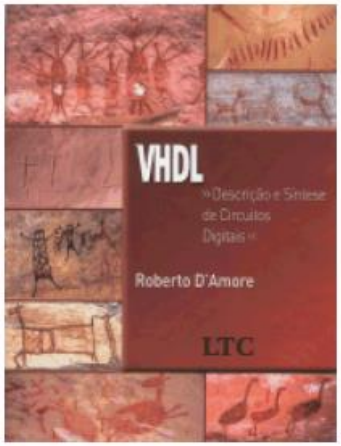
SEL632 – Linguagens de Descrição de Hardware
Aula 12 – Definição de Tipos (2)

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 vetor
- Definição de tipos registro
- Síntese de memórias

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

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)

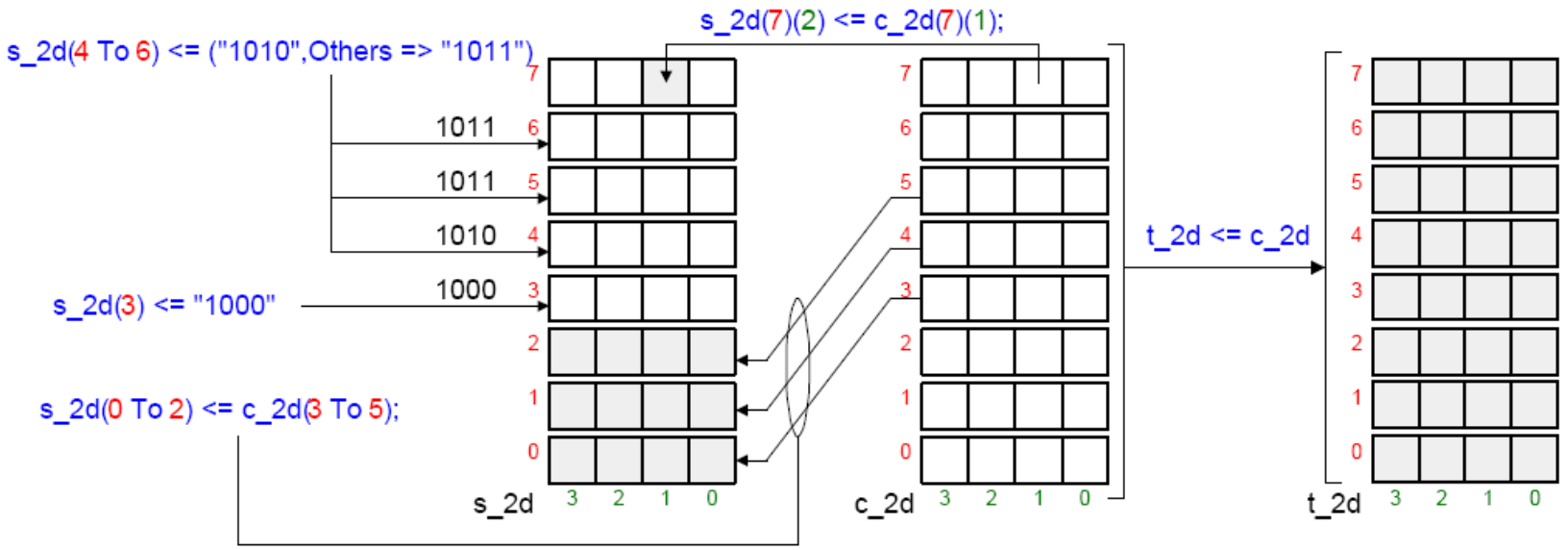
  - indice_v1: corresponde índice do vetor no nível mais baixo na declaração

```
 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)

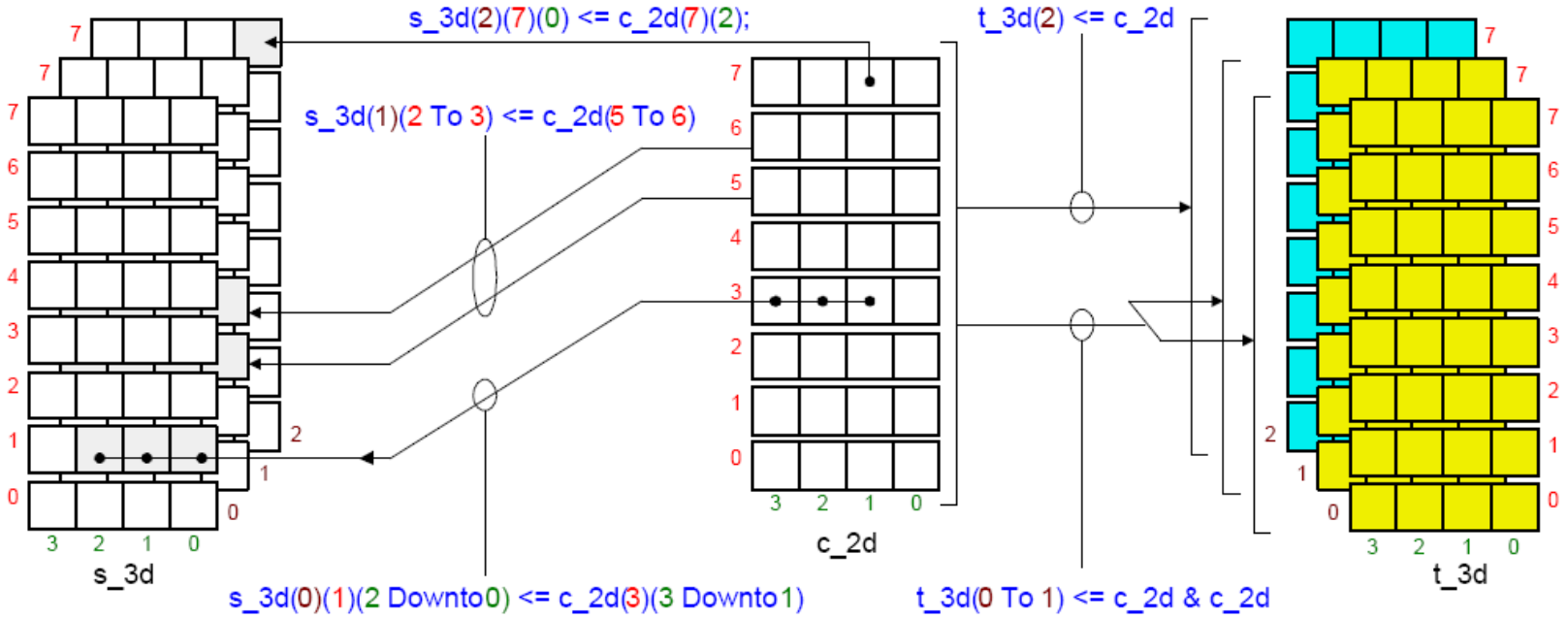
```
TYPE vetor_2d  IS ARRAY(0 TO 7) OF BIT_VECTOR(3 DOWNTO 0);
```



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;
```



