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 do circuito de forma abstrata usando o comando PROCESS.

O Comando PROCESS permite a aplicação de instruções sequenciais e apresenta o formato:

```
PROCESS (<Lista de Sensibilidade>)
BEGIN
    <Descrição Lógica>
END PROCESS;
```

A *lista de sensibilidade* corresponde aos sinais que devem alterar a saída do circuito, e é composta de todos os sinais de entrada para os circuitos combinatórios. Para os registradores assíncronos, a lista seria composta do clock e do reset; e para os registradores síncronos, do clock.

ARCHITECTURE – Comportamental

O comportamento, ou funcionalidade, de um sistema corresponde a uma lista de operações a serem executadas sequencialmente (Comandos Sequenciais) para se obter um determinado resultado.

PROCESS: O modo formal de se fazer uma lista sequencial de operações. Estrutura:

```
<Nome_opcional>: PROCESS(sensibilidade_sinal_1, ...)
BEGIN
    -- Comandos sequenciais do processo
END PROCESS <Nome_opcional>;
```

Lista de sensibilidade: Devem constar variáveis e sinais cuja alteração deve levar à reavaliação da saída. Sinais com inicialização assíncrona devem constar obrigatoriamente na lista. Sinais síncronos não necessariamente.

ARCHITECTURE – Comportamental

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

- Trecho entre BEGIN e END é executado sequencialmente, (a ordem importa).
- O bloco do processo é considerado com um comando único.
- Diversos processos podem ser definidos numa arquitetura.
- O processo como um todo é executado concorrentemente como as demais declarações ou outros processos.

ARCHITECTURE – Comportamental

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

- Durante a simulação, o processo é disparado quando há alteração em algum sinal/variável na lista de sensibilidade.
- Apenas comandos sequenciais podem ser inseridos em um processo (por exemplo, WHEN-ELSE e WITH-SELECT não são permitidos).
- Numa sequência de atribuições ao mesmo sinal, prevalece o valor da última atribuição.

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 seqüenciais.
- 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" e "IF-THEN-ELSE- 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
    saida <= B;
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

EXEMPLO:

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

Comandos em VHDL – Sequenciais
IF-THEN-ELSIF-ELSE-END IF

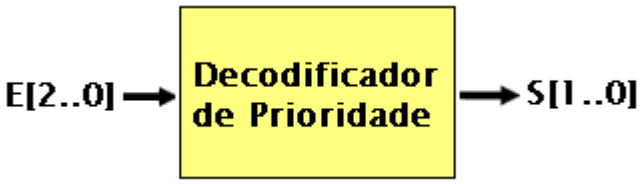
```
IF condicao_1 THEN
        -- Comandos sequenciais e/ou atribuições
ELSIF condicao_2 THEN
        -- Comandos sequenciais e/ou atribuições
ELSE
        -- Comandos sequenciais e/ou atribuições
END IF;
```

EXEMPLO:

```
IF A = B THEN
        saida0 <=´’0’;
ELSIF B < C THEN
        saida1 <= ‘0’;
ELSE
        saida1 <= ‘1’;
END IF;
```

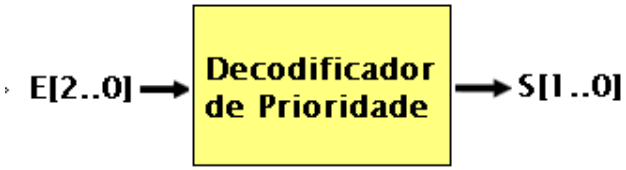
Exemplo de arquitetura de um decodificador de prioridade, com descrição comportamental utilizando estrutura :"IF-THEN-ELSE-END IF"

