Pontifícia Universidade Católica
do Rio Grande do Sul

FACULDADE DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LINGUAGEM DE DESCRIÇÃO DE HARDWARE

VHDL

PROF. JULIANO D'ORNELAS BENFICA

Porto Alegre

2009

Sumário

1 Introdução

1.1 O que é VHDL?

VHDL é uma forma de se descrever, através de um programa, o comportamento de um circuito ou componente digital.

1.2 O que Significa VHDL?

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

Hardware

Decription

Language

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

1.3 Vantagens e Desvantagens

Vantagens de se utilizar VHDL:

- Projeto independente da tecnologia;
- Reconfiguração parcial e dinâmica;

- Intercâmbio de projetos entre grupos de pesquisa;
- Reduz tempo de projeto;
- Aumento da performance;
- Fabricação direta de um circuito integrado.

Desvantagens de se utilizar VHDL:

- Hardware gerado é menos otimizado.

2 Estrutura de um Programa em VHDL

A estrutura de um programa VHDL, baseia-se em 4 blocos:

- **PACKAGE** Onde são declarados as constantes, tipos de dados, sub-programas;
- **ENTITY** Onde são declarados os pinos de entrada e saída;
- **ARCHITECTURE** Onde são definidas as implementações do projeto;
- **CONFIGURATION** Onde são definidas as arquiteturas que serão utilizadas.

Abaixo está representado a estrutura de um programa descrito em VHDL:

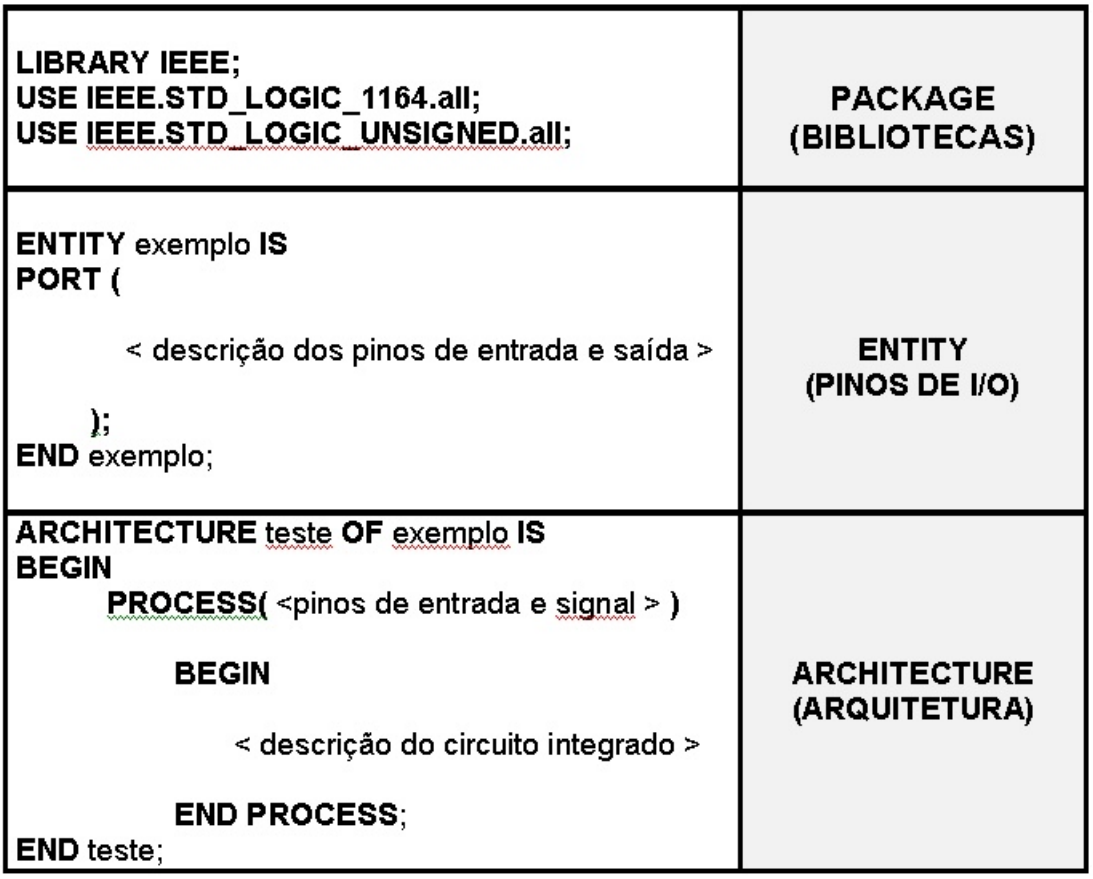

Figura 1: Estrutura de um Programa em VHDL

2.1 ENTITY (Entidade)

2.1.1 Tipos de Portas:

- **IN** - Porta de entrada. Não pode receber atribuição de valor dentro do programa;
- **OUT** - Porta de saída. Não pode ser utilizado como entrada para outro circuito;
- **BUFFER** - Porta de saída que pode ser atualizada por mais de uma fonte;
- **LINKAGE** - O valor da porta pode ser lido e atualizado.

