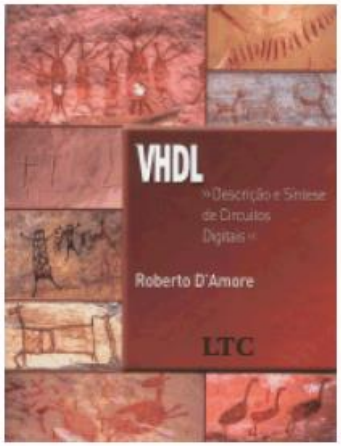
SEL5752 – Disp. Reconfiguráveis e Ling. de Descrição de Hardware
Aula 08 – Subprogramas

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**

Subprogramas

Tópicos

- **Subprogramas:** parâmetros formais e parâmetros reais
- **Função**
- **Procedimento**
- **Sobrecarregamento de subprogramas e operadores**
- **Argumentos sem especificação de limites**
- **Cuidados na descrição**

Subprogramas

- **Objetivo**:
  - isolar declarações freqüentemente utilizadas, para facilitar o entendimento da descrição

- **Tipos**:
  - função: retorna com um único argumento
    invocada por uma expressão (exemplo: `s <= f_soma(a, b);`)
  - procedimento: pode retornar vários argumentos
    invocados por um comando (exemplo: `p_soma(a, b, s);`)

- **Código de um subprograma**:
  - declarações seqüenciais

- **Síntese de subprogramas**: cada chamada de um subprograma:
  - necessário uma unidade funcional
  - não resulta em economia de recursos

Subprogramas

- **Podem ser invocados**:
  - regiões de código concorrente
  - regiões de código seqüencial

- **Permitida a chamada recursiva**
  - normalmente não é aceita pelas ferramentas de síntese.

- **Objetos declarados localmente**:
  - descartados ao final da execução ← importante ! (exemplos serão mostrados)

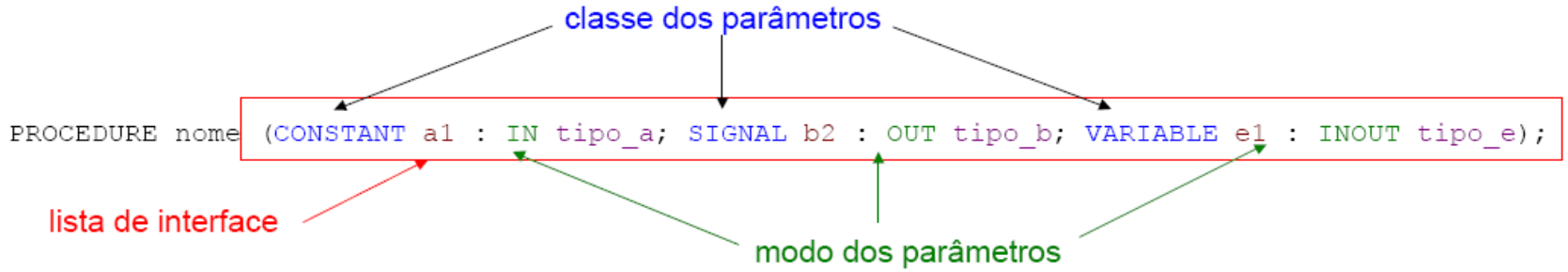
- **Divididos em duas partes**:
  - primeira define o nome e os parâmetros de entrada e saída,
  - a segunda estabelece as operações executadas.

Subprogramas

- **Troca de informação**:
  - feita através de parâmetros da classes:
    - Constant Signal Variable

- **Parâmetros**:
  - declarados numa lista de interface

- **Lista de interface identifica**:
  - classe nome modo (sentido que informação é transmitida) e tipo dos objetos



Subprogramas

- **Parâmetro real**:

  - parâmetro passado ao subprograma ou retornado do subprograma

- **Parâmetro formal**:

  - parâmetro declarado no subprograma

- **Associação de classes em subprogramas:**

Parâmetro real		Parâmetro formal	Observações
constante sinal variável	↔	constante	modo IN: valores tratados como constantes
sinal	↔	sinal	modo IN: atributos podem ser referenciados
variável	↔	variável	modo IN: valores tratados como constantes

Subprogramas

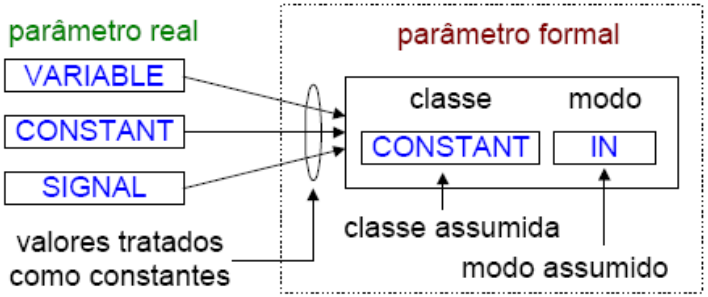
- **Parâmetro formal** **constante** **ou** **variável**:
  - apenas o valor é transferido para o subprograma
  - informação tratada como constante

