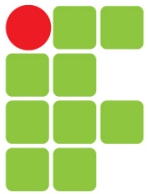
VHDL: Tipos e pacotes

Engenharia Eletrônica

Prof. Renan Augusto Starke

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

13 de novembro 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:
  - Unidades básicas de projeto
  - Pacotes
  - Bibliotecas
  - Tipos básico

VHDL

Vhsic (Very High Speed Integrated Circuit)
Hardware
Description
Language

O que é VHDL?

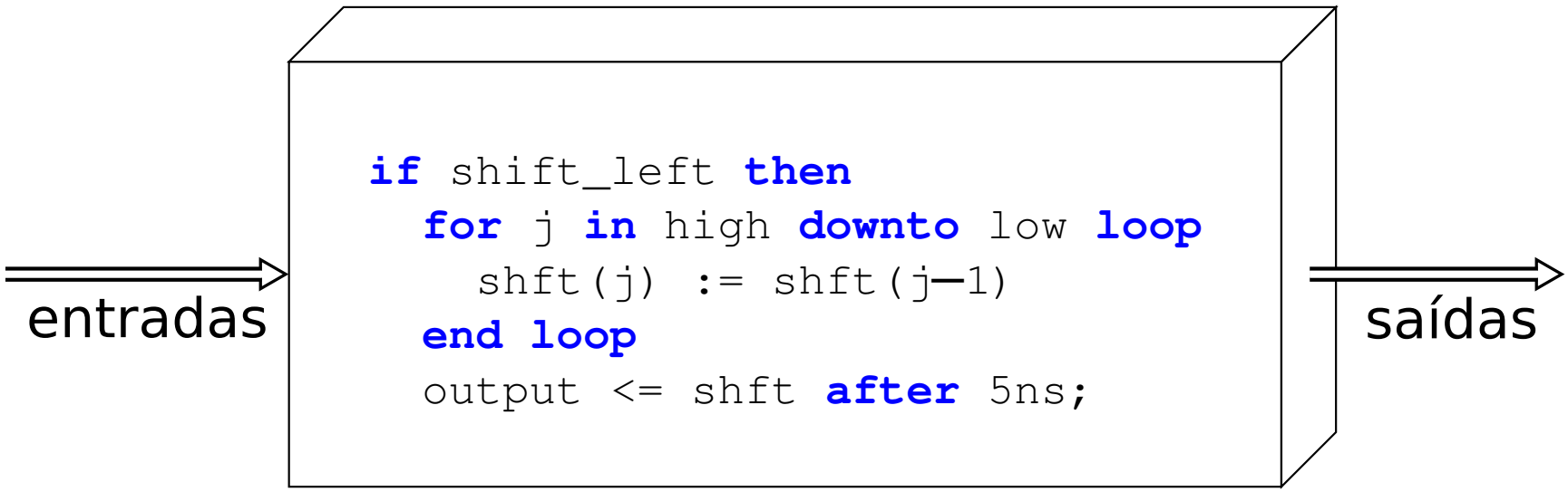
- Padrão industrial IEEE para linguagem de descrição de **hardware**
- Linguagem de descrição em **alto nível** para simulação e síntese

Terminologia

- HDL: Uma linguagem de descrição de hardware é uma linguagem de programação utilizada para **modelar** um elemento de hardware
- Modelagem comportamental: Um componente é descrito através de sua resposta de entrada e saída
- Modelagem estrutural: Um componente é descrito pela interconexão de baixo nível de seus componentes