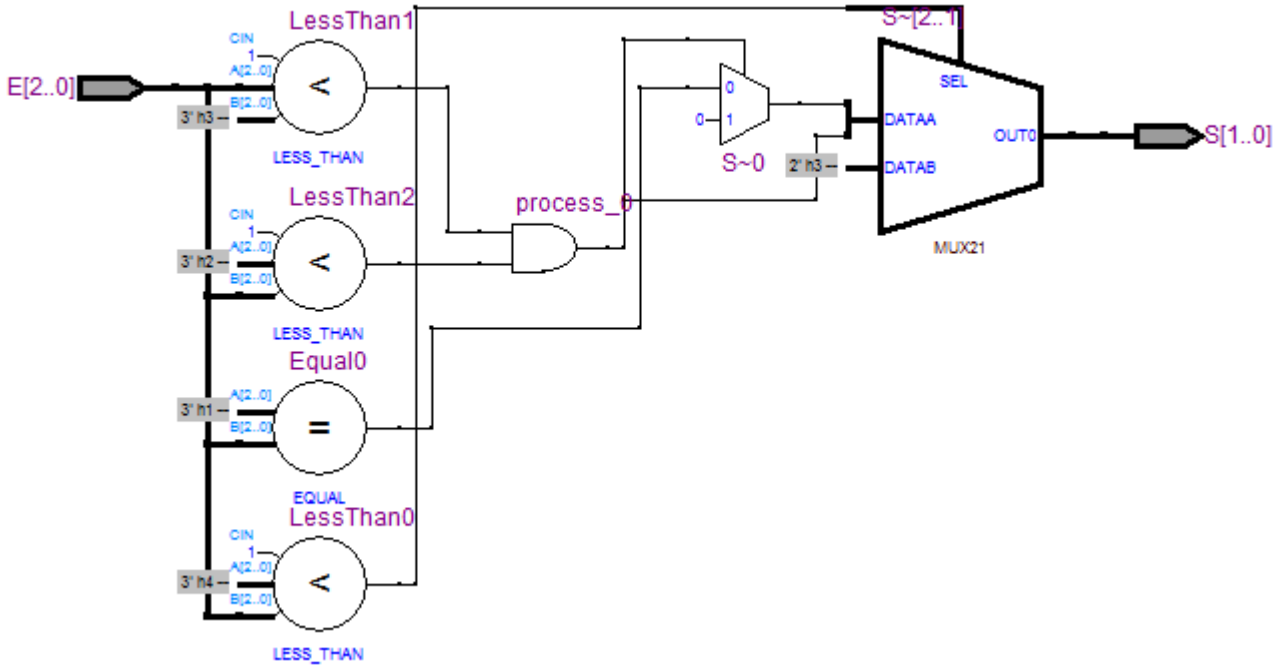
```
ENTITY dec_prior1 IS
  PORT(E : IN  BIT_VECTOR(2 DOWNTO 0);
       S : OUT BIT_VECTOR(1 DOWNTO 0));
END dec_prior1;

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;
```



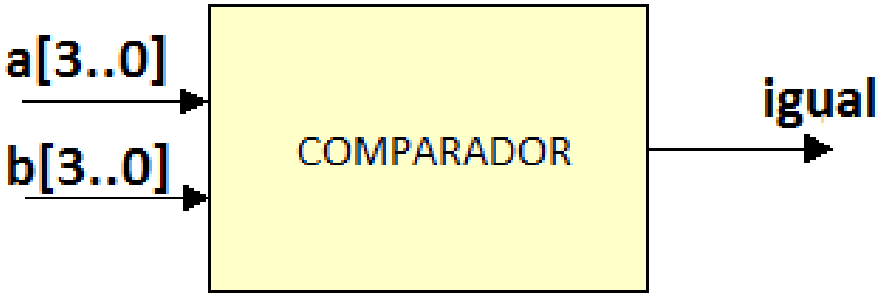
Exemplo de arquitetura do de um decodificador de prioridade, com descrição comportamental (“IF-THEN-ELSE END IF”)

- Circuito gerado



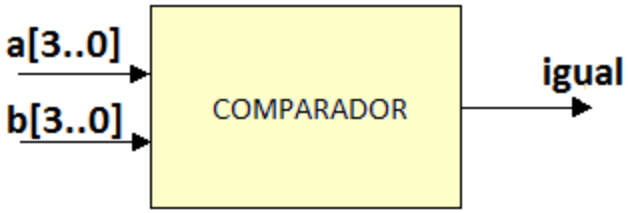


Prática nº7
Comparador e Igualdade – Descrição Comportamental



Exemplo de Arquitetura de um Comparador de Igualdade – Descrição Comportamental Usando Comando Sequencial IF-THEN-ELSIF-ELSE- END IF

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

BEGIN -- Início da Arquitetura
    PROCESS(a, b) -- Lista de Sensibilidade
    BEGIN -- Início do Process
        IF (a = b) THEN igual <= '1';
        ELSE igual <= '0';
        END IF;
    END PROCESS;
END comportamental;
```

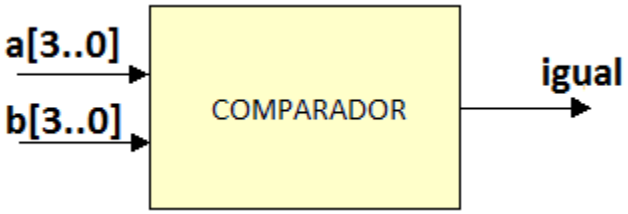


O comando PROCESS(a, b) indica que os sinais **a** e **b** formam a *lista de sensibilidade*.

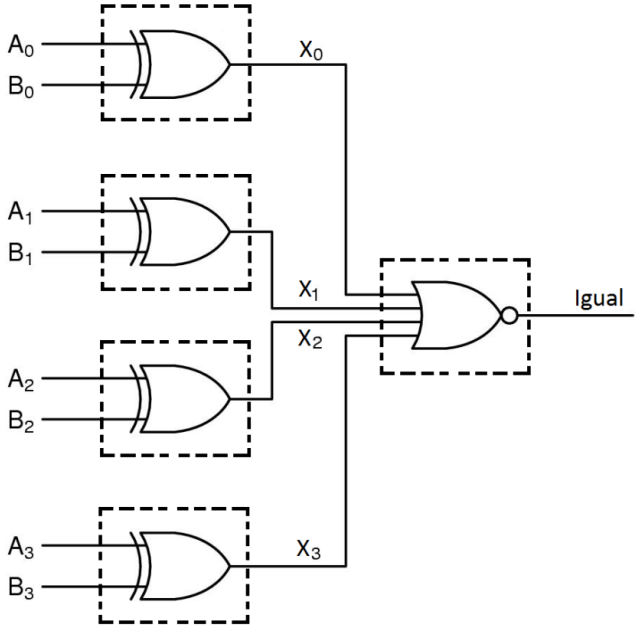
A saída **igual** será igual a '1' caso as entradas **a** e **b** sejam iguais, e será igual a '0', caso contrário.

Exemplo de Arquitetura de um Comparador de Igualdade – Descrição por Fluxo de Dados Usando Comando Concorrente WHEN-ELSE

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

ARCHITECTURE fluxo_dados OF comparador IS

BEGIN -- Início da Arquitetura

    igual <= '1' WHEN (a = b) ELSE
             '0';
END fluxo_dados;
```