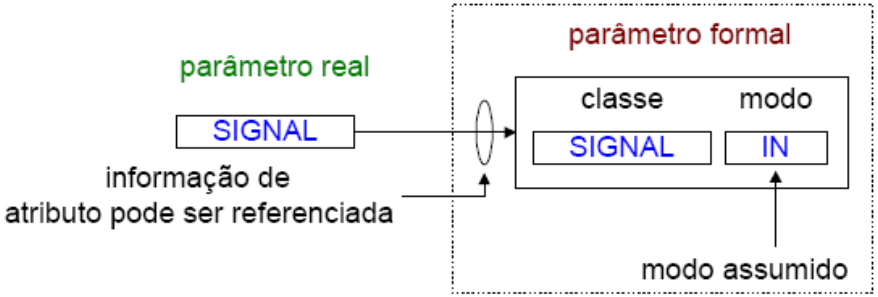
- **Parâmetro formal** **sinal**:
  - informações de atributo podem ser referenciadas
  - exemplo:

    atributo EVENT pode ser empregado em um subprograma

Subprogramas

- **Passagem de dados para funções:**



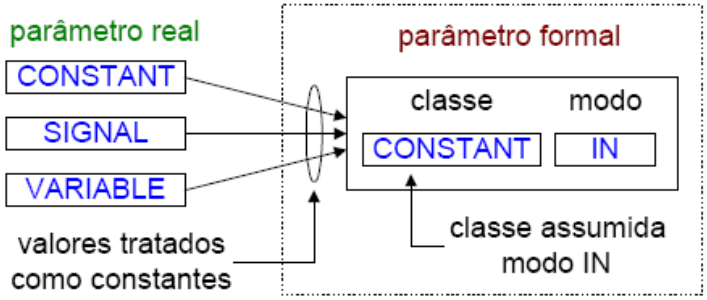


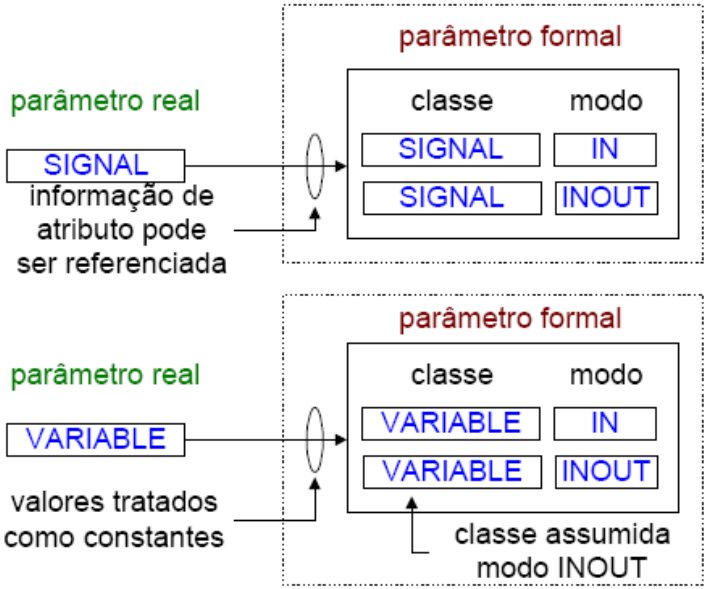
Associações possíveis: função		
Parâmetro real (passado)		Parâmetro formal (declarado)
constante sinal variável	→	constante
sinal	→	sinal

- **Nota**: Parâmetro formal classe Variable → não é permitido em funções

Subprogramas

- **Passagem de dados para procedimentos:**

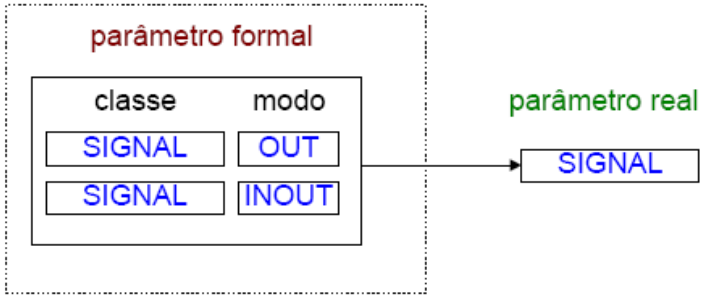


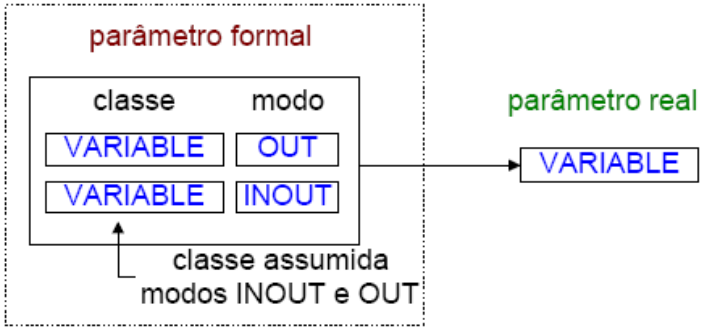


Associações possíveis em procedimentos		
Parâmetro real (passado)		Parâmetro formal (declarado)
constante sinal variável	→	constante
sinal	→	sinal
variável	→	variável

