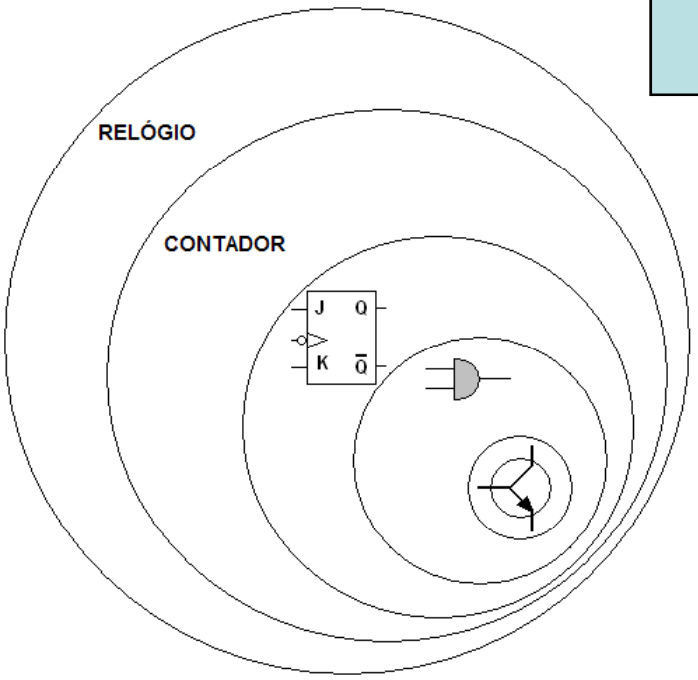
O QUE É UM SISTEMA?

Executa uma função com base em estímulos internos e/ou externos

Interligação de “módulos” que podem ser elementares ou não.

Descrito de forma **Hierarquizada**, em camadas, a partir de funções mais elementares



Vários níveis de complexidade

Modelação com diferentes níveis de abstracção

Equações Diferenciais

SISTEMA GENÉRICO...

ENTRADAS
SAÍDAS
TRANSFORMAÇÃO

A Transformação define o comportamento...

Um sistema pode ser decomposto em sub-sistemas ou MÓDULOS

Sistema Digital como caso particular de um sistema.

É neste contexto que as linguagens de descrição para Hardware aparecem

VHDL – VHSIC @HDL

Presentemente as ferramentas de *Software* são uma componente essencial no projecto de **sistemas digitais.**

-Melhoram a produtividade
-Capacidade de prever o comportamento final

VHDL – VHSIC @HDL

VHSIC-HDL
Very High Speed Integrated Circuit Hardware Description Language

Desenvolvido nos meados dos anos 80 pelo **DoD** e **IEEE**

VHDL

Começou por ser uma linguagem para modelação e documentação.
Presentemente existem ferramentas de síntese baseadas em VHDL.

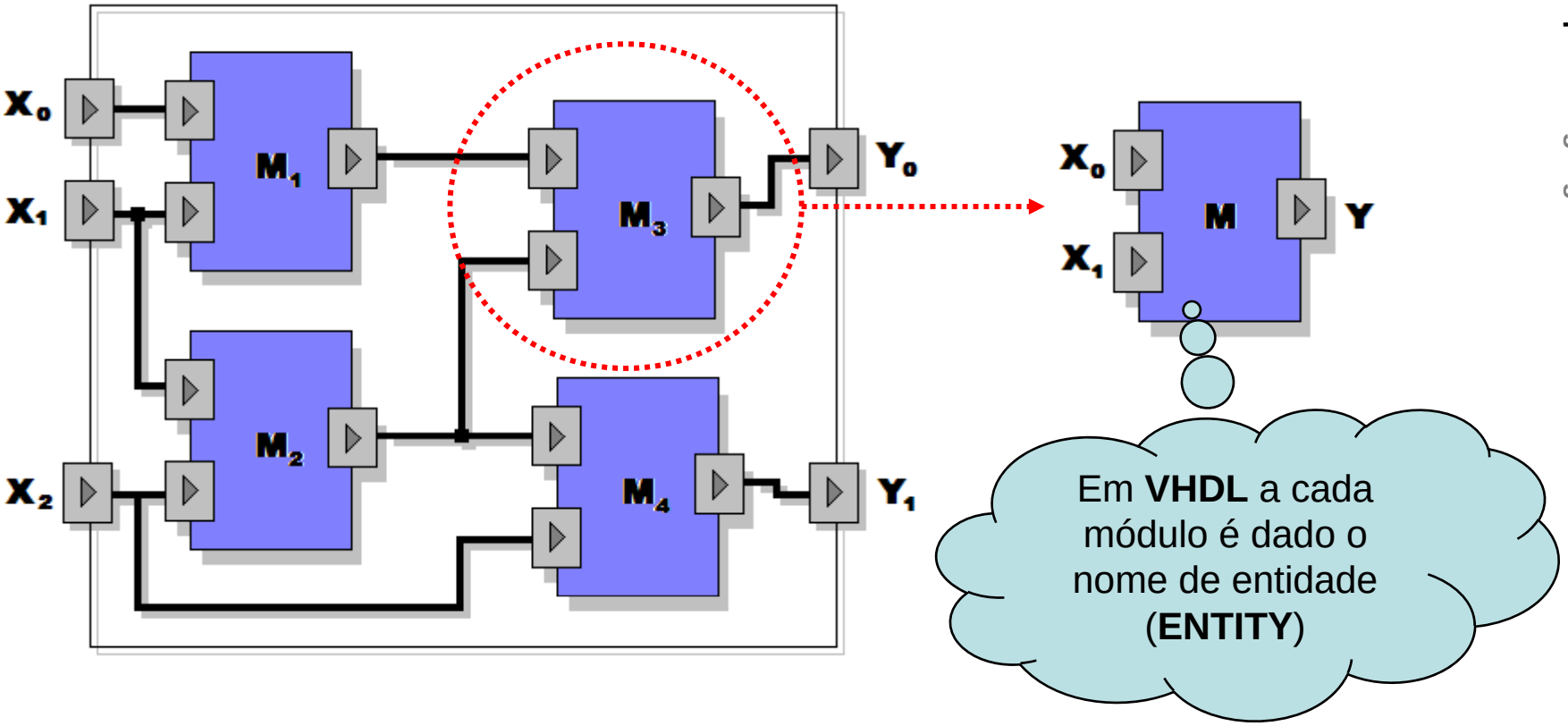
- Linguagem muito Descritiva
-Com reminiscências de Pascal e Ada
-CASE INSENSITIVE

- Decomposição Hierárquica do projecto
- Cada elemento possui uma **Interface** e um **Comportamento**
- O comportamento pode ser especificado por um algoritmo ou estrutura
- Temporização, concorrência e sequenciação podem ser modelados
- As operações lógicas podem ser simuladas

- Presentemente apenas definida para sistemas digitais
- Alguns esforços para estender para sistemas analógicos e híbridos

