

# Modelagem de Circuitos Combiacionais usando VHDL

Revisão 2023-2

Universidade do Vale do Itajaí– UNIVALI

Laboratory of Embedded and Distributed Systems – LEDS



---

# Introdução ao VHDL

- ⦿ Definição
- ⦿ Fluxos de simulação e síntese
- ⦿ Aspectos gerais

# O que é VHDL?

---

- ⦿ É uma linguagem de descrição de hardware
- ⦿ Desenvolvida a partir da necessidade do Departamento de Defesa (DoD) dos EUA para unificar a documentação de projetos de seus fornecedores no contexto do programa VHSIC, substituindo os diagrama esquemáticos
- ⦿ **VHDL = VHSIC HDL**
  - ⦿ VHSIC = Very High Speed Integrated Circuit
  - ⦿ HDL = Hardware Description Language

# O VHDL ou a VHDL?

---

- ⦿ Assim como toda e qualquer linguagem de programação, em português, costuma-se “masculinizar” o gênero da linguagem
  - ⦿ o C
  - ⦿ o Python
  - ⦿ o JAVA
  - ⦿ o Assembly
  - ⦿ o VHDL

# Cronologia e uso do VHDL

---

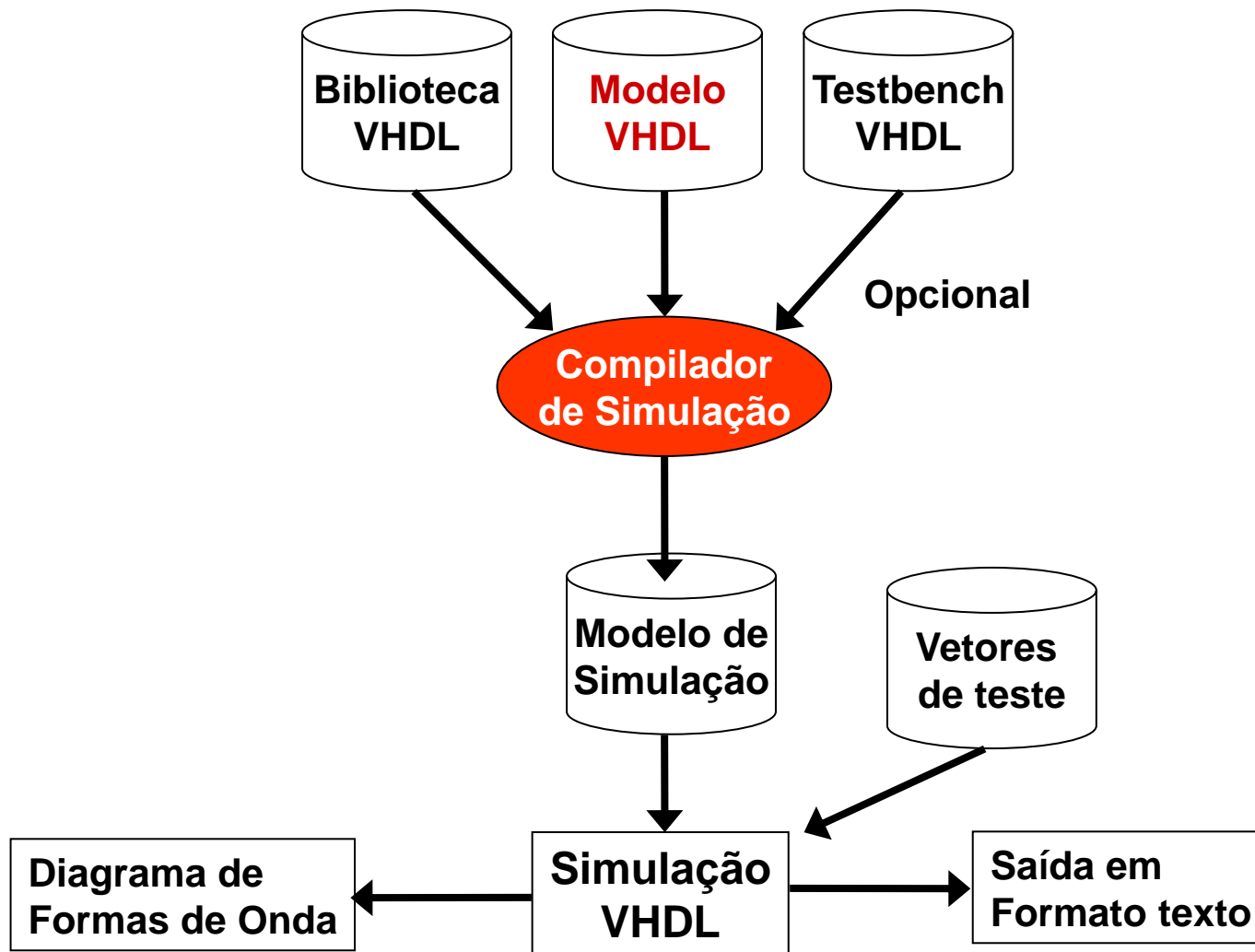
## ⦿ Marcos históricos

- ⦿ 1980: desenvolvimento da linguagem
- ⦿ 1987: padronização pelo IEEE (IEEE Std 1076-1987)
- ⦿ 1993: revisão do padrão (IEEE Std 1076-1993)
- ⦿ 2008: nova revisão (IEEE Std 1076-2008)