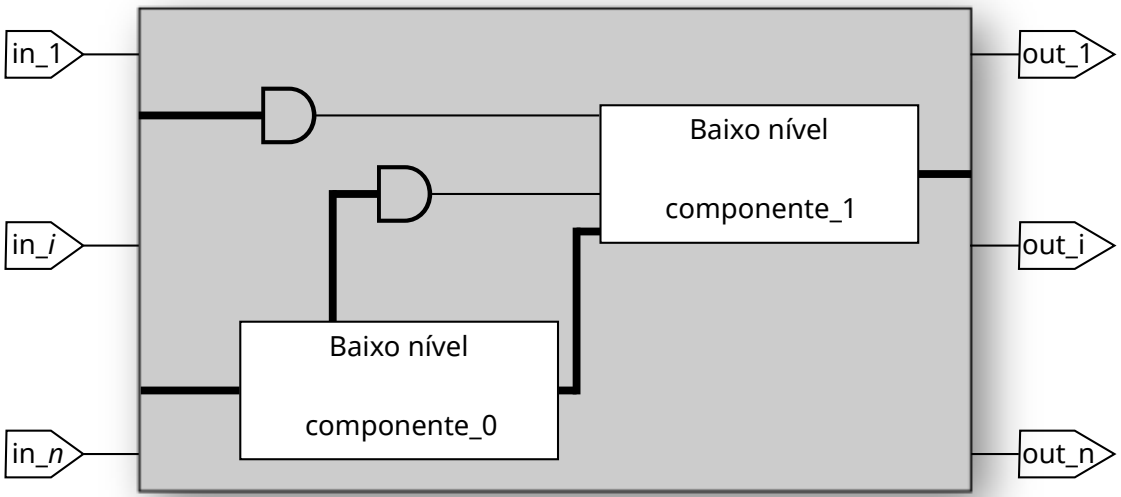
Modelagem comportamental

- Apenas a funcionalidade do circuito, sem estrutura
- Nenhuma intenção específica de hardware



Modelagem estrutural

- Funcionalidade e estrutura do circuito
- Respectivo a um hardware específico



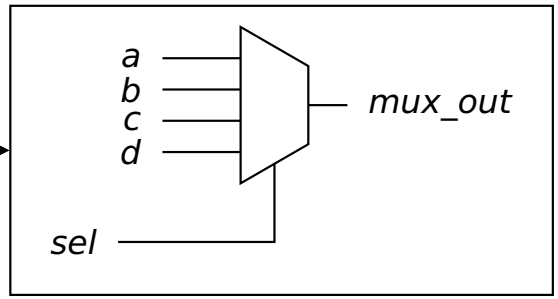
Terminologia (cont.)

- *Register Transfer Level* (RTL): Um tipo de modelagem comportamental. Objetiva a síntese
  - Hardware é implementado ou inferido
  - Sintetizável
- Síntese: Tradução do HDL para um circuito para então sua otimização e representação
- Processo: Unidade básica de execução em VHDL
  - Execução de processos são convertidos para hardware equivalente

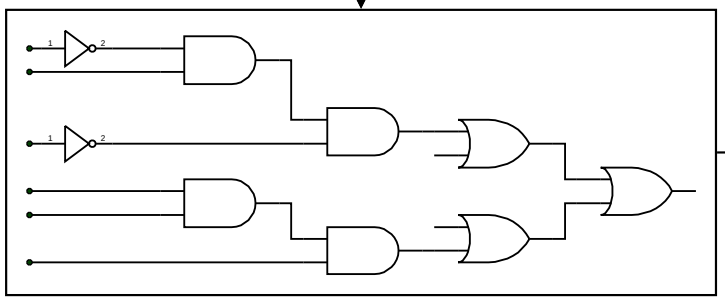
Síntese RTL

```
process (a,b,c,d,sel)
begin
  case (sel) is
    when "00" => mux_out <= a;
    when "01" => mux_out <= b;
    when "10" => mux_out <= c;
    when "11" => mux_out <= d;
  end case
end
```

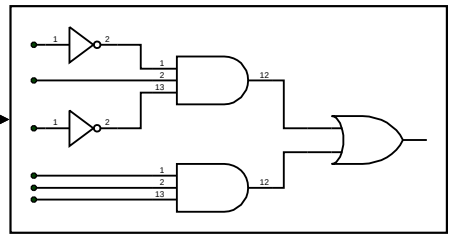
inferido



tradução



otimiz.



