

# Introdução a VHDL

## Aula 2

**Professora Luiza Maria Romeiro Codá**

# Revisão Aula 1:

## VHDL - Estrutura de uma descrição

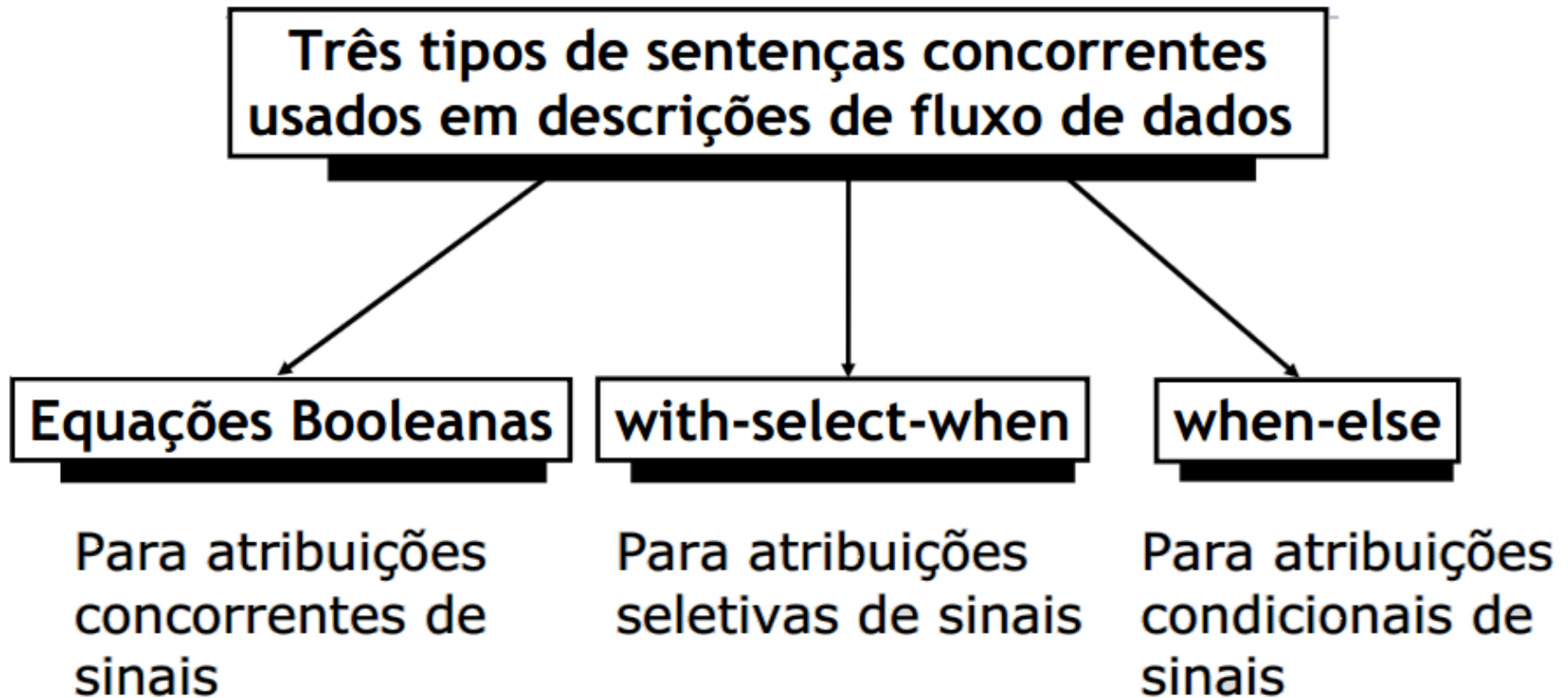


# ARCHITECTURE

- **Descrição por Fluxo de Dados (*Data-Flow*):**  
Descreve o que o sistema deve fazer utilizando expressões lógicas e comandos concorrentes.
- **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.

