FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

T566 –SISTEMAS DIGITAIS AVANÇADOS

Aula 9 - VHDL

Prof. Danilo Reis

O que é?

VHDL ou "VHSIC Hardware Description Language" (Linguagem de descrição de hardware VHSIC "Very High Speed Integrated Circuits")

Histórico

- **VHDL** foi desenvolvida pelo DARPA em 1980;
- Visava substituir utilização de esquemáticos em ASICs;
- Padronizada em 1987 pela IEEE;
- Pequenas alterações em 2000 e 2002;

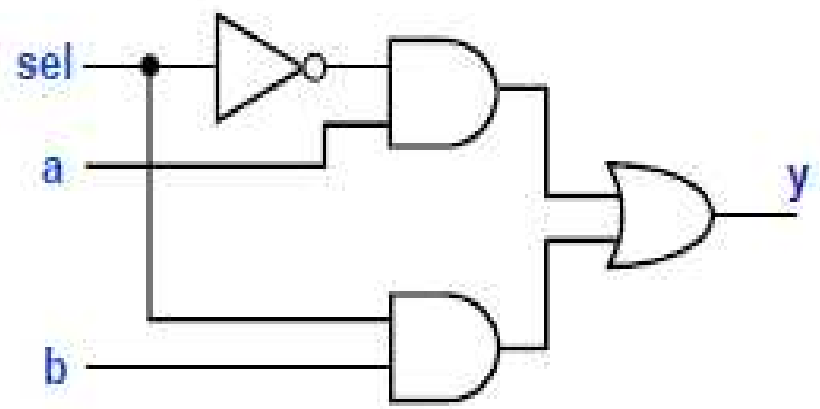
Primeiros Conceitos

Uma descrição VHDL é dividida em duas partes fundamentais:

- **Entidade (Entity)** – Descreve a interface do sistema
- digital descrito com o mundo externo. Apresenta a
- definição dos pinos de entrada e saída.
- **Arquitetura (Architecture)** – Descreve o
- comportamento ou a estrutura do sistema digital.
- Define como a função do sistema é realizada.

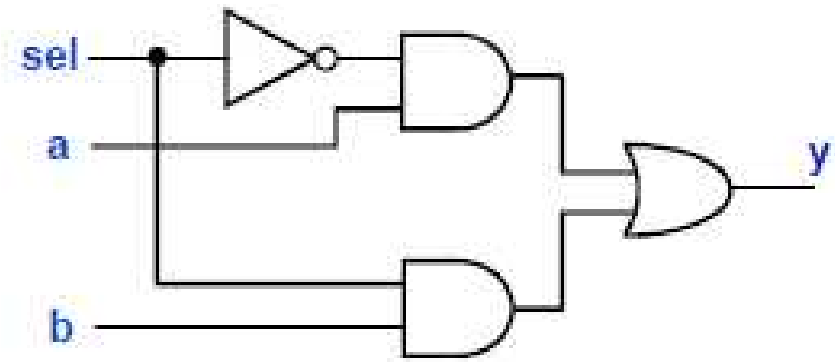
Primeiros Conceitos

Circuito Exemplo:



Primeiros Conceitos: Entidade

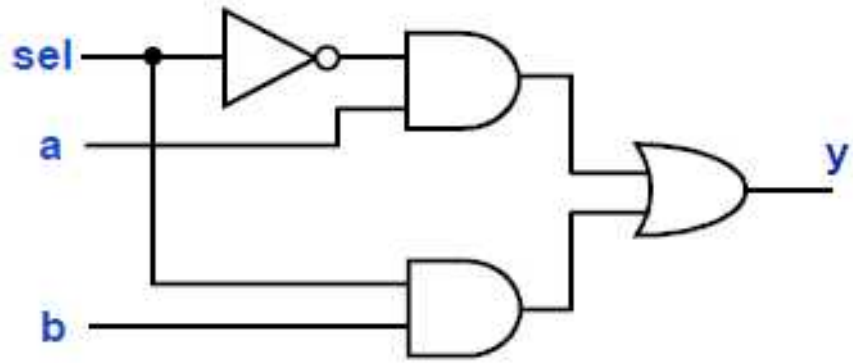
```
ENTITY exemplo1 IS
   PORT ( sel : IN BIT;
          a : IN BIT;
          b : IN BIT;
          y : OUT BIT);
END exemplo1;
```