Vetor multidimensional

- **Exemplos**: vetores multidimensionais
  declarações de constantes empregando os vetores
- **Definição**: não suportada por muitas ferramentas de síntese
- **vetor_2m:** arranjo de 2 dimensões de elementos tipo bit
- **vetor_3d:** arranjo de 3 dimensões de elementos tipo bit

```
--Type vetor_2m  Is Array (Integer Range 0 To 7, 3 Downto 0) Of Bit;
  Type vetor_2m  Is Array     (0 To 7, 3 Downto 0) Of Bit; -- limites: assumido tipo Integer
  Type vetor_3m  Is Array (0 To 2, 0 To 7, 3 Downto 0) Of Bit;

  Constant ma: vetor_2m := (('1','0','0','0'), Others => ('1','0','1','0'));
  Constant mb: vetor_2m := (("1000"),Others => ("1010"));
  Constant mc: vetor_2m := (0=>('1','1','0','0'), Others => ('1','0','0','1'));
  Constant md: vetor_2m := (("1100"), Others => ('1','0','0','1'));
  Constant me: vetor_2m := (0 To 2 =>('0','1','1','0'), Others => ('1','0','1','1'));
  Constant mg: vetor_2m := (Others => (Others => ('1')));  -- preenche com mesmo valor
  Constant mh: vetor_3m := (Others => (Others => (Others => ('1'))));
  Constant mi: vetor_3m := ((Others => ('0','0','0','0')),(Others => ('0','0','0','1')),(Others => ('0','0','1','0')));
  Constant mj: vetor_3m := (("0000", Others =>"0000"),("0001", Others =>"0001"),(Others =>"0010"));
```

me(a,b)

a \ b	3	2	1	0
7	1	0	1	1
6	1	0	1	1
5	1	0	1	1
4	1	0	1	1
3	1	0	1	1
2	0	1	1	0
1	0	1	1	0
0	0	1	1	0