RTL

- Projeto combinacional:
  - Capturar o comportamento lógico combinacional através de uma tabela verdade ou equação
  - Converter o comportamento desejado em um circuito
- Projeto sequencial:
  - Capturar o comportamento sequencial através de uma máquina de estados finitos
  - Converter o comportamento desejado em circuito
- Projeto RTL:
  - Capturar o comportamento desejado em alto nível utilizando uma máquina de estados finitos
  - Converter o comportamento em um circuito, contudo inicialmente não diferencia-se lógica sequencial, combinacional ou RTL

RTL (cont.)

- Projeto RTL:

1. Capturar o comportamento do sistema em alto nível através de uma máquina de estados.
2. Criar um fluxo de dados (*datapath)*
3. Conectar o *datapath* em um controlador
4. Derivar o máquina de estados finitos (FSM) do controlador.

Síntese VHDL e outros padrões HDL

- VHDL:
  - “Diga-me como o o circuito deve-se comportar e eu darei o hardware que faz o trabalho”
- Verilog:
  - Similar ao VHDL
- ABEL, PALASM, AHDL:
  - “Diga-me qual hardware e eu darei-o para você”
- System Verilog, SystemC:
  - Flexível, porém voltado a modelagem completa do sistema.

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

entity mux_using_with is
    port (
        din_0   :in  std_logic; -- Mux first input
        din_1   :in  std_logic; -- Mux Second input
        sel     :in  std_logic; -- Select input
        mux_out :out std_logic  -- Mux output
    );
end entity;

architecture behavior of mux_using_with is
begin
    with (sel) select
    mux_out <= din_0 when '0',
               din_1 when others;
end architecture;
```

Verilog

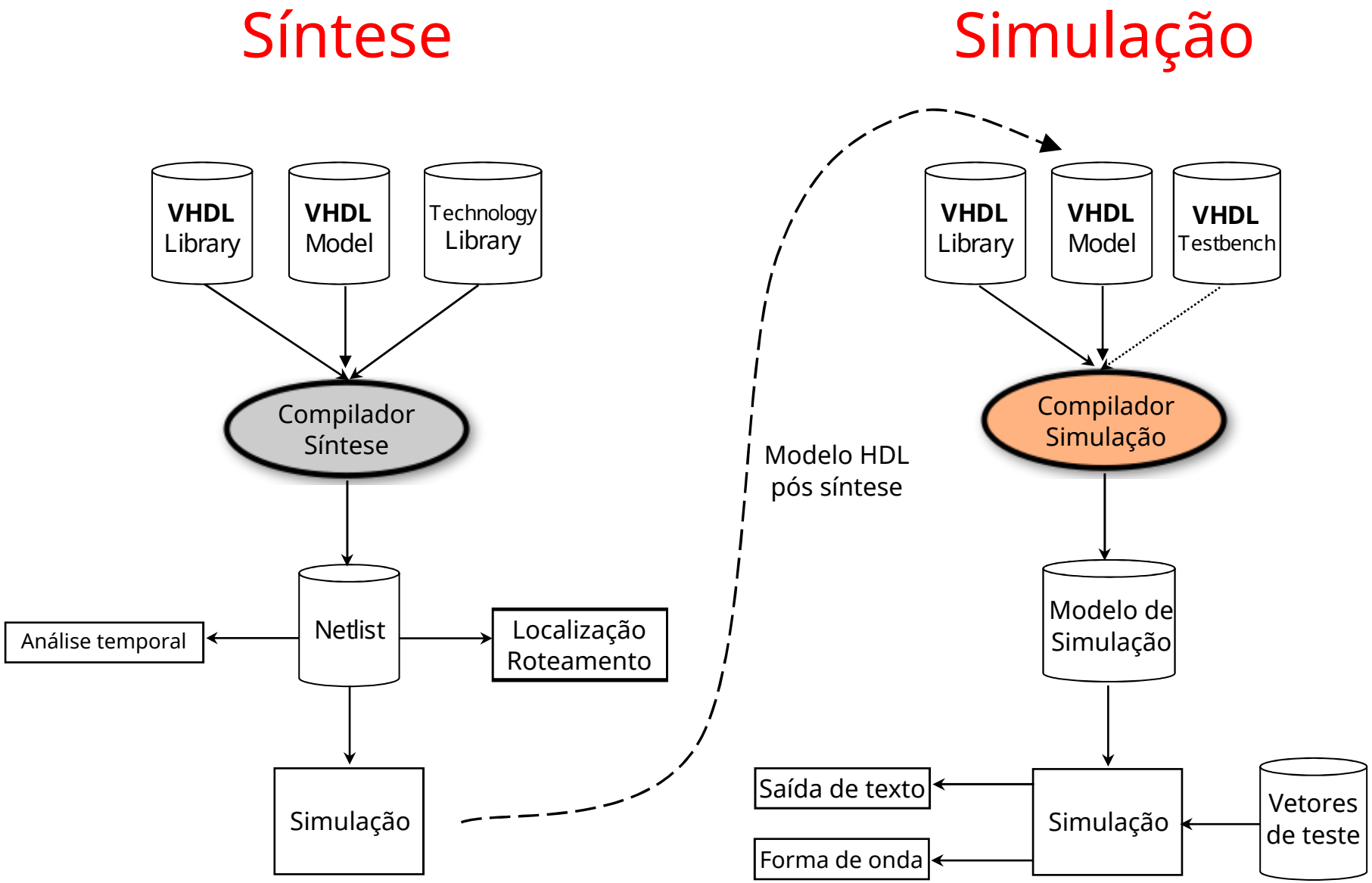
```
module  mux_using_assign(

