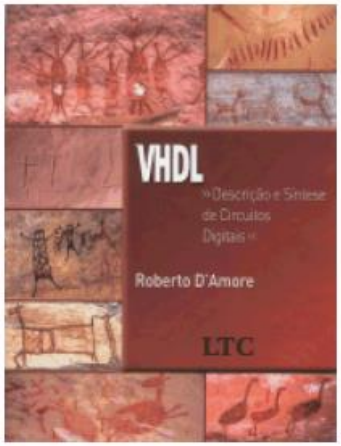
SEL5752/SEL0632 – Linguagens de Descrição de Hardware
Aula 6 – Componentes e Esquemas

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**

Componentes e esquemas de iteração

Tópicos

- **Componentes**
- **Definição de genéricos**
- **Comando GENERATE**
  - esquema de geração FOR
  - esquema de geração IF
- **Comando LOOP**
  - esquema de interação FOR
  - esquema de iteração WHILE
  - comandos: NEXT EXIT
  - laços infinitos

Componentes

- **Componente:**
  - uma descrição (entidade + arquitetura) empregada por uma outra entidade
- **Emprego:**
  - interligação de múltiplas entidades de projeto
    - projeto hierárquico
- **Declaração de um componente** (primeiro passo para utilização de um componente)
  - similar a declaração de entidade
  - exemplo:

```
COMPONENT nome_componente
  PORT  (sinal_a      : modo_a  tipo_sinal_a;
         sinal_b      : modo_b  tipo_sinal_b;
         sinal_c      : modo_c  tipo_sinal_c);
END COMPONENT;
```

Componentes

- **A solicitação de um componente contém:**
  - rótulo de denominação qualquer (x1, y2 ...)
  - nome do componente (nome_componente)
  - mapa de ligações (Port Map (....))

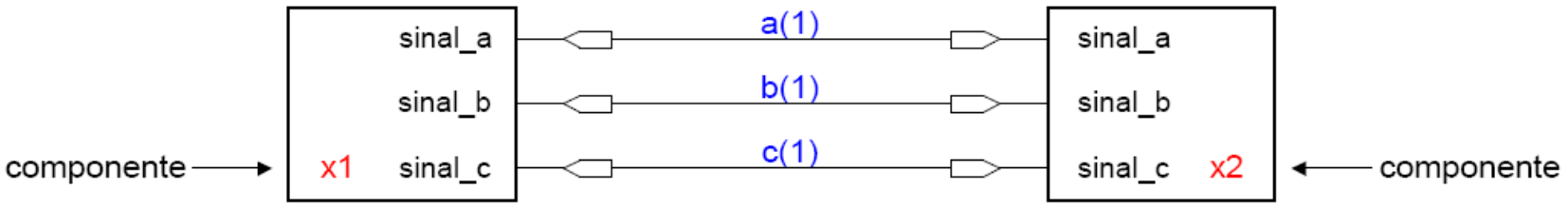
- **Mapa de ligações**:

  - pode seguir mesma ordem estabelecida na declaração do componente:

```
x1: nome_componente PORT MAP(a(1), b(1), c(1));
```

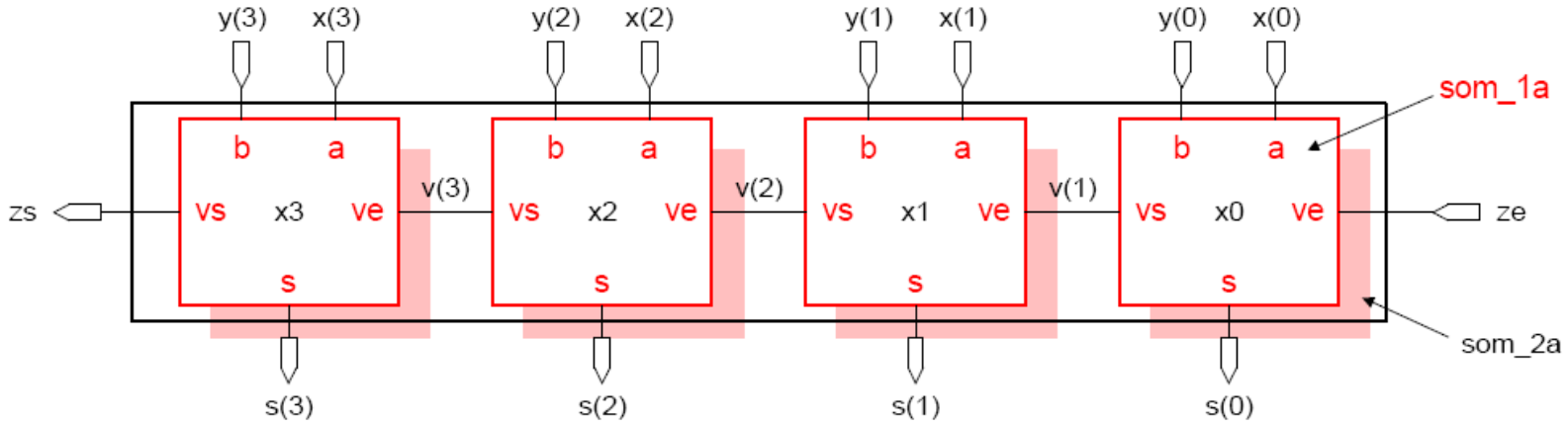
  - pode seguir uma nova seqüência definida no mapa:

```
x2: nome_componente PORT MAP(sinal_b => b(1),sinal_a => a(1),sinal_c => c(1));
```



