Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação

SEL0384 – Laboratório de Sistemas Digitais I

Profa. Luiza Maria Romeiro Codá

Laboratório de Eletrônica Digital SEL-EESC-USP

EESC · USP

Introdução a VHDL

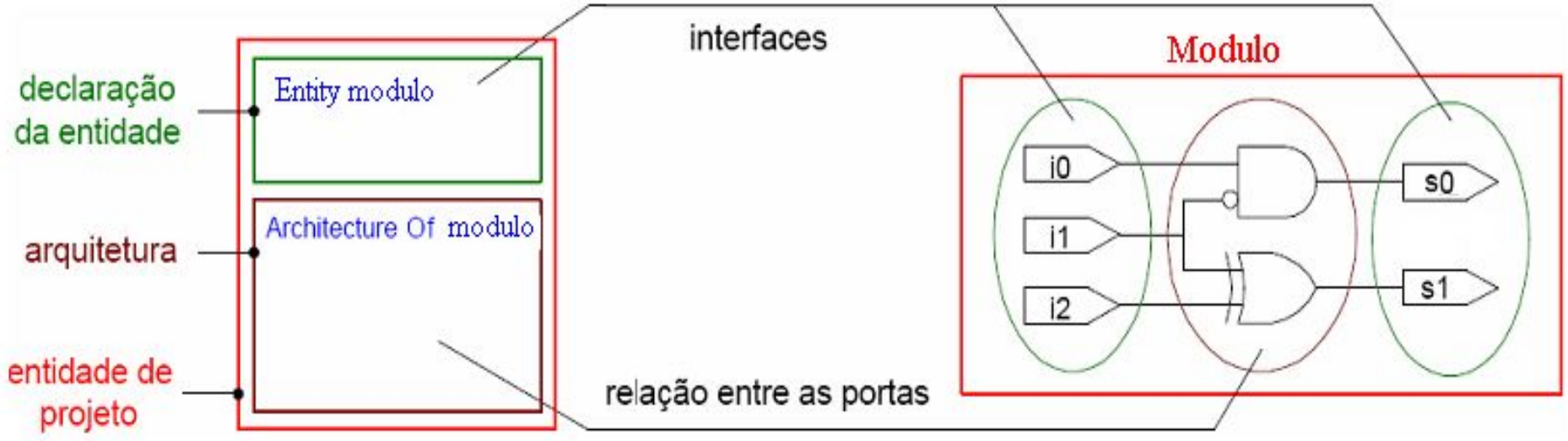
Aula 2

Declaração de Sinais (SIGNAL)
Declaração de Vetores
Descrição de circuitos Combinacionais:

- Construção WHEN ELSE
- Construção WITH SELECT

Professora Luiza Maria Romeiro Codá

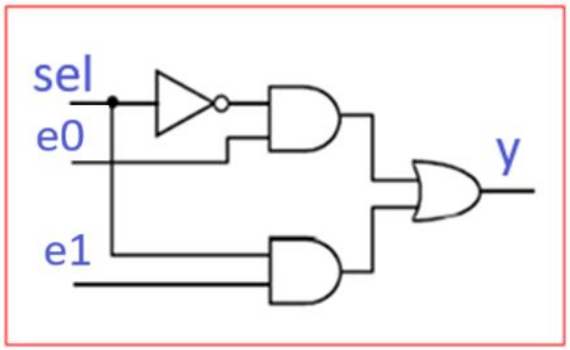
Revisão Aula 1: VHDL - Estrutura de uma descrição



Descrição VHDL de um MUX 2x1

Utilizando operadores lógicos

mux2_1



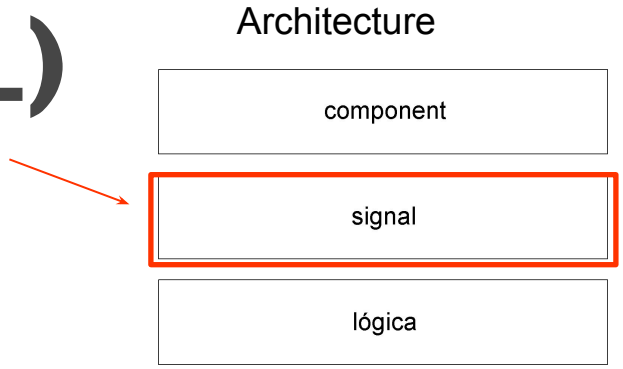
$$y = \overline{sel}.e0 + sel.e1$$

```
ENTITY mux2_1 IS
   PORT(sel, e0, e1 : IN  BIT;
        y           : OUT BIT);
END mux2_1;

ARCHITECTURE logica OF mux2_1 IS
BEGIN
   y <= ( e0 AND (NOT sel )) OR (e1 AND sel);

END logica;
```

Sinal (SIGNAL)



- Declarado entre a declaração da arquitetura e o BEGIN
- Sinais são utilizados para comunicação entre componentes.
- Sinais podem ser interpretados como fios físicos, reais.
- todo PORT é um sinal

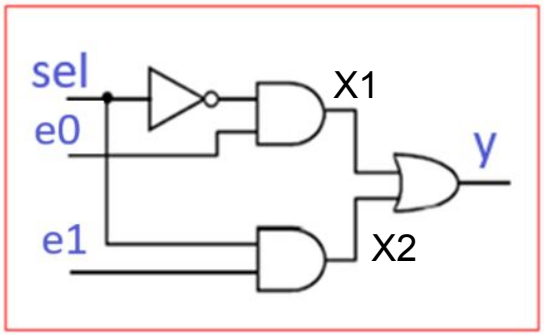
Declaração de Sinal:

```
SIGNAL <nome_do_sinal>: <tipo>[:= valor];
SIGNAL X : BIT;
SIGNAL X : BIT :='0';
```

Descrição VHDL de um MUX 2x1

Utilizando operadores lógicos e SIGNAL

mux2_1



$$x1 = \overline{sel.}\, e0$$
$$x2 = sel.e1$$
$$y = x1 + x2$$

```
ENTITY mux2_1 IS
   PORT(sel, e0, e1 : IN  BIT;
        y           : OUT BIT);
END mux2_1;

ARCHITECTURE logica OF mux2_1 IS
SIGNAL x1, x2 : BIT; -- declaração SIGNAL antes de BEGIN
BEGIN
   x1 <=  e0 AND (NOT sel );
   x2 <=  e1 AND sel;
    y <=  x1 OR x2;
END logica;
```

Vetores

Vetores são conjuntos (tipo ARRAY) de elementos tratados pelo mesmo nome.

