

Sistemas Reconfiguráveis

Eng. de Computação

Profs. Francisco Garcia e Antônio Hamilton Magalhães

Aula 4 – Linguagem de descrição de *hardware* (HDL)

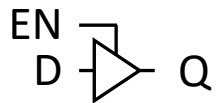
Pacote IEEE-1164

Código concorrente

Revisão – *tri-state*

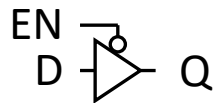
Lógica de três estados (*tri-state*)

Buffer *tri-state*
com *enable* ativo
em nível alto



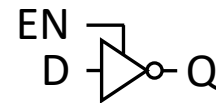
Entradas		Saída
EN	D	Q
0	X	Z
1	0	0
1	1	1

Buffer *tri-state*
com *enable* ativo
em nível baixo



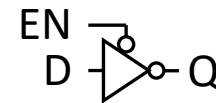
Entradas		Saída
EN	D	Q
1	X	Z
0	0	0
0	1	1

Inversor *tri-state*
com *enable* ativo
em nível alto



Entradas		Saída
EN	D	Q
0	X	Z
1	0	1
1	1	0

Inversor *tri-state*
com *enable* ativo
em nível baixo



Entradas		Saída
EN	D	Q
1	X	Z
0	0	1
0	1	0

VHDL



- Um problema não resolvido pelo padrão VHDL original foi a "lógica de múltiplos valores", onde a força do *drive* de um sinal (nenhum, fraco ou forte) e valores desconhecidos também são considerados.
- Para resolver isso, foi criado, em 1993, o padrão IEEE 1164, que definia os tipos lógicos de 9 valores: `std_logic` escalar e sua versão vetorial `std_logic_vector`. Sendo um subtipo resolvido de seu tipo pai `std_ulogic`, os sinais do tipo `std_logic` permitem múltiplos direcionamentos para modelagem de estruturas de barramento, em que a função de resolução conectada lida com atribuições conflitantes de forma adequada.

VHDL – IEEE 1164

Tipos definidos por IEEE 1164

- STD_LOGIC / STD_LOGIC_VECTOR
- STD_ULOGIC / STD_ULOGIC_VECTOR

Obs.: É necessário declarar biblioteca:

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;
```

VHDL – IEEE 1164

Tipo STD_LOGIC

STD_LOGIC / STD_LOGIC_VECTOR → 8 valores:

- 'X' → desconhecido forçado
- '0' → nível baixo forçado
- '1' → nível alto forçado
- 'Z' → alta impedância
- 'W' → desconhecido fraco
- 'L' → nível baixo fraco
- 'H' → nível alto fraco
- '-' → não importa (*don't care*)

Alguns desses valores são apenas para simulação ou documentação.

'0', '1' e 'Z' são sintetizáveis.

Quando um nó é excitado por mais de uma fonte, o conflito é resolvido pela tabela:

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

VHDL – IEEE 1164

Tipo STD_ULOGIC

STD_ULOGIC / STD_ULOGIC_VECTOR → 9 valores:

- Os 8 valores de **STD_LOGIC** mais o valor 'U' → não iniciado (*uninitialized*)
- O valor U se sobrepõe a todos os demais da tabela apresentada no slide anterior

Obs.: Como o objetivo dessa disciplina é descrever circuitos sintetizáveis, vamos usar, basicamente, os valores '0', '1' e 'Z'. Eventualmente, vamos atribuir o valor '-' a objetos com a finalidade de obter a máxima simplificação possível. Isso será visto posteriormente


Obs.: A grande maioria dos PLDs possui pinos com capacidade de trabalhar com lógica *tri-state*, para poderem ser usados como entrada e/ou saída (I/O), mas isso não se aplica à lógica interna dos dispositivos. Entretanto, quando esse comportamento é descrito para um nó interno do dispositivo, em muitos casos o compilador é capaz de criar um circuito funcionalmente equivalente, porém com uma estrutura completamente diferente.

Códigos em VHDL

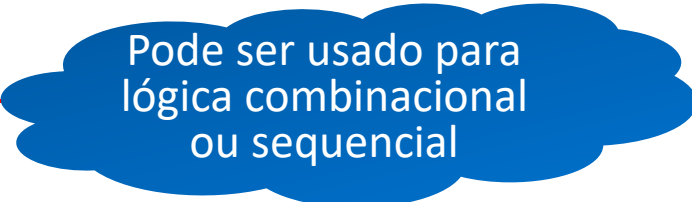


Existem dois tipos de código em VHDL:

- Código concorrente (paralelo)
- Código sequencial
 - Tem de estar dentro de um **PROCESS**



Geralmente usado
para lógica
combinacional



Pode ser usado para
lógica combinacional
ou sequencial

Lógica combinacional : A saída atual depende apenas do estado atual das entradas.

Lógica sequencial : A saída atual depende não só do estado atual das entradas mas também dos estados anteriores; deve existir uma memória (flip-flop).

Código concorrente



A linguagem VHDL é inerentemente concorrente (paralela)

- Todas as expressões e declarações são avaliadas ao mesmo tempo (exceto o que está dentro de um **PROCESS**)
- A ordem das expressões não importa
- O código concorrente é usado, em geral, para lógica combinacional
- No código concorrente é permitido:
 - Uso direto dos operadores
 - A construção **WHEN / ELSE**
 - A construção **WITH / SELECT / WHEN**

Código concorrente

Exemplos:

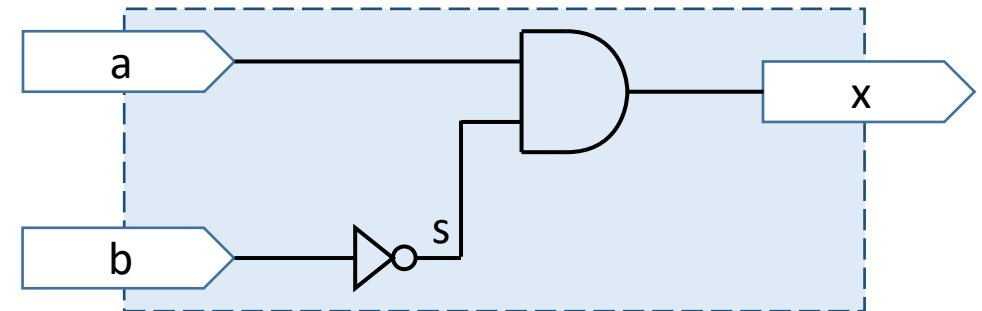
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----

ENTITY Cap4ex1 IS
    PORT (
        a, b : IN STD_LOGIC;
        x    : OUT STD_LOGIC
    );
END cap4ex1;

-----

ARCHITECTURE arch OF Cap4ex1 IS
    SIGNAL s : STD_LOGIC;
BEGIN
    s <= NOT b;
    x <= a AND s;
END arch;
```



```
ARCHITECTURE arch OF Cap4ex1 IS
    SIGNAL s : STD_LOGIC;
BEGIN
    x <= a AND s;
    s <= NOT b;
END arch;
```

A ordem não importa

Usaremos o tipo **STD_LOGIC** em todos os exemplos e exercícios

Código concorrente

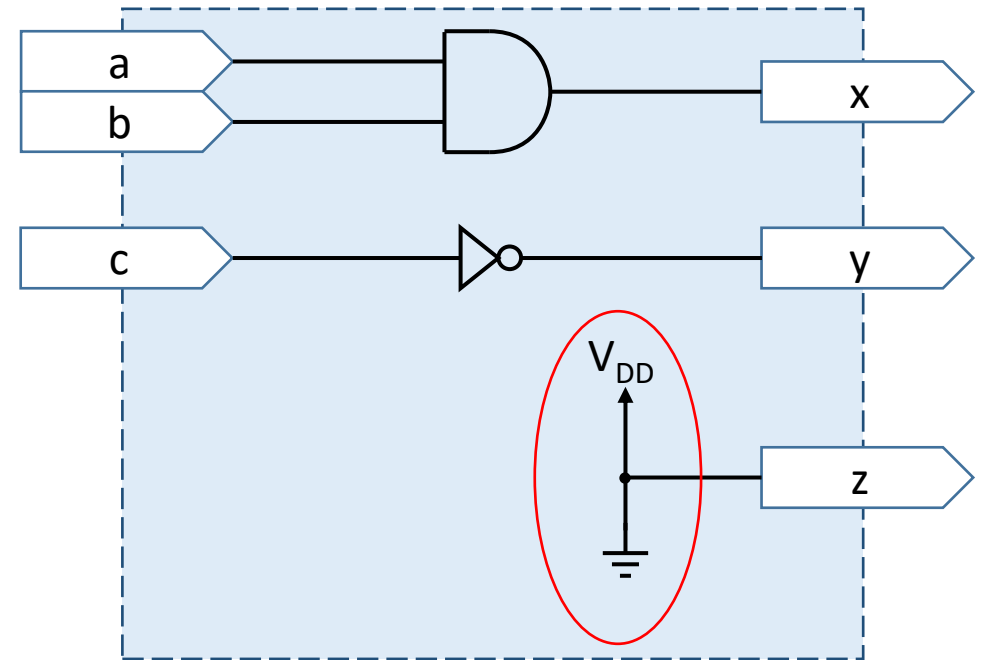
Exemplos:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----
ENTITY Cap4ex2 IS
    PORT (
        a, b, c : IN STD_LOGIC;
        x, y, z : OUT STD_LOGIC
    );
END Cap4ex2;

