

UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO  
CURSO: CIÊNCIA DA COMPUTAÇÃO

# Microprocessadores & Microcontroladores



Aula 3 – Introdução ao VHDL

Prof. Leonardo Augusto Casillo

# Introdução

O que significa VHDL?

**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit  
**H**ardware  
**D**escription  
**L**anguage

*Linguagem de Descrição de Hardware com ênfase em  
Circuitos Integrados de altíssima velocidade.*

# O que significa Linguagem de Descrição de Hardware (HDL)?

Uma linguagem de descrição de hardware descreve o que um sistema faz e como;

Um sistema descrito em linguagem de hardware pode ser implementado em um dispositivo programável (ex: **FPGA** – **F**ield **P**rogrammable **G**ate **A**rray), permitindo o uso em campo do sistema;

Existem dezenas de HDLs:

- AHDL, VERILOG, Handel-C, SDL, ISP, ABEL ...

# Breve Histórico

- final de 1960: primeiras Linguagem de Hardware;
- 1973: projeto **CONLAN** (CONsensus LANguage);
- 1983: relatório final do **CONLAN** e a Linguagem **ADA**;
- 1983: **DoD** inicia programa VHSIC (participação da **IBM**, **Intermetrics** e **Texas Instruments**;
- 1986: a **Intermetrics** desenvolve compilador e simulador, criado um grupo de padronização da IEEE para VHDL;
- 1988: primeiros *softwares* são comercializados;
- 1991: começou-se um novo processo de padronização;
- 1992: modificações propostas foram avaliadas e votadas;
- 1993: um novo padrão é publicado, chamado VHDL-93;
- 1997: publicado o manual de referência da linguagem.

# Vantagens e Desvantagens do VHDL

## • Vantagens

- Projeto independente da tecnologia;
- Facilidade na atualização dos projetos;
- Reduz tempo de projeto e custo;
- Elimina erros de baixo nível;
- Simplifica a documentação

## • Desvantagens

- Hardware gerado é menos otimizado;
- Falta de pessoal treinado para lidar com a linguagem;
- Simulações geralmente mais lentas que outras implementações.

# Características do VHDL

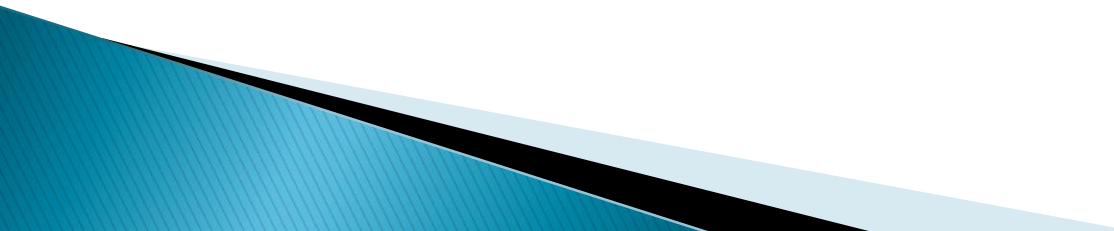
- Permite, através de simulação, verificar o comportamento do sistema digital;
- Permite descrever hardware em diversos níveis de abstração, por exemplo:
  - Algorítmico ou comportamental.
  - Transferência entre registradores (RTL).
- Favorece projeto “top-down”.

Hoje utilizada para **SIMULAÇÃO** e **SÍNTESE**

# Características do VHDL

- ▶ VHDL é análogo a uma linguagem de programação
- ▶ VHDL provê mecanismos para modelar a concorrência e sincronização que ocorrem a nível físico no hardware
- ▶ Projetar um sistema em VHDL é geralmente muito mais difícil que escrever um programa para fazer a mesma coisa utilizando uma linguagem de programação como C
- ▶ O código VHDL é executado em um simulador
  - Não há um “executável”

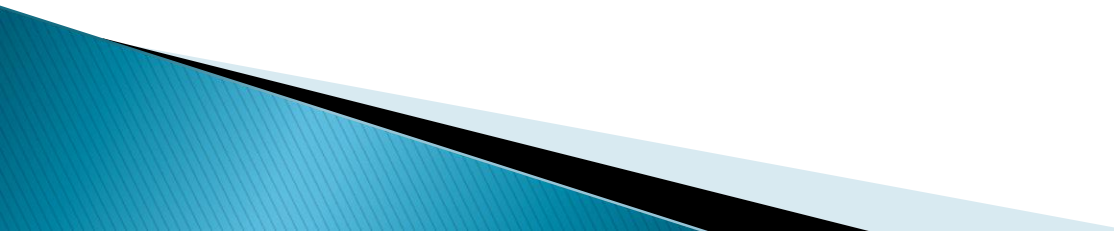
# Ciclo de Projeto:

- **Especificação:** determinar requisitos e funcionalidade do projeto.
  - **Codificação:** descrever em VHDL todo o projeto, segundo padrões de sintaxe.
  - **Simulação do Código-Fonte:** simular o código em ferramenta confiável a fim de verificar preliminarmente cumprimento da especificação;
- 