O pacote padrão VHDL define o tipo BIT_VECTOR, formado por elementos do tipo BIT

A declaração de um vetor define em que lado fica o LSB:

```
Va <=(3 DOWNTO 0); -- Declara um vetor cujo LSB fica à direita.
```


```
Vb <=(0 TO 3); -- Declara um vetor cujo LSB fica à esquerda.
```

Vetores podem ser referenciados de duas maneiras:

- Nomes indexados
  O nome do vetor é seguido do índice do elemento desejado.

```
vetor(0);
```

- Partes de vetores
  O nome do vetor é seguido de um intervalo de índices.

```
vc <= (3 DOWNTO 0);
```

Vetores - Atribuição

Ao se atribuir valores a vetores, a expressão que contém o valor pode ser um valor, outro vetor (ou parte deste), nomes indexados, ou um agregado. E a atribuição de valores de vetores utiliza-se haspas(“)

Há basicamente 2 tipos de atribuição a vetores:

- Atribuição direta
  Na atribuição direta, os valores são passados para o vetor inteiro, ou partes deste se va <= (3 DOWNTO 0) então:

```
va <= “1011”; --não deixar espaço entre os bits do vetor
```

- Por agregados
  Na atribuição por agregado, é possível atribuir valores a cada elemento do vetor separadamente.

```
va <= (1 => ‘1’, 2 => ‘1’, OTHERS => ‘0’); -- va <= “0110”
 --A posição 1 (va(1)) e a 2 (va(2)) recebem ‘1’ e as outras ‘0’
```

Vetores – atribuição direta utilizando CONSTANT

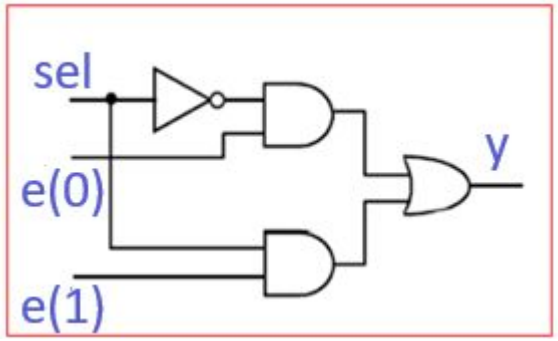
```
ENTITY exemplo1_v IS
   PORT(va, vb, vc, vd, ve, vf, vg, vh : OUT BIT_VECTOR(4 DOWNTO 0));
END exemplo1_v;
ARCHITECTURE teste OF exemplo_v1 IS
        -- Definição de constantes
   CONSTANT c1   : BIT_VECTOR(4 DOWNTO 0) := “01011”;
   CONSTANT zero : BIT := ‘0’;
   CONSTANT um   : BIT := ‘1’;
BEGIN
   va <= c1; -- va <= "01011"    -- Atribuição por constante
   vb <= “01011”; -- vb <= "01011" Atribuição com valor direto
   vc <= "01" & va(2) & zero & um;--vc <= "01001" Concatenação
   vd(4 DOWNTO 3) <= “01”;         -- Atribuição parcial
   vd(2 DOWNTO 0) <= “0” & vc(2 TO 3); -- Atribuição parcial
              --vd <= "01001“
  ve <= (‘0’,’1’,’0’,’1’,’1’); -- ve <= "01011“   Notação posicional
   vf <= (1 => ‘1’, OTHERS => ‘0’)  -- vf <= “00010” Associação por nomes
   vg <= (zero, vb(3), um OR va(0), ‘0’, ’0’)-- vg <="00100"Agregado com operações
   vh (4 DOWNTO 3 => “00”, 1 => ‘1’, OTHERS => ‘1’); -- Agregado com faixa discreta
                                                      -- vh <= "00111
END teste;
```

0	1	0	Vc(2)	Vc(3)

Descrição VHDL de um MUX 2x1

Utilizando operadores lógicos e vetores

mux2_1



$$y = \overline{sel}.e[0] + sel.e[1]$$

Declaração da entrada e como vetor

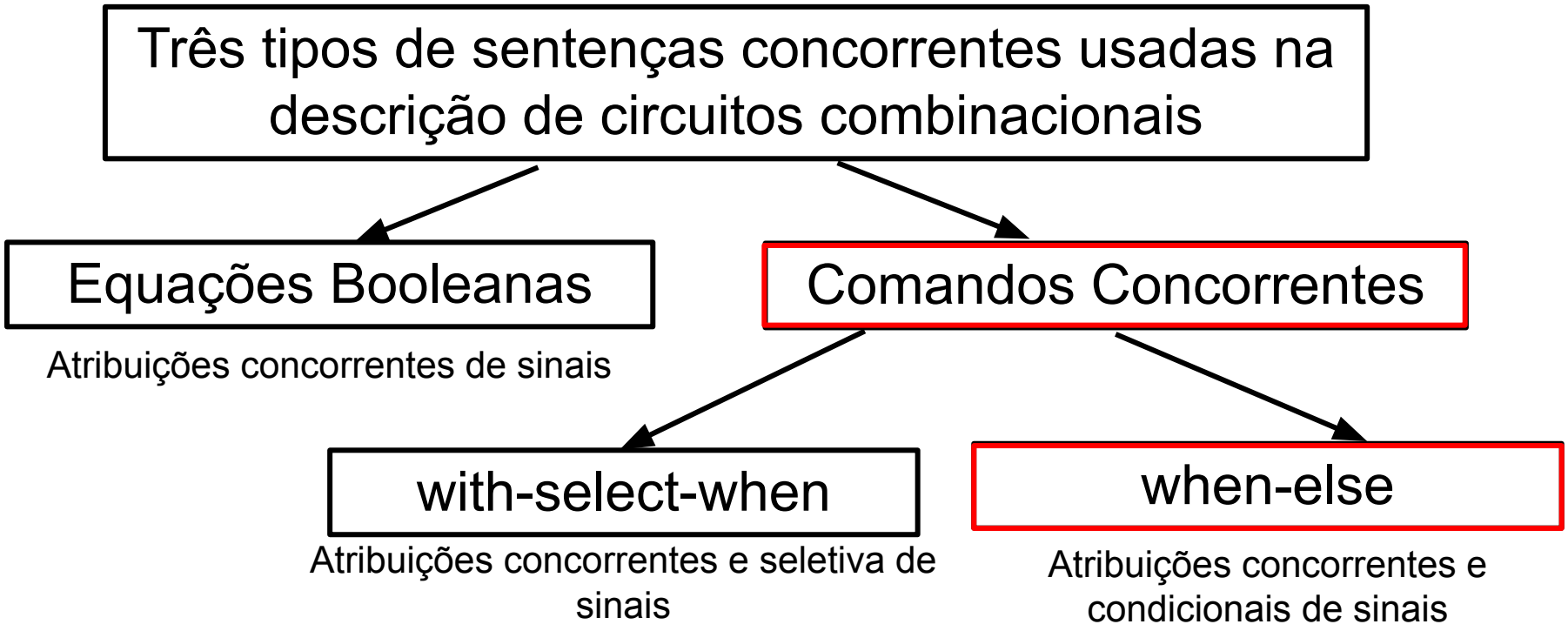
```
ENTITY mux2_1 IS
   PORT(sel      : IN  BIT;
        e        : IN BIT_VECTOR ( 1 DOWNTO 0);
        s        : OUT BIT);
END mux2_1;

ARCHITECTURE logica OF mux2_1 IS
BEGIN
   y <= ( e(0) AND (NOT sel )) OR (e(1) AND sel);

END logica;
```

ARCHITECTURE – Circuitos Combinacionais

Descrição da arquitetura utiliza expressões lógicas ou comandos concorrentes



Comando Concorrente WHEN-ELSE

- Atribuição condicional de sinais.
- Útil para expressar funções lógicas em forma de tabela
- Descrever circuito combinacional

```
sinal_destino <= expressao_a WHEN condicao_1 ELSE
                 expressao_b WHEN condicao_2 ELSE
                 expressao_c;
```

Na construção WHEN-ELSE, a ordem da apresentação das condições indica a precedência de execução (a primeira com prioridade máxima e a última com prioridade mínima).