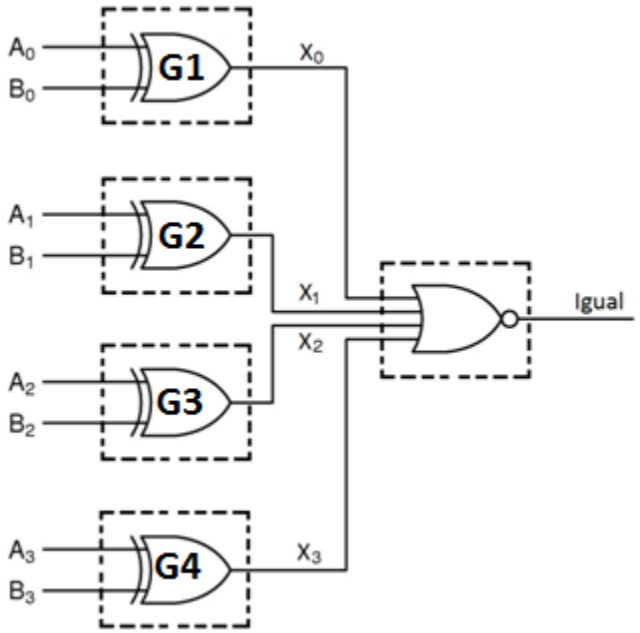
Exemplo de Arquitetura de um Comparador de Igualdade – Descrição por Fluxo de Dados Usando Expressões Lógicas




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

ARCHITECTURE logica OF comparador IS
    SIGNAL x : BIT_VECTOR(3 DOWNTO 0);
BEGIN -- Início da Arquitetura
     x <= a XOR b;
    igual <= NOT (x(0) OR x(1) OR X(2) OR x(3));
END logica;
```

Exemplo de Arquitetura de um Comparador de Igualdade – Descrição Estrutural

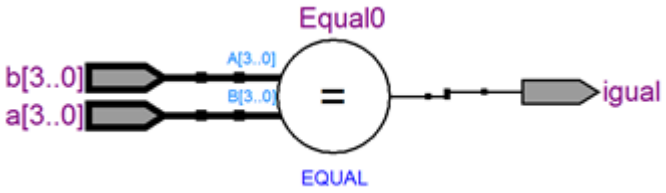
```
ENTITY or_x IS
  PORT(a, b : IN  BIT;
       y    : OUT BIT);
END or_x;
ARCHITECTURE fluxo_dados OF or_x IS
BEGIN
  y <= (NOT a AND  b) OR ( a AND NOT b);
END fluxo_dados;
```



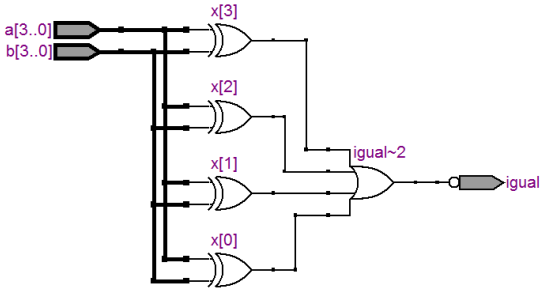

```
ENTITY comparador IS
  PORT (a, b  : IN  BIT_VECTOR(3 DOWNTO 0);
        igual : OUT BIT);
END comparador;