Componentes

- **Exemplo**: Somador de 4 bits formado por somadores de 1 bit

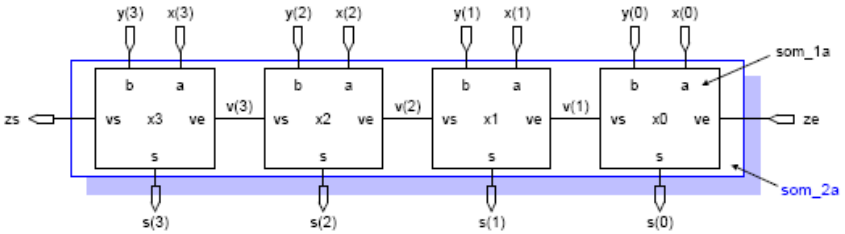


- **Descrição do somador de 1 bit** (componente a ser referenciado)

```
 1 ENTITY som_1a IS
 2   PORT    (a, b, ve  : IN  BIT;
 3            s, vs     : OUT BIT);
 4 END som_1a;
 5
 6 ARCHITECTURE teste OF som_1a IS
 7
 8 BEGIN
 9   s  <=  a XOR b  XOR ve;                          -- soma
10   vs <= (a AND b) OR (a AND ve) OR (b AND ve); -- vai um
11 END teste;
```

Componentes

- **Descrição do somador empregando o componente**: somador de 1 bit

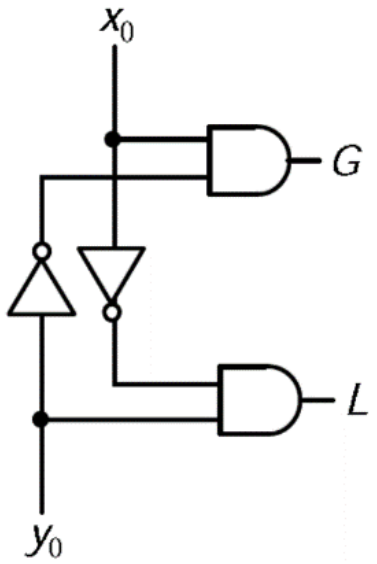



```
 1 ENTITY som_2a IS
 2   PORT   ( x, y  : IN  BIT_VECTOR (3 DOWNTO 0);  -- entradas do somador
 3            ze    : IN  BIT;                      -- entrada  vem um
 4            s     : OUT BIT_VECTOR (3 DOWNTO 0);  -- soma
 5            zs    : OUT BIT );                    -- vai um
 6 END som_2a;
 7
 8 ARCHITECTURE estrutural OF som_2a IS
 9
10 COMPONENT som_1a
11   PORT    (a, b, ve : IN  BIT;  s, vs : OUT BIT);
12 END COMPONENT;
13
14 SIGNAL v    : BIT_VECTOR (3 DOWNTO 1);  -- vai um interno
15
16 BEGIN
17   x0: som_1a PORT MAP(    x(0),      y(0),     ze,        s(0),      v(1));
18   x1: som_1a PORT MAP(    x(1),      y(1),     v(1),      s(1),      v(2));
19   x2: som_1a PORT MAP(b =>y(2), a => x(2), s =>s(2), ve =>v(2), vs =>v(3));
20   x3: som_1a PORT MAP(    x(3),      y(3),     v(3),      s(3),      zs);
21 END estrutural;
```

Comparador de magnitude

Para dois valores de um bit (x_0 e y_0), montamos a seguinte tabela verdade:
De onde obtemos G = $x_0\overline{y_0}$ e L = $\overline{x_0}y_0$ e o circuito:

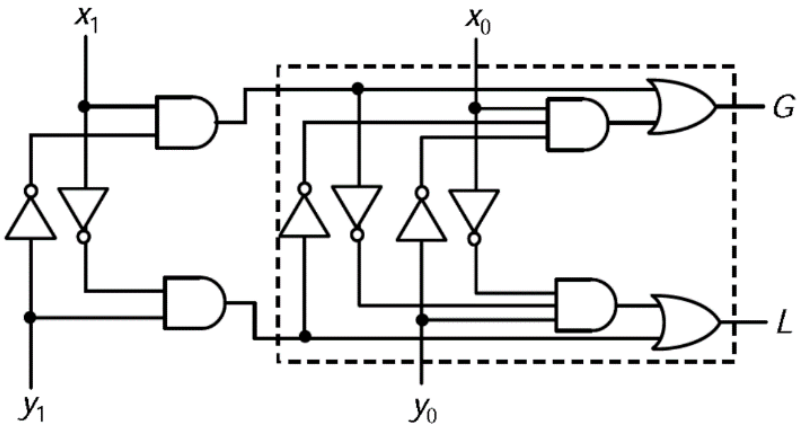
x_0	y_0	G ($x_0 > y_0$)	L ($x_0 < y_0$)
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0