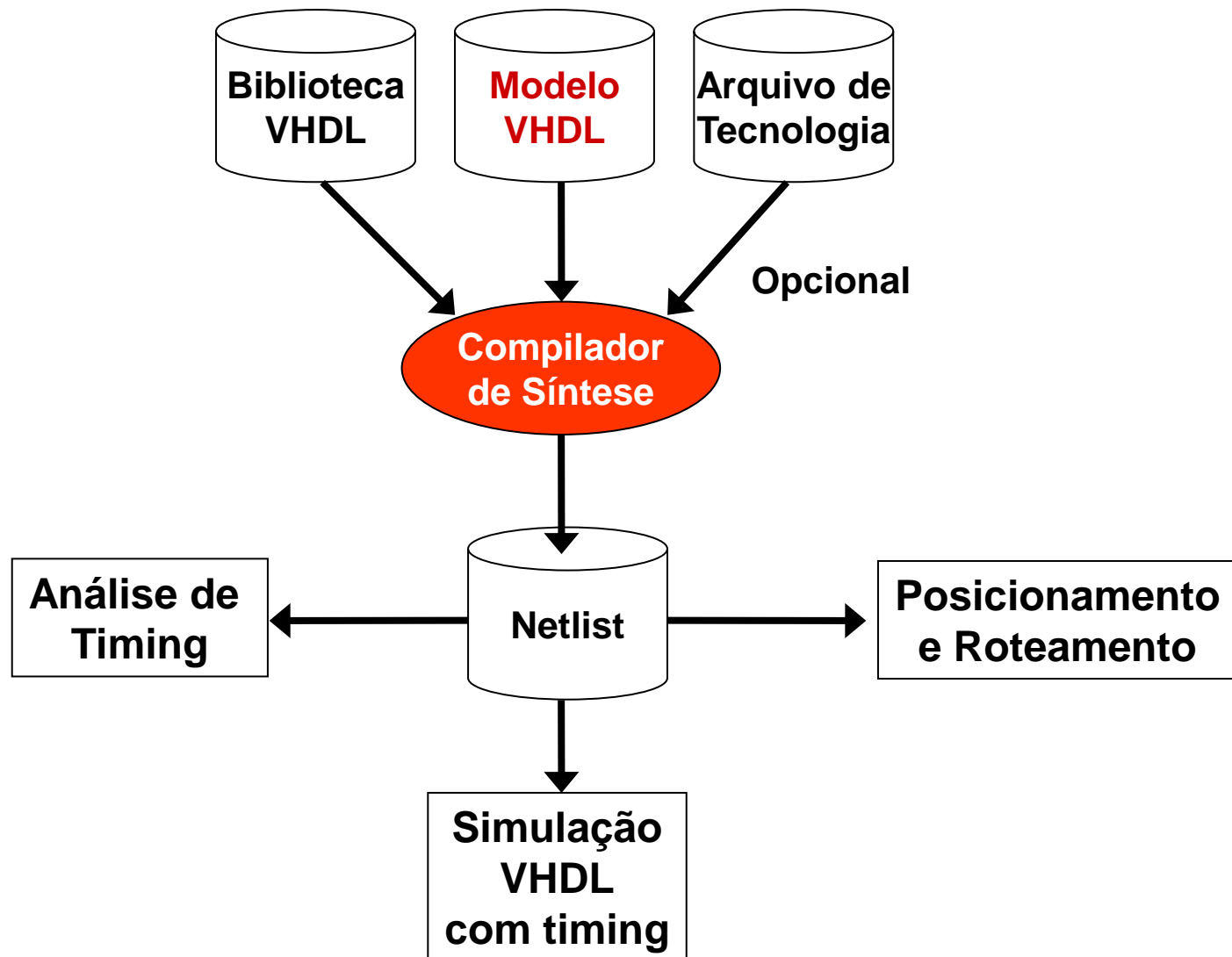
## ⦿ Uso

- ⦿ Documentação
- ⦿ Simulação
- ⦿ Síntese

# Simulação funcional de um modelo VHDL



# Síntese de um modelo VHDL



# Aspectos gerais do VHDL

- Dois conjuntos de construtores: Simulação e Síntese
- Características
  - É baseado em palavras reservadas (**begin**, **end**, ...)
  - É insensível à caixa (**begin** = **Begin** = **BEGIN**)
  - Declarações terminadas por ponto e vírgula “;”
  - Comentários marcados precedidos por duplo hífen “-- **Comment**”

```
entity mux2_1bit is  
port ( sel : in bit;  -- selector  
       a   : in bit;  -- data input  
       b   : in bit;  -- data input  
       s   : out bit); -- data output  
end mux2_1bit;  
-----  
architecture arch_1 of mux2_1bit is  
begin  
    s <= (a and not sel) or (b and sel);  
end arch_1;
```



---

# Noções Básicas de VHDL

- ⦿ Tipos de descrição
- ⦿ Entidade
- ⦿ Arquitetura
- ⦿ Configuração
- ⦿ Componentes

# Tipos de descrição

## ⦿ Dataflow (o que é)

```
s <= (a and not sel) or  
      (b and sel);
```

## ⦿ Comportamental (o que faz)

```
if (sel='0') then  
    s <= a;  
else  
    s <= b;  
end if;
```

## ⦿ Estrutural (o que é)

```
u0 : not_gate port map (sel, seln);  
u1 : and2_gate port map ( a, seln, w1);  
u2 : and2_gate port map ( b, sel, w2);  
u3 : or2_gate port map ( w1, w2, s);
```

# Tipos de descrição

## ⦿ **Mista** (combinação dos outros tipos)

```
w1 <= a and not sel;  
w2 <= b and sel;  
  
u0 : or2_gate port map (w1, w2, s);
```

# Estrutura de um modelo VHDL

- Modelo VHDL = Entidade + Arquitetura
- Entidade (`entity`)
  - Define a interface do circuito – sinais de entrada e saída
  - Os sinais podem ser de 4 modos: `in`, `out`, `buffer` ou `inout`
  - O tipo mais básico de sinal é o tipo `bit`

```
entity mux2_1bit is
port ( sel : in bit;    -- selector
      a   : in bit;    -- data input
      b   : in bit;    -- data input
      s   : out bit);  -- data output
end mux2_1bit;
```

