VHDL - VHSIC Hardware Description Language

Organização e Arquitetura de Computadores I

Leonardo Augusto Casillo

2003-2

Referências bibliográficas

PELLERIN, David. TAYLOR, Douglas. *VHDL Made Easy.*
Prentice-Hall PTR. 1997.

SKANHILL, Kevin. *VHDL for programmable logic.*
Addison-Wesley. 1996.

ASHENDER, Peter. *The VHDL Cookbook*
Disponível na internet. 1990.

Apostilas disponíveis na página do curso:
Profs.: Fernando Moraes, David Déharbe, Anderson Terroso.

VHDL - VHSIC Hardware Description Language

Introdução

- **O que significa VHDL?**

  Very **H**igh **S**peed **I**ntegrated **C**ircuit

  Hardware

  Description

  Language

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 linguagens de HDLs:

– 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: recomeçou-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 de se utilizar VHDL

- **Vantagens**
  - **Projeto independente da tecnologia;**
  - **Ferramentas de CAD compatíveis entre si;**
  - **Facilidade na atualização dos projetos;**
  - **Reduz tempo de projeto e custo;**
  - **Elimina erros de baixo nível;**
  - **Reduz "*time-to-market*".**
- **Desvantagens**
  - **Hardware gerado é menos otimizado;**
  - **Controlabilidade/Observabilidade de projeto reduzidas;**
  - **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**

– 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:*

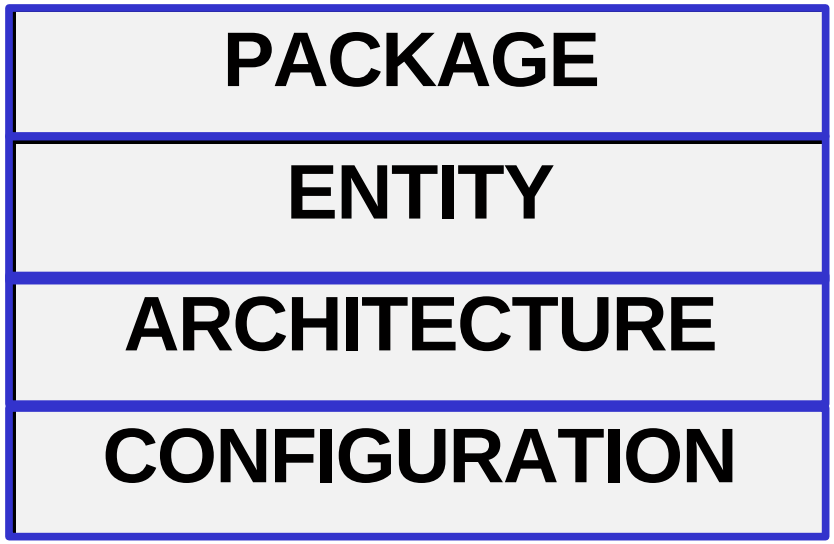
- **Síntese: compilação de um código VHDL para uma descrição abstrata.**
- **Otimização: seleção da melhor solução de implementação para uma dada tecnologia.**
- **Fitting: lógica sintetizada e otimizada mapeada nos recursos oferecidos pela tecnologia.**

– Ciclo de Projeto:

- **Simulação do modelo:** **resultados mais apurados de comportamento e *timing*.**

- **Geração:** **configuração das lógicas programáveis ou de fabricação de ASICs.**

Componentes de um projeto VHDL



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

LIBRARY IEEE; **USE IEEE.STD_LOGIC_1164.all;** **USE IEEE.STD_LOGIC_UNSIGNED.all;**	**PACKAGE (BIBLIOTECAS)**
ENTITY exemplo **IS** **PORT (** < descrição dos pinos de entrada e saída > **);** **END** exemplo;	**ENTITY (PINOS DE I/O)**
ARCHITECTURE teste **OF** exemplo **IS** **BEGIN** **PROCESS(** <pinos de entrada e signal > **)** **BEGIN** < descrição do circuito integrado > **END PROCESS**; **END** teste;	**ARCHITECTURE (ARQUITETURA)**

Entity (Entidade)