# Ciclo de Projeto:

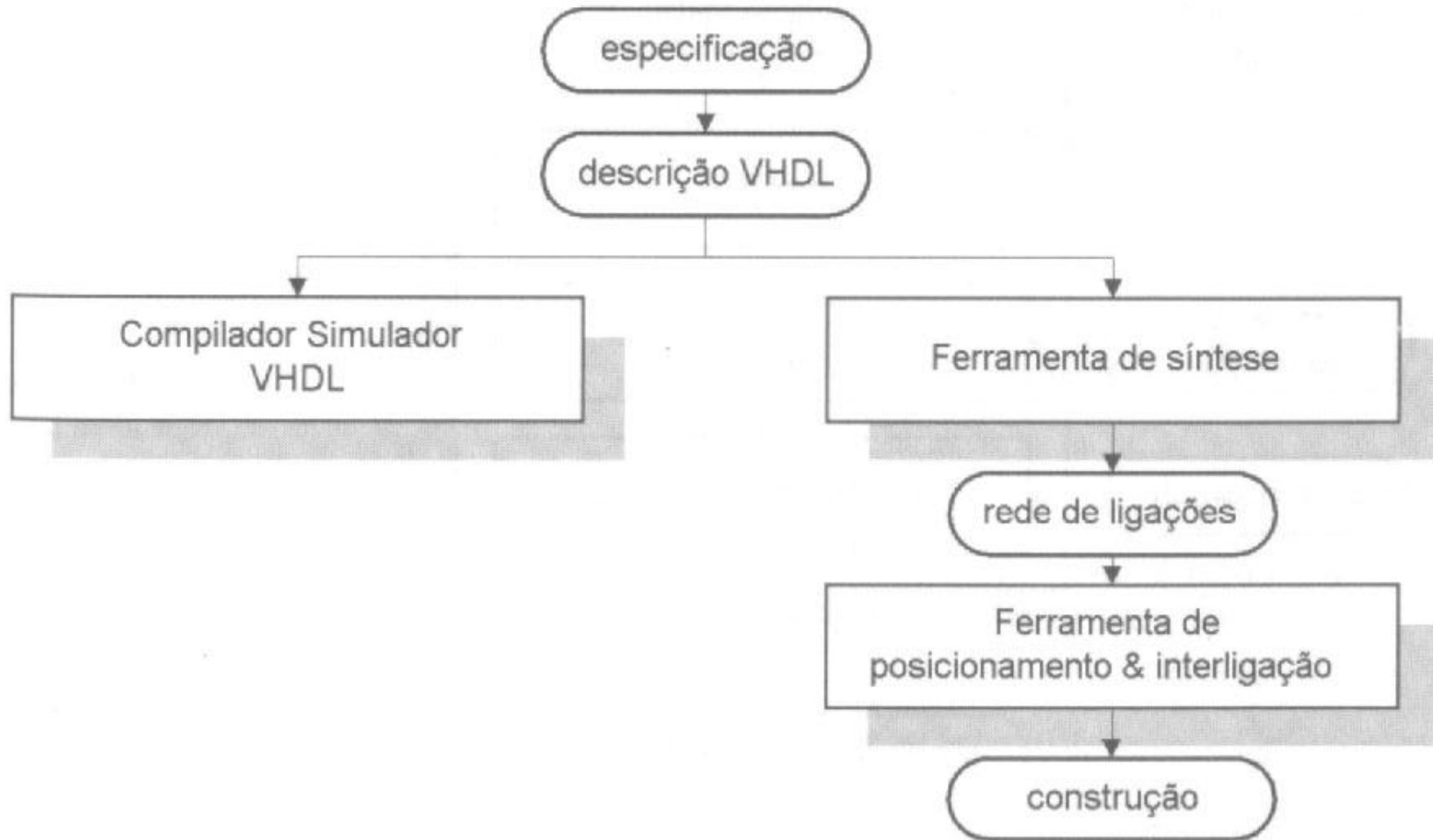
## – Síntese, otimização e *Fitting*:

- ❑ **Síntese:** compilação de um código VHDL para uma descrição abstrata.
  - ❑ **Otimização:** seleção da melhor solução de implementação para uma dada tecnologia.
  - ❑ **Fitting:** lógica sintetizada e otimizada mapeada nos recursos oferecidos pela tecnologia.
- 

# Ciclo de Projeto:

- **Simulação do modelo:** resultados mais apurados de comportamento e *timing*.
- **Geração:** configuração das lógicas programáveis ou de fabricação de ASICs.

# Etapas de Projeto:



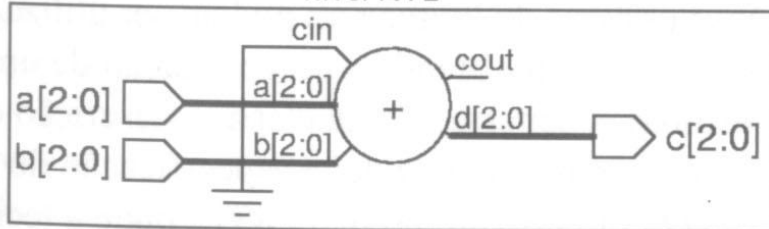
# Etapas de Projeto:

descrição VHDL

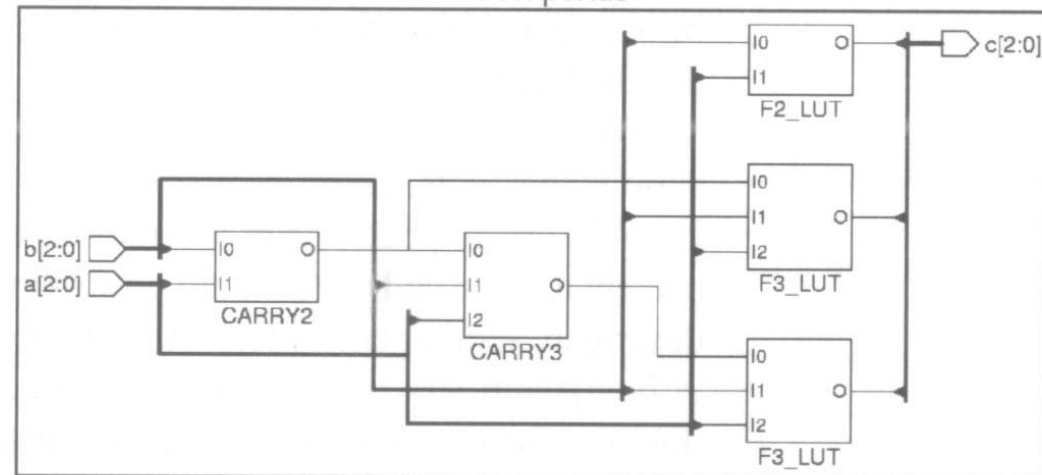
```
ENTITY soma IS
  PORT (a, b : IN  INTEGER RANGE 7 DOWNT0 0;
        c   : OUT INTEGER RANGE 7 DOWNT0 0);
END soma;

ARCHITECTURE teste OF soma IS
BEGIN
  c <= a+b;
END teste;
```

nível RTL



nível portas



otimização velocidade / área

# Componentes de um projeto

<b>PACKAGE</b>
<b>ENTITY</b>
<b>ARCHITECTURE</b>
<b>CONFIGURATION</b>

- Package (Pacote): **constantes, bibliotecas;**
- Entity (Entidade): **pinos de entrada e saída;**
- Architecture (Arquitetura): **implementações do projeto;**
- Configuration (Configuração): **define as arquiteturas que serão utilizadas.**

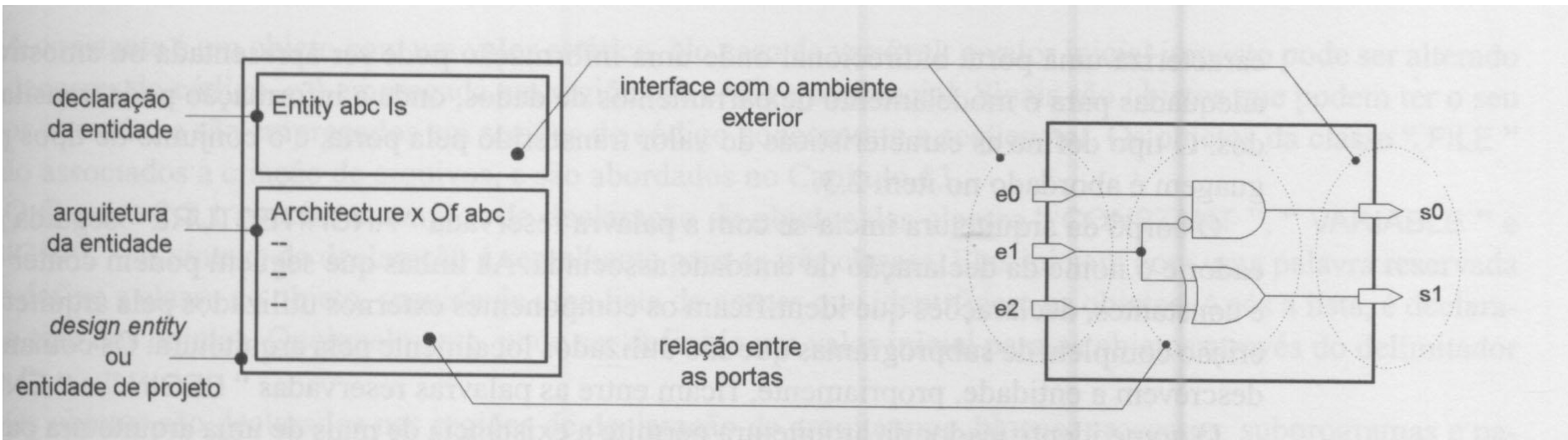
<b>LIBRARY IEEE;</b> <b>USE IEEE.STD_LOGIC_1164.all;</b> <b>USE IEEE.STD_LOGIC_UNSIGNED.all;</b>	PACKAGE (BIBLIOTECAS)
<b>ENTITY</b> exemplo <b>IS</b> <b>PORT</b> ( <descrição dos pinos de I/O> ); <b>END</b> exemplo;	ENTITY (PINOS DE I/O)
<b>ARCHITECTURE</b> teste <b>OF</b> exemplo <b>IS</b> <b>BEGIN</b> ... <b>END</b> teste;	ARCHITECTURE (ARQUITETURA)

# Entity (Entidade)

**Abstração que descreve um sistema, uma placa, um chip, uma função ou uma porta lógica.**

```
Entity <nome_da_entidade> is  
port (  
    entrada_1      : in <tipo>;  
    entrada_2      : in <tipo>;  
    saída_1 : out <tipo>;  
    ...  
);  
end <nome_da_entidade>;
```