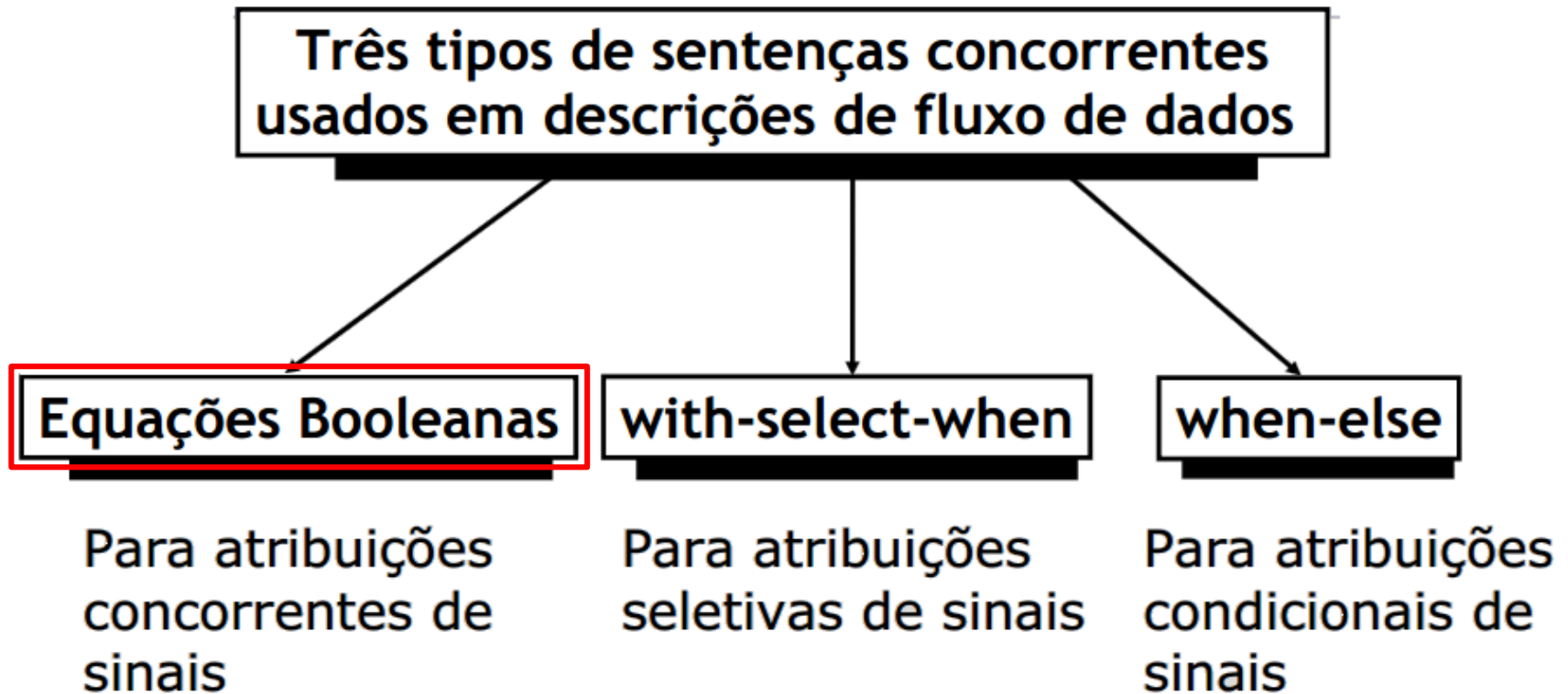
# ARCHITECTURE – Fluxo de Dados

Descrição por **Fluxo de Dados**: Comandos (Sentenças) Concorrentes



# ARCHITECTURE – Fluxo de Dados

Descrição por **Fluxo de Dados**: Comandos (Sentenças) Concorrentes



# ARCHITECTURE

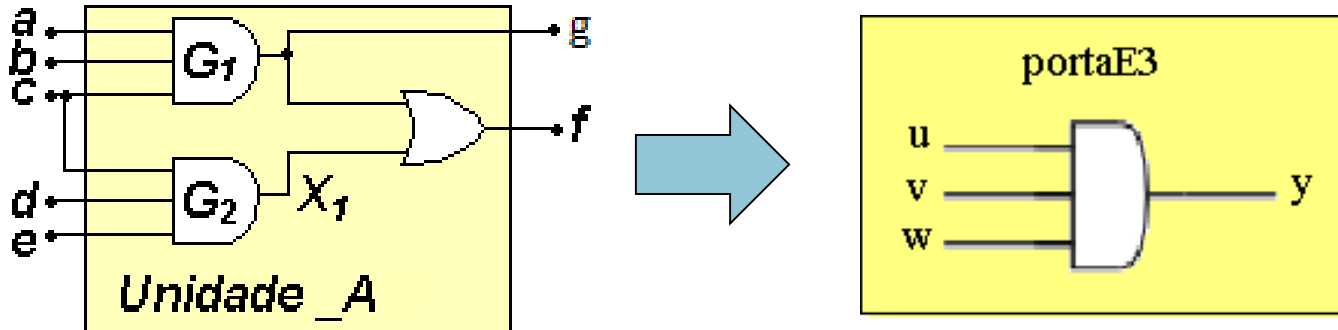
- **Descrição por Fluxo de Dados (*Data-Flow*):**  
Descreve o que o sistema deve fazer utilizando expressões lógicas e comandos concorrentes.
- **Descrição Estrutural:**  
Descreve como é o hardware em termos de interconexão de componentes.
- **Descrição Comportamental:**  
Descreve o que o sistema deve fazer de forma abstrata.

# ARCHITECTURE - Estrutural

## Descrição Estrutural:

- Descreve como é o hardware em termos de **interconexão de componentes**. O mapeamento de entradas e saídas dos componentes é feito através do comando **PORT MAP**.
- A declaração de um componente pode referenciar uma Entidade descrita em outro projeto VHDL, ou uma Entidade descrita no mesmo arquivo onde será utilizada (projeto atual).
- Na descrição Estrutural é feita a associação dos pinos de cada componente com os sinais utilizados no projeto (comando **PORT MAP**).

# ARCHITECTURE - Estrutural



## Instanciação de componente:

**PORT MAP:** especifica as conexões entre ports de uma entity (componente) e sinais na architecture onde o componente foi instanciado.

Existem duas formas de se fazer port map, e não podem ser misturadas:

- associação posicional

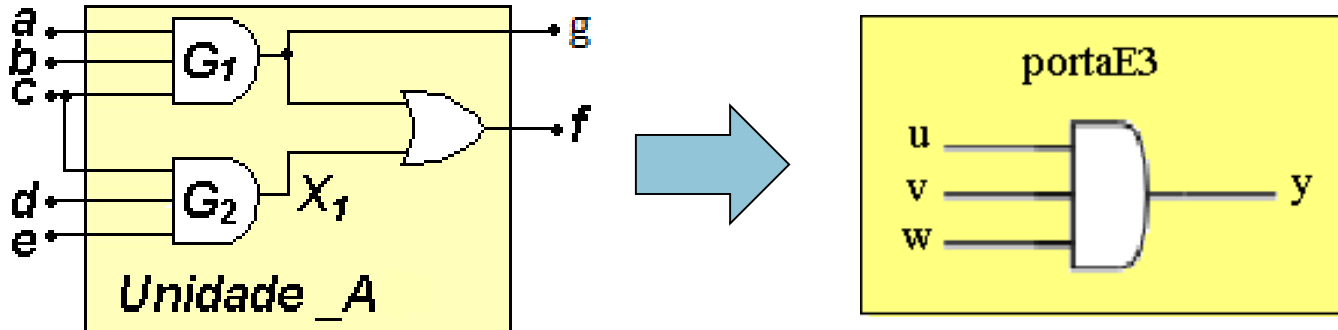
Ex: G1 : portaE3 **PORT MAP** (u => a, v => b, w => c, y => g) ;

- associação por nome.