Subprogramas

- **Passagem de dados de procedimentos:**





Associações possíveis procedimentos		
Parâmetro formal (declarado)		Parâmetro real (recebido)
sinal	→	sinal
variável	→	variável

Funções

- **Duas partes: uma declaração e um corpo**

- **Declaração define:**
  - nome da função
  - lista dos parâmetros de entrada
  - tipo do argumento de retorno

- **Declaração da função não é necessária se o corpo estiver:**
  - na região de declarações de uma entidade,
  - no interior de uma arquitetura
    de um procedimento
    de uma outra função

```
-- declaracao da funcao
FUNCTION nome_funcao (SIGNAL    a : IN tipo_a;
                      SIGNAL    b :    tipo_b;      -- modo default IN
                      CONSTANT  c : IN tipo_c;
                                d :    tipo_d)      -- classe default CONSTANT,
                                                    -- modo   default IN
                      RETURN           tipo_r ;
```

Funções

- **Corpo da função:**
  - cabeçalho (cópia das informações contidas na declaração da função)
  - região de declarações: tipos, constantes, variáveis etc.
  - descrição do comportamento da função

```
-- corpo da funcao
FUNCTION nome_funcao (SIGNAL    a : IN tipo_a;
                      SIGNAL    b :    tipo_b;   -- modo default IN
                      CONSTANT  c : IN tipo_c;
                                d :    tipo_d)   -- classe default CONSTANT,
                                                 -- modo default IN
                      RETURN           tipo_r IS
 --
 -- declaracao de tipo, constante, variavel
 --
 BEGIN
    --
    -- regiao de codigo sequencial
    --
    RETURN expressao;                            -- valor de retorno
 END nome_funcao;
```

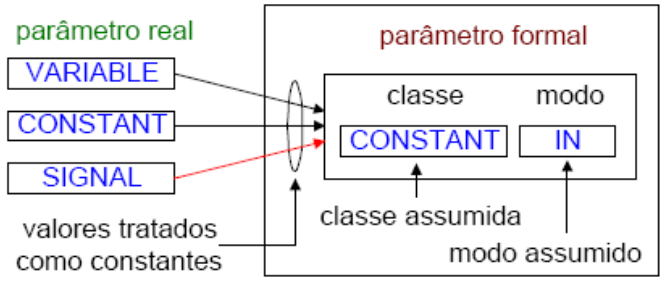
Exemplo: corpo e chamada de uma função

- **Função invocada de uma região de código concorrente**

- **Parâmetros formais classe Constant**

  (atributos dos sinais não podem ser referenciados)

- **Parâmetros reais classe Signal**

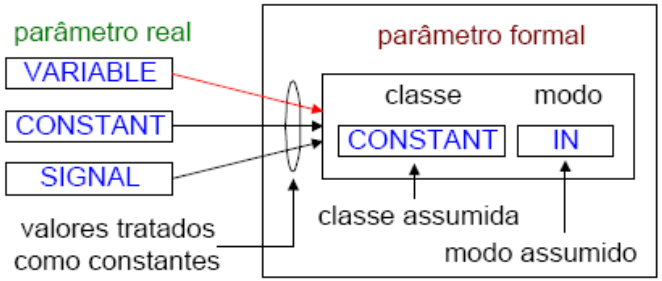



```
 1 ENTITY sb_fct0 IS
 2   PORT (a_i, b_i  : IN  INTEGER RANGE 0 TO 15;
 3         s_o       : OUT INTEGER RANGE 0 TO 15);
 4 END sb_fct0;
 5
 6 ARCHITECTURE exemplo OF sb_fct0 IS
 7
 8 FUNCTION soma (a : INTEGER; b : INTEGER) RETURN INTEGER IS
 9   VARIABLE s : INTEGER;
10 BEGIN
11   s :=  a + b;
12   RETURN s;
13 END soma;
14
15 BEGIN
16    s_o <= soma(a_i, b_i);
17 END exemplo;
```

Exemplo: corpo e chamada de uma função

- **Função invocada de uma região de código seqüencial**
- **Parâmetros formais classe Constant**