# Entity (Entidade)





# Entity (Entidade)

**GENERIC:** passagem de informações estáticas

**PORT:** correspondem ao pinos de entrada e saída.

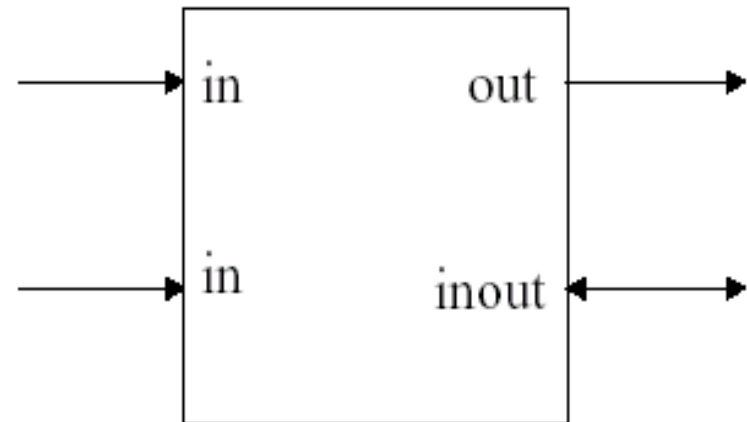
## Modos de operação:

❑ **IN:** porta de entrada;

❑ **OUT:** porta de saída (não podem ser usados como entradas, nem seus valores utilizados na lógica interna);

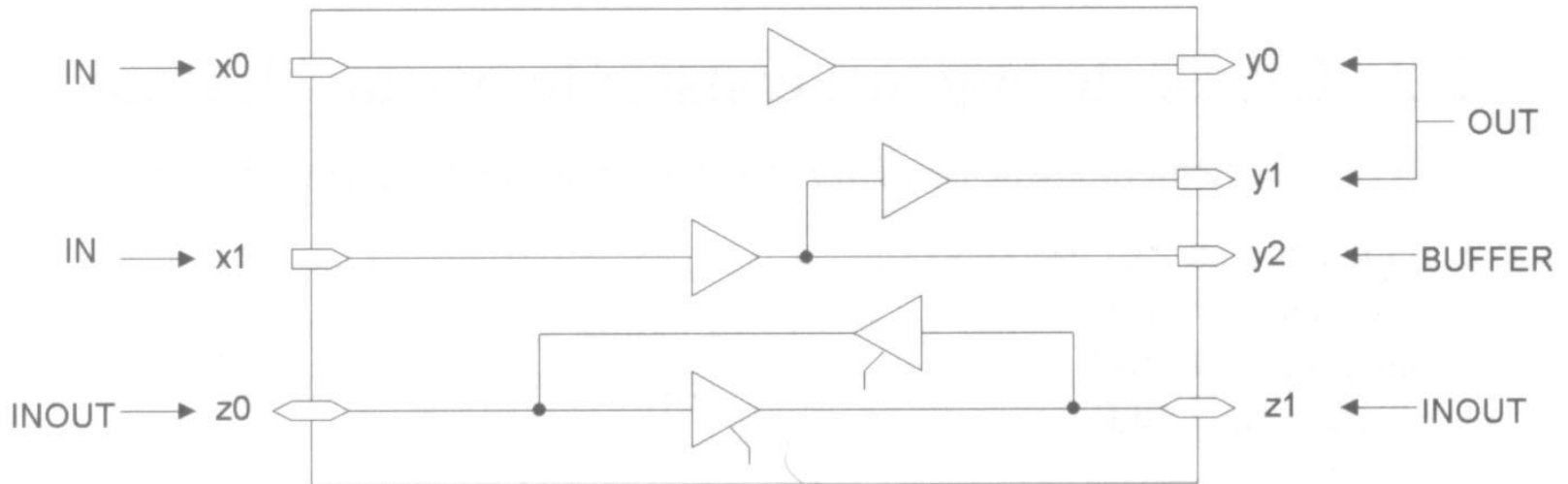
❑ **INOUT:** porta de entrada e saída;

❑ **BUFFER:** saída com possibilidade de realimentação.



# Entity (Entidade)

```
ENTITY entidade_abc IS  
  
  GENERIC (n          : INTEGER := 5);  
  
  PORT (x0, x1       : IN        tipo_a;    -- entradas  
        y0, y1       : OUT        tipo_b;    -- saidas  
        y2           : BUFFER     tipo_c;    -- saida  
        z0, z1       : INOUT     tipo_d);    -- entrada / saida  
  
END entidade_abc;
```



# Entity (Entidade)

## Tipos mais utilizados:

bit	Assume valores '0' ou '1'. <b>x: in bit;</b>
bit_vector	Vetor de bits. <b>x: in bit_vector(7 downto 0);</b> <b>x: in bit_vector(0 to 7);</b>
std_logic	<b>x: in std_logic;</b>
std_logic_vector	<b>x: in std_logic_vector(7 downto 0);</b> <b>x: in std_logic_vector(0 to 7);</b>
boolean	Assume valores TRUE ou FALSE

# Entity (Entidade)

## STD\_LOGIC:

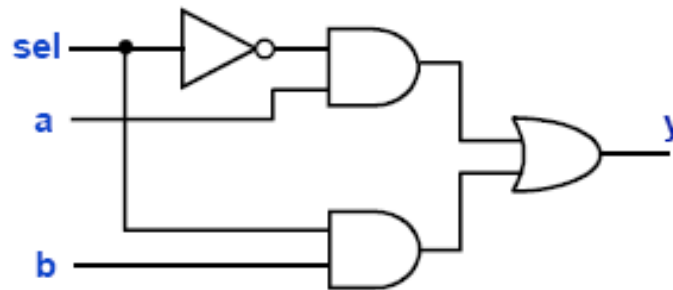
- Definida pela biblioteca IEEE;
- Pode assumir nove valores:

<b>'U':</b> não inicializada	<b>'Z':</b> alta impedância
<b>'X':</b> desconhecida	<b>'W':</b> desconhecida fraca
<b>'0':</b> valor '0'	<b>'L':</b> '0' fraca (Low)
<b>'1':</b> valor '1'	<b>'H':</b> '1' fraca (High)
<b>'-':</b> <i>Don't care.</i>	

# Entity (Entidade)

## Circuito exemplo:

```
ENTITY exemplo1 IS  
  PORT ( sel : IN BIT;  
         a : IN BIT;  
         b : IN BIT;  
         y : OUT BIT);  
END exemplo1;
```



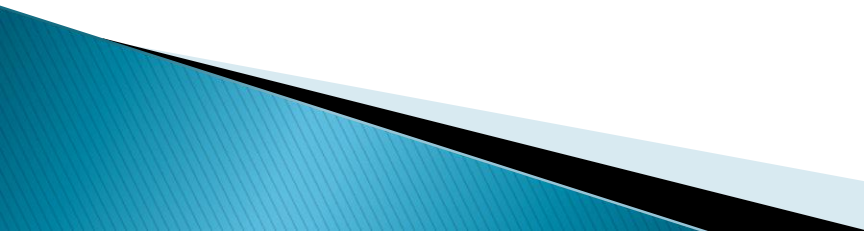
- ▶ A extensão de um arquivo em VHDL é ".vhd". O nome do arquivo DEVE ser o mesmo nome da entidade. No caso acima, o arquivo deve ser salvo com o nome exemplo1.vhd.

# Architecture (Arquitetura)

- ▶ **Especificação do funcionamento do circuito**
- ▶ Formada por:
  - Declarações: sinais, constantes, componentes, subprogramas
  - Comandos: blocos, atribuições a sinais, chamadas a subprogramas, instanciação de componentes, processos
- ▶ Uma entidade pode ter várias arquiteturas: VHDL provê meios de especificar qual arquitetura se deseja utilizar

# Architecture (Arquitetura)

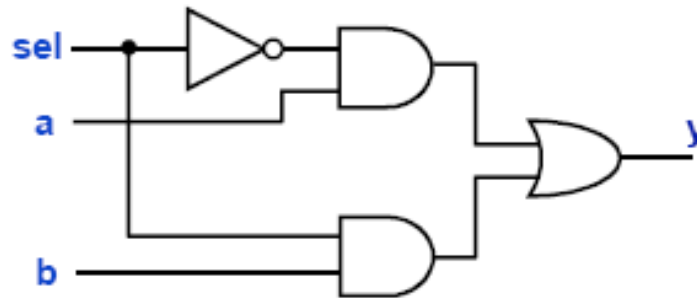
```
ARCHITECTURE nome_identificador OF entidade_abc IS
  --
  -- regiao de declaracoes:
  --   declaracoes de sinais e constantes
  --   declaracoes de componentes referenciados
  --   declaracao e corpo de sub-programas
  --   definicao de novos tipos de dados locais
  --
BEGIN
  --
  -- comandos concorrentes
  --
END;
```



# Architecture (Arquitetura)

## Arquitetura do circuito exemplo:

```
ARCHITECTURE comportamento OF exemplo1 IS  
BEGIN  
    y <= (a AND (NOT(sel))) OR (b AND sel);  
END comportamento;
```





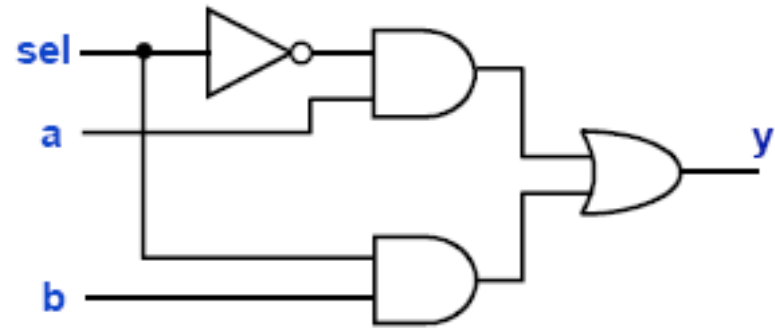
# Architecture (Arquitetura)

## Circuito exemplo completo em VHDL:

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

```
ENTITY exemplo1 IS  
    PORT ( sel, a, b : IN BIT;  
          y : OUT BIT);  
END exemplo1;
```

```
ARCHITECTURE comportamento OF exemplo1 IS  
BEGIN  
    y <= (a AND (NOT(sel))) OR (b AND sel);  
END comportamento;
```

