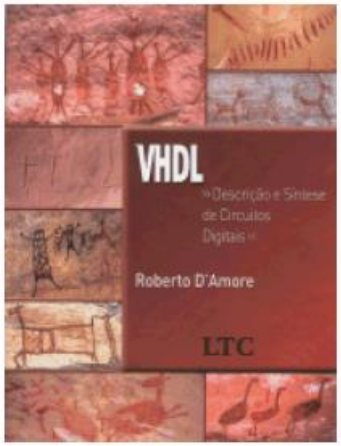
SEL5752/SEL0632 – Linguagens de Descrição de Hardware
Aula 11 – Definição de Tipos (1)

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**

Definição de tipos

Tópicos

- Definição de tipos enumerados
- Definição de tipos físicos
- Definição de sub-tipos
- Definição de tipos registro
- Definição de tipos vetor
- Síntese de memórias

Tipos

- **Tipo:**
  - representa o conjunto de valores que um objeto pode armazenar

    e

  - conjunto de operações que podem ser realizadas com o objeto

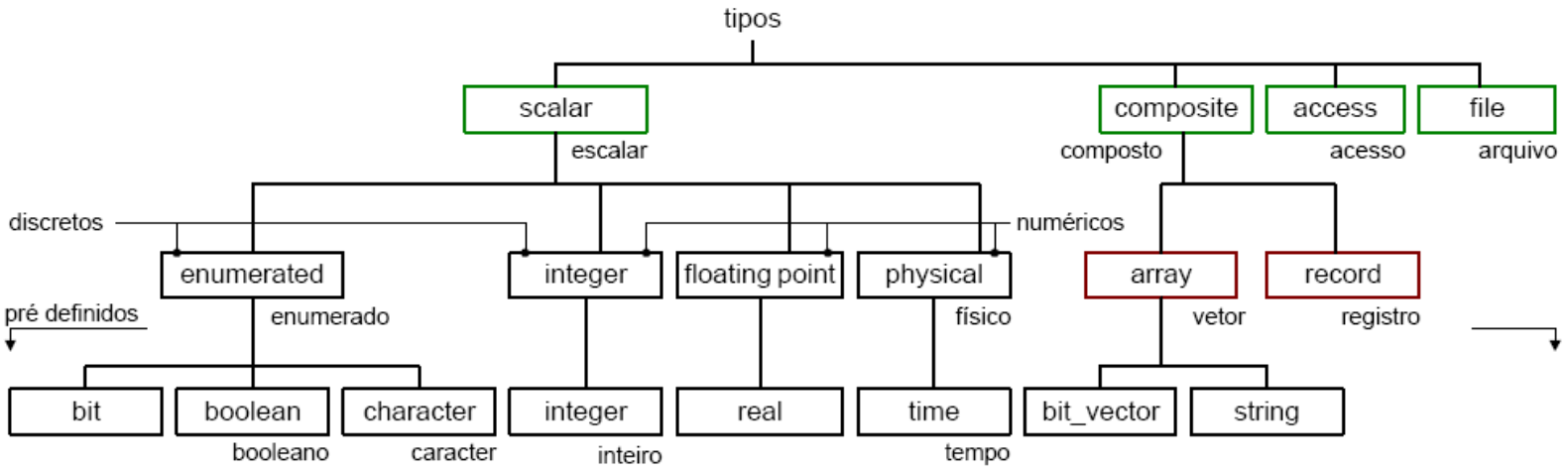
- **Exemplo:**
  - pacote padrão da linguagem:
    - objeto do tipo BOOLEAN pode assumir valores FALSE e TRUE
      - definidas operações lógicas como AND e OR
    - objeto do tipo INTEGER
      - operações lógicas não são definidas