Comparador de magnitude

Para dois valores de dois bits ($x=x_1x_0$ e $y=y_1y_0$), temos a seguinte tabela verdade:
De onde obtemos G = $x_1\overline{y_1} + \overline{(\overline{x_1}y_1)}(x_0\overline{y_0})$ e L = $\overline{x_1}y_1 + \overline{(x_1\overline{y_1})}(\overline{x_0}y_0)$ e o circuito

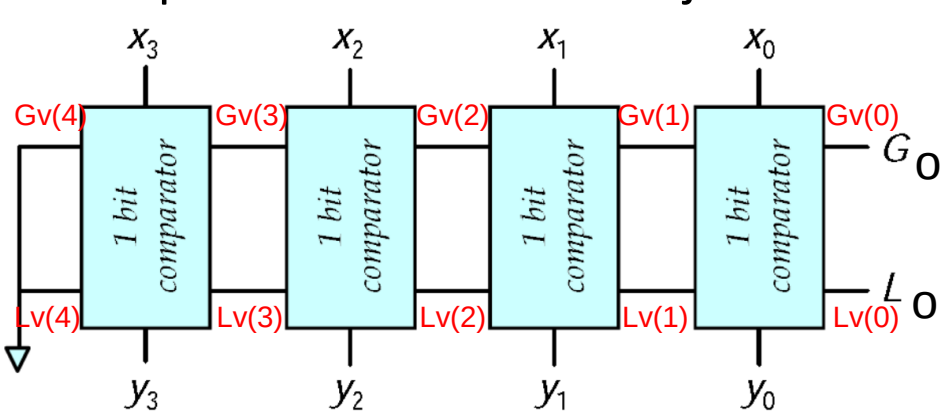
x_1	x_0	y_1	y_0	G (x > y)	L (x < y)
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

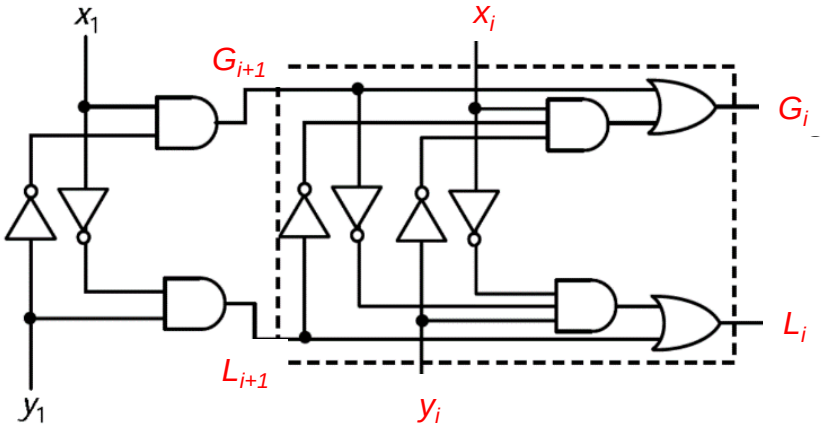


Para dois valores de 3 bits, a coisa começa a ficar complicada...

Comparador de magnitude

É possível implementar uma estrutura em “escada”, onde cada módulo corresponde ao circuito tracejado.





Implementar em VHDL, utilizando operadores lógicos, o circuito tracejado, utilizando o *template* indicado no site da disciplina.

Apresente a descrição do comparador em escada de 4 bits utilizando como componente a entidade comp1bit abaixo:

```
ENTITY comp1bit IS          -- Comparador de 1 bit para implem. em escada
   PORT
   (
      xi, yi    : IN bit;   -- bits dos valores a serem comparados
      Gi, Li    : IN bit;   -- resultado das comparacoes anteriores
      Go, Lo    : OUT bit   -- saida da comparacao entre xi e yi
   );
END ENTITY;

ARCHITECTURE logical OF comp1bit IS
BEGIN

   Go <= Gi or (not Li and     xi and not yi);
   Lo <= Li or (not Gi and not xi and     yi);

END logical;
```