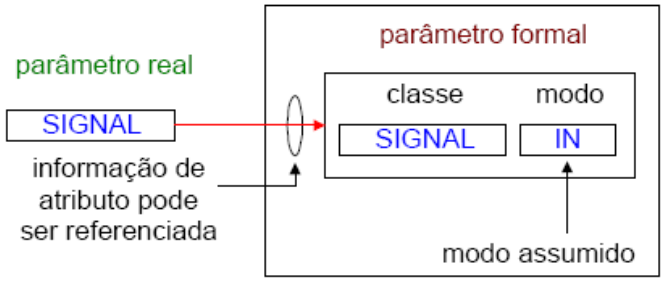
  (atributos dos sinais não podem ser referenciados)
- **Parâmetros reais classe Variable**




```
 1 ENTITY sb_fct1 IS
 2   PORT (a_i, b_i   : IN  INTEGER RANGE 0 TO 15;
 3         s_o        : OUT INTEGER RANGE 0 TO 15);
 4 END sb_fct1;
 5
 6 ARCHITECTURE exemplo OF sb_fct1 IS
 7
 8 FUNCTION soma (a : INTEGER; b : INTEGER) RETURN INTEGER IS
 9 BEGIN
10   RETURN a + b;
11 END soma;
12
13 BEGIN
14   abc: PROCESS(a_i, b_i)
15   VARIABLE a, b: INTEGER RANGE 0 TO 15;
16   BEGIN
17     a := a_i; b := b_i;
18     s_o <= soma(a, b);
19   END PROCESS;
20 END exemplo;
```

Exemplo: corpo e chamada de uma função

- **Função necessita dos atributos do sinal**:
  - detectar a borda de subida de um sinal
- **Parâmetros formais classe Signal**
  (atributos podem ser referenciados)




```
 1 ENTITY sb_fct8 IS
 2   PORT (ck, d : IN  BIT;
 3         q     : OUT BIT);
 4 END sb_fct8;
 5
 6 ARCHITECTURE teste OF sb_fct8 IS
 7
 8 FUNCTION subida (SIGNAL ck : IN BIT) RETURN BOOLEAN IS
 9 BEGIN
10   RETURN (ck'EVENT) AND (ck = '1');
11 END subida;
12
13 BEGIN
14   abc: PROCESS (ck)
15   BEGIN
16     IF subida(ck) THEN
17       q <= d;
18     END IF;
19   END PROCESS abc;
20 END teste;
```

Procedimentos

- **Invocados por um comando**

  exemplo: soma(a, b, s); (procedimento denominado soma)

- **Permitem o retorno de mais de um valor**

- **Classe dos objetos permitidas:**

  SIGNAL VARIABLE CONSTANT

- **Modo dos objetos:**

  IN OUT INOUT

- **Podem alterar o valor dos parâmetros passados**

- **Comando RETURN:**
  - termina o procedimento
  - não é obrigatório

Procedimentos

- **Similar as funções: duas partes:** uma declaração e um corpo

- **Declaração define:**
  - nome do procedimento
  - lista dos parâmetros de entrada e saída

- **Declaração do procedimento não é necessária se o corpo estiver:**
  - na região de declarações de uma entidade,
  - no interior de uma arquitetura
    de um outro procedimento
    de uma função

```
-- declaracao do procedimento
PROCEDURE nome_proced (SIGNAL   a : IN    tipo_a;
                       VARIABLE c : OUT   tipo_c;
                       VARIABLE e : INOUT tipo_e);
```

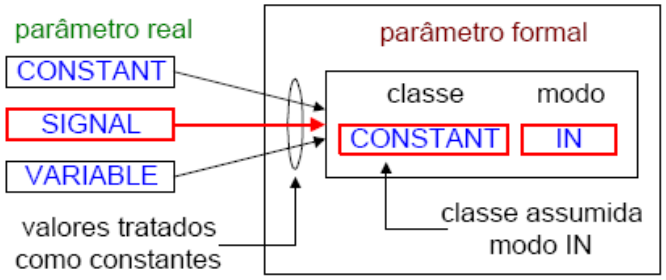
Procedimentos

- **Corpo do procedimento:**
  - cabeçalho (cópia das informações contidas na declaração da função)
  - região de declarações: tipos constantes variáveis
  - descrição do comportamento da função

```
-- declaracao do corpo
PROCEDURE nome_proced (SIGNAL   a : IN    tipo_a;
                       VARIABLE c : OUT   tipo_c;
                       VARIABLE e : INOUT tipo_e) IS
  --
  -- declaracao de tipo, constante, variavel
  --
BEGIN
   --
   -- regiao de codigo sequencial
   --
END nome_proced;
```

Exemplo: corpo e chamada de um procedimentos (continua na prox. transparência)