-----
ARCHITECTURE arch OF Cap4ex2 IS
BEGIN
    z <= '1';
    x <= a AND b;
    y <= NOT c;
    z <= '0';
END arch;
```

Illegal.
Múltiplas
atribuições



Construção WHEN / ELSE

- Sintaxe:

```
signal <= expression1 WHEN condition1 ELSE  
expression2 WHEN condition2 ELSE  
.....  
expressionN;
```

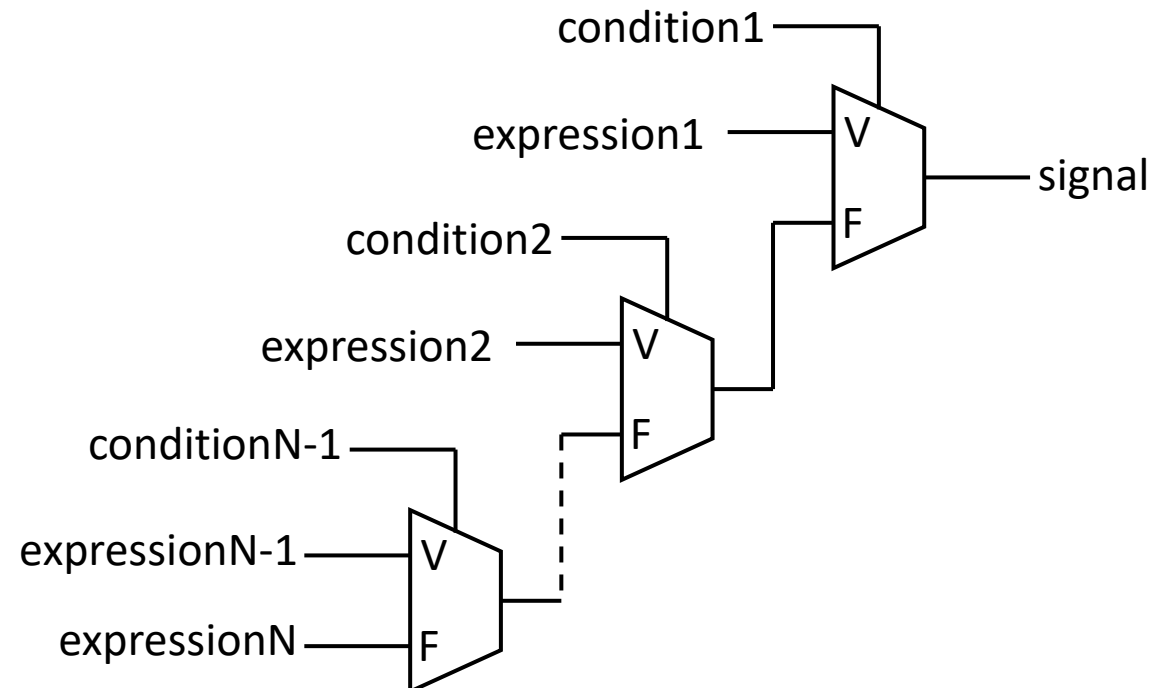
Apenas 1
operador de
atribuição

Termina
com ";"

verdadeira
/ falsa

Não usa
separadores

Obs.: Não existe IF / THEN / ELSE no código concorrente