Importante: o nome do arquivo vhdl DEVE ser o mesmo nome da entidade. Neste caso, o nome do arquivo seria *exemplo1.vhd*

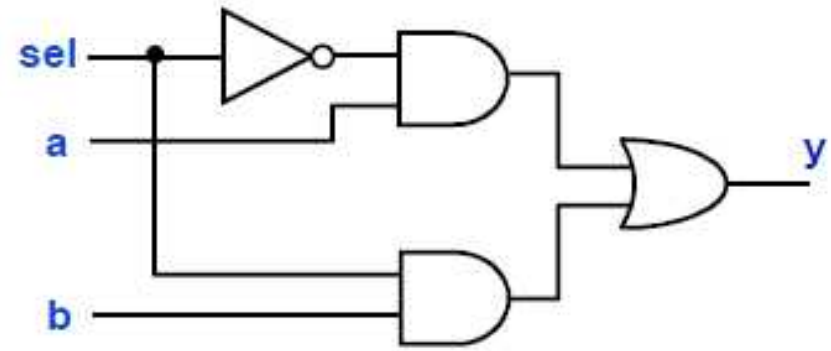
Primeiros Conceitos: Arquitetura

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

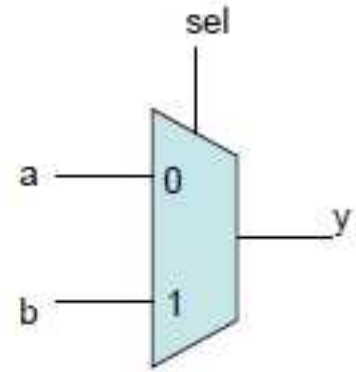


Primeiros Conceitos: circuito completo

```
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;
```



Multiplexador 2 para 1

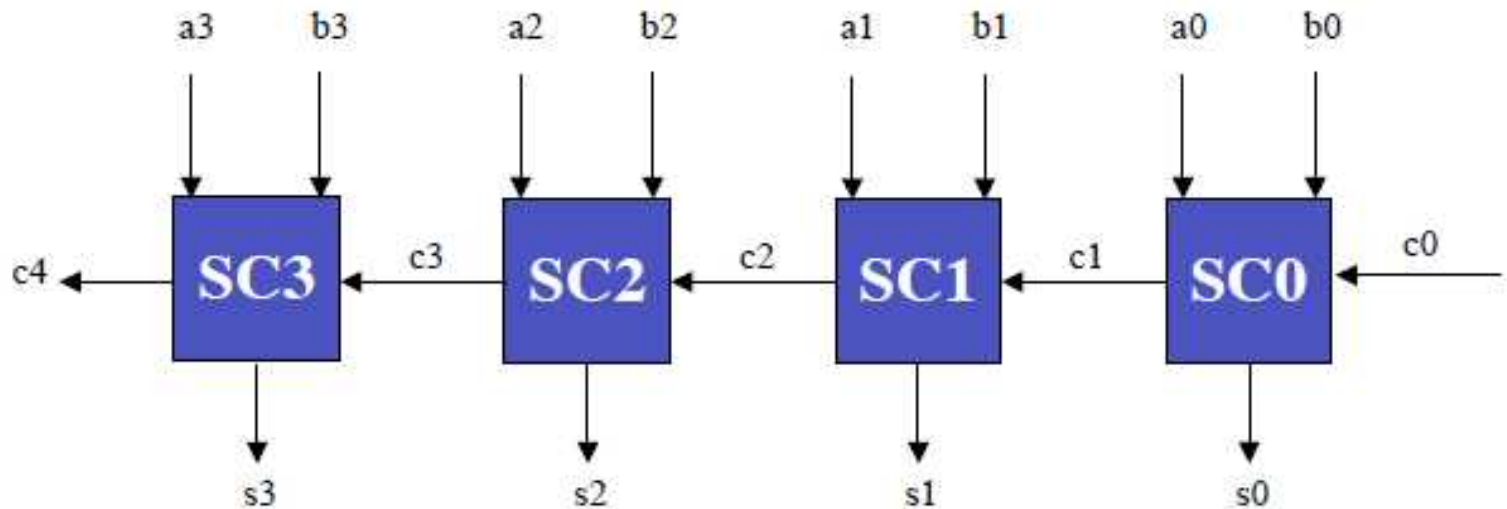



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux2para1 IS
PORT ( sel, a, b : IN STD_LOGIC;
          y : OUT STD_LOGIC);
END mux2para1;