Vetor multidimensional

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

```
 1 ENTITY teste_f1 IS
 2 END teste_f1;
 3
 4 ARCHITECTURE teste OF teste_f1 IS
 5   TYPE vetor_2m  IS ARRAY         (0 TO 7, 3 DOWNTO 0) OF BIT;
 6   TYPE vetor_3m  IS ARRAY (0 TO 2, 0 TO 7, 3 DOWNTO 0) OF BIT;
 7
 8  SIGNAL   s_2m, t_2m : vetor_2m;
 9  CONSTANT c_2m: vetor_2m := (0 TO 2 =>('0','0','0','0'), OTHERS
                                          =>('1','0','1','1'));
10  SIGNAL   s_3m, t_3m:  vetor_3m;
11
12 BEGIN
13   s_2m(7,2) <= c_2m(7,1);                  -- 1 elemento
14   t_2m <= c_2m;                            -- vetor completo
15
16   s_3m(2,7,3) <= c_2m(7,1);                 -- 1 elemento
17   t_3m <= s_3m;
18 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

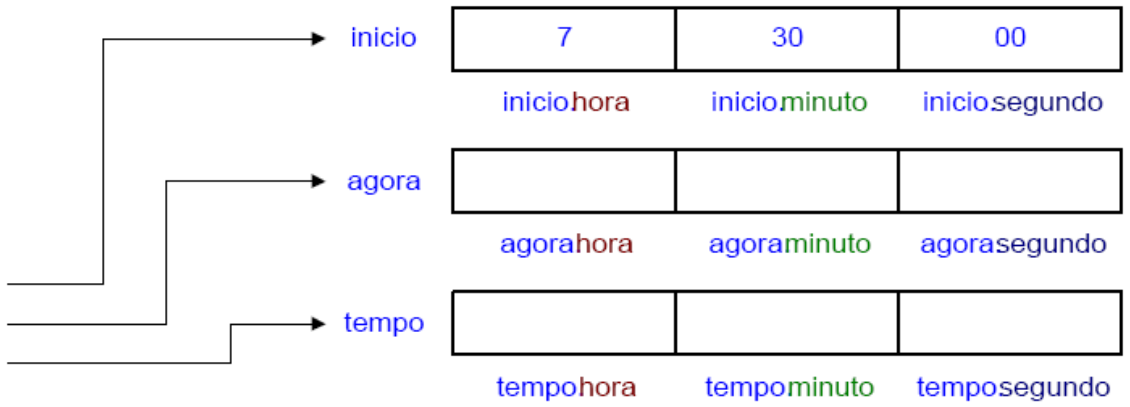
- **Modo de atribuição de valores**:

  - 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;
```



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
```

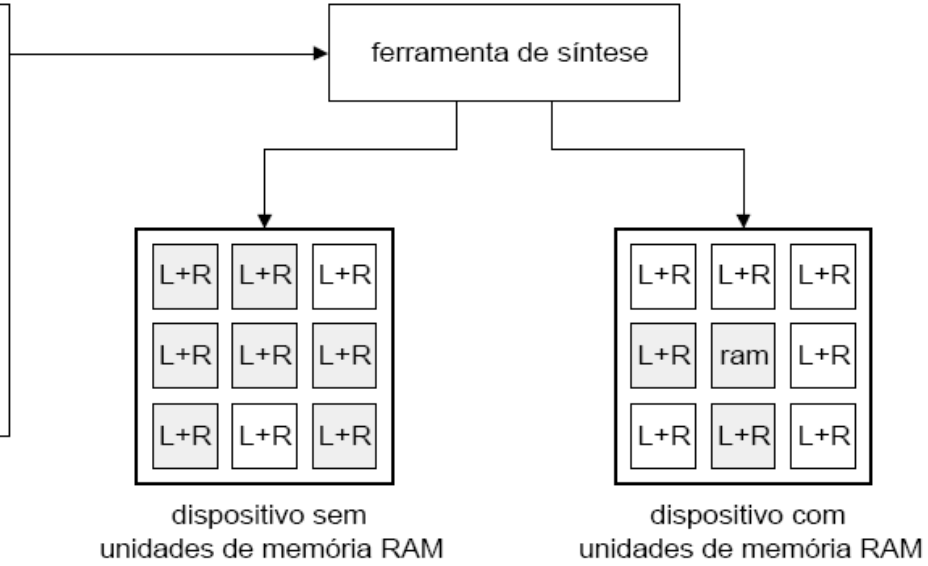
```
17  a   <= dado.r_a;
18  tmp <= dado.r_tmp;
19  opr <= dado.r_opr;
20
21  dado <= ( 2, 50 ns,'1'),
22          ( 1, 35 ns,'0') AFTER 100 ns,
23          ( 3, 70 ns,'1') AFTER 200 ns;
24
25   abc: PROCESS(a, tmp, opr)
26   BEGIN
27    IF opr = '1' THEN s <= a +s AFTER tmp;
28    ELSE             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