Tipos

- **Classes de tipos da linguagem:**
  SCALAR COMPOSITE ACCESS FILE

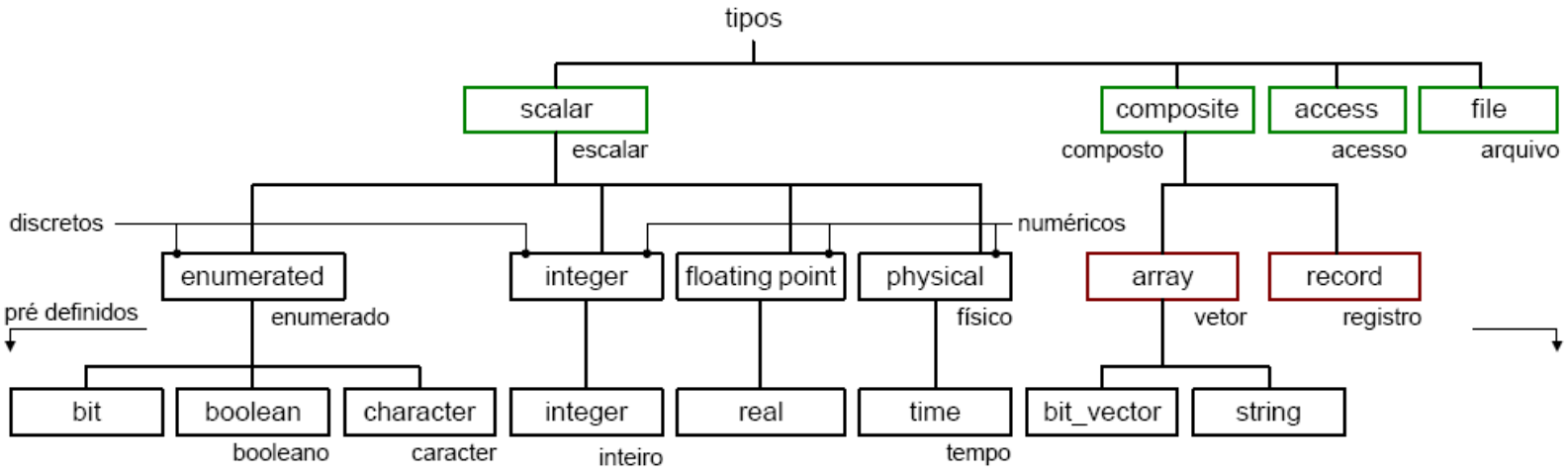
- **Tipos escalares:**
  - são ordenados
  - os valores não contêm outros elementos

- **Tipos compostos:**
  - os valores contêm outros elementos
  - ARRAY os elementos são do mesmo tipo
  - RECORD os elementos podem ser de tipos diferentes



Tipos

- **ACCESS**
  - empregado para valores alocados dinamicamente.
- **FILE**
  - empregado para representação de arquivos;
  - um objeto do tipo FILE
    - contém valores armazenados no sistema de arquivos do ambiente



Tipos

- **Conforme visto no capítulo 2:**
  - é possível definir novos tipos

- **Definição de um novo tipo:**
  - conjunto de valores que um objeto pode armazenar :
    - declaração TYPE
  - conjunto de operações que podem ser realizadas com o objeto :
    - estabelecidas através de funções

Definição de tipos enumerados

- **Tipos enumerados** → pertencem à classe dos tipos escalares
- **Pré definidos os tipos**: BIT BOOLEAN CHARACTER
- **Declaração de um novo tipo**:

```
TYPE temperatura IS (baixa, media, alta);
TYPE cores IS  ('R', 'G', 'B');
```

- **Valores que o objeto pode assumir**:

  - apresentados ordenadamente na especificação

- **Primeiro elemento definido no tipo** → menor valor
- **Último elemento definido** → maior valor

Definição de físicos

- **Pertencem à classe dos escalares**
- **Empregados para representação de grandezas físicas**
  - exemplo: tensão, corrente, velocidade, etc.
- **Definida uma unidade básica e unidades sucessivas múltiplas desta**
- **Tipo TIME definido no pacote padrão para a grandeza tempo**
  - unidade básica o valor fs
- **Exemplo da definição um tipo físico**
  - denominado resitencia
  - unidade básica o valor ohm

```
TYPE resistencia IS RANGE 0 TO 1E7
  UNITS ohm;
        Kohm = 1000 ohm;
        Mohm = 1000 Kohm;
  END UNITS;
```

Definição de sub-tipos

- **Sub-tipo** → estabelece um novo limite no campo definido

- **Sub-tipo e o tipo não são tipos distintos**:
  - apenas o limite é redefinido

- **Operações e transferência de valores entre**:
  - objetos de um tipo e sub-tipo são permitidas

- **Exemplo dos sub-tipos escalares natural positive definidos no pacote padrão**:

```
TYPE    integer  IS RANGE -2147483648 TO 2147483647;
SUBTYPE natural  IS integer RANGE 0 TO integer'HIGH;
SUBTYPE positive IS integer RANGE 1 TO integer'HIGH;
```

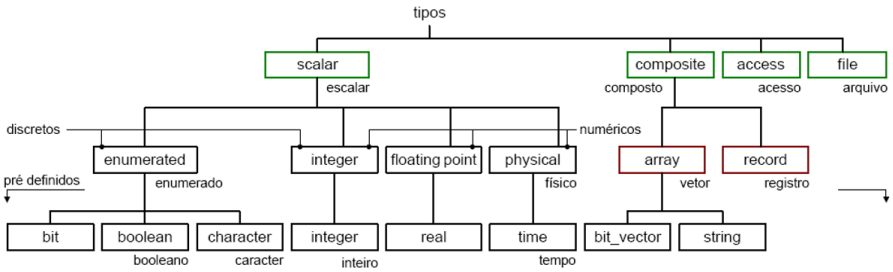
- **Definição de um sub-tipo**:
  - palavra reservada SUBTYPE
  - nome do sub-tipo
  - nome do tipo ao qual ele pertence
  - novos limites estabelecidos