ARCHITECTURE  estrutural  OF   comparador IS
  SIGNAL x :  BIT_VECTOR( 3 DOWNTO 0);
  COMPONENT or_x IS
    PORT (a, b : IN  BIT;
          y    : OUT BIT);
  END COMPONENT;
BEGIN -- Início da Arquitetura
  G1 : or_x PORT MAP(a(0), b(0),  x(0));
  G2 : or_x PORT MAP(a(1), b(1),  x(1));
  G3 : or_x PORT MAP(a(2), b(2),  x(2));
  G1 : or_x PORT MAP(a(3), b(3),  x(3));
 igual <= NOT(x(0) OR x(1) OR X(2) OR x(3));
END estrutural;
```

Circuitos gerados para o Comparador de Igualdade pelas diferentes descrições

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

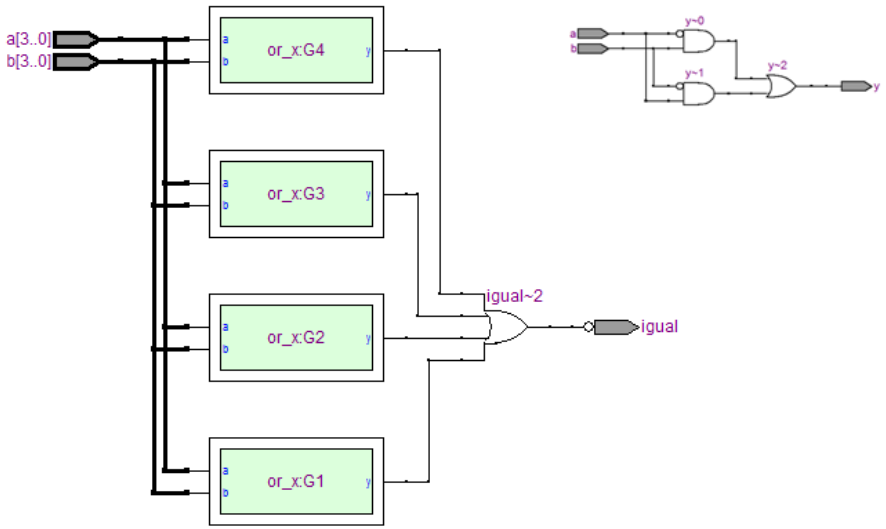


Fluxo de Dados – Expr. Lógicas



Flow Status	Successful - Thu May 01 20:43:51 2014
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	comparador
Top-level Entity Name	comparador
Family	MAX7000S
Device	EPM7128SLC84-7
Timing Models	Final
Met timing requirements	Yes
Total macrocells	2 / 128 (2 %)
Total pins	13 / 68 (19 %)

Estrutural



Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição

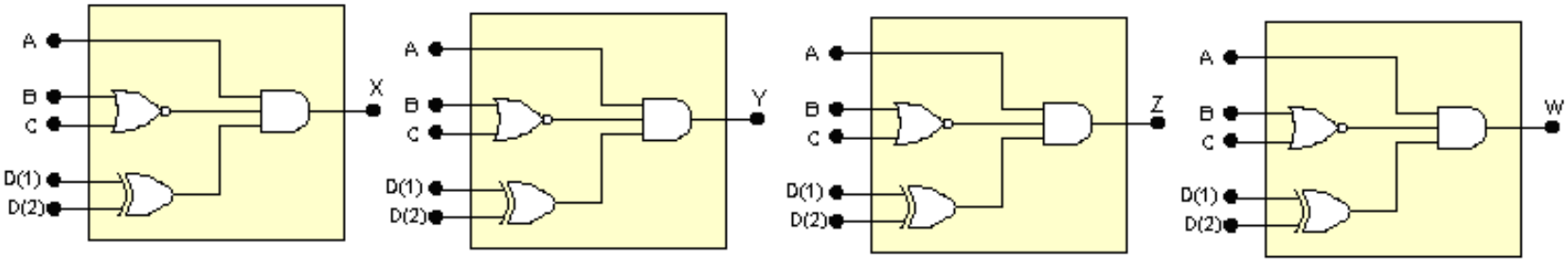


Figura 1 Arranjo de portas lógicas

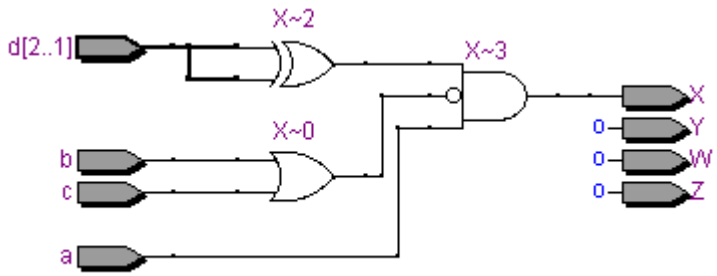
Tabela 1

D(2)	D(1)	C	B	A	Y
L	H	L	L	H	H
H	L	L	L	H	H

Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição

Item a:

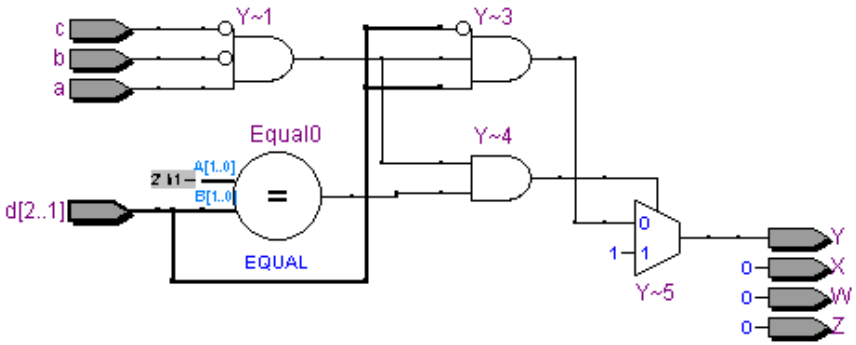
```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  X, Y, W, Z    : OUT BIT );
END pratica8;