Ex: G1 : portaE3 **PORT MAP** (a, b, c, d); --segue a ordem da declaração dos PORTS



# ARCHITECTURE - Estrutural

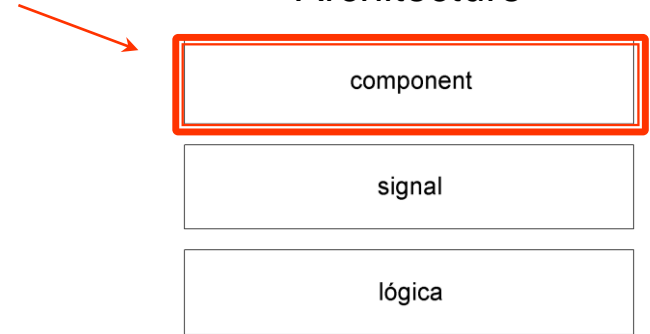


## PORT MAP - Continuação

- ✓ Qualquer combinação de ports e sinais é permitida, desde que haja compatibilidade entre ports e sinais. (devem ser do mesmo tipo)
  - ✓ Todos os elementos do port do componente devem ser associados a algum sinal.
  - ✓ Os ports não conectados podem ser especificados como **OPEN**
  - ✓ no port map, (um port não usado pode ser deixado omitido no port map, porém não é recomendado).
- Ex: G3 : portaE3 **PORT MAP** (u => a, v => b, w => c, y => OPEN) ;

# ARCHITECTURE - Estrutural

Architecture

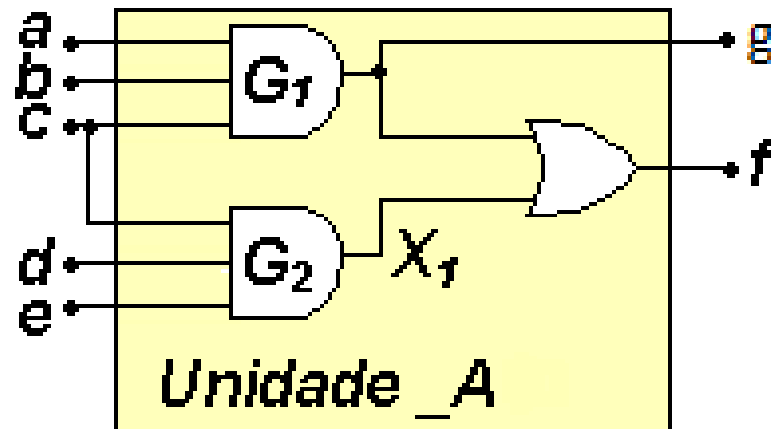


Formato da declaração:

```
ARCHITECTURE <nome_da_architecture> OF <nome_da_entity> IS
-- Declaração de Sinais e Componentes
COMPONENT <nome_do_componente> IS
PORT(a, b : IN BIT;
      s  : OUT BIT);
END COMPONENT;
BEGIN
-- Instanciação (chamada) de Componentes
<rótulo_do_componente> : <nome_do_componente> PORT MAP(
    <nome_pino1> => <sinal_1>, <nome_pino_n> => <sinal_n>);
END nome_da_architecture;
```

# Exemplo:

Prática nº 1 com Descrição Estrutural:  
Utilizando comando PORT MAP



# ARCHITECTURE - Estrutural

```
-- declaração da entidade do componente
ENTITY <nome_do_componente> IS
    PORT(-- declaração dos pinos de entradas,saídas do componente);
END <nome_do_componente>;
```

```
-- Seção de declaração da arquitetura do componente:
ARCHITECTURE <nome_identificador> OF <nome_do_componente> IS
-- declaração de sinais, etc do componente
BEGIN
-- Corpo da arquitetura do componente
END <nome_identificador>;
```

```
-- declaração da entidade do projeto principal
ENTITY <nome_do_proj_principal> IS
    PORT(-- declaração dos pinos de entradas,saídas do projeto principal);
END <nome_do_proj_principal> ;
```

```
-- Seção de declaração da arquitetura do componente:
ARCHITECTURE <nome_identificador> OF <nome_do_proj_principal> IS
-- declaração de sinais, de componentes,etc
```

```
COMPONENT <nome_do_componente> IS
    PORT(-- declaração dos pinos de entradas,saídas do componente);
END COMPONENT;
```

```
BEGIN
-- Corpo da arquitetura do componente
G1: <nome_do_componente> PORT MAP ( --sinais do projeto principal
    --ligados nas entradas e saídas do componente);
END <nome_identificador>;
```

# ARCHITECTURE - Estrutural

Exemplo:

-- Declaração do projeto do componente portaE3

```
ENTITY portaE3 IS
```

```
PORT (u,v,w : IN BIT;
```

```
      y   : OUT BIT);
```

```
END portaE3;
```

