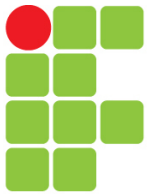
VHDL: Sinais e operadores

Engenharia Eletrônica

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

16 de setembro de 2019



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópico

Objetivos

- Tópicos da aula de hoje:
  - Constantes
  - Sinais
  - Atribuição de sinais
  - Operadores

VHDL

Vhsic (Very High Speed Integrated Circuit)
Hardware
Description
Language

O que é VHDL?

- Padrão industrial IEEE para linguagem de descrição de **hardware**
- Linguagem de descrição em **alto nível** para simulação e síntese

Objetos VHDL e suas classes

- Objetos:
  - Usados para descrever a funcionalidade de um módulo
  - Atribuídos valores e tipos

- Classes de objetos:
  - Define o comportamento do objeto e que operações podem ser realizadas
  - Tipos:
    - CONSTANT
    - SIGNAL
    - VARIABLE
    - FILE

Tópico

Constantes

- Associam um valor a um nome
- Declaração:
  - Pode ser declarada em uma ENTITY, ARCHITECTURE ou PACKAGE

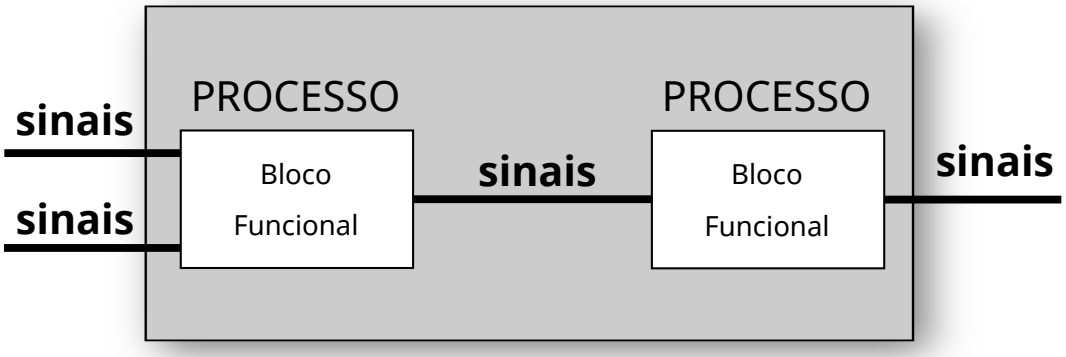
```
constant <nome> : <tipo_de_dado> := <valor>;
constant bus_width : integer := 16;
```

- Não podem ser alteradas em tempo de execução
  - Note que *generics* são constantes (parâmetros) que podem ser sobrescritos passando novos valores durante **compilação**, mas não em execução
- Melhora-se legibilidade do código
- Melhora-se flexibilidade do código

Tópico

Sinais

- Sinais representam interconexões físicas (fios) que se comunicam entre processos (funções)



- Declaração:
  - Pode ser declarada em uma ENTITY, ARCHITECTURE ou PACKAGE

```
signal temp : std_logic_vector(7 downto 0);
```

Atribuição de valores ao sinais

```
signal temp : std_logic_vector(7 downto 0);
```

- Atribuição de sinais utiliza o operador: <=
- Exemplos:
  - Todos os bits:

```
temp <= "10101010";
temp <= x"aa" ; -- (1076-1993)
-- VHDL suporta 'o' para octal e 'b' para binário
```