Apresente a descrição do comparador em escada de 4 bits utilizando como componente a entidade comp1bit abaixo:

```
ENTITY compNbits IS          -- Comparador de 1 bit para implem. em escada
   PORT
   (
      x, y    : IN bit_vector(3 downto 0);  -- valores a serem comparados
      G, L    : OUT bit  -- saida da comparacao entre x e y
   );
END ENTITY;

ARCHITECTURE ladder OF compNbits IS
BEGIN
   signal Gv, Lv: bit_vector(3 downto 1);

   u3: comp1bit port map (x(3), y(3),  '0' ,  '0' , Gv(3), Lv(3));
   u2: comp1bit port map (x(2), y(2), Gv(3), Lv(3), Gv(2), Lv(2));
   u1: comp1bit port map (x(1), y(1), Gv(2), Lv(2), Gv(1), Lv(1));
   u0: comp1bit port map (x(0), y(0), Gv(1), Lv(1),   G  ,   L  );

END ladder;
```

Definição de genéricos

- **Genéricos**: fornecem um meio de levar informações externas estáticas para entidades de projeto e blocos

- **Informações**:
  - características de desempenho
  - parâmetros que definem a lei de formação na interligação de componentes.

- **Exemplos:**
  - declaração com definição do valor
  - declaração sem definição do valor

```
ENTITY nome_entidade IS
 GENERIC(generico_1 :tipo_do_generico_1 := valor_inicial; -- declaracao com valor
         generico_2 :tipo_do_generico_2);                 -- apenas declaracao
 PORT    (......);
END nome_entidade;
```

Definição de genéricos - exemplo

- **Registrador**: número de bits definido pelo genérico n
- **Após a declaração**, o genérico pode ser empregado
  - vide linhas 5 e 6

```
 1 ENTITY flipn_3 IS
 2  GENERIC(n   : INTEGER := 3); -- declaracao e definicao do valor do generico
 3  PORT   (ck  : IN  BIT;                          -- relogio
 4          rst : IN  BIT;                          -- rst=1 leva q=000 assincrono
 5          d   : IN  BIT_VECTOR(n-1 DOWNTO 0);     -- definido pelo generico
 6          q   : OUT BIT_VECTOR(n-1 DOWNTO 0)); -- definido pelo generico
 7 END flipn_3;
 8
 9 ARCHITECTURE teste OF flipn_3 IS
10 BEGIN
11   PROCESS (ck, rst)
12   BEGIN
13    IF       (rst = '1')          THEN q <= (OTHERS => '0'); -- q=00...0
14    ELSIF (ck'EVENT AND ck ='1') THEN q <= d;               -- armazena dado
15    END IF;
16   END PROCESS;
17 END teste;
```

Comando GENERATE

- esquema de geração FOR
- esquema de geração IF

Comando GENERATE

- **Cópia de comandos concorrentes:**
  - esquema de geração FOR
  - esquema de geração IF

- **Aplicação: geração automática**
  - circuitos regulares
  - circuitos que seguem uma lei de formação
  - exemplo:
    - unidades lógicas aritméticas
    - somadores
    - multiplicadores

Comando GENERATE - esquema de geração FOR

- **FOR identificador IN limites_da_geracao GENERATE**
  - Identificador: não necessita ser declarado
  - Limites da geração:
    - faixa discreta de valores: (0 TO 7) (23 DOWNTO 0)
    - atributos que retornem faixa de valores: a'RANGE b'REVERSE_RANGE

```
abc: FOR indentificador IN valor_inicial TO valor_final GENERATE
   -- comando concorrente
END GENERATE abc;
```

- **Exemplos**:

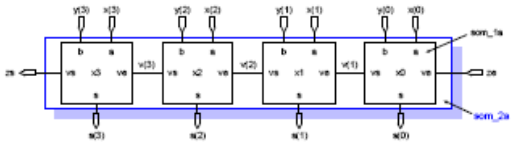
```
def: FOR i IN sinal'RANGE GENERATE
    -- comando concorrente
END GENERATE def;

ghi: FOR x IN valor_final DOWNTO valor_inicial GENERATE
    -- comando concorrente
END GENERATE;                    -- ultimo rotulo opcional

xyz: FOR k IN sinal'REVERSE_RANGE GENERATE
    -- comando concorrente
END GENERATE;                   -- ultimo rotulo opcional
```

Comando GENERATE

- **Comandos podem ser aninhados:**

```
linha: FOR i IN 0 TO 3 GENERATE
  coluna: FOR j IN 0 TO 3 GENERATE
      x: componente_abc PORT MAP (entrada_a(i),entrada_b(j),saida(i+1)) ;
  END GENERATE coluna;
END GENERATE linha;
```