-- Descrição da arquitetura por Fluxo de Dados

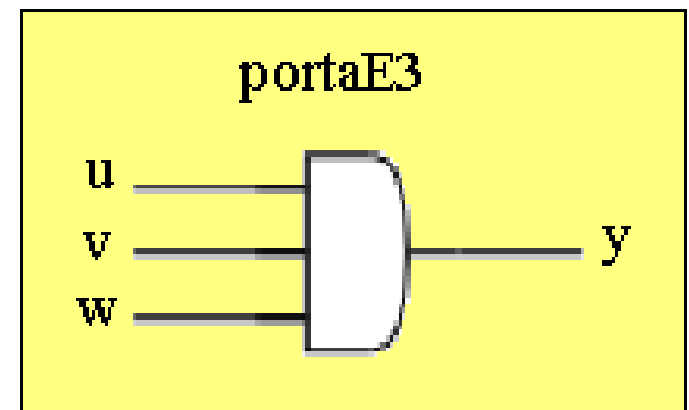
-- do componente portaE3

```
ARCHITECTURE fluxo_dados OF portaE3 IS
```

```
BEGIN
```

```
    y <= u AND v AND w;
```

```
END fluxo_dados;
```



# ARCHITECTURE - Estrutural

## Exemplo:

-- Declaração da entidade do projeto de hierarquia mais alta

```
ENTITY unid_A IS
```

```
  PORT(a, b, c, d, e : IN  BIT;
```

```
       f             : OUT BIT;
```

```
       g             : BUFFER BIT);
```

```
END unid_A;
```

-- Declaração de arquitetura do projeto principal usando

-- descrição estrutural

```
ARCHITECTURE estrutural OF unid_A IS
```

```
SIGNAL X1 : BIT;    -- Declaração de sinal
```

```
COMPONENT portaE3 IS -- Declaração de componente
```

```
  PORT (u,v,w : IN  BIT;
```

```
       y      : OUT BIT);
```

```
END COMPONENT;
```

```
BEGIN
```

-- Instanciação de componentes:

-- atribuição posicional

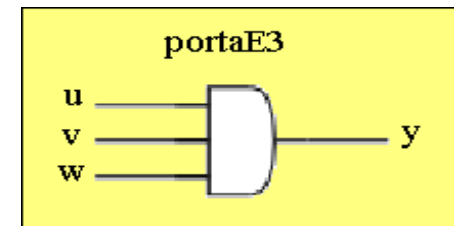
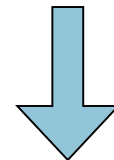
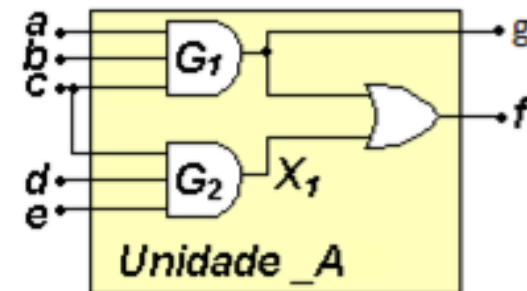
```
G1 : portaE3 PORT MAP (u => a, v => b, w => c, y => g);
```

```
G2 : portaE3 PORT MAP (c, d, e, X1); -- atribuição direta
```

```
f <= X1 OR g;
```

```
END estrutural;
```

--não pode misturar atribuição posicional e direta no mesmo PORT MAP



# Vetores - Introdução

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 DOWNT0 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’, OTHERS => ‘0’); -- va <= “0010”  
--A posição 1 (va(1)) recebe ‘1’ e as outras ‘0’
```



# Vetores – atribuição direta utilizando **CONSTANT**

```
ENTITY exemplo1_v IS
```

```
  PORT(va, vb, vc, vd : OUT BIT_VECTOR(4 DOWNT0 0));
```

```
END exemplo1_v;
```

```
ARCHITECTURE teste OF exemplo_v1 IS
```

```
  -- Definição de constantes
```

```
  CONSTANT c1 : BIT_VECTOR(4 DOWNT0 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 DOWNT0 3) <= "01"; -- Atribuição parcial
```

```
  vd(2 DOWNT0 0) <= "0" & vc(2 TO 3); -- Atribuição parcial
```

```
  --vd <= "01001"
```

```
END teste;
```

0	1	0	<u>Vc(2)</u>	<u>Vc(3)</u>
---	---	---	--------------	--------------

# Vetores – Exemplo de atribuição por agregado

```
ENTITY exemplo2_v IS
```

```
  PORT(va, vb, vc, vd : OUT BIT_VECTOR(4 DOWNTO 0));
```

```
END exemplo_v2;
```

```
ARCHITECTURE teste OF exemplo2_v IS
```

```
  -- Definição de constantes
```

```
  CONSTANT zero : BIT := '0';
```

```
  CONSTANT um  : BIT := '1';
```

```
BEGIN
```

```
  va <= ('0','1','0','1','1'); -- va <= "01011"  Notação posicional
```

```
  vb <= (1 => '1', OTHERS => '0') -- vb <= "00010" Associação por nomes
```

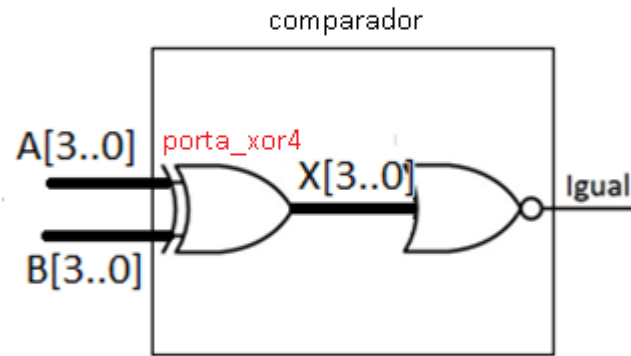
```
  vc <= (zero, vb(3), um OR va(0), '0', '0')-- vc <= "00100" Agregado com operações
```