Síntese de memórias

- **Memória em um projeto**:
  - deve considerar a tecnologia escolhida para a implementação do circuito
  - é um recuso de grande impacto no custo final (área ou elementos disponíveis)
- **Algumas Famílias de FPGA's**:
  - contém unidades de memória RAM
- **Ferramentas de síntese podem empregar este recurso**

```
Entity ram_3 IS
 Port (dado_in  : In  Bit_Vector(3 Downto 0);
       dado_out : OUT Bit_Vector(3 Downto 0);
       endereco : IN  Integer Range 0 TO 7;
       we       : IN  BIT);
End ram_3;

Architecture teste Of ram_3 Is
 Type   org_mem Is Array(0 TO 7) Of Bit_Vector(3 Downto
0);
 Signal memoria : org_mem;
Begin
 Process(endereco, we)
 Begin
  If we'Event And we='1' Then memoria(endereco) <=
dado_in;
  End If;
 End Process;
 dado_out <= memoria(endereco);
End teste;
```

= unidades empregadas



Síntese de memórias

- **Organização destas unidades de memória**:
  - depende do fabricante e família do dispositivo

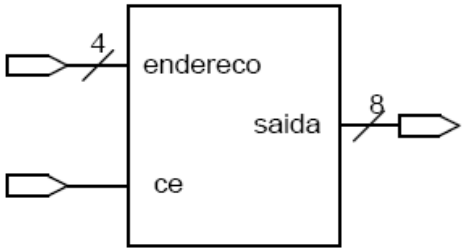
- **Unidade de memória RAM**:
  - pode ser empregada para implementar memórias ROM
  - ferramenta de síntese define os dados na programação do dispositivo
  - dados permanecem ao longo da operação

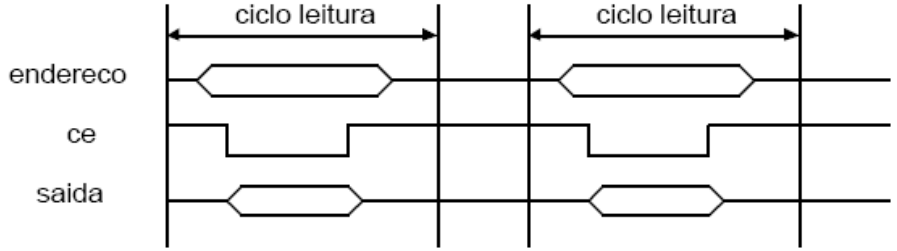
- **Descrição de memórias emprega normalmente**:
  - vetores compostos com elementos do tipo vetor

- **Outras opções**:
  - memórias ROM
    declarações: CASE WHEN WITH SELECT

Memórias ROM exemplo

- **Memória ROM de 8 bit de dados por 15 posições de endereço**
- **Linhas**: endereco definem a posição dos dados
- **Linhas**: saida informação armazenada
- **Linha**: ce habilita a memória em nível lógico baixo





Memórias ROM (exemplo - código) (continua prox. imagem)

- **Organização da memória** → declaração do tipo arranjo_memoria (linha 12)
- **Tipo arranjo_memoria define o número de bits**: 8
- **Número de endereços** → definido pela constante dados que emprega o tipo
- **dados** → vetor com 16 elementos, valores definidos na declaração do objeto

```
 1 LIBRARY ieee;
 2 USE ieee.std_logic_1164.ALL;
 3 USE ieee.numeric_std.ALL;
 4
 5 ENTITY rom_1 IS
 6   PORT(endereco : IN  UNSIGNED(3 DOWNTO 0); -- valores de 0 a 15
 7        ce       : IN  STD_LOGIC;           -- habitita memoria
 8        saida    : OUT UNSIGNED(7 DOWNTO 0));-- saida de dados da memoria
 9 END rom_1 ;
10
11 ARCHITECTURE teste OF rom_1 IS
12   TYPE     arranjo_memoria IS ARRAY (NATURAL RANGE <>) OF UNSIGNED(7 DOWNTO 0);
13   CONSTANT dados : arranjo_memoria(0 TO 15) :=
14   ("00000011","10011111","00100101","00001101","10011000", -- valores pos. 0,1,2,3,4
15    "01001001","01000001","00011111","00011001","00001001", -- valores pos. 5,6,7,8,9
16    "01001001","01000001","00010111","00000001","01001001","01000110"); -- restantes
17
18
19
```

Memórias ROM (exemplo - código) (continuação da imagem anterior)

- **Seleção um elemento do objeto** = **operação de leitura um endereço da memória**

- **A posição do vetor é dada por um tipo inteiro**:
  - necessário converter o tipo UNSIGNED de endereco para INTEGER
  - realizado pela função to_integer

