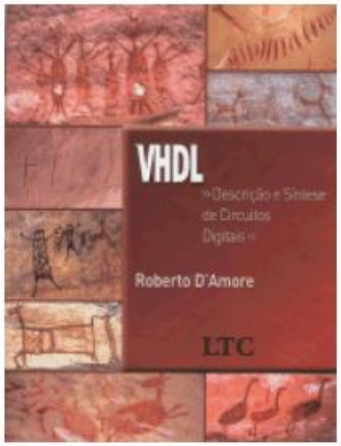
SEL5752/SEL0632 – Linguagens de Descrição de Hardware
Aula 02 – Introdução

Prof. Dr. Maximiliam Luppe

Livro adotado:

VHDL - Descrição e Síntese de Circuitos Digitais
Roberto d'Amore

ISBN 85-216-1452-7
Editora LTC www.ltceditora.com.br



Para informações adicionais consulte: **www.ele.ita.br/~damore/vhdl**

Introdução

Tópicos

- Histórico
- Aspectos gerais da linguagem
- Síntese de circuitos

- Entidade de projeto
- Classes de objetos: constante, variável e sinal
- Tipos
- Operadores

- Construção concorrente WHEN ELSE
- Construção concorrente WITH SELECT
- Processos e lista de sensibilidade
- Construção seqüencial IF ELSE
- Construção seqüencial CASE WHEN
- Circuitos síncronos

Primeiro contato com a linguagem

Tópicos

- **Entidade de projeto**

  Declaração da entidade e declaração da arquitetura

- **Classes de objetos**: constante, variável e sinal
- **Tipos**

  Tipos escalares

  Tipos compostos

  Definição de novos tipos

- **Operadores**
- **Exemplos de utilização**

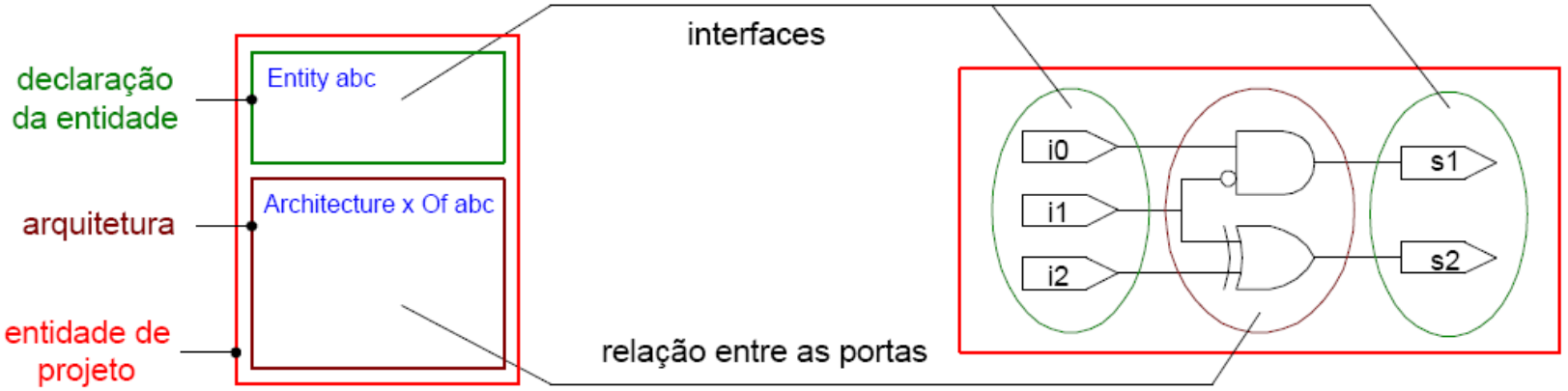
Entidade de projeto

- **Pode representar:**

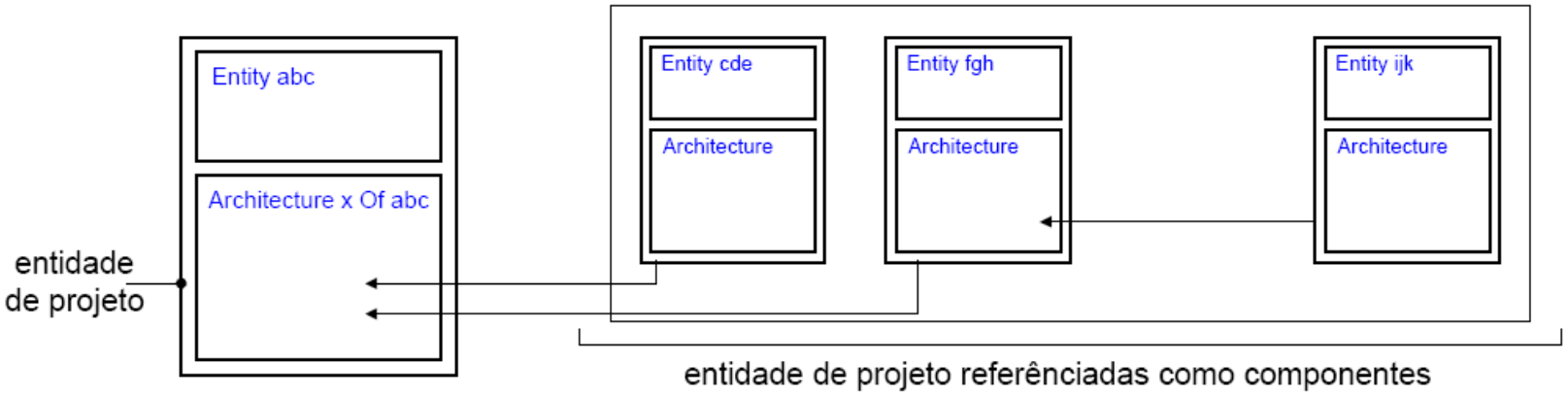
  uma simples porta lógica ..... a um sistema completo

- **Composta de duas partes:**

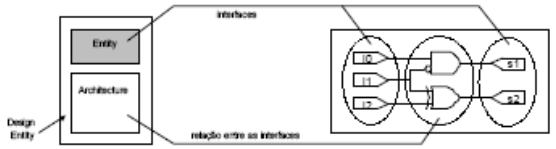
  - Declaração da entidade
    - define portas de entrada e saída da descrição
    - equivalente ao símbolo de um bloco em captura esquemática
  - Arquitetura
    - descreve as relações entre as portas
    - equivalente ao esquema contido no bloco em cap. esquemática



- **Uma entidade de projeto**
  - pode ser descrita na forma: interligação de outras entidades
  - estabelece um projeto hierárquico
  - não existe limite para o nível de hierarquia
  - diferentes estilos de descrição podem ser empregados



Declaração da entidade

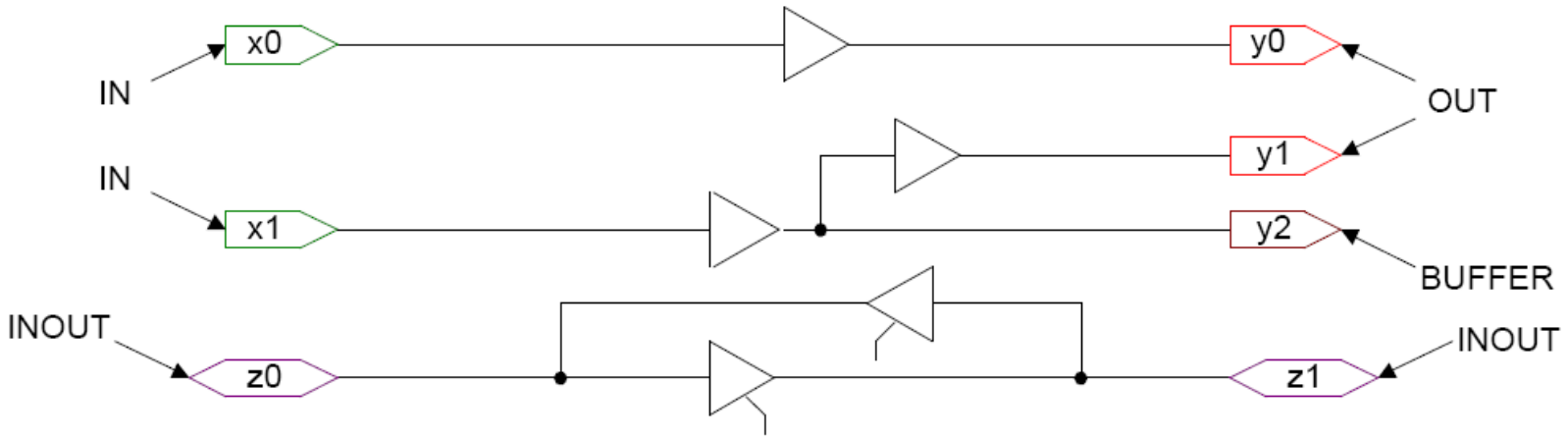


- ENTITY: inicia a declaração
- PORT: define modo e tipo das portas
  - modo IN : entrada
  - modo OUT : saída
  - modo BUFFER : saída - pode ser referenciada internamente
  - modo INOUT : bidirecional
- END: termina a declaração

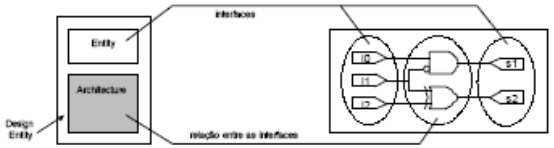
```
ENTITY entidade_abc IS

  PORT (x0, x1     : IN        tipo_a;    -- entradas
        y0, y1     : OUT       tipo_b;    -- saidas
        y2         : BUFFER    tipo_c;    -- saida
        z0, z1     : INOUT     tipo_d);   -- entrada / saida

END entidade_abc;
```

- **Exemplo**: declaração de uma entidade e esquema da arquitetura

```
ENTITY entidade_abc IS

  PORT (x0, x1      : IN        tipo_a;    -- entradas
        y0, y1      : OUT       tipo_b;    -- saidas
        y2          : BUFFER    tipo_c;    -- saida
        z0, z1      : INOUT     tipo_d);   -- entrada / saida

END entidade_abc;
```



Declaração da arquitetura



- ARCHITECTURE: inicia a declaração
- linhas que seguem podem conter:
  - declaração de sinais e constantes
  - declaração de componentes referenciados
  - descrição de subprogramas locais
  - definição de novos tipos
- BEGIN: inicia a descrição
- END: termina a descrição

```
ARCHITECTURE estilo_abc OF entidade_abc IS
  -- declaracoes de sinais e constantes
  -- declaracoes de componentes referenciados
  -- descricao de sub-programas locais
  -- definicao de novos tipos de dados locais
  --
BEGIN
  --
  -- declaracoes concorrentes
  --
END estilo_abc;
```

Classe de objetos: constante, variável e sinal

- Objetos: são elementos que contêm um valor armazenado
- Quatro tipos:
  - CONSTANT: valor estático
  - VARIABLE: valor imposto pode ser alterado
    regiões de código seqüencial
  - SIGNAL: valor imposto pode ser alterado
    regiões de código seqüencial e concorrente
  - FILE: associado a criação de arquivos
- Exemplos de atribuição de valores entre objetos

```
sinal_2    <= sinal_1;        -- atribuicao valor para sinal
sinal_3    <= variavel_1;     -- atribuicao valor para sinal
sinal_4    <= constante_1;    -- atribuicao valor para sinal

variavel_2 := sinal_1;        -- atribuicao valor para variavel
variavel_3 := variavel_1;     -- atribuicao valor para variavel
variavel_4 := constante_1;    -- atribuicao valor para variavel
```

- **Exemplo de uma descrição completa**
  - definidas três portas: uma entrada, duas de saída
  - tipo das portas: INTEGER
  - operações: transferência de valores
  - declarados: s1 sinal interno e c1 constante -- linhas 7 e 8
  - observar concorrência no código:

    linha 11: s1<= x1 -- s1 recebe o valor da porta de entrada x1

    linha 10: y1<= s1 -- valor de s1 transferido para porta de saída y1

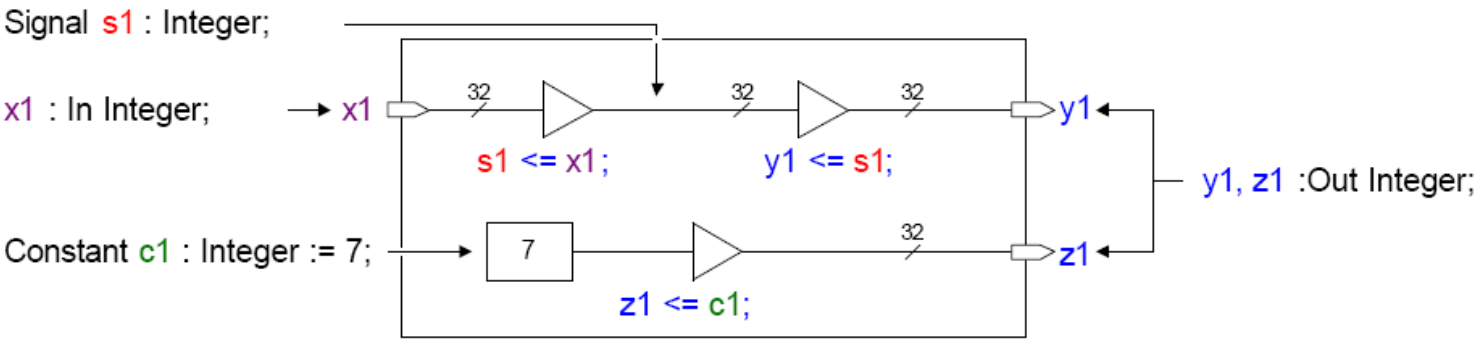
```
 1 ENTITY atrib_1 IS
 2   PORT (x1      : IN  INTEGER;
 3         y1,z1   : OUT INTEGER);
 4 END;
 5
 6 ARCHITECTURE teste OF atrib_1 IS
 7   SIGNAL   s1 : INTEGER;
 8   CONSTANT c1 : INTEGER := 7;
 9 BEGIN
10   y1 <= s1;
11   s1 <= x1;
12
13   z1 <= c1;
14 END teste;
```

- Exemplo de uma descrição completa

- observar concorrência no código:

  linha 10: y1<= s1 -- valor de s1 transferido para saída y1

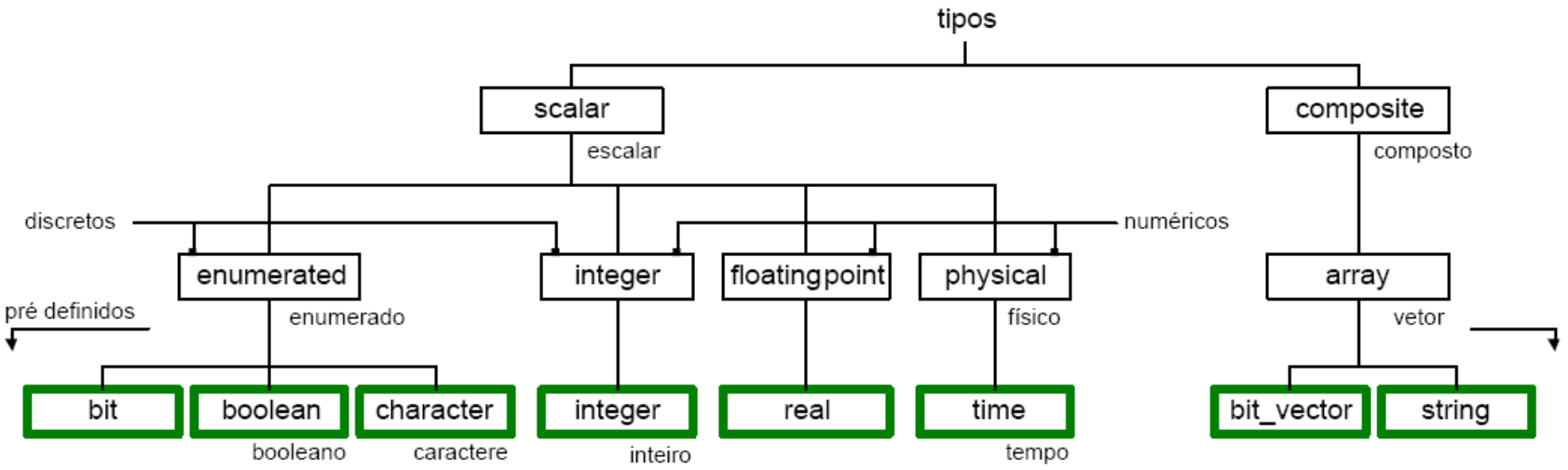
  linha 11: s1<= x1 -- s1 recebe o valor da entrada x1



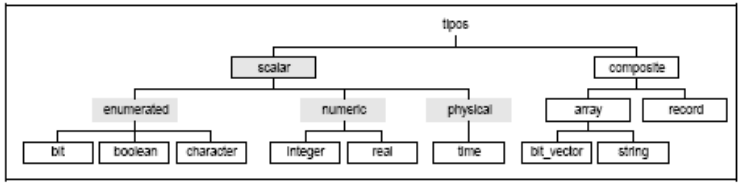

```
 5
 6 ARCHITECTURE teste OF atrib_1 IS
 7   SIGNAL   s1 : INTEGER;           -- declaracao de um sinal tipo inteiro
 8   CONSTANT c1 : INTEGER := 7;      -- declaracao de uma constante tipo inteiro
 9 BEGIN
10   y1 <= s1;                        -- regiao de codigo concorrente
11   s1 <= x1;
12
13   z1 <= c1;
14 END teste;
```

Tipos

- **Objetos**: devem ser declarados segundo uma especificação de tipo
- **Objetos de tipos diferentes**: não é permitida a transferência de valores
- **Alguns tipos definidos no pacote padrão VHDL**:



Tipos escalares

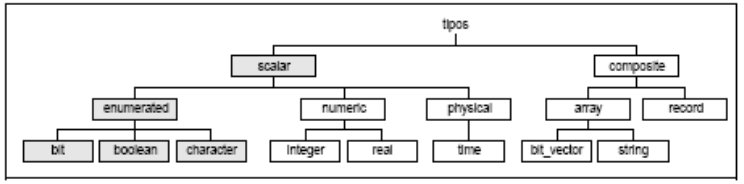


- **São ordenados**: podem ser aplicados operadores relacionais (maior, menor)

tipo	valor	exemplos
BIT	um, zero	1, 0
BOOLEAN	verdadeiro, falso	TRUE, FALSE
CHARACTER	caracteres ASCII	a, b, c, A, B, C, ?, (
INTEGER	$-2^{31} \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B# 2#11_11_011#
NATURAL	$0 \leq x \leq 2^{31}-1$	
POSITIVE	$1 \leq x \leq 2^{31}-1$	
REAL	$-1.0\text{x}10^{308} \leq x \leq +1.0\text{x}10^{308}$	1.23, 1.23E+2, 16#7.B#E+1
TIME	ps $=10^3$ fs ns $=10^3$ ps us $=10^3$ ns ms $=10^3$ us sec $=10^3$ ms min =60 sec hr =60 min	1 us, 100 ps, 1 fs

IEEE754 32 bits $\pm 1.18\text{x}10^{-38} \leq x \leq \pm 3.4\text{x}10^{38}$
IEEE754 64 bits $\pm 2.23\text{x}10^{-308} \leq x \leq \pm 1.8\text{x}10^{308}$

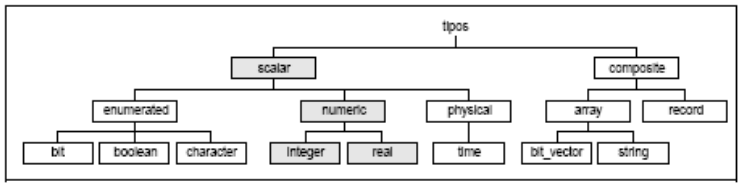
Tipos escalares



- **Classe**: enumerados
  - BIT: empregado para representar níveis lógicos alto e baixo
  - BOOLEAN: empregado em comandos que executam uma decisão
  - CHARACTER: qualquer caracter ASCII padrão (estendido versão VHDL 1993)

classe	tipo	valor	exemplos
enumerado	BIT	um, zero	1, 0
	BOOLEAN	verdadeiro, falso	TRUE, FALSE
	CHARACTER	caracteres ASCII	a, b, c, A, B, C, ?, (