ATENÇÃO:

- Atribuição de valor a sinais do tipo BIT : A <= '1' ou A <= '0‘; valor entre aspas simples (‘ 0 ‘)

- Atribuição de valor a sinais do tipo BIT_VECTOR(1 DOWNTO) : A <= “11” ou A <= “00”; valor entre aspas dupla (“00”)

ARCHITECTURE – Usando Comando Concorrente WHEN-ELSE

Multiplex de 2 x 1

Circuito1

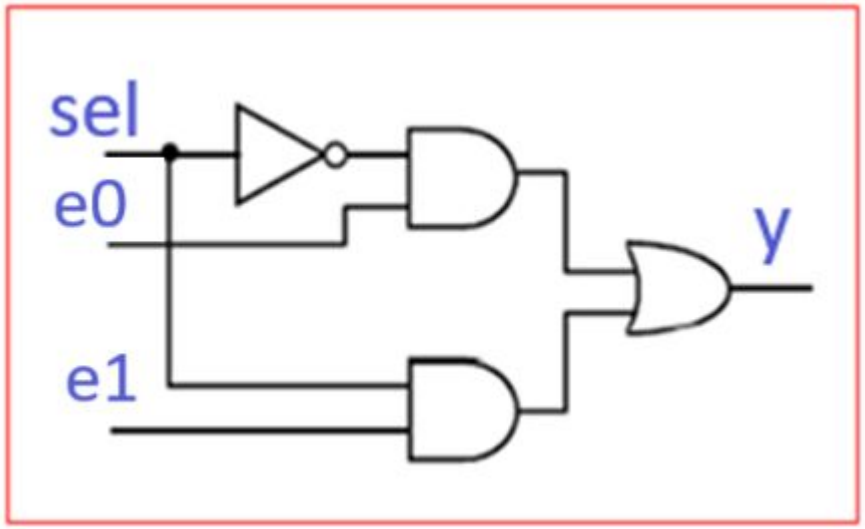


Tabela Verdade do Circuito 1

sel	e1	e0	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

ARCHITECTURE – Usando Comando Concorrente WHEN-ELSE

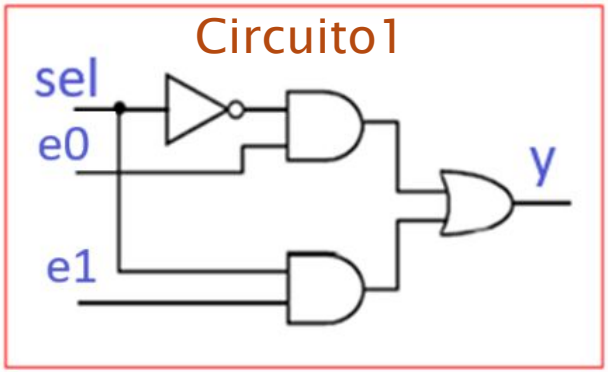
Multiplex de 2 x 1

```
ENTITY circuito1 IS
   PORT(sel, e0, e1 : IN  BIT;
                y          : OUT BIT);
END circuito1;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WHEN-ELSE
ARCHITECTURE fluxo_dados OF circuito1 IS
BEGIN
 y <= '1' WHEN  (sel='0' AND  e0 ='0' AND  e1='1') ELSE
        '1' WHEN   (sel='0' AND  e0='1' AND   e1='1') ELSE
        '1' WHEN   (sel='1' AND  e0='1' AND  e1='0') ELSE
        '1' WHEN   (sel='1' AND  e0='1' AND  e1='1') ELSE
        '0';
END fluxo_dados;
```

sel	e1	e0	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

ARCHITECTURE – Usando Comando Concorrente WHEN-ELSE

Multiplex de 2 x 1



Circuito1 Sintetizado

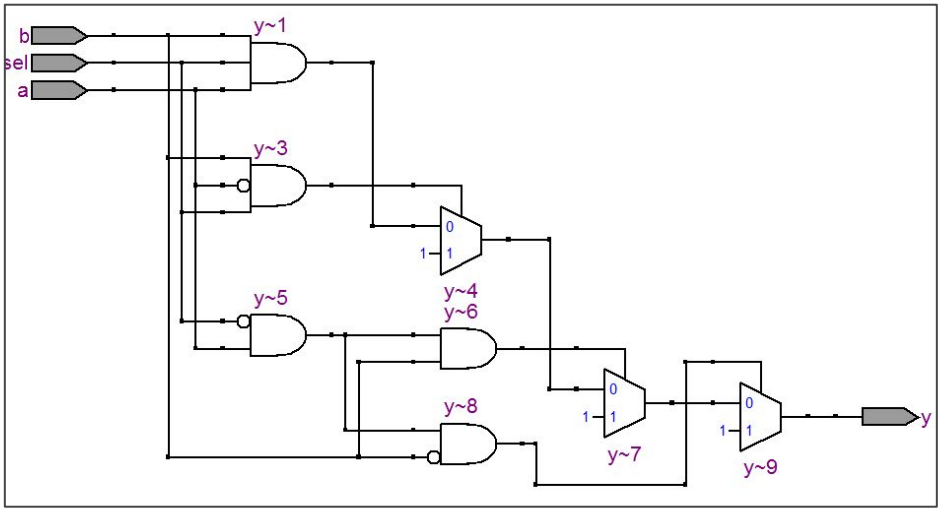


Tabela Verdade do Circuito 1

sel	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

ARCHITECTURE –
Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE e vetor

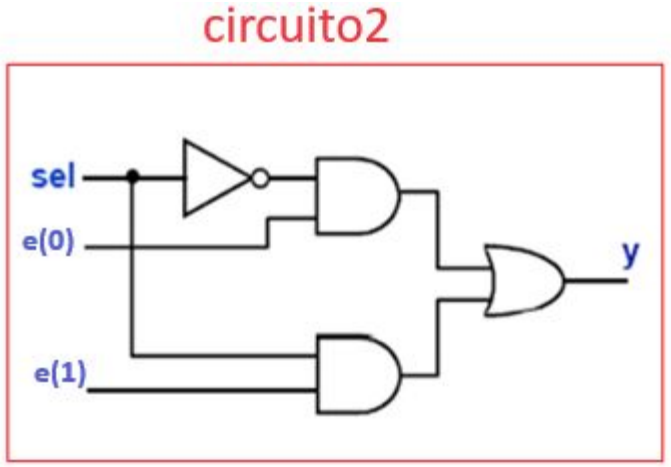


Tabela Verdade do Circuito2

sel	e[1]	e[0]	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

ARCHITECTURE – Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE

1ª. Maneira:

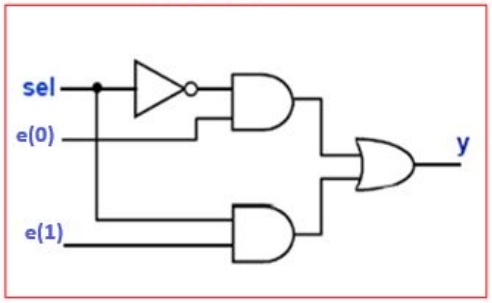
```
ENTITY circuito2_a IS
    PORT(sel      : IN  BIT;
         e        : IN BIT_VECROR (1 DOWNTO 0);
         y        : OUT BIT);
END circuito2_a;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WHEN-ELSE
ARCHITECTURE concorrente OF circuito2_a IS
BEGIN
 y <= '1' WHEN (sel='0' AND  e(0)='1' AND  e(1) ='0') ELSE
      '1' WHEN  (sel='0' AND  e(0)='1' AND  e(1) ='1') ELSE
      '1' WHEN  (sel='1' AND  e(0)='0' AND  e(1) ='1') ELSE
      '1' WHEN  (sel='1' AND  e(0)='1' AND  e(1) ='1') ELSE
      '0';
END concorrente;
```

sel	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

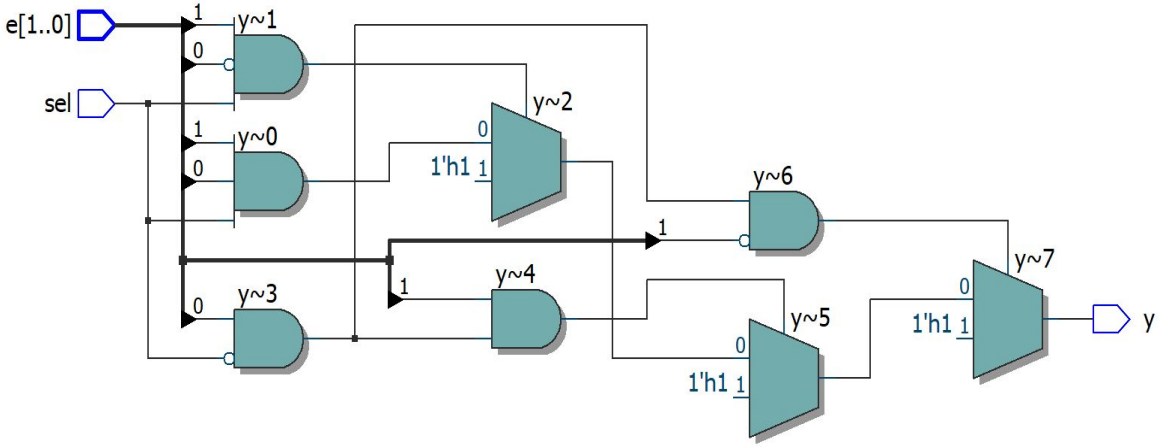
ARCHITECTURE – Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE

1ª. Maneira:

circuito2 a



Circuito Gerado:



sel	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

ARCHITECTURE –
Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE e vetor

2ª. Maneira:

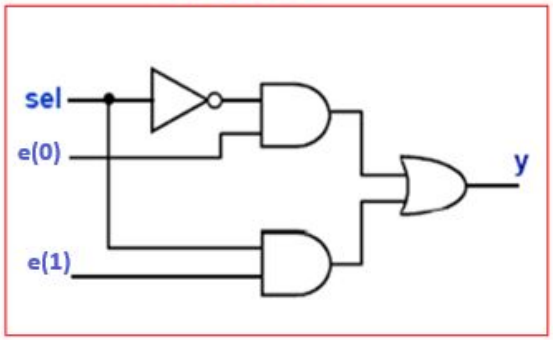
```
ENTITY circuito2_b IS
    PORT(sel      : IN  BIT;
         e        :  IN  BIT_VECTOR(1 DOWNTO 0);
         y        : OUT BIT);
END circuito2_b;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WHEN-ELSE
ARCHITECTURE concorrente OF circuito2_b IS
BEGIN
 y <=     '1' WHEN (sel='0' AND  e = “01”)   ELSE
          '1' WHEN (sel='0' AND  e = “11”)  ELSE
          '1' WHEN (sel='1' AND  e = “10” )ELSE
          '1' WHEN (sel='1' AND  e = “11” ) ELSE
           '0';
END concorrente;
```

sel	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

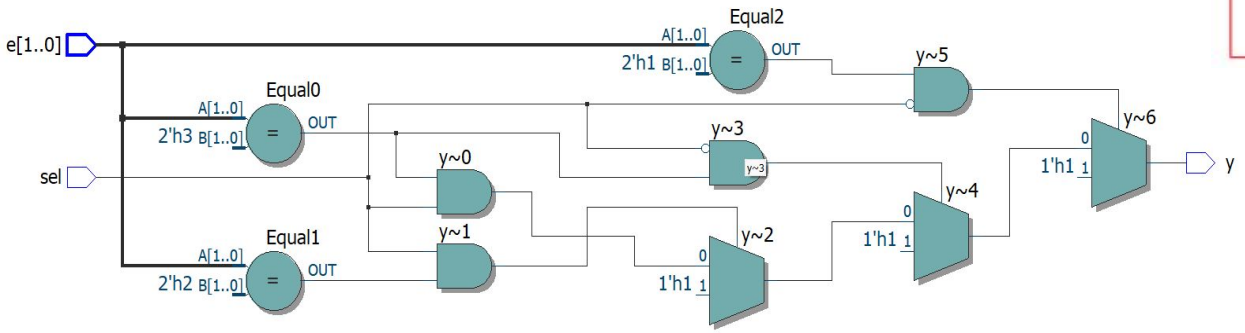
ARCHITECTURE –
Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE e vetor

2ª. Maneira:

circuito2 b



Circuito Gerado:



sel	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

ARCHITECTURE – Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE e vetor

3ª. Maneira:

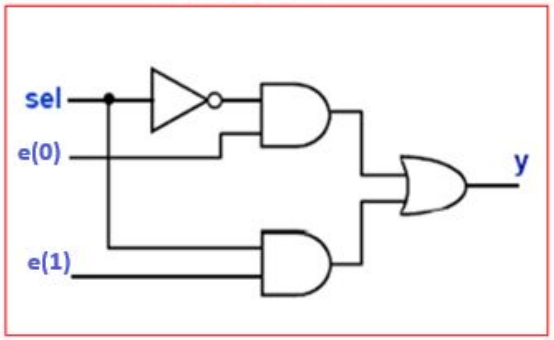
```
ENTITY circuito2_c IS
    PORT  e     :  IN  BIT_VECTOR(2 DOWNTO 0); -- onde e(2) é a entrada sel
          y     : OUT BIT);
END circuito2_c ;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WHEN-ELSE
ARCHITECTURE concorrente OF circuito2_c IS
BEGIN
 y <=     '1' WHEN   e = “001”    ELSE
          '1' WHEN   e = “011”    ELSE
          '1' WHEN    e = “110”   ELSE
           '1' WHEN    e = “111”  ELSE
           '0';
END concorrente;
```

