

Introdução a VHDL

Professora Luiza Maria Romeiro Codá

Livro texto: “VHDL– Descrição e Síntese de Circuitos Digitais “
Roberto D’Amore
Editora LTC

Introdução a VHDL

Aula 1

HDL – *Hardware Description Language* (Linguagem de Descrição de Hardware)

Linguagem para descrever o funcionamento de um sistema (o que e como o sistema faz).

O sistema descrito em HDL pode ser implementado em um dispositivo programável HCPLD (Dispositivo Programável de Alta Complexidade) (ex.: FPGA, CPLD).

FPGA = *Field Programmable Gate Array*

CPLD = *Complex Programmable Logic Device*

Existem dezenas de HDLs:

AHDL, Verilog, VHDL, Hendel-C, SDL, ABEL, ISP, etc.

VHDL – Introdução

VHDL é uma linguagem que possibilita o circuito eletrônico ser descrito com sentenças como uma linguagem de programação possibilitando a simulação e síntese do circuito em um dispositivo programável de alta complexidade (HCPLD)

VHSIC: *Very High Speed Integrated Circuits*

ASIC: *Application-Specific Integrated Circuit*

VHDL: *VHSIC Hardware Description Language*

VHDL – Histórico

VHDL é uma linguagem para descrever sistemas digitais padronizada pelo IEEE, criada durante o programa VHSIC do governo americano, iniciado em 1980. Teve sua origem dada pela necessidade de documentar o funcionamento de ASICs, e posteriormente foram criados simuladores e sintetizadores capazes de interpretar esta linguagem.

Resumo da versões:

- 1981: iniciada pelo Depto. De Defesa dos EUA;
- 1983–1985: Desenvolvimento da linguagem pelas empresas Intermetrics, IBM e Texas instruments;
- 1986: Direitos transferidos para o IEEE;
- 1987: publicação do padrão IEEE–(VHDL 87);
- 1994: padrão revisado (VHDL 93);
- 1996: Ferramentas comerciais para Simulação e Sínteses para o padrão IEEE 1076'93;
- 2000 : Revisão de VHDL – IEEE 1076^a;
- 2002 : Pequenas alterações;
- 2008: foi aprovado pelo REVCOM a mais recente versão, IEEE 1076–2008.

VHDL – Introdução

Algumas vantagens:

- Facilidade de atualização dos projetos
- Diferentes alternativas de implementação, permitindo vários níveis de abstração
- Verificação do comportamento do sistema digital através de simulação
- Redução do tempo de desenvolvimento e custo do projeto
- Eliminação de erros de baixo nível do projeto
- Projeto independente da tecnologia

Algumas desvantagens

- Dificuldade para otimização no *hardware* gerado
- Necessidade de treinamento para lidar com a linguagem

VHDL – Características

Favorece projeto “*Top-Down*”.

Permite descrever o sistema em diferentes níveis de abstração:

- Nível de sistema
- Nível de transferência entre registradores (RT *level*)
- Nível lógico
- Nível de circuito

Permite três diferentes estilos de descrição:

- Comportamental
- Estrutural
- Fluxo de Dados ou Físico

VHDL – Características

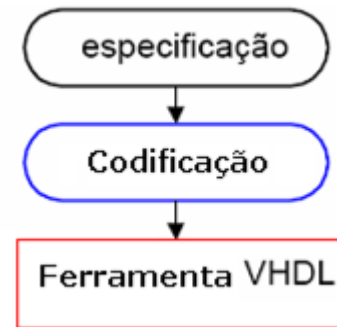
A linguagem VHDL é análoga a uma linguagem de programação.

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 do que escrever um programa que realiza a mesma função utilizando uma linguagem de programação de médio/alto nível, como C.

O código VHDL é interpretado em um simulador ou sintetizado em *hardware* (não gera código objeto).

Ciclo de Projeto



Especificação: Determinar requisitos e funcionalidades do projeto.

Codificação: Descrever em VHDL o projeto seguindo as regras de sintaxe.