  - Parcial:

```
temp (7 downto 4) <= "1010";
```

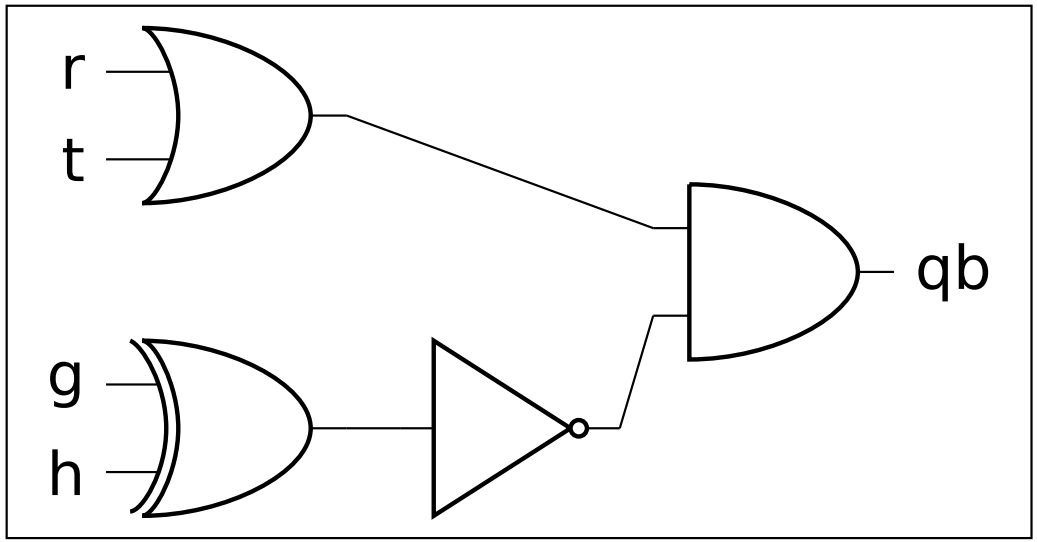
  - Bit:

```
temp(7) <= '1';
```

- Use aspas normais (“ ”) para atribuição multi-bit e apas simples (’ ’)para bit único

Sinais como interconexão

```
library ieee;
use ieee.std_logic_1164.all;

entity simp is
  port (
    r, t, g, h : in std_logic;
    qb : out std_logic
  );
end entity simp;

architecture logic of simp is

  signal qa : std_logic;

begin

  qa <= r or t;
  qb <= (qa and not (g xor h));

end architecture logic;
```



Declaração de sinal dentro da arquitetura

— r, t, g, h, e qb são sinais por padrão
— qa precisa ser declarado como conexão intermediária

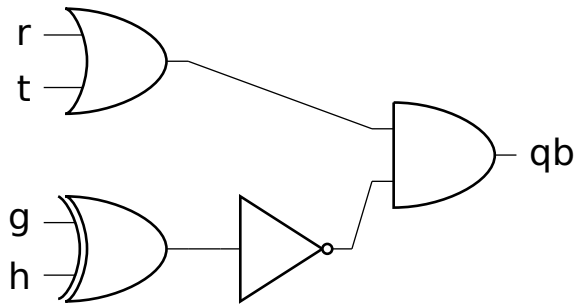
Atribuição de sinais concorrentes

- Atribuição de sinais usando expressões
- Representam um processo **implícito** concorrente
  - Estas atribuições são sensíveis a todos os sinais à direita da atribuição
- Três tipos:
  - Atribuição simples
  - Atribuição condicional
  - Atribuição seletiva

Atribuição simples

- Formato: < *nome_sinal* ><=< *expressao* >;
- Exemplo:

```
-- 2 processos implícitos
qa <= r or t;
qb <= (qa and not (g xor h));

-- parenteses dão a ordem de precedência
```