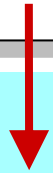
# Estrutura de um modelo VHDL

- ⦿ **Arquitetura (architecture)**
  - ⦿ Define a implementação do circuito
  - ⦿ Requer um nome e deve referenciar a entidade

```
architecture arch_1 of mux2_1bit is
begin
    s <= (a and not sel) or (b and sel);
end arch_1;
```

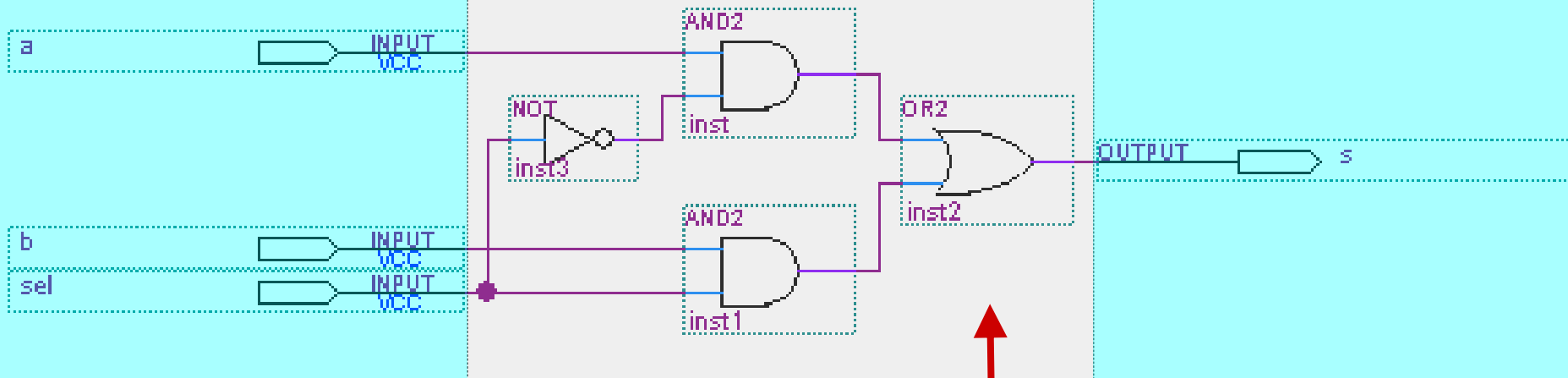
# Estrutura de um modelo VHDL

```
entity mux2_1bit is  
  port ( sel : in  bit;  
         a   : in  bit;  
         b   : in  bit;  
         s   : out bit);  
end mux2_1bit;
```



## ENTIDADE

## ARQUITETURA



```
architecture arch_1 of mux2_1bit is  
begin  
  s <= (a and not sel) or (b and sel);  
end arch_1;
```

# Estrutura de um modelo VHDL

## Arquitetura (architecture)

- Podem existir múltiplas arquiteturas para uma mesma entidade

```
architecture arch_1 of mux2_1bit is
begin
    s <= (a and not sel) or (b and sel);
end arch_1;
```

```
architecture arch_2 of mux2_1bit is
begin
    process(sel, a, b)
    begin
        if (sel='0') then
            s <= a;
        else
            s <= b;
        end if;
    end process;
end arch_2;
```

# Estrutura de um modelo VHDL

## ⦿ Configuração (configuration)

- ⦿ Define qual arquitetura deve ser utilizada
- ⦿ Requer um nome e deve referenciar a entidade

```
configuration config_mux2_1bit of mux2_1bit is  
  for arch_1  
    end for;  
end;
```

- ⦿ Entidade, arquiteturas e configuração podem ser descritos em arquivos separados ou em único arquivo



# Estrutura de um modelo VHDL

## 🕒 Múltiplos arquivos

```
entity mux2_1bit is
port ( sel : in  bit;  -- selector
      a   : in  bit;  -- data input
      b   : in  bit;  -- data input
      s   : out bit); -- data output
end mux2_1bit;
```

```
architecture arch_1 of mux2_1bit is
begin
    s <= (a and not sel) or (b and sel);
end arch_1;
```

```
architecture arch_2 of mux2_1bit is
begin
    process(sel, a, b)
    begin
        if (sel='0') then
            s <= a;
        else
            s <= b;
        end if;
    end process;
end arch_2;
```

```
configuration config_mux2_1bit of mux2_1bit
is
    for arch_1
    end for;
end;
```

# Estrutura de um modelo VHDL

## ⦿ Múltiplos arquivos

```
entity mux2_1bit is
port ( sel : in bit;  -- selector
      a   : in bit;  -- data input
      b   : in bit;  -- data input
      s   : out bit); -- data output
end mux2_1bit;
```

```
architecture arch_1 of mux2_1bit is
begin
    s <= (a and not sel) or (b and sel);
end arch_1;
```

```
architecture arch_2 of mux2_1bit is
begin
    process(sel, a, b)
    begin
        if (sel='0') then
            s <= a;
        else
            s <= b;
        end if;
    end process;
end arch_2;
```

```
configuration config_mux2_1bit of mux2_1bit
is
    for arch_1
    end for;
```

```
end;
```

## ⦿ Arquivo único (+ usado)

```
entity mux2_1bit is
port ( sel : in bit;  -- selector
      a   : in bit;  -- data input
      b   : in bit;  -- data input
      s   : out bit); -- data output
end mux2_1bit;
```

```
-----
architecture arch_1 of mux2_1bit is
begin
    s <= (a and not sel) or (b and sel);
end arch_1;
```

```
-----
architecture arch_2 of mux2_1bit is
begin
    process(sel, a, b)
    begin
        if (sel='0') then
            s <= a;
        else
            s <= b;
        end if;
    end process;
end arch_2;
```

```
-----
configuration config_mux2_1bit of mux2_1bit
is
    for arch_1
    end for;
```

```
end;
```