```
  vd (4 DOWNTO 3 => "00", 1 => '1', OTHERS => '1');-- Agregado com faixa discreta
```

```
  -- vd <= "00111"
```

```
END teste;
```

# Comparador de Igualdade- usando vetores



# Comparador de Igualdade - Utilizando vetor

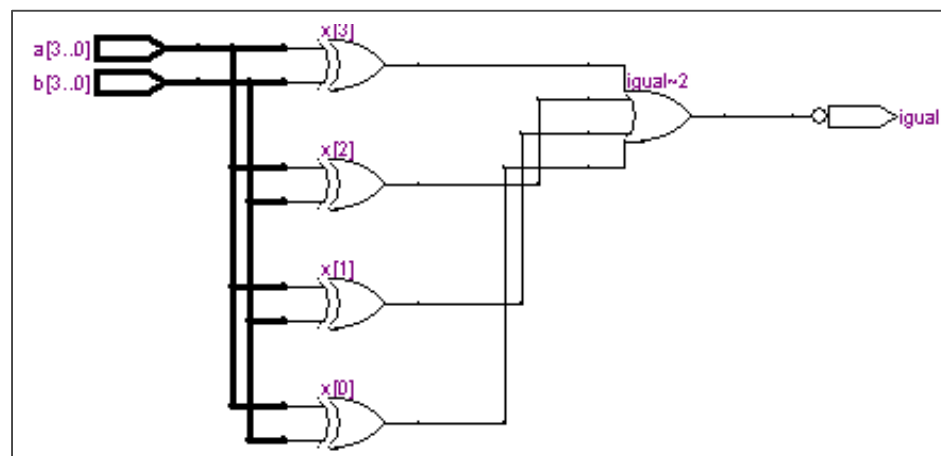
```
ENTITY comparador_V IS  
    PORT ( a,b: IN BIT_VECTOR( 3 DOWNT0  
0);  
          igual : OUT BIT);  
END comparador_V;
```

```
ARCHITECTURE a OF comparador_V IS  
    SIGNAL x : BIT_VECTOR( 3 DOWNT0 0);  
BEGIN
```

```
    x <= a XOR b;  
    -- ou de outra maneira:  
    --x(0) <= a(0) XOR b(0);  
    --x(1) <= a(1) XOR b(1);  
    --x(2) <= a(2) XOR b(2);  
    --x(3) <= a(3) XOR b(3);  
    igual <= NOT ( x(0) OR x(1) OR x(2) OR
```