ARCHITECTURE comportamento OF mux2para1 IS
BEGIN
   WITH sel SELECT
       y <= a WHEN '0',
              b WHEN OTHERS;
END comportamento;
```

Descrição de um Operador Aritmético

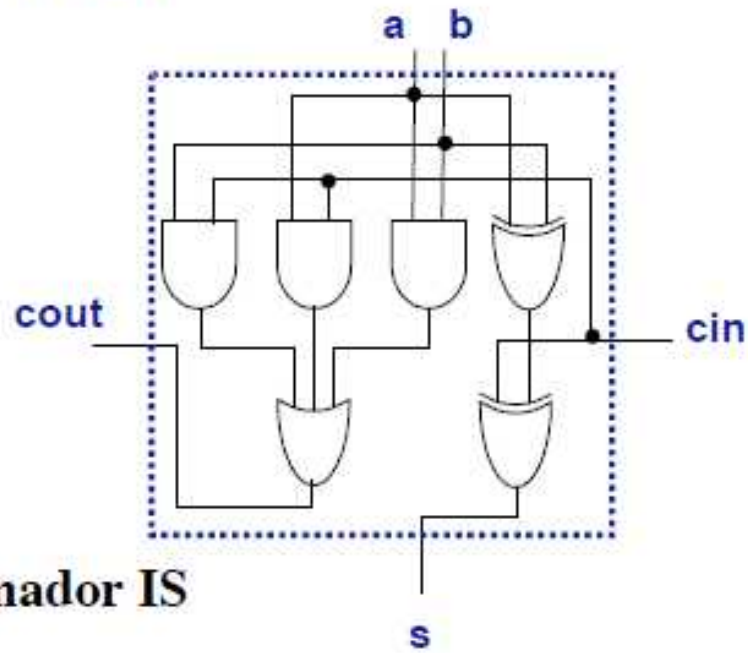
Somador de 4 bits:

- Com hierarquia
- Usando 4 somadores completos de 1 bit



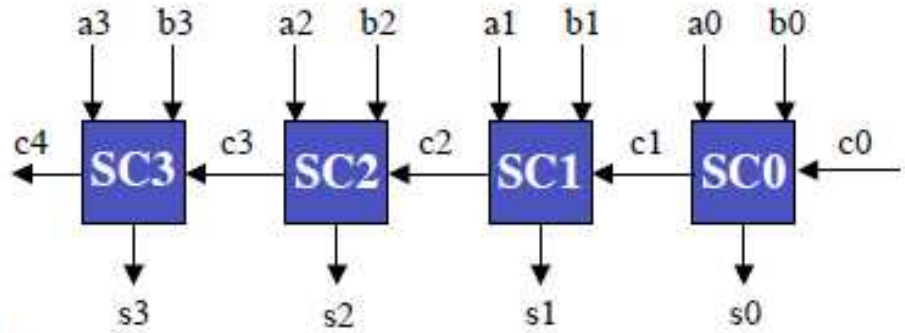
Somador Completo de 1 bit

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

ENTITY somador IS
   PORT (cin, a, b : IN STD_LOGIC;
         s, cout : OUT STD_LOGIC);
END somador;

ARCHITECTURE comportamento OF somador IS
BEGIN
     s <= a XOR b XOR cin;
     cout <= (a AND b) OR (a AND cin) OR (b AND cin);
END comportamento;
```

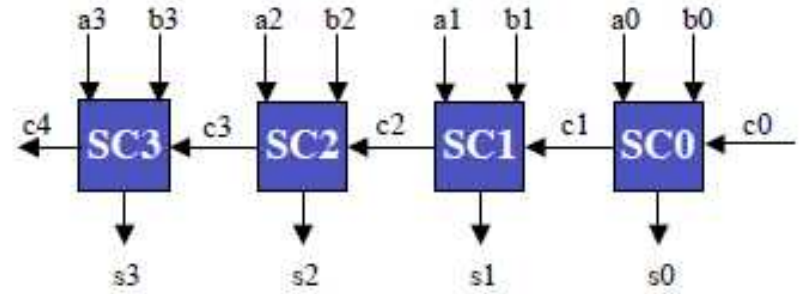


Somador de 4 bits (Nível Mais Alto da Hierarquia)