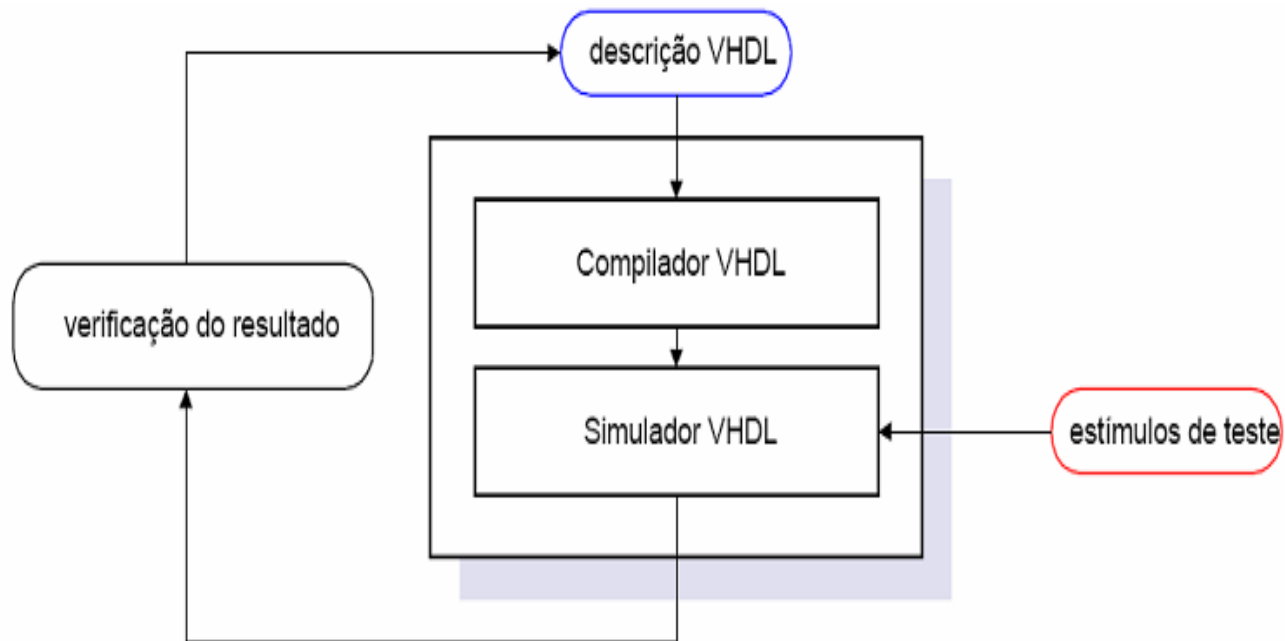
Ferramenta: Submeter a descrição a um *software* para verificar a correspondência entre especificação e código e sintetizar o circuito:

Compilação: Transforma o arquivo texto em informações sobre o circuito.

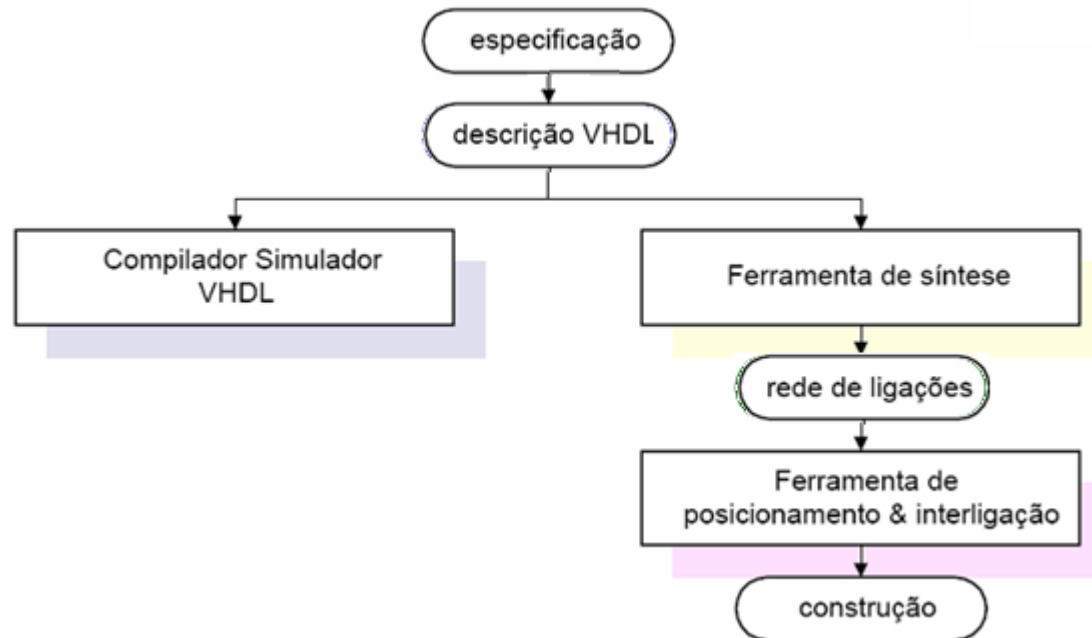
Simulação: Verificação preliminar do funcionamento do circuito.

Síntese: Configuração do circuito na tecnologia escolhida.

Ciclo de Projeto



Ciclo de Projeto



Sintaxe – Nomes e Comentários

Nomes em VHDL :

VHDL não é *case sensitive* (não distingue caracteres maiúsculos e minúsculos).

apenas letras, dígitos e sublinhados podem ser usados;

O primeiro caractere deve ser uma letra;

O último caractere não pode ser um sublinhado;

Não são permitidos dois sublinhados consecutivos.

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

Os comentários em VHDL ocorrem após dois traços “– –”.

As sentenças são terminadas por “;”.

Atribuição de valores a sinais: “<=”.

Atribuição de valores a variáveis “:=”.

Sintaxe – Palavras Reservadas

abs access after alias all and architecture array assert attribute	file for function	nand new next nor not null	then to transport type	disconnect downto	labeled libraries linkage literal loop	range record register reject rem report return rol ror	wait when while with
begin block body buffer bus	generate generic group guarded	of on open or others out	unaffected units until use	else elsif end entity exit	map mod	select severity shared signal sla sll sra srl subtype	xor xnor
case component configuration constant	if impure in inertial inout is	package port postponed procedure process pure	variable				

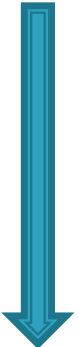
Sintaxe – Operadores

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

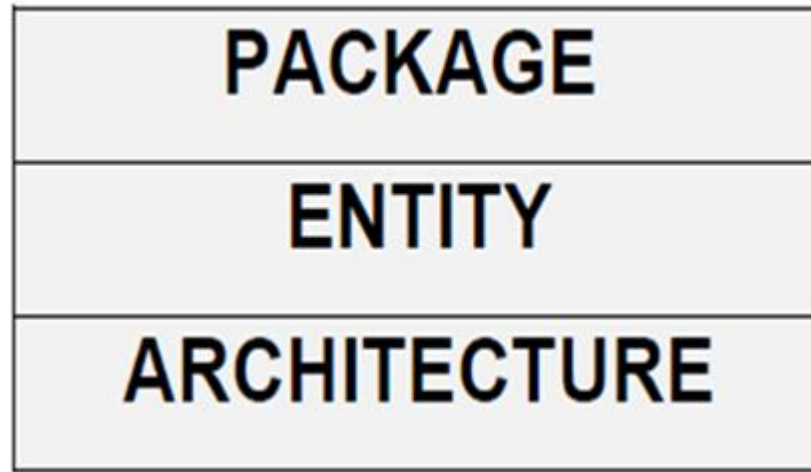
Sintaxe – Funções Lógicas

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

Sintaxe – Precedência de Operadores

Precedência	Classe	Operadores
<div>Menor</div> <div></div> <div>Maior</div>	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		

VHDL – Estrutura de uma descrição



Package (Pacote): Constantes, bibliotecas.

Entity (Entidade): Pinos de entrada e saída.

Architecture (Arquitetura): Implementações do projeto.

VHDL – Estrutura de uma descrição

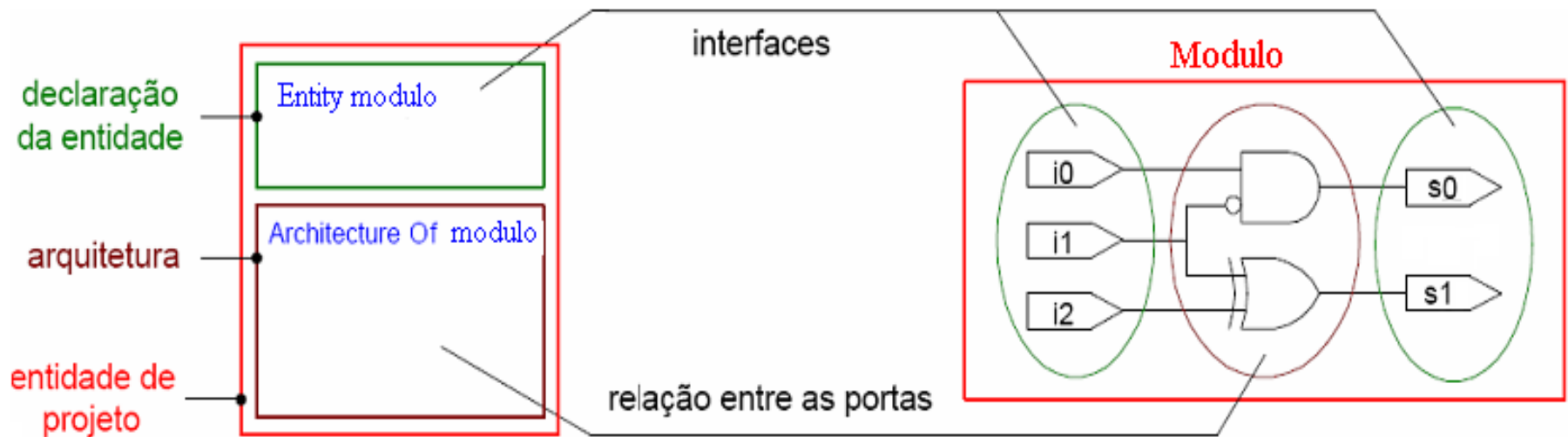
Exemplo:

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 (ARQUITETURA)

Estrutura de uma descrição VHDL –

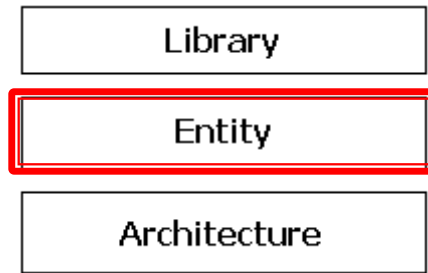
Utilizando apenas as bibliotecas da ferramenta de trabalho :

Obs: Não necessita declarar PACKAGE



ENTITY

A declaração da Entidade do Projeto define a interface entre a entidade e o meio externo, por exemplo, os pinos de entradas e saídas.



A declaração de cada pino é composta por 3 elementos:

Nome do pino

Modo de Operação

Tipo de Dados

Formato da declaração de Entidade:

```
ENTITY <nome_da_entidade> IS
    PORT(<nome> : <modo> <tipo>
        );
END <nome_da_entidade> ;
```

ENTITY

PORT: Corresponde aos pinos de entrada e saída.

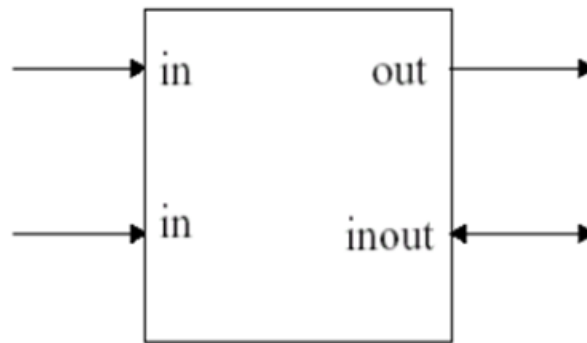
Modos de Operação:

IN: Pino de entrada.

OUT: Pino de saída. Não pode ser lido internamente pela própria Entidade.

INOUT: Pino de entrada/saída (bidirecional selecionável).

BUFFER: Pino de saída que pode ser lido internamente.



ENTITY

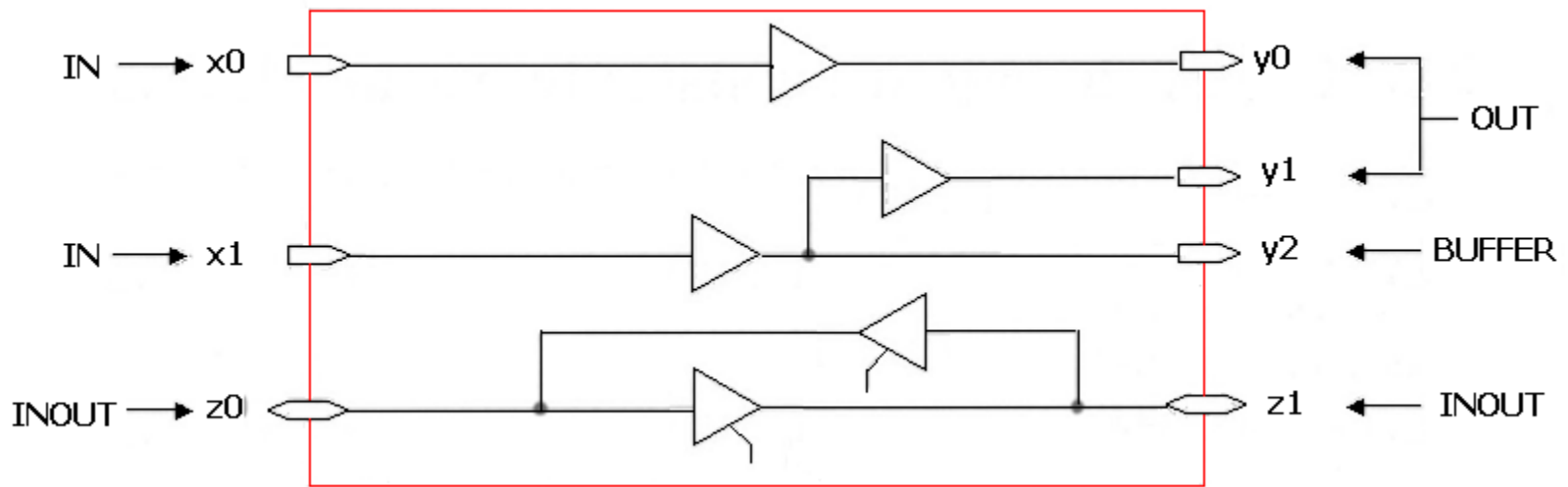
Abstração que descreve a interface de um sistema, uma placa, um chip, uma função ou uma porta lógica. Descrição geral:

```
ENTITY <nome_da_entidade> IS
    PORT(entrada_1 : IN      <tipo>;
          entrada_2 : IN      <tipo>;
          .....
          saída_1   : OUT     <tipo>;
          saída_2   : INOUT   <tipo>;
          saída_3   : BUFFER  <tipo>
    );
END <nome_da_entidade>;
```


ENTITY

Exemplo:

```
ENTITY entidade_exemplo IS
    PORT(x0, x1 : IN      <tipo_a>; -- Entradas
          y0, y1 : OUT    <tipo_b>; -- Saídas
          z0, z1 : INOUT  <tipo_c>; -- Entrada/saída
          y2       : BUFFER <tipo_d>); -- Saída realimentada internamente
END entidade_exemplo;
```



Tipos em VHDL

- ✓ Tipos pré-definidos na biblioteca work:
- ✓ Tipos definidos pelo projetista : usa a palavra reservada **TYPE** (será visto na aula 6)

Tipos em VHDL

✓ Tipos pré-definidos na biblioteca work:

- **BIT** : assume valor '0' ou '1'
- **BIT_VECTOR**: designa um conjunto de bits. Ex: "10101" ou x"00FF"
- **BOOLEAN**: assume valores { true, false} (Útil apenas para descrições abstratas, onde um sinal só pode assumir dois valores)
Obs: Em VHDL os valores booleanos (false and true) não são idênticos ao lógico '0' e '1'.
- **REAL** : Ex: -1.0 / +2.35 / 37.0 / -1.5E+23
(Utilizado durante desenvolvimento da especificação)
- **INTEIRO**: são números que variam de $(-2^{31} - 1) \leq x \leq (2^{31} - 1)$.
Ex: +1 / 5 / 1232 / -1234
- **CHARACTER** : é definido por caracter entre aspas simples "a", "x", "A"
VHDL não é "case sensitive", exceto para o tipo Character.
- **STRING**: tipo que designa um conjunto de caracteres.
Ex: "vhdl"

ENTITY

Tipos mais utilizados

Biblioteca	TIPO	Exemplo de utilização	Explicação
Work	BIT	x: IN BIT;	Entrada x assume valores lógicos '0' ou '1'.
	BIT_VECTOR	x: IN BIT_VECTOR(7 downto 0); y: IN BIT_VECTOR(0 to 7);	Define entrada vetor de 8 bits: x(7) x(6) ... x(1) x(0), onde x(0) é o LSB e y(0) y(1)y(6) y(7), onde y(0) é o MSB.
	INTEGER	X: IN INTEGER RANGE 0 TO 10;	Vetor de bits manipulado como um número inteiro. No caso, 4 bits.
IEEE	STD_LOGIC	X: IN STD_LOGIC;	Assume os valores mostrados na tabela a seguir.
	STD_LOGIC_VECTOR	x: IN STD_LOGIC_VECTOR(7 downto 0); y: IN STD_LOGIC_VECTOR(0 to 7);	Define entrada vetor de 8 bits: x(7) x(6) ... x(1) x(0), onde x(0) é o LSB e y(0) y(1)y(6) y(7), onde y(0) é o MSB.

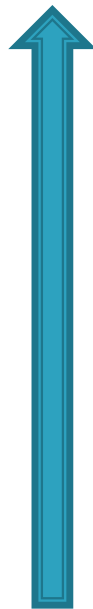
OBS: A biblioteca WORK é incluída automaticamente no projeto VHDL.

ENTITY

Tipos mais utilizados

Biblioteca IEEE: Tipos STD_LOGIC e STD_LOGIC_VECTOR

Precedência

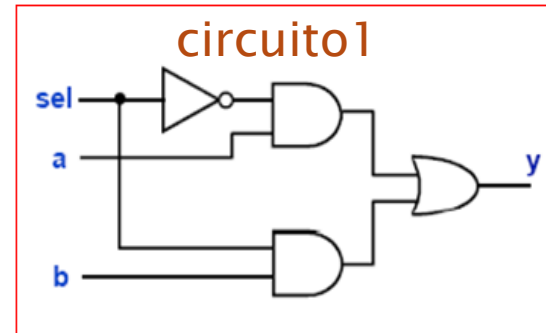


Valor		Estado Lógico	
U		Não Inicializado	
X		Desconhecido Forte	
0	1	Nível Baixo Forte	Nível Alto Forte
W		Desconhecido Fraco	
L	H	Nível Baixo Fraco	Nível Alto Fraco
Z		Alta Impedância	
-		Não Importa	

ENTITY – Exemplos

Usando a Biblioteca padrão (“Work”):

```
ENTITY circuito1 IS
    PORT (sel : IN BIT;
          a, b : IN BIT;
          y : OUT BIT
    );
END circuito1;
```

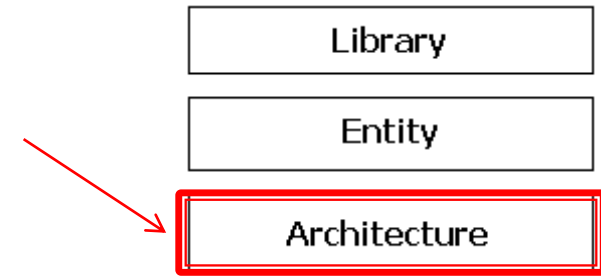


Usando o pacote std_logic_1164 da Biblioteca IEEE:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY circuito1 IS
    PORT(sel, a, b : IN STD_LOGIC;
          y : OUT STD_LOGIC
    );
END circuito2;
```

Observação: A extensão de um arquivo em VHDL é “.vhd”.
O nome do arquivo deve ser o mesmo nome da entidade.
No caso dos exemplo o arquivo deve ser salvo como
“circuito1.vhd”

ARCHITECTURE



A Arquitetura descreve a relação entre as entradas e saídas do circuito, ou seja, descreve o comportamento (funcionamento) do circuito.

Uma Arquitetura consiste de duas partes:

Declaração da Arquitetura
Corpo da Arquitetura

Arquiteturas Múltiplas:

A última arquitetura compilada é a que é utilizada

ARCHITECTURE

É formada por:

Declarações: Sinais, constantes componentes, subprogramas, etc.

Comandos: Blocos, atribuições a sinais, instanciação de componentes, chamadas de subprogramas, processos, etc.

Uma entidade pode ter várias arquiteturas:

Apenas uma delas pode estar ativa (o VHDL provê meios de escolher qual arquitetura utilizar). Isto possibilita criar variantes de um mesmo projeto.

ARCHITECTURE

Descrição geral:

```
-- Seção de declaração da arquitetura:
ARCHITECTURE <nome_identificador> OF <nome_entidade> IS
-- Região de declarações:
-- Declarações de sinais e constantes
-- Declarações de componentes
-- Declaração e corpo de subprogramas
-- Definição de novos tipos utilizados nesta arquitetura
BEGIN
-- Corpo da arquitetura:
-- Comandos concorrentes
END <nome_identificador>;
```

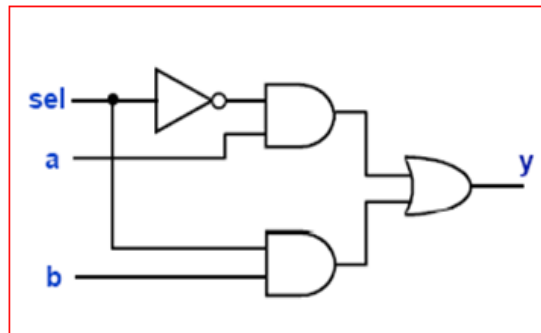
ARCHITECTURE

Exemplo completo de uma descrição em VHDL utilizando a biblioteca padrão “Work”:

```
ENTITY circuito1 IS
    PORT(sel, a, b : IN  BIT;
          y         : OUT BIT);
END circuito1;

ARCHITECTURE funcionamento OF circuito1 IS
BEGIN
    y <= ( a AND (NOT sel )) OR (b AND sel);

END funcionamento;
```



ARCHITECTURE

A arquitetura de uma entidade pode ser descrita de três formas distintas de abstração, mas que, em geral, conduzem a uma mesma implementação:

- ▶ **Descrição por Fluxo de Dados (*Data-Flow*):**

Descreve o que o sistema deve fazer utilizando expressões lógicas e/ou comandos concorrentes.

- ▶ **Descrição Estrutural:**

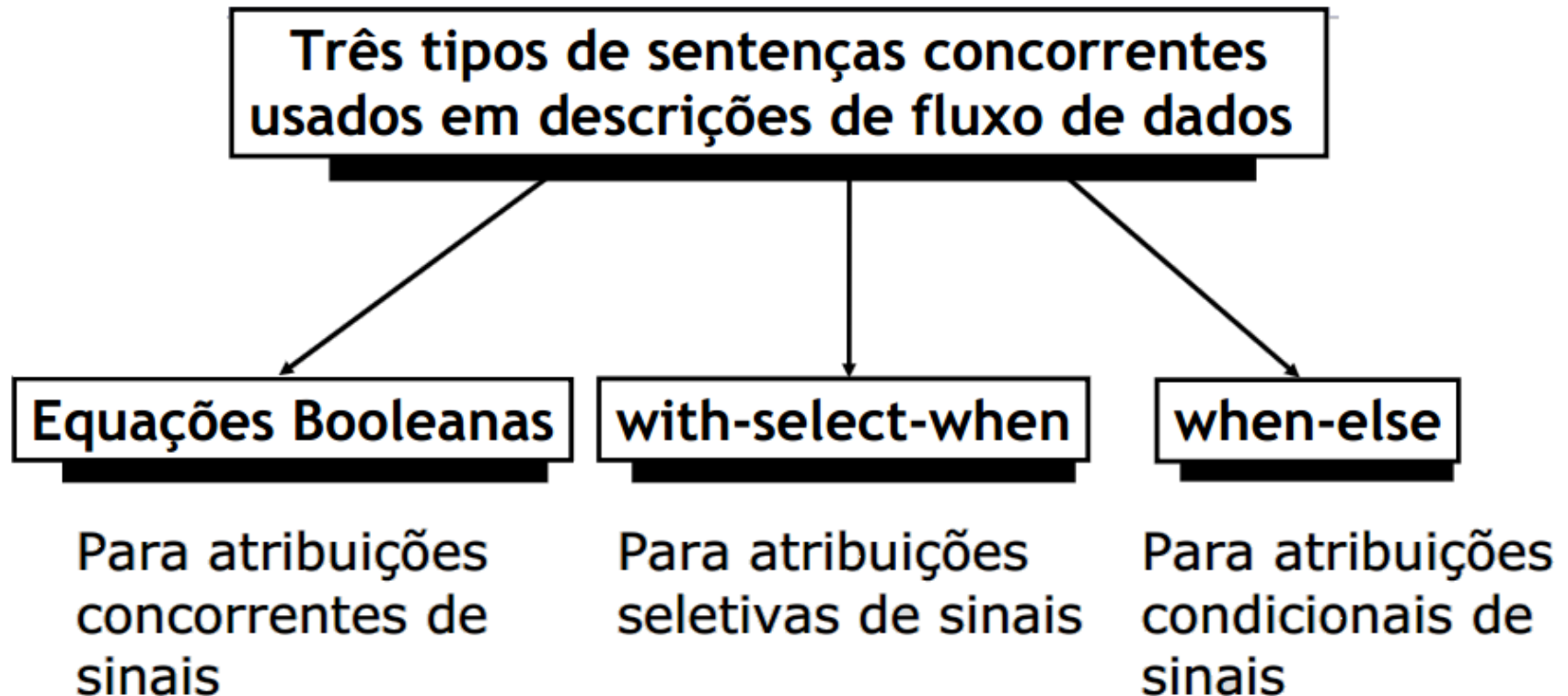
Descreve como é o hardware em termos de interconexão de componentes.

- ▶ **Descrição Comportamental:**

Descreve o que o sistema deve fazer de forma abstrata.

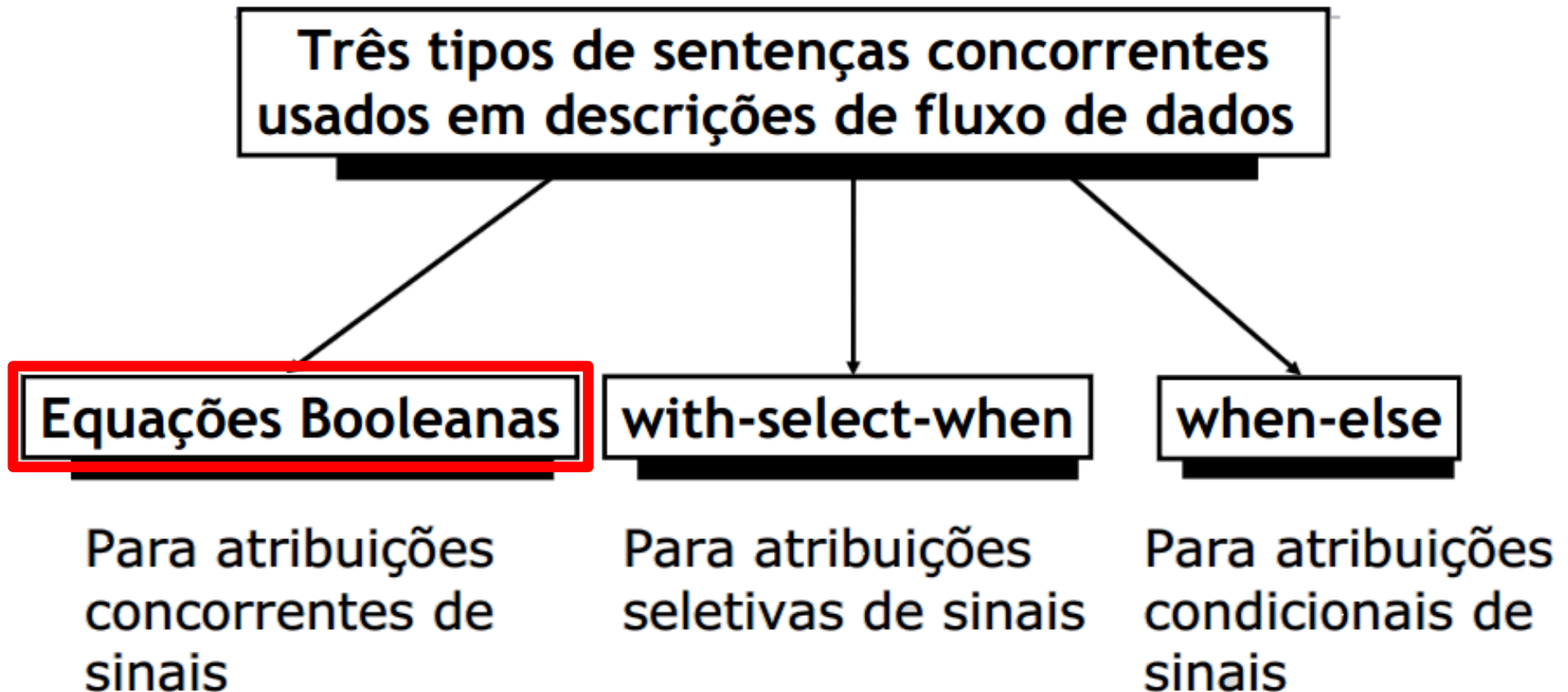
ARCHITECTURE – Fluxo de Dados

Descrição por Fluxo de Dados: Comandos (Sentenças) Concorrentes



ARCHITECTURE – Fluxo de Dados

Descrição por Fluxo de Dados: Comandos (Sentenças) Concorrentes

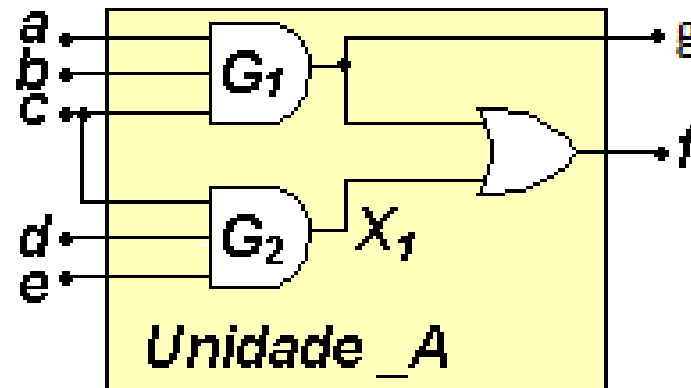


Prática nº3

Descrição por Fluxo de Dados :

- criar projeto, compilar, simular, inserir a pinagem e sintetizar no dispositivo
- Modificar projeto para utilizar SIGNAL

Obs: pinos do tipo OUT não podem ser referenciados internamente ao projeto.

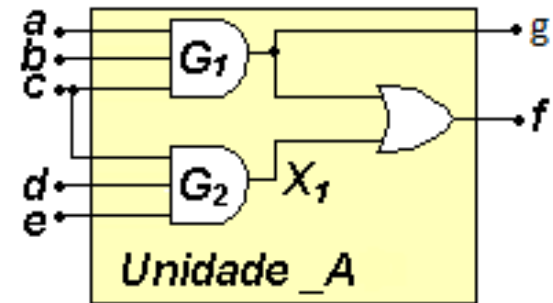


ARCHITECTURE – Fluxo de Dados

Descreve o que o sistema deve fazer utilizando expressões lógicas (equações Booleanas)

Exemplo:

```
ENTITY unidade_A IS
PORT( a, b, c, d, e: IN BIT;
      f
      : OUT BIT;
      g
      : BUFFER BIT);
END unidade_A;
```



```
ARCHITECTURE fluxo_dados OF unidade_A IS
BEGIN
    g <= (a AND b AND c);
    f <= g OR ( c AND d AND e);
END fluxo_dados;
```

Sinal

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

Declaração de Sinal:

```
SIGNAL <nome_do_sinal>: <tipo>[ := valor];
```

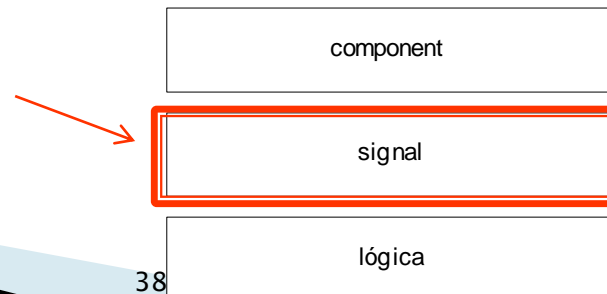
```
SIGNAL X : BIT;
```

```
SIGNAL X : BIT := '0';
```

```
SIGNAL V : BIT_VECTOR(3 DOWNT0 0);
```

```
SIGNAL V : BIT_VECTOR(3 DOWNT0 0) := "0111";
```

Architecture



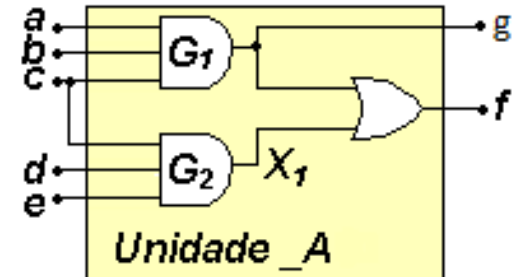
ARCHITECTURE – Fluxo de Dados

Usando a declaração de sinal (SIGNAL):

Exemplo:

```
ENTITY unidade_A1 IS
PORT( a, b, c, d, e: IN BIT;
      f              : OUT BIT;
      g              : BUFFER BIT);
END unidade_A1;
```

```
ARCHITECTURE fluxo_dados OF unidade_A1 IS
    SIGNAL X1: BIT;
BEGIN
    g    <= a AND b AND c;
    x1   <= c AND d AND e;
    f    <= X1 OR g;
END fluxo_dados;
```



ARCHITECTURE – Fluxo de Dados

sinal	Pino do fpga	
a	E16	Chave 0 do grupo A
b	H22	Chave 1 do grupo A
c	F16	Chave 2 do grupo A
d	F19	Chave 3 do grupo A
e	H21	Chave 4 do grupo A
f	R5	Segmento a do display 1
g	V2	Segmento a do display 0