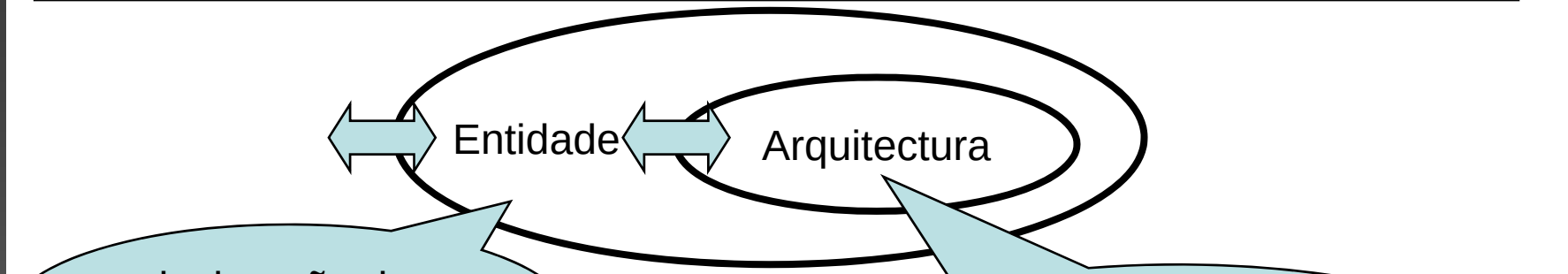
ENTIDADE vs. ARQUITECTURA

Sistema Decomposto em MÓDULOS



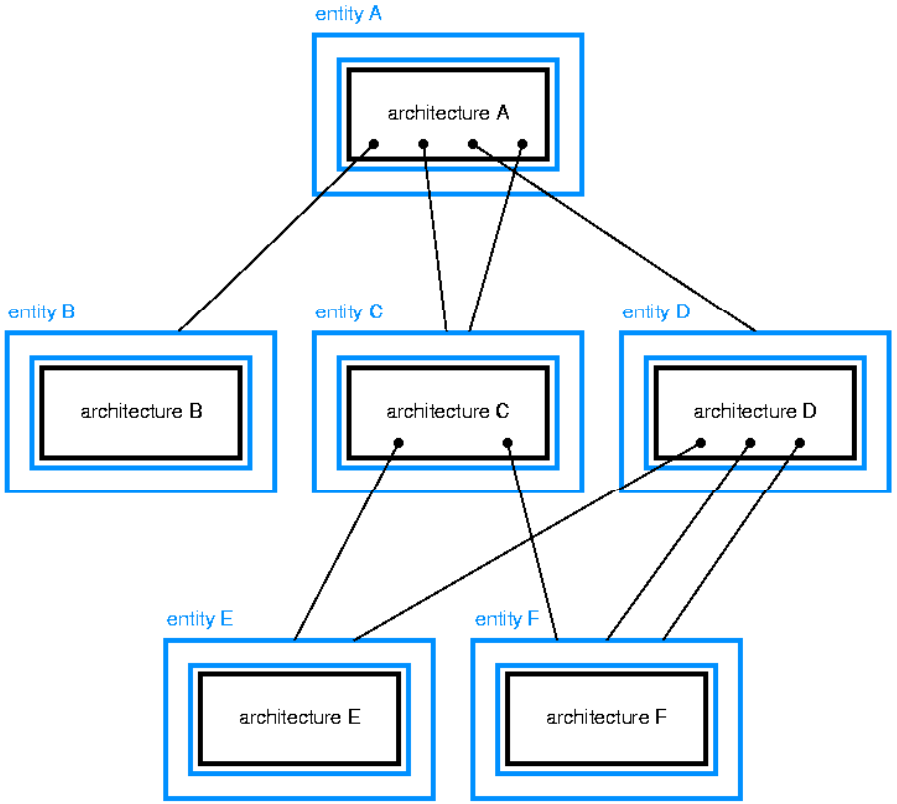
- As entradas e saídas das entidades são designadas por portas (**PORT**).
- Cada sub-Módulo é uma instância de uma entidade
- Cada instância é uma entidade.
- As entidades estão ligadas por sinais (**SIGNAL**)

ENTIDADE vs. ARQUITECTURA



declaração das entradas/saídas de um módulo

descrição detalhada da estrutura interna do módulo e seu comportamento



- Uma arquitectura pode usar várias entidades
- Diversas arquitecturas podem usar a mesma entidade

Estrutura Hierárquica de Desenho

Componentes de um projecto **VHDL**

PACKAGE
ENTITY
ARCHITECTURE
CONFIGURATION

Pacote: constantes, bibliotecas;
Entidade: pinos de entrada e saída;
Arquitectura: implementações do projeto;
Configuração: define as arquiteturas que serão utilizadas.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;
```

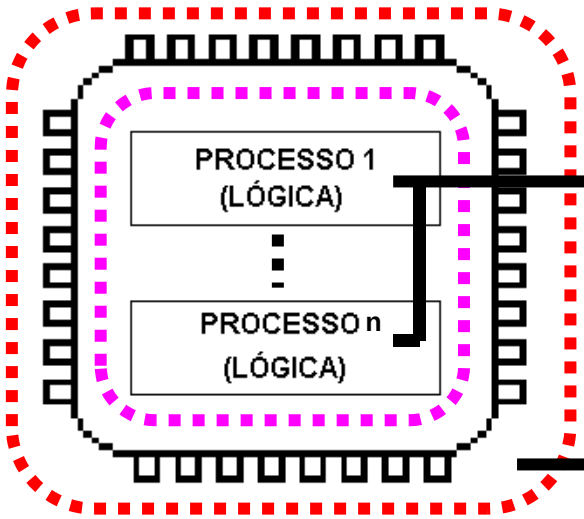
PACKAGE (BIBLIOTECAS)

```
ENTITY exemplo IS
PORT (

        < descrição dos pinos de entrada e saída >

    );
END exemplo;
```

ENTITY (PINOS DE I/O)

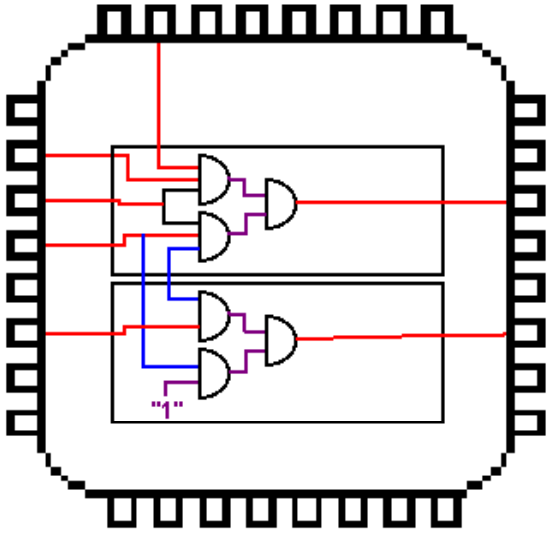
```
ARCHITECTURE teste OF exemplo IS
BEGIN
      PROCESS( <pinos de entrada e signal> )

          BEGIN

              < descrição do circuito integrado >

          END PROCESS;
END teste;
```

ARCHITECTURE (ARQUITETURA)



ARCHITECTURE
(ARQUITETURA)
PROCESSOS

ENTITY
(ENTIDADE)
PINOS DE I/O



PINOS DE
ENTRADA E SAÍDA
(I/O - INPUT/OUTPUT)

SINAL (SIGNAL)

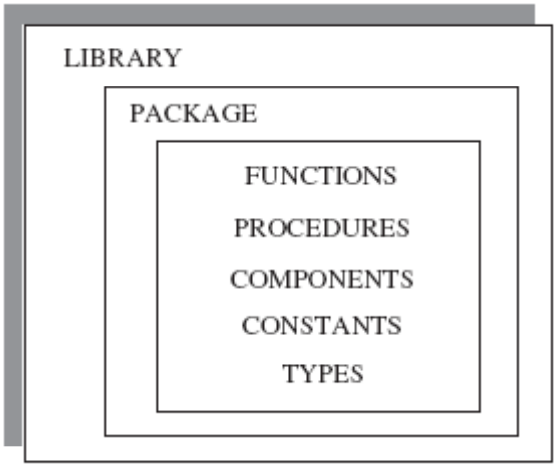
VARIÁVEL (VARIABLE)

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.all;

entity placa is
port  (
        a : in bit_vector( 6 downto 0);
        b : out bit_vector( 7 downto 0)
      );
end placa;

architecture TTL of placa is
signal pino_1 bit;
signal pino_2 bit;
signal pino_3 bit;
signal pino_4 bit;
signal pino_5 bit;

begin
        CI_X : process( a )
        begin
                <descrição do processo>
        end process CI_Y;

        CI_Y : process( a )
        begin
                <descrição do processo>
        end process CI_Z;

end TTL;
```

Package

Os pacotes (biblioteca) contém uma coleção de elementos incluindo descrição do tipos de dados.

```
LIBRARY library_name;
USE library_name.package_name.package_parts;
```