```
x(3));  
END a;
```

Mesmo Circuito sintetizado:

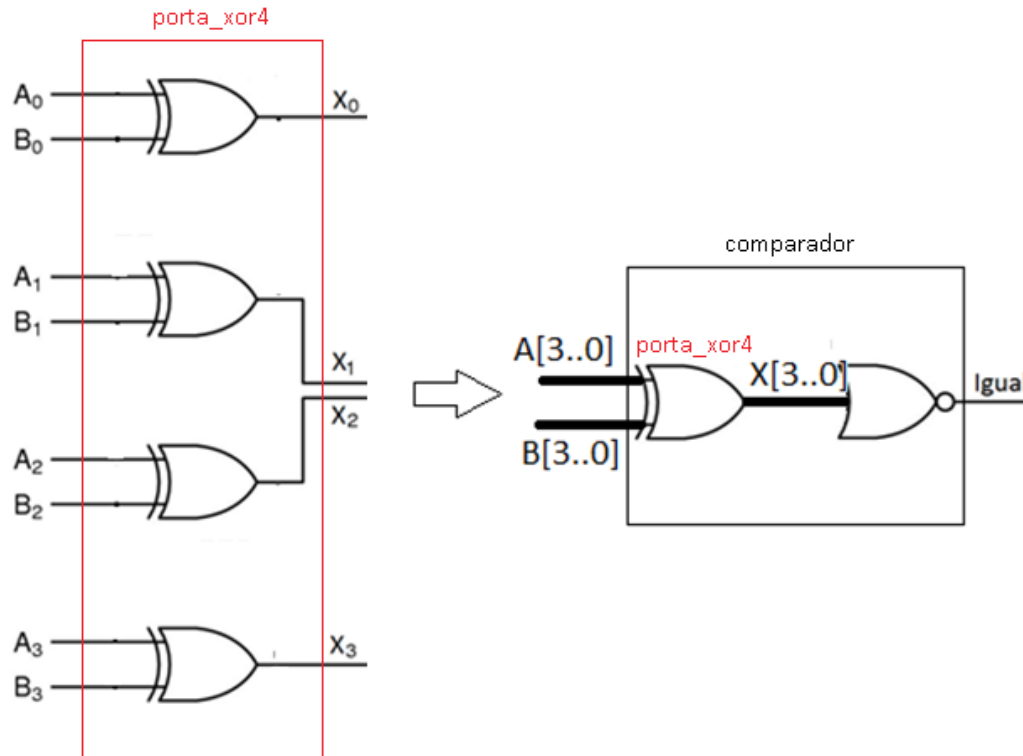


# Prática nº2

## Comparador de Igualdade - Descrição estrutural usando vetores

Com o software Quartus II escreva o projeto em linguagem VHDL para o comparador de igualdade de 4 *bits*, usando arquitetura estrutural (interconexão de componentes usando PORT MAP). Crie um componente porta\_xor4 .Crie os PORTS e sinais como vetores.

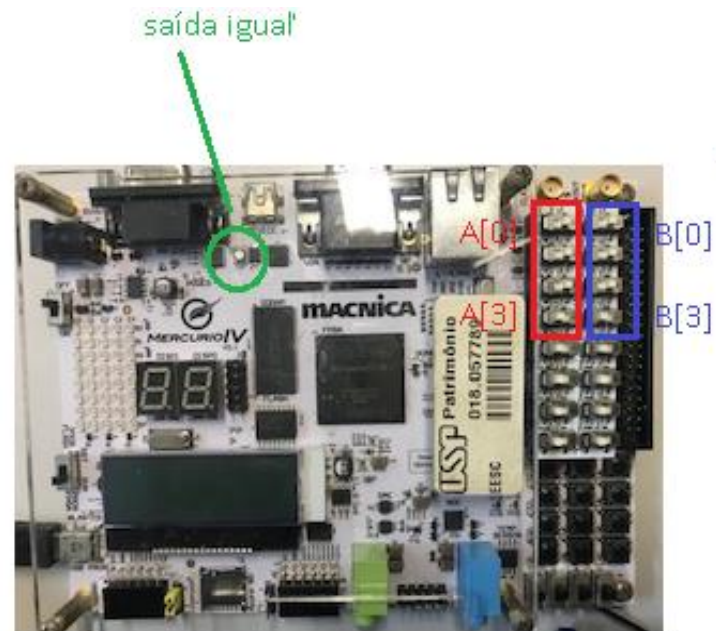
Funcionamento: A saída "Igual" só assume nível lógico alto se as entradas forem iguais ( $A = B$ ).



# Prática nº2

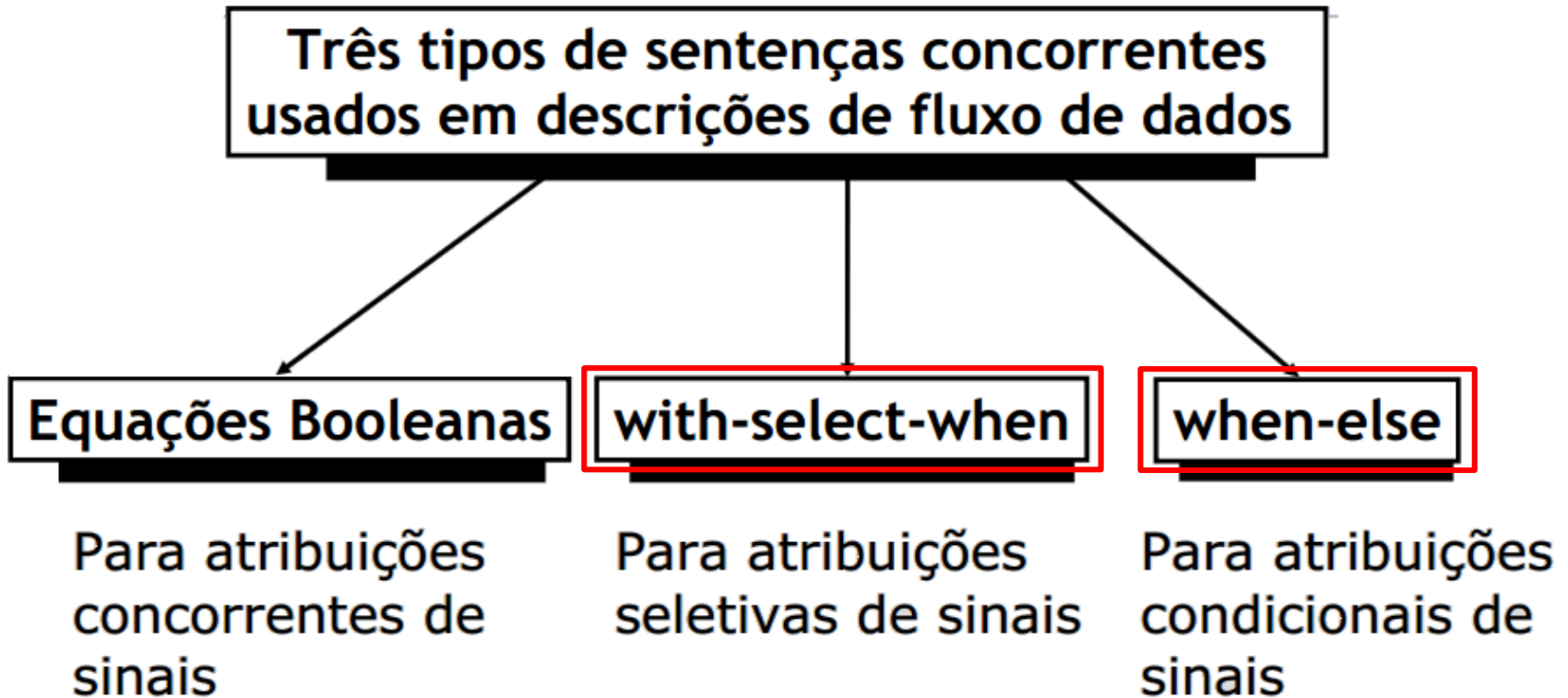
## Pinagem

Nome do sinal	Pino do FPGA	Módulo mercurio
igual	E7	LED_B
A[0]	E16	Chaves da Placa expansora Grupo A
A[1]	H22	
A[2]	F16	
A[3]	F19	
B[0]	J21	Chaves da Placa expansora Grupo B
B[1]	K21	
B[2]	H20	
B[3]	H18	



# ARCHITECTURE – Fluxo de Dados

Descrição por Fluxo de Dados: Comandos (Sentenças) 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).

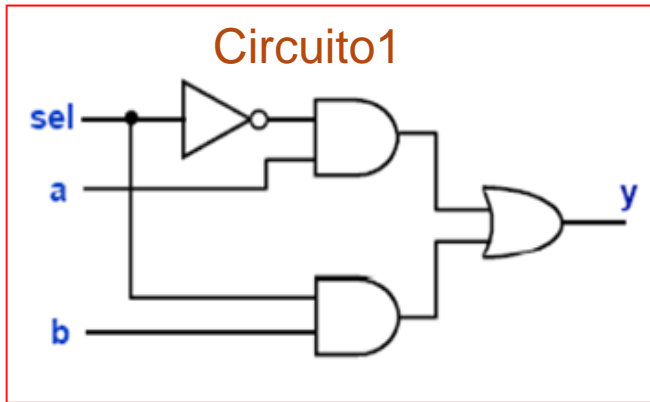
## **ATENÇÃO:**

- Atribuição de valor a sinais do tipo **BIT** ou **STD\_LOGIC**: A <= '1' ou A <= '0'; valor entre aspas simples (' 0 ')
- Atribuição de valor a sinais do tipo **BIT\_VECTOR(1 DOWNT0 )** ou **STD\_LOGIC\_VECTOR (1 DOWNT0 0)**: A <= "11" ou A <= "00"; valor entre aspas dupla ("00")



# ARCHITECTURE – Fluxo de Dados Usando Comando Concorrente **WHEN-ELSE**

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 – Fluxo de Dados Usando Comando Concorrente **WHEN-ELSE**

