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

VHDL

Prof. Geomar A Schreiner
gschreiner@uffs.edu.br

By Caimi

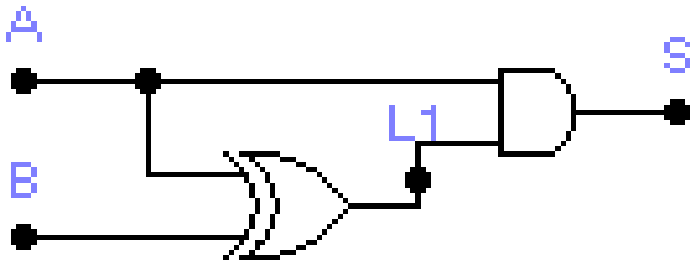
Representação de um sistema em VHDL

Descrições:

a) algorítmica

b) fluxo de dados

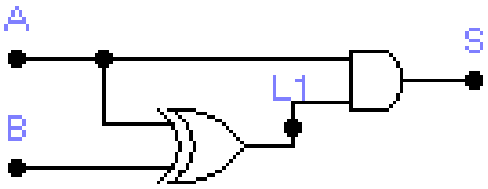
c) estrutural



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
```




```
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;
```

Representação de um sistema em VHDL

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

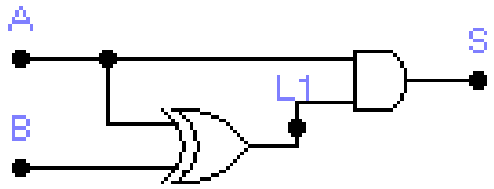
- ✓ Notar que na declaração do processo há uma lista de ativação

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

Representação de um sistema em VHDL

b) fluxo de dados

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

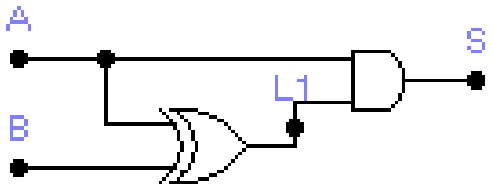



```
architecture comport_fluxo of
                        comportamento is
begin
   s <= ‘1’ when B < A
            else '0';
end comport_fluxo ;
```

Representação de um sistema em VHDL

c) estrutural

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

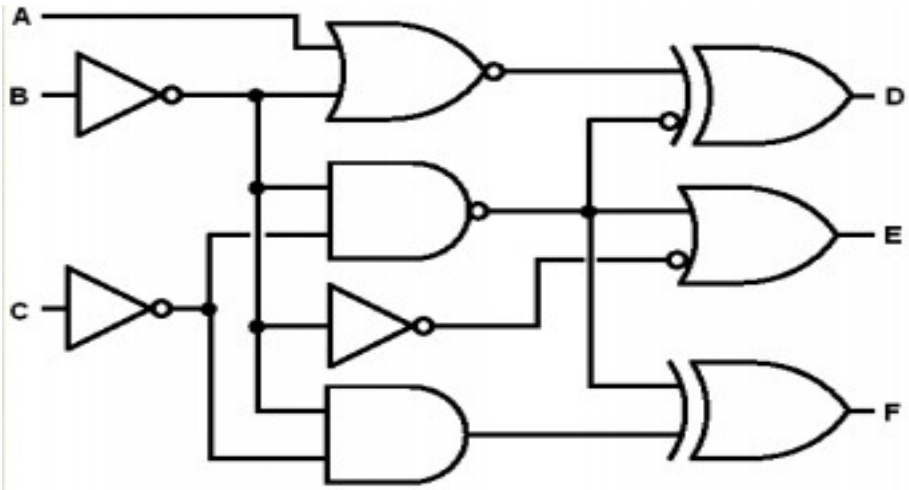


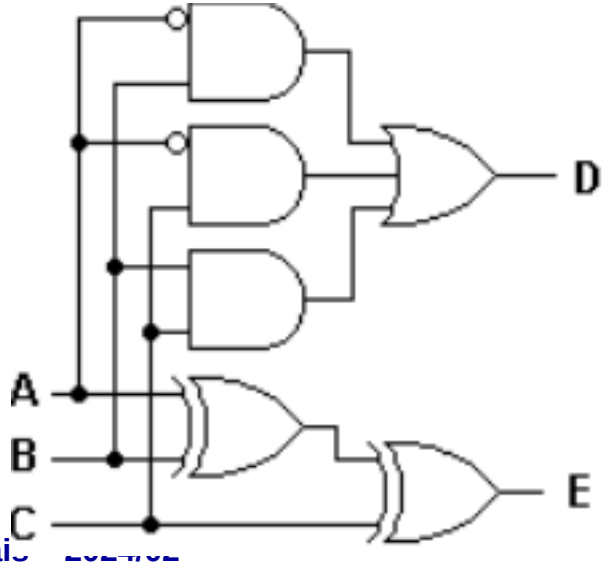

```
architecture comp_estrutural of
                    comportamento is
   signal L1: STD_LOGIC;
   component  XOR2 is
    port (A1,B1: in std_logic; X1: out
std_logic);
    end component;
   component  And2 is
    port (A2,B2: in std_logic; X2: out
std_logic);
    end component
begin
   U1:  xor2 port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

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

Sintaxe VHDL

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





Sintaxe VHDL

✓ *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

Sintaxe VHDL

✓ *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.

Sintaxe VHDL

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”

Sintaxe VHDL

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;*

Sintaxe VHDL

VHDL é uma linguagem fortemente tipada

(integer 1 ≠ real 1.0 ≠ 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

Sintaxe VHDL

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.

Sintaxe VHDL

- ✓ *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)

Sintaxe VHDL

✓ *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

Sintaxe VHDL

✓ *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

Sintaxe VHDL

✓ *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´);
```

Sintaxe VHDL

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;
```

Sintaxe VHDL

```
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

Sintaxe VHDL

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

Sintaxe VHDL

- ***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

Sintaxe VHDL

✓ 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 : **STD_LOGIC_VECTOR** (0 to 7)

variable x, y : **integer**;

Sintaxe VHDL

- ✓ *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 : **STD_LOGIC** := ´0´;
- **signal** bus : **STD_LOGIC_VECTOR**;

Sintaxe VHDL

- ✓ *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

Sintaxe VHDL

- ✓ *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

Sintaxe VHDL

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.

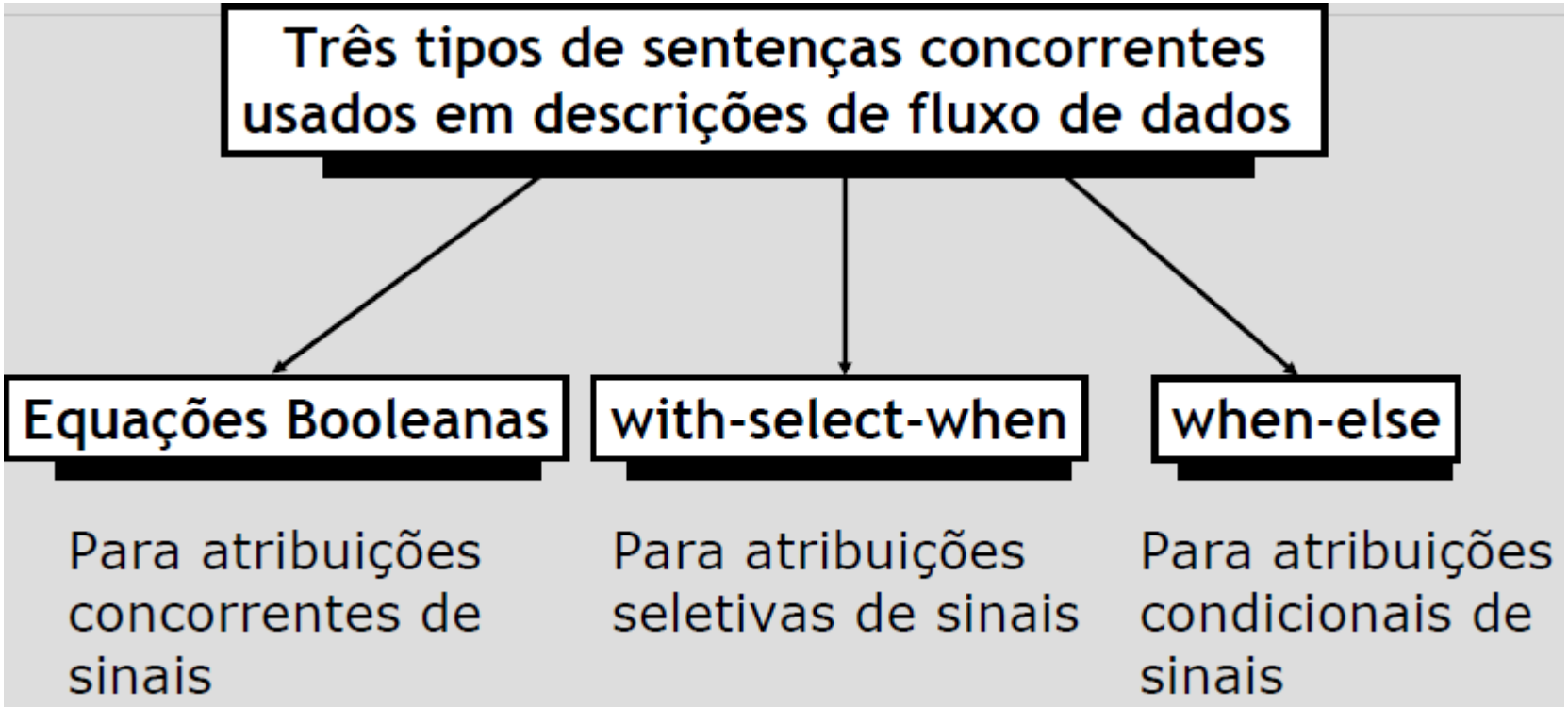
Comandos exclusivos de processos: atribuição de variáveis, if, case, for, while, wait (não se pode usá-los fora de processos!)

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.

Sintaxe VHDL

Sentenças concorrentes:



Sintaxe VHDL

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;
```

Sintaxe VHDL

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

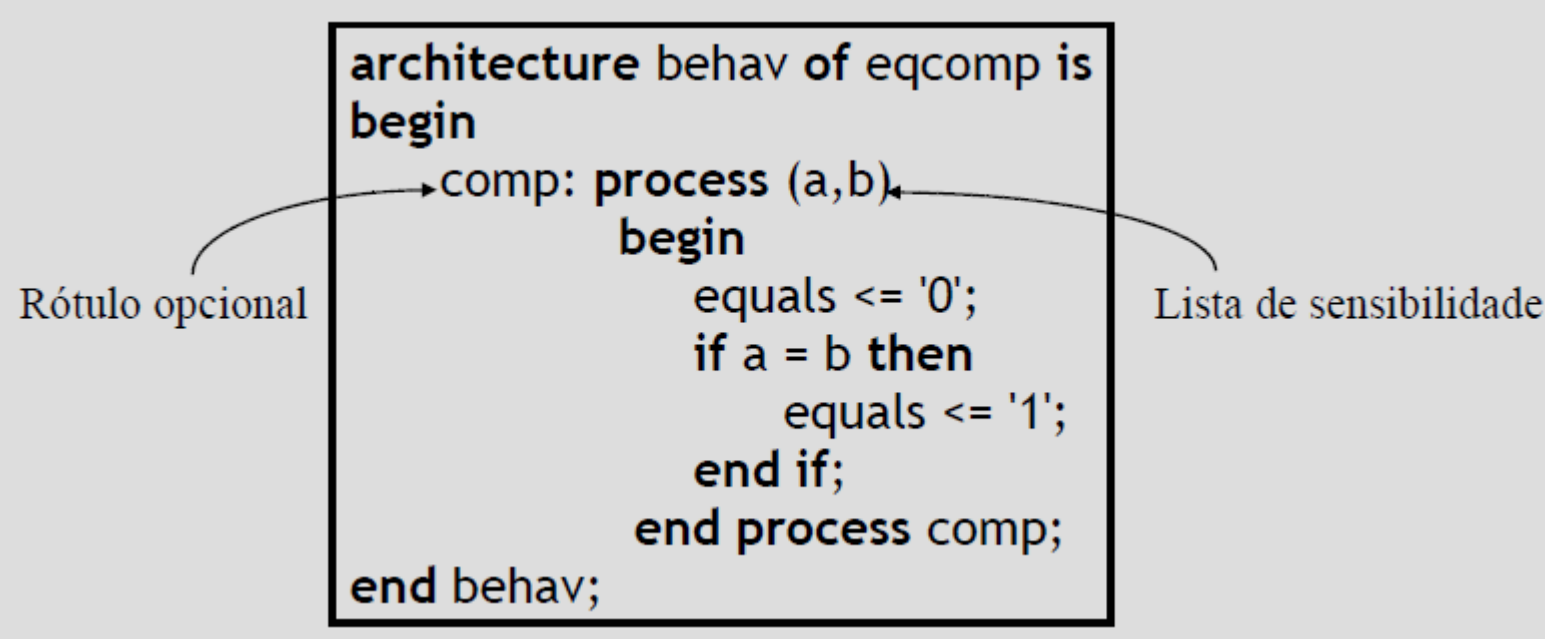
Sintaxe VHDL

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.

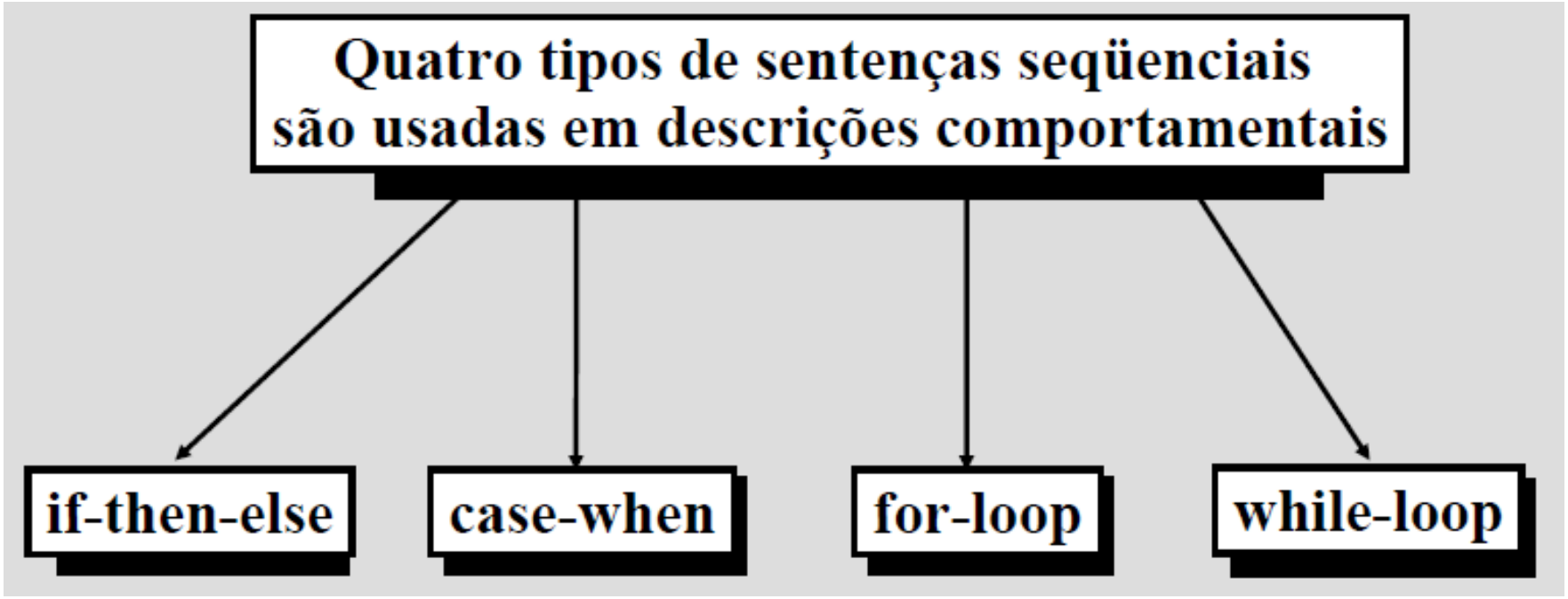
Sentenças sequenciais: Processo

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



Sintaxe VHDL

Sentenças sequenciais:



Sintaxe VHDL

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 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 10 loop
    b(i) <= buf(i);
    exit when buf(i) = NULL;
end loop;
```

Comando for - loop

Qual a função do laço abaixo ?

```
1   FUNCTION conv (byte : word8) RETURN INTEGER IS
2      VARIABLE result : INTEGER := 0;
3      VARIABLE k : INTEGER := 1;
4   BEGIN
5      FOR index IN 0 TO 7 LOOP
6         IF (STD_LOGIC'(byte(index)) = '1')
7            THEN
8            result := result + k;
9         END IF;
10        k := k * 2;
11     END LOOP;
12     RETURN result;
13  END conv;
```

Sintaxe VHDL

Comando for - loop

Qual a função do laço abaixo ?

```
1   [PURE | IMPURE] FUNCTION < function_name > (
2       < parameter1_name > : < parameter1_type > := < default_value > ;
3         < parameter2_name > : < parameter2_type > := < default_value > ;
4   ...) RETURN < return_type > IS
5      < constant_or_variable_declaration >
6   BEGIN
7      < code_performed_by_the_function >
8      RETURN < value >
9   END FUNCTION;
```

Sintaxe VHDL

Comando while - loop

```
WHILE index < length AND str(index) /= ' ' LOOP
    index := index + 1;
END LOOP;
```

Sintaxe VHDL

Comando case - when

```
1   porta_programável : PROCESS (Mode, A, B)
2   BEGIN
3      CASE Mode IS
4         WHEN "000" => saida <= A AND B;
5         WHEN "001" => saida <= A OR B;
6         WHEN "010" => saida <= A NAND B;
7         WHEN "011" => saida <= A NOR B;
8         WHEN "100" => saida <= NOT A;
9         WHEN "101" => saida <= NOT B;
10        WHEN OTHERS => saida <= '0'
11     END CASE;
12  END PROCESS porta_programavel;
```

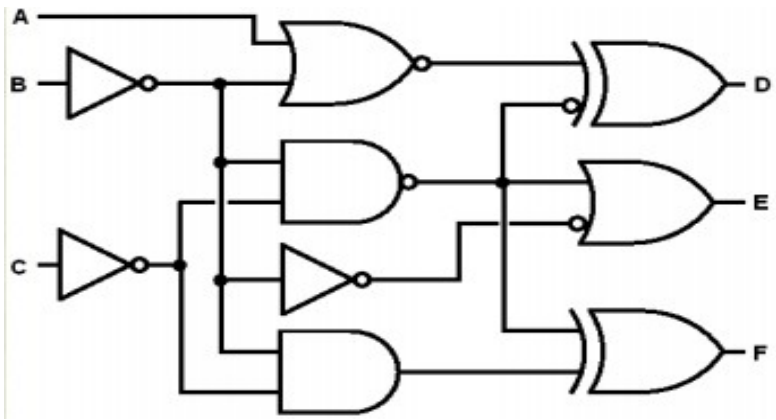
Comando null

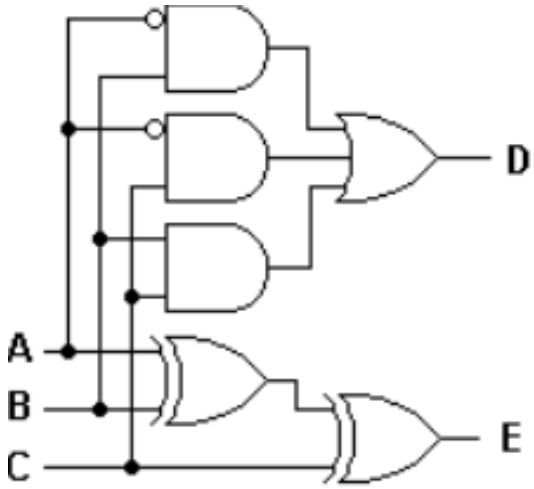
serve, por exemplo, para indicar "faça nada" em uma condição de case.

```
CASE controller_command IS
    WHEN forward => marcha <= '1';
    WHEN reverse => marcha <= '0';
    WHEN idle => NULL;
END CASE;
```

Sintaxe VHDL

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





Sintaxe VHDL

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;
```

Sintaxe VHDL

5. Projete de 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;
```