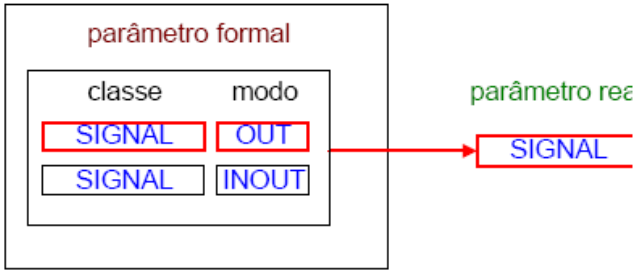
- **Procedimento invocado de uma região de código concorrente**
- **Parâmetro formal modo In classe Constant**
  (classe não especificada: assumido Constant)
- **Parâmetro real classe Signal**




```
 1 ENTITY sb_prc0 IS
 2   PORT (a_i, b_i, c_i : IN  INTEGER RANGE 0 TO 15;
 3         som_o, sub_o  : OUT INTEGER RANGE 0 TO 15);
 4 END sb_prc0;
 5
 6 ARCHITECTURE teste OF sb_prc0 IS
 7
 8 PROCEDURE soma_sub( a, b, c          : IN  INTEGER RANGE 0 TO 15;
 9                     SIGNAL som, sub : OUT INTEGER RANGE 0 TO 15) IS
10 BEGIN
11   som <= a + b;
12   sub <= a - c;
13 END soma_sub;
14
15 BEGIN
16   soma_sub(a_i, b_i, c_i, som_o, sub_o);
17 END teste;
```

Exemplo: corpo e chamada de um procedimentos (continuação transp. anterior)

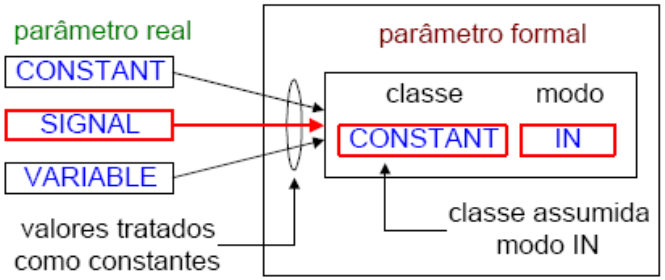
- **Parâmetros formais modo OUT**:
  - devem pertencer a classe Signal
  - são empregados em uma área de código concorrente
  - note: objeto da classe Variable em região de código concorrente → erro




```
 1 ENTITY sb_prc0 IS
 2   PORT (a_i, b_i, c_i : IN  INTEGER RANGE 0 TO 15;
 3         som_o, sub_o  : OUT INTEGER RANGE 0 TO 15);
 4 END sb_prc0;
 5
 6 ARCHITECTURE teste OF sb_prc0 IS
 7
 8 PROCEDURE soma_sub( a, b, c          : IN  INTEGER RANGE 0 TO 15;
 9                     SIGNAL som, sub : OUT INTEGER RANGE 0 TO 15) IS
10 BEGIN
11   som <= a + b;
12   sub <= a - c;
13 END soma_sub;
14
15 BEGIN
16   soma_sub(a_i, b_i, c_i, som_o, sub_o);
17 END teste;
```

Exemplo: corpo e chamada de um procedimentos (continua na prox. transparência)

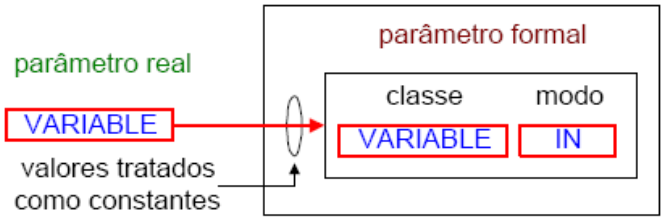
- **Procedimento invocado de uma região de código seqüencial** : 1ª opção
- **Parâmetro formal modo In classe Constant**
- **Parâmetro real classe Signal**




```
 8 PROCEDURE soma_sub(CONSTANT a   : IN  INTEGER RANGE 0 TO 15;    -- 1a opcao
 9                    VARIABLE b   : IN  INTEGER RANGE 0 TO 15;
10                    SIGNAL   c   : IN  INTEGER RANGE 0 TO 15;
11                    VARIABLE som : OUT INTEGER RANGE 0 TO 15;
12                    SIGNAL   sub : OUT INTEGER RANGE 0 TO 15) IS
13 BEGIN
14   som := a + b;
15   sub <= a - c;
16 END soma_sub;
17
18 BEGIN
19   abc: PROCESS(a_i, b_i, c_i)
20   VARIABLE b_v, som_v : INTEGER RANGE 0 TO 15;
21   BEGIN
22     b_v := b_i;
23     soma_sub(a_i, b_v, c_i, som_v, sub_o);
24     som_o <= som_v;
25   END PROCESS;
26 END teste;
```

Exemplo: corpo e chamada de um procedimentos (continuação transp. anterior)