Comando GENERATE - esquema de geração FOR

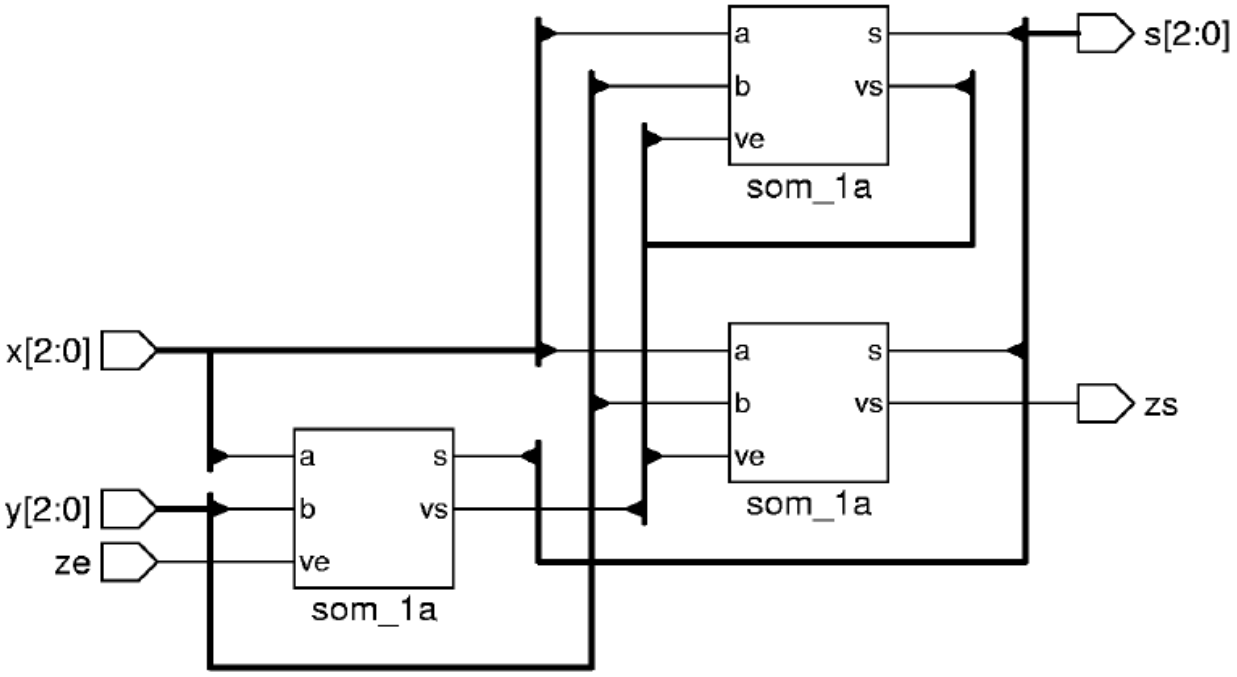


- **Exemplo de um somador empregando componente**

```
 1 ENTITY som_6aa IS
 2   GENERIC(n      : INTEGER  := 3 );                  -- numero de bits
 3   PORT    (x, y  : IN  BIT_VECTOR (n-1 DOWNTO 0);   -- entradas do somador
 4            ze    : IN  BIT;                         -- vem um
 5            s     : OUT BIT_VECTOR (n-1 DOWNTO 0);   -- saida
 6            zs    : OUT BIT);                        -- vai um
 7 END som_6aa;
 8
 9 ARCHITECTURE estrutural OF som_6aa IS
10   COMPONENT som_1a
11     PORT   (a, b, ve : IN  BIT;  s, vs  : OUT BIT);
12   END COMPONENT;
13   SIGNAL v : BIT_VECTOR (n DOWNTO 0);  -- vai um interno
14 BEGIN
15   v(0) <= ze;
16   zs   <= v(n);
17   abc: FOR i IN 0 TO n-1 GENERATE
18     centro: som_1a PORT MAP (x(i),  y(i),   v(i),    s(i),   v(i+1));
19   END GENERATE abc;
20 END estrutural;
```

Comando GENERATE - esquema de geração FOR

- **Exemplo de um somador empregando componente**
  - circuito sintetizado:



Comando GENERATE -- esquema de geração FOR

- **Exemplo de um somador empregando comandos concorrentes**

```
 1 ENTITY som_6c IS
 2   GENERIC(n      : INTEGER  := 3 );                  -- numero de bits
 3   PORT    (x, y  : IN  BIT_VECTOR (n-1 DOWNTO 0);   -- entradas do somador
 4            ze    : IN  BIT;                         -- vem um
 5            s     : OUT BIT_VECTOR (n-1 DOWNTO 0);   -- saida
 6            zs    : OUT BIT);                        -- vai um
 7 END som_6c;
 8
 9 ARCHITECTURE teste OF som_6c IS
10   SIGNAL v : BIT_VECTOR (n DOWNTO 0);  -- vai um interno
11 BEGIN
12   v(0) <= ze;
13   zs   <= v(n);
14   abc: FOR i IN 0 TO n-1 GENERATE
15     s(i)   <=  x(i) XOR y(i)  XOR v(i);
16     v(i+1) <= (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
17   END GENERATE abc;
18 END teste;
```