-- Modelo por Fluxo de Dados - EXPRESSÃO LÓGICA

ARCHITECTURE  fluxo_dados  OF pratica8 IS

BEGIN

   X<= A AND NOT(B OR C)AND ( D(2) XOR D(1));

END fluxo_dados;
```



Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição

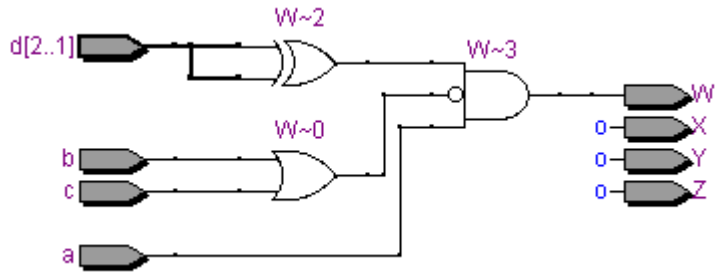
Item b:

```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  X, Y, W, Z    : OUT BIT );
END pratica8;

-- Modelo por Fluxo de Dados – Comando concorrente : WHEN ELSE

ARCHITECTURE  fluxo_dados  OF pratica8 IS

BEGIN

   y <=     '1' WHEN (a='1' AND  b='0' AND c='0' AND d = "01") ELSE
            '1' WHEN (a='1' AND  b='0' AND c='0' AND d(1)= '1' AND d(2)='0') ELSE
             '0';
END fluxo_dados;
```



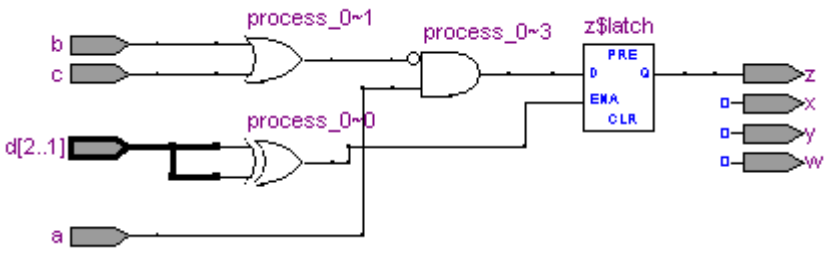
Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição
Item c:

```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  X, Y, W, Z    : OUT BIT );
END pratica8;

-- Modelo comportamental – usando EXPRESSÃO LÓGICA

ARCHITECTURE  fluxo_dados  OF pratica8 IS
BEGIN

PROCESS (a,b,c,d)
BEGIN
        w <= A AND NOT(B OR C)AND ( D(2) XOR D(1));
END PROCESS;


END fluxo_dados;
```



Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição
Item d (1ª. Maneira):

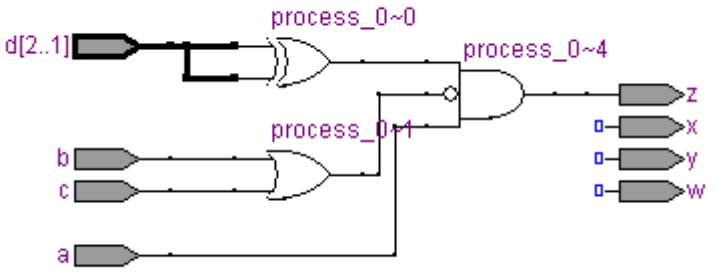
```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  x,y, w, z     : OUT BIT );
END pratica8;

-- Modelo comportamental – usando EXPRESSÃO LÓGICA

ARCHITECTURE  comportamental  OF pratica8 IS
BEGIN

PROCESS (a,b,c,d)
BEGIN
        IF (d(1)/=d(2)) THEN
            IF ( (b  OR c='0') AND (a = '1' ) )THEN
                  z<= '1';
            ELSE
                  z <= '0';
            END IF;
        END IF;
END PROCESS;

END comportamental;
```



Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição
Item d (2ª. Maneira):

:

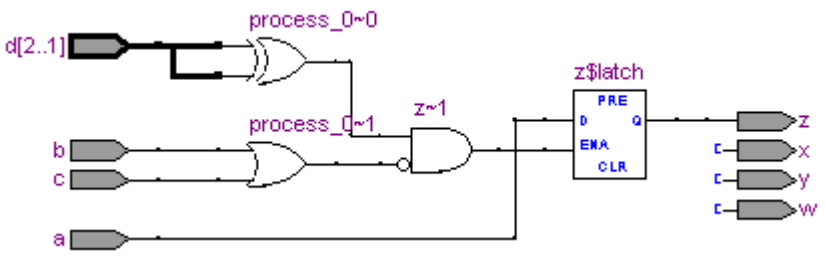
```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  x,y, w, z     : OUT BIT );
END pratica8;