# Componentes

- ◉ Permitem implementar um modelo formado por uma hierarquia
  - ◉ Cada entidade é um componente e uma ocorrência de uma entidade é uma instância desse componente
  - ◉ As instancias devem ser conectadas a sinais da interface e a sinais internos (mapeamento)

- ◉ Mapeamento

-  Posicional

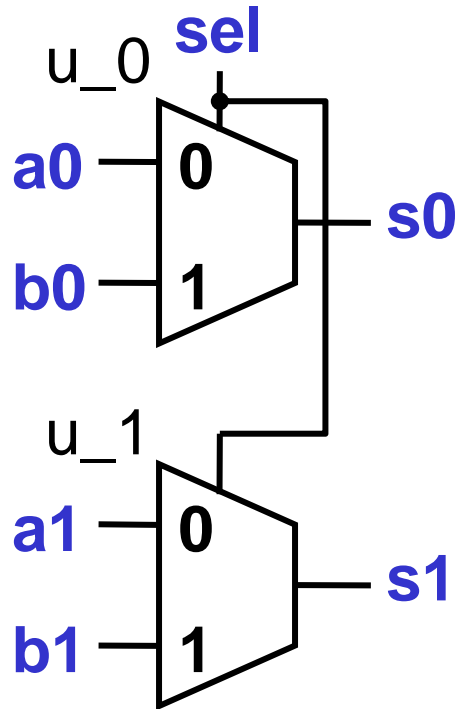
-  Nomeado

```
-- Components instantiation
u_0 : mux2_1bit port map (sel, a(0), b(0), s(0));

u_1 : mux2_1bit port map (a    => a(1),
                           b    => b(1),
                           sel  => sel,
                           s    => s(1));
```

# Componentes

## 🔴 Descrição hierárquica – Exemplo: Multiplexador com 2 canais de 2 bits



```
entity mux2_2bits is
port (
    sel : in bit;           -- selector
    a   : in bit_vector(1 downto 0); -- data input
    b   : in bit_vector(1 downto 0); -- data input
    s   : out bit_vector(1 downto 0)); -- data output
end mux2_2bits;

architecture arch_1 of mux2_2bits is
-- Components declaration
component mux2_1bit is
port ( sel : in bit;      -- selector
        a   : in bit;     -- data input
        b   : in bit;     -- data input
        s   : out bit);  -- data output
end component;

begin
-- Components instantiation
u_0 : mux2_1bit port map (sel, a(0), b(0), s(0));
u_1 : mux2_1bit port map (a  => a(1),
                          b  => b(1),
                          sel => sel,
                          s  => s(1));

end arch 1;
```

# Componentes

## ⦿ Descrição hierárquica – Exemplo: Multiplexador com 2 canais de 2 bits

```
entity mux2_2bits is
port (
    sel : in  bit;                -- selector
    a    : in  bit_vector(1 downto 0); -- data input
    b    : in  bit_vector(1 downto 0); -- data input
    s    : out bit_vector(1 downto 0); -- data output
end mux2_2bits;
```

⦿ Este exemplo ilustra o uso do tipo composto `bit_vector`

# Componentes

## ⦿ Descrição hierárquica – Exemplo: Multiplexador com 2 canais de 2 bits

```
-- Components declaration
component mux2_1bit is
port ( sel : in  bit;  -- selector
      a   : in  bit;  -- data input
      b   : in  bit;  -- data input
      s   : out bit); -- data output
end component;
```

# Componentes

## ⦿ Descrição hierárquica – Exemplo: Multiplexador com 2 canais de 2 bits

```
-- Components instantiation
u_0 : mux2_1bit port map (sel, a(0), b(0), s(0));

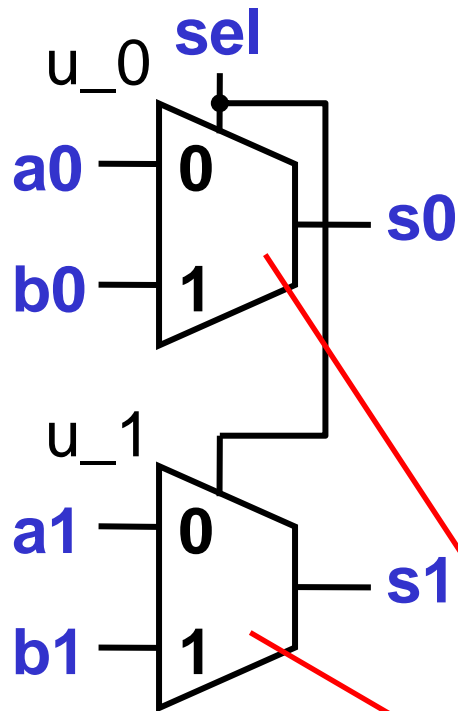
u_1 : mux2_1bit port map (a    => a(1),
                           b    => b(1),
                           sel => sel,
                           s    => s(1));
```

⦿ Instância `u_0` utiliza mapeamento posicional

⦿ Instância `u_1` utiliza mapeamento nomeado

# Componentes

## 🔴 Descrição hierárquica – Exemplo: Multiplexador com 2 canais de 2 bits

[illegible]



# Pacotes

---

- Um pacote é uma forma conveniente para armazenar e usar informações ao longo de um modelo completo (tipicamente com múltiplas entidades)
- Um pacote é um arquivo VHDL com declarações e definições comuns às entidades do modelo
- Um pacote consiste de
  - Declaração do pacote
  - Declaração de tipos
  - Declaração de subprogramas
  - Corpo do pacote (opcional)
  - Definição de subprogramas (se declarados)
- VHDL possui dois pacotes implícitos (*built in*)
  - Standard
  - Textio