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

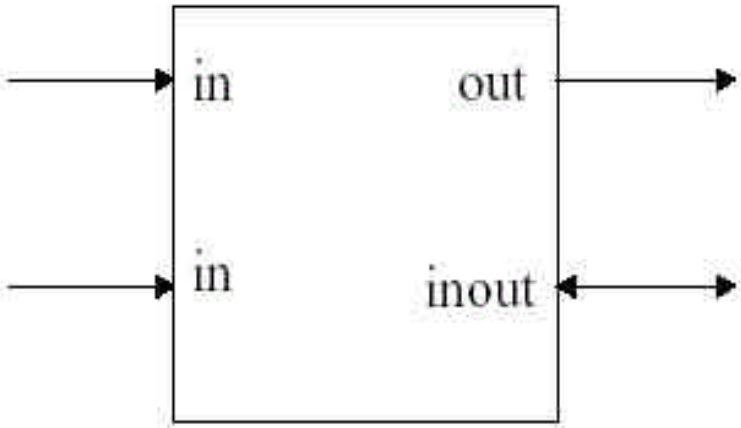
```
Entity <nome_da_entidade> is
port  (
        entrada_a       : in <tipo>;
        entrada_b       : in <tipo>;
        saída           : out <tipo>;
        );
end <nome_da_entidade>;
```

Entity (Entidade)

Ports: **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)

Tipos mais utilizados:

bit	Assume valores ‘0’ ou ‘1’. **x: in bit;**
bit_vector	Vetor de bits. **x: in bit_vector(7 downto 0);** **x: in bit_vector(0 to 7);**
std_logic	**x: in std_logic;**
std_logic_vector	**x: in std_logic_vector(7 downto 0);** **x: in std_logic_vector(0 to 7);**
boolean	Assume valores TRUE ou FALSE

Entity (Entidade)

STD_LOGIC:

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

'U': não inicializada	**'Z': alta impedância**
'X': desconhecida	**'W': desconhecida fraca**
'0': valor '0'	**'L': '0' fraca (Low)**
'1': valor '1'	**'H": '1' fraca (High)**
'-': *Don't care.*	

Architecture (Arquitetura)

A função de uma *entity* é determinada pela sua *architecture.*

A organização de uma *architecture* é dada por:

Declarações:

sinais, constantes, componentes, subprogramas.

Comandos:

begin

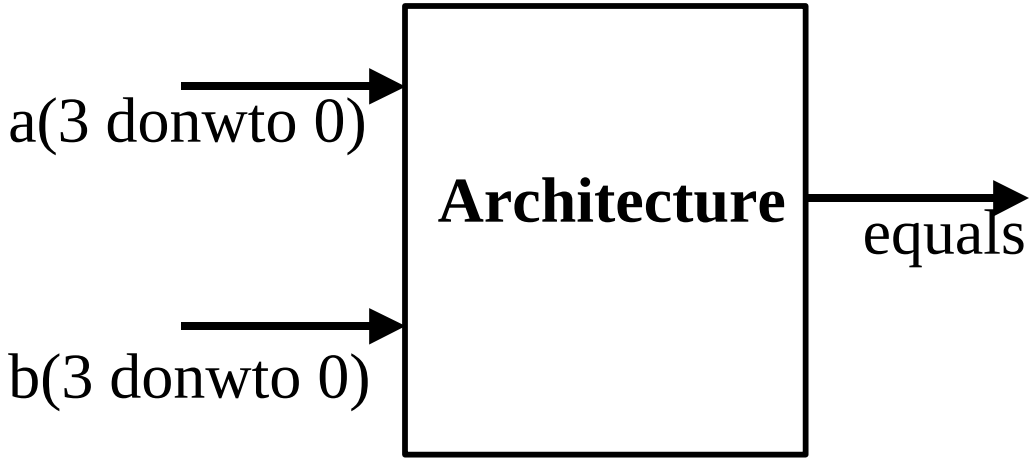
blocos, atribuições a sinais, chamadas a subprogramas, instanciação de componentes, processos.

end

Architecture (Arquitetura)

A arquitetura de uma entidade pode ser descrita por três formas distintas: descrição comportamental, descrição por fluxo de dados (*data-flow*) e descrição estrutural.