IEEE.std_logic_arith – Funções Aritméticas

IEEE.std_logic_signed – Funções Aritméticas com Sinal

IEEE.std_logic_unsigned – Funções Aritméticas sem Sinal

IEEE.std_logic_1164 – Operações lógicas

Entity

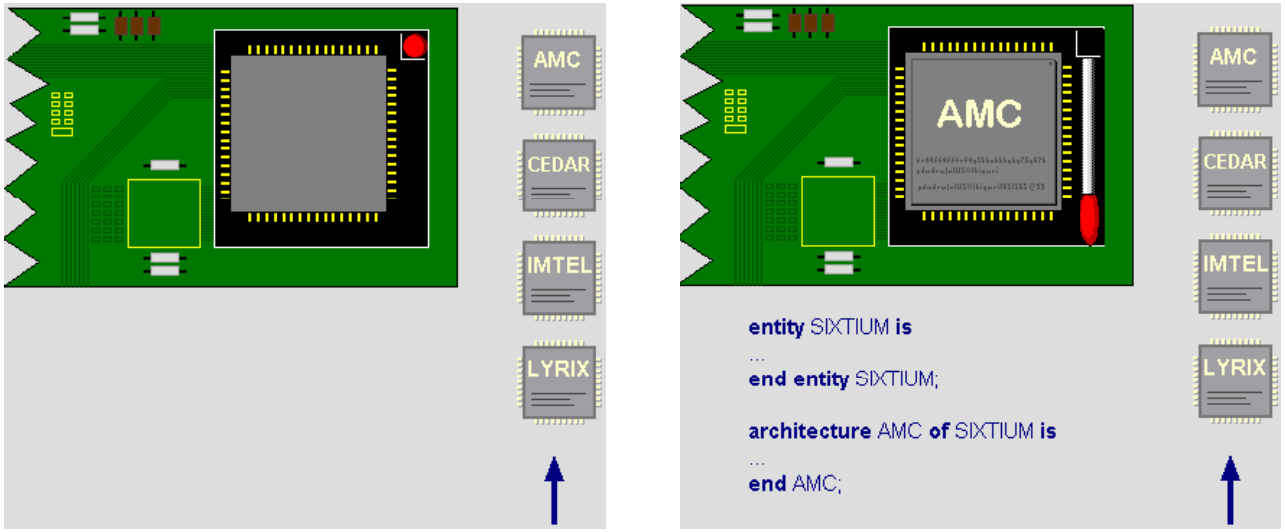
Descrição da interface de I/O do sistema com a placa.

```
ENTITY <nome> IS
PORT(
        sinal_controlo: IN <tipo>;
        entrada:        IN <tipo>;
        saída:          OUT <tipo>;
    );
END <nome>;
```

IN, OUT, INOUT, BUFFER

<tipo> : bit,
bit_vector,
std_logic_vector,
real,
inteiro, etc.

Architecture




```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Uma ARQUITECTURA possui:

-Parte DECLARATIVA (opcional) – onde, por exemplo, SINAIS, CONSTANTES e VARIÁVEIS, são declaradas

-Código – A partir de BEGIN

Arquitetura Concorrente

A arquitetura concorrente é uma forma mais complexa de descrever um sistema, geralmente apresenta várias processos dentro de uma arquitetura.

```
architecture SomeArch of SomeEnt is
begin
P1: process (A,B)
begin
  somestatement;
  somestatement;
  somestatement;
  somestatement;
end process P1;
P2: process (A,C)
begin
  somestatement;
  somestatement;
  somestatement;
end process P2;
P3: process (B)
begin
  somestatement;
  somestatement;
end process P3;
end architecture SomeArch;
```

```
architecture SomeArch of SomeEnt is
begin
P1: process (A,B)
begin
  somestatement;
  somestatement;
  somestatement;
  somestatement;
end process P1;
P2: process (A,C)
begin
  somestatement;
  somestatement;
  somestatement;
end process P2;
P3: process (B)
begin
  somestatement;
  somestatement;
end process P3;
end architecture SomeArch;
```

Tipos de Dados

BIT: assume valores ‘0’ e ‘1’

LIBRARY **std**
PACKAGE **standard**

BIT_VECTOR: é um conjunto de bits. Ex.: “010001”

BOOLEAN: assume valores *TRUE* ou *FALSE*

REAL: sempre com ponto decimal. Ex.: -3.2, 4.56, 6.0, -2.3E+2

INTEGER: Inteiros de 32 bit (de -2,147,483,647 a +2,147,483,647)

```
SIGNAL x: BIT; -- x é declarado como um sinal de um bit.

SIGNAL y: BIT_VECTOR (3 DOWNTO 0);
                -- y é um vector de 4 bit (o bit mais à esquerda
                   é o MSB)

SIGNAL w: BIT_VECTOR (0 TO 7);
                -- y é um vector de 8 bit (o bit mais à direita
                   é o MSB).
```

Afectação de Valores aos Sinais…

```
x <= '1'; --x é um sinal de 1 bit cujo valor é '1‘

y <= "0111"; --y é um sinal de 4 bit cujo valor é "0111" (MSB='0')

w <= "01110001“; --w é um sinal de 8 bit cujo valor é "01110001"
                    (MSB='1').
```

Pelica para bit e aspas para vector!!!

LIBRARY **ieee**
PACKAGE **ieee.std_logic_1164**

STD_LOGIC e STD_LOGIC_VECTOR
(‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘–’)
STD_ULOGIC e STD_ULOGIC_VECTOR
(‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘–’)

```
‘X’    Forcing Unknown (synthesizable unknown)
‘0’    Forcing Low (synthesizable logic ‘1’)
‘1’    Forcing High (synthesizable logic ‘0’)
‘Z’    High impedance (synthesizable tri-state buffer)
‘W’    Weak unknown
‘L’    Weak low
‘H’    Weak high
‘–’    Don’t care
‘U’    Unresolved
```

```
x0 <= '0'; -- bit, std_logic, ou std_ulogic com valor '0'
x1 <= "00011111"; -- bit_vector, std_logic_vector,
-- std_ulogic_vector, signed, or unsigned
x2 <= "0001_1111"; -- underscore allowed to ease visualization
x3 <= "101111" -- binary representation of decimal 47
x4 <= B"101111" -- binary representation of decimal 47
x5 <= O"57" -- octal representation of decimal 47
x6 <= X"2F" -- hexadecimal representation of decimal 47
n <= 1200; -- integer
m <= 1_200; -- integer, underscore allowed
IF ready THEN... -- Boolean, executed if ready=TRUE
y <= 1.2E-5; -- real, not synthesizable
q <= d after 10 ns; -- physical, not synthesizable
```

```
SIGNAL a: BIT;
SIGNAL b: BIT_VECTOR(7 DOWNTO 0);
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
a <= b(5); -- legal (same scalar type: BIT)
b(0) <= a; -- legal (same scalar type: BIT)
c <= d(5); -- legal (same scalar type: STD_LOGIC)
d(0) <= c; -- legal (same scalar type: STD_LOGIC)
a <= c; -- illegal (type mismatch: BIT x STD_LOGIC)
b <= d; -- illegal (type mismatch: BIT_VECTOR x STD_LOGIC_VECTOR)
e <= b; -- illegal (type mismatch: INTEGER x BIT_VECTOR)
e <= d; -- illegal (type mismatch: INTEGER x STD_LOGIC_VECTOR)
```

Physical: representam uma medida física: voltagem, capacitância, tempo
Tipos pré-definidos: fs, ps, ns, um, ms, sec, min, hr.

Intervalos: permite determinar um intervalo de utilização dentro de um determinado tipo.

range <valor_menor> **to** <valor_maior>
range <valor_maior> **downto** <valor_menor>

Array: em VHDL um array é definido como uma coleção de elementos todos do mesmo tipo.

Operadores

• **Operadores de atribuição:**

<= Usado para atribuir um valor a um SIGNAL.

:= Usado para atribuir um valor a uma VARIABLE, CONSTANT, ou GENERIC. Usada também para estabelecer os valores iniciais de sinais

=> Usado para atribuir valores a elementos individuais de um vector ou para OTHERS.

Operadores (cont)

- **operadores lógicos: and, or, nand, nor, xor, xnor e not**
- **operadores numéricos:** soma (**+**), subtracção (**-**), divisão (***/***), multiplicação (*******), módulo (**mod**), remanescente (**rem** - ex.: 6 rem 4 = 2), expoente (********), valor absoluto (**abs**)
- **operadores relacionais:** igual (**=**), diferente (***/=***), menor do que (**<**), menor ou igual (**<=**), maior do que (**>**), maior ou igual (**>=**)
- **operadores de deslocamento:** **sll** (shift left logical), **srl** (shift right logical), **sla** (shift left arithmetic), **sra** (shift right arithmetic), **rol** (rotate left logical), **ror** (rotate right logical)
- **operador de concatenação:** consiste em criar um novo vector a partir de dois vectores já existentes, por exemplo:

```
dado1 : bit_vector(0 to 7);                                          [01011011]
dado2 : bit_vector(0 to 7);                                          [11010010]
novo_dado : bit_vector(0 to 7);
novo_dado <= (dado1(0 to 1) & dado2(2 to 5) & dado1(6 to 7));    [01010011]
```

Comandos Sequênciais

- **Atribuição de Variáveis**