- Expressões utilizam operadores VHDL para descrever comportamento

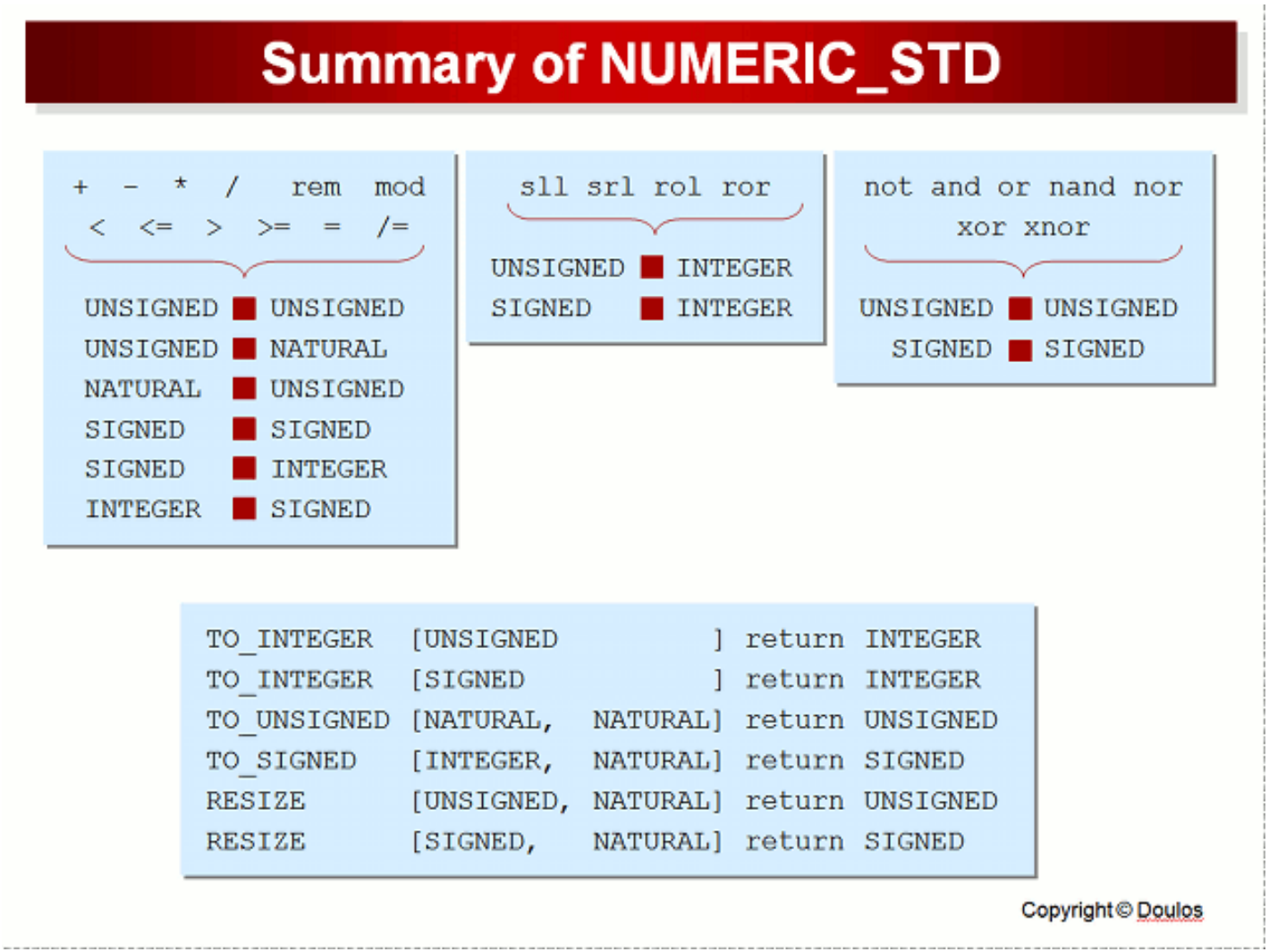
Tópico

Operadores VHDL

Tipo		Nome/Símbolo	Prioridade
Lógico		not and or nand nor xor xnor	Mais alta
Relacional		= /= < <= > >=	
Shift (1)(2)		sll srl sla sra rol ror	
Aritmético	Adição e sina	+ -	
	Concatenação	&	
	Multiplicação	* / mod rem	
Outros		** abs	Mais baixa

- **: Exponencial
- abs: Valor absoluto
- (1): Não suportado no VHDL'97
- (2): Suportado pelo pacote NUMETIC_STD para tipos SIGNED/UNSIGNED

Operadores VHDL



Funções aritméticas

```
entity opr is
  port (
    a : in integer range 0 to 16;
    b : in integer range 0 to 16;
    sum : out integer range 0 to 32
  );
end entity opr;
```

```
architecture example of opr is
begin
  sum <= a + b;
end architecture example;
```

