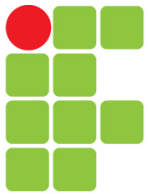
VHDL: Subprogramas e tipos

Engenharia Eletrônica

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

20 de abril de 2020



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópico

Objetivos

- Tópicos da aula de hoje:
  - Subprogramas
  - Tipos

Subprogramas

Um subprograma é um maneira de isolar um conjunto de comandos usados frequentemente com o objetivo de melhorar o entendimento da descrição/código.

- Há dois tipos de **subprogramas**
  - Função: realiza um cálculo e retorna um valor
  - Procedimento: realiza uma sequencia definida de instruções sequenciais
- O uso de subprogramas não resulta em uma economia de recursos no circuito final sintetizado, pois cada chamada pode levar à criação de unidade funcionais

Subprogramas

- Um subprograma é definido por uma declaração e um corpo
  - A declaração pode ser opcional
  - Não é opcional se o subprograma é chamado antes da declaração do corpo
- Subprograma consistem em instruções sequenciais (como processos)
- Podem ser declarados em um processo, arquitetura ou pacote
  - Lugar da declaração determina escopo e visibilidade
  - Se declarados em um pacote, a declaração do subprograma vai na declaração do pacote e o corpo, no corpo do pacote

Tópico

Funções

- É uma seção com código **sequencial**.
- Seu objetivo é lidar com problemas comuns como:
  - Conversão de dados.
  - Operações lógicas.
  - Operações aritméticas.
  - Novas operações e atributos.
- A ideia de criar função é compartilhar e reutiliza código, objetivando um código principal limpo e curto.

Função

- Declaração:

```
FUNCTION <nome> ( <parametros> ) RETURN <tipo>;

function ones_count (signal a : std_logic_vector) return variable;
```

- Corpo:

```
function ones_count (signal a :
      std_logic_vector) is
  variable r : integer;
begin
  r := 0;

  for i in a'range loop
    if a(i) /= '0' then
      r := r + 1 ;
    end if;
  end loop;

  return r;
end function ones_count;
```

- Chamada:
  - Deve retornar um valor **único** baseado nos parâmetros de entrada
  - Deve ser chamada em uma **expressão**

```
total_ones <= ones_count (input)
              WHEN test_ones = '1';
```

Exemplo

- Declaração no código principal (entity/architecture).
- Sobrecarga do nome.

```
entity name_overload is
  port (ai, bi, ci : in  integer range 0   to 15;
        ar, br     : in  real    range 0.0 to 15.0;
        si, ti     : out integer range 0   to 31;
        sr         : out real    range 0.0 to 31.0);
end name_overload;

architecture teste of name_overload is
  function soma (a, b : integer) return integer is     -- 1a funcao
  begin
    return a + b;
  end soma;

  function soma (a, b, c : integer) return integer is  -- 2a funcao
  begin
    return a + b + c;
  end soma;

  function soma (x, y : real) return real is           -- 3a funcao
  begin
    return x + y + 1.0;
  end soma;
begin
  si <= soma(ai,bi);    -- ai,bi operandos tipo integer    -> 1a funcao
  ti <= soma(ai,bi,ci); -- ai,bi,ci operandos tipo integer -> 2a funcao
  sr <= soma(ar,br);    -- ar,br operandos tipo real       -> 3a funcao
 end teste;
```

Função

- Declaração no pacote principal (package/library).
- Sobrecarga do operador.

```
library ieee;
use ieee.std_logic_1164.all;

package my_package is
  function "+" (x, y : std_logic_vector)  return std_logic_vector;
end package my_package;

package body my_package is

  function "+" (x, y : std_logic_vector)  return std_logic_vector is

    variable sum : std_logic_vector (x'length-1 downto 0);
    variable carry : std_logic;

  begin
    carry := '0';
    for i in x'reverse_range  loop
      sum(i) :=  x(i) xor y(i) xor carry;
      carry    := (x(i) and y(i)) or
        (carry and (x(i) or y(i)));
    end loop;

    return sum;
  end "+";

end package body my_package;
```

Função

- Declaração no pacote principal (package/library).
- Sobrecarga do operador.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-- Utilização das funções e
-- procedimentos de my_package
use work.my_package.all;

entity testbench is

end entity testbench;

architecture RTL of testbench is

  signal a_logic : std_logic_vector (3 downto 0);
  signal b_logic : std_logic_vector (3 downto 0);
  signal sum : std_logic_vector (3 downto 0);

begin

  -- Exemplo overload ("+" para std_logic_vector)
  a_logic <= "0101";
  b_logic <= "0010";
  sum <= a_logic + b_logic;
end;
```

Tópico

Procedimento

- Declaração:

```
PROCEDURE <nome> ( <parametros> );

procedure soma_sub( a, b, c          : in  integer range 0 to 15;
                    signal som, sub  : out integer range 0 to 15);
```

- Corpo:

```
procedure soma_sub( a, b, c :
        in  integer range 0 to 15;
  signal som, sub :
        out integer range 0 to 15) is
begin
  som <= a + b;
  sub <= a - c;
end soma_sub;
```

- Chamada:
  - Pode ter entradas e saídas
  - Pode retornar nenhuma ou múltiplas saídas
  - **Deve** ser chamado em uma instrução sequencial separada

```
soma_sub(a_i, b_i, c_i, som_con, sub_con);
```

Exemplo

```
entity sb_prc0 is
  port (a_i, b_i, c_i     : in  integer range 0 to 15;
        som_con, sub_con : out integer range 0 to 15);
end sb_prc0;

architecture teste of sb_prc0 is

procedure soma_sub( a, b, c : in  integer range 0 to 15;
                    signal som, sub : out integer range 0 to
                        15) is
begin
  som <= a + b;
  sub <= a - c;
end soma_sub;

begin
  soma_sub(a_i, b_i, c_i, som_con, sub_con);
end teste;
```

Funções vs procedimentos

Funções

- Sempre executam em tempo zero
  - Não podem interromper sua execução
  - Não podem contar atrasos, eventos ou controle de tempo
- Devem ter pelo menos um argumento de entrada
- Argumentos não precisam ser entradas ou saídas
- Retornam apenas **um** valor

Procedimentos

- Pode executar em tempo zero:
  - Pode contar atrasos, eventos e controle de tempo
- Pode ter nenhum ou várias entradas e saídas
- Pode modificar nenhum ou vários valores

Exercícios

Tópico

Tipos

- VHDL já possui os tipos de dados para representação de hardware
  - BIT
  - BOOLEAN
  - STD_LOGIC

- Há ainda:
  - Criação de subtipos
  - Tipos enumerados
  - Vetores
  - Registros
  - Pseudônimos

Subtipos

- **Subtype** é um tipo restrito, baseado em um tipo já definido
- É sintetizável se o tipo base for sintetizável
- Usado para melhorar legibilidade do código
  - Pode ser colocado em um pacote para uso em todo o projeto

```
architecture logic of subtype_test is

  subtype word is std_logic_vector (31 downto 0);

  signal mem_read, mem_write : word;
  subtype dec_count is integer range 0 to 9;
  signal ones, tens : dec_count;

begin
  (...)
```

Dados enumerados

- Permite-se criar um dado que associa nome a um número
  - Deve-se instanciar o objeto depois (variável, sinal ou constante)
- Importante para:
  - Legibilidade do código
  - Criação de máquinas de estado finitas
- Definição:

```
TYPE <nome_do_tipo> IS (itens separados por virgula);
```

```
architecture logic of subtype_test is
  type enum is (IDLE, FILLl, HEAT_W, WASH, DRAIN); -- definição

  signal dshwshr_st : enum; -- instância
  (...)
begin
  drain_led <= '1' WHEN dshwsher_st = drain ELSE '0';
```

Vetores (arrays)

- Cria-se tipos multidimensionais
  - Deve-se instanciar o objeto depois (variável, sinal ou constante)

- Utilizado para criar (e sintetizar) memórias além de manter dados de simulação

- Definição:

```
type <nome_do_tipo> is array (<limite inteiro —>
    profundidade do array) of <tipo_do_dado>;
```

Vetores (arrays)

```
architecture logic of my_memory is
  -- cria um novo tipo vetor chamado mem
  -- com 64 endereços de 8 bits cada
  type mem is array (0 to 63) of std_logic_vector (7 downto 0);

  -- cria as instâncias (memórias) efetivamente
  -- 2 arrays de 64x8 bits
  signal mem_64x8_a, mem_64x8_b : mem;

begin

  mem_64x8_a(12) <= x"af";
  mem_64x8_b(50) <= "11110000";