# Pacotes

## ⦿ Exemplo de pacote

```
package package_example is
  -- Constants Declaration (optional)

  -- Type Declaration (optional)

  -- Subtype Declaration (optional)

  -- Signal Declaration (optional)

  -- Component Declaration (optional)
  component mux2_1bit is
    port ( sel : in bit; -- selector
          a   : in bit; -- data input
          b   : in bit; -- data input
          s   : out bit); -- data output
  end component;
end package_example;
```



# Bibliotecas

- ⦿ Uma biblioteca contém um ou mais pacotes
- ⦿ Bibliotecas de recursos
  - ⦿ Pacotes padrão
  - ⦿ Pacotes IEEE
  - ⦿ Pacotes específicos da ferramenta EDA (ex. pacotes Altera)
  - ⦿ Qualquer biblioteca de unidades de projeto que são referenciadas em um projeto
- ⦿ Exemplo de uso de biblioteca
  - ⦿ Deve-se utilizar a cláusula `library` para referenciar a biblioteca
  - ⦿ Deve-se utilizar a cláusula `use` para referenciar um pacote da biblioteca
  - ⦿ O sufixo `.all` referencia todos os objetos do pacote

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

# Bibliotecas

## ○ Biblioteca de trabalho (WORK)

- Biblioteca na qual a unidade de projeto corrente está sendo compilada
- Não precisa ser referenciada com a cláusula `library`

```
library work;  
use work.package_example.all;
```

# Biblioteca STD

---

- ⦿ **Contém os seguintes pacotes**

- ⦿ `Standard`

- Suporta os tipos: `bit`, `bit_vector`, `boolean`, `integer`, `real`, `time` e `string`

- Funções de operadores para os tipos suportados

- ⦿ `Textio`

- Suporte a operações com arquivos

- ⦿ **É uma biblioteca implícita e não precisa ser referenciada**

## Tipos de dados escalares do pacote Standard

Classe	Tipo	Valores	Exemplo
Enumerado	bit	Um e zero	'1', '0'
	boolean	Verdadeiro e falso	TRUE, FALSE
	character	Caracteres ASCII	a, b, c,..., A, B, C
Numérico	integer	$-2^{31} \leq x \leq 2^{31}-1$	19, 8#57#, 16#A2#, 2#11#
	real	$-3.65 \times 10^{47} \leq x \leq +3.65 \times 10^{47}$	2.63, 1.75E+2, 16#8.C#

⦿ No tipo `integer` é conveniente limitar a faixa de representação para economizar recursos. Ex:

```
signal int_8bit : integer range 0 to 255;
```

# Biblioteca STD

## Tipos de dados compostos do pacote Standard

Classe	Tipo	Valores	Exemplo
Vetor	<code>bit_vector</code>	Tipo <code>bit</code>	"1110", B"11_10", O"32", X"A2"
	<code>string</code>	Tipo <code>character</code>	"teste", "oi mundo"

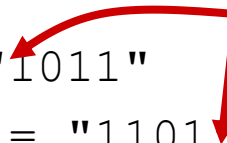
## Na declaração de um tipo composto, seus limites são definidos por `to` e `downto`

## Exemplos

```
constant x: bit_vector (0 to 3) := "1011"
```

```
constant x: bit_vector(3 downto 0) := "1101"
```

**bit 0**





- ⦿ **Contém os seguintes pacotes**

- ⦿ `std_logic_1164`

- Descreve os tipos `std_logic` e `std_logic_vector` e funções relacionadas

- ~~⦿ `std_logic_arith`~~

- ~~Descreve os tipos `signed` e `unsigned`, funções de conversão, aritméticas e relacionais~~

- ⦿ `numeric_std`

- Faz referência aos pacotes `numeric_bit` e `numeric_std` os quais descrevem funções que permitem à ferramenta de síntese usar os tipos `bit_vector` e `std_logic_vector` para sintetizar os tipos `signed` e `unsigned`

- ⦿ **Tipos de dados do pacote** `std_logic_1164`

- ⦿ `std_logic`

- 9 valores lógicos ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')

- Possui funções de resolução associadas para a modelagem de múltiplos drivers acionando o mesmo sinal (ex. barramento)**

- ⦿ `std_logic_vector`

- Vetor de elementos `std_logic`

- ⦿ `std_ulogic`

- Não possui funções de resolução

- ⦿ `std_ulogic_vector`

- Vetor de elementos `std_ulogic`

- ⦿ **Funções disponíveis no pacote** `std_logic_1164`
  - ⦿ Operadores lógicos  
`and`, `or`, `nand`, `nor`, `xor`, `xnor`, `not`
  - ⦿ Funções de conversão entre `bit`, `std_logic` e `std_ulogic`  
`to_bit`, `to_bitvector`, `to_stdlogicvector`, e outras
  - ⦿ Funções de detecção de borda de subida  
`rising_edge` e `falling_edge`

## ⦿ **Tipos de dados do pacote** `numeric_std`

### ⦿ `unsigned`

Número inteiro sem sinal

Composto por elementos do tipo `std_logic` (ou `bit`)