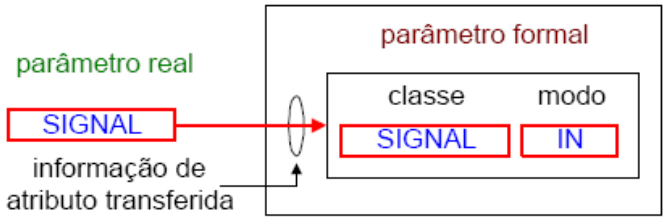
- **Procedimento invocado de uma região de código seqüencial** : 2ª opção
- **Parâmetro formal modo In classe Variable**
- **Parâmetro real classe Variable**




```
 8 PROCEDURE soma_sub(CONSTANT a   : IN  INTEGER RANGE 0 TO 15;
 9                    VARIABLE b   : IN  INTEGER RANGE 0 TO 15;   -- 2a opcao
10                    SIGNAL   c   : IN  INTEGER RANGE 0 TO 15;
11                    VARIABLE som : OUT INTEGER RANGE 0 TO 15;
12                    SIGNAL   sub : OUT INTEGER RANGE 0 TO 15) IS
13 BEGIN
14   som := a + b;
15   sub <= a - c;
16 END soma_sub;
17
18 BEGIN
19   abc: PROCESS(a_i, b_i, c_i)
20   VARIABLE b_v, som_v : INTEGER RANGE 0 TO 15;
21   BEGIN
22     b_v := b_i;
23     soma_sub(a_i, b_v, c_i, som_v, sub_o);
24     som_o <= som_v;
25   END PROCESS;
26 END teste;
```

Exemplo: corpo e chamada de um procedimentos (continuação transp. anterior)

- **Procedimento invocado de uma região de código seqüencial** : 3ª opção
- **Parâmetro formal modo In classe Signal**
- **Parâmetro real classe Signal**




```
 8 PROCEDURE soma_sub(CONSTANT a   : IN  INTEGER RANGE 0 TO 15;
 9                    VARIABLE b   : IN  INTEGER RANGE 0 TO 15;
10                    SIGNAL   c   : IN  INTEGER RANGE 0 TO 15;    -- 3a opcao
11                    VARIABLE som : OUT INTEGER RANGE 0 TO 15;
12                    SIGNAL   sub : OUT INTEGER RANGE 0 TO 15) IS
13 BEGIN
14   som := a + b;
15   sub <= a - c;
16 END soma_sub;
17
18 BEGIN
19   abc: PROCESS(a_i, b_i, c_i)
20   VARIABLE b_v, som_v : INTEGER RANGE 0 TO 15;
21   BEGIN
22     b_v := b_i;
23     soma_sub(a_i, b_v, c_i, som_v, sub_o);
24     som_o <= som_v;
25   END PROCESS;
26 END teste;
```

Sobrecarregamento de subprogramas - *overloading*

- **Mesma designação para mais de um subprograma**
  - funcionalmente semelhantes
  - diferem na lista de parâmetros

- **Critério de seleção do subprograma**: (compilador)
  - número de parâmetros
  - tipo dos parâmetros.

- **Impossibilidade de se identificar o subprograma:**
  - condição de erro

Exemplo de sobrecarregamento: Três funções com a mesma designação → soma

```
 1 ENTITY sb_over1 IS
 2   PORT (ai, bi, ci : IN  INTEGER RANGE 0   TO 15;
 3         ar, br     : IN  REAL    RANGE 0.0 TO 15.0;
 4         si, ti     : OUT INTEGER RANGE 0   TO 31;
 5         sr         : OUT REAL    RANGE 0.0 TO 31.0);
 6 END sb_over1;
 7
 8 ARCHITECTURE teste OF sb_over1 IS
 9   FUNCTION soma (a, b : INTEGER) RETURN INTEGER IS     -- 1a funcao
10   BEGIN
11     RETURN a + b;
12   END soma;
13
14   FUNCTION soma (a, b, c : INTEGER) RETURN INTEGER IS  -- 2a funcao
15   BEGIN
16     RETURN a + b + c;
17   END soma;
18
19   FUNCTION soma (x, y : REAL) RETURN REAL IS           -- 3a funcao
20   BEGIN
21     RETURN x + y + 1.0;
22   END soma;
23 BEGIN
24   si <= soma(ai,bi);     -- ai,bi operandos tipo integer    -> 1a funcao
25   ti <= soma(ai,bi,ci); -- ai,bi,ci operandos tipo integer -> 2a funcao
26   sr <= soma(ar,br);     -- ar,br operandos tipo real       -> 3a funcao
27 END teste;
```

Sobrecarregamento de operadores pré-definidos

- **Em VHDL operadores como**: AND + =
  - são funções declaradas na biblioteca padrão:
  - recebem um ou dois parâmetros e retornam um valor

- **Regras de sobrecarga de uma função**:
  - podem ser aplicadas nos operadores pré definidos

- **Nome da função** → deve ser declarado entre aspas

- **Sobrecarga de operadores**:
  - restrita aos símbolos existentes
  - expande a abrangência de uma operação pré-definida
    - objetivo: facilitar a leitura de um código