Definição de sub-tipos - exemplo

- **Linha 18**: válida para qualquer valor transferido para b_type
- **Linha 19**: bs_btype recebendo valor diferente de: vermelho verde azul
  - mensagem de erro na simulação
  - valor fora do limites definidos para o sub-tipo cores_primarias

```
 2 PACKAGE tipos IS
 3   TYPE cores IS (vermelho, verde, azul, marrom, laranja, amarelo, violeta,
                                                                    branco);
 4   SUBTYPE cores_primarias IS cores RANGE vermelho TO azul;
 5 END tipos;
 6
 7 USE WORK.tipos.ALL;
 8
 9 ENTITY array_t3 IS
10   PORT (a_type   : IN  cores;
11         b_type   : OUT cores;
12         b_sbtype : OUT cores_primarias);
13 END array_t3;
14
15 ARCHITECTURE teste OF array_t3 IS
16
17 BEGIN
18   b_type   <= a_type;  -- valida para qualquer valor
19   b_sbtype <= a_type;  -- valida para os valores: vermelho, verde, azul
20 END teste;
```

Definição de tipos registro



- **Tipo RECORD**:

  - tipo composto de elementos nomeados

- **Elementos compõem**:

  - não necessitam ser do mesmo tipo

- **Um objeto do tipo registro**:

  - é referenciado por um identificador

- **Elemento do objeto**: referenciado pelo nome que o identifica

  identificador_objeto.nome_elemento

- **Modo de atribuição de valores**:

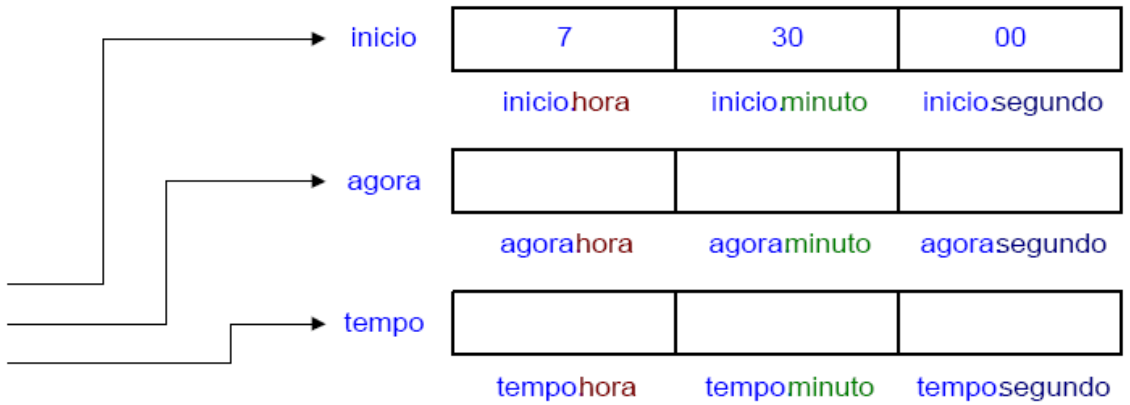
  - similar ao empregado nos tipos da classe vetor

Definição de tipos registro

- **Tipo registro denominado**: instante
- **Composto dos elementos** : hora minuto segundo
- **Objetos declarados segundo este tipo**: inicio agora tempo
- **Exemplo**:
  - campo minuto da constante inicio referenciado no formato: inicio. minuto

```
Type instante Is
 Record
  hora   : Integer Range 0 To 23;
  minuto : Integer Range 0 To 59;
  segundo: Integer Range 0 To 59;
 End Record;

Constant inicio : instante := (7,30,00);
Signal   agora  : instante;
Variable tempo : instante;
```



agora <= (17,10,05);