  din_0       , // Mux first input
  din_1       , // Mux Second input
  sel         , // Select input
  mux_out       // Mux output
  );

  input din_0, din_1, sel ;
  output mux_out;
  wire  mux_out;

  assign mux_out = (sel) ? din_1 : din_0;

endmodule
```

SystemC

```
SC_MODULE( mux2_1 )
{
  sc_in<sc_int<WORD_SIZE> > in0;
  sc_in<sc_int<WORD_SIZE> > in1;

  sc_in<sc_uint<3> > addr;

  sc_out<sc_int<WORD_SIZE> > mux_out;

  void do_mux ();

    SC_CTOR( mux2_1  )
  {
      SC_METHOD(do_mux);
      sensitive << in0 << in1 << addr;
    }

};
```

```
void mux2_1::do_mux()
{
  sc_int<WORD_SIZE> local_in0;
  sc_int<WORD_SIZE> local_in1;

  sc_uint<3> local_addr;

  local_in0 = in0.read();
  local_in1 = in1.read();
  local_addr = addr.read();

  switch (local_addr)
  {
    case 0:
      mux_out.write(local_in0);

(...)
```

Fluxo de Síntese e Simulação típico para RTL



Conceitos básico de VHDL

- Duas construções:
  - Simulação
  - Síntese e simulação
- A linguagem VHDL é construída por palavras reservadas
- A maior parte da linguagem **não é** sensível a caixa (case sensitive)
- Declarações VHDL são terminadas por **ponto e vírgula – ;**
- Comentários começam com “–”
- Declarações VHDL devem ser analisadas com execuções concorrentes

Tópico

Unidades básicas de projeto

- **Entity**:
  - Utilizado para definir a visão externa do modelo (símbolo)
- **Architecture**:
  - Utilizado para definir a função do modelo (esquemático)
- **Configuration**:
  - Utilizado para associar uma **Architecture** com uma **Entity**
- **Package**:
  - Conjunto de informações que pode ser referenciado por modelos VHDL (**Library**)
  - Consiste em duas partes: declaração e corpo

Declaração **Entity**