-- Modelo comportamental – usando EXPRESSÃO LÓGICA

ARCHITECTURE  comportamental  OF pratica8 IS
BEGIN

PROCESS (a,b,c,d)
BEGIN
        IF (d(1)/=d(2)) AND ( (b  OR c='0') AND (a = '1' ) )THEN
                z<= '1';
           ELSE
                z <= '0';
        END IF;
END PROCESS;

END comportamental;
```



Resposta: Prática nº8
Arranjos de Portas Lógicas – Estilos Variados de Descrição
Item d (3ª. Maneira):

:

```
ENTITY pratica8 IS

        PORT    ( a,b,c     : IN BIT ;
                  d         : IN BIT_VECTOR(2 DOWNTO 1) ;

                  -- Output ports
                  x,y, w, z     : OUT BIT );
END pratica8;

ARCHITECTURE  comportamental  OF pratica8 IS
BEGIN

PROCESS (a,b,c,d)
BEGIN
        IF (d(1)/=d(2))  THEN
            IF ( (b  OR c)='0')THEN
                IF  (a = '1' ) THEN
                z<= '1';
           ELSE
                z <= '0';
                END IF;
            END IF;
        END IF;
END PROCESS;
END comportamental;
```



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;
```

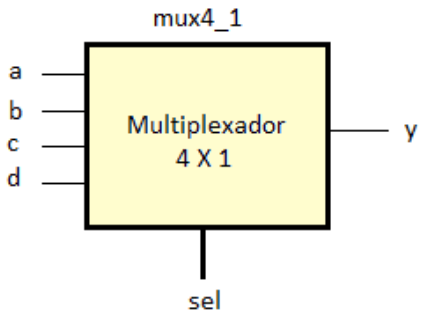
Comandos em VHDL – Sequenciais
CASE-WHEN

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

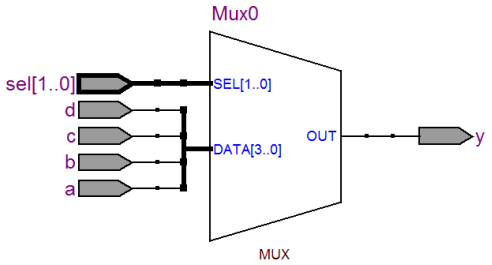
```
CASE seletor IS  -- Existindo apenas 4 estados do seletor e para
                 -- alguns estados da entrada levam à mesma saída
        WHEN estado1_seletor =>
                saída <= saida_1;
        WHEN estado2_seletor =>
                saída <= saída_2;
        WHEN OTHERS  =>
         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



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.

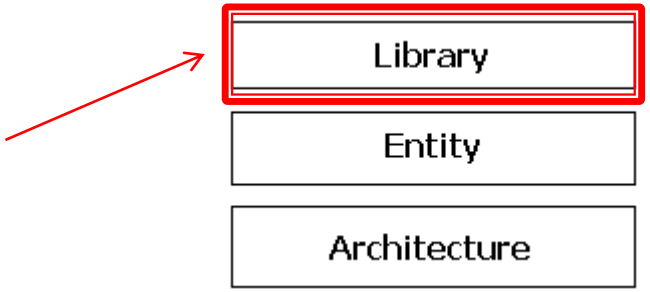
Comandos em VHDL – Comparação entre WITH-SELECT e CASE-WHEN

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 unidades de projetos compiladas. Quando antes da declaração da entidade é referenciada uma biblioteca, a declaração " LIBRARY" deve aparecer seguido do nome da biblioteca LIBRARY nome;);

Caso partes da biblioteca sejam usadas deve-se declarar através da cláusula: "USE nome_do_pacote.ALL;) :

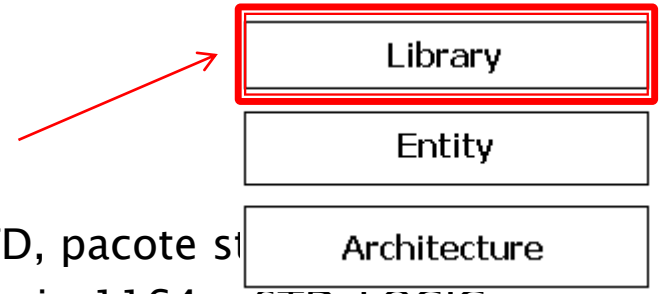
Ex:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed
```

Observações:

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

LIBRARY



Pacotes com definições e tipos de dados: – Biblioteca STD, pacote st BOOLEAN, INTEGER, REAL – Biblioteca IEEE, pacote std_logic_1164 • STD_LOGIC, STD_ULOGIC – Biblioteca IEEE, pacote std_logic_arith • Signed, unsigned • Funções de conversão como: conv_integer(p), conv_unsigned(p,b), conv_signed(p,b) e conv_std_logic_vector(p,b). – Biblioteca IEEE, pacotes std_logic_signed e std_logic_unsigned • Funções que permitem operações com dados STD_LOGIC_VECTOR do tipo SIGNED ou UNSIGNED, respectivamente.