e(2)	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

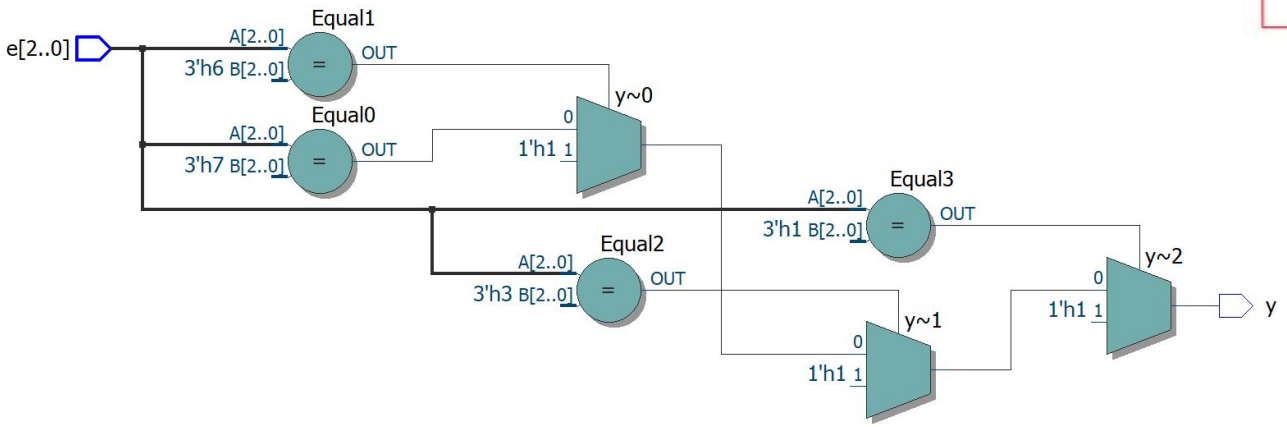
ARCHITECTURE –
Descrição de um multiplex 2 x 1 usando Comando Concorrente WHEN-ELSE e vetor

3ª. Maneira:

circuito2 c



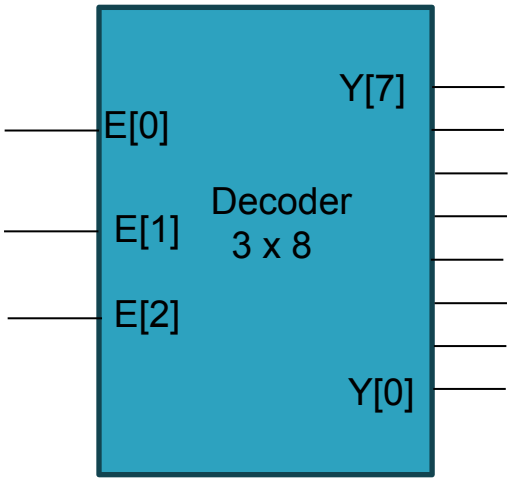
Circuito Gerado:



e(2)	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Prática nº8a:

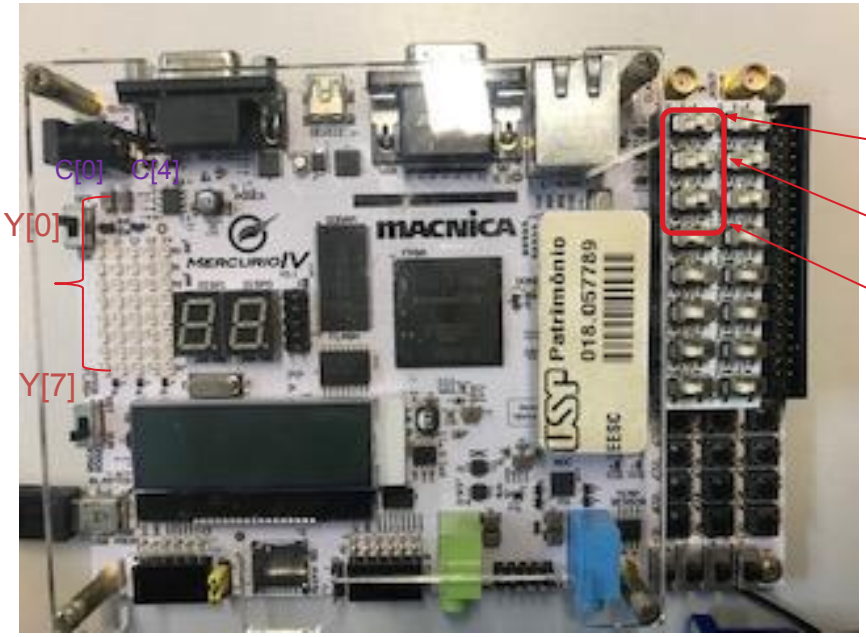
Fazer o projeto em VHDL usando comando WHEN ELSE e vetores de um decodificador 3 x 8 como mostra a tabela verdade. Sintetize no FPGA e teste. Crie pinos para as colunas e ligue-os ao terra



ENTRADAS			SAÍDAS							
E[2]	E[1]	E[0]	Y[7]	Y[6]	Y[5]	Y[4]	Y[3]	Y[2]	Y[1]	Y[0]
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Prática Nº8a : Decodificador 3 x 8