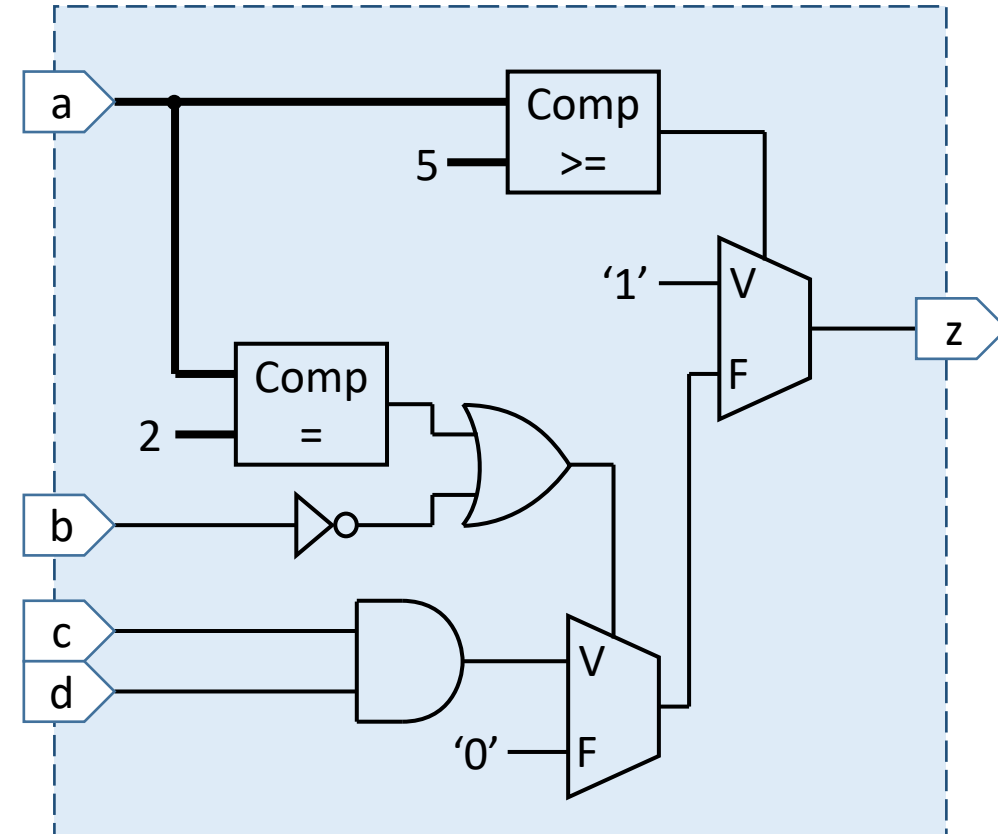
Construção WHEN / ELSE

Exemplos:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

-----
ENTITY Cap4ex3 IS
    PORT (
        a : IN INTEGER RANGE 0 TO 7;
        b, c, d : IN STD_LOGIC;
        z : OUT STD_LOGIC
    );
END Cap4ex3;

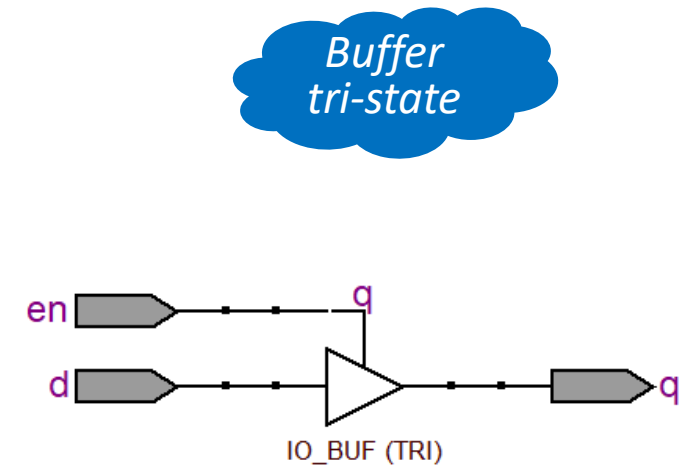
-----
ARCHITECTURE arch OF Cap4ex3 IS
BEGIN
    z <= '1' WHEN a >= 5 ELSE
        c AND d WHEN a = 2 OR b = '0' ELSE
        '0';
END arch;
```