```
ENTITY circuito1 IS
```

```
  PORT(sel, a, b : 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 a='1' AND b='0') ELSE
```

```
    '1' WHEN (sel='0' AND a='1' AND b='1') ELSE
```

```
    '1' WHEN (sel='1' AND a='0' AND b='1') ELSE
```

```
    '1' WHEN (sel='1' AND a='1' AND b='1') ELSE
```

```
    '0';
```

```
END fluxo_dados;
```

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 – Fluxo de Dados Usando Comando Concorrente WHEN-ELSE

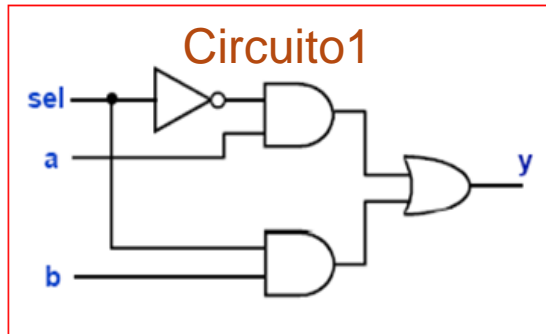
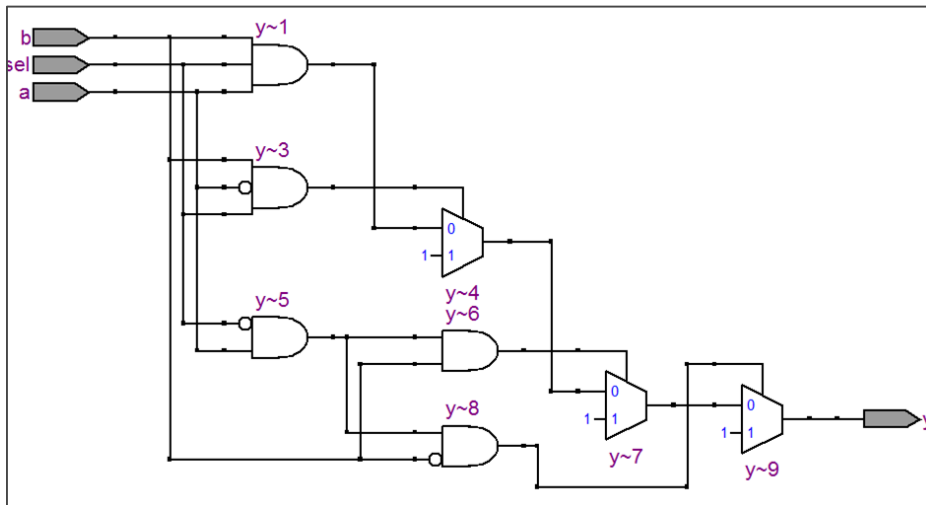


Tabela Verdade do Circuito 1

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

Circuito1 Sintetizado



# 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.

# Palavra reservada **UNAFECTED**

- Pode ser empregada na atribuição de valor a um sinal quando não se deseja afetar o valor do sinal

Formato para construção:

```
sinal_destino <= expressão_a WHEN condição_1 ELSE  
                UNAFECTED WHEN condição_2 ELSE -- valor não alterado na  
condição_2                expressão_b;
```

```
WITH expressão_escolha SELECT  
    sinal_destino <= expressão_a WHEN condição_1,  
    expressão_b WHEN condição_2,  
                UNAFECTED WHEN OTHERS;  
-- valor não alterado nas condições restantes
```

# Exemplo de descrição usando Palavra reservada **UNAFECTED** no uso do comando **WHEN ELSE**