- **Habilitação da interface de saída saida**: declaração WHEN ELSE

```
 4
 5 ENTITY rom_1 IS
 6   PORT(endereco : IN  UNSIGNED(3 DOWNTO 0); -- valores de 0 a 15
 7        ce       : IN  STD_LOGIC;           -- habitita memoria
 8        saida    : OUT UNSIGNED(7 DOWNTO 0));-- saida de dados da memoria
 9 END rom_1 ;
10
11 ARCHITECTURE teste OF rom_1 IS
12   TYPE     arranjo_memoria IS ARRAY (NATURAL RANGE <>) OF UNSIGNED(7 DOWNTO 0);
13   CONSTANT dados : arranjo_memoria(0 TO 15) :=
14   ("00000011","10011111","00100101","00001101","10011000", -- valores pos. 0,1,2,3,4
15    "01001001","01000001","00011111","00011001","00001001", -- valores pos. 5,6,7,8,9
16    "01001001","01000001","00010111","00000001","01001001","01000110"); -- restantes
17 BEGIN
18   saida <= dados(to_integer(endereco)) WHEN ce='0' ELSE (OTHERS=>'Z');
19 END teste;
```

Memórias ROM (síntese)

- **Memória do tipo ROM inferida com sucesso**
- **Resultados da síntese da descrição** - tecnologia alvo sem unidades de memória
- **Lógica combinacional empregada**

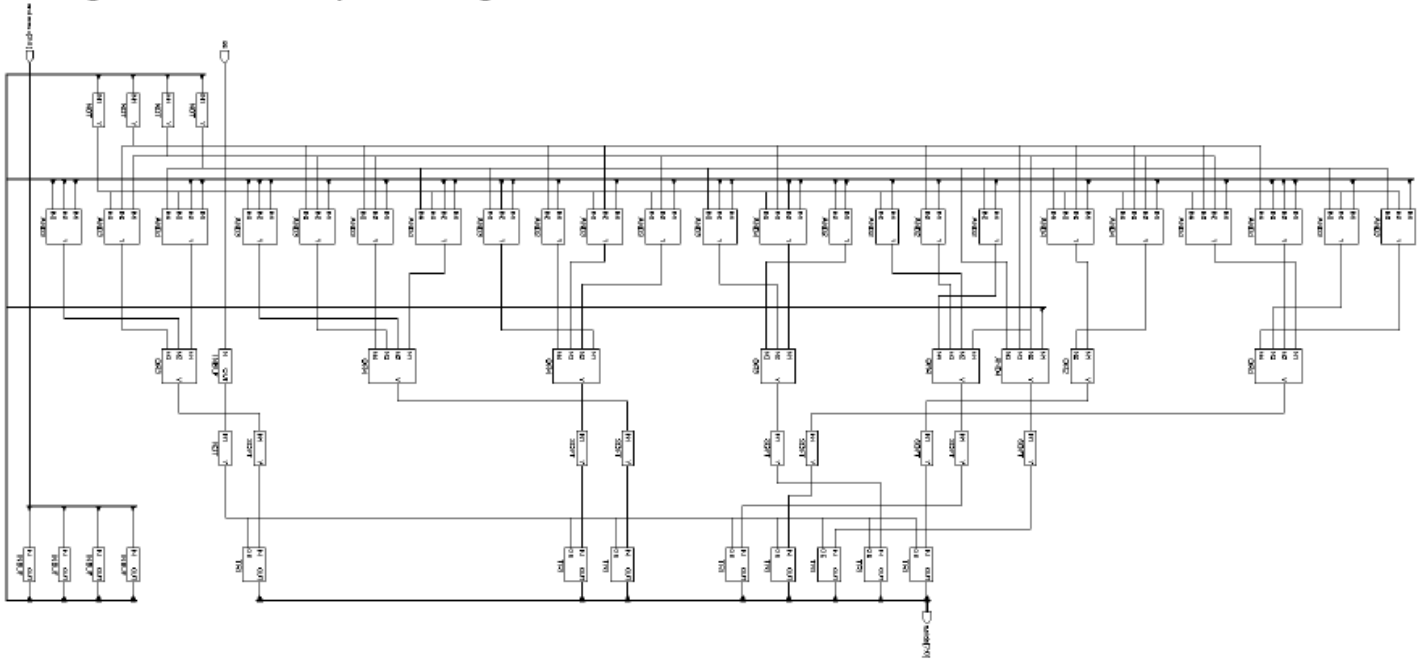
```
"/vhdl/rom_1.vhd", line 18:Info, Inferred rom instance 'ix96' of type 'rom_memenab_8_4_16_4'

Total accumulated area :
 Number of LCs :                          8
 Number of TRIs :                         8
 Number of accumulated instances :       58
```