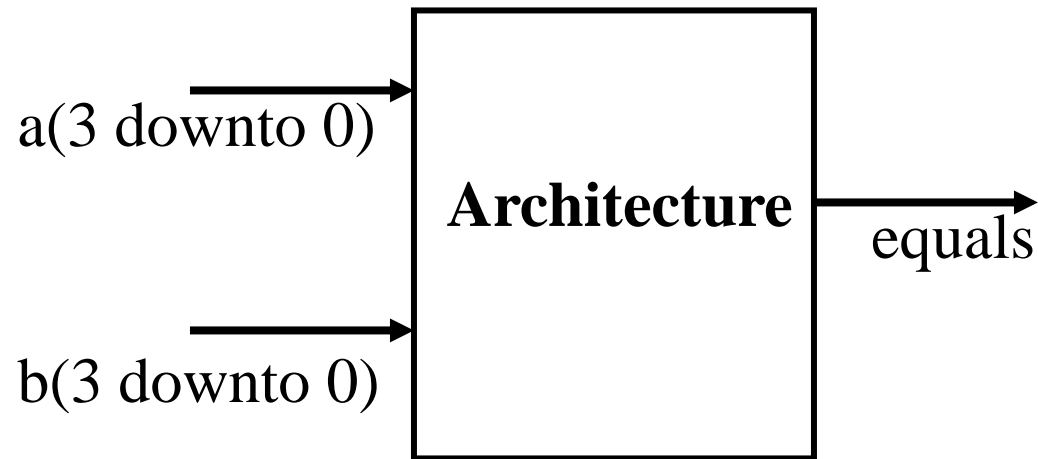


# Architecture (Arquitetura)

- **Descrição comportamental:**
  - descreve o que o sistema deve fazer de forma abstrata
- **Descrição por fluxo de dados (*data-flow*):**
  - descreve o que o sistema deve fazer utilizando expressões lógicas
- **Descrição estrutural:**
  - descreve como é o hardware em termos de interconexão de componentes

# Architecture (Arquitetura)

**Exemplo simples: Comparador de 4 bits**



**se  $a = b$  então**

**Equals = 1**

**senão**

**Equals = 0**

# Descrição por Data-Flow

-- comparador de 4 bits

entity comp4 is

port ( a, b: in bit\_vector (3 downto 0);  
equals: out bit );

end comp4;

architecture fluxo of comp4 is

begin

equals <= '1' when (a=b) else '0';

end fluxo;



# Descrição Comportamental

-- comparador de 4 bits

entity comp4 is

port ( a, b: in bit\_vector (3 downto 0);  
equals: out bit );

end comp4;

architecture comport of comp4 is

begin

comp: process (a,b) -- lista de sensibilidade

begin

if a = b then

equals <= '1' ;

else

equals <= '0' ;

end if;

end process comp;

end comport;

# Descrição Estrutural

-- comparador de 4 bits

entity comp4 is

```
    port (    a, b: in bit_vector (3 downto 0);
            equals: out bit    );
```

end comp4;

architecture estrut of comp4 is

signal x: bit\_vector (3 downto 0);

begin

```
    U0: xnor port map (a(0), b(0), x(0));
```

```
    U1: xnor port map (a(1), b(1), x(1));
```

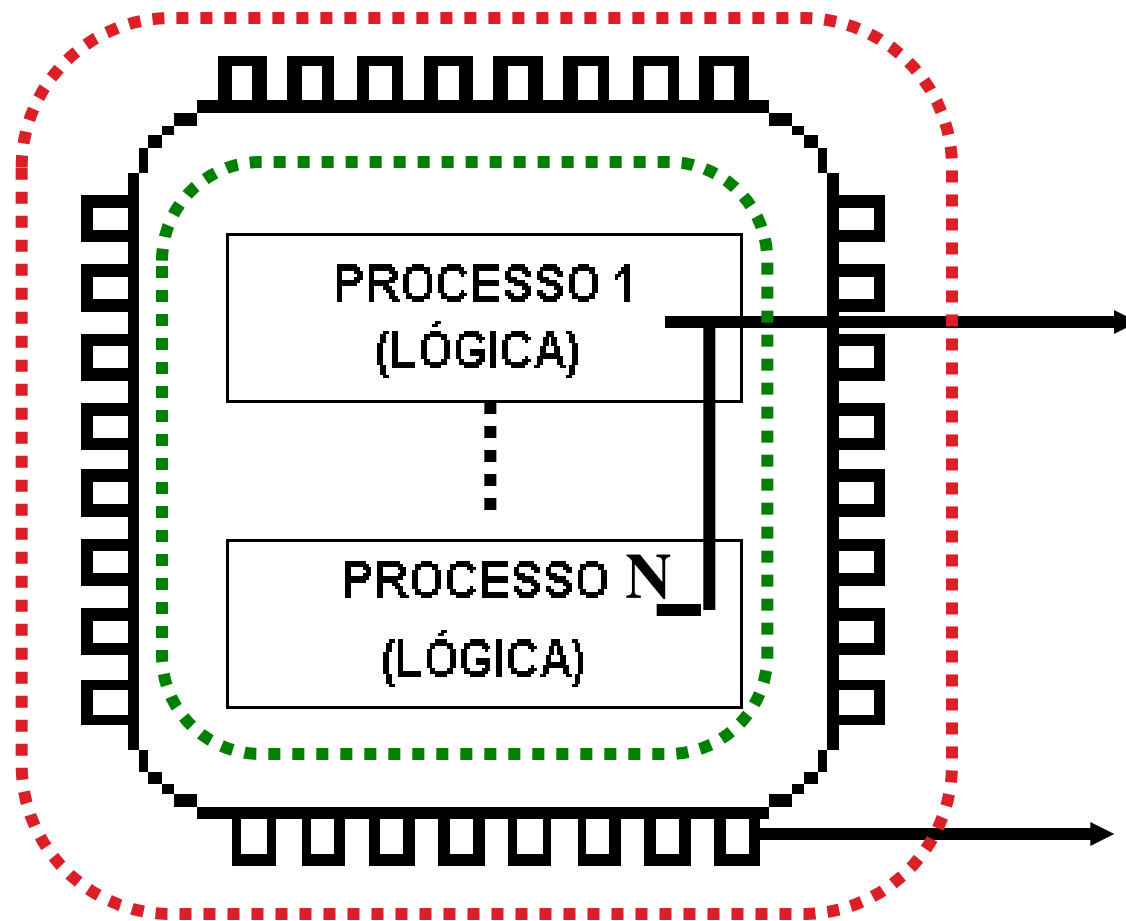
```
    U2: xnor port map (a(2), b(2), x(2));
```

```
    U3: xnor port map (a(3), b(3), x(3));
```

```
    U4: and4 port map (x(0), x(1), x(2), x(3), equals);
```