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

ENTITY somador4bits IS
   PORT (c0, a3, a2, a1, a0, b3, b2, b1, b0 : IN STD_LOGIC;
         s3, s2, s1, s0, c4 : OUT STD_LOGIC);
END somador4bits;

ARCHITECTURE estrutura OF somador4bits IS
   SIGNAL c1, c2, c3: STD_LOGIC;
   COMPONENT somador
      PORT (cin, a, b : IN  STD_LOGIC;
            s, cout : OUT  STD_LOGIC);
END COMPONENT;

BEGIN
     SC0: somador PORT MAP (c0, a0, b0, s0, c1);
     SC1: somador PORT MAP (c1, a1, b1, s1, c2);
     SC2: somador PORT MAP (c2, a2, b2, s2, c3);
     SC3: somador PORT MAP (c3, a3, b3, s3, c4);
END estrutura;
```



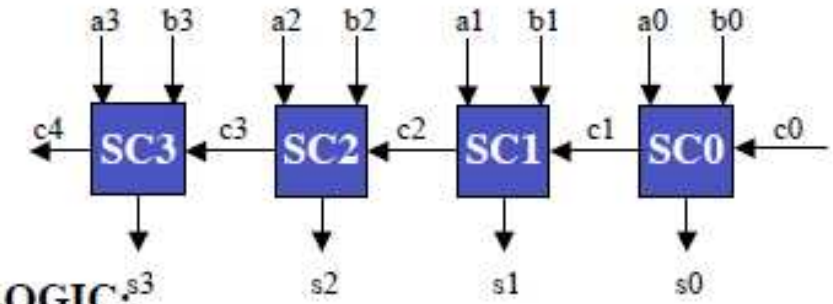
Somador de 4 bits




```
BEGIN
    SC0: somador PORT MAP (c0, a0, b0, s0, c1);
    SC1: somador
        PORT MAP (cin => c1,
                  a => a1,
                  b => b1,
                  s =>  s1,
                  cout => c2);
    SC2: somador
        PORT MAP (s => s2,
                  b => b2,
                  a => a2,
                  cin => c2,
                  cout =>  c3);
    SC3: somador PORT MAP (c3, a3, b3, s3, c4);
END comportamento;
```

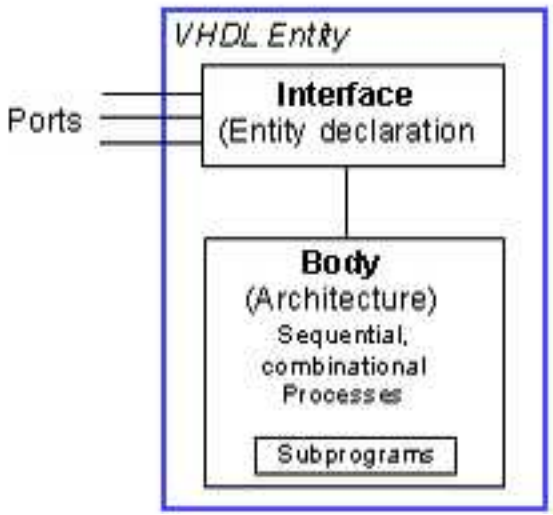
Quando o mapeamento dos pinos é explícito, não é necessário seguir a ordem da declaração

Somador de 4 bits sem hierarquia




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

ENTITY somador4bits IS
   PORT (cin, a3, a2, a1, a0, b3, b2, b1, b0 : IN STD_LOGIC;
         s3, s2, s1, s0, cout : OUT STD_LOGIC);
END somador4bits;

ARCHITECTURE comportamento OF somador4bits IS
   SIGNAL c1, c2, c3: STD_LOGIC;
BEGIN
   s0 <= a0 XOR b0 XOR c0;
   c1 <= (a0 AND b0) OR (a0 AND c0) OR (b0 AND c0);
   s1 <= a1 XOR b1 XOR c1;
   c2 <= (a1 AND b1) OR (a1 AND c1) OR (b1 AND c1);
   s2 <= a2 XOR b2 XOR c2;
   c3 <= (a2 AND b2) OR (a2 AND c2) OR (b2 AND c2);
   s3 <= a3 XOR b3 XOR c3;
   c4 <= (a3 AND b3) OR (a3 AND c3) OR (b3 AND c3);
END comportamento;
```

Estrutura Básica



Módulo definido com interface e body. Onde a interface define as entradas e saídas do módulo e o body o comportamento e implementação deste.

Estrutura Entity

```
entity NAME_OF_ENTITY is [ generic generic_declarations);]
     port (signal_names: mode type;
           signal_names: mode type;
               :
           signal_names: mode type);
end [NAME_OF_ENTITY] ;
```

O NAME_OF_ENTITY e um identificador definido pelo usuário

· signal_names consiste lista de um ou mais identificadores que especificam a interface de sinais com o mundo externo.

· **mode**: indica a direção do sinal:

o **in** – define sinal como input

o **out** – define o sinal como output da entidade.

o **buffer** – define o sinal como saída e o valor pode ser lido dentro da arquitetura da entidade

o **inout** – define o sinal como input ou output.

Estrutura Entity

type: são built-in ou user-defined sinal type. Exemplos de types são bit, bit_vector, Boolean, character, std_logic, e std_ulogic.

- *bit* – pode ter valor 0 e 1
- *bit_vector* – e um vetor de bits (e.g. bit_vector (0 to 7)
- *std_logic, std_ulogic, std_logic_vector, std_ulogic_vector*: podem ter 9 valores para indicar o valor e tamanho de um sinal. Std_ulogic e std_logic são preferidos à bit ou bit_vector types.
- *boolean* – podem ter valores TRUE e FALSE
- *integer* – podem ter valores na faixa dos inteiros
- *real* – podem ter valores na faixa dos reais
- *character* – qualquer character
- *time* – indica tempo

· **generic:** a declaração generic é opcional e determina as constantes locais usadas para timing e sizing (exemplo: bus widths) da entity. Um generic pode ter um valor default. A sintaxe para um generic:

generic (
constant_name: type [:=value] ;
constant_name: type [:=value] ;

Estrutura Entity

· **generic:** a declaração generic é opcional e determina as constantes locais usadas para timing e sizing (exemplo: bus widths) da entity. Um generic pode ter um valor default. A sintaxe para um generic:

generic (
constant_name: type [:=value] ;
constant_name: type [:=value] ;
:
constant_name: type [:=value]);

Nomes

Apenas letras, dígitos e sublinhados podem ser usados;
* O primeiro caracetere deve ser uma letra;
* O último caractere não pode ser um sublinhado;
* Não são permitidos dois sublinhados consecutivos

Não é sensível à “Caixa Alta ou Baixa”
– inputa, INPUTA e InputA se referem à mesma variável.

- Comentários
  – ‘--’ marca um comentário até o final da linha atual
  – Se você deseja comentar múltiplas linha, um ‘--’ precisa ser colocado no início de cada linha.
- As sentenças são terminadas por ‘;’
- Atribuição de valores aos sinais: ‘<=’
- Atribuição de valores às variáveis: ‘:=