Exemplo: sobrecarregamento do operador de adição

- **Uso: adição de dois valores do tipo bit_vetor**
- **Operador + aplicado entre operandos do tipo bit_vetor**
  - a função + (linhas 8 a 17) empregada na linha 19

```
 1 ENTITY sb_over2 IS
 2   GENERIC(n : INTEGER := 3);
 3   PORT (a, b : IN  BIT_VECTOR(n-1 DOWNTO 0);
 4         s    : OUT BIT_VECTOR(n-1 DOWNTO 0));
 5 END sb_over2;
 6
 7 ARCHITECTURE teste OF sb_over2 IS              -- retorna tipo
 8   FUNCTION "+" (x, y : BIT_VECTOR(n-1 DOWNTO 0)) RETURN BIT_VECTOR IS
 9     VARIABLE v : BIT := '0';
10     VARIABLE s : BIT_VECTOR(n-1 DOWNTO 0);
11   BEGIN
12     FOR i IN 0 TO n-1 LOOP
13       s(i) :=  x(i) XOR y(i) XOR v;
14       v    := (x(i) AND y(i)) OR (v AND (x(i) OR y(i)));
15     END LOOP;
16     RETURN s;
17   END "+";
18 BEGIN
19   s <= a + b;  -- a,b operandos tipo bit-vector
20 END teste;
```

Argumento sem especificação de limites

- **Limites dos vetores em parâmetros formais:**
  - podem ser deixados em aberto

- **A dimensão dos vetores:**
  - determinada pelo tamanho do parâmetro

- **Emprego:**
  - criação de funções e procedimentos de uso mais amplo
  - redução da extensão do código (diferentes tamanhos podem ser empregados)

- **Cuidados:**
  - uma boa descrição deve prever:
    - vetor crescente (a TO b)
    - como decrescente (b DOWNTO a)

Exemplo: sobrecarregamento do operador de adição

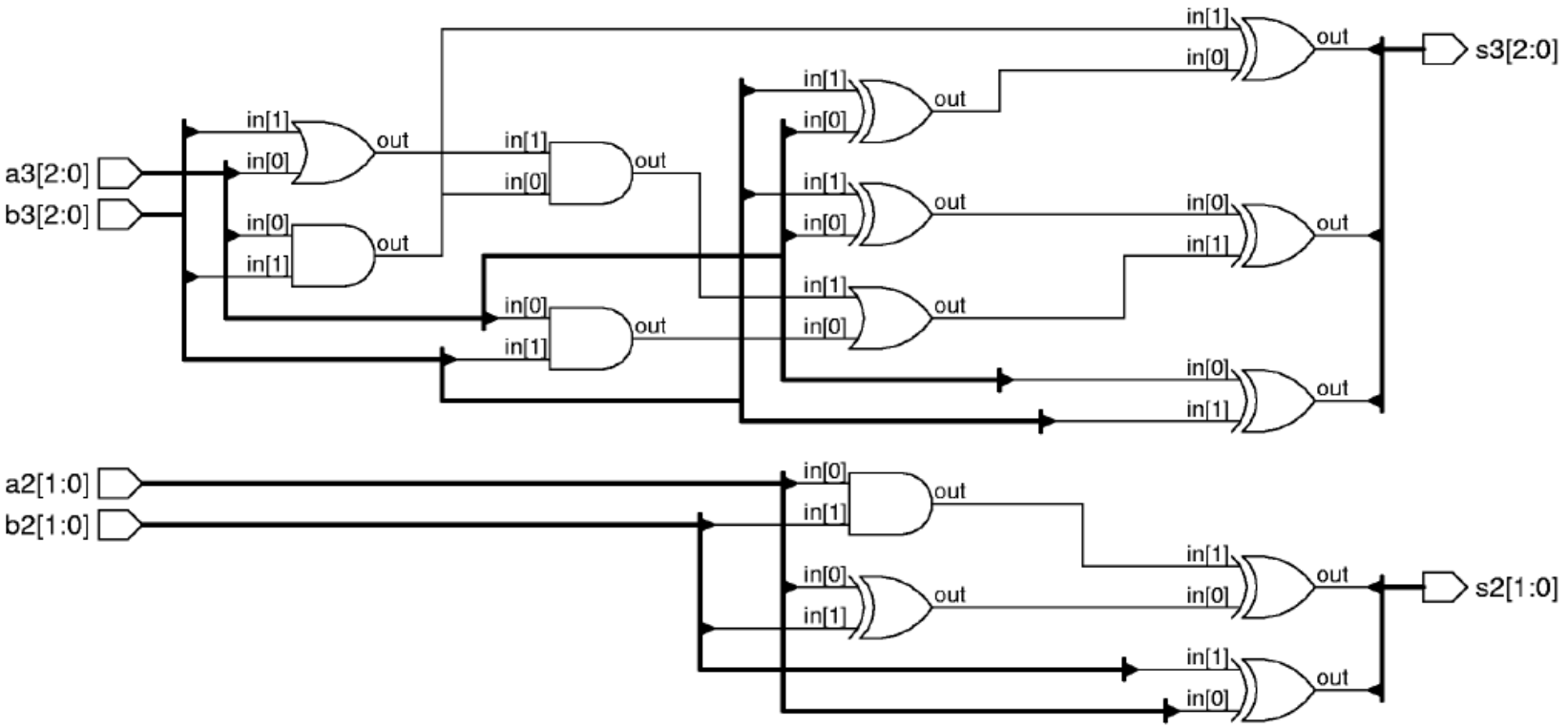
- **Uso**: adição de dois valores do tipo bit_vetor -