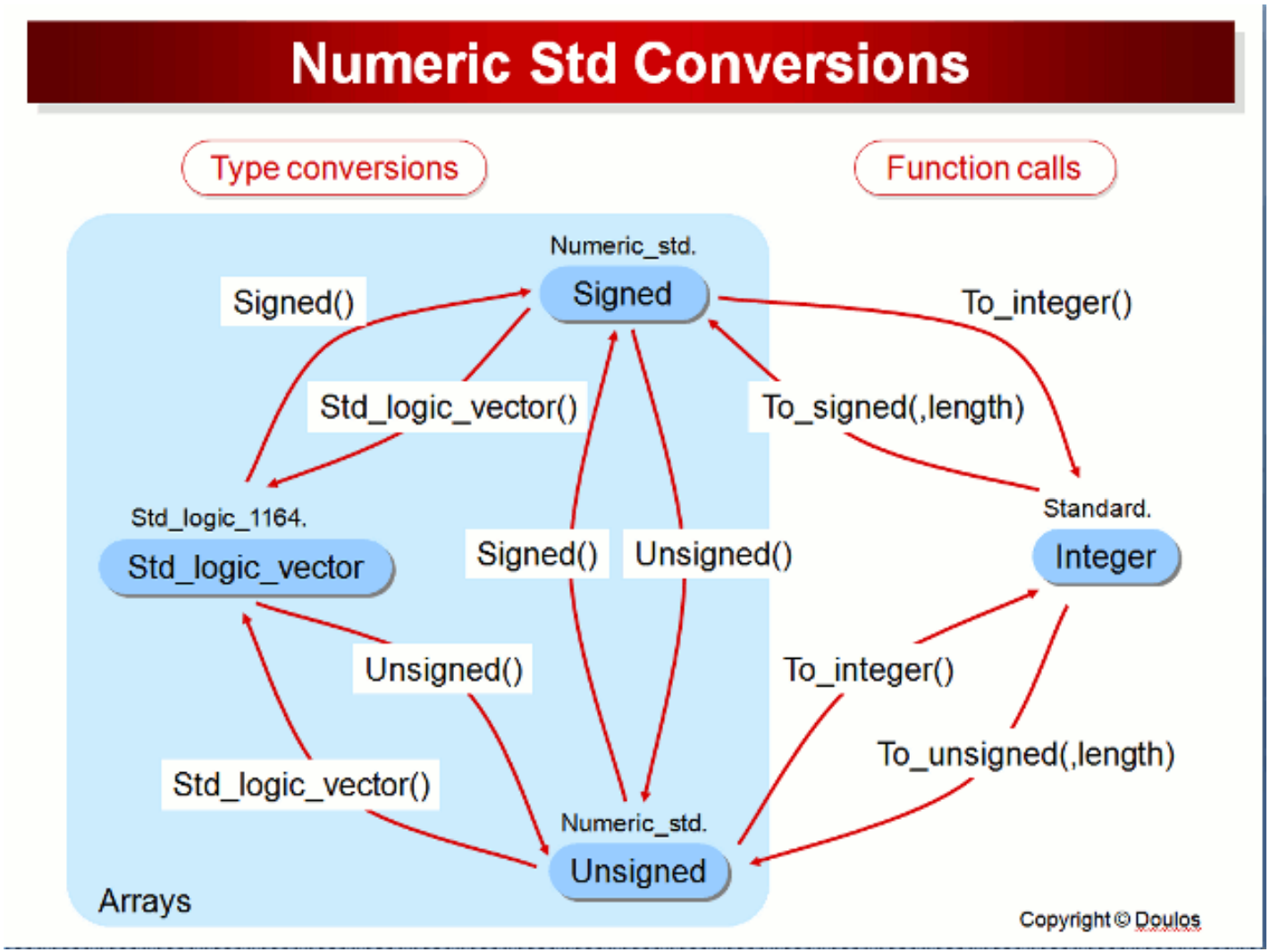
O compilador VHDL entende esta operação porque o operador '+' é automaticamente definido para o tipo INTEGER.

- Note que a biblioteca STD e o pacote STANDARD não precisam ser referenciados.

Sobrecarga de operadores

- A linguagem VHDL define operações aritméticas e lógicas apenas para tipos definidos no pacote STANDARD
- Para usar outros tipos, deve-se **converter** utilizando funções.
- **Antigamente** utilizava-se pacotes que sobrecarregam operadores na biblioteca **IEEE**, mas estão **desatualizados**.
  - **STD_LOGIC_ARITH**: funções aritméticas
  - **STD_LOGIC_SIGNED**: funções aritméticas sinalizadas
  - **STD_LOGIC_UNSIGNED**: funções aritméticas não sinalizadas

Conversão de tipos



Conversão de tipos

A não ser que seja necessário utilizar **std_logic_vector**, é preferível utilizar os tipos **signed** ou **unsgigned** diretamente, especialmente para **sinais** mas também para **portas**. Evita-se execesso de conversão e mehora-se a documentação.