```
1  architecture topologia_arquitetura of teste is
2          signal A, B, J, G  : bit_vector(1 downto 0);
3          signal E, F: bit;
4  begin
5
6          process ( A, B, E, F, G, J)
7
8  variable C, D, H, Y : bit_vector(1 downto 0);
9  variable W          : bit_vector(3 downto 0);
10 variable Z          : bit_vector(7 to 0);
11 variable X          : bit;
12 variable DATA       : bit_vector(31 downto 0);
13
14         begin
15                 A <= "11", B <= "01", J <= "10", G <= "00'';
16                 E <= '0', F <= '1';
17                 C              := "01";
18                 X              := E nand F;
19                 Z(0 to 3)      := C & D;
20                 Z(4 to 0)      := (not A) & (A nor B);
21                 D              := ('1', '0');
22                 W              := (2 downto 1 => G, 3 => '1', others => '0');
23                 DATA           := (31 downto 28 => '1', others => '0');
24         end process;
25 end topologia_arquitetura;
```

- **Atribuição de Sinais**

```
1  architecture topologia_arquitetura of teste is
2          signal A, B, C : bit;
3          signal D: integer;
4  begin
5
6          process ( A, B, C, D)
7
8  variable L, M, N, Q bit;
9
10         begin
11                 A <= '0', '1' after 20ns, '0' after 40ns
12                 B <= not L;
13                 C <= ((L andM) xor (N nand Q));
14                 D <= 3, 5 after 20ns, 7 after 40ns, 9 after 60ns;
16
17         end process;
18 end topologia_arquitetura;
```

- **Diferenças entre SIGNAL e VARIABLE**
  - Quando se utiliza SIGNAL, a atribuição ocorre no final do processo, enquanto que a atribuição VARIABLE ocorre simultaneamente.
  - Nos próximos dois acetatos, mostra-se a diferença entre estas atribuições.

    <= (atribuição de signal)

    := (atribuição de variável)

A seta a vermelho mostra a execução passo-a-passo...

```
D <= 2;
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1
B = 1
C = 1
D = 2
E = 1
```

```
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1          Av = 0
B = 1          Bv = 0
C = 1
D = 2
E = 1          Ev = 0
```

```
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1          Av = 2
B = 1          Bv = 0
C = 1
D = 2
E = 1          Ev = 0
```

```
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1          Av = 2
B = 1          Bv = 3
C = 1
D = 2
E = 1          Ev = 0
```

```
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1          Av = 3
B = 1          Bv = 3
C = 1
D = 2
E = 1          Ev = 0
```

```
process (C, D)
variable Av, Bv, Ev : integer := 0;
begin
        Av := 2;
        Bv := Av + C;
        Av := D +1;
        Ev := Av * 2;
        A <= Av;
        B <= Bv;
        E <= Ev;
end process;
A = 1          Av = 3
B = 1          Bv = 3
C = 1
D = 2
E = 1          Ev = 6
```

A seta a **vermelho** mostra a execução passo-a-passo…

D <= 2;

process (C, **D**) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;**	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;**	**process** (C, D) **begin** **A <= 2**; B <= A + C; A <= D +1; E <= A * 2; **end process;**	**process** (C, D) **begin** A <= 2; **B <= A + C** A <= D +1; E <= A * 2; **end process;**
A = 1 B = 1 C = 1 D = 1 E = 1	A = 1 B = 1 C = 1 **D = 2** E = 1	A = 1 A <= 2 B = 1 C = 1 D = 2 E = 1	A = 1 A <= 2 B = 1 B <= A + C C = 1 D = 2 E = 1
process (C, D) **begin** A <= 2; B <= A + C; **A <= D +1;** E <= A * 2; **end process;**	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; **E <= A * 2;** **end process;**	**process** (C, D) **begin** A <= 2; B <= A + C; A <= D +1; E <= A * 2; **end process;**	**process** (C, D) **begin** A <= 2; B <= A + C A <= D +1; E <= A * 2; **end process;**
A = 1 A <= D + 1 B = 1 B <= A + C C = 1 D = 2 E = 1	A = 1 A <= D + 1 B = 1 B <= A + C C = 1 D = 2 E = 1 E <= A * 2;	A = 1 A <= 3 B = 1 B <= 2 C = 1 D = 2 E = 1 E <= 2;	A = **3** B = **2** **e não 3** C = 1 D = 2 E = **2** **e não 6**

A diferença entre os valores deve-se à forma de atribuição…

Atributos dos Dados

‘LOW: Retorna o índice inferior do vector
‘HIGH: Retorna o índice superior do vector
‘LEFT: Returna o índice mais à esquerda
‘RIGHT: Returna o índice mais à direita
‘LENGTH: Returna a dimensão do vector
‘RANGE: Returna a gama do vector
‘REVERSE_RANGE: Retorna a gama do vector em ordem inversa

Exemplo:

```
SIGNAL d : STD_LOGIC_VECTOR (7 DOWNTO 0);
```

d'LOW=0, d'HIGH=7, d'LEFT=7, d'RIGHT=0, d'LENGTH=8,
d'RANGE=(7 downto 0), d'REVERSE_RANGE=(0 to 7).

Atributos dos Sinais

‘EVENT: Retorna TRUE quando um evento ocorre
‘STABLE: Retorna TRUE quando nenhum evento ocorre
‘ACTIVE: Retorna TRUE se o sinal é ‘1’
‘QUIET <time>: Retorna TRUE se nenhum evento ocorreu em *time*
‘LAST_EVENT: Retorna o tempo dispendido desde o último evento
‘LAST_ACTIVE: Retorna o tempo dispendido desde o sinal =‘1’
‘LAST_VALUE: Retorna o valor do sinal desde o último evento.

Exemplo:

```
IF (clk'EVENT AND clk='1')...
IF (NOT clk'STABLE AND clk='1')...
WAIT UNTIL (clk'EVENT AND clk='1');
IF RISING_EDGE(clk)... – chamada a uma função
```

Modo Concorrente vs. Modo Sequencial

VHDL	CONCORRENTE	WHEN
		GENERATE
		AND, NOT, etc.
		BLOCK
	SEQUENCIAL	PROCESS
		FUNCTION
		PROCEDURE

Circuitos Lógicos Combinatório
=> Código Concorrente

Circuitos Lógico Sequenciais
=> Código Sequencial

CONCORRENTE: WHEN

WHEN / ELSE:

```
assignment WHEN condition ELSE
assignment WHEN condition ELSE
...;
```

WITH / SELECT / WHEN:

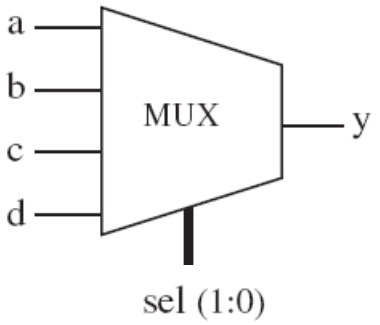
```
WITH identifier SELECT
assignment WHEN value,
assignment WHEN value,
...;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
   y: OUT STD_LOGIC);
END mux;
---------------------------------------
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel="00" ELSE
         b WHEN sel="01" ELSE
         c WHEN sel="10" ELSE
         d;
END mux1;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN STD_LOGIC_VECTOR (1 DOWNTO 0);
   y: OUT STD_LOGIC);
END mux;
---------------------------------------
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN "00",
            b WHEN "01",
            c WHEN "10",
            d WHEN OTHERS;
END mux2;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
   sel: IN INTEGER RANGE 0 TO 3;
   y: OUT STD_LOGIC);
END mux;
ARCHITECTURE mux2 OF mux IS
BEGIN
   WITH sel SELECT
      y <=  a WHEN 0,
            b WHEN 1,
            c WHEN 2,
            d WHEN 3;
END mux2;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY mux IS
   PORT ( a, b, c, d: IN STD_LOGIC;
      sel: IN INTEGER RANGE 0 TO 3;
      y: OUT STD_LOGIC);
END mux;
ARCHITECTURE mux1 OF mux IS
BEGIN
   y <=  a WHEN sel=0 ELSE
         b WHEN sel=1 ELSE
         c WHEN sel=2 ELSE
         d;
END mux1;
```



SEQUENCIAL: PROCESS

```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

> Um PROCESS deve ser instalado no código principal.

> É executado sempre que um sinal, na sua lista de sensibilidades

> Condição de WAIT é preenchida.

As VARIABLES são opcionais assim como o uso de etiquetas !!