Módulo Mercúrio IV



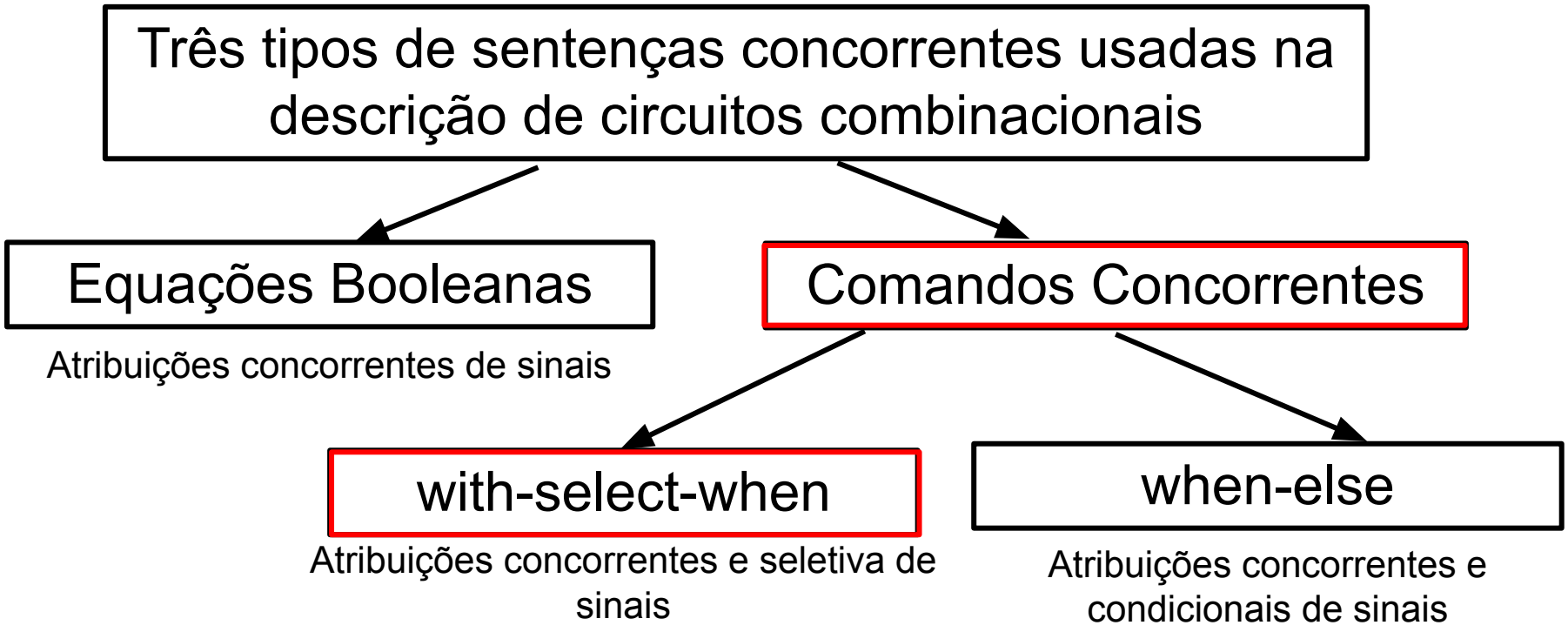
Prática Nº8a VHDL: Decodificador 3 x 8

Pinagem

Sinal	**Pino do FPGA**	**Descrição**
E(0)	E16	Chave 0 do grupo A
E(1)	H22	Chave 1 do grupo A
E(2)	F16	Chave 2 do grupo A
Y(0)	F10	LEDM_R[0] linha 0 da matriz de LEDS
Y(1)	C8	LEDM_R[1] linha 1 da matriz de LEDS
Y(2)	E9	LEDM_R[2] linha 2 da matriz de LEDS
Y(3)	G9	LEDM_R[3] linha 3 da matriz de LEDS
Y(4)	F9	LEDM_R[4] linha 4 da matriz de LEDS
Y(5)	F8	LEDM_R[5] linha 5 da matriz de LEDS
Y(6)	G8	LEDM_R[6] linha 6 da matriz de LEDS
Y(7)	H11	LEDM_R[7] linha 7 da matriz de LEDS
C[0]	J7	Coluna 0 da matriz de LEDS
C[1]	J6	Coluna 1 da matriz de LEDS
C[2]	K8	Coluna 2 da matriz de LEDS
C[3]	J8	Coluna 3 da matriz de LEDS
C[4]	L8	Coluna 4 da matriz de LEDS

ARCHITECTURE – Circuitos Combinacionais

Descrição da arquitetura utiliza expressões lógicas ou comandos concorrentes



Comando Concorrente
WITH-SELECT

```
WITH expressao_escolha SELECT                    -- expressão_escolha =
sinal_destino <= expressao_a WHEN condicao_1, -- Condição 1
                 expressao_b WHEN condicao_2 | condicao_3,  -- 2 ou 3
                  expressao_c WHEN (condicao_2 OR condicao_3),
                 expressao_d WHEN condicao_4 TO condicao_6, -- 4 até 6
                 expressao_e WHEN OTHERS;   -- Restantes
```

Na construção WITH-SELECT todas as condições apresentam a mesma prioridade, e todas as possibilidades devem ser apresentadas. Pode-se usar a palavra reservada OTHERS para representar todas as condições não explicitadas.

Comando Concorrentes - Comparação
WHEN-ELSE x WITH-SELECT

WHEN-ELSE: Ordem das condições indica a prioridade.
Explicitação de todas as possibilidades **não** é necessária.

WITH-SELECT: Todas as condições têm mesma prioridade.
Explicitação de todas as possibilidades **é** necessária.

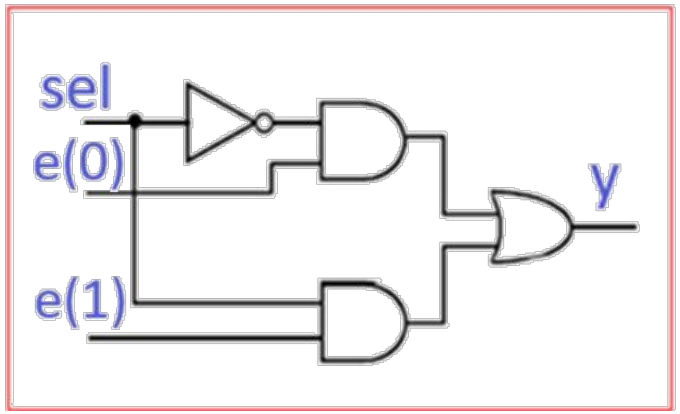
Exemplo 1: **Descrição de um Mux de 2x1 usando o comando concorrente WITH SELECT.**

```
ENTITY mux2_1_sel IS
   PORT(sel       : IN  BIT;
                e      :  IN  BIT_VECTOR(1 DOWNTO 0);
               y       : OUT BIT);
END mux2_1_sel ;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WITH-SELECT
ARCHITECTURE concorrente OF mux2_1_sel IS
BEGIN
   WITH sel SELECT
      y <=          e(0)  WHEN  '0',
                    e(1)  WHEN  '1';
-- Outra maneira de descrever:
--      y <=  e(0)  WHEN  '0',
--            e(1)  WHEN OTHERS;
END concorrente;
```

Tabela Verdade simplificada

sel	e(1)	e(0)	y
0	X	X	e(0)
1	X	X	e(1)

Exemplo 1: **Descrição de um Mux de 2x1 usando o comando concorrente WITH SELECT.**



Circuito1 Sintetizado:

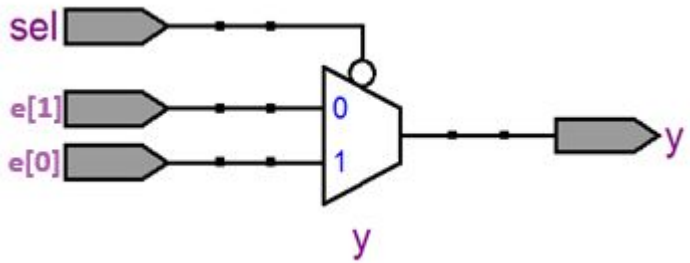


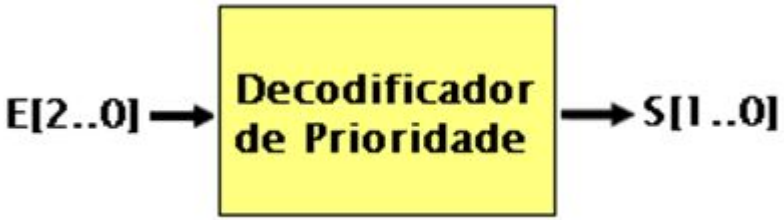
Tabela Verdade do mux2x1

sel	e(1)	e(0)	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Tabela Verdade simplificada

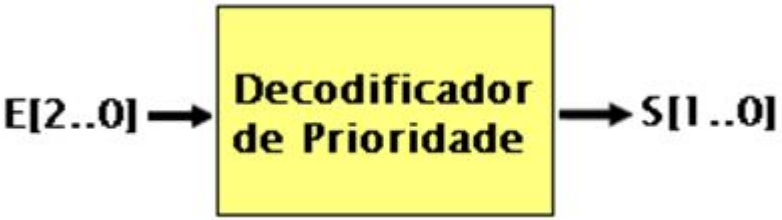
sel	e(1)	e(0)	y
0	X	X	e(0)
1	X	X	e(1)

Exemplo 2: **Descrição de um Decodificador de Prioridade usando o comando concorrente WITH SELECT.**