```
entity <nome> is
  Generic declarations
  Port declarations
end entity <nome>;
```

- Analogia: símbolo do componente
  - <nome> pode ser qualquer nome alfanumérico
- **Port declarations**:
  - Utilizado para descrever as entradas e saídas (pinos)
- **Generic declarations**:
  - Utilizado para passar informações ao modelo
- Fechamento (end entity):
  - **end entity <nome>;** – VHDL 93 e posterior
  - **end entity;** – VHDL 93 e posterior
  - **end;** – Todas as versões do VHDL

Entity: Declaração Port

```
entity <nome> is
  Generic declarations
  Port (
    signal clk   : in bit;
    q            : out bit -- signal é implícito
  );
end entity <nome>;
```

- Estrutura: <classe> nome : <modo> < tipo >;
  - <classe>: o que pode ser feito
  - nome: identificar para posterior referência
  - <modo>: direção:
    - **IN**: entrada
    - **INOUT**: bidirecional
    - **OUT**: saída
    - **BUFFER**: saída com realimentação interna
  - <tipo> tipo e tamanho (detalhado mais tarde)

Entity: Declaração Generic

```
entity <nome> is
  Generic (
    constant tplh, tphl: time := 5 ns;
    tphz, tplz         : time := 3 ns; -- constant é implícito
    default_value      : integer := 1;
    cnt_dir            : string := "up"
  );
  Port declarations
end entity <nome>;
```

- Valores **Generic** podem ser sobrescritos durante a compilação com passagem de parâmetros (parecido com macros em C)
- **Generic** devem ser sempre **constantes** durante a compilação

Declaração **Architecture**

- Analogia: esquemático interno do componente
  - Descreve a funcionalidade e a temporização de um modelo
- **Deve** estar associada a uma **Entity**
- **Architectures** executam concorrentemente
- Estilos:
  - Comportamental (como opera):
    - RTL: descritos em termos de registadores
    - Funcional: sem temporização
  - Estrutural: *netlist*
    - Baixo nível (*Gate/component*)
  - Híbrido: estilos comportamental e estrutural

Declaração **Architecture** (cont.)

```
architecture <identificador> of <nome_entity> is
  -- secção de declaração (exemplos)
  signal val     : integer;
  signal temp    : integer := 1;    -- sinais com valor padrão
  constant load : boolean := true; -- constantes
  -- declação de tipos (mais depois)
  -- declaração de componentes (mais depois)
  -- declaração de subprogramas (mais depois)
  -- corpo de subprogramas (mais depois)
  -- declaração de subtipos
  -- declaração de atributos
  -- especificação de atributos
begin
  -- implementação de processos
  -- chamadas concorrentes de procedimentos
  -- atribuição concorrentes de sinais
  -- instâncias de componentes
  -- implementação de "generates"
end architecture <identificador>;
```

VHDL: modelo básico

```
entity <nome_entity> is
  generic (...);
  port (...);
end entity <nome_entity>;
```

```
architecture <nome_arch> of <nome_entity> is
  -- sinais internos
  -- tipos enumerados
  -- componentes
begin
  -- atribuição de sinais
  -- processos
  -- instâncias de componentes
end architecture <nome_arch>;
```

Declaração **Configuration**