Exemplo: Contador Mod10

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------------
ENTITY counter IS
   PORT (clk : IN STD_LOGIC;
         digit : OUT INTEGER RANGE 0 TO 9);
END counter;
-----------------------------------------
ARCHITECTURE counter OF counter IS
BEGIN
   count: PROCESS(clk)
      VARIABLE temp : INTEGER RANGE 0 TO 10;
   BEGIN
      IF (clk'EVENT AND clk='1') THEN
         temp := temp + 1;
         IF (temp=10) THEN temp := 0;
         END IF;
      END IF;
      digit <= temp;
   END PROCESS count;
END counter;
```

Comandos de Alteração do Fluxo

- Comando **WAIT**

- Este comando tem a finalidade de causar uma suspensão do processo declarado ou procedimento.
- O comando wait pode ser utilizado de quatro formas diferentes, são elas:

1	**wait until** <conditional>**;**	**wait until** CLK'**event and** CLK = '1';
2	**wait on** <signal_list>**;**	**wait on** a, b;
3	**wait for** <time>**;**	**wait for** 10ns;
4	**wait;**	**wait;**

- Comando **IF-THEN-ELSE**

MODELO	**exemplo 1:**	**exemplo 2:**

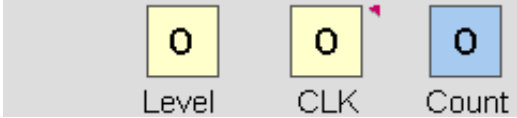
if condição_1 **then**
  <comandos>
elsif condição_2 **then**
  <comandos>
else
  <comandos>
end if;
	if (A = '0'**) then**
	  B <= "00";
	else
	  B <= "11";
	end if;
		if (CLK**'event and** CLK ='1'**) then**
		  FF <= '0';
		elsif (CLK**'event and** CLK ='0'**) then**
		  FF <= '1';
		elsif (J = '1'**) and (**K='1'**) then**
		  FM <= '1';
		end if;

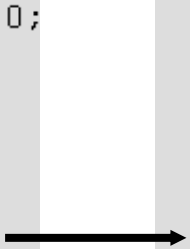
- Comando **LOOP FOR - WHILE**

<label opcional>: **for** <parâmetros> **in** <valor_final> **loop**
<seqüência de comandos>
end loop <label opcional>**;**

1	**process (A)**
2	
3	**begin**
4	Z <= “0000”;
5	**for** i **in** 0 **to** 3 **loop**
6	**if**(A = i) **then**
7	Z(i) <= ‘1’;
8	**end if;**
9	**end loop;**
10	**end process;**

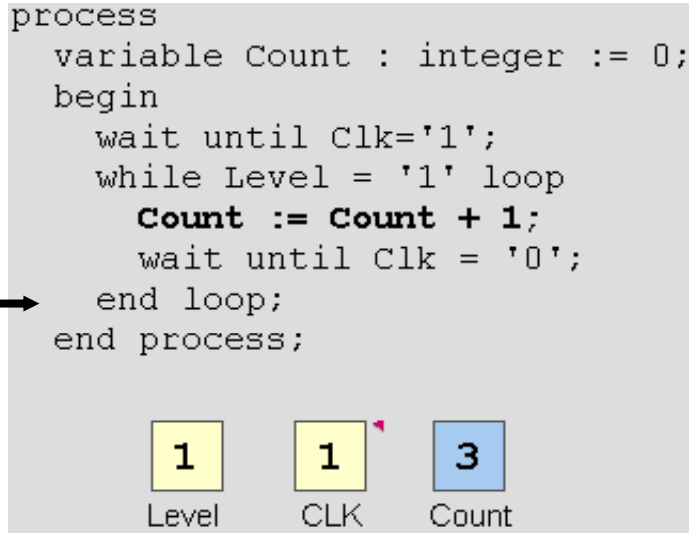
```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```



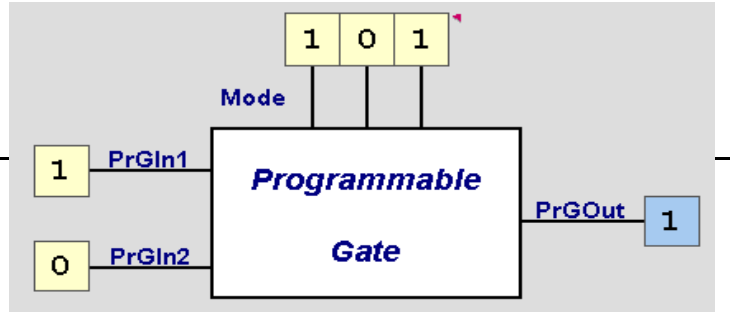


```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```

```
process
  variable Count : integer := 0;
  begin
    wait until Clk='1';
    while Level = '1' loop
      Count := Count + 1;
      wait until Clk = '0';
    end loop;
  end process;
```



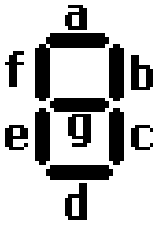
- Comando **CASE**




```
porta_programável : process (Mode, PrGIn1, PrGIn2)
        begin
                case Mode is
                        when “000”      => PrGOut   <=   PrGIn1 and PrGIn2;
                        when “001”      => PrGOut   <=   PrGIn1 or PrGIn2;
                        when “010”      => PrGOut   <=   PrGIn1 nand PrGIn2;
                        when “011”      => PrGOut   <=   PrGIn1 nor PrGIn2;
                        when “100”      => PrGOut   <=   not PrGIn1;
                        when “101”      => PrGOut   <=   not PrGIn2;
                        when  others    => PrGOut   <=   ‘0’
                end case;
end process porta_programavel;
```

Codificador BCD para sete-segmentos.

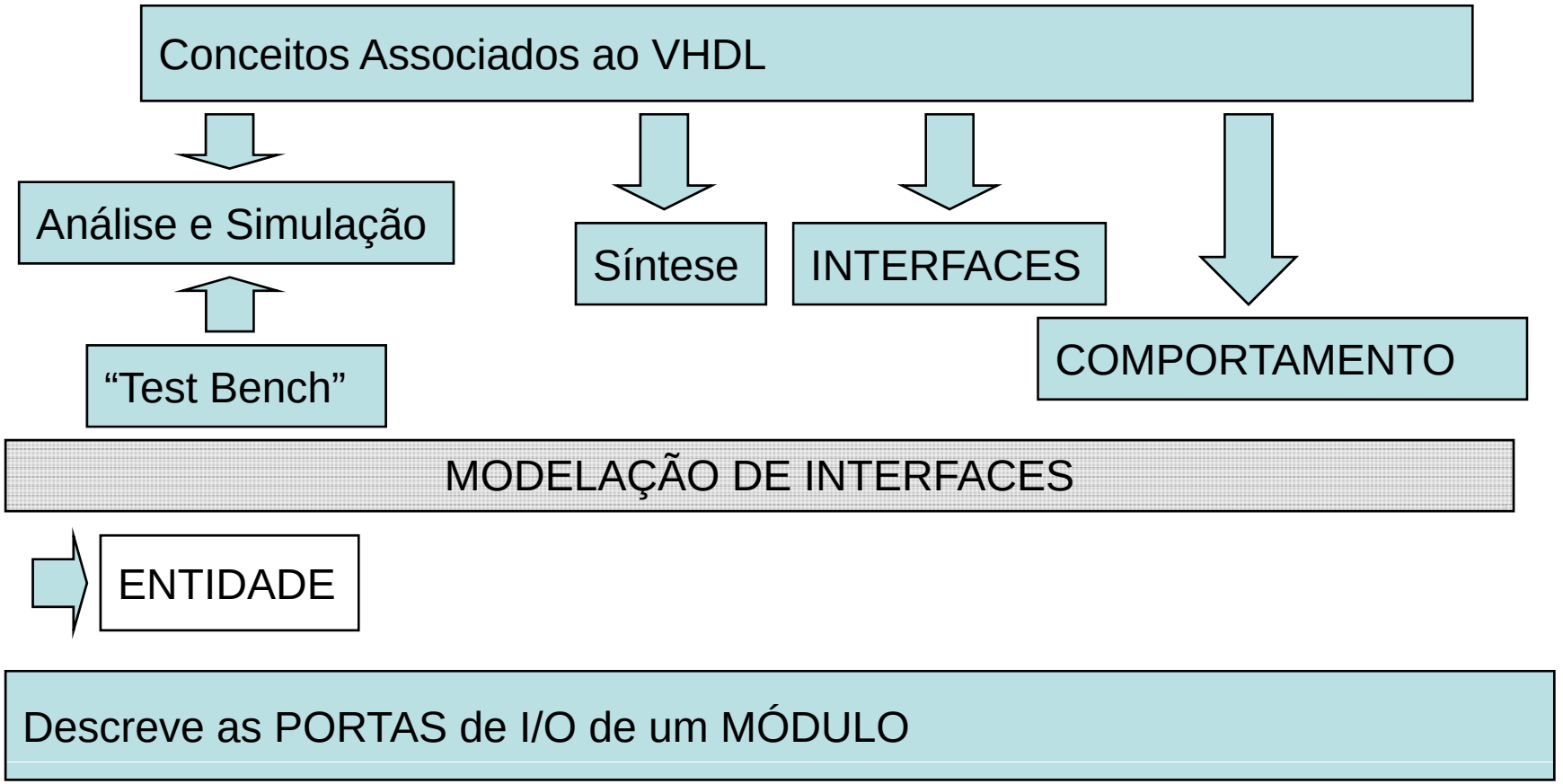
```
case codigo is
        when “0000” =>
                digito <=“0111111”;
        when “0001” =>
                digito <=“0000110”;
        when “0010” =>
                digito <=“1011011”;
        when “0011” =>
                digito <=“1001111”;
         .....
end case;
```

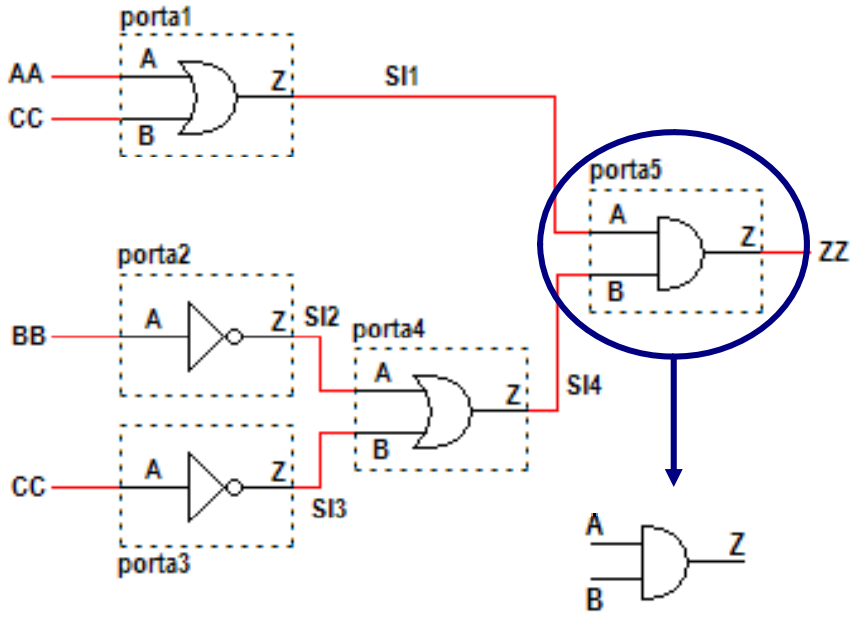


ULA (Unidade Lógica Aritmética)

