



Universidade Federal da Fronteira Sul
Curso de Ciência da Computação
Campus Chapecó

VHDL

Prof. Luciano L. Caimi
lcaimi@uffs.edu.br

VHDL: Uma linguagem para descrever sistemas digitais

Outras linguagens de descrição de hardware:

VERILOG, AHDL, Handel-C, SDL, ISP, Esterel, ... (existem dezenas)

Originalmente para especificar hardware, hoje, simulação e síntese, também!

Origem:

DoD 1980

Linguagem para documentar e descrever hardware “Very High Speed Integrated Circuits” (VHSIC), iniciado em 1980.

VHDL → VHSIC Hardware Description Language

Padrão IEEE em 1986 (Institute of Electrical and Electronics Engineers).

Em setembro de 2008 foi aprovada a mais recente versão, IEEE 1076-2008.

Linguagem utilizada mundialmente por empresas de CAD (simulação, síntese, propriedade intelectual). Verilog muito usada nos EUA.

✓ Benefícios

- ✓ Especificação do sistema digital:
 - ✓ Projetos independentes da tecnologia (implementação física é postergada)
 - ✓ Ferramentas de CAD compatíveis entre si
 - ✓ Flexibilidade: re-utilização, escolha de ferramentas e fornecedores
 - ✓ Facilidade de atualização dos projetos
 - ✓ Permite explorar, em um nível mais alto de abstração (em relação a álgebra de boole)
 - ✓ Permite, através de simulação, verificar o comportamento do sistema digital

✓ **Benefícios (cont...)**

✓ **Nível físico:**

- ✓ Reduz tempo de projeto (favorece níveis mais abstratos de projeto)
- ✓ Reduz custo
- ✓ Elimina erros de baixo nível

Consequência: reduz “time-to-market”

✓ Desvantagens

- ✓ Hardware gerado é menos otimizado
- ✓ Controlabilidade/Observabilidade de projeto reduzidas
- ✓ Falta de pessoal treinado para desenvolver com a linguagem.

✓ **Características**

✓ **Suporte para sentenças concorrentes:**

No projeto real de sistemas digitais todos os elementos do sistema estão ativos simultaneamente e realizam suas tarefas ao mesmo tempo

✓ **Suporte para sentenças Seqüenciais:**

Permite controle seqüencial como em um programa comum (isto é, case, if-then-else, loop, etc.)

✓ **Suporte para Bibliotecas:**

Primitivas definidas pelo usuário e pré-definidas pelo sistema podem residir em uma biblioteca.

✓ **Suporte a Projeto Hierárquico**

✓ **Independente de Tecnologia**

✓ **Características (cont...)**

✓ **Projeto Genérico:**

Descrições genéricas são configuráveis em tamanho, características físicas, temporização, condições de operação, etc.

✓ **Uso de subprogramas:**

Habilidade de definir e usar funções e procedimentos;

Subprogramas são utilizados para a conversão explícita de tipos, redefinição de operadores, etc.

✓ **Características (cont...)**

✓ **Suporta declaração de tipos:**

não está limitada a tipos de dados como Bit ou Booleanos, permite tipos inteiros, de ponto flutuante, enumerados, assim como tipos definidos pelos usuários.

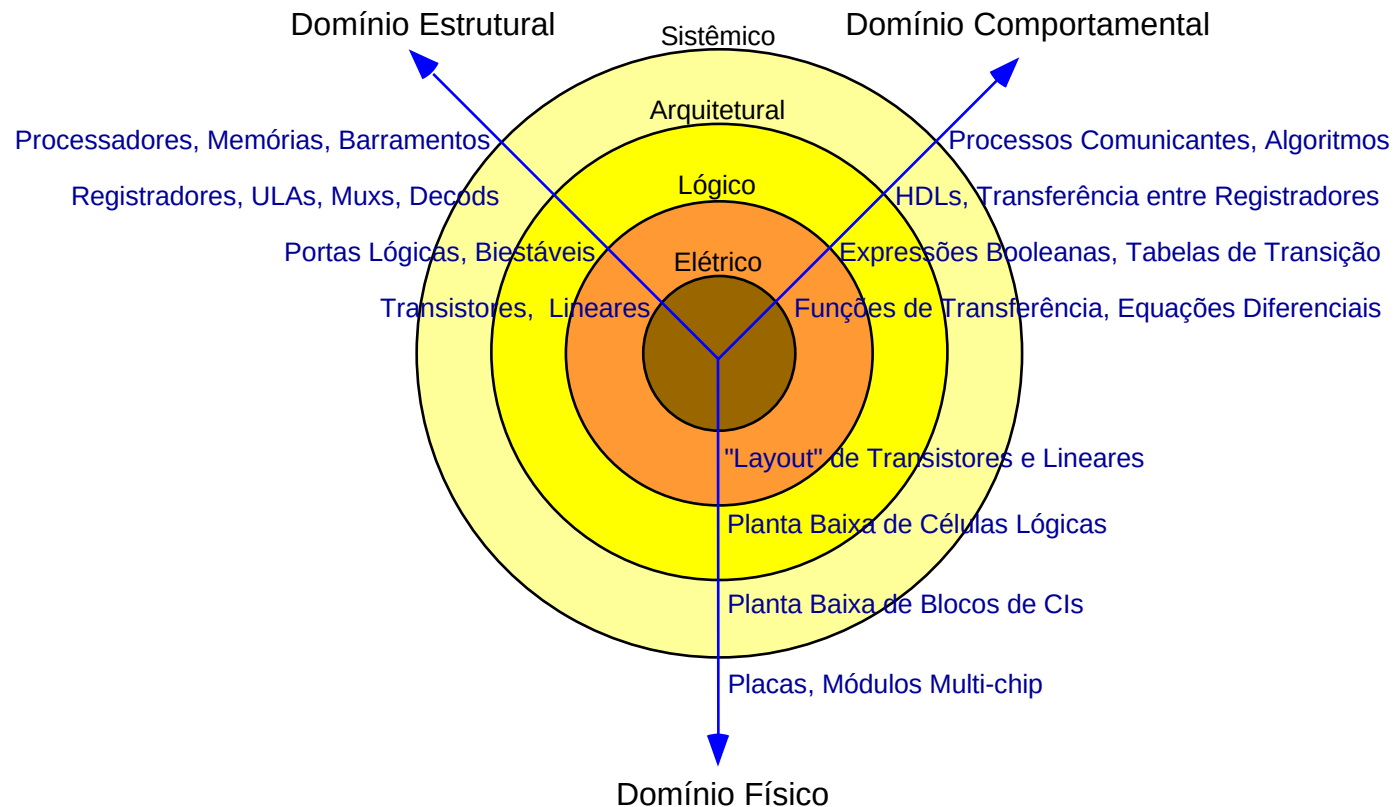
possibilita definição de novos operadores para os novos tipos criados pelo usuário.

✓ **Controle de Temporização :**

Habilidade para especificar temporização em todos os níveis.

Construções para detecção de rampa do sinal (subida ou descida), especificação de atraso, etc. estão disponíveis.

► Modelo de Gajski-Kuhn



Círculo → Nível de abstração;

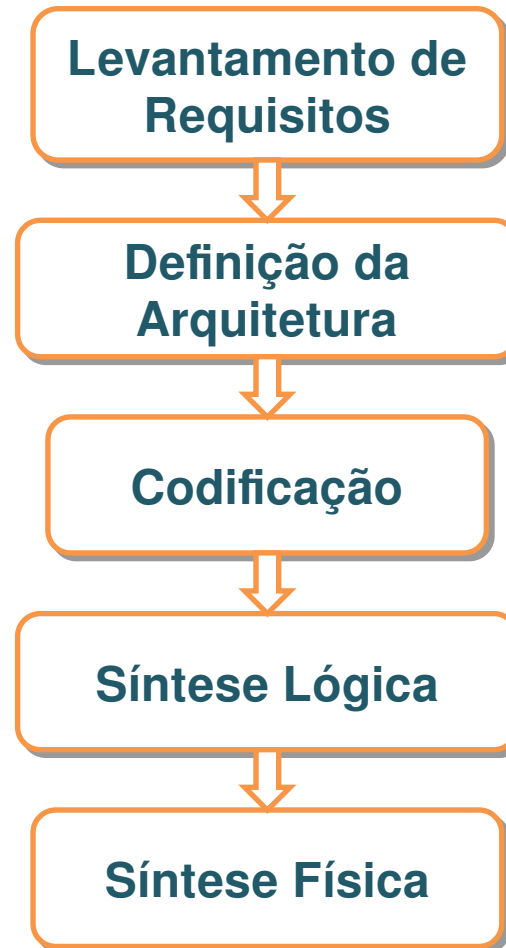
Eixo → domínio de descrição;

Intersecção círculo-eixo (vértices) → Modelo de descrição;

Transformação entre níveis (aresta no grafo) → síntese.

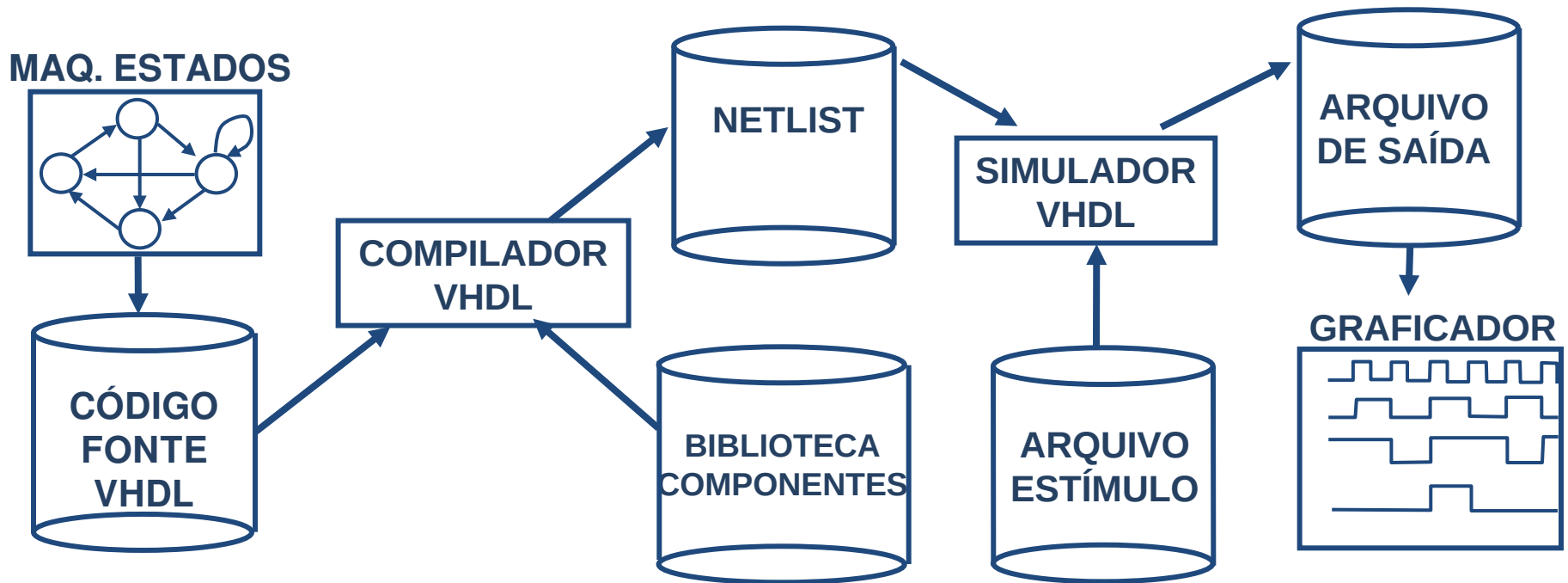
- ✓ Permite descrever hardware em diversos níveis de abstração
 - Algorítmico (também chamado Comportamental)
 - Transferência entre registradores (RTL)
 - Nível lógico com atrasos unitários
 - Nível lógico com atrasos arbitrários
 - Estrutural (interconexão entre componentes)

✓ Fluxo de projeto (resumido)



Introdução

✓ Codificando... Compilando... Simulando...



Representação de um sistema em VHDL



- ✓ Cada módulo tem sua própria “*entity*” e “*architecture*”.
- ✓ As arquiteturas podem ser descritas tanto a nível **comportamental** quanto **estrutural** ou uma mistura.
 - Comportamental**: Algorítmica ou fluxo de dados;
 - Estrutural**: indica componentes e conexões;
- ✓ Toda a comunicação ocorre **através das portas declaradas** em cada *entity*, observando-se o tipo, tamanho, se é sinal ou barramento e a direção.
- ✓ Várias funções e tipos básicos são armazenados em bibliotecas (*library*). A biblioteca “IEEE” sempre é incluída.
- ✓ Biblioteca do usuário (default): work. Todos os arquivos contidos no diretório de trabalho fazem parte da biblioteca do usuário.

Representação de um sistema em VHDL



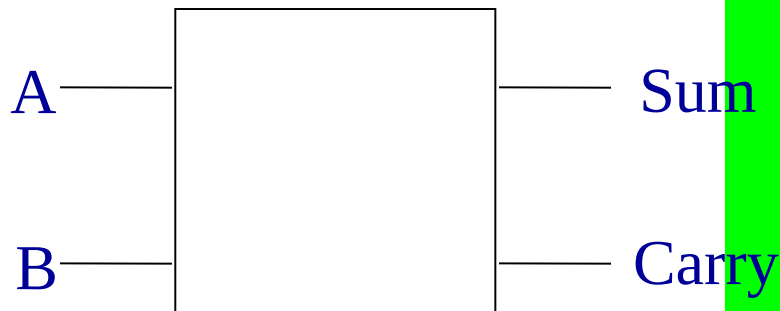
- ✓ VHDL não é case sensitive
Bit \leftrightarrow bit \leftrightarrow BIT
- ✓ Comentários: dois hifens adjacentes (--)
-- esta linha é um comentário

Representação de um sistema em VHDL



Entity

- ✓ *Especifica somente a interface*
- ✓ *Não contém definição do comportamento*
- ✓ *Direção: in, out, inout, buffer*



```
entity halfadder is
    port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        sum: out STD_LOGIC;
        carry: out STD_LOGIC
    );
end halfadder;
```

Architecture

- ✓ *Especifica o comportamento da entity*
- ✓ *Deve ser associada a uma entity específica*
- ✓ *Uma entity pode ter várias architectures*

```
architecture comp of halfadd is
begin
    sum <= A xor B;
    carry <= A and B;
end comp;
```


Representação de um sistema em VHDL



Entity:

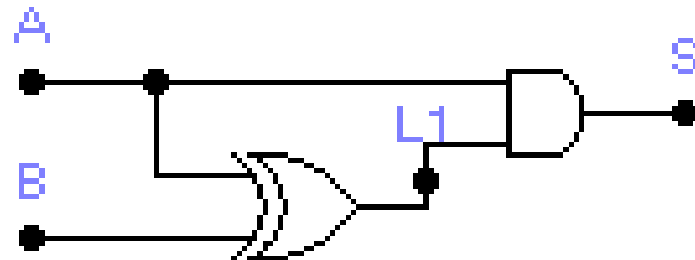
```
entity nome is  
  [generic (lista de parâmetros);]  
  [port (lista de parâmetros);]  
  [declarações ]  
  [begin sentenças ]  
end [ entity ] [ nome ];
```

Architecture:

```
architecture nome of nome_entidade is  
  [declarações ]  
  begin  
    [sentenças concorrentes ]  
end [ architecture ] [ nome ];
```

Descrições:

- a) algorítmica
- b) fluxo de dados
- c) estrutural

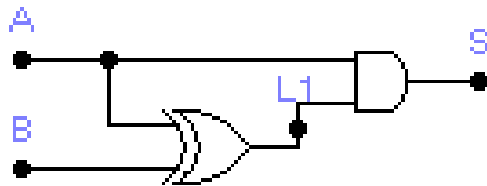


Representação de um sistema em VHDL



a) algorítmica

```
entity comportamento is
    port (
        A: in STD_LOGIC;
        B: in STD_LOGIC;
        S: out STD_LOGIC
    );
end comportamento;
```



```
architecture comport_algor of
    comportamento is
begin
    process(A,B)
    begin
        if(B < A) then
            s <= '1';
        else
            s <= '0';
        end if;
    end process;
end comport_algor;
```

Primitiva de base (concorrência): **process**

Observar diferença entre variável e sinal:

Variável: interna ao processo, do tipo natural, atribuição
IMEDIATA

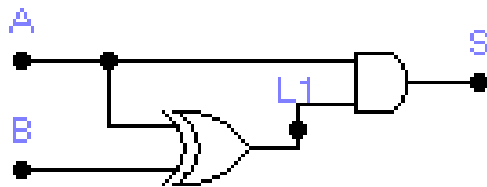
Sinal: global, com atribuição ao término do processo

✓ Notar que na declaração do processo há uma lista de ativação

Significado: o processo está em espera até um sinal da lista de ativação mudar.

b) fluxo de dados

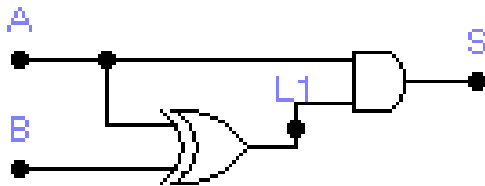
```
entity comportamento is  
    port (  
        A: in STD_LOGIC;  
        B: in STD_LOGIC;  
        S: out STD_LOGIC  
    );  
end comportamento;
```



```
architecture comport_fluxo of  
    comportamento is  
begin  
    s <= '1' when B < A  
        else '0';  
end comport_fluxo ;
```

c) estrutural

```
entity comportamento is
  port (
    A: in STD_LOGIC;
    B: in STD_LOGIC;
    S: out STD_LOGIC
  );
end comportamento;
```



```
architecture comp_estrutural of
  comportamento is
    signal L1: STD_LOGIC;
    component XOR2 is
      port (A1,B1: in std_logic; X1: out
std_logic);
    end component;
    component And2 is
      port (A2,B2: in std_logic; X2: out
std_logic);
    end component
  begin
    U1: entity xor2 port map (A,B,L1);
    U2: entity and2 port map (A,L1,S);
  end comp_estrutural ;
```

(existem outros 2 arquivos com os pares entidade arquitetura para AND2 e XOR2)

✓ **Operadores e Expressões**

são fórmulas que realizam operações sobre objetos de mesmo tipo.

Operações lógicas: and, or, nand, nor, xor, not

Operações relacionais: =, /=, <, <=, >, >=

Operações aritméticas: - (unária), abs

Operações aritméticas: +, -

Operações aritméticas: *, /

Operações aritméticas: mod, rem, **

Concatenação: &

Menor

PRIORIDADE

Maior

✓ *Operadores*

Operadores Lógicos – trabalham com tipos BIT, BOOLEAN, STD_LOGIC, vetores de tamanho igual e **NÃO EM INTEIROS**.

Operadores Relacionais – trabalham com qualquer tipo de escalar ou tipo array UNI-DIMENSIONAL cujo tipo de elemento é um tipo discreto (enumeração ou inteiro).

Operadores Aritméticos – trabalham com inteiro, real e STD_LOGIC_VECTOR.

Observações:

- ✓ Operações lógicas são realizadas sobre tipos bit e boolean.
- ✓ Operadores aritméticos trabalham sobre inteiros e reais. Incluindo-se o package da Synopsys, por exemplo, pode-se somar vetores de bits.
- ✓ Todo tipo físico pode ser multiplicado/dividido por inteiro ou ponto flutuante.
- ✓ Concatenação é aplicável sobre caracteres, strings, bits, vetores de bits e arrays.

Exemplos: “ABC” & “xyz” resulta em: “ABCxyz”

“1001” & “0011” resulta em: “10010011”

Qual/quais das linhas abaixo é/são incorreta/s? Justifique a resposta.

variable A, B, C, D : bit_vector (3 downto 0);

variable E,F,G : bit_vector (1 downto 0);

variable H,I,J,K : bit;

[] A := B xor C and D ;

[] H := I and J or K;

[] A := B and E;

[] H := I or F;

VHDL é uma linguagem fortemente tipada

(integer 1 \neq real 1.0 \neq bit '1')

- ✓ auxilia a detectar erros no início do projeto
exemplo: conectar um barramento de 4 bits a um barramento de 8 bits

Tópicos

Escalares

Objetos

Expressões

Escalar é o oposto ao array

character / bit / boolean / real / integer /
physical_unit
std_logic (IEEE)

✓ *Bit*

Assume valores '0' e '1' (usar aspas simples)

Declaração explícita: bit('1'), pois neste caso '1'
também pode ser 'character'.

bit não tem relação com o tipo boolean.

bit_vector: tipo que designa um conjunto de bits.

Exemplo: "001100" ou x"00FF". **Não é escalar.**

✓ *Boolean*

Assume valores *true* e *false*.

Útil apenas para descrições abstratas, onde um sinal só pode assumir dois valores

✓ *Real*

Utilizado durante desenvolvimento da especificação

Sempre com o ponto decimal

Exemplos: -1.0 / +2.35 / 37.0 / -1.5E+23

✓ *Inteiros*

Exemplos: +1 / 1232 / -1234

NÃO é possível realizar operações lógicas sobre inteiros
(deve-se realizar a conversão explícita)

Vendedores provêem versões próprias: signed, **bit_vector**
(este tipo permite operações lógicas e aritméticas)

✓ *Character*

VHDL **não** é “case sensitive”, **exceto** para caracteres.

valor entre aspas simples: ‘a’, ‘x’, ‘0’, ‘1’, ...

declaração explícita: `character` (‘1’), pois neste caso ‘1’ também pode ser ‘bit’.

string: tipo que designa um conjunto de caracteres. Exemplo: “xuxu”.

✓ *Physical*

Representam uma medida: voltagem, capacitância, tempo

Tipos pré-definidos: fs, ps, ns, um, ms, sec, min, hr

✓ *Intervalos (range)*

sintaxe: *range valor_baixo* **to** *valor_alto*

range valor_alto **downto** *valor_baixo*

integer range 1 to 10 **NÃO** integer range 10 to 1

real range 1.0 to 10.0 **NÃO** integer range 10.0 to 1.0

declaração sem **range** declara todo o intervalo

declaração **range<>** : declaração postergada do intervalo

✓ *Enumerações*

Conjunto ordenando de nomes ou caracteres.

Exemplos:

```
type logic_level is ('0', '1', 'X', 'Z');
```

```
type octal is ('0', '1', '2', '3', '4', '5', '6', '7');
```


Array *coleção de elementos de mesmo tipo*

type word is array (31 downto 0) of bit;

type memory is array (address) of word;

type transform is array (1 to 4, 1 to 4) of real;

type register_bank is array (byte range 0 to 132) of integer;

✓ *array sem definição de tamanho*

type vector is array (integer range <>) of real;

exemplos de arrays pré definidos:

type string is array (positive range <>) of character;

type bit_vector is array (natural range <>) of bit;

preenchimento de um array: posicional ou por nome

type a is array (1 to 4) of character;

posicional: ('f', 'o', 'o', 'd')

por nome: (1 => 'f', 3 => 'o', 4 => 'd', 2 => 'o')

valores default: ('f', 4 => 'd', others => 'o')

```
signal z_bus : bit_vector (3 downto 0);  
signal c_bus : bit_vector (0 to 3);
```

```
z_bus <= c_bus;
```

z_bus(3)	←	c_bus(0)
z_bus(2)	←	c_bus(1)
z_bus(1)	←	c_bus(2)
z_bus(0)	←	c_bus(3)

```
z_bus(3) <= c_bus(2);
```

Obs.:

- tamanho dos arrays deve ser o mesmo
- elementos são atribuídos por posição, pelo número do elemento

Objetos podem ser escalares ou vetores (arrays)

Devem obrigatoriamente iniciar por uma letra, depois podem ser seguidos de letras e dígitos (o caracter “_” pode ser utilizado). Não são case sensitive, ou seja XuXu é o mesmo objeto que XUXU ou xuxu.

Constantes / Variáveis / Sinais

✓ **Constante: nome dado a um valor fixo**

sintaxe: *constant identificador : tipo [:=expressão];*

correto: *constant gnd: real := 0.0;*

incorreto *gnd := 4.5; -- atribuição a constante fora da declaração*

constantes podem ser declaradas em qualquer parte, porém é aconselhável declarar as freqüentemente utilizadas em um package

✓ Variáveis

utilizadas em processos, sem temporização, atribuição a elas é imediata.

sintaxe:

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

exemplo:

variable indice : **integer range** 1 to 50 := 50;

variable ciclo_maquina : **time range** 10 ns to 50 ns := 10ns;

variable memoria : **bit_vector** (0 to 7)

variable x, y : **integer**;

✓ Sinais

Comunicação entre módulos.

Temporizados.

Podem ser declarados em entity, architecture ou em package.

Não podem ser declarados em processos, podendo serem utilizados no interior destes.

sintaxe:

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

exemplo

- **signal** cont : **integer range 50 downto 1**;
- **signal** ground : **bit** := '0';
- **signal** bus : **bit_vector**;

✓ Sinais x variáveis

Uma diferença fundamental entre variáveis e sinais é o atraso da atribuição

```
ARCHITECTURE signals OF test IS
  SIGNAL a, b, c, out_1, out_2 : BIT;
BEGIN
  PROCESS (a, b, c)
  BEGIN
    out_1 <= a NAND b;
    out_2 <= out_1 XOR c;
  END PROCESS;
END signals;
```

Time	a	b	c	out_1	out_2
0	0	1	1	1	0
1	1	1	1	1	0
1+d	1	1	1	0	0

✓ *Sinais x variáveis*

Uma diferença fundamental entre variáveis e sinais é o atraso da atribuição

```
ARCHITECTURE variables OF test IS  
  SIGNAL a, b, c: BIT;  
  VARIABLE out_3, out_4 : BIT;  
BEGIN  
  PROCESS (a, b, c)  
  BEGIN  
    out_3 := a NAND b;  
    out_4 := out_3 XOR c;  
  END PROCESS;  
END variables;
```

Time	a	b	c	out_3	out_4
0	0	1	1	1	0
1	1	1	1	0	1

VHDL provê facilidades de **paralelismo** entre diferentes processos e atribuição de sinais.

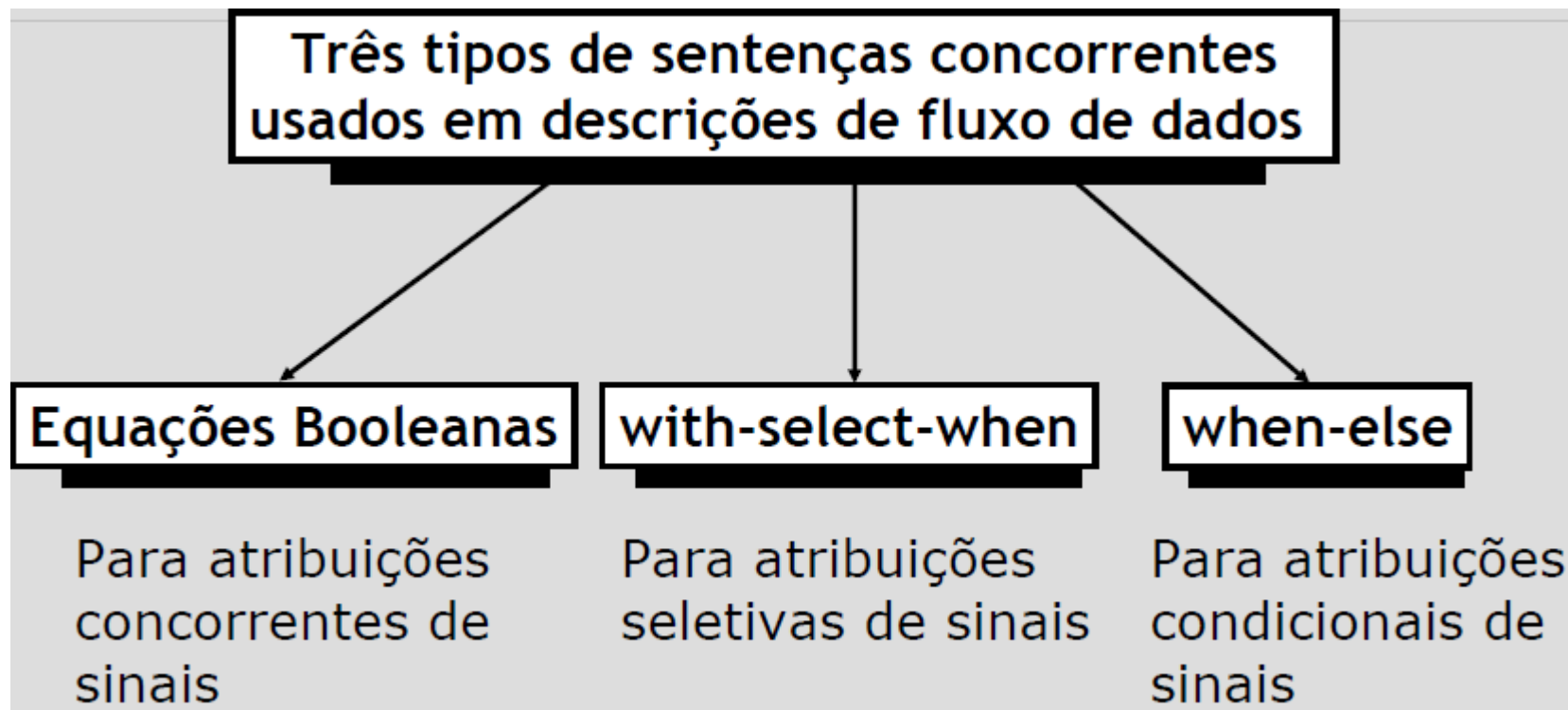
Dentro dos processos pode-se especificar um conjunto de ações seqüenciais, executadas passo a passo. É um estilo de descrição semelhante a outras linguagens de programação.

Comandos exclusivos de processos: **atribuição de variáveis, if, case, for, while, wait** (não se pode usá-los fora de processos!)

Variáveis não passam valores fora do processo na qual foram declaradas, são locais. Elas sequer existem fora de um processo.

As atribuições são seqüenciais, ou seja, a ordem delas importa.

Sentenças concorrentes:



Sentenças concorrentes: equações booleanas

```
entity control is port(mem_op, io_op, read, write: in bit;  
                      memr, memw, io_rd, io_wr:out bit);  
end control;
```

```
architecture control_arch of control is  
begin  
    memw <= mem_op and write;  
    memr <= mem_op and read;  
    io_wr <= io_op and write;  
    io_rd <= io_op and read;  
end control_arch;
```

Sentenças concorrentes: with – select – when

```
entity mux is port(a,b,c,d: in std_logic_vector(3 downto 0);  
                  s: in std_logic_vector(1 downto 0);  
                  x: out std_logic_vector(3 downto 0));  
end mux;
```

```
architecture mux_arch of mux is  
begin  
  with s select  
    x <= a when "00",  
         b when "01",  
         c when "10",  
         d when others;  
end mux_arch;
```

Sentenças concorrentes: when - else

```
architecture mux_arch of mux is
begin
    x <= a when (s = "00") else
        b when (s = "01") else
        c when (s = "10") else
        d;
end mux_arch;
```

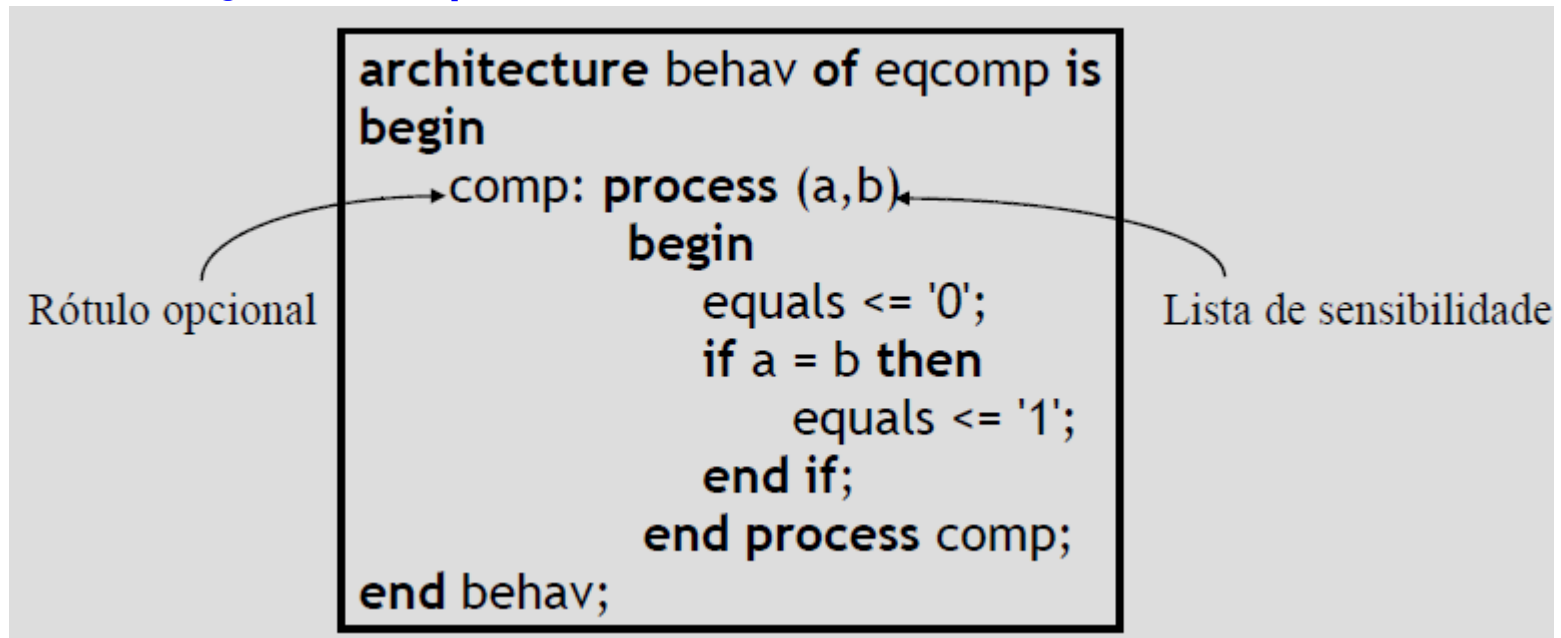
Pode ser
qualquer
condição
simples

Sentenças sequenciais:

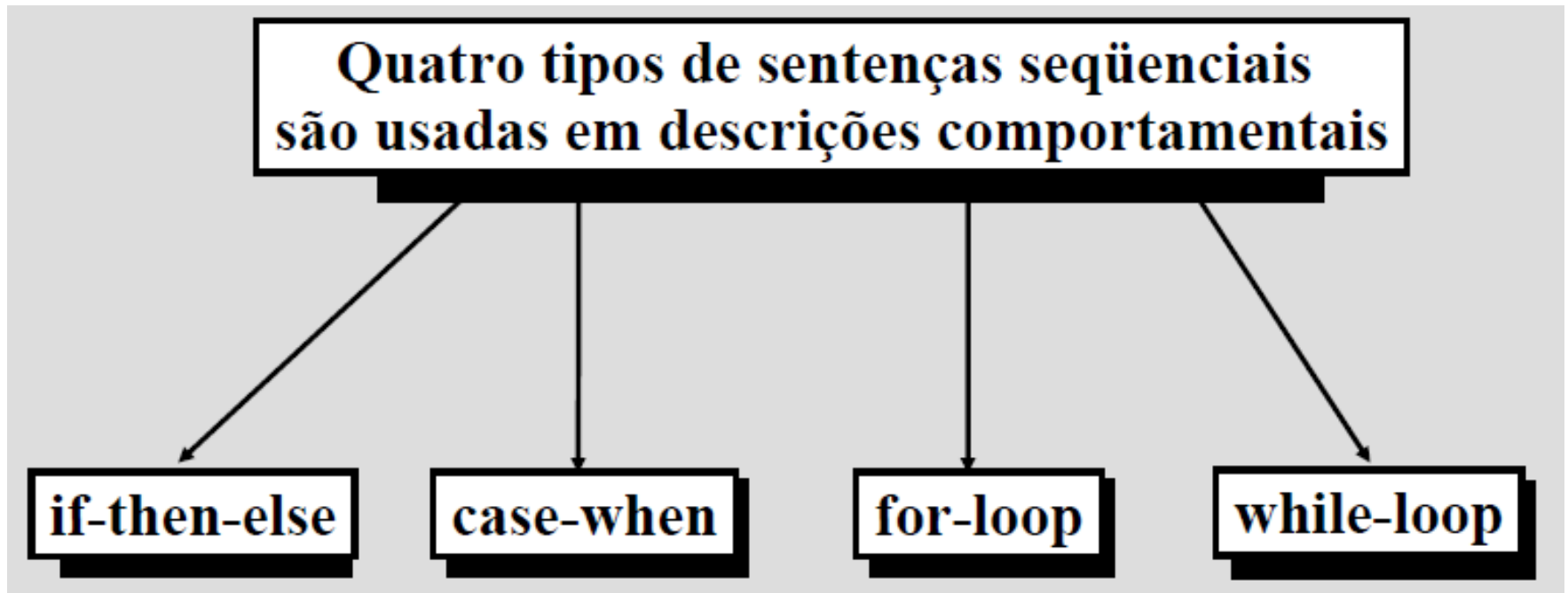
- ✓ Sentenças sequenciais são contidas em processos, funções ou procedimentos;
- ✓ Dentro de um processo a atribuição de um sinal é sequencial do ponto de vista da simulação;
- ✓ A ordem na qual as atribuições de sinais são feitas AFETAM o resultado.

Sentenças sequenciais: Processo

- ✓ Um processo é uma construção em VHDL que guarda algoritmos;
- ✓ Um processo tem uma lista de sensibilidade que identifica os sinais cuja variação irão causar a execução do processo;



Sentenças sequenciais:



Comando if - then - else

```
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;
```

IMPORTANTE:

teste de borda de subida: **if clock'event and clock='1' then ...**

teste de borda de descida: **if clock'event and clock='0' then ...**

a seqüência na qual estão definidos os 'ifs' implica na prioridade das ações.

Comando if - then - else

Qual a implementação em hardware da seguinte seqüência de comandos ?

```
process(A, B, control)
begin
    if( control = '1' ) then
        Z <= B;
    else
        Z <= A;
    end if;
end process;
```

Comando for - loop

- ✓ *para descrever comportamento / estruturas regulares*
- ✓ *o “for” declara um objeto, o qual é alterado somente durante o laço*
- ✓ *internamente o objeto é tratado como uma constante e não deve ser alterado.*

```
for item in 1 to last_item loop  
    table(item) := 0;  
end loop;
```

Comando for (só em processos)

exit: termina o laço

```
for i in 1 to max_str_len loop  
    a(i) := buf(i);  
    exit when buf(i) = NUL;  
end loop;
```

Comando for - loop

Qual a função do laço abaixo ?

```
function conv (byte : word8) return integer is
    variable result : integer := 0;
    variable k : integer := 1;
begin
    for index in 0 to 7 loop
        if ( std_logic'(byte(index)) = '1')
            then result := result + k;
        end if;
        k := k * 2;
    end loop;
    return result;
end conv ;
```

Comando while - loop

```
while index < length and str(index) /= ' ' loop  
    index := index + 1;  
end loop;
```

Comando case - when

porta_programável :

process (**Mode**, A, B)

begin

case Mode is

when "000" => saída <= A and B;

when "001" => saída <= A or B;

when "010" => saída <= A nand B;

when "011" => saída <= A nor B;

when "100" => saída <= not A;

when "101" => saída <= not B;

when others => saída <= '0'

end case;

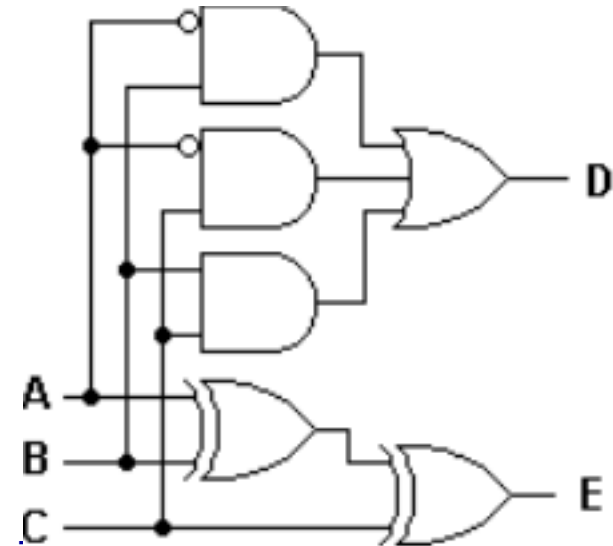
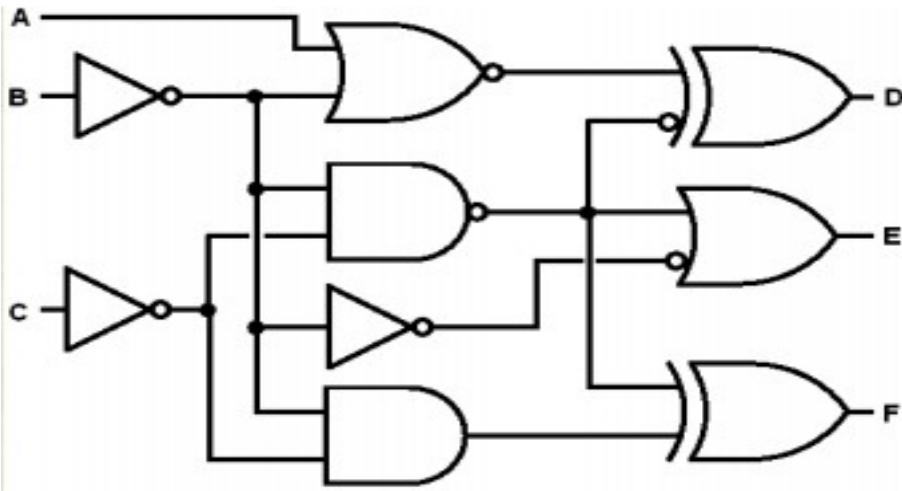
end process porta_programável;

Comando null

*serve, por exemplo, para indicar “faça nada”
em uma condição de case.*

```
case controller_command is  
    when forward => marcha <= '1';  
    when reverse => marcha <= '0';  
    when idle => null;  
end case;
```

1. Faça a descrição de uma porta lógica OU de 4 entradas
2. Refaça o exercício acima, considerando agora que as entradas são um array
3. Dados os esquemáticos, obtenha descrições VHDL compatíveis



4. Projete e descreva em VHDL um decodificador 3 para 8. Utilize a seguinte entidade para descrever sua solução em VHDL:

```
entity decoder_3to8 is
  port (
    A : in bit_vector (2 downto 0); -- Decoder Input
    Y : out bit_vector (7 downto 0) -- Decoder Output
  );
end entity decoder_3to8;
```

5. Projete e descreva em VHDL um multiplexador 4 para 1. Utilize a seguinte entidade para descrever a sua solução em VHDL:

```
entity mux is
  port (
    I : in bit_vector (3 downto 0); -- Input
    SEL : in bit_vector (1 downto 0); -- Select
    Y : out bit -- Output
  );
end entity mux;
```