Construção WHEN / ELSE

Exemplos:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----
ENTITY Cap4ex4 IS
    PORT (
        d, en : IN STD_LOGIC;
        q : OUT STD_LOGIC
    );
END ENTITY;
-----
ARCHITECTURE arch OF Cap4ex4 IS
BEGIN
    q <= d WHEN en = '1' ELSE 'Z';
END arch;
```



Circuito RTL do exemplo Cap4ex4

Esse circuito foi gerado
pela ferramenta
RTL Viewer do Quartus II

Construção WITH / SELECT / WHEN

- Sintaxe:

WITH identifier **SELECT**

signal <= expression1 **WHEN** value1,

expression2 **WHEN** value2,

.....

expressionN **WHEN** valueN;

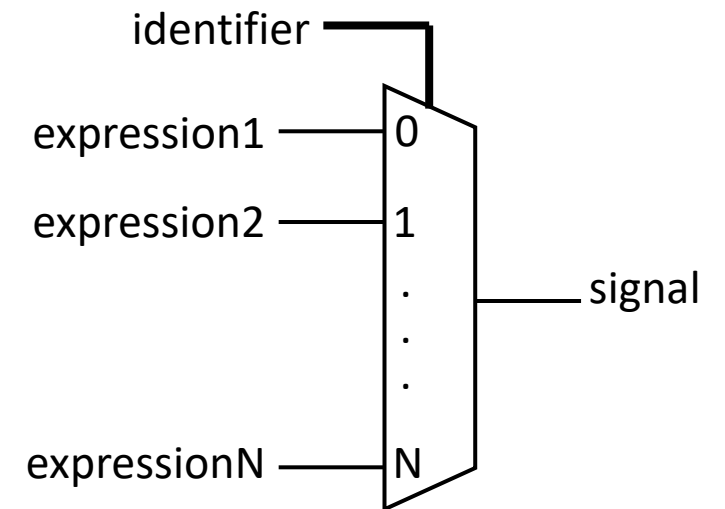
Valores de
identifier

Usa “,” para
separar

Apenas 1
operador de
atribuição

Termina
com “;”

Obs.: Não existe CASE / SELECT no código
concorrente



Construção WITH / SELECT / WHEN

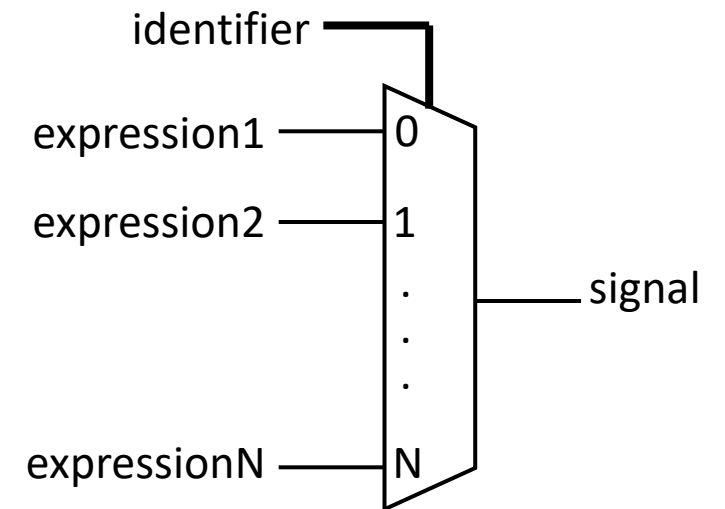
- Obs:
- A ordem dos valores não importa
- Todos os possíveis valores de **identifier** têm de ser testados
- Se necessário, o último pode ser **WHEN OTHERS**
- Podem ser testados mais de um valor por linha:

WHEN value1 | value2,

WHEN value3 **TO** value7,

Faixa de
valores

A “|”
significa
OR



Construção WITH / SELECT / WHEN

Exemplos:

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
-----  
ENTITY Cap4ex5 IS  
    PORT (  
        a, b, c : IN STD_LOGIC;  
        s       : OUT STD_LOGIC  
    );  
END ENTITY;
```