- **Resultados da síntese da descrição** - tecnologia alvo com unidades de memória:
- **Unidades de memória RAM:** empregadas com 128
  - corresponde a organização da memória descrita no código: 16×8

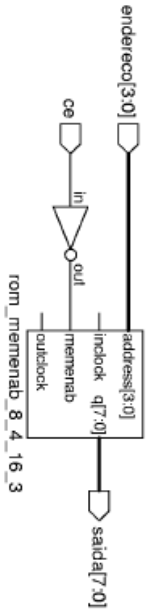
```
"/vhdl/rom_1.vhd", line 18:Info, Inferred rom instance 'ix96' of type 'rom_memenab_8_4_16_3'

Total accumulated area :
 Number of LCs :                          0
 Number of Memory Bits :                128
 Number of accumulated instances :        2
```

Memórias ROM (síntese)

- Diagramas nível porta lógica



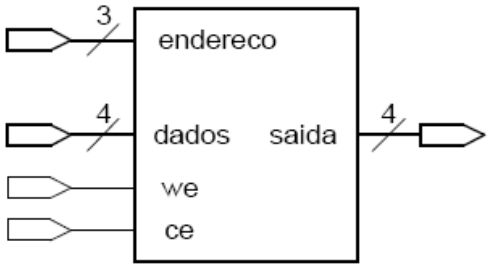
sem unidades de memória

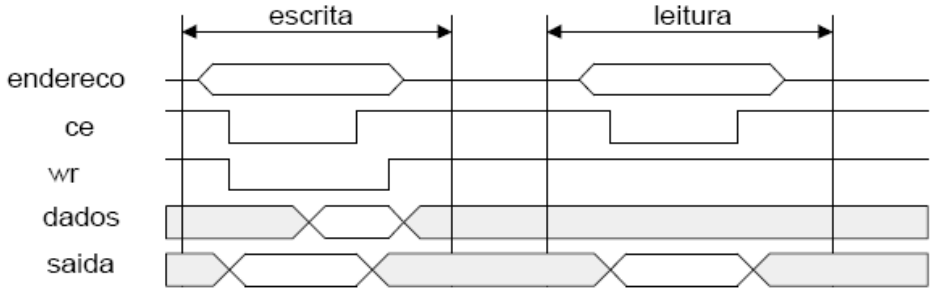


com unidades de memória

Memórias RAM (exemplo)

- **Memória RAM de 8 bit de dados por 15 posições de endereço**
- **Linhas**: endereco definem a posição dos dados
- **Linhas**: saida informação armazenada
- **Linhas**: dado informação a ser armazenada
- **Linha**: we operação leitura / escrita
- **Linha**: ce habilita a memória em nível lógico baixo





Memórias RAM (exemplo - código) (continua prox. imagem)

- **Organização da memória** → declaração do tipo arranjo_memoria (linha 12)
- **Tipo arranjo_memoria define**: 8 bits 32 endereços
- **Sinal memoria** → armazena as informações

```
 5 ENTITY ram_1 IS
 6    port (dado_entrada : IN  UNSIGNED(3 DOWNTO 0);
 7          dado_saida   : OUT UNSIGNED(3 DOWNTO 0);
 8          endereco     : IN  UNSIGNED(2 DOWNTO 0);
 9          we, ce       : IN  STD_LOGIC);
10 END ram_1;
11
12 ARCHITECTURE teste OF ram_1 IS
13   TYPE   arranjo_memoria IS ARRAY(0 TO 7) OF UNSIGNED(3 DOWNTO 0);
14   SIGNAL memoria : arranjo_memoria;
15 BEGIN
16
17
18
19
20
21
22
23
24
```

Memórias RAM (exemplo - código) (continuação da imagem anterior)

- **Seleção um elemento do objeto** = **operação de leitura um endereço da memória**
- **Escrita de um dado**: borda de subida de ce com wr=0
- **Informação de um endereço**: sempre disponível em dado_saida

```
 5 ENTITY ram_1 IS
 6    port (dado_entrada : IN  UNSIGNED(3 DOWNTO 0);
 7          dado_saida   : OUT UNSIGNED(3 DOWNTO 0);
 8          endereco     : IN  UNSIGNED(2 DOWNTO 0);
 9          we, ce       : IN  STD_LOGIC);
10 END ram_1;
11
12 ARCHITECTURE teste OF ram_1 IS
13   TYPE   arranjo_memoria IS ARRAY(0 TO 7) OF UNSIGNED(3 DOWNTO 0);
14   SIGNAL memoria : arranjo_memoria;
15 BEGIN
16   PROCESS(ce, endereco)
17   BEGIN
18     IF rising_edge(ce) THEN   -- dado armazenado na subida de "ce" com "wr=0"
19       IF we = '0' THEN memoria(to_integer(endereco)) <= dado_entrada;
20       END IF;
21    END IF;
22    END PROCESS;
23    dado_saida <= memoria(to_integer(endereco));
24 END teste;
```