Ex:

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

– Biblioteca STD, pacote standard

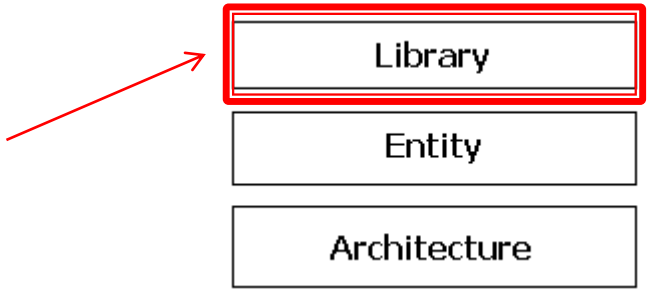
Tipo de dados: BIT, BOOLEAN, INTEGER, REAL

```
USE IEEE.std_logic_unsigned.ALL;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_signed
```

Observações:

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

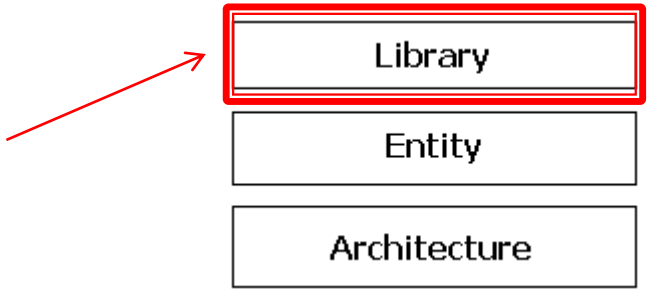
LIBRARY



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



Pacotes mais utilizados da biblioteca IEEE:

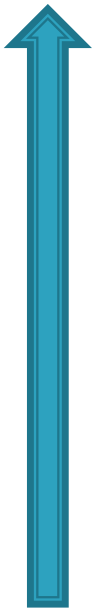
LIBRARY IEEE;	Usada para incluir a biblioteca IEEE;
USE IEEE.std_logic_1164.ALL;	Define o tipo STD_LOGIC (VECTOR), usado em descrições mais fiés a um circuito real.
USE IEEE.std_logic_arith.ALL;	Define os tipos SIGNED e UNSIGNED.
USE IEEE.std_logic_signed.ALL;	Permite tratar sinais do tipo STD_LOGIC_VECTOR 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.

Tipos em VHDL (continuação) :

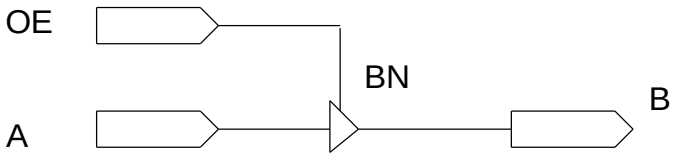
CONVERSÃO ENTRE TIPOS:

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

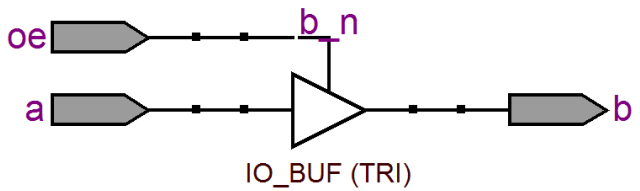
Observação: Não é permitida a transferência de valores entre objetos de tipos diferentes.