Continua na próxima página

Construção WITH / SELECT / WHEN

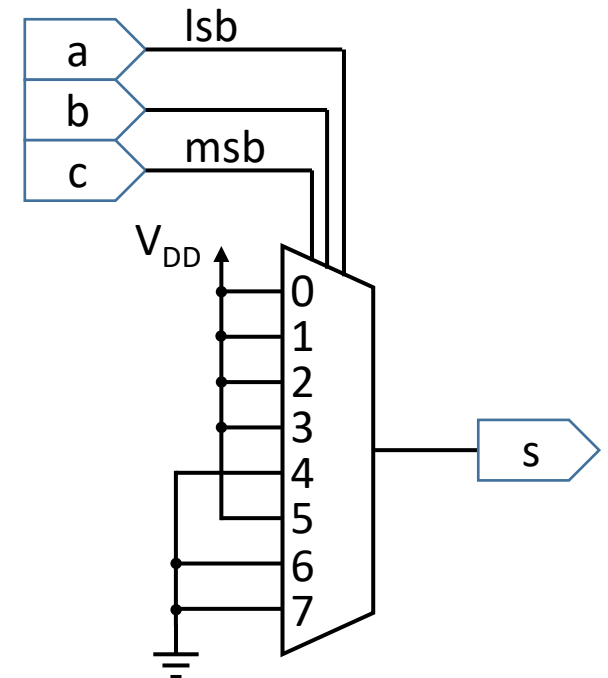
Exemplos:

```
ARCHITECTURE arch1 OF Cap4ex5 IS
    SIGNAL entradas : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    entradas <= c & b & a;

    WITH entradas SELECT
        s <= '1' WHEN "000",
              '1' WHEN "001",
              '1' WHEN "010",
              '1' WHEN "011",
              '0' WHEN "100",
              '1' WHEN "101",
              '0' WHEN "110",
              '0' WHEN "111";
END arch1;
```

Operador de concatenação

Aqui os valores foram inseridos em ordem crescente mas, na realidade, a ordem não interessa.



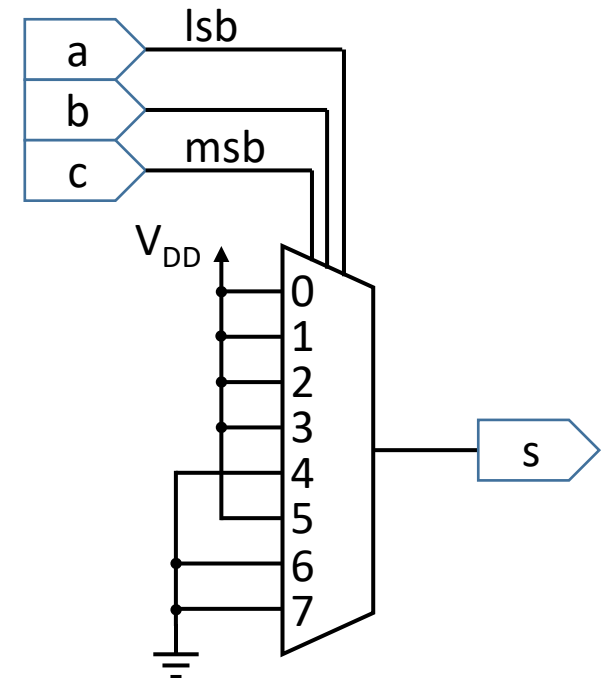
Construção WITH / SELECT / WHEN

Exemplos:

```
ARCHITECTURE arch2 OF Cap4ex5 IS
    SIGNAL entradas : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    entradas <= c & b & a;

    WITH entradas SELECT
        s <= '0' WHEN "100",
              '0' WHEN "110",
              '0' WHEN "111",
              '1' WHEN OTHERS;
END arch2;
```