Comando GENERATE - esquema de geração IF

- **Cópia das declarações:**
  - controlada pelo comando → IF
    - condição verdadeira: cópia efetuadas
- **Exemplo:**

```
abc: IF condicao  GENERATE
  -- comando concorrente
  -- comando concorrente
END GENERATE abc;
```

Comando GENERATE - esquema de geração IF

- **Exemplo de um somador empregando componentes**

```
 1 ENTITY som_7a IS
 2   GENERIC(n     : INTEGER  := 3 );                 -- numero de bits
 3   PORT    (x, y  : IN  BIT_VECTOR (n-1 DOWNTO 0);  -- entradas so somador
 4            ze    : IN  BIT;                        -- vem um
 5            s     : OUT BIT_VECTOR (n-1 DOWNTO 0);  -- saida
 6            zs    : OUT BIT);                       -- vai um
 7 END som_7a;
 8
 9 ARCHITECTURE estrutural OF som_7a IS
10   COMPONENT som_1a
11     PORT    (a, b, ve : IN  BIT;  s, vs  : OUT BIT);
12   END COMPONENT;
13   SIGNAL v : BIT_VECTOR (n-1 DOWNTO 1);  -- vai um interno
14 BEGIN
15  global: FOR i IN 0 TO n-1 GENERATE
16    def: IF i =0 GENERATE
17      primeira: som_1a PORT MAP (x(i), y(i), ze,  s(i), v(i +1)); -- 1a celula
18    END GENERATE def;
19    abc: IF (i > 0) AND (i < n-1) GENERATE
20      centro: som_1a PORT MAP  (x(i), y(i), v(i), s(i), v(i +1)); -- celulas centro
21    END GENERATE abc;
22    ghi: IF i =n-1 GENERATE
23      ultima: som_1a PORT MAP  (x(i), y(i), v(i), s(i), zs);      -- ultima celula
24    END GENERATE ghi;
25  END GENERATE global;
26 END estrutural;
```

Apresente a descrição do comparador em escada de N bits utilizando GENERIC, GENERATE, tendo a entidade abaixo como componente:

```
ENTITY comp1bit IS          -- Comparador de 1 bit para implem. em escada
   PORT
   (
      xi, yi    : IN bit;   -- bits dos valores a serem comparados
      Gi, Li    : IN bit;   -- resultado das comparacoes anteriores
      Go, Lo    : OUT bit   -- saida da comparacao entre xi e yi
   );
END ENTITY;

ARCHITECTURE logical OF comp1bit IS
BEGIN

   Go <= Gi or (not Li and     xi and not yi);
   Lo <= Li or (not Gi and not xi and     yi);

END logical;
```

Comando LOOP

- esquema de interação FOR
- esquema de iteração WHILE
- comandos: NEXT EXIT
- laços infinitos

Comando LOOP

- **Repetição da execução de comandos seqüenciais:**
  - esquemas de iteração: FOR e WHILE

- **Aplicação**: geração automática
  - circuitos regulares
  - rotinas de conversão (não necessariamente aplicadas em síntese)

Comando LOOP modo iterativo FOR

- **FOR identificador IN limites_da_iteracao LOOP**
  - Identificador: não necessita ser declarado
  - Limites da iteração:
    - faixa discreta de valores: (0 TO 7) (23 DOWNTO 0)
    - atributos que retornem faixa de valores: a'RANGE b'REVERSE_RANGE
- **Exemplos:**

```
abc: FOR i IN valor_inicial TO valor_final LOOP     -- faixa crescente
   -- comando sequencial
END LOOP abc;

def: FOR i IN valor_final DOWNTO valor_inicial LOOP -- faixa decrescente
   -- comando sequencial
END LOOP def;

ghi: FOR i IN x'RANGE LOOP  -- faixa: elementos na faixa definida, exemplo
   -- comando sequencial    --    (final DOWNTO inicial), (inicial TO final)
END LOOP ghi;
```

Comando LOOP - modo iterativo FOR

- **Exemplo de um somador - comandos seqüenciais**

```
 1 ENTITY som_8a IS
 2   GENERIC(n     : INTEGER  := 3 );                -- numero de bits
 3   PORT    (x, y : IN  BIT_VECTOR (n-1 DOWNTO 0);  -- entradas do somador
 4            ze   : IN  BIT;                        -- vem um
 5            s    : OUT BIT_VECTOR (n-1 DOWNTO 0);  -- saida
 6            zs   : OUT BIT);                       -- vai um
 7 END som_8a;
 8
 9 ARCHITECTURE teste OF som_8a IS
10 BEGIN
11   abc: PROCESS (x, y, ze)
12     VARIABLE v : BIT_VECTOR (n DOWNTO 0);  -- vai um interno
13   BEGIN
14     v(0) := ze;
15     abc: FOR i IN 0 TO n-1 LOOP
16       s(i)   <=  x(i) XOR y(i)  XOR v(i);
17       v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
18     END LOOP abc;
19     zs   <= v(n);
20   END PROCESS;
21 END teste;
```