end estrut;





**ARCHITECTURE  
(ARQUITETURA)  
PROCESSOS**

**ENTITY  
(ENTIDADE)  
PINOS DE I/O**

# Identificadores

- ▶ Usados como referência a todos os objetos declarados
- ▶ Regras
  - Primeiro caractere deve ser uma LETRA
  - Não é CASE-SENSITIVE
    - Ex: Teste = teste = TESTE
  - Existem palavras reservadas
    - Ex: and, mux



# Sinais

- ❑ Comunicação de módulos em uma estrutura
- ❑ Temporizados.
- ❑ Podem ser declarados em *entity*, *architecture* ou *package*
- ❑ Não podem ser declarados em processos, mas podem ser utilizadas no interior destes.
- ❑ Sintaxe:
  - ❑ – **signal** identificador(es) : **tipo** [restrição] [:=expressão];
- ❑ Exemplo
  - ❑ – **signal** cont : **integer** range 50 **downto** 1;
  - ❑ – **signal** ground : **bit** := '0';

# Package (Pacotes)

Os pacotes (bibliotecas) contêm uma coleção de elementos incluindo descrição do tipos de dados

Analogia com C/C++: *#include* <library.h>.

Para incluir uma biblioteca no código VHDL (início do código):

**LIBRARY** <nome\_da\_biblioteca> e/ou

**USE** <nome\_da\_biblioteca>.all



# Package (Pacotes)

**Para utilizar STD\_LOGIC, deve-se inserir no início do código:**

```
library IEEE; ←  
use ieee.std_logic_1164.all; ←  
use ieee.std_ulogic_arith.all;  
use ieee.std_ulogic_unsigned.all;
```

**Biblioteca do usuário (default): **work.****



# Package (Pacotes)

**Permite a reutilização de um código já escrito.**

**Armazena:**

- **Declaração de tipos**
- **Declaração de constantes**
- **Declaração de subprogramas**
- **Declaração de mnemônicos**

**Pode ser dividido em parte de declaração e parte de corpo (opcional).**



# Package (Pacotes)

## Exemplo de Packages:

```
package <biblioteca> is  
  function soma(a,b: bit) return bit;  
  subtype dado is bit_vector(32 downto 0);  
  constant mascara : bit_vector(3 downto 0) := "1100";  
  alias terceiro_bit: bit is dado(3);  
end <biblioteca>.
```