Os elementos mais significativos são posicionados à esquerda

## ⦿ **Tipos de dados do pacote** `std_logic_signed`

### ⦿ `signed`

Número inteiro com sinal

Representação em complemento de dois

O bit mais à esquerda é o bit de sinal

⦿ **Funções disponíveis no pacote** `numeric_std`

Tipo do operador	Nome do operador (função)
Lógico	<code>and</code> , <code>or</code> , <code>nand</code> , <code>nor</code> , <code>xor</code> , <code>xnor</code>
Relacional	<code>=</code> , <code>/=</code> , <code>&lt;</code> , <code>&lt;=</code> , <code>&gt;</code> , <code>&gt;=</code>
Adição e subtração	<code>+</code> , <code>-</code>
Concatenação	<code>&amp;</code>
Multiplicação e divisão	<code>*</code> , <code>/</code> , <code>mod</code> , <code>rem</code>
Sinalização	<code>-</code>
Miscelânea	<code>**</code> , <code>abs</code> e <code>not</code>
Deslocamento	<code>shift_right</code> , <code>shift_left</code> , <code>srl</code> , <code>sll</code>
Rotação	<code>rotate_right</code> , <code>rotate_left</code> , <code>ror</code> , <code>rol</code>
Conversão	<code>to_integer</code> , <code>to_unsigned</code> , <code>to_signed</code>
...	...

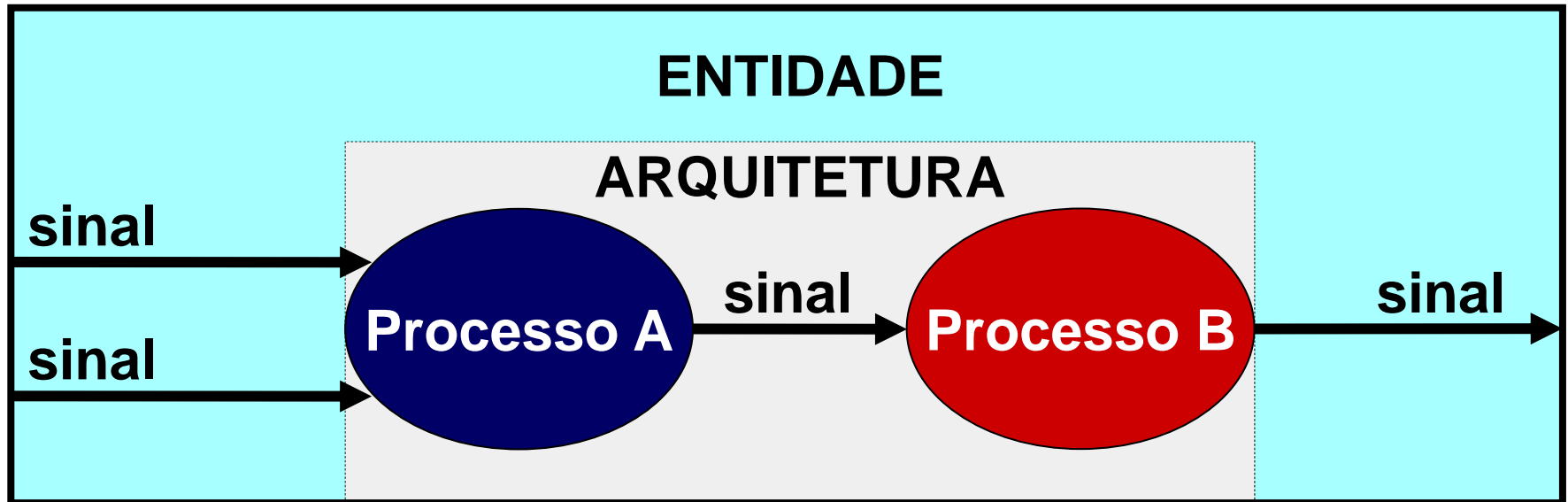
---

# Sinais e Variáveis

- ⦿ Sinais
- ⦿ Processos implícitos
- ⦿ Processos explícitos
- ⦿ Variáveis

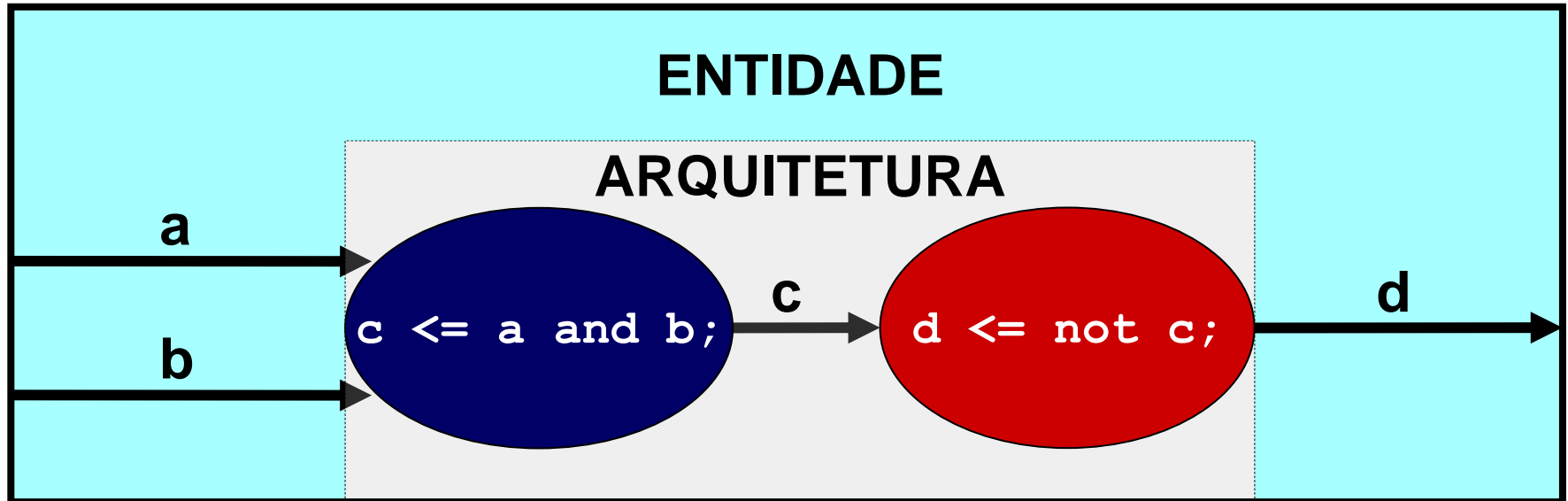
# Sinais

- Representam interconexões físicas (fios) que realizam a comunicação de informações entre os processos
- Declarados na entidade, na arquitetura ou em um pacote