Comando LOOP - modo iterativo FOR

- **Exemplo de limites de iteração definidos por**:

  - atributos: RANGE e REVERSE_RANGE

```
 1 ENTITY rev_bit0 IS
 2   GENERIC(n      : INTEGER  := 4 );
 3   PORT    (x     : IN  BIT_VECTOR (n-1 DOWNTO 0);
 4            s, t  : OUT BIT_VECTOR (n-1 DOWNTO 0));
 5 END rev_bit0;
 6
 7 ARCHITECTURE teste OF rev_bit0 IS
 8 BEGIN
 9   abc: PROCESS (x)
10   BEGIN
11     abc: FOR i IN n-1 DOWNTO 0 LOOP
12       t(t'LEFT -i) <= x(i);
13     END LOOP abc;
14     def: FOR i IN    x'RANGE   LOOP -- faixa: x'RANGE =(n-1 DOWNTO 0)
15       s(s'LEFT -i) <= x(i);
16     END LOOP def;
17   END PROCESS;
18 END teste;
```

Comando LOOP - modo iterativo FOR

- **Exemplo - rotina de conversão:**

  - entrada: tipo bit_vector retorna: tipo inteiro

```
 1 ENTITY loop_f1 IS
 2   PORT (e_bit         : IN  BIT_VECTOR (3 DOWNTO 0); -- entrada vetor de bits
 3         inteiro_for   : OUT INTEGER RANGE 0 TO 15);  -- saida  inteiro
 4 END loop_f1;
 5
 6 ARCHITECTURE teste OF loop_f1 IS
 7 BEGIN
 8   abc_for: PROCESS (e_bit)
 9     VARIABLE temp : INTEGER;
10   BEGIN
11     temp := 0;
12     FOR i IN e_bit'RANGE LOOP     -- iteracoes: tamanho do vetor entrada
13       IF e_bit(i) = '1' THEN temp := temp + 2**i;
14       END IF;
15     END LOOP;
16     inteiro_for <= temp;
17   END PROCESS;
18 END teste;
```

Comando LOOP - modo iterativo FOR

- **Exemplo anterior - rotina de conversão:**

  - tipo bit_vector para inteiro

- **Circuito sintetizado corretamente:**

  - apesar do distanciamento circuito $\leftrightarrow$ descrição
  - 2^i aceito por determinadas ferramentas de síntese

Comando LOOP modo iterativo WHILE

- **WHILE condicao_teste LOOP**
  - Iteração executada enquanto:
    - condição de teste verdadeira
- **Exemplo**:

```
-- nota: teste executado no inicio do LOOP

abc: WHILE condicao_teste LOOP  -- enquanto condicao_teste = verdadeira
  -- comando sequencial
  -- comando sequencial
  -- atualizacao variavel de indice
END LOOP abc;
```

Comando LOOP - modo iterativo WHILE

- **Exemplo de um somador - declarações seqüenciais**

```
 1 ENTITY som_9a IS
 2   GENERIC(n     : INTEGER  := 3 );                -- numero de bits
 3   PORT    (x, y : IN  BIT_VECTOR (n-1 DOWNTO 0);  -- entradas so somador
 4            ze   : IN  BIT;                        -- vem um
 5            s    : OUT BIT_VECTOR (n-1 DOWNTO 0);  -- saida
 6            zs   : OUT BIT);                       -- vai um
 7 END som_9a;
 8
 9 ARCHITECTURE teste OF som_9a IS
10 BEGIN
11   abc: PROCESS (x, y, ze)
12     VARIABLE i : INTEGER := 0;
13     VARIABLE v : BIT_VECTOR (n DOWNTO 0);  -- vai um interno
14   BEGIN
15     i := 0;        -- deve ser atualizado a cada iteracao
16     v(0) := ze;
17     abc: WHILE i <= n-1 LOOP     -- executado enquanto verdadeiro
18       s(i)   <=  x(i) XOR y(i)  XOR v(i);
19       v(i+1) := (x(i) AND y(i)) OR (x(i) AND v(i)) OR (y(i) AND v(i));
20       i := i+1;
21     END LOOP abc;
22     zs   <= v(n);
23   END PROCESS;
24 END teste;
```