```
if (mode = 1) then
        case comando is
                when “000” =>
                        resposta <= oper1 and oper2;
                when “001” =>
                        resposta <= oper1 or oper2;
                when “010” =>
                        resposta <= oper1 nor oper2;
                .....
        end case;
else
```

- Os módulos podem ser descritos de várias formas distintas
- Dependendo da aplicação pode não descrever, internamente, a forma exacta mas apenas o seu comportamento.
- O VHDL permite que o comportamento seja definido em forma de um programa executável.





nome da entidade

tipo do sinal

```
ENTITY porta_and IS
        PORT(A,B: IN BIT;Z: OUT BIT);
END porta_and
```

nome da portas

direcção do sinal

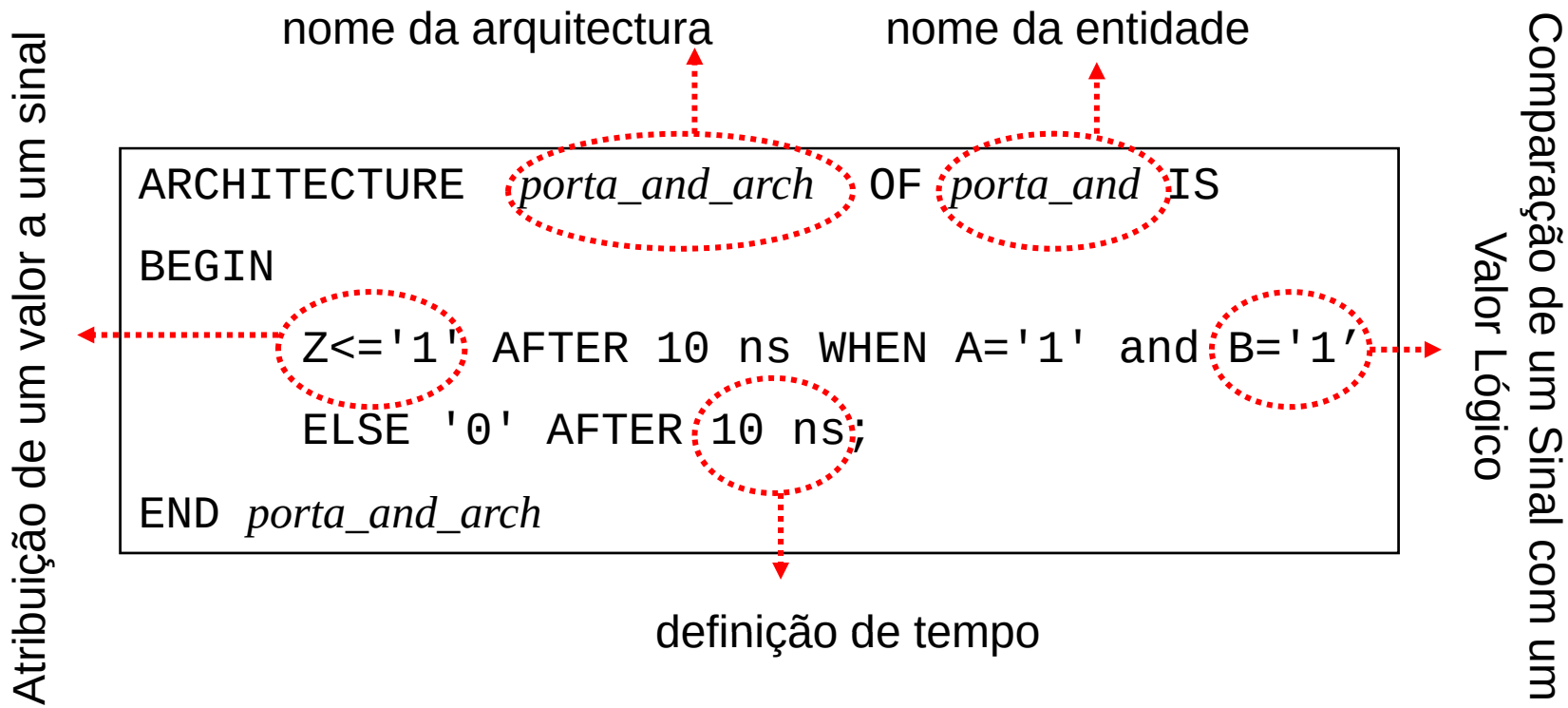
identifica o final da declaração

As palavras em maiúsculas são reservadas.

MODELAÇÃO DE COMPORTAMENTOS

ARQUITECTURA

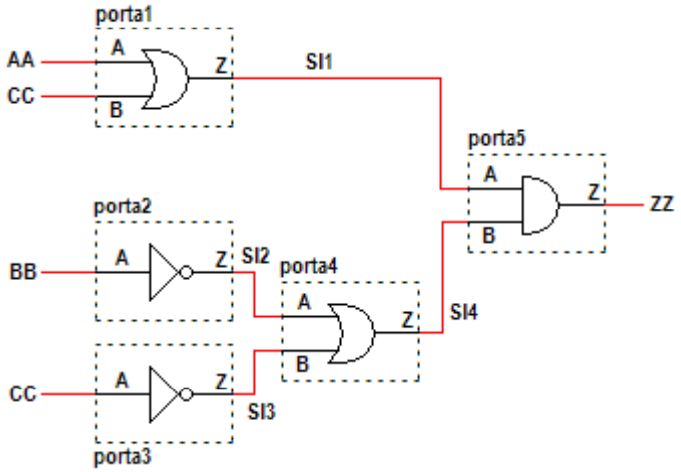
Descreve o algoritmo executado pelo MÓDULO
É dividida em duas partes principais
- Parte Declarativa
- Parte Descritiva