E2	E1	E0	S1	S0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

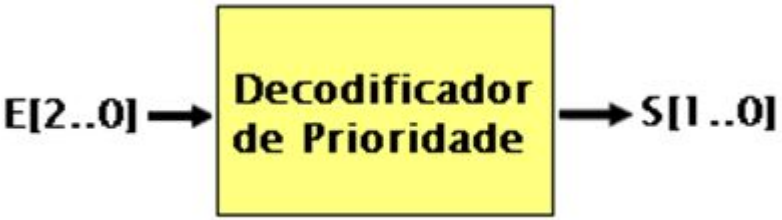
Exemplo 2: **Descrição de um Decodificador de Prioridade usando o comando concorrente WITH SELECT.**




```
EENTITY dec_3x8 IS
   PORT(E : IN  BIT_VECTOR(2 DOWNTO 0);
            S : OUT BIT_VECTOR(1 DOWNTO 0));
END dec_3x8 ;
ARCHITECTURE concorrente OF dec_3x8 IS
BEGIN
   WITH E SELECT
        S <= "11" WHEN "111"|"110"|"101"|"100",
             "10" WHEN "011"|"010",
             "01" WHEN "001",
             "00" WHEN "000"; -- Ou "00" WHEN OTHERS;
END concorrente;
```

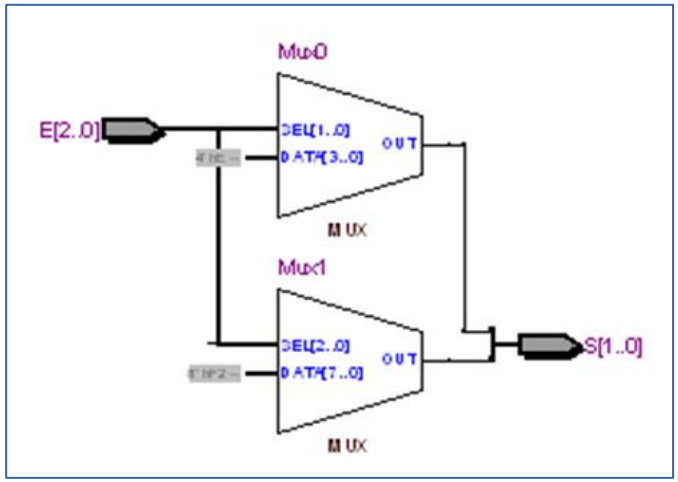
ARCHITECTURE Circuitos Combinacionais

Decodificador de Prioridade Utilizando Comando WITH SELECT

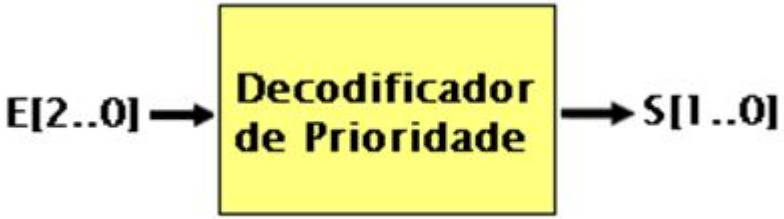



```
EENTITY dec_prior IS
   PORT(E : IN  BIT_VECTOR(2 DOWNTO 0);
        S : OUT BIT_VECTOR(1 DOWNTO 0));
END dec_prior;
ARCHITECTURE fluxo_dados OF dec_prior IS
BEGIN
   WITH E SELECT
        S <= "11" WHEN "111"|"110"|"101"|"100",
             "10" WHEN "011"|"010",
             "01" WHEN "001",
             "00" WHEN "000"; -- Ou "00" WHEN OTHERS;
END fluxo_dados;
```

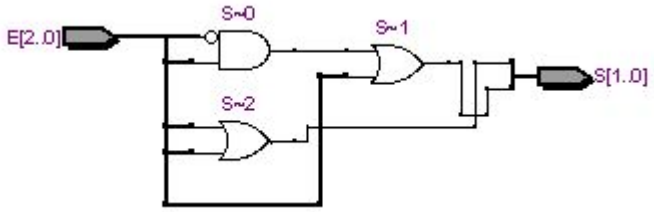
Circuito sintetizado:



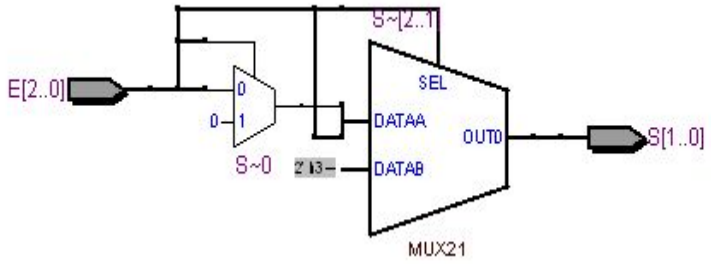
Decodificador de Prioridade:
Circuitos gerados



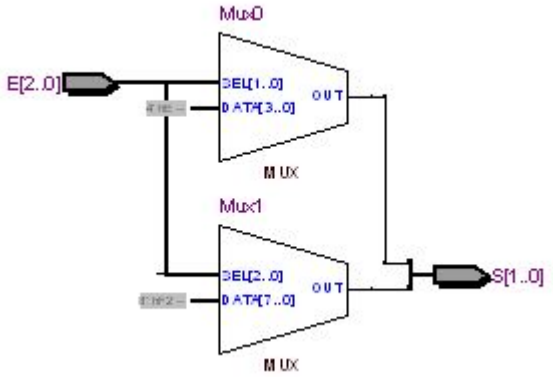
Usando Expressões Lógicas



Usando comando WHEN ELSE

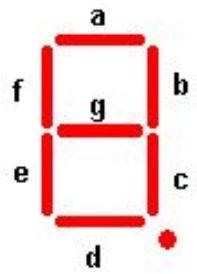


Usando comando WITH-SELECT



Prática nº8b

Utilizando o software Quartus II escreva o projeto em linguagem VHDL de um decodificador BCD para display de 7 segmentos catodo comum (segmentos acendem com nível '1'), usando comando concorrente WITH SELECT. Utilize como entradas um vetor **b** de 4 bits e como saída um vetor **d** com 7 bits. Sintetize no módulo mercúrio IV. Mostrar os números de 0 a 9 e a partir de 10 de A a F (em Hexadecimal).