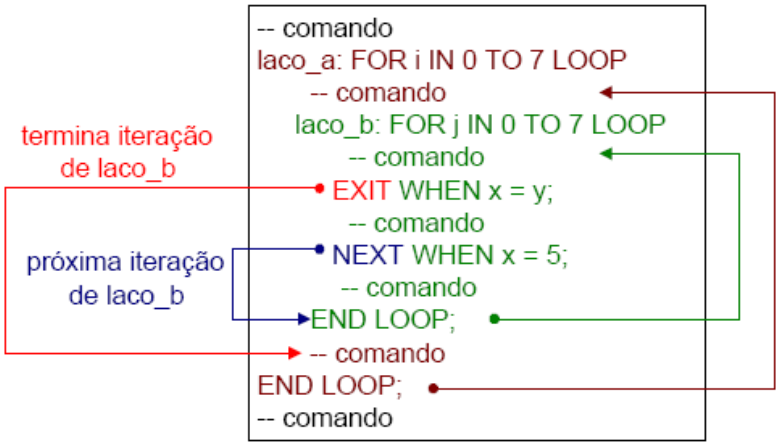
Comando LOOP - modo iterativo WHILE

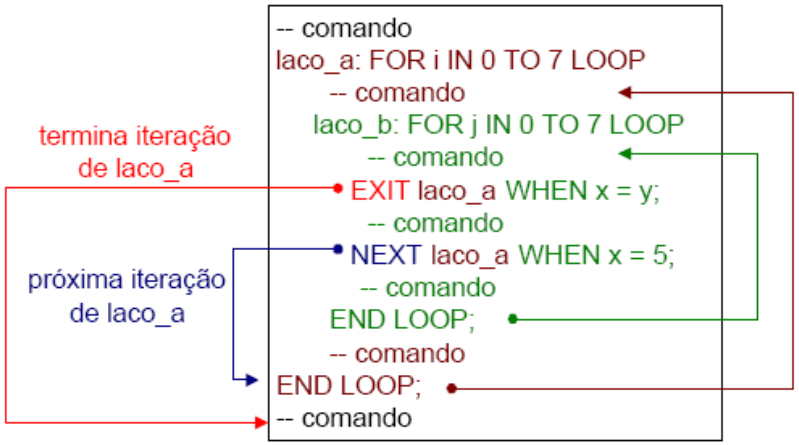
- **Exemplo rotina de conversão:**
  - tipo bit_vector para inteiro

```
 1 ENTITY loop_w1 IS
 2   GENERIC (n           : INTEGER := 4);
 3   PORT (e_bit          : IN  BIT_VECTOR (n-1 DOWNTO 0); -- entrada bit_vector
 4         inteiro_while  : OUT INTEGER RANGE 0 TO 2**n-1);-- saida integer
 5 END loop_w1;
 6
 7 ARCHITECTURE teste OF loop_w1 IS
 8
 9 BEGIN
10   abc_while: PROCESS (e_bit)
11     VARIABLE temp, i : INTEGER;
12   BEGIN
13     temp := 0;
14     i := 0;
15     WHILE (i /= e_bit'LENGTH) LOOP     -- iteracoes: tamanho vetor entrada
16       IF e_bit(i) = '1' THEN temp := temp + 2** i;
17       END IF;
18       i := i +1;
19     END LOOP;
20     inteiro_while <= temp;
21   END PROCESS;
22 END teste;
```

Comandos NEXT e EXIT

- **NEXT**: salto para o final do laço,
  execução dos comandos restantes no esquema de iteração omitidas
- **EXIT**: leva ao término das operações do laço em execução
- **Podem ser subordinadas a uma condição**: palavra reservada WHEN
- **Rótulo**: pode indicar o laço para retomada das operações
- **Permitidos unicamente no interior de laços**
- **Exemplos**:





Laços infinitos

- **Esquema de geração:**
  - sem controle: executado sempre
    - equivalente a WHILE TRUE LOOP
- **Ambos os casos:**
  - necessário prever esquema para abortar a execução
    - Comando EXIT por exemplo
- **Exemplo:**

```
ghi: LOOP  -- executado sempre
  -- comando sequencial
  -- comando sequencial
END LOOP ghi;


def: WHILE TRUE LOOP  -- executado sempre
  -- comando sequencial
  -- comando sequencial
END LOOP def;
```

Laço infinito exemplo - término da execução: EXIT

- **Exemplo rotina de conversão:**
  - entrada: tipo bit_vector retorna: tipo inteiro

```
 1 ENTITY loop_e1 IS
 2   GENERIC (n          : INTEGER := 4);
 3   PORT (e_bit         : IN  BIT_VECTOR (n-1 DOWNTO 0); -- entrada bit_vector
 4         inteiro_exit : OUT INTEGER RANGE 0 TO 2**n-1);-- saida integer
 5 END loop_e1;
 6
 7 ARCHITECTURE teste OF loop_e1 IS
 8
 9 BEGIN
10   abc_for: PROCESS (e_bit)
11     VARIABLE temp, i : INTEGER;
12   BEGIN
13     temp := 0;
14     i := 0;
15     LOOP
16       IF e_bit(i) = '1' THEN temp := temp + 2**i;
17       END IF;
18       i := i +1;
19       EXIT WHEN i = e_bit'LENGTH;
20     END LOOP;
21     inteiro_exit <= temp;
22   END PROCESS;
23 END teste;
```

Leitura adicional:

- **Solicitação direta de componentes** - versão VHDL-1993
- **Mapa de genéricos na solicitação de componentes**