Memórias RAM (síntese)

- **Memória do tipo RAM inferida com sucesso**
- **Resultados da síntese da descrição** - tecnologia alvo sem unidades de memória
- **Lógica combinacional e seqüencial:** 32 *flip flops* empregados

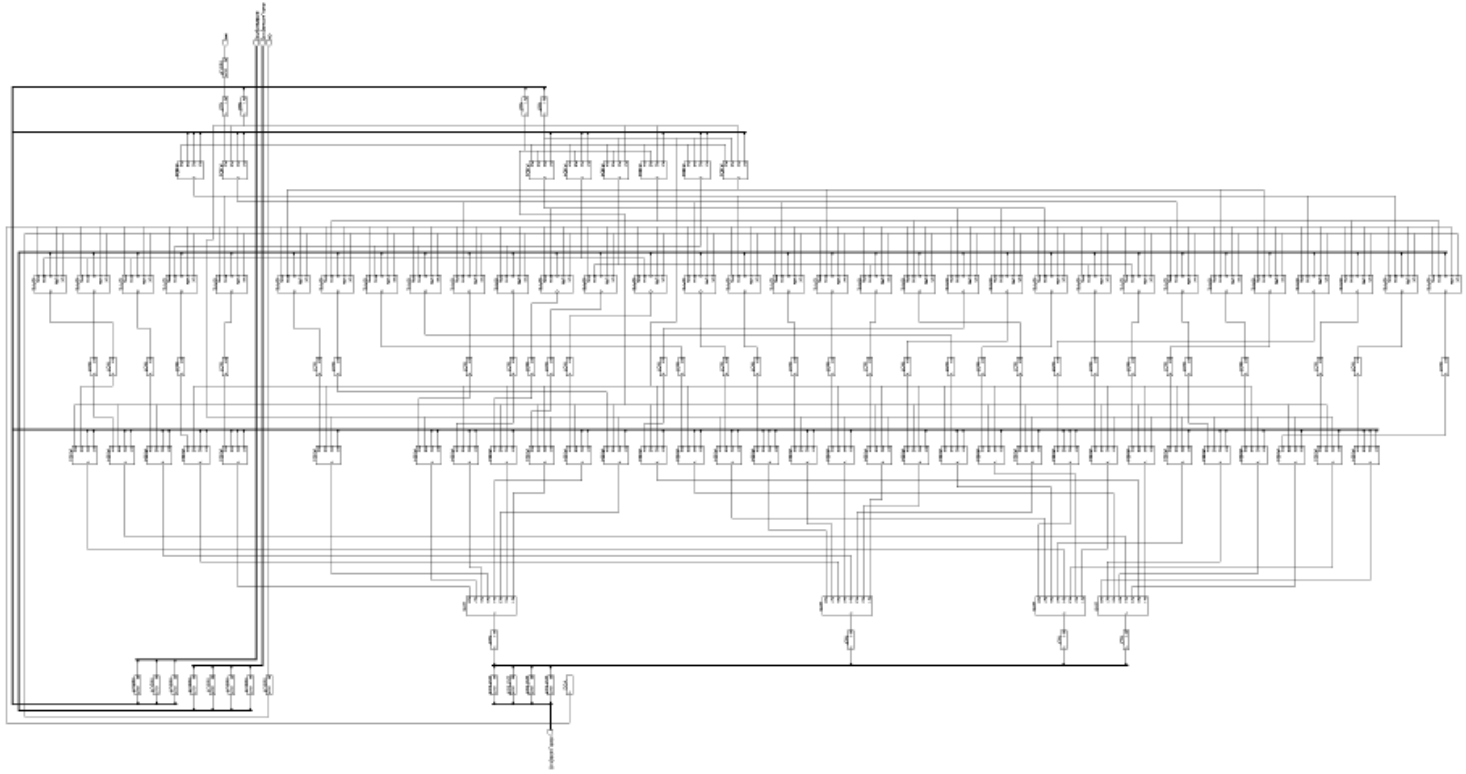
```
"/vhdl/ram_1.vhd",line 14: Info, RAM detected for 'memoria'.
"/vhdl/ram_1.vhd",line 14: Info, created RAM instance 'ix20' for 'memoria'.
Total accumulated area :
 Number of DFFs :                        32
 Number of EXPs :                        16
 Number of LCs :                         32
 Number of accumulated instances :      130
```