```
ENTITY exemplo1 IS
  PORT(sel : IN BIT_VECTOR(1 DOWNT0 0);
        ya : BUFFER BIT; -- saída deve ser criada como BUFFER
        yu : OUT BIT --pode criar saída como OUT
        );
END exemplo1;

ARCHITECTURE a OF exemplo1 IS
BEGIN
  ya <= '0' WHEN sel = "00" ELSE
  '1' WHEN sel = "10" ELSE
  ya;
-- ya e yu apresentam o mesmo comportamento
  yu <= '0' WHEN sel = "00" ELSE
  '1' WHEN sel = "10" ELSE
  UNAFECTED;
END a;
```

# Exemplo de descrição usando Palavra reservada **UNAFFECTED** no uso do comando **WITH SELECT**

```
ENTITY exemplo2 IS
  PORT(sel : IN BIT_VECTOR(1 DOWNT0 0);
        xa : BUFFER BIT;
        xu : OUT BIT);
END exemplo2;
```

```
ARCHITECTURE a OF exemplo2 IS
BEGIN
```

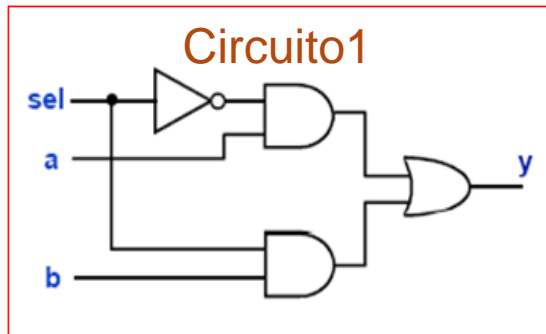
```
    WITH sel SELECT
      xa <= '0' WHEN "00",
        '1' WHEN "10",
      xa  WHEN OTHERS ;
-- xa e xu apresentam o mesmo comportamento
    WITH sel SELECT
      xu <= '0' WHEN "00",
        '1' WHEN "10",
      UNAFFECTED WHEN OTHERS;
END a;
```



# ARCHITECTURE – Fluxo de Dados

## Usando Comando Concorrente **WITH-SELECT**

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 – Fluxo de Dados

## Usando Comando Concorrente **WITH-SELECT**

```
ENTITY circuito1 IS
  PORT(sel, a, b : IN BIT;
        y      : OUT BIT);
END circuito1;
-- Arquitetura por Fluxo de Dados usando o Comando Concorrente
-- WITH-SELECT
ARCHITECTURE fluxo_dados OF circuito1 IS
BEGIN
  WITH sel SELECT
    y <= a WHEN '0',
      b WHEN '1';
-- Outra maneira de descrever:
--   y <= a WHEN '0',
--     b WHEN OTHERS;
END fluxo_dados;
```

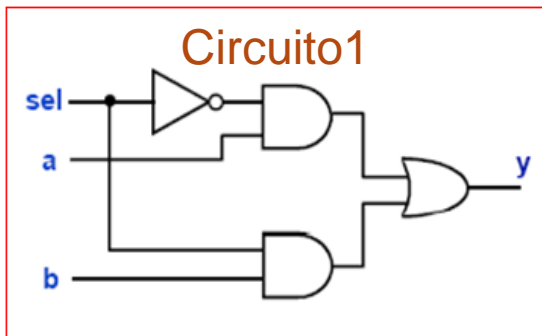
Tabela Verdade simplificada

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

# ARCHITECTURE – Fluxo de Dados

## Usando Comando Concorrente **WITH-SELECT**

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

Circuito1 Sintetizado:

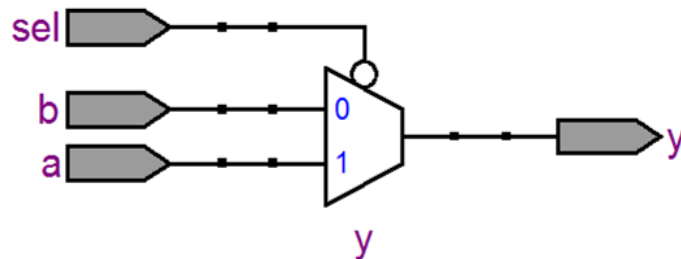


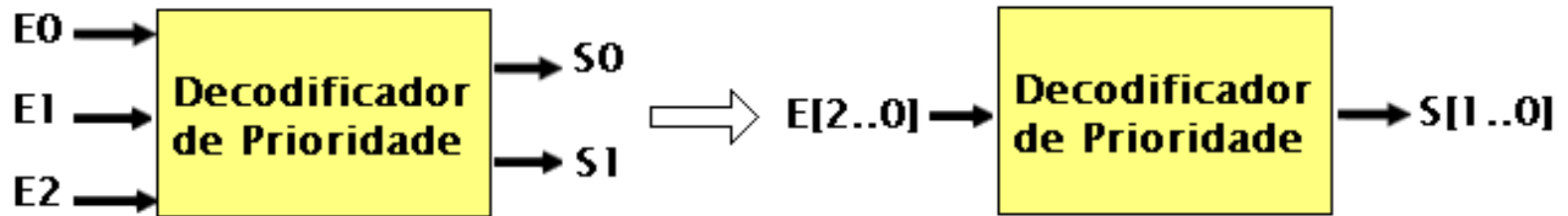
Tabela Verdade simplificada

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

# Prática nº3

## Decodificador de Prioridade – Fluxo de Dados

Descrição por Expressões Lógicas, WHEN\_ELSE e WITH\_SELECT



$$S0 = E2 + \overline{E1}.E0$$
$$S1 = E2 + E1$$

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