```
ENTITY mAND IS
GENERIC(tp:TIME:=10 ns); -- tp - tempo de propagação
PORT(A,B: in BIT; Z: out BIT);
END ENTITY mAND;
ARCHITECTURE mAND_arch of mAND IS
BEGIN
Z<='1' AFTER tp WHEN (A AND B)='1' ELSE '0' AFTER tp;
END mAND_arch;
```

```
ENTITY mOR IS
GENERIC(tp:TIME:=10 ns); -- tp - tempo de propagação
PORT(A,B: in BIT; Z: out BIT);
END ENTITY mOR;
ARCHITECTURE mOR_arch of mOR IS
BEGIN
Z<='1' AFTER tp WHEN (A OR B)='1' ELSE '0' AFTER tp;
END mOR_arch;
```

```
ENTITY mNOT IS
GENERIC(tp:TIME:=10 ns); -- tp - tempo de prop.
PORT(A: in BIT; Z: out BIT);
END ENTITY mNOT;
ARCHITECTURE mNOT_arch of mNOT IS
BEGIN
Z<='1' AFTER tp WHEN (NOT A)='1' ELSE '0' AFTER tp;
END mNOT_arch;
```



SIMULAÇÃO DO COMPONENTE

TEST BENCH

Definição dos componentes do sistema

Identifica as portas dos componentes
Parâmetros de simulação são passados como GENERIC

Cria 8 sinais do tipo booleano

Comentários após dois hífens seguidos

Cria um processo (sem lista de sensibilidades) dentro da arquitectura

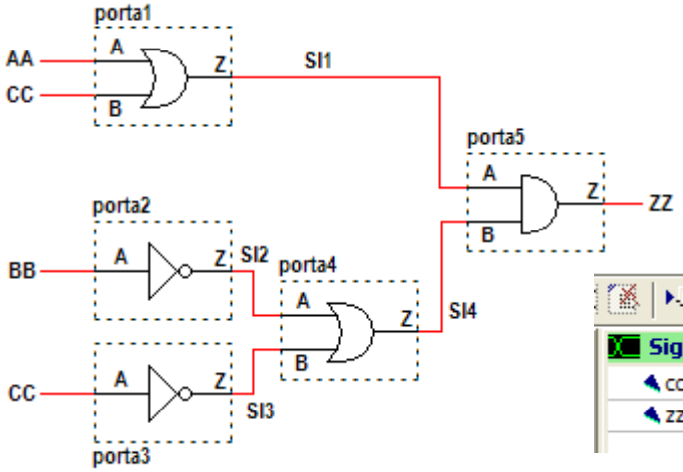
Note-se como é feita a afectação dos sinais...

Aguarda 100 nanosegundos....

```
ENTITY testbench IS
END testbench;
ARCHITECTURE testbench_arch OF testbench IS
COMPONENT mAND
          PORT(A,B: in BIT; Z: out BIT);
END COMPONENT;
COMPONENT mOR
          PORT(A,B: in BIT; Z: out BIT);
END COMPONENT;
COMPONENT mNOT
          PORT(A: in BIT; Z: out BIT);
END COMPONENT;
SIGNAL AA,BB,CC,SI1,SI2,SI3,SI4,ZZ:BIT;
BEGIN
AA<='0';
BB<='1';
-- Instanciar componentes
          porta1: mOR PORT MAP (A=>AA,B=>CC,Z=>SI1);
          porta2: mNOT PORT MAP (A=>BB,Z=>SI2);
          porta3: mNOT PORT MAP (A=>CC,Z=>SI3);
          porta4: mOR PORT MAP (A=>SI2,B=>SI3,Z=>SI4);
          porta5: mAND PORT MAP (A=>SI1,B=>SI4,Z=>ZZ);
teste:PROCESS
          BEGIN
          CC<='0';
          WAIT FOR 100 ns;
          CC<='1';
          WAIT FOR 100 ns;
          CC<='0';
          WAIT FOR 100 ns;
END PROCESS teste;
END testbench_arch;
```

Observe a figura do acetato anterior...

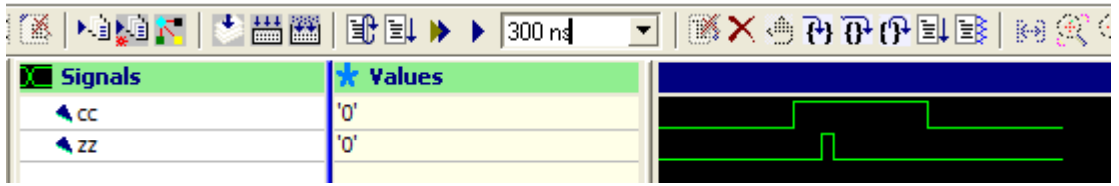
RESULTADOS DE SIMULAÇÃO



AA<=0, BB<=1

CC=0->1

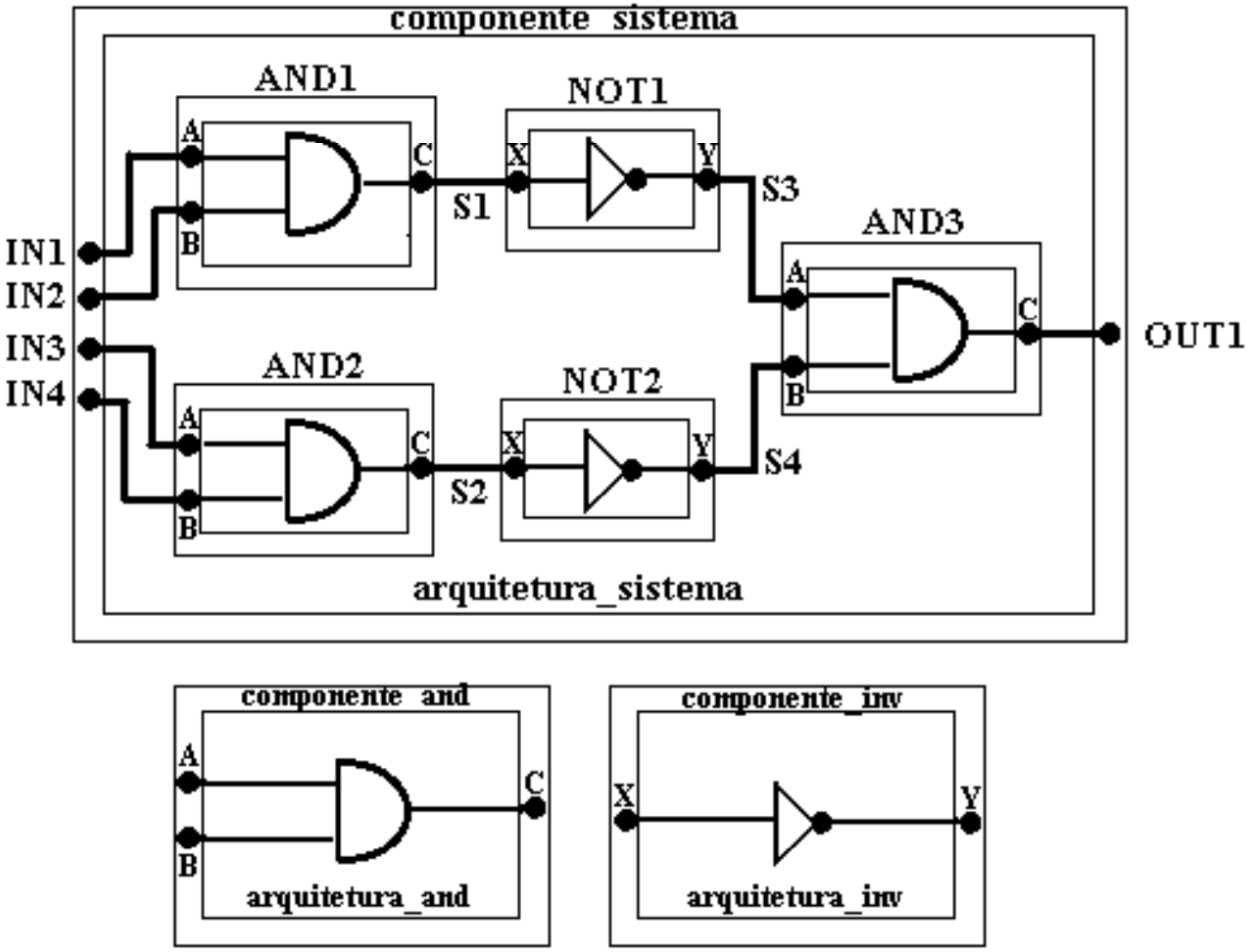
CC \ AA BB	00	01	11	10
0	0	0		
1		0	0	



Uma versão alternativa mais compacta...