Exemplo simples: Comparador de 4 bits



Descrição Comportamental

```
-- comparador de 4 bits
entity comp4 is
    port (    a, b: in bit_vector (3 downto Æ);
              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 < = ‘O’ ;
end if;
end process comp;
end comport;
```

Mesmo nome da entidade

Processos conconrrentes

Descrição por Data-Flow

```
-- comparador de 4 bits
entity comp4 is
    port (   a, b: in bit_vector (3 downto Æ);
             equals: out bit);
end comp4;

architecture fluxo of comp4 is
begin
equals < = ‘1’ when (a=b) else ‘0’;
end fluxo;
```

Expressões lógicas

Descrição Estrutural

```
-- comparador de 4 bits
entity comp4 is
    port (   a, b: in bit_vector (3 downto Æ);
             equals: out bit);
end comp4;

architecture estrut of comp4 is
signal x bit_vector (0 to 3);
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;
```

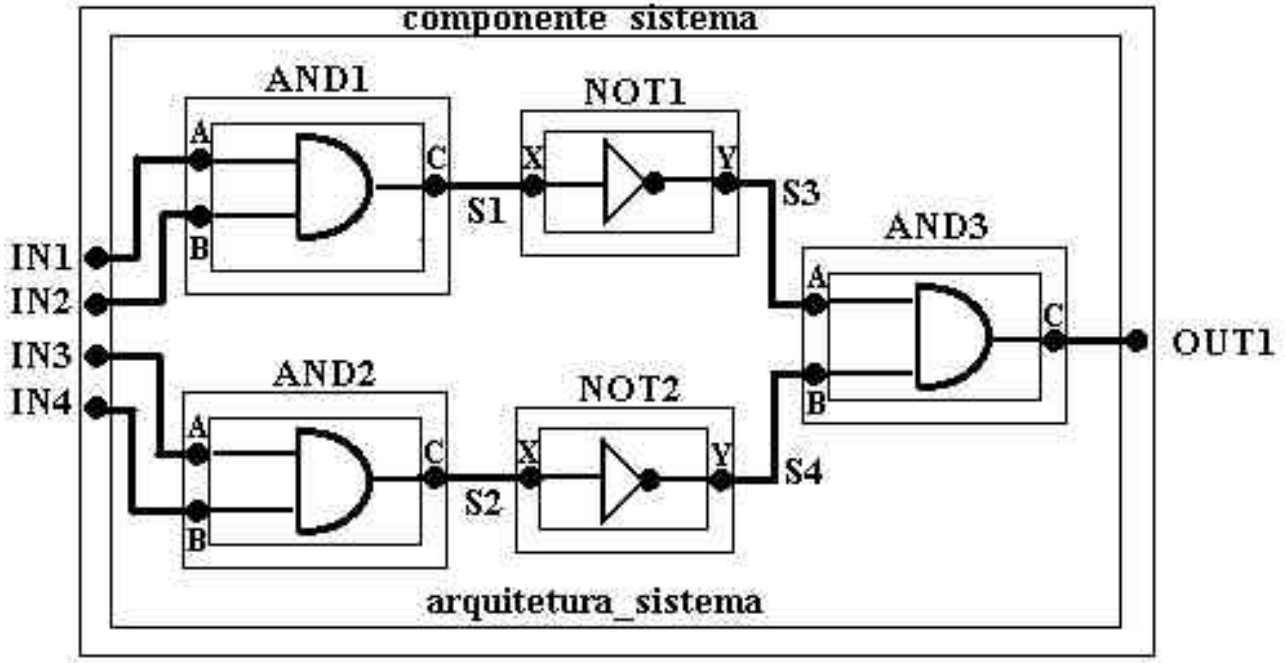
Ligações entre componentes

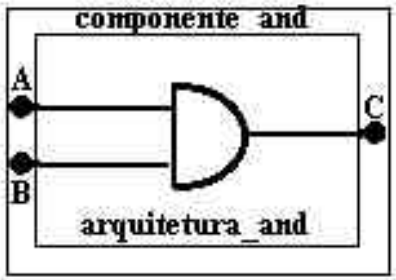
Sinais

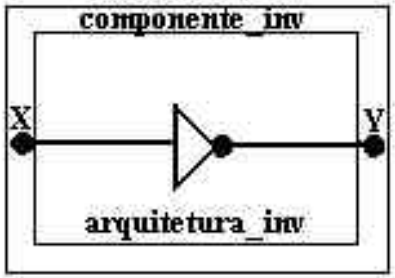
- **Comunicação de módulos em uma estrutura**
- **Temporizados.**
- **Podem ser declarados em *entity*, *architecture* ou *package***
- **Não podem ser declarados em processos, mas podem ser utilizadas no interior destes.**
- **Sintaxe:**
- **– signal identificador(es) : tipo [restrição] [:=expressão];**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