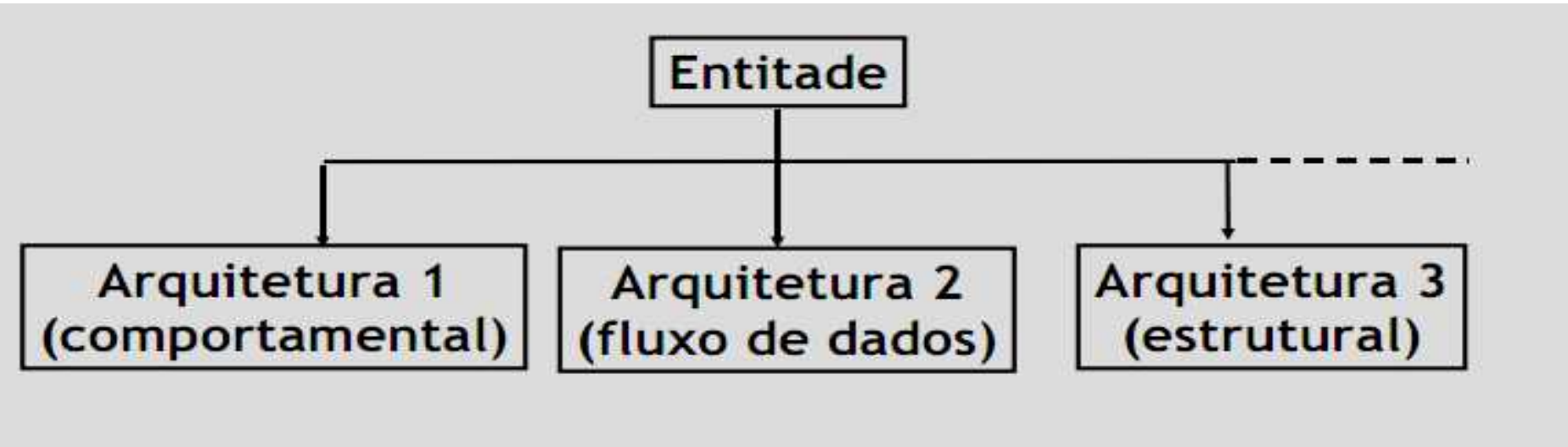
Nomes

Nomes Legais	Nomes Ilegais
rs_clk	_rs_clk
ab08B	sinal#1
A_1023	A__1023
	rs_clk_

Estrutura Architectuture

É o corpo do sistema, onde são feitas as atribuições, operações, comparações, etc…

Declarado como *architecture* nome *of* entidade *is*. Poder existir várias *architecture* na mesma *entity*.



Estrutura Architectuture

```
   architecture architecture_name of NAME_OF_ENTITY is
-- Declarations
      -- components declarations
      -- signal declarations
      -- constant declarations
      -- function declarations
      -- procedure declarations
      -- type declarations
:

begin
-- Statements
    :
 end architecture_name;
```

Exemplo

```
      entity XNOR2 is
    port (A, B: in std_logic;
          Z: out std_logic);
    end XNOR2;

architecture behavioral_xnor of XNOR2 is
-- signal declaration (of internal signals X, Y)
    signal X, Y: std_logic;
begin
X <= A and B;
Y <= (not A) and (not B);
Z <= X or Y;
    End behavioral_xnor;
```

Process

Diretiva usada quando se quer fazer uma lista de operações a serem executadas. Implementada dentro de *architecture*. Possui forma estruturada.

Exemplo de *process*:

atrib : process begin A ⇐ X; B ⇐ Y; end process atrib;

Package

Usado quando precisa-se usar um comando que não existe nas bibliotecas padrão. Deve ser definido antes do inicio da entity. Para usar a package é necessário usar duas declarações: libraryuse. O package mais conhecido é o STD_LOGIC_1164 da IEEE por conter a maioria dos comandos adicionais usados na linguagem.

Exemplo de package:

library IEEE; use IEEE.std_logic_1164.all;
Outras Librarys

- IEEE 1076.1 VHDL analógica e de sinal misto
- IEEE 1076.1.1 VHDL-AMS pacotes padrão (stdpkgs)
- IEEE 1076.2 VHDL pacotes matemáticos (math)
- IEEE 1076.3 VHDL pacotes sintetizado (vhdlsynth)
- IEEE 1076.3 VHDL pacotes sintetizado - Ponto flutuante (fphdl)
- IEEE 1076.4 VHDL bibliotecas para ASIC: vital

Sinais

- Sinais são utilizados para comunicação entre componentes.
- Sinais podem ser interpretados com fios físicos, reais.

Exemplo:

```
SIGNAL <nome_sinal> : <tipo> [:= <valor>];
SIGNAL enable : BIT;
SIGNAL output : bit_vector(3 downto 0);
SIGNAL output : bit_vector(3 downto 0) := "0111";
```

Variáveis

Usadas apenas em processos e subprogramas (funções e procedimentos), as variáveis usualmente não estão disponíveis para múltiplos componentes e processos. Todas as atribuições de variáveis tem efeito imediato.

VARIABLE <nome_variavel> : <tipo> [:= <valor>];

Exemplo:

VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000"; VARIABLE freq : INTEGER;

Diferenças Variáveis x Sinais

Usadas apenas em processos e subprogramas (funções e procedimentos), as variáveis usualmente não estão disponíveis para múltiplos componentes e processos. Todas as atribuições de variáveis tem efeito imediato.