Exemplo:

```
Entity <nome_da_entity> is
port
(
  sinal_controle : in <tipo>;
  parcela_1 : in <tipo>;
  parcela_2 : out <tipo>
);
end <nome_da_entity>;
```

2.1.2 Tipos de Sinais

<tipo> => bit, bit_vector, std_logic, std_logic_vector, boolean, real.

- **BIT** - Assume valor '0' ou '1'. Ex: x : in bit;
- **BIT_VECTOR** - vetor de bits. Ex: x : in bit_vector(7 downto 0); ou x : in bit_vector(0 to 7);
- **STD_LOGIC** - Assume os valores:
  - 'U': Não inicializado.
  - 'X': Sem valor determinado (Don't Care).
  - '0': Nível lógico 0.
  - '1': Nível lógico 1.
  - 'Z': Alta Impedância (TRI-STATE).
  - 'W': Sinal fraco, não pode-se afirmar se deve ser 0 ou 1.

  - 'L': Sinal fraco que provavelmente irá para 0.
  - 'H': Sinal fraco que provavelmente irá para 1.
  - '-': Don't care.

- **STD_LOGIC_VECTOR** - Vetor de bits. Ex: x : in std_logic_vector(7 downto 0);
- **Integer** - Assume valores inteiros decimais. Ex: x : integer range 0 to 100;
- **Boolean** - Assume valores true ou false (verdadeiro ou falso). Ex: x : out boolean;
- **Real** - Assume valores decimais em ponto flutuante, sempre com ponto decimal. Ex.: -3.2, 4.56, 6.0, -2.3E+2.

2.1.3 Intervalos

Permite determinar um intervalo de utilização dentro de um determinado tipo.

range <valor_menor> **to** <valor_maior>

range <valor_maior> **downto** <valor_menor>

3 Tipos de Dados

3.1 Constantes e Variáveis

Ambos os tipos constantes e variáveis devem ser definidas antes de serem utilizadas na descrição VHDL. Os elementos do tipo constante não podem ser sobrescritos, enquanto que os do tipo variável podem ser alterados a qualquer momento.

3.1.1 Constantes

A declaração de uma constante é feita através do uso da palavra-chave **constant**, como nos exemplos:

constant valor_de_pi : real := 3.141592653;

constant ativado : bit := 1;

constant atraso : time := 5ns;

3.1.2 Variáveis

A declaração de variáveis segue a mesma estrutura, utilizando a palavra-chave **variable**. Uma variável pode ser iniciada com

algum valor prévio na sua declaração, o qual irá manter até que seja feita a primeira escrita.

OBS: Uma variable pode ser considerada como uma variável local, ou seja, só pode ser acessado por um processo local. E deve ser declarada entre o Process e o Begin

A seguir, alguns exemplos:

variable numero_de_contagens : integer := 50;

variable liga_saida1 : bit; (sem inicialização)

variable liga_saida1 : bit:='0'; (com inicialização)

variable tempo_medida : time := 0ns;

As atribuições em VHDL para constantes e variáveis são feitas com o operador **:=**

3.1.3 Sinais

Um sinal (**signal**) é utilizado para interligação de blocos em VHDL, funcionando de maneira similar a uma variável. O sinal traz porém a capacidade de permitir a ligação (ao ser ligado a uma porta de entrada e saída) ou troca de valores (quando opera com variáveis internas) entre blocos funcionais distintos.

As atribuições de sinais em VHDL é feita normalmente com o

operador <=

A seguir, alguns exemplos:

signal numero_de_contagens : integer := 50;

signal liga_saida1 : bit; (sem inicialização)

signal liga_saida1 : bit:='0'; (com inicialização)

signal tempo_medida : time := 0ns;

OBS: Um signal pode ser considerado como uma variável global, ou seja, todos os processos podem acessar esta variável. E deve ser declarada entre o Architecture e o Begin

3.1.4 Exemplos de Atribuições

- Para atribuições de **bit ou std_logic** usa-se o valor entre 'aspa simples'.

  Exemplos:

  signal x : bit;

  variable y: std_logic;

  No programa a atribuição fica:

  x<='1'; (por exemplo)

  y:='1'; (por exemplo)

- Para atribuições de **bit_vector ou std_logic_vector** usa-se

o valor entre ”aspa dupla”.

Exemplos:

signal x : bit_vector(3 downto 0);

variable y: std_logic_vector(1 downto 0);

No programa a atribuição fica:

x<=”1010”; (por exemplo)

y:=”11”; (por exemplo)

- Para atribuições de **Números Inteiros** usa-se o valor sem aspas nenhuma.

  Exemplos:

  signal x : integer range 0 to 100;

  variable y: integer range 2 to 5;

  No programa a atribuição fica:

  x<=99; (por exemplo)

  y:=4; (por exemplo)

3.1.5 Tipo Escalares

Variáveis tipo escalares são aquelas cujos valores são indivisíveis.