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 bit;
        y : out bit
        );
end componente_inv;

architecture arquitetura_inv of
componente_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 bit;
        b : in bit;
        c : out bit
        );
end componente_and;

architecture arquitetura_and of componete_and is

        begin
                c <= a and b;
end arquitetura_and;
```

Programa 3

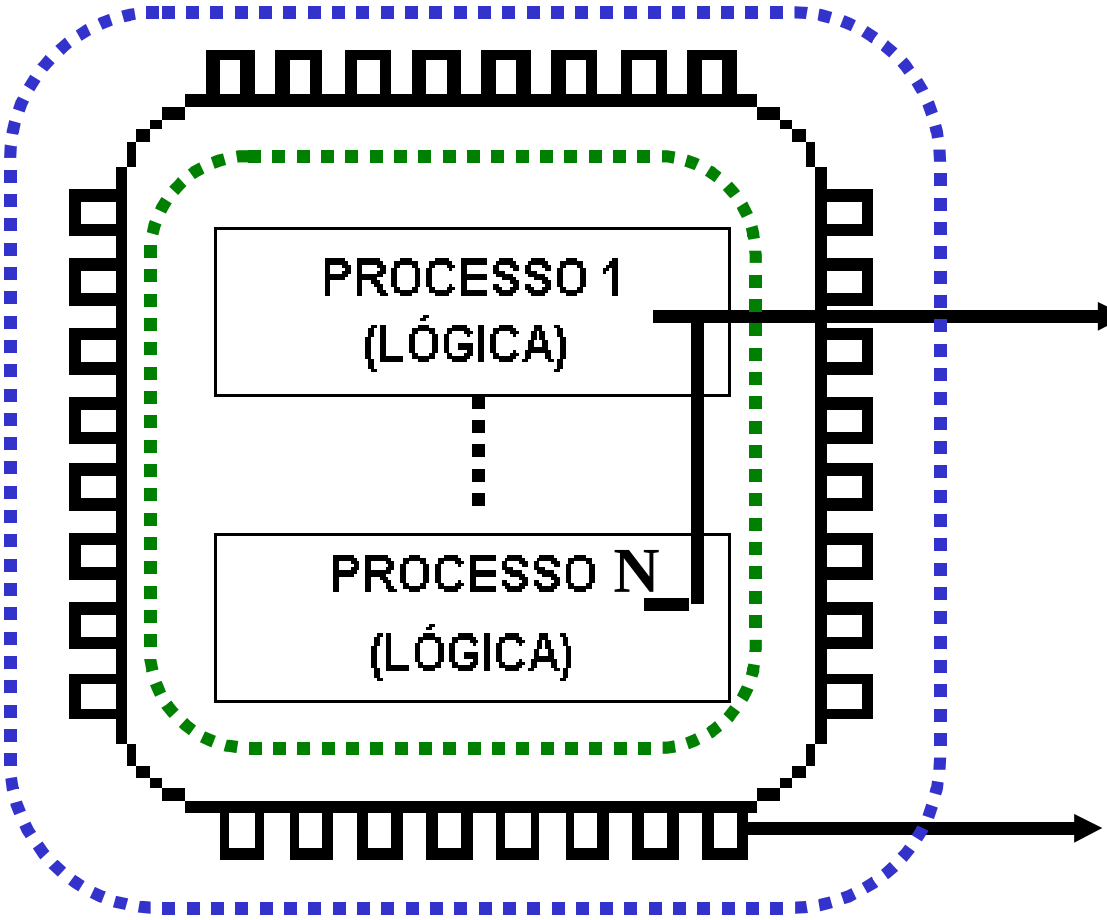
```
-------------------------------------------------
-- Arquivo componente_sistema.vhd
-- Modelo da porta AND
-------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

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

architecture arquitetura_sistema of componente_sistema is
```

Declaração Dos Componentes

```
component componente_and
            port( a: in bit; b : in bit; c : out bit);
end component;

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

signal s1, s2, s3, s4 : bit;

        begin
```

Conexão entre Componentes

```
            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;
```



ARCHITECTURE
(ARQUITETURA)
PROCESSOS

ENTITY
(ENTIDADE)
PINOS DE I/O

Package (Pacotes)

Os pacotes (bibliotecas) contém 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 donwto 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;
```

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.**