- Valores numéricos positivos (contadores, endereços): **unsigned**.
- Valores positivos e negativos (dados): **signed**.
- Valores não numéricos (sinais de controle, agredados): **std_logic(_vector)**

Uso de operadores

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

Inclusão das bibliotecas para impementar os operadores.

```
entity overload is
    port (
        a : in std_logic_vector (4 downto 0);
        b : in std_logic_vector (4 downto 0);
        sum : out std_logic_vector (4 downto 0)
    );
end entity overload;
```

Tipos IEE

```
architecture example of overload is
begin
    sum <= a + b;
end architecture example;
```

Deve-se converter para operar. Este código está incorreto.

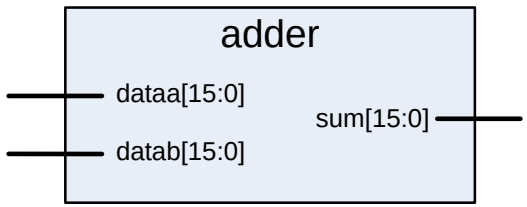
Exercício 1

- Corrija o somador de 4-bits.

- Simule no ModelSim e verfique o funcionamento.

Exercício 2

- Objetivos:
  - Construir um somador com o operador '+'
  - Praticar a estrutura ENTITY-ARCHITECTURE
  - Verificar o efeito das bibliotecas na compilação
- Escreva o código para um somador de n-bits
  - Use GENERICS para mudar o tamanho da portas de acordo com a necessidade
  - Use os nomes conforme o diagrama abaixo (bloco, portas, e caixa minúscula)
  - Entradas e saídas devem ser declaradas como **unsigned**
  - Use o pacote **ieee.numeric_standard.all** para operações aritméticas.



Exercício 3

- Objetivos:
  - Construir um multiplicador com o operador ’*’
  - Praticar a estrutura ENTITY-ARCHITECTURE
  - Verificar o efeito das bibliotecas na compilação
- Escreva o código para um somador de n-bits
  - Use GENERICS para mudar o tamanho da portas de acordo com a necessidade
  - Use os nomes conforme o diagrama abaixo (bloco, portas, e caixa minúscula)
  - Entradas e saídas devem ser declaradas como **unsigned**
  - Use o pacote **ieee.numeric_standard.all** para operações aritméticas.

Tópico

Atribuição condicional

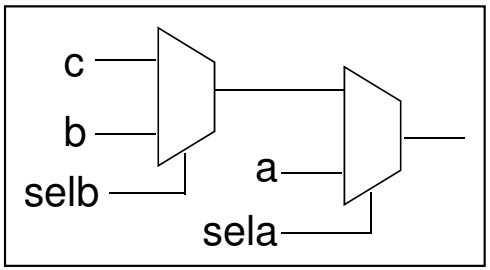
- Formato:

```
<nome_sinal>    <=  <sinal/valor> WHEN <condicao_1> ELSE
                    <sinal/valor> WHEN <condicao_2> ELSE
                    (...)
                    <sinal/valor> WHEN <condicao_n> ELSE
                    <sinal/valor>;
```

- Exemplo:

```
q <= a when sela = '1' else
     b when selb = '1' else
     c;
```

Processo implícito



Atribuição condicional – exemplos

```
s0 <= i0 or i1  when na = 8 or nb = 2  else
      i1        when na = 3 and nb = 5 else
      i0 and i1 when na = 7 else
      i0;
```

Construção composta

```
entity mux_1 is
  port (
    i0, i1, i2, i3 : in bit;
    s0, s1 : in bit;
    ot : out bit);
end mux_1;

achitecture test of mux_1 is
begin
  ot <= i0 when s1 = '0' and s0='0' else
        i1 when s1 = '0' and s0='1' else
        i2 when s1 = '0' and s0='0' else
        i3;
end teste;
```

Mux

Atribuição selecionável

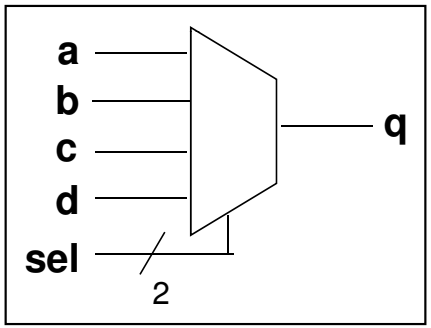
- Formato:

```
WITH <expressao>  SELECT
  <nome_sinal> <=  <sinal/valor> WHEN <condicao_1>
                   <sinal/valor> WHEN <condicao_2>
                   (...)
                   <sinal/valor> WHEN OTHERS;
```