# Package (Pacotes)

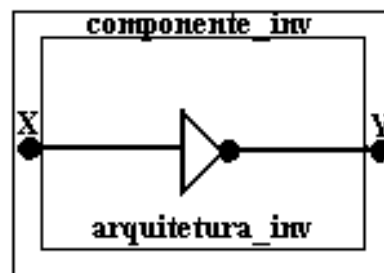
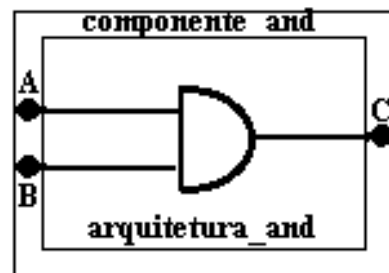
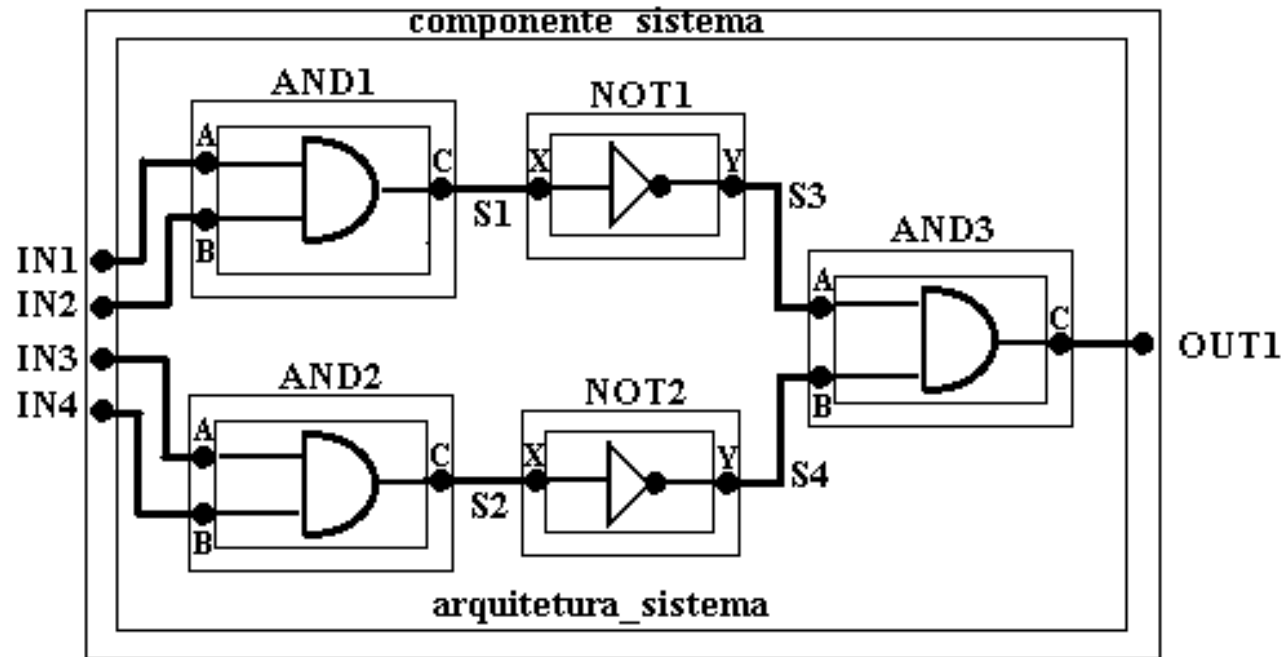
```
package basic_ops is
  component and2
    port(A, B : in bit; S : out bit);
  end component;
  component or2
    port(A, B : in bit; S : out bit);
  end component;
  component inv
    port(A in bit; S : out bit);
  end component;
end basic_ops;
```

```
package body basic_ops is
  architecture behev of and2 is
    S <= A AND B;
  end behev;
  architecture behev of or2 is
    S <= A OR B;
  end behev;
  architecture behev of inv is
    S <= NOT A;
  end behev;
end basic_ops;
```

# Especificando a Estrutura de um Sistema

O **component** é exatamente a descrição de um componente

O **port map** é um mapeamento deste componente em um sistema maior.



## Programa 1

-----  
-- Arquivo componente\_inv.vhd  
-- Modelo do inversor  
-----

**library** IEEE;  
**use** IEEE.std\_logic\_1164.all;

**entity** componente\_inv **is**  
**port**(  
    x : in std\_logic;  
    y : out std\_logic  
);  
**end** componente\_inv;

**architecture** arquitetura\_inv **of** compo-  
nente\_inv **is**

**begin**  
        y <= **not** x;  
**end** arquitetura\_inv;

## Programa 2

-----  
-- Arquivo componente\_and.vhd  
-- Modelo da porta AND  
-----

**library** IEEE;  
**use** IEEE.std\_logic\_1164.all;

**entity** componente\_and **is**  
**port**(  
    a : in std\_logic;  
    b : in std\_logic;  
    c : out std\_logic  
);  
**end** componente\_and;

**architecture** arquitetura\_and **of** com-  
ponente\_and **is**

**begin**  
        c <= a **and** b;  
**end** arquitetura\_and;



### Programa 3

-----  
-- Arquivo componente\_sistema.vhd  
-----

```
library IEEE;
use IEEE.std_logic_1164.all;

entity componente_sistema is
port(
    in1 : in std_logic;
    in2 : in std_logic;
    in3 : in std_logic;
    in4 : in std_logic;
    out1 : out std_logic
);
end componente_sistema;

architecture arquitetura_sistema of componente_sistema is

    component componente_and
        port( a: in std_logic; b : in std_logicbit; c : out std_logic);
    end component;

    component componente_inv
        port( x: in std_logic; y : out std_logic);
    end component;

    signal s1, s2, s3, s4 : std_logic;

    begin
        and1 : componente_and port map (a => in1, b => in2, c => s1);
        and2 : componente_and port map (a => in3, b => in4, c => s2);
        and3 : componente_and port map (a => s3, b => s4, c => ut1);
        inv1 : componente_inv port map (x => s1, y => s3);
        inv2 : componente_inv port map (x => s2, y => s4);
    end arquitetura_sistema;
```

# Considerações importantes

1. VHDL **NÃO É** uma linguagem de programação
2. O VHDL deve ser descrito após a arquitetura, e não a arquitetura após o VHDL.

# Considerações importantes

	Linguagem de Programação	VHDL
Propósito	Software	Hardware
Entrada	Texto e Ferramentas Visuais	
Desenvolvimento	Compilação e Ligação / Interpretação	Compilação para Simulação e Síntese para Hardware
Depuração	Execução e Visualização dos Resultados	Simulação e Visualização das Formas de Onda
Instruções	Somente Seqüenciais	Concorrentes e Seqüenciais

# Ferramenta Utilizada

- ▶ Quartus II Web Edition
- ▶ [www.altera.com](http://www.altera.com)
- ▶ Compilador e Simulador integrados
- ▶ Licença gratuita para estudantes

# Exercícios

- ▶ Instalar, ler a apostila e repetir os exemplos apresentados no Quartus II
- ▶ Criar uma pasta **Componentes** e copiar os arquivos componente\_inv e componente\_and
- ▶ Criar os arquivos componente\_or, componente\_xor, componente\_xnor, componente\_nand e componente\_nor