3.1.6 Tipo Inteiros

Variáveis do tipo inteiro podem variar de -2147483647 a +2147483647, por serem representadas por 32 bits. Algumas vezes entretanto é útil definir-se novas faixas de operação para algumas variáveis. VHDL permite a definição destas faixas a partir das estruturas **is range/to ou is range/downto**, usadas para variações ascendentes e descendentes respectivamente.

Algumas destas estruturas pode ser observadas nos exemplos:

type dia_da_semana is range 1 to 31;

type valor_contagem is range 10 downto 0;

Em uma declaração de tipo, o valor inicial de uma variável (caso não atribuído) será igual ao valor mais a esquerda da faixa definida. Por exemplo uma variável do tipo dia_da_semana teria valor inicial 1 enquanto que uma do tipo valor_contagem teria 10.

3.1.7 Tipo Ponto Flutuante

Variáveis do tipo ponto flutuante são compostas por duas partes: uma mantissa e uma parte exponencial. Existe um tipo ponto flutuante pré-definido em VHDL que é o tipo real, que varia de -1.0E+38 até +1.0E38, com pelo menos seis dígitos de precisão. É possível a definição de outras variáveis do ponto flutuante conforme a necessidade da aplicação. Alguns exemplos são apresentados a

seguir:

type valor_leitura is range -5.0 to 5.0;

type porcentagem is range 0.0 to 100.0;

3.1.8 Tipos Físicos

Variáveis do tipo físico (physical) são utilizadas na representação de quantidades físicas do mundo real, tais como comprimento, massa, tempo e tensão. Na definição destas variáveis é possível a atribuição das unidades respectivas através do uso da palavra-chave units. Abaixo apresenta-se um exemplo de uma medida física de resistores em VHDL:

```
type resistencia is range 0 to 1E9
units
  ohm;
  kohm = 1000 ohm;
  Mohm = 1000 kohm;
end units resistencia;
```

Existe um tipo físico muito importante pré-definido em VHDL que é o tipo time, usado para operações com tempo. A declaração deste tipo pode ser observada a seguir:

```
type time is range implementation_defined
units
  fs;
  ps = 1000 fs;
  ns = 1000 ps;
  us = 1000 ns;
  ms = 1000 us;
  sec = 1000 ms;
  min = 60 sec;
  hr = 60 min;
end units;
```

3.1.9 Tipo Enumeração

O tipo especial de enumeração nada mais é do que uma lista ordenada de possíveis valores que uma determinada variável pode conter. Por exemplo a variável display definida por:

type display is ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');

Pode assumir qualquer um dos 10 possíveis valores '0' a '9'. Se por acaso esta variável devesse representar alguns valores hexadecimais, deveria ser definida como:

type display is ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F');

3.1.10 Tipos Compostos

Variáveis de tipo composto são montadas com um agrupamento de variáveis de tipos já conhecidos.

Arrays (Vetores): De uma forma geral, arrays são conjuntos de valores de um determinado tipo. O agrupamento de determinado tipo em um conjunto é especialmente importante para operações com tratamento de memória ou manipulação de matrizes. A declaração de um array deve obrigatoriamente conter além do nome do array, o tipo de variável (ou variáveis) que deverá comportar e normalmente a faixa de validade (dimensão) do array. A forma mais simples de compreender a forma de declaração de arrays é através de exemplos práticos. Na declaração abaixo, por exemplo:

type matriz_entrada is array (0 to 9) of integer;

Está-se definindo uma matriz de 10 elementos de tipo inteiro (integer) chamada matriz_entrada. Assim como em outras estruturas VHDL é possível o uso da palavra chave downto para ordenação descendente, o que, em alguns casos, é bem útil. Por ser um arranjo mais genérico a declaração de arrays pode conter também valores não numéricos, como no exemplo a seguir:

type dias_semana is (DOM, SEG, TER, QUA, QUI, SEX, SAB);

type dias_uteis is array (SEG to SEX) of dias_semana;

Onde se define um array chamado dias_uteis que contém somente um conjunto dos cinco dias da semana de SEG a SEX a partir do conjunto dias_semana. Em muitos casos arrays de uma única dimensão não são suficientes para as operações que se deseja. VHDL permite a definição de arrays multi-dimensionais de forma facilitada. Por exemplo, pode-se definir o array que representa uma imagem bidimensional de 10x20 da seguinte forma:

type imagem is array (1 to 10, 1 to 20) of integer;

Note que, no exemplo, imagem inicia da posição 1. Não é necessário se iniciar sempre do valor 0. Da mesma forma não há a obrigação de que os tipos definidos para cada dimensão de um array sejam iguais. É possível por isso construir-se arrays multidimensionais de tipos distintos como no exemplo:

type estado is (ativado, desativado, falha);

type conjunto_processos is array (0 to 10, estado) of bit;

Que define uma matriz de bits, cujas dimensões são dadas por tipos distintos. VHDL permite ainda a construção de arrays sem definição de limites através dos símbolos <>, como pode ser visto a seguir:

type matriz is array (natural range <>) of integer;

Que define um array chamado matriz de inteiros, cuja dimensão não foi especificada, mas que deve estar contida dentro do espaço de números naturais (natural). Normalmente o tamanho de um array deste tipo é definido quando se atribui no programa uma constante ou variável que irá utilizar este tipo de array. Por exemplo:

variable matriz1 : matriz := (0.0, 0.0, 0.0, 0.0, 0.0);

Define uma variável do tipo matriz (sem definição de limite) chamada matriz1 que tem 5 posições. A atribuição de valores a arrays merecem alguns comentários, pois em VHDL são disponibilizados alguns formatos especiais. A atribuição mais simples já foi mostrada e trata do preenchimento direto dos valores de um array ou de um elemento, como na listagem a seguir:

```
type medidor is array (1 to 3) of integer := (0.0, 0.0, 0.0);

medidor(1) := entrada1;

medidor(2) := entrada2;

medidor(3) := (entrada1 + entrada2)/2;
```

A primeira linha permite o preenchimento de todo array com valores iniciais. As demais linhas acessam cada uma das posições colocando os valores apropriados, respectivamente. Para arrays muito grandes entretanto alguns recursos são especialmente úteis. Por exemplo para o caso a seguir:

```
type 7_segm is (1 to 7) of bit;
```

variable display1: 7_segm := (1 =>1, 2 =>0, 3 =>0, 4 =>0, 5 =>0, 6 =>1,7 =>1);

variable display2: 7_segm := (1 =>1, 2 to 5 => 0; 6 to 7 =>1);

variable display3: 7_segm := (1 | 6 | 7 => 1, 2 to 5 => 0);

variable display4: 7_segm := (1 | 6 | 7 => 1, others => 0);

3.1.11 Cometário em VHDL

O comentário em VHDL é dado por **- -** na frente do comentário.

4 Operadores e Expressões

4.1 Operadores

4.1.1 Operadores Lógicos

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
and	Lógica E	bit, boolean ou array(bit,boolean)	mesmo do anterior	boolean
or	Lógica OR	bit, boolean array(bit,boolean)	mesmo do anterior	boolean
nand	Lógica E negada	bit, boolean array(bit,boolean)	mesmo do anterior	boolean
nor	Lógica OR Negada	bit, boolean array(bit,boolean)	mesmo do anterior	boolean
xor	Lógica OR exclusivo	bit, boolean array(bit,boolean)	mesmo do anterior	boolean
xnor	Lógica XOR negada	bit, boolean array(bit,boolean)	mesmo do anterior	boolean

Figura 2: Tabela de Operadores Lógicos em VHDL.

4.1.2 Operadores Numéricos

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
+	Adição	numérico	mesmo do anterior	igual aos operandos
-	Subtração	numérico	mesmo do anterior	igual aos operandos
&	Concatenação	array 1 dimensão	mesmo do anterior	igual aos operandos
*	Multiplicação	inteiro ou ponto flutuante	mesmo do anterior	igual aos operandos
		físico	inteiro ou ponto flutuante	físico
		inteiro ou ponto flutuante	físico	físico
/	Divisão	inteiro ou ponto flutuante	mesmo do anterior	igual aos operandos
		físico	inteiro ou ponto flutuante	físico
		físico	físico	inteiro
mod	Módulo	inteiro	mesmo do anterior	igual aos operandos
rem	Resto da divisão	inteiro	mesmo do anterior	igual aos operandos
**	Potenciação	Inteiro ou ponto flutuante	inteiro	igual ao operador da esquerda
abs	Valor absoluto		numérico	numérico
not	Negação		bit, boolean ou array (bit, boolean)	igual ao operador da direita

Figura 3: Tabela de Operadores Numéricos em VHDL.

Exemplos de uso dos operadores numéricos:

Operador	Operação	Exemplo
+	Adição	I := i + 2;
-	Subtração	J <= j -10;
*	Multiplicação	M := fator * n;
/	Divisão	K := i / 2;
**	Potenciação	I := i ** 3;
ABS	Valor absoluto	Y <= ABS(tmp)
MOD	Módulo	Z <= MOD(t);
REM	resto	R <= REM(tot);

Figura 4: Tabela de Operadores Numéricos em VHDL.

4.1.3 Operadores Relacionais

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
=	Igualdade	qualquer	mesmo do anterior	boolean
/=	Desigualdade	qualquer	mesmo do anterior	boolean
<	Menor que	escalar ou array	mesmo do anterior	boolean
<=	Menor ou igual	escalar ou array	mesmo do anterior	boolean
>	Maior que	escalar ou array	mesmo do anterior	boolean
>=	Maior ou igual	escalar ou array	mesmo do anterior	boolean

Figura 5: Tabela de Operadores Relacionais em VHDL.

4.1.4 Operadores de Deslocamento

Operador	Operação	Tipo do operador da esquerda	Tipo do operador da direita	Tipo do resultado
sll	desloc. lógico para esquerda	array de bit ou boolean	inteiro	array de bit ou boolean
srl	desloc. lógico para direita	array de bit ou boolean	inteiro	array de bit ou boolean
sla	desl. aritmético para esquerda	array de bit ou boolean	inteiro	array de bit ou boolean
sra	desl. aritmético para direita	array de bit ou boolean	inteiro	array de bit ou boolean
rol	rotação para a esquerda	array de bit ou boolean	inteiro	array de bit ou boolean
ror	rotação para a direita	array de bit ou boolean	inteiro	array de bit ou boolean

Figura 6: Tabela de Operadores de Deslocamento em VHDL.

Exemplos:

sll - shift left logical 01101001 => 11010010 (acrescenta 0 no bit - sign.)

srl - shift right logical 01101001 => 00110100 (acrescenta 0 no bit + sign.)

sla - shift left arithmetic 01101001 => 11010011 (repete o bit - sign.)

sra - shift right arithmetic 01101001 => 00110100 (repete o bit + sign.)

rol - rotate left logical 01101001 => 11010010 (desloca todos bits para esq.)

ror - rotate right logical 01101001 => 10110100 (desloca todos

bits para dir.)

4.1.5 Operadores de Concatenação

Esta operação consiste em criar um novo vetor a partir de dois vetores já existentes, por exemplo:

dado1 : bit_vector(0 to 7);

[01011011]

dado2 : bit_vector(0 to 7);

[11010010]

novo_dado : bit_vector(0 to 7);

novo_dado <= (dado1(0 to 1) & dado2(2 to 5) & dado1(6 to 7));

[01010011]

4.2 Representação de Números e Caracteres

4.2.1 Números Inteiros

VHDL define dois tipos básicos de números: inteiros e reais. Um número inteiro consiste de um número sem ponto decimal.

Exemplos de números inteiros:

23 0 146

4.2.2 Números Reais

Os números reais, por sua vez, permitem a representação de números fracionários. Por exemplo:

23.1 0.2 34.118

Pode-se trabalhar com notação exponencial, desde que o número seja seguido pela letra 'E' ou 'e'.

Exemplos: 3E2 18E-6 3.229e9

4.2.3 Bases

Além disso, pode-se trabalhar em VHDL com bases diferentes da decimal, bastando para isto indicar a base seguido pelo número entre '#'. Para bases maiores que 10, deve-se utilizar as letras 'A' a 'F' (ou 'a' a 'f'), indicando as bases 11 a 15. Não são permitidas bases maiores que 16. Como exemplo, tem-se o número 253 representado em diversas bases distintas:

2#11111101# - Representação base 2 (binária).

16#FD# - Representação base 16 (hexadecimal).

16#0fd# - Representação base 16 (hexadecimal).

8#0375# - Representação base 8 (octal).

A fim de facilitar a interpretação de números longos é permitido em VHDL utilizar-se o caractere underline ('_') entre dígi-

tos. Isto não altera o valor representado pelo número, apenas facilita a sua identificação visual. Por exemplo: 123_456 3.141_592 2#111_1100_0000_0000#

4.2.4 Caracteres

A representação de caracteres em VHDL é permitida pelo uso de aspas simples como nos exemplos: **'A' 'b' ';' '0'**

4.2.5 Strings

As strings, em VHDL, são definidas entre aspas duplas e representam uma seqüência de caracteres, mas precisam estar inteiramente definidas em uma linha.

Exemplos de strings:

"Teste de operacao"

"2000 iteracoes"

4.2.6 Strings de Bit

Assim como strings de caracteres, é possível a definição de strings de bits ou valores numéricos especiais (como representados por outras bases). O especificador de base pode ser:

-B para binário

-O para octal

-X para hexadecimal

A seguir alguns exemplos de strings de bit:

Base binária:

B”010001”

b”1001_1101_1100”

Base octal:

o”372”

O”740”

Base hexadecimal:

X”D4”

x”0F2”

5 Estruturas e Comandos em VHDL

5.1 Comando IF-THEN-ELSIF-ELSE

Existem inúmeras formas de se utilizar if-then-else(elsif), abaixo segue um quadro explicativo do if-then-else e ao lado dois exemplos simples. O próximo quadro apresenta um exemplo usando este comando.

MODELO:

```
if condição_1 then
  <comandos>
elsif condição_2 then
  <comandos>
else
  <comandos>
end if;
```

EXEMPLO 1:

```
if ( A = '0') then
```

```
  B <= "00";
else
  B <= "11";
end if;
```

EXEMPLO 2:

```
if (CLK'event and CLK ='1') then
  FF <= '0';
elsif (J='0') then
  FF <= '1';
elsif (J = '1') and (K='1') then
  XX <= '1';
end if;
```

Exercício 1: Implemente um programa em VHDL que seja capaz de detectar se dois sensores foram acionados (em nível lógico 1), caso sim acione um led (nível lógico 1) caso não apague o led (nível lógico 0).

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY detecta IS
PORT(
```

```
  led : out bit;
  sensor1,sensor2 : in bit
);
END detecta;

ARCHITECTURE arq_detecta OF detecta IS
begin

process(sensor1,sensor2)
begin
  if(sensor1='1' and sensor2='1')then
   led<='1';
  else
   led<='0';
  end if;
end process;

end arq_detecta;
```

Exercício 2: Implemente um MUX 8:1 em VHDL.

Exercício 3: Implemente um decodificador binário - 7 segmentos que mostre em um display ânodo comum os números de 0 à 7. Os números são mostrados no display de acordo com a

mudança nos valores de 3 chaves de entrada que correspondem ao valor binário.

Exercício 4: Implemente uma ULA (unidade lógica e aritmética) de 4 operações entre dados de 2 bits sendo as operações soma, subtração, and e or.

5.2 Comando WAIT

Este comando tem a finalidade de causar uma suspensão do processo declarado ou procedimento. O comando wait pode ser utilizado de quatro formas diferentes, são elas:

- wait until <conditional>; wait until CLK'event and CLK = '1';
- wait on <signal_list>; wait on a, b;
- wait for <time>; wait for 10ns;
- wait; wait;

Interpretação da tabela acima: Na linha 1, o comando wait until implica que a operação está suspensa até (until) que haja uma transição ('event) e (and) o CLK seja igual a 1, ou seja, está aguardando uma borda de subida;

Na linha 2, onde a sequência de operação é suspensa até que haja uma variação em qualquer dos sinais a ou b.

Na linha 3, suspende a operação por 10ns;

Na linha 4, suspende a operação.

5.3 Comando CASE

O comando case seleciona a execução que ocorrerá de uma lista de alternativas.

MODELO:

```
case variave is
  when valor_variavel1 => <comandos>
  when valor_variavel2 => <comandos>
  when valor_variaveln => <comandos>
  when others => <comandos>
end case;
```

EXEMPLO 1:

```
case entrada is
  when "000" => saida  <=  "010";
  when "001" => saida  <=  "100";
  when "010" => saida  <=  "001";
  when "011" => saida  <=  "011";
  when "100" => saida  <=  "111";
```

```
  when "101" => saida  <=  "101";
  when others => saida <=  "000";
end case;
```

Exercício 1: Implemente um decodificador binário - 7 segmentos que mostre em um display ânodo comum os números de 0 à 7. Os números são mostrados no display de acordo com a mudança nos valores de 3 chaves de entrada que correspondem ao valor binário.

5.4 Comando WHILE-LOOP

A estrutura while-loop é um caso especial de estrutura de loop, onde uma condição é testada inicialmente antes de qualquer operação do loop e caso seja verdadeira, o loop é executado. Se a condição não for satisfeita, o loop pára de ser executado, seguindo-se além deste.

Um exemplo desta estrutura é apresentada a seguir:

```
numero := valor_entrada;
raiz_quadrada := 0;
while raiz_quadrada*raiz_quadrada < numero loop
  raiz_quadrada := raiz_quadrada + 0.1;
end loop;
```

onde o cálculo simples de uma raiz quadrada é feito até que se encontre o valor mais próximo. Note que se o valor de entrada for zero, a condição do loop não é satisfeita e o mesmo não é executado, resultando no valor raiz_quadrada := 0 que é o valor correto.

5.5 Comando WHILE-FOR

O uso de estrutura de for-loop é utilizado principalmente para execução de operações a serem repetidas por um número determinado de vezes. Por exemplo a sequência abaixo descreve o cálculo do fatorial do número 10.

```
resultado := 1;
for n in 1 to 10 loop
  resultado := resultado *n;
end loop;
```

A estrutura for-loop permite também a realização de iterações em ordem decrescente. Neste caso basta utilizar-se a palavra-chave downto no lugar de to, como em:

```
contador := 5;
resultado := 0;
for contador in 10 downto 1 loop
  resultado := contador;
end loop;
```

5.6 Comando EXIT

O comando EXIT serve para forçar a saída de um laço conforme exemplos abaixo:

exit; => utilizado para terminar um while, for;

exit <label_loop>; => termina o laço e desloca para o label;

exit <label_loop> when <condição>; => termina o laço quando (when) condição é satisfeita;

5.6.1 Estrutura Loop com Saída Forçada por Exit

Um estrutura LOOP pode entretanto precisar de uma condição de saída. Isto é previsto em VHDL pela palavra-chave exit, que força a saída do loop quando uma determinada condição for atendida. Por exemplo:

```
loop
  loop
    exit when reset_contador = '1';
    contador := contador +1;
    saida := contador;
  end loop;
  contador := 0;
end loop;
```

A estrutura acima implementa um contador que reseta sua contagem se a condição reset_contador = '1' for verdadeira. Note que é possível o encadeamento de loops, ou seja a montagem de um loop dentro de outro. A linguagem VHDL permite a atribuição de nomes a loops e a seleção de qual saída será forçada a partir destes nomes.

Um exemplo de uso de estrutura for com diversos loops encadeados é descrito a seguir:

```
loop_externo : loop
  contador := 0;
  loop_intermediario : loop
    loop_interno : loop
      saida := contador;
      exit when contagem_habilitada = '0';
      exit loop_intermediario when reset_contador = '1';
    end loop loop_interno;
    contador := contador +1;
  end loop loop_intermediario;
end loop loop_externo;
```

Onde existem duas condições de saída. A primeira testa se a contagem está desabilitada (contagem_habilitada = '0'), quando então sai do loop_interno para proceder uma nova contagem. A

segunda condição de saída testa se reset_contador = '1' e caso for, salta para fora do loop_intermediario, resetando o contador.

5.7 Comando NEXT

O comando NEXT é utilizado para terminar prematuramente execução de uma iteração do tipo while, for ou um loop infinito.

5.7.1 Estrutura Loop com Reinício Forçado por Next

Esta estrutura é similar ao caso de saída anterior, só que a palavra-chave next serve para reiniciar a sequência de operações do loop. Na descrição a seguir tem-se um monitor de temperatura, que só executa as medidas de emergência (aquecedor:= OFF e alarme:=ON) se a temperatura medida exceder o limite. Note que com o uso de next a execução não sai fora do loop

```
loop
  medida_temperatura = entrada_medidor;
  next when medida_temperatura < temperatura_maxima;
  aquecedor := OFF;
  alarme := ON;
end loop;
```

Assim como no caso do exit, também é possível com next, a definição de qual loop deve ser iniciado quando se estiver trabal-

hando com loops encadeados. O formato de utilização é o seguinte:

...

next nome_loop when condicao = true

...

6 Funções e Procedimentos

6.1 Funções em VHDL

Uma função (function) em VHDL permite o retorno de um resultado na própria linha de execução.

Este tipo de recurso é bastante útil para uso em expressões.

Modelo:

```
function nome_da_function(parâmetros de entrada)
return tipo_da_variável is
--declaração de variáveis aqui
begin
  <comandos>
  return nome_da_variável_de retorno;
end nome_da_function;
```

Exemplo de chamada da function:

x<=nome_da_function(y);

No exemplo a seguir, tem-se implementada uma função que re-

aliza a conversão de um número em formato BCD para decimal.

```
function bcd_para_decimal (valor_bcd : integer 0 to 255)
return integer is
  variable resultado : integer range 0 to 99;
  variable valor_temp : integer range 0 to 255;
  variable nibble: integer range 0 to 15;
  begin
  nibble := valor_bcd;
  valor_temp := (dado - nibble)/2;
  if valor_bcd > 9 then
    resultado := valor_temp + valor_temp/4 + nibble;
  else
    resultado := valor_bcd;
  end if;
  return resultado;
end function bcd_para_decimal;
```

Exemplo de chamada da function acima:

...

if bcd_para_decimal(valor_entrada) > valor_limite then

atuador <= OFF;

else

atuador <= ON;

end if;

...

A função é usada para ajustar o formato de uma variável chamada valor_entrada, retornando na própria linha de chamada o valor da conversão. Note que a variável (ou variáveis) de entrada de uma função não precisa conter o modo (como in ou inout), uma vez que isto é subentendido. A variável de saída é declarada fora dos parênteses e não precisa identificador, apenas o tipo.

6.2 Procedimentos em VHDL

Um procedimento (procedure) em VHDL é um conjunto de operações que executa uma determinada finalidade. Este procedimento pode ser chamado de qualquer parte do programa VHDL, de forma similar à chamada de funções feita em várias outras linguagens. Para a chamada de um procedimento em uma descrição VHDL basta escrever-se o nome do procedimento seguido de ponto-e-vírgula (;).

Modelo:

```
procedure nome_procedure is
begin
  <comandos>
end nome_procedure;
```

Por exemplo:

```
procedure contagem is
begin
  if(contador<=100)then
    contador <= contador +1;
  else
    contador<=0;
  end if;
end contagem;
```

...

contagem; – chamada do procedimento de atraso

... Implementa um procedimento de contagem até 100. Note que contador deve ser uma variável global (SIGNAL).

6.2.1 Procedimentos com Parâmetros em VHDL

Para ser mais completa, a utilização de procedimentos em VHDL deveria permitir a passagem de parâmetros quando de sua chamada. Isto é também previsto pela linguagem. Para tanto deve-se inicialmente prever na declaração do procedimento o parâmetro (ou parâmetros) que poderá ser passado, descrevendo-se o mesmo entre parênteses. Na sua descrição, o parâmetro é composto por um identificador, um modo (in, out ou inout) e o tipo. Algumas

vezes utiliza-se a palavra-chave func_code na especificação de tipo, quando deseja-se que o tipo do parâmetro seja dado pelo programa que chama o procedimento.

No exemplo a seguir tem-se uma descrição do uso de um procedimento de envio, por um pino serial chamado dout, de um caractere (dado) passado como parâmetro:

```
procedure envia_serial (dado : in func_code) is
variable indice : dado_serial := 0;
variable teste : positive := 0;
variable temp : bit := start_bit ;
begin
  if (clock='1') then
    if teste >1 and teste <(tamanho_dado_serial + 1)then
      teste := teste +1;
      if indice <= tamanho_dado_serial then
        temp := dado(indice);
        indice := indice +1;
      end if;
     end if;
  end if;
  if teste = (tamanho_dado_serial +1) then
    temp := stop_bit;
    dout <= temp;
```

```
    return;
  end if
  dout <= temp;
end procedure envia_serial;
```

Ex: de Chamada da procedure:

dado := 34;

envia_serial(dado);

...

6.2.2 Procedimentos com Parâmetros e Retornos em VHDL

Para ilustrar que procedimentos também poder retornar valores a seguir apresenta-se um exemplo de um algoritmo para conversão de valor decimal para BCD:

```
procedure converte_para_BCD (numero: in integer range 0
to 39; mais_sig, menos_sig : out integer range 0 to 9) is
begin
  if numero > 39 then return;
  elsif numero >= 30 then
    mais_sig := 3;
    menos_sig := numero - 30;
  elsif numero >= 20 then
    mais_sig := 2;
```

```
    menos_sig := numero - 20;
  elsif numero >= 10 then
    mais_sig := 1;
    menos_sig := numero - 10;
  else mais_sig := 0;
    menos_sig := numero;
  end if;
end procedure converte_para_BCD;
```

Ex: de Chamada da procedure:

converte_para_BCD (dado_lido, dezena, unidade);

Observe que é possível ter mais de um ponto de saída de um procedimento. No modelo apresentado, por exemplo, se o valor da variável for maior que 39 o procedimento é abortado. Finalizações de procedimentos são possíveis através da palavra-chave return.

Além da passagem de valores como parâmetros, um procedimento permite a passagem de sinais, o que na verdade se constitui pela ligação de um pino de entrada (ou saída) de um módulo com a interface do procedimento que está sendo chamado. Assim, se fosse desejado, por exemplo, para o procedimento apresentado anteriormente deixar a atribuição do pino de saída serial a critério do programa que fosse utilizá-lo, poderia-se alterar a declaração do procedimento para:

procedure envia_serial (dado : in func_code signal dout: out bit)
is

...

7 Conversão de Tipos

A conversão, ou seja, a transformação de um tipo de variável para outro é permitido em VHDL. Para isto, temos as seguintes funções de conversão de dados:

7.1 STD_LOGIC para BIT

function to_bit (s : std_ulogic) return bit;

Ex:

variable x : bit;

variable y : std_logic;

x := to_bit(y);

7.2 STD_LOGIC_VECTOR para BIT_VECTOR

function to_bitvector (s : std_ulogic_vector) return bit_vector;

Ex:

variable x : bit_vector(3 downto 0);

variable y : std_logic_vector(3 downto 0);

x := to_bitvector(y);

7.3 BIT_VECTOR para STD_LOGIC_VECTOR

function to_stdlogicvector (b : bit_vector) return std_logic_vector;

Ex:

variable x : bit_vector(3 downto 0);

variable y : std_logic_vector(3 downto 0);

y := to_stdlogicvector(x);

7.4 PARA INTEGER

function conv_integer(arg: integer) return integer;

function conv_integer(arg: unsigned) return integer;

function conv_integer(arg: signed) return integer;

function conv_integer(arg: std_ulogic) return small_int;

Exemplos:

signal b : std_logic;

signal u1 : unsigned (3 downto 0);

signal s1 : signed (3 downto 0);

signal i1, i2, i3 : integer;

u1 <= ”1001”;

s1 <= ”1001”;

b <= ’X’;

i1 <= conv_integer(u1); – 9

i2 <= conv_integer(s1); – -7

i3 <= conv_integer(b);

7.5 PARA STD_LOGIC_VECTOR

function conv_std_logic_vector(arg: integer, size: integer) return std_logic_vector;

function conv_std_logic_vector(arg: unsigned, size: integer) return std_logic_vector;

function conv_std_logic_vector(arg: signed, size: integer) return std_logic_vector;

function conv_std_logic_vector(arg: std_ulogic, size: integer) return std_logic_vector;

Exemplos:

signal u1 : unsigned (3 downto 0);

signal s1 : signed (3 downto 0);

signal v1, v2, v3, v4 : std_logic_vector (3 downto 0);

signal i1, i2 : integer;

u1 <= ”1101”;

s1 <= ”1101”;

i1 <= 13;

i2 <= -3;

v1 <= conv_std_logic_vector(u1, 4); – = ”1101”

v2 <= conv_std_logic_vector(s1, 4); – = ”1101”

v3 <= conv_std_logic_vector(i1, 4); – = ”1101”

v4 <= conv_std_logic_vector(i2, 4); – = ”1101”

8 Bibliografia

1. Apostila VHDL Prof. Ronaldo Hüsemann - UFRGS

2. Apostila VHDL Prof. Anderson Terroso - PUCRS

3. Livro VHDL Descrição e Síntese de Circuitos Digitais - Roberto D'Amore

4. Livro Eletrônica Digital Curso Prático - Ricardo Zelenovsky / Alexandre Mendonça

5. Livro Sistemas Digitais - Pricípios e Aplicações - Ronald J. Tocci