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;
ARCHITECTURE  a OF tri_state IS --  declaração da arquitetura
       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 ;
```

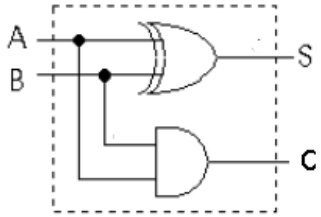


Circuito sintetizado:

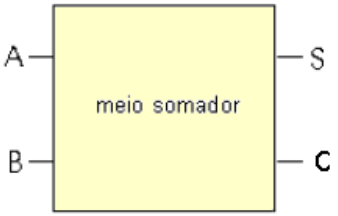


Prática nº9
Junção de Projetos

9.1

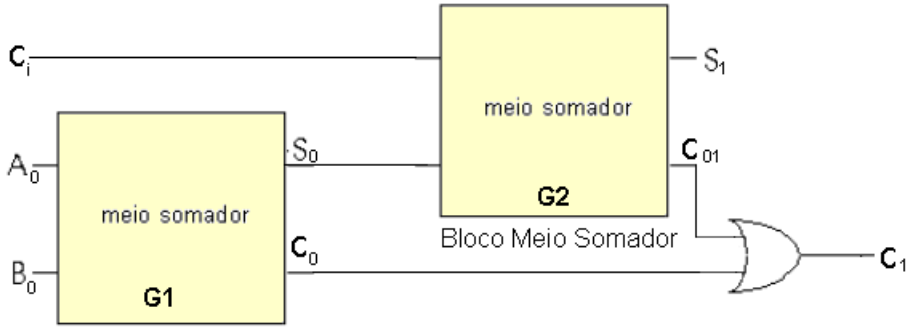


Circuito do Meio Somador



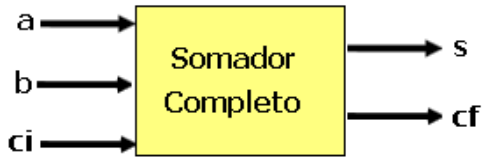
Bloco Meio Somador

9.2



Bloco Meio Somador

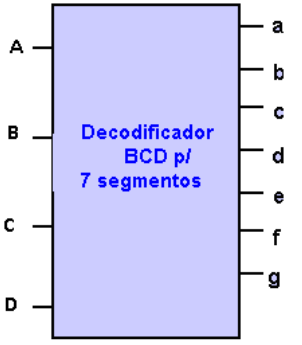
9.3



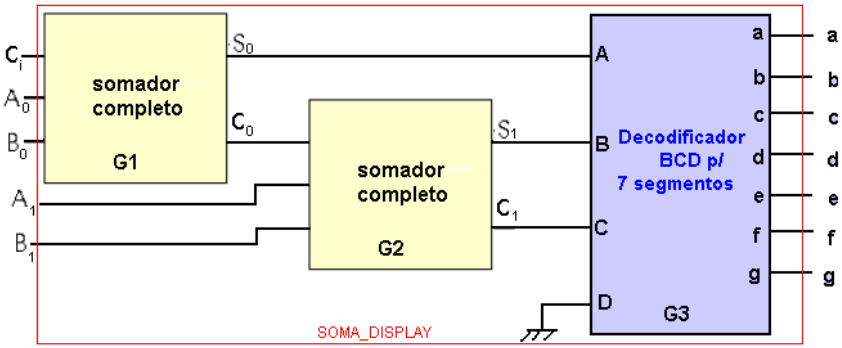
$s = a \oplus b \oplus ci$

$cf = (a \cdot b) + ci\,(a + b)$

9.3



9.4



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 Estrutural Usando Meio-Somadores

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

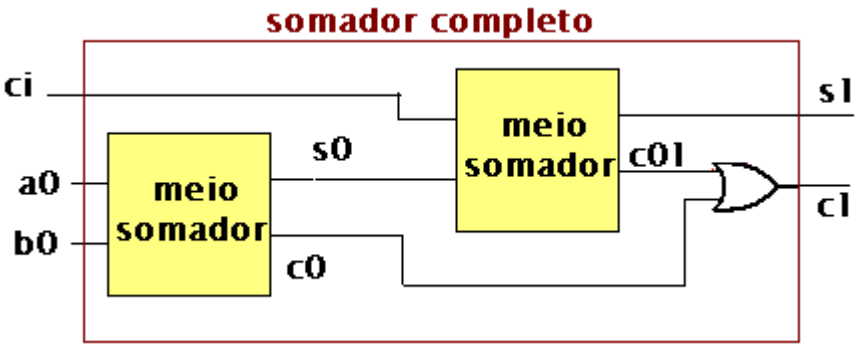
Expressões:

$s = a \oplus b$

$cf = a.b$

Tabela Verdade:

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



Projeto do bloco meio somador:

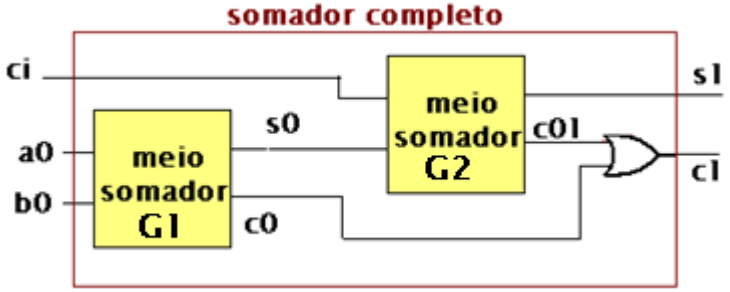
$s = a \oplus b$

$cf = a.b$

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;
```

✓ Descrição estrutural do somador completo usando blocos meio somadores:




```
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 o projeto  do meio somador entidade e arquitetura
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL; -- declaração da bibiloteca e pacote
-- nome da entidade do meio_somador
ENTITY meio_somador IS
                PORT( a,b :IN STD_LOGIC;
                        s,c :OUT STD_LOGIC);
END meio_somador ;

ARCHITECTURE fluxo_dados OF meio_somador IS
BEGIN
      s <= a XOR b;
      c <= a AND b;
END fluxo_dados;
-- 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,c :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, c => c01) ;
            c1 <= c0 OR  c01;
  END estrutural ;
```

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  BIT;
         s, vai_1    : OUT BIT);
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;
```

Prática nº9- Resolução

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
ENTITY LED_MSD_DISPLAY  IS
        PORT(MSD  : IN BIT_VECTOR(3 DOWNTO 0);
         MSD_7SEG : OUT BIT_VECTOR(6 DOWNTO 0));
END LED_MSD_DISPLAY;
ARCHITECTURE a OF LED_MSD_DISPLAY IS -- Decodificador BCD para display de 7 segment
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 nº9- Resolução

```
LIBRARY 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;
```