- Exemplo:

```
WITH sel SELECT
  q <= a when "00",
       b when "01",
       c when "10",
       d when others;
```

Processo implícito



- Todas as condições **devem** ser consideradas
- **WHEN OTHERS** considera todas as condições não consideradas nas cláusulas anteriores

Atribuição condicional – exemplos

```
with s0 select  -- s0 é tipo caracter
  x0 <= i0 and i1 when 'a',
        i0 or i1  when 'b' | 'c',
        i0 xor i1 when 'd' to 'g',
        i0        when 'x' downto 'k'
        i1 when others;

with b1 and b0 select -- b1 e b0 tipo bit
  x1 <= i0 when '0',
        i1 when '1';
```

Construção composta

```
entity mux_2 is
  port (
    i0, i1, i2, i3 : in bit;
    s0, s1  : in bit;
    ot : out bit);
end mux_1;

achitecture test of mux_1 is
  signal sel : bit_vector(1 downto 0);
begin
 sel <= s1 & s2;

 with sel select
  ot <= i0 when "00",
        i1 when "01",
        i2 when "10",
        i3 when "11";
end teste;
```

Mux

Atribuição condicional – exemplos

```
library ieee;
use ieee.std_logic_1164.all;

entity cmpl_sig is
  port (
    a, b, sel : in std_logic;
    z : out std_logic
  );
end entity cmpl_sig;

architecture logic of cmpl_sig is
begin

  with sel select
     z <= a when '0',
          b when '1',
         '0' when others;

end architecture logic;
```

sel é **STD_LOGIC**

Possíveis valores de STD_LOGIC são '0', '1', 'X',...

portanto **WHEN OTHERS** é necessário para 'X', ...

Atribuição concorrente de sinais

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity cmpl_sig is
    port (
      a, b, sel : in std_logic;
      x, y, z : out std_logic
    );
end entity cmpl_sig;

architecture logic of cmpl_sig is
begin
  -- atribução simples
  x <= (a and not sel) or (b and sel);

  -- atribuição condicional
 y <= a when sel='0' else
      b;

 -- atribuição selecionada
 with sel select
  z <= a when '0',
       b when '1',
      'x' when others;

end architecture logic;
```

Atribuição de sinais acontece em paralelo,

portanto a ordem das cláusulas

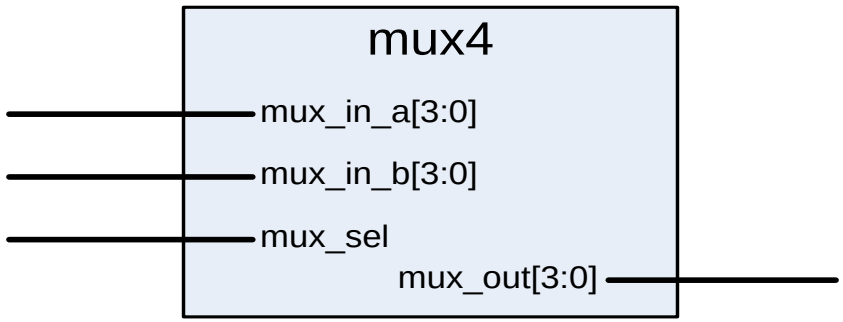
não afeta o funcionamento

Comparação entre “when else” e “with select”

- Na construção “WHEN ELSE”:
  - ordem das cláusulas indica precedência
  - a última cláusula tem **menor** prioridade
- Na construção “WITH SELECT”:
  - Todas as condições possuem a mesma prioridade
  - Todas possíveis entradas **devem** ser avaliadas
- Sem otimização, circuitos mais eficientes podem ser projetados
- Na síntese, ferramentas mais modernas podem gerar a mesma construção

Exercício

- Construa um multiplexador de 4-bits:
  - Crie uma versão com SELECT.
  - Crie outra versão com WHEN.
  - Entradas e saídas devem ser declaradas como unsigned (compatível com o somador).



- Construa componente MUXs e DEMUX com palavras variáveis (generics).
- Pesquise se é possível fazer um MUX completamente genérico: número de postas e bits das palavras.
- Lista Moodle.

Tópico

Referências

- *Introduction to Altera Devices and Design Software*

- PEDRONI, Volnei A. Eletrônica digital moderna e VHDL. Rio de Janeiro: Elsevier, 2010. 619 p., ISBN 9788535234657.