- **Resultados da síntese da descrição** - tecnologia alvo com unidades de memória:
- **Unidades de memória RAM empregadas com 32 bits**
  - corresponde a organização da memória descrita no código: 8×4

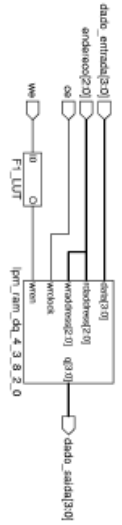
```
"/vhdl/ram_1.vhd",line 14: Info, RAM detected for 'memoria'.
"/vhdl/ram_1.vhd",line 14: Info, created RAM instance 'ix20' for 'memoria'.
Total accumulated area :
 Number of LCs :                          0
 Number of Memory Bits :                 32
 Number of accumulated instances :        2
```

Memórias ROM (síntese)

- **Diagramas nível porta lógica**



sem unidades de memória



com unidades de memória

Pseudônimos Versão VHDL-1987

- **Pseudônimo** - ALIAS
- **Nome alternativo para um objeto das classes**:
  - constante, sinal variável
- **Pseudônimo pode ser**:
  - todo um objeto ou uma parte dele
- **Objetivo**:
  - facilitar o acesso ao objeto
  - melhorar a leitura do código
- **Um pseudônimo**:
  - não é um novo objeto,
  - somente uma designação alternativa para o objeto (parte ou todo)
- **Pseudônimos não são válidos para vetores multidimensionais**

Pseudônimos Versão VHDL-1987 exemplo

- **2 nomes alternativos propostos para o objeto dado**
- **dado**: vetor de 8 bits
- **dado_alto** : novo nome para 4 bits mais significativos de dado
- **dado_baixo** : novo nome para 4 bits restantes.

```
 1 ENTITY alias_a IS
 2   PORT(h, l   : IN  BIT_VECTOR(3 DOWNTO 0);
 3        h_l    : OUT BIT_VECTOR(7 DOWNTO 0));
 4 END alias_a;
 5
 6 ARCHITECTURE teste OF alias_a IS
 7   SIGNAL dado       : BIT_VECTOR(7 DOWNTO 0);
 8   ALIAS  dado_alto  : BIT_VECTOR(3 DOWNTO 0) IS dado(7 DOWNTO 4);
 9   ALIAS  dado_baixo : BIT_VECTOR(3 DOWNTO 0) IS dado(3 DOWNTO 0);
10 BEGIN
11   dado_alto  <= NOT h;              -- equivalente a: dado(7 DOWNTO 4) <= NOT h;
12   dado_baixo <= NOT l;              -- equivalente a: dado(3 DOWNTO 0) <= NOT l;
13   h_l <= dado_alto & dado_baixo;
14 END teste;
```

Pseudônimos Versão VHDL-1993

- **O escopo de um pseudônimo é ampliado**:
  - um nome alternativo para um item nomeado
- **Item objeto**, (*object aliasis*):
  - corresponde ao conceito da versão VHDL-87:
  - objetos das classes: constante variável sinal arquivo
- **Item não objeto**, (*nonobject aliasis*):
  - nome de subprogramas,
  - nome de tipos,
  - nome de atributos,
  - pacotes, etc.
- **Pseudônimos não são permitido nos casos:**
  - rótulos
  - parâmetros de declarações LOOP GENERATE

Pseudônimos Versão VHDL-1993 exemplo em itens não objetos

- **Designações alternativas para dois subprogramas**: logica_funcao logica_proced
- **Pseudônimos de subprogramas**:
  - necessário identificar os tipos na ordem como eles foram definidos entre [ ]
- **A palavra reservada RETURN**: necessária somente para funções

```
 7 ARCHITECTURE teste OF alias_b IS
 8 FUNCTION logica_funcao (a, b : BIT_VECTOR) RETURN BIT_VECTOR IS
 9   BEGIN
10     RETURN a AND NOT b;
11 END logica_funcao;
12
13 PROCEDURE logica_proced(a, b : IN BIT_VECTOR; SIGNAL d, e : OUT BIT_VECTOR) IS
14   BEGIN
15     d <= a  OR NOT b;
16     e <= a XOR NOT b;
17  END logica_proced;
18
19 -- identificador    item       [          assinatura                        ];
20 ALIAS lg_func IS logica_funcao [BIT_VECTOR, BIT_VECTOR  RETURN BIT_VECTOR];
21 ALIAS lg_proc IS logica_proced [BIT_VECTOR, BIT_VECTOR, BIT_VECTOR, BIT_VECTOR];
22
23 BEGIN
24  s <= lg_func(x, y);
25  lg_proc(x, y, t, u);
26 END teste;
```