# Sinais

- ⦿ A atribuição é feita usando o operador **<=**
- ⦿ Uma atribuição a um sinal possui um processo implícito associado





# Sinais

## ⦿ Atribuindo constantes a sinais

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

```
temp          <= "10101100";
```

```
temp          <= X"AC";
```

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

```
temp          <= (others => '0');
```

```
temp          <= "1010" & (others => '0');
```

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

# Tipos de atribuição a sinais concorrentes (ex: mux2)

## ○ Atribuição simples

```
s <= (a and not sel) or (b and sel);
```


## ○ Atribuição condicional

```
s <= a when sel='0' else  
      b;
```

## ○ Atribuição selecionada

```
with sel select  
      s <= a when '0'  
      b when others;
```

No caso de `sel` ser do tipo `std_logic`, `when others` garante um valor definido para `s` (ou seja `b`) quando `sel` possuir qualquer outro valor diferente de '0'



# Processos explícitos

- ⦿ **Necessários na implementação de funções compostas por atribuição sequenciais (ex. if-then-else)**

```
<process_label>:                -- (optional)
process (<sensitive_list>)
    -- Constant declarations
    -- Type declarations
    -- Variable declarations
begin
    -- Signal Assignment Statement (optional)
    -- Variable Assignment Statement (optional)
    -- Procedure Call Statement (optional)
    -- If Statement (optional)
    -- Case Statement (optional)
    -- Loop Statement (optional)
end process <process_label>;
```

# Processos explícitos

- Um processo é executado infinitamente se não for quebrado por uma declaração **wait** ou por uma **lista de sensibilidade**
- Uma lista de sensibilidade infere uma declaração **wait** no final do processo
- Um processo pode ter múltiplos **wait**, mas não pode ter um **wait** e uma lista de sensibilidade
- As atribuições a sinais em um processo são efetivadas ao término da sua execução

```
architecture behavior of mux_2x1 is
begin
    process (a,b,sel)
    begin
        if (sel='0') then
            s <= a;
        else
            s <= b;
        end if;
    end process;
end behavior;
```

```
architecture behavior of mux_2x1 is
begin
    process
    begin
        if(sel='0') then
            s <= a;
        else
            s <= b;
        end if;
        wait(a,b,sel);
    end process;
endbehavior;
```

# Variáveis

- ⦿ Declaradas dentro de processos e constituem-se em objetos de armazenamento temporário
- ⦿ A atribuição é feita usando o operador **:=**
- ⦿ São atualizadas no momento da atribuição

```
architecture behavior of mux_2x1 is
begin
  process (a,b,sel)
    variable temp : bit;
  begin
    if(sel='0') then
      temp := a;
    else
      temp := b;
    end if;

    s <= temp;
  end process;
End behavior;
```

# Variáveis

## ⦿ Atribuindo constantes a variáveis

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

```
temp          := "10101100";
```

```
temp          := X"AC";
```

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

```
temp          := (others => '0');
```

```
temp          := "1010" & (others => '0');
```

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

# Sinais x Variáveis

	Sinal	Variável
Atribuição	$\leq$	$:=$
Utilidade	Representa um fio no circuito	Representa um local de armazenamento temp.
Escopo	Global (comunicação entre processos)	Local (dentro do processo)
Comportamento	Atualizado no final do processo	Atualizada imediatamente

# Estilo de codificação

---

- ⦿ **Define como as palavras reservadas e os identificadores devem ser escritos**
- ⦿ **Estilo adotado no LEDS-UNIVALI**
  - ⦿ Cada sinal e variável deve ser declarado em uma linha própria
  - ⦿ Uso intensivo de comentários para identificar sinais e variáveis
  - ⦿ Tabulação = 2 ou de 4 pontos
  - ⦿ Um processo para cada sinal
  - ⦿ Palavras reservadas em caixa-baixa
  - ⦿ Identificadores do modelo em caixa-alta, exceto prefixos e sufixos
  - ⦿ Sinais ativos em nível baixo devem ser identificados pelo sufixo `_n`