(tamanho e/ou ordenação do vetor não é/são definido/s)

```
 1 ENTITY sb_over3 IS
 2   PORT (a2, b2 : IN  BIT_VECTOR(1 DOWNTO 0);
 3         a3, b3 : IN  BIT_VECTOR(2 DOWNTO 0);
 4         s2     : OUT BIT_VECTOR(1 DOWNTO 0);
 5         s3     : OUT BIT_VECTOR(2 DOWNTO 0));
 6 END sb_over3;
 7
 8 ARCHITECTURE teste OF sb_over3 IS
 9   FUNCTION "+" (x, y : BIT_VECTOR) RETURN BIT_VECTOR IS
10     VARIABLE v : BIT := '0';
11     VARIABLE s : BIT_VECTOR(x'LENGTH-1 DOWNTO 0);
12   BEGIN
13     FOR i IN 0 TO x'LENGTH-1 LOOP
14       s(i) :=  x(i) XOR y(i) XOR v;
15       v    := (x(i) AND y(i)) OR (v AND (x(i) OR y(i)));
16     END LOOP;
17     RETURN s;
18   END "+";
19 BEGIN
20   s2 <= a2 + b2;  -- a2,b2 operandos 2 bits tipo bit-vector
21   s3 <= a3 + b3;  -- a3,b3 operandos 3 bits tipo bit-vector
22 END teste;
```

Exemplo: sobrecarregamento do operador de adição

- **Circuito sintetizado:**



Comportamento de variáveis declaradas em subprogramas

- **Variável declarada em um subprograma**:

  - perdura até o término do subprograma

- **Variável declarada em um processo**:

  - persiste até o término da simulação

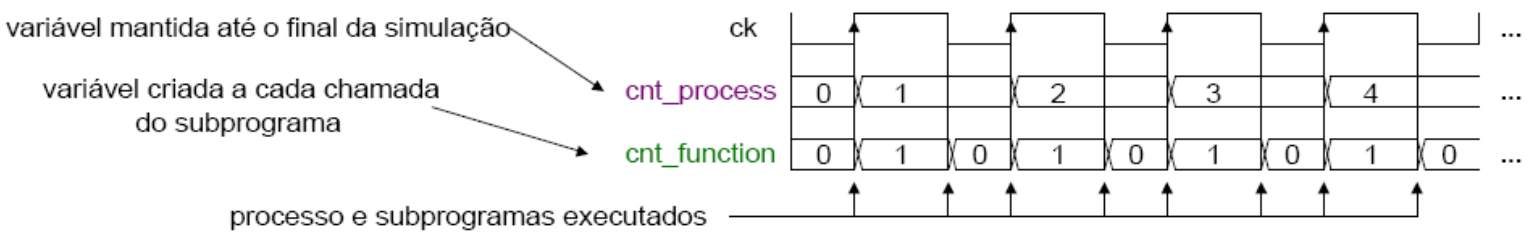
- **Assim**:

  - para cada chamada de um subprograma: a variável criada e inicializada
  - o seu valor não é mantido

Comportamento de variáveis declaradas em subprogramas

- **Exemplo**: variável em função

```
 8 FUNCTION conta_f (SIGNAL ck : IN BIT) RETURN INTEGER IS
 9   VARIABLE af : INTEGER RANGE 0 TO 15;
10 BEGIN
11   IF (ck'EVENT) AND (ck = '1') THEN af := af +1;
12   END IF;
13   RETURN af;
14 END conta_f;
15
16 BEGIN
17   abc: PROCESS (ck)
18     VARIABLE a : INTEGER RANGE 0 TO 15;
19   BEGIN
20     IF (ck'EVENT) AND (ck = '1') THEN  a := a +1;
21     END IF;
22     cnt_process <= a;
23    END PROCESS;
24
25   cnt_function <= conta_f(ck);
26 END exemplo;
```



Comportamento de variáveis declaradas em subprogramas

- **Exemplo**: variável em procedimento

```
 8 PROCEDURE conta_p(SIGNAL ck :IN BIT; SIGNAL a_o :OUT INTEGER RANGE 0 TO 15) IS
 9   VARIABLE ap : INTEGER RANGE 0 TO 15;
10 BEGIN
11   IF (ck'EVENT) AND (ck = '1') THEN  ap := ap +1;
12   END IF;
13   a_o <= ap;
14 END conta_p;
15
16 BEGIN
17   abc: PROCESS (ck)
18     VARIABLE a : INTEGER RANGE 0 TO 15;
19   BEGIN
20     IF (ck'EVENT) AND (ck = '1') THEN a := a +1;
21     END IF;
22     cnt_process <= a;
23   END PROCESS;
24
25   conta_p(ck, cnt_procedure);
26 END exemplo;
```