
ENTITY ou4 IS
 PORT ( y : IN BIT_VECTOR(3 DOWNTO 0);
         w : OUT BIT );
END ou4;
ARCHITECTURE a OF ou4 IS
BEGIN
w <= y(0) OR y(1) OR y(2) OR y(3);
END a;
```

Prática nº4- Resolução

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY LED_MSD_DISPLAY  IS
  PORT(MSD  : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
   MSD_7SEG : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END LED_MSD_DISPLAY;
ARCHITECTURE a OF LED_MSD_DISPLAY IS -- Decodificador BCD para display de 7 segmentos
BEGIN
PROCESS (MSD)
BEGIN
  CASE MSD IS
  WHEN "0000" =>              MSD_7SEG <= "0000001"; --abcdefg
  WHEN "0001" =>           MSD_7SEG <= "1001111";
  WHEN "0010" =>           MSD_7SEG <= "0010010";
  WHEN "0011" =>           MSD_7SEG <= "0000110";
  WHEN "0100" =>           MSD_7SEG <= "1001100";
  WHEN "0101" =>           MSD_7SEG <= "0100100";
  WHEN "0110" =>      MSD_7SEG <= "0100000";
  WHEN "0111" =>           MSD_7SEG <= "0001111";
  WHEN "1000" =>           MSD_7SEG <= "0000000";
  WHEN "1001" =>           MSD_7SEG <= "0001100";
--pode-se optar por fazer decodificar bcd para hexa e então
-- coloca-se todas as combinações das entradas sem o WHEN OTHERS
  WHEN OTHERS =>
              MSD_7SEG <= "0110000";
  END CASE;
END PROCESS;
END a;
```

Prática nº5- Resolução

```
LLIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY display_b IS
    PORT(a0, b0, ci, a1, b1 : IN  BIT;
   sete_seg                 : OUT BIT);
END display_b ;
ARCHITECTURE estrutural OF display_b r IS
SIGNAL s0,c0,s1,c1:  STD_LOGIC;
-- declaração do componente somador
COMPONENT somador IS
                 PORT (a, b, ci : IN STD_LOGIC;
                s, co : OUT STD_LOGIC);
END COMPONENT;
COMPONENT LED_MSD_DISPLAY  IS
  PORT(MSD : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                MSD_7SEG : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END COMPONENT;
BEGIN
 G1: somador PORT MAP (a0,b0,ci,s0,c0);
 G2: somador PORT MAP (a1,b1,c0,s1,c1);
 G3:LED_MSD_DISPLAY PORT
MAP(MSD(0)=>s0,MSD(1)=>s1,MSD(2)=>c1,MSD(3)=>'0',MSD_7SEG =>sete_seg);
END estrutural;
```