

1. VHDL

As Linguagens de Descrição de Hardware descrevem um sistema digital através de um arquivo de texto. Esta descrição é um modelo do sistema que será executado através da utilização de uma ferramenta (software) EDA (Electronic Design Automation). O projetista cria um arquivo de texto, seguindo certo conjunto de regras (sintaxe da linguagem), e usa um compilador para criar dados de programação do dispositivo lógico programável (PLD). Um sistema descrito em linguagem de hardware pode ser implementado em um dispositivo programável permitindo assim o uso em campo do seu sistema, tendo a grande vantagem da alteração do código a qualquer momento.

Existem várias linguagens: VERILOG, Handel-C, SDL, ISP, ABEL, etc. O IEEE (Institute of Electrical and Electronic Engineers) padronizou a linguagem VHDL, cujo significado é:

V – Very High Speed Integrated Circuit (circuitos integrados de altíssima velocidade).

HDL – Hardware Description Language (linguagem de descrição de hardware).

A linguagem VHDL foi desenvolvida pelo Departamento de Defesa dos Estados Unidos por volta de 1980. Apresenta uma descrição textual, um algoritmo, para desenvolver o circuito, sem necessidade de especificar explicitamente as ligações entre componentes. VHDL é utilizada para as tarefas de documentação, descrição, síntese, simulação, teste e verificação formal.

O uso de uma linguagem formal de descrição de hardware como o VHDL, ao invés da descrição por diagramas esquemáticos, apresenta as seguintes vantagens:

- Projeto independente da tecnologia;
- Pode-se utilizar a descrição do projeto em vários tipos de plataforma, de um simulador para outro;
- Pode-se utilizar um projeto em VHDL em diferentes projetos;
- Permite, através de simulação, verificar o comportamento do sistema digital;
- Facilidade na atualização dos projetos;
- Reduz tempo de projeto e custo;
- O objetivo do projeto fica mais claro do que na representação por esquemáticos;
- O volume de documentação diminui, já que um código bem comentado em VHDL substitui com vantagens o esquemático e a descrição funcional do sistema.

Uma desvantagem do uso do VHDL em relação ao diagrama esquemático é que o hardware gerado pela descrição VHDL não é otimizado.

1.1. Características do VHDL

- Suporta projetos com múltiplos níveis de hierarquias. Portanto, a descrição geral do circuito pode consistir na interligação de outras descrições menores.
- Favorece projeto “top-down”, onde projetos complexos partem de um nível de especificação mais elevado para um mais baixo.
- 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);

. Nível de portas lógicas (Gate Level).

- Pode mesclar diferentes níveis de abstração em um mesmo código.
- Permite o uso de comandos executados concorrentemente (com exceção de regiões específicas no código), assim como os elementos de um sistema digital executam tarefas simultaneamente.
- Possibilita delimitar regiões de código sequencial (subprogramas e processos) onde a execução dos comandos segue a ordem de sua apresentação no código.

1.2. Conceitos Básicos de VHDL

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

PACKAGE
ENTITY
ARCHITECTURE
CONFIGURATION

PACKAGE ou LIBRARY: conjunto de sub-programas que descrevem, elementos e componentes já programados para serem reutilizados.

ENTITY (Entidade): define as portas de entradas e saídas dos circuitos na descrição.

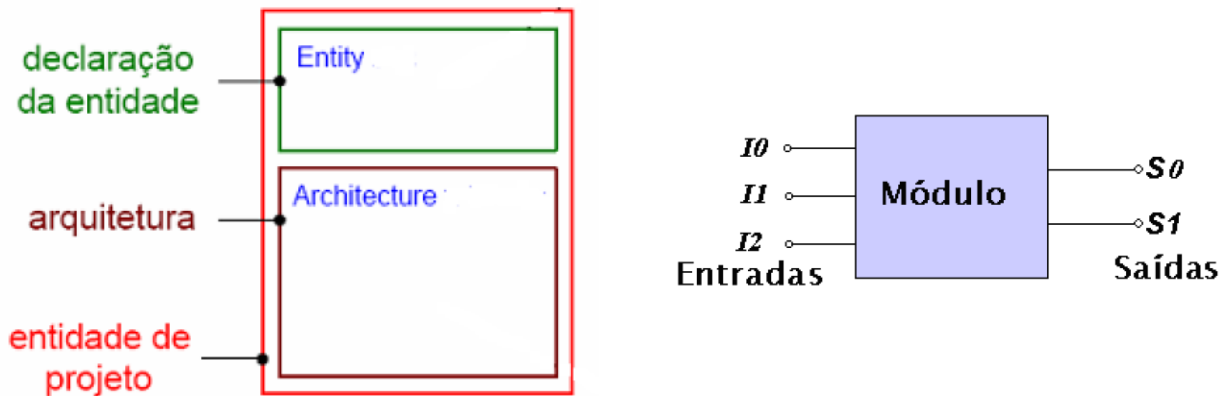
ARCHITECTURE (Arquitetura): implementações do projeto; descreve as relações entre as portas.

CONFIGURATION (Configuração): define as arquiteturas que serão utilizadas.

A estrutura de uma descrição VHDL com os seus blocos definidos é mostrada a seguir:

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

A descrição em VHDL de qualquer módulo lógico é representada pela Entidade de Projeto, como mostrado na figura seguinte. Deve ser composta de ao menos duas estruturas: entidade ("entity") e arquitetura ("architecture").



Supondo o circuito mostrado na figura acima, o bloco Entity deve informar todas as entradas e saídas, no caso, I0, I1, I2, S0 e S1. O bloco Architecture deve informar o circuito interno.

1.3. Declaração de Package ou Library

Pode ser um conjunto de sub-programas (contendo bibliotecas ou funções) que operem com um tipo de dado assim como pode ser um conjunto de declarações necessárias para modelar um determinado projeto. Os pacotes mais utilizados estão citados no quadro seguinte.

BIBLIOTECA	PACOTE	TIPOS DE DADOS	DESCRIÇÃO
IEEE	STD_LOGIC_1164	STD_LOGIC e STD_LOGIC_VECTOR	Define o padrão para descrever a interconexão entre tipos de dados usados na linguagem VHDL
IEEE	STD_LOGIC_ARITH	Especifica tipos de dados sinalizados e não sinalizados	Funções de conversão, operações aritméticas e comparações para uso de dados sinalizados e não sinalizados
IEEE	STD_LOGIC_UNSIGNED	STD_LOGIC_VECTOR	Define funções que permitem usar tipos de dados STD_LOGIC_VECTOR, como se fossem tipo de dado não sinalizado
IEEE	STD_LOGIC_SIGNED	STD_LOGIC_VECTOR	Define funções que permitem usar tipos de dados STD_LOGIC_VECTOR, como se fossem tipo de dado sinalizado
IEEE	NUMERIC_STD	Define operações aritméticas seguindo o padrão IEEE. Substitui os pacotes usados juntos STD_LOGIC_ARITH, STD_LOGIC_UNSIGNED e STD_LOGIC_SIGNED	IEEE
STD	STANDARD	BIT(0 ou 1) BIT_VECTOR(3 Downto 0); BIT_VECTOR(0 to 3); BOOLEAN(falso ou verdadeiro); INTEIRO(positivo ou negativo)	STD
STD	TEXTIO	Define os arquivos de operações.	STD
WORK	<definido pelo usuário>	Biblioteca corrente de trabalho	WORK

OBS: As bibliotecas STD e WORK não necessitam ser referenciadas no projeto VHDL, pois são assumidas automaticamente pela linguagem.

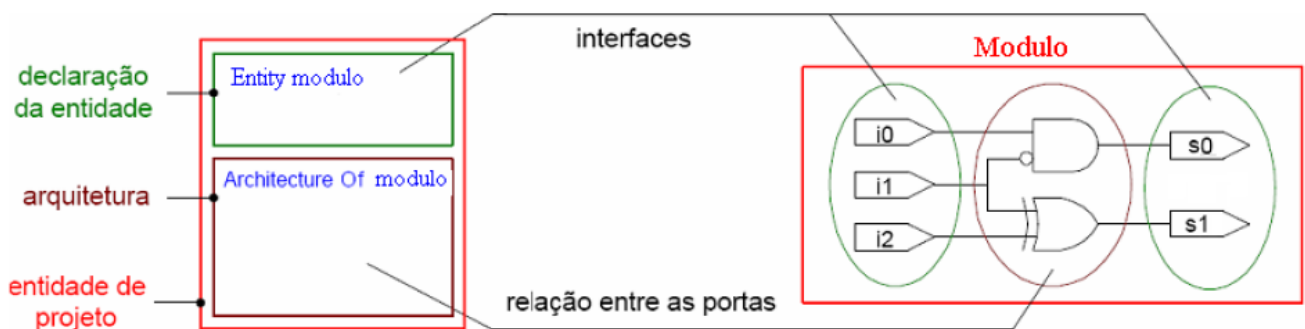
A declaração do uso de pacotes e bibliotecas é feita da seguinte forma:

```
LIBRARY <nome_da_biblioteca>
USE <nome_do_pacote_biblioteca> .ALL
```

O uso de .all implica que todos os elementos da biblioteca podem ser usados.

1.4. Declaração da Entidade

Na declaração de uma entidade descreve-se o conjunto de entradas e saídas que constituem o projeto, que é equivalente ao símbolo de um bloco em captura esquemática.



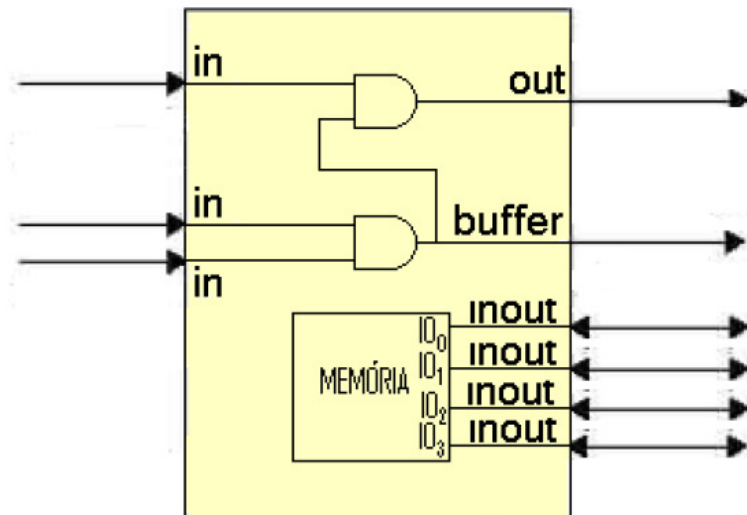
Uma declaração genérica para o circuito com entradas I0, I1, I2 e saídas S0, S1 é:

```
ENTITY modulo IS
PORT ( I0, I1, I2 :    modo_1    tipo_a;    -- entradas
        S0, S1    :    modo_2    tipo_b);    -- saídas
END modulo;
```

A palavra reservada “ENTITY” inicia a declaração, seguida do nome que a identifica e da palavra reservada IS. PORT lista as entradas e as saídas da entidade de projeto, definindo o modo e tipo das portas. END termina a declaração, devendo ser seguida do nome da Entidade de Projeto e de ponto e vírgula (;).

Existem quatro modos, representados na próxima figura:

- Modo IN : entrada.
- Modo OUT: saída.
- Modo BUFFER: saída que pode ser realimentada internamente.
- Modo INOUT: bidirecional.



Os principais tipos são:

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	Valores TRUE ou FALSE.

Uma vez que a biblioteca IEEE_1164 tenha sido incluída na listagem VHDL, utiliza-se notações diferentes para bit e bit_vector:

BIT é substituído por STD_LOGIC

BIT_VECTOR é substituído por STD_LOGIC_VECTOR

Essa mudança possibilita mudar de BIT, o qual tem apenas dois valores ('0' e '1') para o STD_LOGIC mais útil, o qual possui nove valores possíveis:

'0' = 0 forte

'1' = 1 forte

'X' = forçando desconhecido

'Z' = alta impedância (circuito aberto)

'w' = desconhecido fraco

'L' = 0 fraco

'H' = 1 fraco

'U' = inicialização desconhecida

'-' = não importa (dont care)

No caso do circuito com entradas I0, I1, I2 e saídas S0, S1, poderíamos ter a descrição:

```
ENTITY modulo IS  
PORT ( I0, I1, I2  :    IN BIT;    -- entradas  
        S0, S1      :    OUT BIT); -- saídas  
END modulo;
```

1.5. Declaração da Arquitetura

Descreve o comportamento da entidade, o seu funcionamento interno, isto é, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos.

A declaração de uma arquitetura pode conter comandos concorrentes ou sequenciais. Sua organização pode conter declaração de sinais, constantes, componentes, operadores lógicos, comandos (ex: BEGIN, END), etc. VHDL permite ter mais de uma arquitetura para a mesma entidade. Uma arquitetura consiste de duas partes: a seção de declaração da arquitetura e o corpo da arquitetura.

Um exemplo de declaração de arquitetura é:

```
ARCHITECTURE comportamento OF ent IS  
SIGNAL c_internal : small_int;  
BEGIN  
    c_internal <= a0 + b0;  
    c0 <= c_internal;  
    c1 <= c_internal + a1 + b1;  
END comportamento;
```

A seção de declaração da arquitetura ("architecture") é a área entre a chave architecture e a chave begin. Nesse espaço pode-se declarar objetos que são localizados na arquitetura. Após a seção de declaração tem-se o corpo da arquitetura o qual especifica o comportamento da arquitetura.

1.6. Palavras reservadas

A linguagem não é case-sensitive, mas frequentemente são usadas maiúsculas para as palavras reservadas. Uma palavra-chave tem um significado reservado na linguagem e não podem ser usadas para outro propósito.

Abs, acess, after, alias, all, and, architecture, array, assert, attribute.

Begin, block, body, buffer, bus.

Case, component, configuration, constant

Disconnect, downto

Else, elsif, end, entity, exit

File, for, function

Generate, generic, group, guarded

If, impure, in, inertial, inout, is

Lable, libraries, linkage, literal, loop

Map, mod

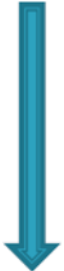
Nand, new, next, nor, not, null

Of, on, open, or, others, out
Package, port, postponed, procedure, process, pure
Range, record, register, reject, rem, report, return, rol, ror
Select, severity, shared, signal, sla, sll, sra, srl, subtype
Then, to, transport, type
Unaffected, units, until, use
Variable
Wait, when, while, with
Xor, xnor

1.7. Símbolos Definidos

Símbolo	Significado	Símbolo	Significado
+	Adição ou número positivo	:	Separação entre uma variável e o tipo
-	Subtração ou número negativo	“	Aspas dupla
/	Divisão	„	Aspas simples ou marca de tick
=	Igualdade	**	Exponenciação
<	Menor do que	=>	Seta indicando “então”
>	Maior do que	=>	Seta indicando “recebe”
&	Concatenador	:=	Associação de valor para variáveis
	Barra vertical	/=	Desigualdade
;	Terminador	>=	Maior do que ou igual a
#	Literal incluído	<=	Menor do que ou igual a
(Parêntese da esquerda	<=	Associação de valor para sinais
)	Parêntese da direita	<>	Caixa
.	Notação de Ponto	--	Comentário

1.8. Precedência de Operadores

Precedência	Classe	Operadores
Menor  Maior	Lógicos	and, or, nand, nor, xor, xnor
	Relacionais	= /= < <= > >=
	Deslocamento	Sll srl sla sra rol ror
	Adição	+ - &
	Sinal	+ -
	Multiplicação	* / mod rem
	Diversos	** Abs not
Obs: O operador "not" apresenta maior precedência		

1.9. Identificadores

Os identificadores são usados para se atribuir nomes à sinais ou processos, devem ser escolhidos de tal maneira que sejam fáceis de serem lembrados e que tenham algum significado no projeto lógico.

- Pode conter:

letras do alfabeto ('A' a 'Z' e 'a' a 'z'),

dígitos decimais('0' a '9'),

caracter underline ('_').

- Precisa começar com uma letra do alfabeto;

- Não pode terminar com um caracter underline;

- Não pode conter dois caracteres underline em sequência.

1.10. Objetos de Dados

Objetos são elementos que contém um valor armazenado. Os objetos de dados mais usados são: signals, variables, e constants.

1.10.1. Signal

Representa sinais lógicos sobre um fio no circuito, os quais interligam componentes. Um sinal não tem memória, portanto se a fonte do sinal é removida, o sinal não terá um valor. PORTS são exemplos de sinais. Podem ser declarados na entidade ou na arquitetura. Não podem ser declarados em processos, mas podem ser utilizados em seu interior.

Sintaxe:

signal identificador(es) : tipo [restrição] [:=expressão];

Exemplo:

SIGNAL cont : INTEGER range 50 DOWNT0 1;

SIGNAL ground : BIT := '0';

1.10.2. Variable

Um objeto VARIABLE lembra seu conteúdo e é usado para cálculos. São utilizadas em processos e devem ser declaradas neles. São atualizadas imediatamente e não correspondem à implementação física, como no caso dos sinais.

Sintaxe se a variável tem valor inicial:

VARIABLE nome_variavel : tipo [restrição] [:=valor_inicial];

Sintaxe se a variável não tem valor inicial:

VARIABLE nome_variavel : tipo [restrição];

1.10.3. Constant

Um objeto CONSTANT deve ser inicializado com um valor quando declarado e seu valor não pode ser mudado. Podem ser declaradas nos níveis de: package, entity, architecture e process. Valem apenas no contexto em que são declaradas.

Exemplos:

SIGNAL x: BIT

VARIABLE y: INTEGER

CONSTANT one: STD_LOGIC_VECTOR(3 DOWNT0 0) := "0001"

1.11. Tipos de Dados

Um tipo é caracterizado por um conjunto de valores que pode assumir e por um conjunto de operações que podem ser realizadas. Os tipos de objetos são divididos em: Escalares e Compostos.

A figura seguinte mostra as subdivisões dos tipos escalares e compostos. Os tipos mais usados estão marcados na figura. É importante observar que não é permitida a transferência de valores entre objetos de tipos diferentes.