# Estilo de codificação

## Estilo adotado no LEDS-UNIVALI (continuação)

### Os seguintes prefixos devem ser utilizados

**i\_** : sinal de entrada (modo `in`)

**o\_** : sinal de saída (modos `out` e `buffer`)

**b\_** : sinal bidirecional (modo `inout`)

**w\_** : sinal interno que implementa um fio

**r\_** : sinal interno que implementa um registrador

**v\_** : variável de processo

**p\_** : parâmetro genérico

**u\_** : identificador de instância de componente ou de processo

**t\_** : identificador de tipo de dado

**c\_** : identificador de constante

**s\_** : identificador de estado de uma máquina de estados

**f\_** : identificador de laço iterativo de geração

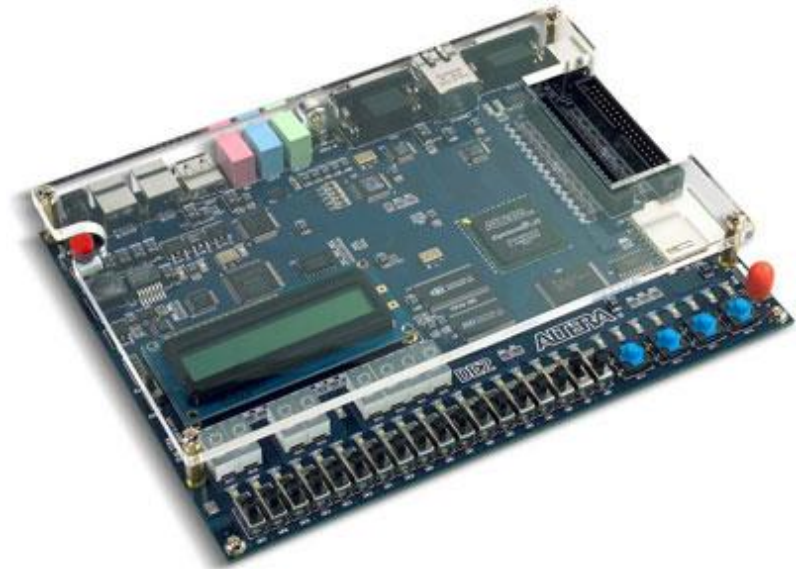
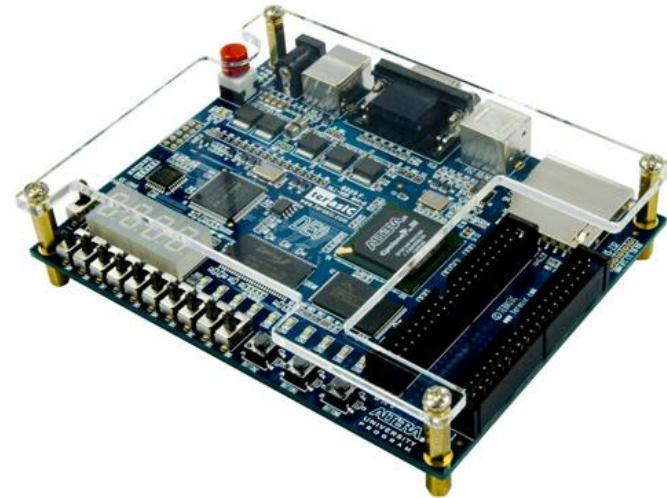
# Ferramentas e kits de desenvolvimento



**ModelSim**®

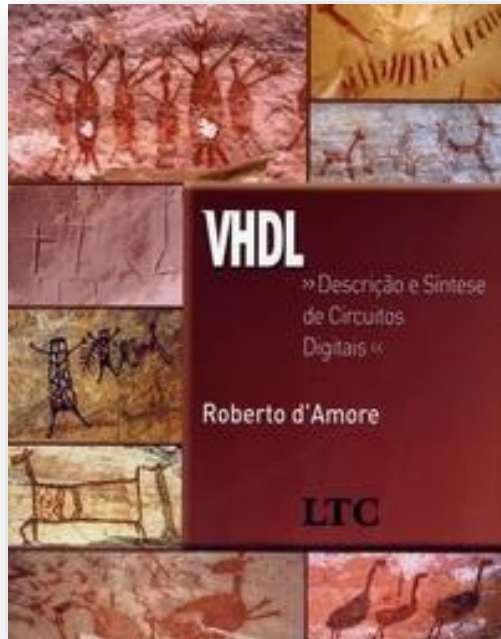
ModelSim Altera Edition Software

**Quartus**® Prime  
Design Suite

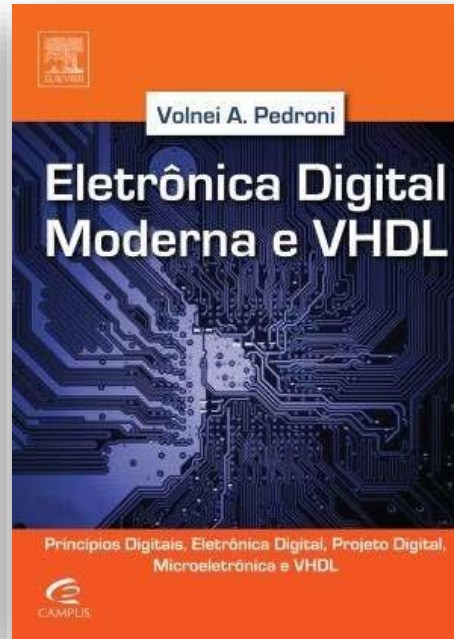


# Considerações Finais

## Para aprender mais sobre VHDL



Roberto d'Amore



Volnei Pedroni

## VHDL Archive



**MIN-Fakultät**  
**Fachbereich Informatik**  
**TAMS**

**Universität Hamburg**  
DER FORSCHUNG | DER LEHRE | DER BILDUNG

UHH > MIN > Informatik > TAMS > Research > System- and VLSI-Design > VHDL

Servers / Internet VHDL  
FAQ  
Documentation  
Papers  
Drafts  
Models and Packages  
Tools

### The Hamburg VHDL Archive

Welcome to the Hamburg VHDL archive! We intend to provide a collection of free, i.e. public-domain or shareware, VHDL documentation, models, and tools.

This service is provided and maintained by **group TAMS, Faculty of Informatics, University of Hamburg.**

### Feedback

We are always looking for additional documents, models, tools and Internet links to improve this server. All contributions, hints, and corrections are welcome. Please contact **Andreas Mäder** or **Norman Hendrich**.

Please consider to **contribute some of your own models and tools to the VHDL-community** in turn for any information downloaded from one of the free VHDL servers.

Legal Notice

Last modified: 04.06.2007

# Acompanhe o LEDS no Instagram

## @leds.lab

Instagram

🔍 Pesquisar



leds.lab

Seguindo



138 publicações

240 seguidores

48 seguindo

LEDS

Laboratory of Embedded and Distributed Systems • SC - Brazil • Check out the latest