VARIABLE <nome_variavel> : <tipo> [:= <valor>];

Exemplo:

VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000"; VARIABLE freq : INTEGER;

Escopo dos objetos

- O VHDL limita a visibilidade dos objetos, dependendo de onde eles são declarados.
- Regras de escopo se aplicam a constantes, variáveis, sinais e arquivos.
- O escopo dos objetos é definido como a seguir:
  - Objetos declarados em um pacote são globais para todas as entidades que usam aquele pacote.
  - Objetos declarados em uma entidade são globais para todas as arquiteturas que utilizam aquela entidade.
  - Objetos declarados em um arquitetura são disponíveis a todas as sentenças naquela arquitetura.
  - Objetos declarados em um processo são disponíveis apenas para aquele processo.

Tipos de Dados

- bit values: '0', '1'
- boolean values: TRUE, FALSE
- integer values: -(2E31) to +(2E31 – 1)
- std_logic values: 'U','X','1','0','Z','W','H','L','-'
  - U' = uninitialized
  - 'X' = unknown
  - 'W' = weak 'X‘
  - 'Z' = floating
  - 'H'/'L' = weak '1'/'0‘
  - '-' = don't care
- std_logic_vector (n downto 0)/std_logic_vector (0 upto n);

Constantes

Servem para aumentar a legibilidade do código e facilitar a sua modificação.

CONSTANT <nome_da_constante> : <tipo> := <valor>;

Exemplo:

CONSTANT PI : REAL := 3.14; CONSTANT WIDTH : INTEGER := 8;

Operadores e expressões

Operadores Lógicos

Os operadores *and*, *or*, *nand*, *nor*, *xor* e xnor exigem dois operandos, já o operador *not* exige apenas um operando.

Deslocamento

As operações podem ser:Restrito a vetores. Exige dois operandos, um sendo o *array* e o outro um *integer*, que é o número de posições a serem deslocadas.

shift left logical (deslocamento lógico a esquerda);

shift right logical (deslocamento lógico a direita);

shift left arithmetic (deslocamento aritmético a esquerda);

shift right arithmetic (deslocamento aritmético a direita);

rotate left logical (rotacionamento lógico a esquerda);

rotate right logical (rotacionamento lógico a direita).

Operadores e expressões

Operadores Aritméticos

+ → soma ou identidade;

- → subtração ou negação;

* → multiplicação;

/ → divisão;

mod → módulo;

rem → resto da divisão;

abs → valor absoluto;

** → exponenciação.

Atribuição e Comparações

⇐ → atribuição;

= → igual;

/= → diferente;

< → menor;

⇐ → menor ou igual;

> → maior;

>= → maior ou igual;

Comandos After e Wait (Não sintetizável)

After tem a finalidade de ativar o estado indicado depois de determinado tempo.

Exemplo de *after*:

x ⇐ '1' after 3s, '0' after 5s, '1' after 7s, '0' after 8s;

O resultado graficamente seria:

Já wait 'segura' o processo por determinado tempo.

Exemplo de *wait*:

x ⇐ '0'; wait for 2s; x ⇐ '1'; wait for 3s; x ⇐ '0'; wait for 1s;

Comando if then else

Se a condição for verdadeira será executado o que estiver dentro de *then*, caso contrário será executado o que estiver dentro de *else*.

Exemplo de *if then else*:

```
cmp : process begin if A = B then C ⇐ 0; else C ⇐ 1; end if;
end process cmp;
```

Comando Case

Quando o teste de condição de uma variável poder assumir vários opções, é recomendado o uso do *case*.

Exemplo de *case*:

converte : process begin case Bin is when "0000" ⇒ Dec ⇐ 0; when ″0001″ ⇒ Dec ⇐ 1; when ″0010″ ⇒ Dec ⇐ 2; when ″0011″ ⇒ Dec ⇐ 3; when others ⇒ Dec ⇐ −1; end case; end process converte;

Comando For Loop

Enquanto o contador estiver dentro da faixa especificada o loop é executado.

Exemplo de *for loop*:

conta : process begin for i in 5 downto 0 loop Num ⇐ Num + 1; end loop; end process conta;

Comando Next

Quando se quer pular determinados comandos e ir diretamente para outro usa-se o comando *next*.

Exemplo de *next*:

soma : process begin aux : for i in 3 downto 0 loop Num ⇐ Num + X; if Num = 10 then Num ⇐ 0; next alfa; end if end loop aux; end process soma;

Referências

- http://pt.wikipedia.org/wiki/VHDL
- http://www.dcc.ufrj.br/~gabriel/circlog/vhdl.pdf
- http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html