  (...)
end architecture logic;
```

Vetores (arrays) – generate

```
architecture RTL of registers is
  component reg16
    port(
      clk       : in std_logic;
      clear     : in std_logic;
      w_flag    : in std_logic;
      datain    : in std_logic_vector (15 downto 0);
      reg_out   : out std_logic_vector (15 downto 0)
    );
  end component registers;

  type reg_array is array (0 to 3) of std_logic_vector (15 downto 0);
  signal registers : reg_array;

begin
  -- gera três componentes: observem reg_out
  reg_gen: for i in 0 to 3 generate
  regs: reg16
    port map (
      clk => clk,
      clear => rst,
      w_flag => wr,
      datain => data_in,
      reg_out => registers(i)
    );
  end generate;
end architecture RTL;
```

Vetores (arrays)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_1 is
  port(endereco : in  unsigned(3 downto 0); -- valores de 0 a 15
       ce       : in  std_logic;            -- habitita memoria
       saida    : out unsigned(7 downto 0));-- saida de dados da memoria
end rom_1 ;

architecture teste of rom_1 is
  type     arranjo_memoria is array (natural range <>) of unsigned(7 downto 0);

  constant dados : arranjo_memoria(0 to 15) :=
  ("00000011","10011111","00100101","00001101","10011000", -- valores posicoes 0,1,2,3,4
   "01001001","01000001","00011111","00011001","00001001", -- valores posicoes 5,6,7,8,9
   "01001001","01000001","00010111","00000001","01001001","01000110"); -- posicoes restantes

begin
  saida <= dados(to_integer(endereco)) when ce='0' else (others=>'z');
end teste;
```

Registros

- Agrupa-se e tipos diferentes de dados nomeando cada um dos membros
  - Deve-se instanciar o objeto depois (variável, sinal ou constante)
- É sintetizável se os tipos base forem sintetizáveis
- Utilizado para melhorar a legibilidade do código
- Definição:

```
type <nome_to_tipo> is record
    nome_elemento_1 : <tipo_a>;
    nome_elemento_2 : <tipo_b>;
    nome_elemento_n : <tipo_c>;
end record <nome_to_tipo>;
```

Registros

```
architecture teste of rec_ts1a is

 type relogio_t is  record
    segundo : std_logic_vector(5 downto 0);
    minuto : std_logic_vector(5 downto 0);
    hora : std_logic_vector(4 downto 0);
  end record;

signal tempo   : relogio_t;

begin
  tempo.segundo <= "000000";
  tempo.minuto <= "000010";
  tempo.hora <= "000000";
```

Registros

```
entity rec_ts1a is end rec_ts1a;

architecture teste of rec_ts1a is
  type sinais_controle is  record
    r_dado   : integer range 0 to 15;
    r_tempo  : time;
    r_atraso : boolean;
  end record;

  signal dado, soma : integer range 0 to 31;
  signal tempo      : time;
  signal atraso     : boolean;
  signal estimulo   : sinais_controle;

begin
  dado   <= estimulo.r_dado;
  tempo  <= estimulo.r_tempo;
  atraso <= estimulo.r_atraso;

  estimulo <= ( 2, 50 ns, true),
              ( 1, 35 ns, false) after 100 ns,
              ( 5, 70 ns, true)  after 200 ns;

  abc: process(dado, tempo, atraso)
  begin
    if  atraso then
      soma <= dado + soma after tempo;
    else
      soma <= dado + soma;
    end if;
  end proces
```

Pseudônimo

- É uma designação adicional para um objeto (contante, sinal ou variável)
- É sintetizável se os tipos base forem sintetizáveis
- Utilizado para melhorar a legibilidade do código

```
architecture teste of alias_a is
  signal dado       : bit_vector(7 downto 0);
  alias  dado_alto  : bit_vector(3 downto 0) is dado(7 downto 4);
  alias  dado_baixo : bit_vector(3 downto 0) is dado(3 downto 0);
begin
  dado_alto  <= not h;  -- equivalente a: dado(7 downto 4) <= not h;
  dado_baixo <= not l;  -- equivalente a: dado(3 downto 0) <= not l;
  h_l <= dado_alto & dado_baixo;
end teste;
```

Tópico

Referências

- PEDRONI, Volnei A.Circuit Design with VHDL. MIT. 2004.

- D'AMORE, Roberto. VHDL: descrição e síntese de circuitos digitais. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2012. 292 p., il. ISBN 9788521620549.