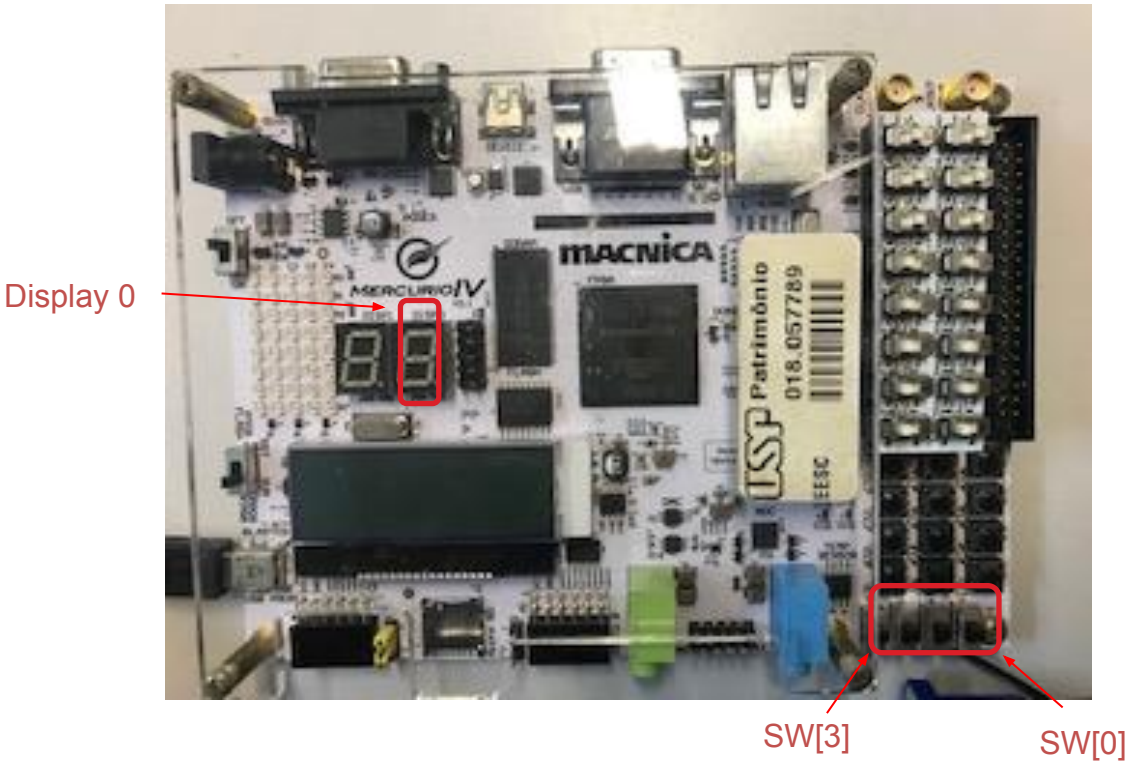


segmento	DISP0	Pinagem do FPGA
a	DISP0_D[0]	V2
b	DISP0_D[1]	V1
c	DISP0_D[2]	U2
d	DISP0_D[3]	U1
e	DISP0_D[4]	Y2
f	DISP0_D[5]	Y1
g	DISP0_D[6]	W2
Pto. decimal	DISP0_D[7]	W1

Prática nº8b

Pinagem do decodificador BCD para display de 7 segmentos

Nome do sinal	Pino do FPGA	
b[0]	V21	Chave SW[0]
b[1]	W22	Chave SW[1]
b[2]	W21	Chave SW[2]
b[3]	Y22	Chave SW[3]
d[0]	V2	Segmento a do display 0
d[1]	V1	Segmento b do display 0
d[2]	U2	Segmento c do display 0
d[3]	U1	Segmento d do display 0
d[4]	Y2	Segmento e do display 0
d[5]	Y1	Segmento f do display 0
d[6]	W2	Segmento g do display 0


Display 0
MACNICA
MERCURIO IV
USP Patrimônio
018.057789
SW[3]
SW[0]

FIM

RESOLUÇÃO DAS PRÁTICAS

Resolução: Prática nº8a VHDL

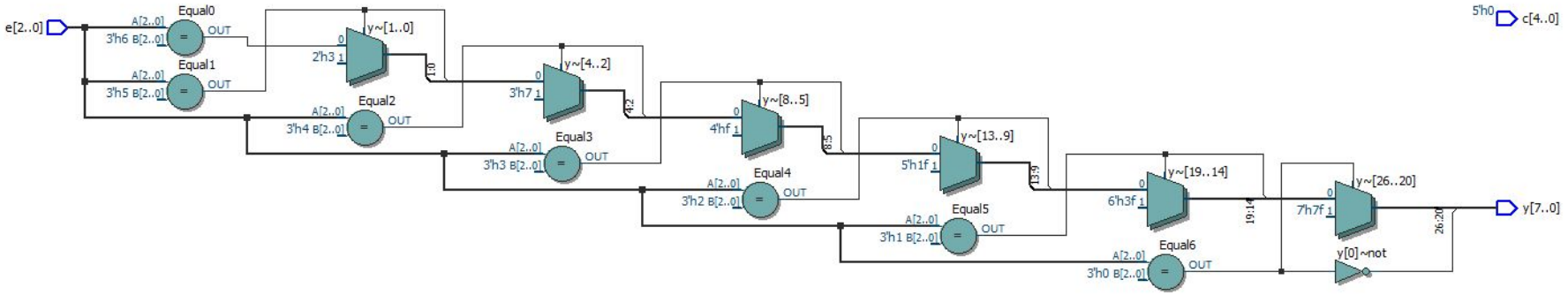
Fazer o projeto em VHDL usando comando WHEN ELSE e vetores de um decodificador 3 x 8 como mostra a tabela verdade

```
ENTITY decoder_3x8 IS
   PORT( e: IN  BIT_VECTOR(2 DOWNTO 0);
            c: OUT ( 4 DOWNTO 0)
           );
END decoder_3x8 ;
ARCHITECTURE concorrente OF decoder_3x8 IS
BEGIN
       s <= "11111110"  WHEN (e = "000" )ELSE
              "11111101" WHEN (e = "001") ELSE
              "11111011" WHEN (e = "010") ELSE
              "11110111" WHEN (e = "011") ELSE
               "11101111" WHEN (e = "100") ELSE
               "11011111" WHEN (e = "101") ELSE
               "10111111" WHEN (e = "110") ELSE
                "01111111";

END concorrente;
```

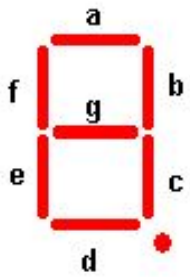
Resolução: Prática nº8a VHDL

Circuito Gerado:



Resolução: Prática n º8b projeto de um decodificador BCD para display de 7 segmentos catodo comum, usando comando concorrente WITH SELECT.

```
ENTITY decoder_bcd_7seg IS
   PORT( b: IN  BIT_VECTOR(3 DOWNTO 0); -- b(3) entrada mais significativa
              d : OUT BIT_VECTOR(6 DOWNTO 0; -- d(6) é o segmento a e d(0) o g
            );
END decoder_bcd_7seg ;
ARCHITECTURE concorrente OF decoder_bcd_7seg IS
BEGIN
       WITH b SELECT
    d <= "1111110"  WHEN   "0000", -- 0
           "0110000" WHEN    "0001", --1
           "1101101" WHEN    "0010", --2
           "1111001" WHEN    "0011", --3
           "0110011" WHEN    "0100", --4
           "1011011" WHEN     "0101", --5
           "1011111" WHEN      "0110", --6
            "1110000" WHEN    "0111", --7
             "1111111"  WHEN   "1000", --8
             "1110011" WHEN    "1001", --9
            "1110111"  WHEN   "1010", --A
             "0011111"  WHEN   "1011", --b
              "1001110"  WHEN   "1100", --C
             "0111101"  WHEN   "1101", --d
              "1001111"  WHEN   "1110", --E
              "1000111"  WHEN   "1111"; --F
END concorrente;
```



Resolução: Prática n º8b

Circuito Gerado: