VHDL

* Objetos de Dados
* Tipos de Dados
* Tipos e Subtipos
* Atributos
* Sentenças Concorrentes e Sequenciais
* Procedimetos e Funções
* Pacotes e Bibliotecas
* Generics
* Tipos de Atraso

Objetos em VHDL

* Há quatro tipos de objetos em VHDL:

  - Constantes
  - Sinais
  - Variáveis
  - Arquivos

* Declarações de arquivo tornam um arquivo disponível para uso em um projeto.
*Arquivos podem ser abertos para leitura e escrita.
* Arquivos fornecem um maneira de um projeto em VHDL se comunicar com o ambiente do computador hospedeiro.

Constantes

*Aumentam a legibilidade do código

*Permitem fácil atualização

```
CONSTANT <constant_name>  :  <type_name> := <value>;

CONSTANT PI : REAL := 3.14;
CONSTANT WIDTH : INTEGER := 8;
```

Sinais

* **Sinais são utilizados para comunicação entre componentes.**

* **Sinais podem ser interpretados com fios físicos, reais.**

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

SIGNAL enable : BIT;
SIGNAL output : bit_vector(3 downto 0);
SIGNAL output : bit_vector(3 downto 0) := "0111";
```

Variáveis

* **Variáveis são usadas apenas em processos e subprogramas (funções e procedimentos)**

* **Variáveis usualmente não estão disponíveis para múltiplos componentes e processos**

* **Todas as atribuições de variáveis tem efeito imediato.**

```
VARIABLE <nome_variavel> : <tipo> [:= <valor>];

VARIABLE opcode : BIT_VECTOR (3 DOWNTO 0) := "0000";
VARIABLE freq : INTEGER;
```

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

```
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

Escopo dos Objetos

* O VHDL limita a visibilidade dos objetos, dependendo de onde eles são declarados.

* O escopo dos objetos é definido como a seguir:

  - Objetos declarados em um pacote são globais para todas as entidades que usam aquele pacote.

  - Objetos declarados em uma entidade são globais para todas as arquiteturas que utilizam aquela entidade.

Escopo dos Objetos

- Objetos declarados em um arquitetura são disponíveis a todas as sentenças naquela arquitetura.

- Objetos declarados em um processo são disponíveis apenas para aquele processo.

* Regras de escopo se aplicam a constantes, variáveis, sinais e arquivos.

Tipos de Dados

Tipos Escalares

* **Tipos Inteiros**
  - **A variação mínima definida pelo padrão é: -2,147,483,647 a +2,147,483,647**

```
ARCHITECTURE test_int OF test IS
BEGIN
        PROCESS (X)
        VARIABLE a: INTEGER;
        BEGIN
                a := 1;  -- OK
                a := -1;  -- OK
                a := 1.0;  -- bad
        END PROCESS;
END TEST;
```

Tipos Escalares

* **Tipos Reais**

- A faixa mínima definida pelo padrão é: -1.0E38 a +1.0E38

```
ARCHITECTURE test_real OF test IS
BEGIN
        PROCESS (X)
        VARIABLE a: REAL;
        BEGIN
                a := 1.3;  -- OK
                a := -7.5;  -- OK
                a := 1;  -- bad
                a := 1.7E13;  -- OK
                a := 5.3 ns;  -- bad
        END PROCESS;
END TEST;
```

Tipos Escalares

* **Tipos Enumerados**
  - **É uma faixa de valores definida pelo usuário**

```
TYPE binary IS ( ON, OFF );
....
ARCHITECTURE test_enum OF test IS
BEGIN
        PROCESS (X)
        VARIABLE a: binary;
        BEGIN
                a := ON; -- OK
                 .....
                a := OFF;  -- OK
                ....
        END PROCESS;
END TEST;
```

Tipos Escalares

* **Tipos Físicos:**
  - **Podem ter os valores definidos pelo usuário.**

```
TYPE resistence IS RANGE 0 to 1000000
UNITS
        ohm;  -- ohm
        Kohm = 1000 ohm;  -- 1 K
        Mohm = 1000 kohm;  -- 1 M
END UNITS;
```

- **Unidades de tempo são os únicos tipos físicos pré-definidos em VHDL.**

Tipos Escalares

As unidades de tempo pré-definidas são:

```
TYPE TIME IS RANGE -2147483647 to 2147483647
UNITS
        fs;                 -- femtosegundo
        ps = 1000 fs;       -- picosegundo
        ns = 1000 ps;       -- nanosegundo
        us = 1000 ns;       -- microsegundo
        ms = 1000 us;       -- millisesegundo
        sec = 1000 ms;      -- segundo
        min = 60 sec;       -- minuto
        hr = 60 min;        -- hora
END UNITS;
```

Tipos Compostos

* **Tipo Array:**
  - **Usados para colecionar um ou mais elementos de um mesmo tipo em uma única construção.**
  - **Elementos podem ser de qualquer tipo VHDL.**

```
TYPE data_bus IS ARRAY (0 TO 31) OF BIT;
0 ...element numbers...31
0 ...array values...1

SIGNAL  X:  data_bus;
SIGNAL Y:  BIT;
          Y <= X(12);  -- Y recebe o valor do 13o elemento
```

Tipos Compostos

* Outro exemplo de vetor uni-dimensional (usando a ordenação DOWNTO)

```
TYPE register IS ARRAY (15 DOWNTO 0) OF BIT;
15 ...element numbers... 0
0 ...array values... 1

Signal X:  register;
SIGNAL Y:  BIT;
Y <= X(4);  -- Y recebe o valor do  5o elemento
```

* A palavra DOWNTO ordena os elementos da esquerda para a direita, com elementos de índice decrescente.

Tipos Compostos

* **Arranjos bi-dimensionais são úteis para a descrição de tabelas da verdade.**

```
TYPE truth_table IS ARRAY(0 TO 7, 0 TO 4) OF BIT;
CONSTANT full_adder: truth_table := (
        "000_00",
        "001_01",
        "010_01",
        "011_10",
        "100_01",
        "101_10",
        "110_10",
        "111_11");
```

Tipos Compostos

* **Tipos Record**
  - **Usados para colecionar um ou mais elementos de diferentes tipos em uma única construção**
  - **Elementos podem ser qualquer tipo VHDL**
  - **Os elementos são acessados através no nome do campo**

```
TYPE binary IS ( ON, OFF );
TYPE switch_info IS
RECORD
        status : binary;
        IDnumber : integer;
END RECORD;
VARIABLE switch : switch_info;
        switch.status := on;  -- estado da chave
        switch.IDnumber := 30;  -- número da chave
```

Tipo Access

* **Access**

  - **Similar aos ponteiros em outras linguagens**

  - **Permitem a alocação dinâmica de memória**

  - **Úteis para a implementação de filas, fifos, etc.**

Subtipos

* **Subtipos**
  - **Permitem o uso de restrições definidas pelo usuário em um certo tipo de dado.**
  - **Podem incluir a faixa inteira de um tipo básico**
  - **Atribuições que estão fora da faixa definida resultam em um erro.**

```
SUBTYPE <name> IS <base type> RANGE <user range>;

SUBTYPE  first_ten  IS  integer  RANGE 1 to 10;
```

Subtipos

```
TYPE byte IS bit_vector(7 downto 0);

signal x_byte: byte;
signal y_byte: bit_vector(7 downto 0);

IF  x_byte = y_byte THEN ...
```


O compilador gera um erro.

O compilador não gera nenhum erro.

```
SUBTYPE byte IS bit_vector(7 downto 0)

signal x_byte: byte;
signal y_byte: bit_vector(7 downto 0);

IF  x_byte = y_byte THEN ...
```

Somador de 4 bits

```
1   ENTITY fig6_22 IS
2   PORT(
3      cin   :IN BIT;
4      a     :IN BIT_VECTOR(3 DOWNTO 0);
5      b     :IN BIT_VECTOR(3 DOWNTO 0);
6      s     :OUT BIT_VECTOR(3 DOWNTO 0);
7      cout  :OUT BIT);
8   END fig6_22;
9
10  ARCHITECTURE a OF fig6_22 IS
11  SIGNAL c :BIT_VECTOR (4 DOWNTO 0);  -- carries exigem matrizes de 5 bits
12
13  BEGIN
14     c(0) <= cin;          -- Lê carry de entrada na matriz de bits
15     s <= a XOR b  XOR c(3 DOWNTO 0); -- Gera soma dos bits
16     c(4 DOWNTO 1) <=      (a AND b)
17                    OR     (a AND c(3 DOWNTO 0))
18                    OR     (b AND c(3 DOWNTO 0));
19     cout <= c(4);         -- leva para a saída o carry do MSB.
20  END a;
```

FIGURA 6.22
Somador em VHDL.

Resumo

* O VHDL tem diversos tipos de dados disponíveis para o projetista.
* Tipos enumerados são definidos pelo usuário
* Tipos físicos representam quantidades físicas
* Os arranjos contém um número de elementos do mesmo tipo ou subtipo.
* Os records podem conter um número de elementos de diferentes tipos ou subtipos.
* O tipo access é basicamente um ponteiro.
* Subtipos são restrições definidas pelo usuário para um tipo básico.

Atributos

* Atributos definidos na linguagem retornam informação sobre certos tipos em VHDL.

  - Tipos, subtipos
  - Procedimentos, funções
  - Sinais, variáveis,constantes
  - Entidades, arquiteturas, configurações, pacotes
  - Componentes

* O VHDL tem diversos atributos pré-definidos que são úteis para o projetista.

* Atributos podem ser definidos pelo usuários para lidar com registros definidos pelo usuário, etc.

Atributos de Sinal

* A forma geral de um atributo é:

<nome> ' <identificador_de_atributo>

* Alguns exemplos de atributos de sinal

X'EVENT -- avaliado como VERDADEIRO quando um evento no sinal X acabou de ocorrer

X'LAST_VALUE - retorna o último valor do sinal X

X'STABLE(t) - avaliado com VERDADEIRO quando nenhum evento ocorrreu no sinal X há pelo menos *t* segundos.

Atributos

'LEFT - retorna o valor mais a esquerda de um tipo

'RIGHT -- retorna o valor mais a direita de um tipo

'HIGH -- retorna o maior valor de um tipo

'LOW -- retorna o menor valor de um tipo

'LENGTH - retorna o número de elementos de um vetor

'RANGE - retorna a faixa de valores de um vetor

Exemplos de Atributos

```
TYPE count is RANGE 0 TO 127;
TYPE states IS (idle, decision,read,write);
TYPE word IS ARRAY(15 DOWNTO 0) OF bit;
```

count'left = 0	states'left = idle	word'left = 15
count'right = 127	states'right = write	word'right = 0
count'high = 127	states'high = write	word'high = 15
count'low = 0	states'low = idle	word'low = 0
count'length = 128	states'length = 4	word'length = 16

```
count'range = 0 TO 127
word'range = 15 DOWNTO 0
```

Exemplo de Registrador

* Este exemplo mostra como atributos podem ser usados na descrição de um registrador de 8 bits.
* Especificações
  - Disparado na subida do relógio
  - Armazena apenas se ENABLE for alto.
  - Os dados tem um tempo de "setup" de 5 ns.

```
ENTITY 8_bit_reg IS
        PORT (enable, clk : IN std_logic;
                    a : IN std_logic_vector (7 DOWNTO 0);
                    b : OUT std_logic_vector (7 DOWNTO 0);
        END 8_bit_reg;
```

Exemplo de Registrador

* Um sinal do tipo std_logic pode assumir os seguintes valores: 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', or '-'
* O uso de 'STABLE detecta violações de "setup"

```
ARCHITECTURE first_attempt OF 8_bit_reg IS
            BEGIN
                PROCESS (clk)
                BEGIN
                  IF (enable = '1') AND a'STABLE(5 ns) AND
                      (clk = '1') THEN
                          b <= a;
                  END IF;
                END PROCESS;
            END first_attempt;
```

* O que acontece se clk for 'X'?

Exemplo de Registrador

* O uso de 'LAST_VALUE assegura que o relógio está saindo de um valor '0'

```
ARCHITECTURE behavior OF 8_bit_reg IS
BEGIN
        PROCESS (clk)
        BEGIN
                IF (enable ='1') AND a'STABLE(5 ns) AND
                  (clk = '1') AND (clk'LASTVALUE = '0') THEN
                  b <= a;
                END IF;
        END PROCESS;
END behavior;
```

Sentenças Seqüenciais e Concorrentes

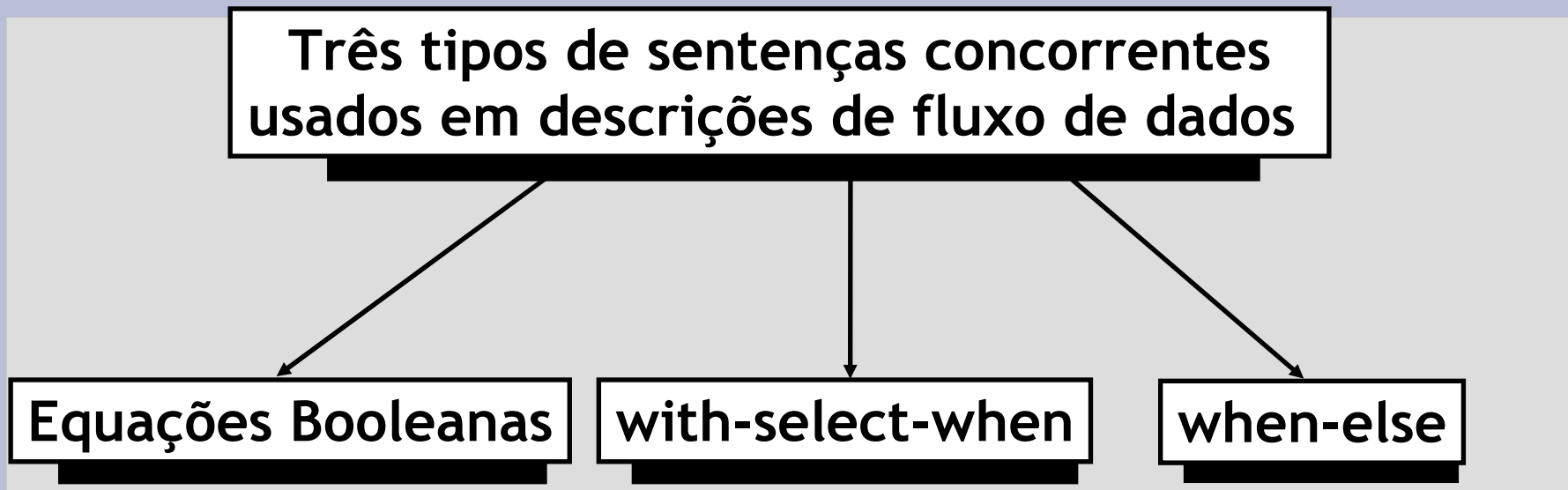
* **O VHDL provê dois tipos de execução:** ***Sequencial e Concorrente.***

* **Tipos diferentes de execução são úteis para a modelagem de circuitos reais.**

* **As sentenças sequenciais enxergam os circuitos do ponto de vista do programador.**

* **As sentenças concorrentes tem ordenação independente e são assíncronas.**

Sentenças Concorrentes



Para atribuições concorrentes de sinais

Para atribuições seletivas de sinais

Para atribuições condicionais de sinais

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;
```

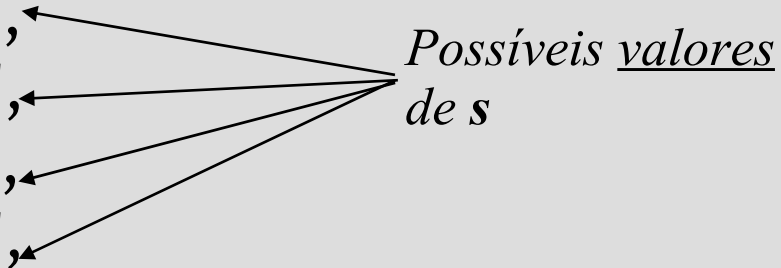
With-select-when

```
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;
```

with-select-when

```
architecture mux_arch of mux is
begin
with s select
        x   <=  a when "00",
                b when "01",
                c when "10",
                d when "11",
                "--" when others;
end mux_arch;
```

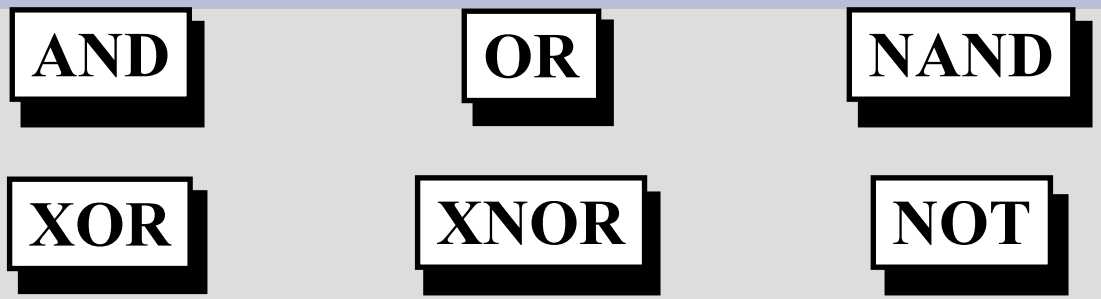


when-else

```
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

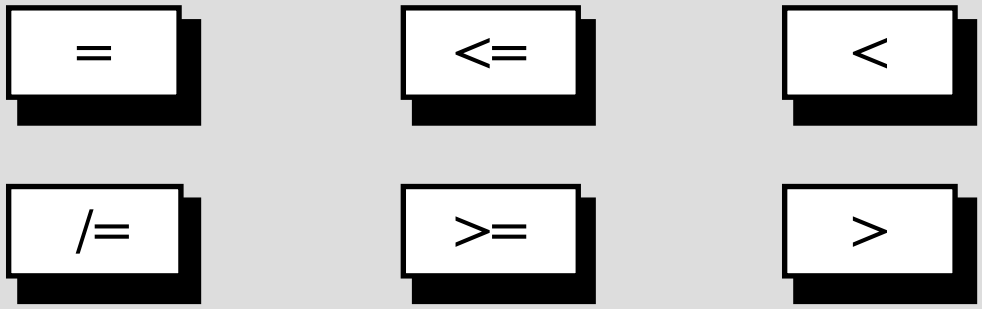
Operadores Lógicos



* Pré-definidos para os tipos:
  - Bit e boolean.
  - Vetores unidimensionais de bits e boolean.

* Operadores lógicos NÃO TEM ordem de precedência:
  X <= A **or** B **and** C
  resultará em erro de compilação.

Operadores Relacionais



* Usados para testar igualdade, diferença e ordenamento.

* (= and /=) são definidos para todos os tipos.

* (<, <=, >, and >=) são definidos para tipos escalares.

* Os tipo de operando em uma operação relacional devem ser iguais.

Operadores Aritméticos

Operadores de Adição

+ - &

Operadores de Multiplicação

* / rem mod

Outros

abs **

Sentenças Seqüenciais

Sentenças seqüenciais são contidas em processos, funções ou procedimentos.

Dentro de um processo a atribuição de um sinal é seqüencial do ponto de vista da simulação.

A ordem na qual as atribuições de sinais são feitas AFETAM o resultado.

Exemplos

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Reg IS
  PORT(Data_in : IN  STD_LOGIC_VECTOR;
       Data_out: OUT STD_LOGIC_VECTOR;
       Wr      : IN  STD_LOGIC ;
       Reset   : IN  STD_LOGIC ;
       Clk     : IN  STD_LOGIC);
END Reg;

ARCHITECTURE behavioral OF Reg IS
BEGIN
  PROCESS(Wr,Reset,Clk)
    CONSTANT Reg_delay: TIME := 2 ns;
    VARIABLE BVZero: STD_LOGIC_VECTOR(Data_in'RANGE):= (OTHERS => '0');
```

Exemplos

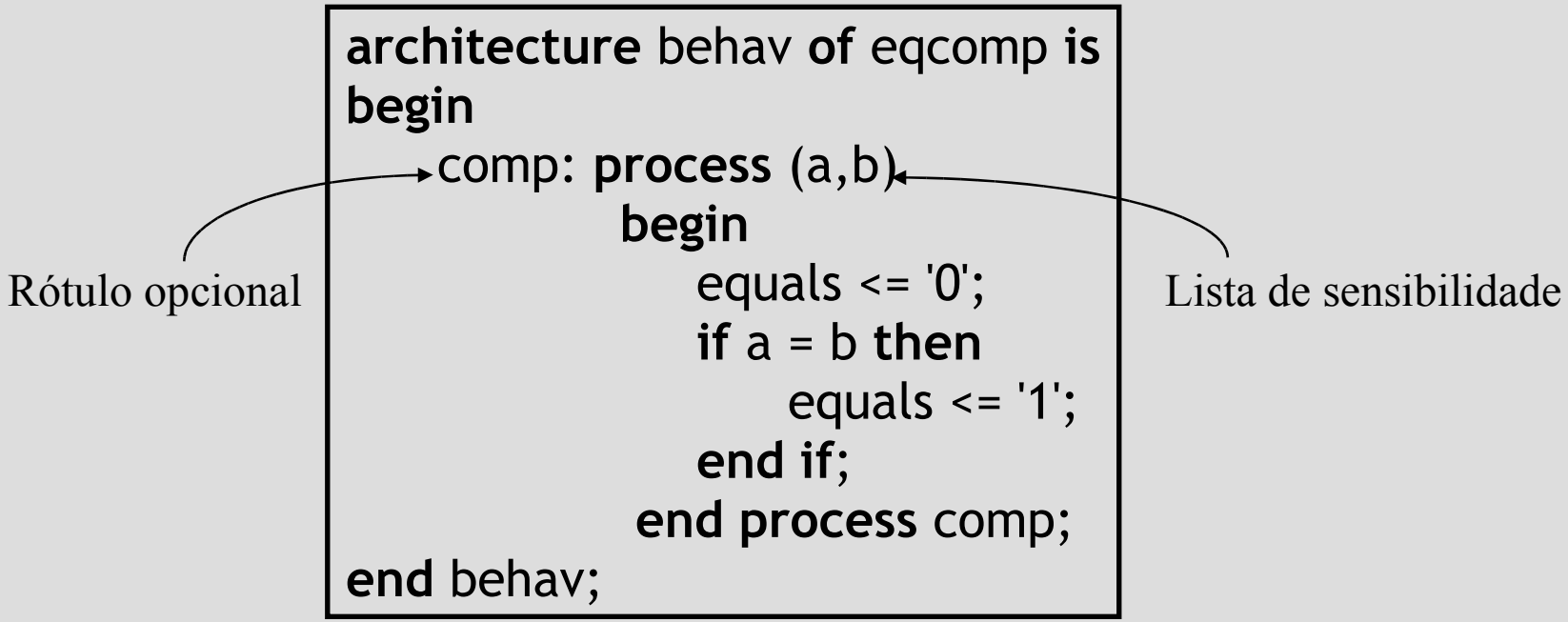
```
  BEGIN
    IF (Reset = '1') THEN
       Data_out <= BVZero AFTER Reg_delay;
    END IF;

    IF (Clk'EVENT AND Clk = '1' AND Wr = '1') THEN
       Data_out <= Data_in AFTER Reg_delay;
    END IF;
  END PROCESS;
END behavioral;
```

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.**



Processo

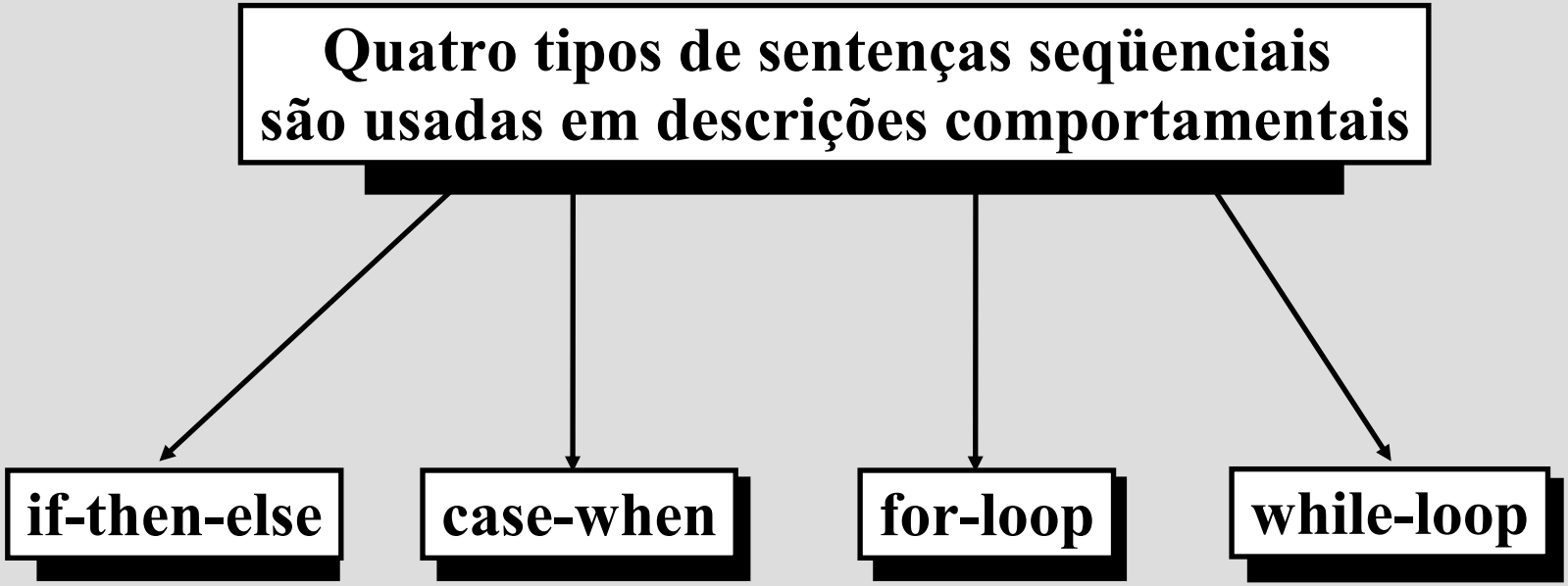
O uso do comando wait

```
Proc1: process (a,b,c)
       begin
            x <= a and b and c;
       end process;
```

Equivalentes

```
Proc2: process
       begin
            x <= a and b and c;
            wait on a, b, c;
       end process;
```

Sentenças Seqüenciais



if-then-else

```
signal step: bit;
signal addr: bit_vector(0 to 7);
          :
          :
p1: process (addr)
    begin
        if addr > x"0F" then
            step <= '1';
        else
            step <= '0';
        end if;
    end process;
```

```
signal step: bit;
signal addr: bit_vector(0 to 7);
          :
          :
p2: process (addr)
    begin
        if addr > x"0F" then
            step <= '1';
        end if;
    end process;
```

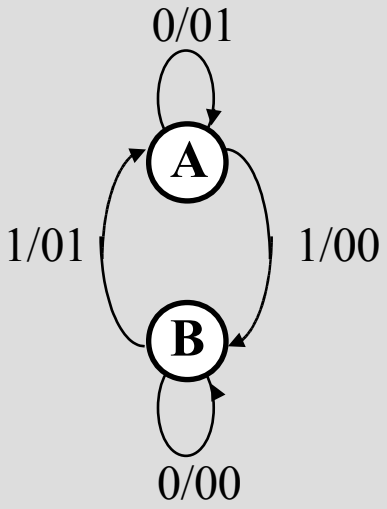
P2 tem uma memória implícita

if-then-else

```
architecture mux_arch of mux is
begin
mux4_1: process (a,b,c,d,s)
            begin
                if  s = "00" then
                    x <= a;
                elsif  s = "01" then
                    x <= b;
                elsif  s = "10" then
                    x <= c;
                else
                    x <= d;
                end if;
             end process;
end mux_arch;
```

case - when

```
case present_state is
     when  A  =>  y <= '0'; z <= '1';
          if  x = '1'  then
               next_state <= B;
          else
               next_state <= A;
          end if;
     when B  =>  y <= '0'; z <= '0';
          if  x = '1'  then
               next_state  <= A;
          else
               next_state <= B;
          end if;
end case;
```



entradas: x
saídas: y,z

Detector de Moeda

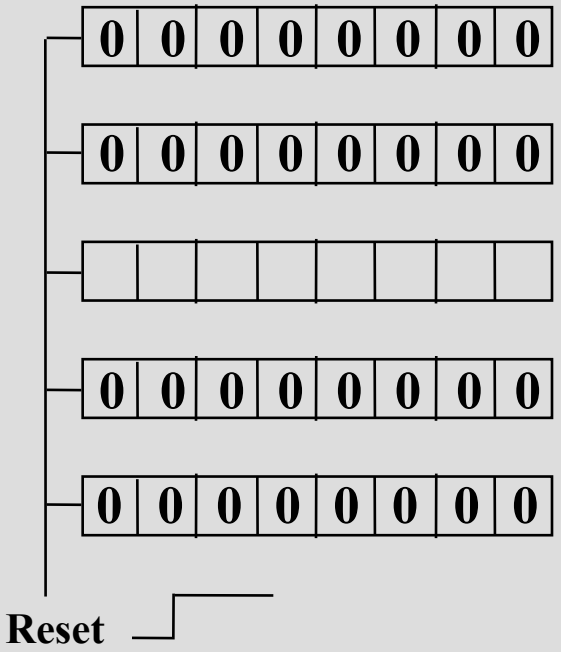
```
ENTITY   fig4_64 IS
PORT( q, d, n:IN BIT;                          -- quarter, dime, nickel
      cents  :OUT INTEGER RANGE 0 TO 25);  -- valor binário das moedas
END fig4_64;
ARCHITECTURE detector of fig4_64 IS
   SIGNAL   moedas:BIT_VECTOR(2 DOWNTO 0);-- grupo de sensores de moedas
   BEGIN
      moedas <= (q & d & n);                   -- atribui sensores para o grupo
      PROCESS (moedas)
         BEGIN
            CASE (moedas) IS
               WHEN "001"  => cents <= 5;
               WHEN "010"  => cents <= 10;
               WHEN "100"  => cents <= 25;
               WHEN others => cents <= 0;
            END CASE;
         END PROCESS;
   END detector;
```

FIGURA 4.64
Um detector de moeda em VHDL.

for-loop

```
type register is bit_vector(7 downto 0);
type reg_array is array(4 downto 0) of register;
signal fifo: reg_array;

process (reset)
begin
      if reset = '1' then
            for i in 4 downto 0 loop
                  if i = 2 then
                        next;
                  else
                        fifo(i) <= (others => '0');
                  end if;
            end loop;
      end if;
end process;
```



while-loop

```
type register is bit_vector(7 downto 0);
type reg_array is array(4 downto 0) of register;
signal fifo: reg_array;

process (reset)
     variable i: integer := 0;
begin
     if reset = '1' then
          while i <= 4 loop
               if  i /= 2 then
                    fifo(i) <= (others => '0');
               end if;
               i := i + 1;
          end loop;
     end if;
end process;
```