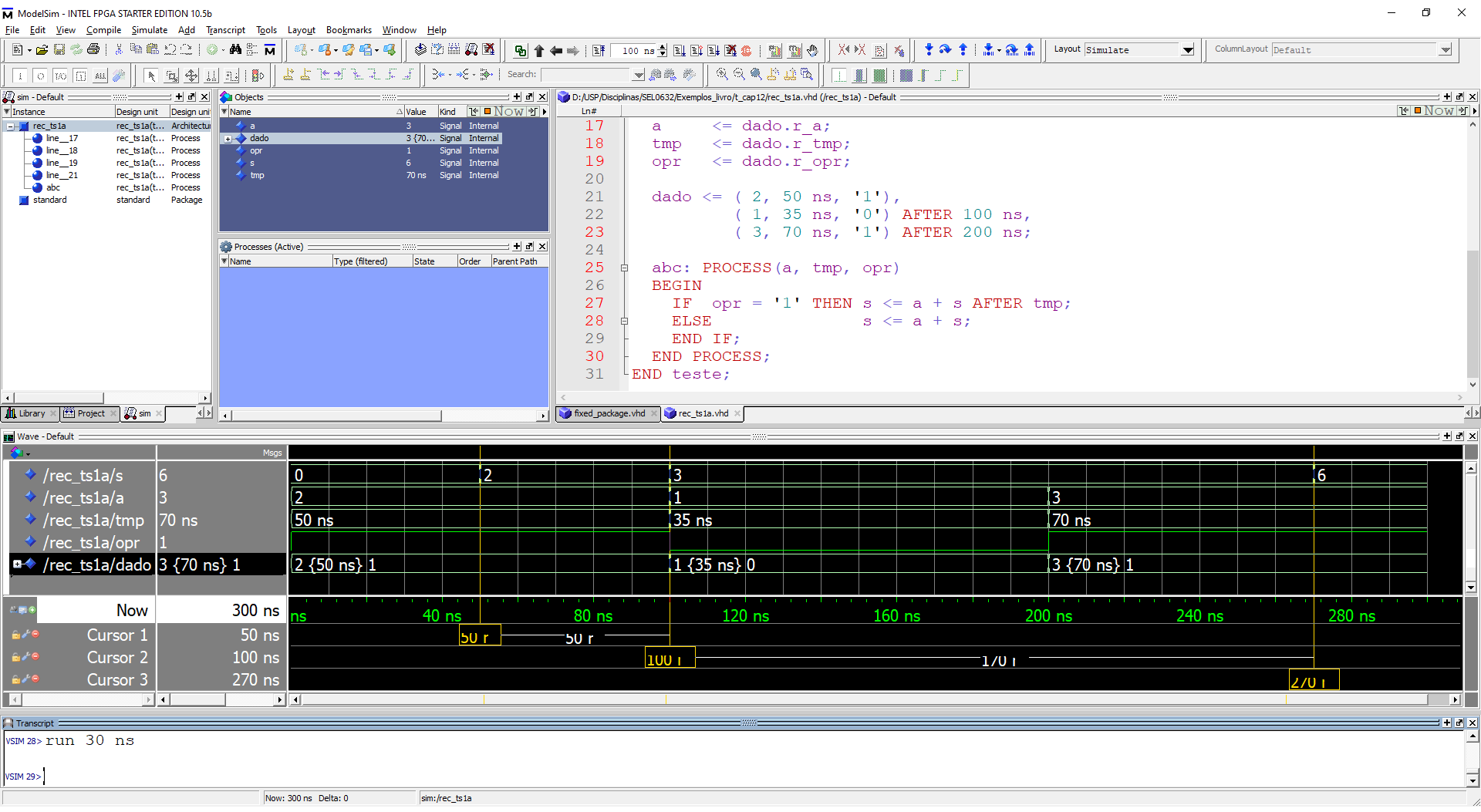
tempo.minuto := 17;

Definição de tipos registro (descrição empregando um tipo registro)

- **sinais_controle**: tipo REGISTER
- **3 elementos**: r_a tipo INTEGER, r_tmp tipo TIME , r_opr tipo BIT
- **Sinal dado**: tipo sinais_controle
- **Estímulos são aplicados através da declaração que inicia na linha 21**

```
 1 ENTITY rec_ts1 IS
 2 END rec_ts1;
 3
 4 ARCHITECTURE teste OF rec_ts1 IS
 5 TYPE sinais_controle IS
 6  RECORD r_a   : INTEGER RANGE 0 TO 15;
 7         r_tmp : TIME;
 8         r_opr : BIT;
 9  END RECORD;
10
11 SIGNAL a, s   : INTEGER RANGE 0 TO 31;
12 SIGNAL tmp    : TIME;
13 SIGNAL opr    : BIT;
15 SIGNAL dado   : sinais_controle;
16 BEGIN
```

```
17  a   <= dado.r_a;
18  tmp <= dado.r_tmp;
19  opr <= dado.r_opr;
20
21  dado <= ( 2, 50 ns,'1'),
22          ( 1, 35 ns,'0') AFTER 100 ns,
23          ( 3, 70 ns,'1') AFTER 200 ns;
24
25   abc: PROCESS(a, tmp, opr)
26   BEGIN
27    IF opr = '1' THEN s <= a +s AFTER tmp;
28    ELSE              s <= a +s;
29    END IF;
30  END PROCESS;
31 END teste;
```


ModelSim - INTEL FPGA STARTER EDITION 10.5b
File Edit View Compile Simulate Add Transcript Tools Layout Bookmarks Window Help
Objects
Name Value Kind
a 3 Signal Internal
dado 3 {70... Signal Internal
opr 1 Signal Internal
s 6 Signal Internal
tmp 70 ns Signal Internal
Processes (Active)
Name Type (filtered) State Order Parent Path
D:/USP/Disciplinas/SEL0632/Exemplos_livro/t_cap12/rec_ts1a.vhd (/rec_ts1a) - Default
17 a <= dado.r_a;
18 tmp <= dado.r_tmp;
19 opr <= dado.r_opr;
20
21 dado <= (2, 50 ns, '1'),
22 (1, 35 ns, '0') AFTER 100 ns,
23 (3, 70 ns, '1') AFTER 200 ns;
24
25 abc: PROCESS(a, tmp, opr)
26 BEGIN
27 IF opr = '1' THEN s <= a + s AFTER tmp;
28 ELSE s <= a + s;
29 END IF;
30 END PROCESS;
31 END teste;
Wave - Default
/rec_ts1a/s 6
/rec_ts1a/a 3
/rec_ts1a/tmp 70 ns
/rec_ts1a/opr 1
/rec_ts1a/dado 3 {70 ns} 1
Now 300 ns
Cursor 1 50 ns
Cursor 2 100 ns
Cursor 3 270 ns
40 ns 80 ns 120 ns 160 ns 200 ns 240 ns 280 ns
Transcript
VSIM 28> run 30 ns
VSIM 29>

Definição de tipos vetor

- **Tipo composto da classe vetor** → agrupa elementos do mesmo tipo em um objeto
- **Vetor unidimensional**:
  - identificação de um elemento → feita através de um índice
- **Vetor multidimensional**:
  - um conjunto de índices identifica o elemento

Definição de tipos vetor

- **Relembrando agregados**
  - é uma expressão indicando o valor de um tipo composto
  - valores especificados pelo valor de cada elemento do tipo composto

```
 CONSTANT um   : BIT := '1';

-- valor 00010, agregado notacao posicional
                              s2 <= ('0','0','0','1','0');

-- valor 00011, agregado associacao por identificadores
                              s3 <= (1=>'1', 0=>'1', OTHERS=>'0');

-- valor 00100, agregado com operacoes
                              s4 <= (zero, '0', um OR '0', '0', '0');

-- valor 00101, agregado faixa discreta
                              s5 <= (4 DOWNTO 3 =>'0', 1=>'0', OTHERS=>'1');
```

Definindo um vetor

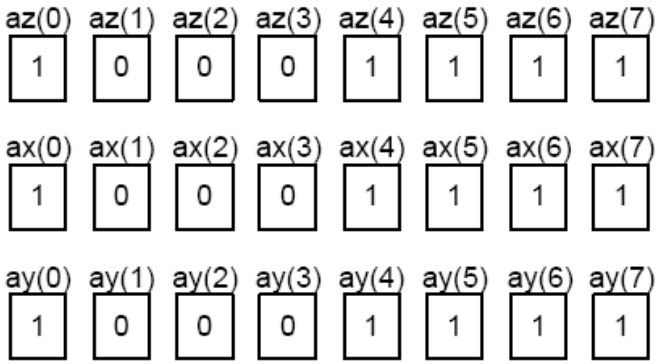
- **Definição de um vetor** → deve ser declarado:
  - identificador limites do vetor tipo de elementos que compõe o vetor
- **Limites do vetor**:
  - são definidos por tipos enumerados
  - assumido o tipo INTEGER quando não especificado

```
--Type vetor_ax   Is Array (Integer Range 0 To 7) Of Bit;
  Type vetor_ax   Is Array (0 To 7) Of Bit; -- assumido tipo Integer

Signal   ax : vetor_ax;
Signal   ay : vetor_ax;
Constant az : vetor_ax := (0 => '1', 1 To 3 => '0', Others => '1');

ax(0 TO 7) <= "10001111";

ay <= (0 => '1', 1 To 3 => '0', Others => '1');
```



Definindo um vetor - (descrição completa)

- **Descrição conforme a figura anterior**
- **Vetor unidimensional composto por 7 elementos do tipo bit**
- **Linhas 5 e 6 do código são equivalentes**

  - ausência da declaração do tipo dos limites, é assumido o tipo INTEGER

```
 1 ENTITY teste_a IS
 2 END teste_a;
 3
 4 ARCHITECTURE teste OF teste_a IS
 5   TYPE vetor_ax    IS ARRAY (INTEGER RANGE 0 TO 7) OF BIT;
 6 --TYPE vetor_ax    IS ARRAY (0 TO 7) OF BIT;  -- assumido tipo inteiro
 7
 8   SIGNAL   ax : vetor_ax;
 9   SIGNAL   ay : vetor_ax;
10   CONSTANT az : vetor_ax := (0 => '1', 1 TO 3 => '0', OTHERS => '1');
11
12 BEGIN
13   ax(0 TO 7) <= "10001111";
14   ay <= (0 => '1', 1 TO 3 => '0', OTHERS => '1');
15 END teste;
```

Definindo um vetor - (outro exemplo)

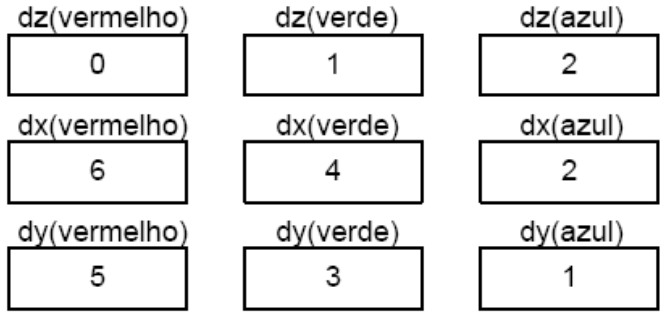
- **Vetor unidimensional**
  - índice: tipo cores (enumerado definido no código)
- **Tipo cores contém três valores**: vermelho verde azul
- **Tipo vetor_dx**
  - elementos do tipo Integer
  - índice do vetor: tipo cores
- **Note**: na ausência da definição da faixa de valores do tipo (Range vermelho To azul) é assumida a faixa completa de valores

```
Type cores Is (vermelho, verde, azul);
Type vetor_dx Is Array (cores Range vermelho To azul) Of Integer Range 0 To 7;
--Type vetor_dx Is Array (cores) Of Integer Range 0 To 7; -- assumida a faixa toda do tipo;

Signal dx: vetor_dx;
Signal dy: vetor_dx;
Constant dz: vetor_dx := (0,1,2); -- definindo valores iniciais

dx(azul) <= 2; dx(verde) <= 4; dx(vermelho) <= 6;

dy(vermelho To azul) <= (5,3,1);
```



Definindo um vetor - (continuação do exemplo anterior)

- **Tipo vetor_dx**
  - elementos do tipo Integer
  - índice do vetor: tipo cores
- **Note a transferência de valores** - linhas 14 e 15

```
 1 ENTITY teste_d IS
 2 END teste_d;
 3
 4 ARCHITECTURE teste OF teste_d IS
 5  TYPE cores IS  (vermelho, verde, azul);
 6  TYPE vetor_dx  IS ARRAY (cores RANGE vermelho TO azul) OF INTEGER RANGE 0 TO 7;
 7 --TYPE vetor_dx  IS ARRAY (cores) OF INTEGER RANGE 0 TO 7;  -- faixa completa
 8
 9   SIGNAL   dx: vetor_dx;
10   SIGNAL   dy: vetor_dx;
11   CONSTANT dz: vetor_dx  := (0,1,2);  -- definindo valores iniciais
12
13 BEGIN
14   dx(azul) <= 2;   dx(verde) <= 4;    dx(vermelho) <= 6;
15   dy(vermelho TO azul) <= (5,3,1);
16 END teste;
```

Definindo um vetor com limites em aberto

- ***unconstrained array***:

  número de elementos não é especificado na declaração do tipo

- **Definição dos limites do vetor** $\rightarrow$ estabelecida:

  - na declaração de um objeto que emprega este tipo
  - na declaração de um sub-tipo do tipo
  - num subprograma (a partir do parâmetro real passado ao subprograma)

Definindo um vetor com limites em aberto

- **Definição do tipo**:
  - após a palavra RANGE é introduzido símbolo <>
    denominado *box* (indica o limite em aberto)

- **Exemplos dos vetores BIT_VECTOR STRING definidos no pacote padrão**:

```
-- definidos no pacote standard:
TYPE bit_vector IS ARRAY (NATURAL  RANGE <>) OF bit;
TYPE string     IS ARRAY (POSITIVE RANGE <>) OF character;
```

- **Definição de um tipo equivalente a BIT_VECTOR**:

```
-- tipo equivalente ao bit_vector
TYPE   vetor_bit IS ARRAY (NATURAL  RANGE <>) OF BIT;

SIGNAL x1 : vetor_bit(3 DOWNTO 0);                    -- objeto que emprega o tipo
```

- **Os limites são estabelecidos na declaração do sinal x1**
- **Declaração do sinal:**
  - formato igual a definição dos limites de um tipo BIT_VECTOR

Definindo um vetor com limites em aberto (exemplo)

- **Limites definidos**: objeto que emprega um tipo e declaração de um sub-tipo
- **Tipo sem_limite**: número de elementos do tipo CHARACTER não especificado
- **Sub-tipo com_limite**: especificada faixa de 0 a 3
- **Sinal saida_1**: limite definido pelo sub-tipo com_limite
- **Sinal saida_2**: necessita da definição dos limites, ele é do tipo sem_limite

```
 1 -- arquivo teste2a.vhd
 2 PACKAGE tipos IS
 3  TYPE    sem_limite IS ARRAY(NATURAL RANGE <>) OF CHARACTER; -- nao defido lim.
 4  SUBTYPE com_limite IS sem_limite(0 TO 3);          -- sub-tipo estabelece lim.
 5 END tipos;
 6
 7 USE WORK.tipos.ALL;
 8
 9 ENTITY teste2a IS
10   PORT(saida_1 : OUT com_limite;              -- limites definidos pelo sub_tipo
11        saida_2 : OUT sem_limite(4 DOWNTO 0)); -- objeto define os limites do tipo
12 END teste2a;
13
14 ARCHITECTURE teste OF teste2a IS
15 BEGIN
16   saida_1 <= ('a','b','c','d');     -- transf. valor para o objeto do tipo
17   saida_2 <= ('a','b','c','d','f'); -- transf. valor para o objeto do sub-tipo
18 END teste;
```

Definindo um vetor com limites em aberto (exemplo)

- **Limites definidos**: num subprograma (a partir do parâmetro real passado)
- **Tipo sem_limite**: número de elementos do tipo CHARACTER não especificado

```
 2 PACKAGE tipos IS
 3  TYPE    sem_limite IS ARRAY(NATURAL RANGE <>) OF CHARACTER; -- lim. nao def.
 4 END tipos;
 6 USE WORK.tipos.ALL;
 7
 8 ENTITY teste2b IS
 9   PORT(entrada : IN  sem_limite(4 DOWNTO 0); -- objeto define os limites do tipo
10        saida   : OUT sem_limite(4 DOWNTO 0); -- objeto define os limites do tipo
11 END teste2b;
12
13 ARCHITECTURE teste OF teste2b IS  ------ parametro real ira' definir os limites
14 --                                /
15 FUNCTION reverte (vetor: sem_limite) RETURN sem_limite IS
16   VARIABLE vetor_reverso : sem_limite(vetor'RANGE);
17 BEGIN
18   FOR i IN vetor'RANGE LOOP
19      vetor_reverso(vetor'LENGTH-1-i):= vetor(i);
20   END LOOP;
21   RETURN vetor_reverso;
22 END reverte;
23
24 BEGIN
25   saida <= reverte(entrada);  -- parametro real "entrada" com limites definidos
26 END teste;
```

Vetor composto de elementos do tipo vetor

- **Exemplos**: vetores compostos com elementos do tipo vetor
  declarações de constantes empregando os vetores
- **Definição**: suportada pela maioria das ferramentas de síntese
- **vetor_2d:** composto 8 elementos do tipo bit_vector, cada 4 quatro bits
- **vetor_3d:** composto 3 elementos do tipo vetor_2d (estrutura três dimensões)

```
--Type vetor_2d  Is Array (Integer Range 0 To 7) Of Bit_Vector(3 Downto 0);
  Type vetor_2d  Is Array (0 To 7) Of Bit_Vector(3 Downto 0);  -- limites : assumido tipo Integer
  Type vetor_3d  Is Array (0 T0 2) Of vetor_2d;

  Constant va: vetor_2d := (('1','0','0','0'), Others => ('1','0','1','0'));
  Constant vb: vetor_2d := (("1000"),Others => ("1010"));
  Constant vc: vetor_2d := (0=>('1','1','0','0'), Others => ('1','0','0','1'));
  Constant vd: vetor_2d := (("1100"), Others => ('1','0','0','1'));
  Constant ve: vetor_2d := (0 To 2 =>('0','1','1','0'), Others => ('1','0','1','1'));
  Constant vg: vetor_2d := (Others => (Others => ('1')));  -- preenche com mesmo valor
  Constant vh: vetor_3d := (Others => (Others => (Others => ('1'))));
  Constant vi: vetor_3d := ((Others => ('0','0','0','0')),(Others => ('0','0','0','1')),(Others => ('0','0','1','0')));
  Constant vj: vetor_3d := (("0000", Others =>"0000"),("0001", Others =>"0001"),(Others =>"0010"));
```

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

Vetor composto de elementos do tipo vetor

- **Exemplos de transferência**
- **Seleção de um elemento do vetor**: objeto(indice_v2)(indice_v1)
  objeto(indice_v3)(indice_v2)(indice_v1)

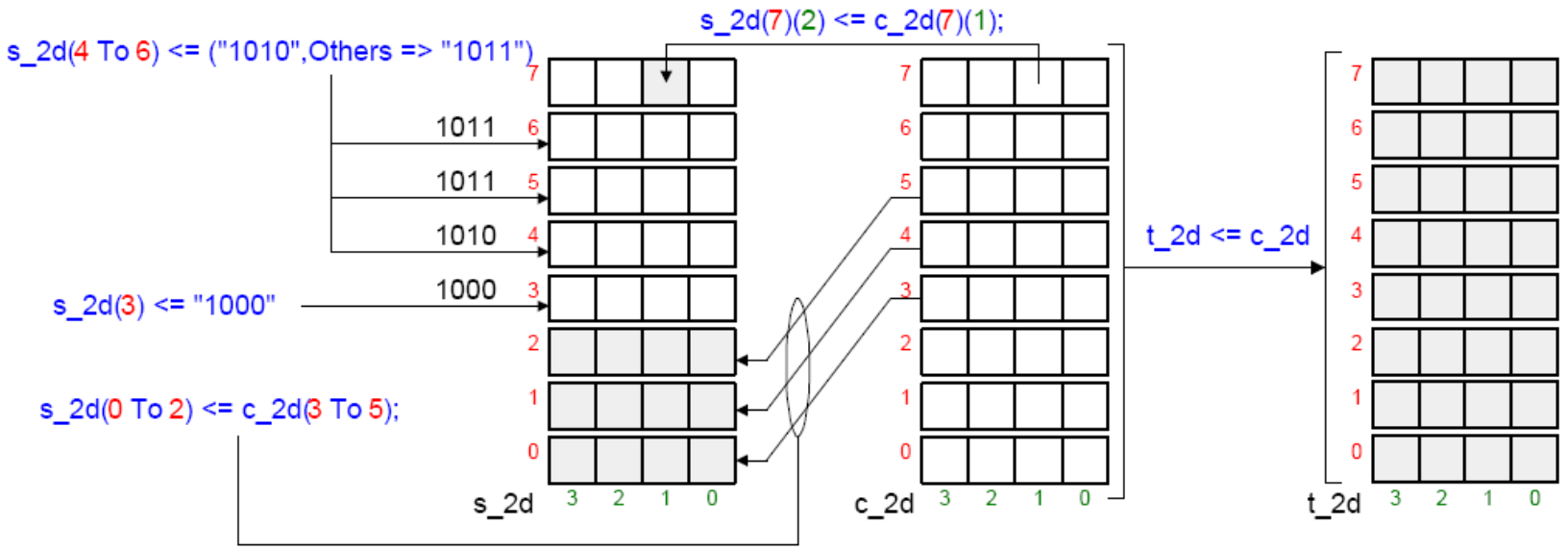
  - indice_v1: corresponde índice do vetor no nível mais baixo na declaração

```
 4 ARCHITECTURE teste OF teste_e1 IS
 5  TYPE vetor_2d  IS ARRAY (0 TO 7) OF BIT_VECTOR(3 DOWNTO 0);
 6  TYPE vetor_3d  IS ARRAY (0 TO 2) OF vetor_2d;
 7
 8  SIGNAL   s_2d, t_2d: vetor_2d;
 9  CONSTANT c_2d: vetor_2d := (0 TO 2 =>('0','0','0','0'), OTHERS
                                          =>('1','0','1','1'));
10  SIGNAL   s_3d, t_3d: vetor_3d;
11
12 BEGIN
13   s_2d(7)(2) <= c_2d(7)(1);                    -- 1 elemento
14   s_2d(3) <= "1000";                           -- 1 indice
15   s_2d(4 TO 6) <= ("1010", OTHERS => "1011"); -- faixa de indices
16   s_2d(0 TO 2) <= c_2d(3 TO 5);                -- faixa de indices
17   t_2d <= c_2d;                                -- vetor completo
18
19   s_3d(2)(7)(3) <= c_2d(7)(1);                   -- 1 elemento
20   s_3d(0)(1)(2 DOWNTO 0) <= c_2d(3)(3 DOWNTO 1); -- faixa
21   s_3d(1)(2 TO 3) <= c_2d(5 TO 6);               -- faixa
22   t_3d(2) <= c_2d;                               -- faixa
23   t_3d(0 TO 1) <= c_2d & c_2d;                   -- faixa
```

Vetor composto de elementos do tipo vetor

- **Vetor composto de elementos do tipo vetor**:
  - possível delimitar de blocos e faixas
- **Mais flexível nas operações** (em comparação com tipos multidimensionais)
- **Exemplo de operações** (vetores com 2 dimensões)

```
TYPE vetor_2d  IS ARRAY(0 TO 7) OF BIT_VECTOR(3 DOWNTO 0);
```



Vetor composto de elementos do tipo vetor

- **Exemplo de operações** (vetores com 2 e 3 dimensões)

```
TYPE vetor_2d  IS ARRAY(0 TO 7) OF BIT_VECTOR(3 DOWNTO 0);
TYPE vetor_3d  IS ARRAY(0 TO 2) OF vetor_2d;
```