Tipos escalares

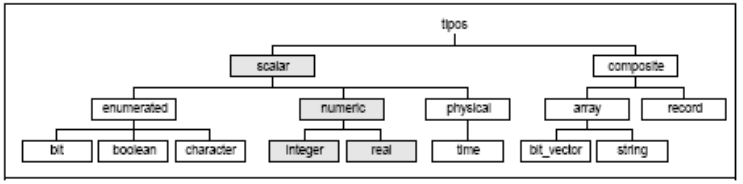


- **Classe**: numéricos
  - INTEGER: representa um número inteiro entre $-2^{31} \leq x \leq 2^{31}-1$
  - Sub-tipos de INTEGER
    - NATURAL: valores entre $0 \leq x \leq 2^{31}-1$
    - POSITIVE: valores entre $1 \leq x \leq 2^{31}-1$
  - Nota: Troca de valores entre tipo $\leftrightarrow$ sub-tipo permitida
  - REAL: ponto flutuante

classe	tipo	valor	exemplos
numérico	INTEGER	$-2^{31} \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B# 2#11_11_011#
	NATURAL	$0 \leq x \leq 2^{31}-1$	
	POSITIVE	$1 \leq x \leq 2^{31}-1$	
	REAL	$-1.0\text{x}10^{308} \leq x \leq +1.0\text{x}10^{308}$	1.23, 1.23E+2, 16#7.B#E+1

IEEE754 32 bits $\pm 1.18\text{x}10^{-38} \leq x \leq \pm 3.4\text{x}10^{38}$
IEEE754 64 bits $\pm 2.23\text{x}10^{-308} \leq x \leq \pm 1.8\text{x}10^{308}$

Tipos escalares - observações quanto a síntese

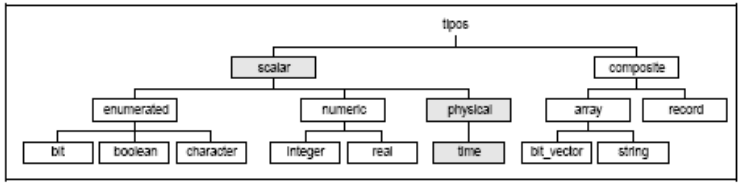


- **Classe**: numéricos
  - INTEGER: representa um número inteiro entre $-2^{31} \leq x \leq 2^{31}-1$
    necessário uma linha de 32 bits para representação!
    conveniente limitar a faixa de valores na declaração
  - REAL: ponto flutuante
    não suportado pelas ferramentas de síntese

classe	tipo	valor	exemplos
numérico	INTEGER	$-2^{31} \leq x \leq 2^{31}-1$	123, 8#173#, 16#7B# 2#11_11_011#
	NATURAL	$0 \leq x \leq 2^{31}-1$	
	POSITIVE	$1 \leq x \leq 2^{31}-1$	
	REAL	$-1.0\text{x}10^{308} \leq x \leq +1.0\text{x}10^{308}$	1.23, 1.23E+2, 16#7.B#E+1

IEEE754 32 bits $\pm 1.18\text{x}10^{-38} \leq x \leq \pm 3.4\text{x}10^{38}$
IEEE754 64 bits $\pm 2.23\text{x}10^{-308} \leq x \leq \pm 1.8\text{x}10^{308}$

Tipos escalares

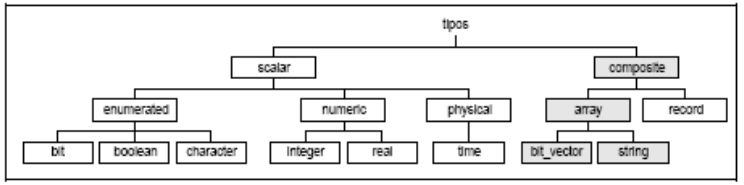


- **Classe**: físicos
  - representa quantidades físicas
  - estabelecidos: uma unidade básica e múltiplos desta unidade

  - TIME: tipo fisco definido no pacote padrão
    - quantidade física: tempo
    - unidade básica: fs = 10^{-15}s
    - múltiplos: ps, ns, us, ms, sec, min, hr
    - síntese: tipo time é ignorado pelas ferramentas de síntese

classe	tipo	valor	exemplos
físico	TIME	ps =10^3 fs ns =10^3 ps us =10^3 ns ms =10^3 us sec =10^3 ms min =60 sec hr =60 min	1 us, 100 ps, 1 fs

Tipos compostos

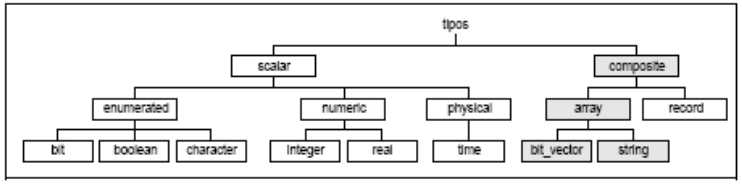


- **Classes**:
  - vetor (array): agrupa elementos de um mesmo tipo

- Classe array (definidos no pacote padrão):
  - BIT_VECTOR: vetor contendo elementos tipo bit
  - STRING: vetor contendo elementos tipo character

Classe	tipo	valor	exemplos
vetor	BIT_VECTOR	1 , 0	”1010”, B”10_10”, O”12”, X”A”
	STRING	tipo “character”	”texto”, ””incluindo_aspas””

Tipos compostos



- **Declarações**:
  - limites definidos por TO e DOWNTO
- **Exemplos de declarações tipos**: bit_vector e string:

SIGNAL a: BIT_VECTOR(0 TO 7) := "10110011"

1	0	1	1	0	0	1	1
a(0)	a(1)	a(2)	a(3)	a(4)	a(5)	a(6)	a(7)

SIGNAL b: BIT_VECTOR(7 DOWNTO 0) := "10110011"

1	0	1	1	0	0	1	1
b(7)	b(6)	b(5)	b(4)	b(3)	b(2)	b(1)	b(0)

SIGNAL c: STRING(1 TO 9) := "Alo mundo"

A	l	o		m	u	n	d	o
c(1)	c(2)	c(3)	c(4)	c(5)	c(6)	c(7)	c(8)	c(9)

SIGNAL d: STRING(9 DOWNTO 1) := "Alo mundo"

A	l	o		m	u	n	d	o
d(9)	d(8)	d(7)	d(6)	d(5)	d(4)	d(3)	d(2)	d(1)

Agregados

- **Uma expressão indicando o valor de um tipo composto**
- **Cada elemento do tipo composto tem o seu valor definido**
- **OTHERS**
  - identifica todos elementos não especificados
  - deve ser a última associação na lista de associações

```
-- valor 00010 agregado notacao posicional
-- posicao:
--       4   3   2   1   0
s2 <= ('0','0','0','1','0');              -- equivalente: s2 <= "00010";


-- valor 00011 agregado associacao por nomes
--
s3 <= (1=>'1', 0=>'1', OTHERS=>'0');     -- equivalente: s3 <= "00011";
```

Definição de novos tipos (visão preliminar)

- **A linguagem permite a criação de novos tipos**
- **Aplicações**:
  - facilitar a leitura do código: estados de uma máquina
  - definir novos tipos físicos: resistência, capacitância etc.
  - novos tipos compostos: definição de memórias
- **Declaração**: palavra reservada TYPE
- **Exemplo**:

```
TYPE estado IS (parado, inicio, caso_1, caso_2, caso_3);
SIGNAL abc : estado := parado;
```

- definido um novo tipo estado

```
TYPE estado IS (parado, inicio, caso_1, caso_2, caso_3);
```

- valores possíveis deste tipo: parado, inicio, caso_1, caso_2, caso_3

- declaração de um sinal do tipo estado

```
SIGNAL abc : estado := parado;
```

Operadores

- **Divididos em classes**:
  - as classes definem a precedência dos operadores
  - operadores de uma mesma classe: igual precedência
- **Maior precedência**: classe diversos
- **Menor precedência**: classe lógicos
- **Operador not**: operador lógico, está na classe diversos devido a precedência

classe	operadores
lógicos	and or nand nor xor xnor
relacionais	= /= < <= > >=
deslocamento	sll srl sla sra rol ror
adição	+ – &
sinal	+ –
multiplicação	* / mod rem
diversos	** abs not

- **Operadores lógicos**:

  - operandos: tipos bit e boolean

    vetores unidimensionais compostos de elementos bit e boolean

    (exemplo: bit_vector)

  - os vetores devem ter o mesmo tamanho

  - operação executada entre elementos de mesma posição

operadores	operando L	operando R	retorna
not and or	bit	bit	bit
nand nor xor xnor	boolean	boolean	boolean
Nota: O operador not pertence a classe diversos			

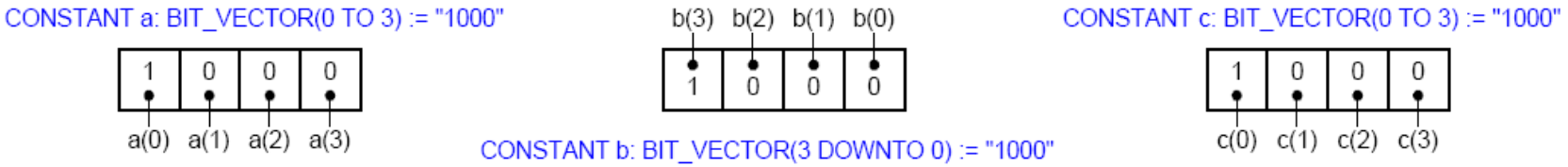
- tipo bit: 0,1 tipo boolean: 0 = false 1 = true

L	R	not L	L and R	L nand R	L or R	L nor R	L xor R	L xnor R
1	1	0	1	0	1	0	0	1
1	0	0	0	1	1	0	1	0
0	1	1	0	1	1	0	1	0
0	0	1	0	1	0	1	0	1

- **Operadores relacionais**:
  - igualdade e desigualdade: qualquer tipo
    - escalares x = y verdadeiro: x mesmo valor de y
    - compostos x = y verdadeiro: iguais cada elemento de mesma posição de x e y
  - ordenação: tipos escalares (bit, boolean, character, integer, real, time)

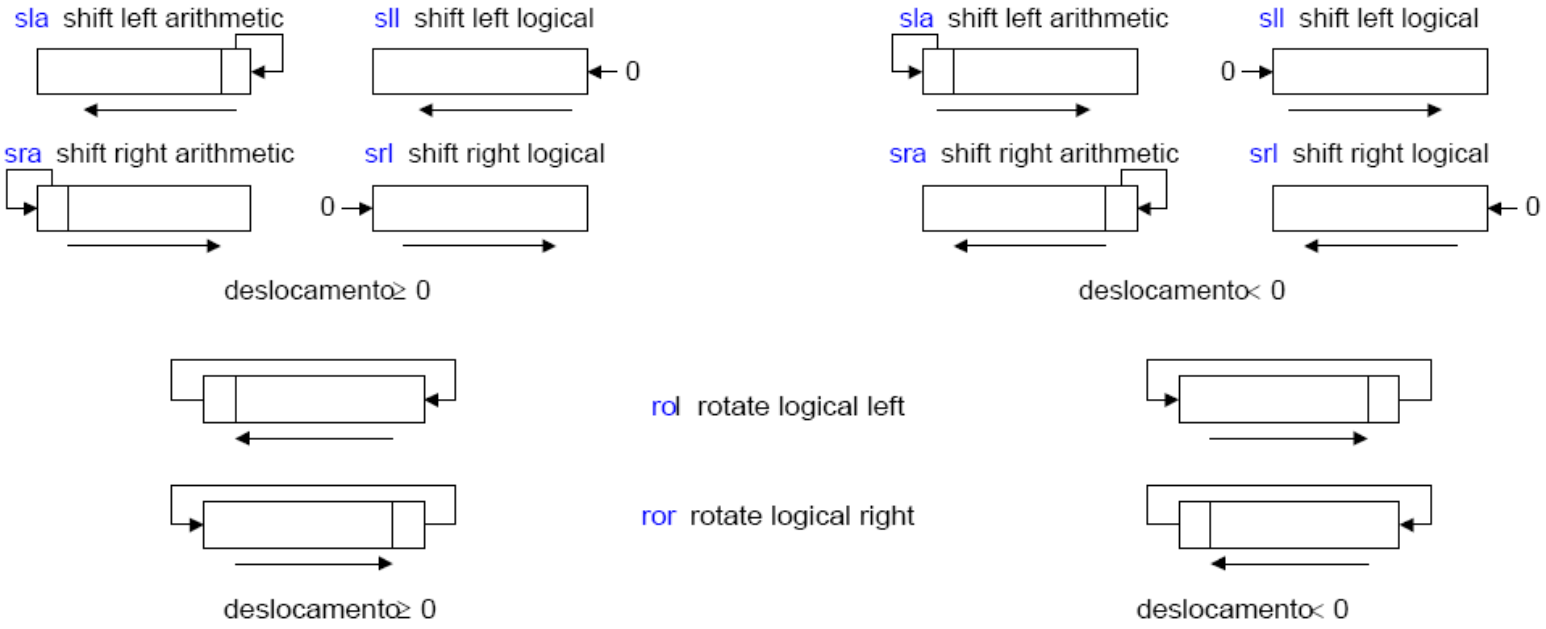
operadores	operando L	operando R	retorna
= /=	qualquer tipo	mesmo tipo de L	boolean
> < >= <=	qualquer tipo escalar	mesmo tipo de L	boolean

- exemplo de valores tipo bit_vector iguais:



- **Operadores deslocamento e rotação**:
  - não suportado pela versão VHDL-1987

operadores	operando L	operando R	retorna
sll srl sla sra rol ror	vetor unidimensional com elementos bit ou boolean	integer	o mesmo tipo de L



- **Operadores adição**
  - adição e subtração: tipo numérico
  - concatenação: vetor unidimensional e elementos (mesmo tipo)

operadores	operando L	operando R	retorna
+ -	tipo numérico	o mesmo tipo de L	mesmo tipo
&	vetor	mesmo vetor de L	vetor mesmo tipo
	vetor	elemento	vetor mesmo tipo
	elemento	vetor	vetor mesmo tipo
	elemento	elemento	vetor

- **Exemplo**:

```
a <= b & c;   -- "a" bit_vector 8 elementos,"b","c" bit_vetor 4 elementos
x <= y & '1'; -- "x" bit_vector 5 elementos,    "y" bit_vetor 4 elementos
```

- **Operadores aritméticos**: identidade, negação
  - possuem o mesmo significado dos operadores matemáticos equivalentes
  - operandos: qualquer tipo numérico

operadores	operando R	retorna
+ -	qualquer tipo numérico	mesmo tipo

- **Operadores diversos**: absoluto, exponenciação
  - possuem o mesmo significado dos operadores matemáticos equivalentes

operadores	operando L	operando R	retorna
abs	qualquer tipo numérico	-	mesmo tipo
**	qualquer tipo integer	integer	mesmo tipo de L
	qualquer tipo real	integer	mesmo tipo de L

280

Integer division and remainder are defined by the following relation:

A = (A/B)*B + (A **rem** B)

where (A **rem** B) has the sign of A and an absolute value less than the absolute value of B. Integer division satisfies the following identity:

285 (–A)/B = –(A/B) = A/(–B)

The result of the modulus operation is such that (A **mod** B) has the sign of B and an absolute value less than the absolute value of B; in addition, for some integer value N, this result must satisfy the relation:

A = B*N + (A **mod** B)

In addition to the above table, the operators * and / are predefined for any physical type.

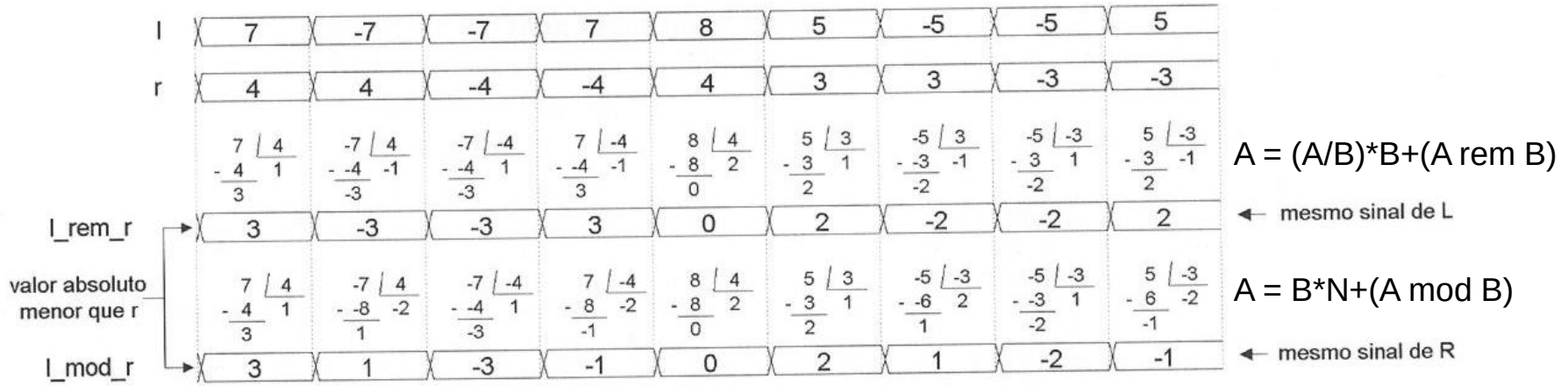


Figura 2.5.1 Exemplos com os operadores " mod " e " rem ".

- **Operadores multiplicação** - tipos físicos
  - multiplicação: operandos tipo (físico - real) e (físico - inteiro)
  - divisão: operando L tipo físico, operando R (inteiro, real, físico)

operadores	operando L	operando R	retorna
*	qualquer tipo physical	integer	mesmo tipo de L
	qualquer tipo physical	real	mesmo tipo de L
	integer	qualquer tipo physical	mesmo tipo de R
	real	qualquer tipo physical	mesmo tipo de R
/	qualquer tipo physical	integer	mesmo tipo de L
	qualquer tipo physical	real	mesmo tipo de L
	qualquer tipo physical	mesmo tipo	integer

- **Exemplo**: Atribuição de valores em sinais, tipos INTEGER e REAL
  - operação: valor 11 atribuído a todas as portas de saída

```
 9 ARCHITECTURE teste OF int_real IS
10   CONSTANT i1 : INTEGER  :=  11;              -- valor 11, base 10
11   CONSTANT i2 : INTEGER  :=  10#11#;          -- valor 11, base 10
12   CONSTANT i3 : INTEGER  :=  2#01011#;        -- valor 11, base 2
13   CONSTANT i4 : INTEGER  :=  2#01_01_1#;      -- valor 11, base 2
14   CONSTANT i5 : INTEGER  :=  5#21#;           -- valor 11, base 5
15   CONSTANT i6 : NATURAL  :=  8#13#;           -- valor 11, base 8
16   CONSTANT i7 : POSITIVE :=  16#B#;           -- valor 11, base 16
17
18   CONSTANT r1 : REAL     :=  11.0;            -- valor 11, base 10
19   CONSTANT r2 : REAL     :=  1.1E01;          -- valor 11, base 10 formato nn.nExx
20   CONSTANT r3 : REAL     :=  2#01011.0#;      -- valor 11, base 2
21   CONSTANT r4 : REAL     :=  8#1.3#E01;       -- valor 11, base 8  formato nn.nExx
22   CONSTANT r5 : REAL     :=  16#B.0#;         -- valor 11, base 16
23 BEGIN
24   ci1 <= i1; ci2 <= i2; ci3 <= i3; ci4 <= i4; ci5 <= i5; ci6 <= i6; ci7 <= i7;
25   cr1 <= r1; cr2 <= r2; cr3 <= r3; cr4 <= r5;
26 END teste;
```

- **Exemplo**: Atribuição de valores em sinais, tipos BIT_VECTOR
  - operação: valor 1011 atribuído a todas as portas de saída
  - diferentes bases de representação
    tipos integer, real formato: 16#B# 16#B.0# (vide exemplo anterior)
    tipos bit_vector formato: X"B"
  - linha 12: caracter _ em 01_0_11 melhora leitura do valor
  - linhas 14 e 15: valor definido para uma parte do vetor pal. res.: DOWNTO

```
 5 ARCHITECTURE teste OF std_a IS
 6   CONSTANT c1   : BIT_VECTOR(4 DOWNTO 0) := "01011";  -- constante
 7   CONSTANT zero : BIT := '0';
 8   CONSTANT um   : BIT := '1';
 9 BEGIN
10   s1 <= c1;                -- valor atraves de constante
11   s2 <= "01011";           -- valor (01011) direto - base binaria
12   s3 <= B"01_0_11";        -- valor (01011) direto - base binaria com separadores
13   s4 <= '0' & X"B";        -- bit (0) concatenado com valor hexadecimal (1011)
14   s5(4 DOWNTO 3) <= "01";           -- valor (01), parte do vetor
15   s5(2 DOWNTO 0) <= zero & um & um;  -- valor (010), parte com concatenacao
16 END teste;
```

- **Exemplo**: Atribuição de valores em sinais, tipos BIT_VECTOR - agregados

  - s2 s4 → atribuição de valor: notação posicional

  - s3 s5 → atribuição de valor: associação de nomes

```
 1 ENTITY std_a1 IS
 2  PORT(s2, s3, s4, s5   : OUT BIT_VECTOR(4 DOWNTO 0));
 3 END std_a1;
 4
 5 ARCHITECTURE teste OF std_a1 IS
 6   CONSTANT zero : BIT := '0';
 7   CONSTANT um   : BIT := '1';
 8 BEGIN
 9   s2 <= ('0','0','0','1','0');            --  00010, agregado notacao posicional
10   s3 <= (1=>'1', 0=>'1', OTHERS=>'0');     --  00011, agregado associacao por nomes
11   s4 <= (zero, '0', um OR '0', '0', '0');        -- 00100, agregado com operacoes
12   s5 <= (4 DOWNTO 3 =>'0', 1=>'0', OTHERS=>'1'); -- 00101, agregado faixa discreta
13 END teste;
```

- **Exemplo**: Atribuição de valores - expressões com operadores lógicos

  - linhas 8 a 14: comentário no código

  - linha 15: x nand y equivale a not (a and b) → ordem das operações importa

```
 1 ENTITY std_xa1 IS
 2   PORT( a, b, c, d          : IN  BIT;
 3         x1, x2, x3, x4, x5 : OUT BIT);
 4 END std_xa1;
 5
 6 ARCHITECTURE exemplo OF std_xa1 IS
 7 BEGIN
 8    x1 <= a OR NOT b;              -- Certo:  operador NOT tem precedencia
 -                                   --         mais elevada
 9    x2 <= a AND b AND c;           -- Certo:  operadores iguais
10 -- x3 <= a AND b  OR c;           -- Errado: expressao ambigua x3=(a.b)+c
 -                                   --                      ou x3=a.(b+c) ?
11    x3 <=(a AND b) OR c;           -- Certo:  empregando parentesis
12 -- x4 <= a AND b  OR  c AND d;    -- Errado: expressao ambigua, operadores
 -                                   --         com mesma precedecia
13    x4 <=(a AND b) OR (c AND d);   -- Certo:  x4 = a.b + c.d
14 -- x5 <= a NAND b  NAND c;        -- Errado: operadores com negacao
 -                                   --         necessitam parentesis
15    x5 <=(a NAND b) NAND c;        -- Certo:  operador com negacao entre
 -                                   --         parentesis
16 END exemplo;
```

- **Exemplo**: Operadores classe adição

  - linhas 10 e 11: operação de concatenação de dois vetores (tipos bit_vector)

  - linha 12: soma de dois tipos inteiros

```
 1 ENTITY std_xc IS
 2   PORT (bv_a,  bv_b  : IN  BIT_VECTOR(1 DOWNTO 0);
 3         int_a, int_b : IN  INTEGER RANGE -32 TO 31;
 4         bv_c,  bc_d  : OUT BIT_VECTOR(3 DOWNTO 0);
 5         int_c        : OUT INTEGER RANGE -64 TO 63);
 6 END std_xc;
 7
 8 ARCHITECTURE teste OF std_xc IS
 9 BEGIN
10   bv_c  <= bv_a & bv_b;
11   bc_d  <= bv_a & '1' & '0';
12   int_c <= -int_a +int_b;
13 END teste;
```

- **Exemplo**: Operadores com tipo STRING
  - Relembrando STRING: vetor contendo elementos tipo character
  - Operações de concatenação de sinais (linhas 9 e 10)
  -Note que os sinais x e y poderiam ser declarados como constante
    - seus valores não são alterados no código.
- Possível sintetizar a descrição:
  - 8 bits para cada caracter ASCII
  - 2 vetores de 9 caracteres → 8×2×9 =144 linhas

para representação do valor " Alo mundo "

```
 1 ENTITY lista1 IS
 2   PORT (c, d   : OUT STRING(1 TO 9));
 3 END lista1;
 4
 5 ARCHITECTURE teste OF lista1 IS
 6    SIGNAL x : STRING(1 TO 3) := "Alo";
 7    SIGNAL y : STRING(1 TO 5) := "mundo";
 8 BEGIN
 9   c <= x & " " & y;
10   d <= x(1 TO 2) & "o " & y;
11 END teste;
```

2.7 Exercícios

2.7.1 Apresente o código de uma entidade que descreva as quatro expressões lógicas contidas na Figura 2.7.1. Nessas expressões, considere o operador de negação com maior precedência e o operador lógico " ou " com menor precedência. A entidade deve ter quatro portas de entrada, " a ", " b ", " c " e " d ", e quatro portas de saída, " s1 ", " s2 ", " s3 " e " s4 ", todas do tipo bit.

$s1 = a + \overline{b}$ $\quad s2 = a + \overline{b} \cdot c + d$ $\quad s3 = (a + \overline{b}) \cdot (c + d)$ $\quad s4 = (a + \overline{b}) \cdot \overline{(c + ad)}$

Figura 2.7.1 Expressões lógicas para o Exercício 2.7.1.

2.7.2 Sintetize a descrição proposta para a solução do Exercício 2.7.1. Observe o resultado no nível RTL. Verifique, no esquema criado, se a expressão " $a + \overline{b}$ ", presente nas expressões dos sinais " s1 ", " s2 " e " s3 ", é implementada por uma mesma porta lógica.

2.7.3 Apresente a descrição de uma entidade com duas portas de entrada e uma porta de saída, todas do tipo " bit_vector ". A Figura 2.7.2 ilustra o problema. Ambas as entradas possuem quatro bits, e o valor presente nessas entradas deve ser transferido para a saída, conforme ilustrado na figura.

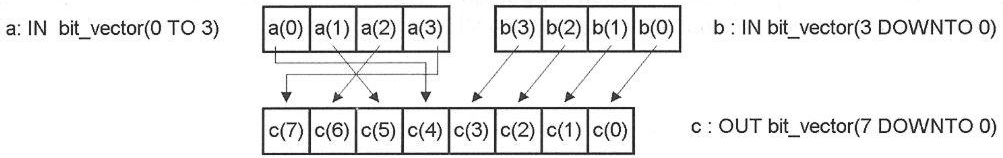


Figura 2.7.2 Exercício 2.7.3.

2.7.4 Considerando a descrição no Quadro 2.7.1, determine, sem o auxílio de um simulador, qual o valor de cada porta de saída. Note que todos os comandos são concorrentes, e, portanto, a ordem nas linhas do código não importa.

```
1  ENTITY std_yc IS
2    PORT (a, b   : OUT BIT_VECTOR (2 DOWNTO 0);
3          c, d   : OUT BIT_VECTOR (0 TO 2));
4  END std_yc;
5
6  ARCHITECTURE teste OF std_yc IS
7    CONSTANT x : BIT_VECTOR(0 TO 7) := B"1101_1001";
8    SIGNAL   y : BIT_VECTOR(3 DOWNTO 0);
9  BEGIN
10   a <= x(1 TO 3);
11   b <= y(3 DOWNTO 1);
12   c <= x(5 TO 7);
13   d <= y(2 DOWNTO 0);
14   y <= x(2 TO 5);
15 END teste;
```

Quadro 2.7.1 Descrição para o Exercício 2.7.4.

2.7.5 No Quadro 2.7.2 é apresentado um código para o teste de operações lógicas. A definição da entidade contém duas entradas e três saídas do tipo " bit_vector ". O valor das saídas é determinado por operações lógicas definidas nas linhas 12, 13 e 14. Desconsiderando inicialmente as linhas 7, 8, 9, 10, 16, 17 e 18, compile a descrição, e simule o resultado para diferentes condições de entrada.

Na arquitetura é definido um vetor composto por elementos do tipo " boolean " (linha 7). Não se preocupe com a definição do tipo, ela será explicada com mais detalhes no Capítulo 12. O intuito dessa definição é verificar o emprego das operações lógicas em vetores contendo elementos " boolean ". Como esse tipo é definido localmente, não é possível definir portas de entrada ou saída empregando esse tipo para entidade. Por esse motivo, foram definidas duas constantes (linhas 9 e 10) para um pequeno teste. Verifique o resultado da simulação. O resultado não é apresentado externamente, mas os simuladores permitem observar sinais internos.

```
1  ENTITY std_c IS
2    PORT( a_bit, b_bit             : IN  BIT_VECTOR(2 DOWNTO 0);
3          not_bit, and_bit, or_bit : OUT BIT_VECTOR(2 DOWNTO 0));
4    END std_c;
5
6  ARCHITECTURE exemplo OF std_c IS
7    TYPE     vetor_booleano IS ARRAY (0 TO 2) OF BOOLEAN;
8    SIGNAL   not_bool, and_bool, or_bool : vetor_booleano;
9    CONSTANT a_bool  : vetor_booleano := (TRUE,  FALSE, TRUE);
10   CONSTANT b_bool  : vetor_booleano := (FALSE, TRUE,  FALSE);
11 BEGIN
12   not_bit <=   NOT a_bit;
13   and_bit <= a_bit AND b_bit;
14   or_bit  <= a_bit  OR b_bit;
15
16   not_bool <= NOT a_bool;
17   and_bool <= a_bool AND b_bool;
18   or_bool  <= a_bool  OR b_bool;
19 END exemplo;
```

Quadro 2.7.2 Descrição para teste de operações lógicas.

2.7.6 No Quadro 2.7.3 é apresentado um código seguindo a versão VHDL-1993 para o teste da operação lógica " xnor ". Altere a opção de compilação do simulador para a versão VHDL-1993 e simule a entidade.

```
1  -- versao VHDL-1993
2  ENTITY std_c93 IS
3    PORT( a_bit, b_bit       : IN  BIT_VECTOR(2 DOWNTO 0);
4          not_bit, xnor_bit : OUT BIT_VECTOR(2 DOWNTO 0));
5  END std_c93;
6
7  ARCHITECTURE exemplo OF std_c93 IS
8    TYPE vetor_booleano IS ARRAY (0 TO 2) OF BOOLEAN;
9    SIGNAL not_bool, xnor_bool : vetor_booleano;
10   SIGNAL a_bool  : vetor_booleano := (TRUE,  FALSE, TRUE);
11   SIGNAL b_bool  : vetor_booleano := (FALSE, TRUE,  FALSE);
12 BEGIN
13   not_bit  <=   NOT a_bit;
14   xnor_bit <= a_bit  XNOR b_bit;
15
16   not_bool  <= NOT a_bool;
17   xnor_bool <= a_bool  XNOR b_bool;
18 END exemplo;
```

Quadro 2.7.3 Descrição segundo a versão VHDL-1993.

2.7.7 No Quadro 2.7.4 é apresentada uma descrição com problemas. Identifique as linhas com erro e proponha uma correção.

```
1  ENTITY errad_1 IS
2    PORT (a, b, c, d : IN  BIT;
3          s          : OUT BIT_VECTOR (5 DOWNTO 0));
4  END errad_1;
5
6  ARCHITECTURE teste OF errad_1 IS
7  BEGIN
8    s(0) <= a AND  b OR c AND d;
9    s(1) <= a NOR b NOR c;
10   s(2) <= a AND  b OR c;
11   s(3) <= NOT (a AND b) NAND c;
12   s(4) <= a XOR  b XOR c;
13 END teste;
```

Quadro 2.7.4 Descrição contendo erros.

2.7.8 No Quadro 2.7.5 é apresentada uma outra descrição com problemas. Identifique as linhas com erro e proponha uma correção.

```
1  ENTITY errad_2 IS
2    PORT (a, b, c, d : IN  BIT;
3          r, s, t, u, v, x    : OUT BIT_VECTOR (0 TO 5));
4  END errad_2;
5
6  ARCHITECTURE teste OF errad_2 IS
7  BEGIN
8    r(0 TO 2) <= a & b & c;
9    s <= a & b & c & "010";
10   t(0 TO 2) & u(3 TO 5) <= "101101";
11   v <= a & (OTHERS => '0');
12   x <= a & (OTHERS => '0') & '1';
13 END teste;
```

Quadro 2.7.5 Outra descrição contendo erros.