Esse código é
equivalente
ao anterior



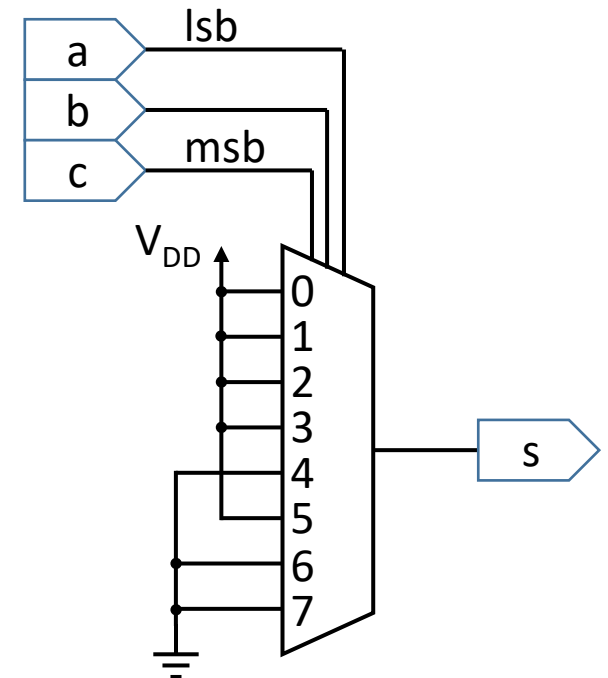
Construção WITH / SELECT / WHEN

Exemplos:

```
ARCHITECTURE arch3 OF Cap4ex5 IS
    SIGNAL entradas : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    entradas <= c & b & a;

    WITH entradas SELECT
        s <= '0' WHEN "100" | "110" | "111",
            '1' WHEN OTHERS;
END arch3;
```

Esse código
também é
equivalente
ao anterior



Circuitos MSI



Vamos ver agora e nas próximas aulas uma série de exemplos de descrição de circuitos combinacionais de média complexidade (MSI = *medium scale integration*) usando as construções estudadas do código concorrente de VHDL.

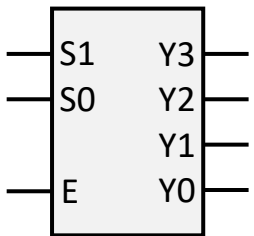
Decodificadores (*decoder*)



- Poucas entradas e muitas saídas
- Apenas uma saída ativa de cada vez
- A entrada de seleção (N bits) indica qual saída a ser ativada
- Pode ter até 2^N saídas
- As saídas podem ser ativas em nível lógico alto ou baixo
- Pode ter nenhuma, uma ou mais entradas de habilitação (*enable*)
- A(s) entrada(s) de habilitação pode(m) ser ativa(s) em nível alto ou baixo

Exemplo dec_2x4

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e *enable* ativo em nível alto



Entradas			Saídas			
E	S1	S0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

E = '0' => desabilitado
Nenhuma saída ativa.

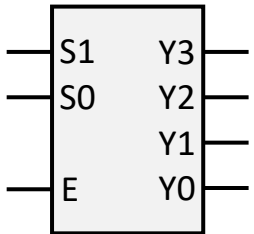
Nesse caso, a entrada
S não interessa

E = '1' => habilitado
Uma única saída ativa.

A entrada S determina
qual saída ficará ativa

Exemplo dec_2x4

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto



Entradas			Saídas			
E	S1	S0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

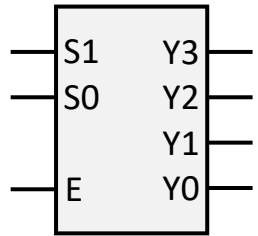
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec_2x4 IS
    PORT (
        s: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
        e: IN STD_LOGIC;
        y: OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
    );
END ENTITY;
```

Continua na próxima página

Exemplo dec_2x4 arch1

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto



Entradas			Saídas			
E	S1	S0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

```
ARCHITECTURE arch1 OF dec_2x4 IS
    SIGNAL entradas  : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN

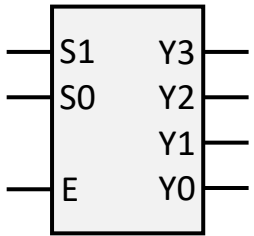
    entradas <= e & s;

    WITH entradas SELECT
        y <= "0001" WHEN "100",
            "0010" WHEN "101",
            "0100" WHEN "110",
            "1000" WHEN "111",
            "0000" WHEN OTHERS;    -- disabled

END arch1;
```


Exemplo dec_2x4 arch2

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto



Entradas			Saídas			
E	S1	S0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

```
ARCHITECTURE arch2 OF dec_2x4 IS
    SIGNAL aux : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN

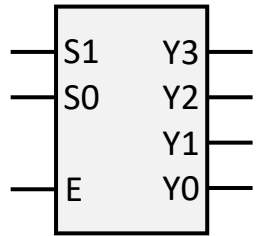
    WITH s SELECT
        aux <= "0001" WHEN "00",
               "0010" WHEN "01",
               "0100" WHEN "10",
               "1000" WHEN "11";

    y <= aux WHEN e = '1' ELSE -- when enabled
        "0000";               -- when disabled

END arch2;
```

Exemplo dec_2x4 arch3

Exemplo dec_2x4: decodificador duas entradas de seleção, quatro saídas ativas em nível alto e enable ativo em nível alto



Entradas			Saídas			
E	S1	S0	Y3	Y2	Y1	Y0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

```
ARCHITECTURE arch3 OF dec_2x4 IS  
BEGIN
```

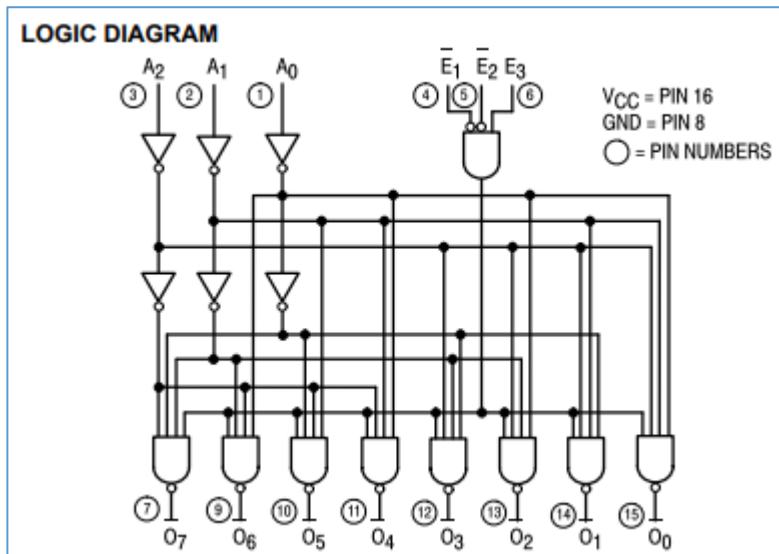
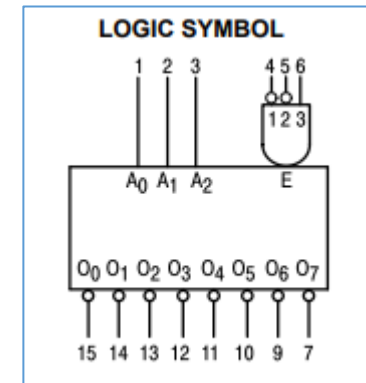
```
    y(0) <= '1' WHEN e = '1' AND s = "00" ELSE '0';  
    y(1) <= '1' WHEN e = '1' AND s = "01" ELSE '0';  
    y(2) <= '1' WHEN e = '1' AND s = "10" ELSE '0';  
    y(3) <= '1' WHEN e = '1' AND s = "11" ELSE '0';
```

```
END arch3;
```

Obs.: Só pode haver uma **ARCHITECTURE** ativa em um projeto.
As outras devem estar comentadas.

Exercício 1

Implementar um decodificador funcionalmente equivalente ao circuito integrado TTL 74LS138.



TRUTH TABLE													
INPUTS						OUTPUTS							
E ₁	E ₂	E ₃	A ₀	A ₁	A ₂	O ₀	O ₁	O ₂	O ₃	O ₄	O ₅	O ₆	O ₇
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

Exercício 1

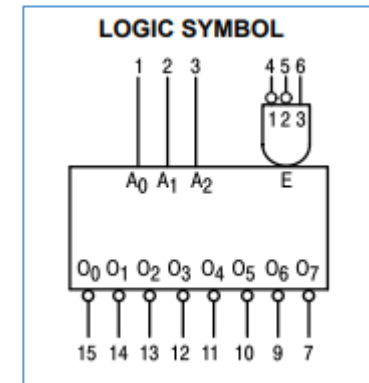
Implementar um decodificador funcionalmente equivalente ao circuito integrado TTL 74LS138.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dec_74LS138 IS
    PORT (
        a: IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- seleção
        ne1, ne2, e3: IN STD_LOGIC;          -- enable
        no: OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
    );
END ENTITY;

.....
```

Comentários





Fim

Até a próxima aula!