```
ENTITY testbench IS
END testbench;
ARCHITECTURE testbench_arch OF testbench IS
SIGNAL A,B,C,Z,nC,nB,PSI,FI : BIT;
BEGIN
A<='0';
B<='1';
C<='0','1' AFTER 100 ns,'0' AFTER 200 ns;
          nC<=NOT C AFTER 10 ns;
          nB<=NOT B AFTER 10 ns;
          FI<=A OR C AFTER 10 ns;
          PSI<=nB OR nC AFTER 10 ns;
          Z<=FI AND PSI AFTER 10 ns;
END testbench_arch;
```

Outro Exemplo:



Programa 1

```
-------------------------------------------------------
-- Arquivo componente_inv.vhd
-- Modelo do inversor
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_inv is
port(
        x : in bit;
        y : out bit
        );
end componente_inv;

architecture arquitetura_inv of
componente_inv is

        begin
                y <= not x;
end arquitetura_inv;
```

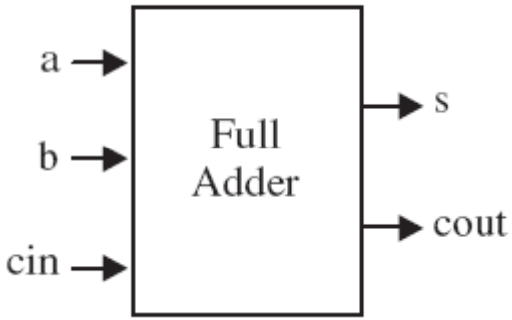
Programa 2

```
-------------------------------------------------------
-- Arquivo componente_and.vhd
-- Modelo da porta AND
-------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_and is
port(
        a : in bit;
        b : in bit;
        c : out bit
        );
end componente_and;

architecture arquitetura_and of componete_and is

        begin
                c <= a and b;
end arquitetura_and;
```

Programa 3

```
-----------------------------------------------------
-- Arquivo componente_sistema.vhd
-- Modelo da porta AND
-----------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;

entity componente_sistema is
port(
        in1 : in bit;
        in2 : in bit;
        in3 : in bit;
        in4 : in bit;
        out1 : out bit
        );
end componente_sistema;

architecture arquitetura_sistema of componente_sistema is

component componente_and
                port( a: in bit; b : in bit; c : out bit);
end component;

component componente_inv
                port( x: in bit; y : out bit);
end component;

signal s1, s2, s3, s4 : bit;

        begin
                and1 : componente_and port map (a => in1, b => in2, c => s1);
                and2 : componente_and port map (a => in3, b => in4, c => s2);
                and3 : componente_and port map (a => s3, b => s4, c => ut1);
                inv1 : componente_inv port map (x => s1, y => s3);
                inv2 : componente_inv port map (x => s2, y => s4);
end arquitetura_sistema;
```

Exemplo: **Somador Completo**



a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

- Modelar a partir das portas lógicas

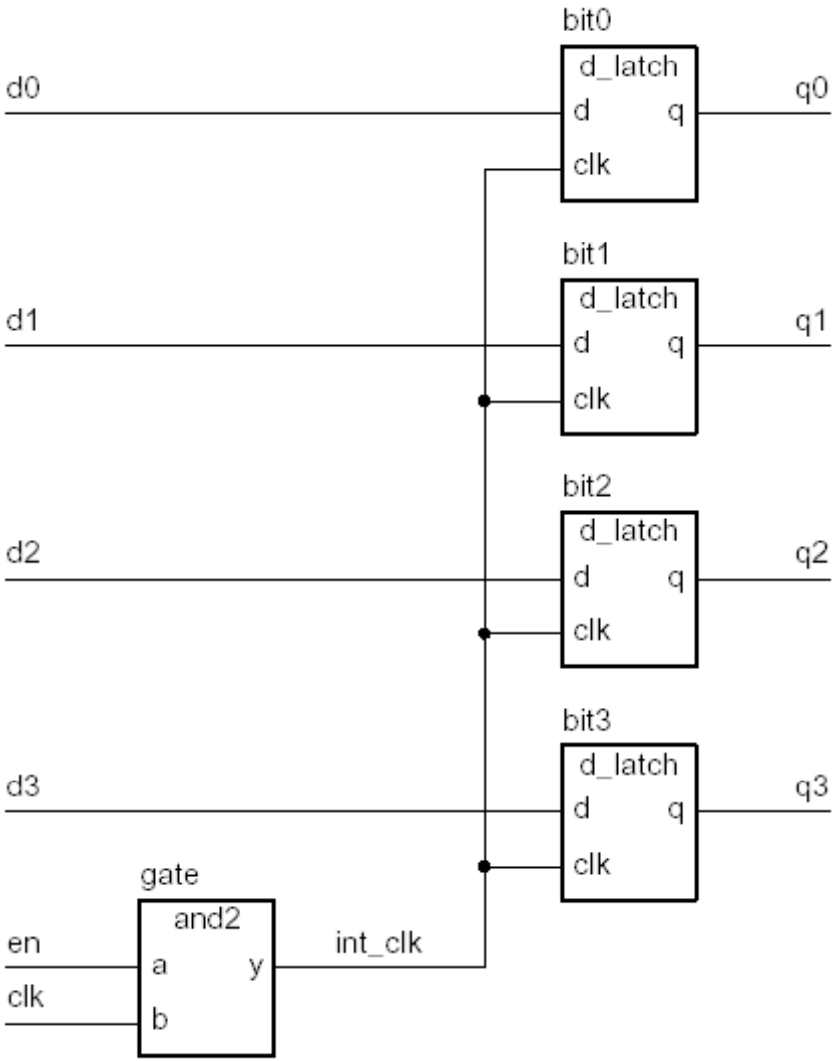
- Modelar comportamento

Exemplo: **Registo de 4 bits**

PARA IMPLEMENTAR QUALQUER CIRCUITO SÍNCRONO É NECESSÁRIO FORÇAR O VHDL A SER SEQUENCIAL E NÃO CONCORRENTE. COMO?

***nome_processo*: PROCESS** (*lista_de_sensibilidades*)
BEGIN
(…)
END PROCESS *nome_processo*

Exemplo: **Registo de 4 bits (cont)**



- Modelar comportamento Latch D (td=2ns)
- Modelar comportamento gate (td=2ns)
- Modelar registo
- Testar Registo

Palavras Reservadas em VHDL

From VHDL 87:	ENTITY	OPEN	WAIT
	EXIT	OR	WHEN
ABS	FILE	OTHERS	WHILE
ACCESS	FOR	OUT	WITH
AFTER	FUNCTION	PACKAGE	XOR
ALIAS	GENERATE	PORT	
ALL	GENERIC	PROCEDURE	*From VHDL 93:*
AND	GUARDED	PROCESS	
ARCHITECTURE	IF	RANGE	GROUP
ARRAY	IN	RECORD	IMPURE
ASSERT	INOUT	REGISTER	INERTIAL
ATTRIBUTE	IS	REM	LITERAL
BEGIN	LABEL	REPORT	POSTPONED
BLOCK	LIBRARY	RETURN	PURE
BODY	LINKAGE	SELECT	REJECT
BUFFER	LOOP	SEVERITY	ROL
BUS	MAP	SIGNAL	ROR
CASE	MOD	SUBTYPE	SHARED
COMPONENT	NAND	THEN	SLA
CONFIGURATION	NEW	TO	SLL
CONSTANT	NEXT	TRANSPORT	SRA
DISCONNECT	NOR	TYPE	SRL
DOWNTO	NOT	UNITS	UNAFFECTED
ELSE	NULL	UNTIL	XNOR
ELSIF	OF	USE	
END	ON	VARIABLE	

FIM

Ref. **LINGUAGEM DE DESCRIÇÃO DE HARDWARE, Prof. Anderson Royes Terroso**