```
configuration <identificador> of <nome_entity> is
  for <nome_architecture>

  end for;
end configuration <identificador>;
-- end;  -- (1076-1987 version)
```

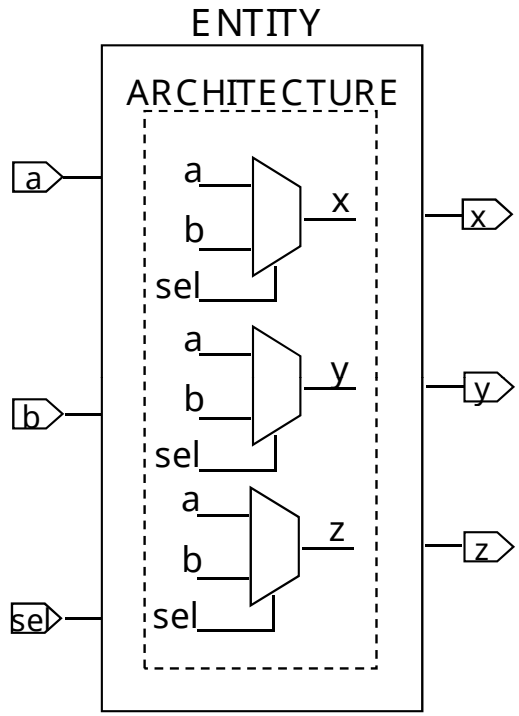
- Faz associações entre modelos:
  - Associa um **entity** com uma **architecture**
  - Associa um componente a um **entity-architecture**
- Usos:
  - Em simulações, permite executar diferentes vetores de estímulos
  - Promove diferentes configurações em projetos para tecnologias diferentes:
    - Modelo comportamental e modelo sintetizável
    - Modelo para FPGA, CPLD ou ASIC

Estrutura de construção básica

```
entity cmpl_sig is
  port (
    a, b, sel : in bit;
    x, y, z   : out bit );
end entity cmpl_sig;
```

```
architecture logic of cmpl_sig is
begin
  x <= (a and not sel) or (b and sel);
  y <= a when sel='0' else b;

  with sel select
  z <= a when '0',
       b when '1',
       '0' when others;
end architecture logic;
```

```
configuration cmpl_sig_conf of cmpl_sig is
  for logic

  end for;
```



Tópico

Pacotes – **Packages**

- Pacotes são uma forma conveniente de armazenar e utilizar informações dentro de um modelo

- Consistem em:
  - Declaração (obrigatório)
    - Declaração de tipos
    - Declaração de subprogramas
  - Corpo (opcional)
    - Definição de subprogramas

- VHDL possui dois pacotes inclusos:
  - **STANDARD**
  - **TEXTIO**

Pacotes – **Packages**

```
package <nome_package> is
  -- constantes
  -- tipos
  -- sinais
  -- suprogramas
  -- componentes
  -- (...)
end package <nome_package>;
```

```
package body <nome_package> is
  -- constantes
  -- tipos
  -- subprogramas
end package body <nome_package>;
```

Exemplo de **Package**

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

```
package filt_cmp is
  type state_type is (idle, tap1, tap2, tap3, tap4);

  component acc
    port (
      xh          : in std_logic_vector (10 downto 0);
      clk, first  : in std_logic;
      yn          : out std_logic_vector (11 downto 4)
    );
  end component;

  function compare (signal a , b : integer) return boolean;

end package filt_cmp;
```

```
package body filt_cmp is
  function compare (signal a , b : integer) return boolean is
    variable temp : boolean;
  begin
    if a < b then
      temp := true;
    else
      temp := false;
    end if;
```

Bibliotecas – **Libraries**

- Uma **Library** é um **diretório** que contém um ou uma coleção de pacotes

- Há dois tipos de **Libraries**:

  - *Working Library*
    - Diretório corrente do projeto

  - *Resource Library*
    - Pacote **STANDARD**
    - Pacotes desenvolvidos pela **IEEE**
    - Pacotes desenvolvidos pelo fabricante (Altera, Xilinx)
    - Ou outra biblioteca de modelos referenciadas pelo projeto

Referenciando modelos em bibliotecas e pacotes

- Todos os pacotes devem ser compilados antes de referenciados
- Bibliotecas implícitas:
  - **WORK**
  - **STD**
- Duas cláusulas são necessárias para referenciar um pacote:
  - Cláusula **Library**
    - Define o nome da biblioteca que será referenciada
    - É um nome simbólico do caminho/diretório
    - Definido pelo compilador
  - Cláusula **use**
    - Especifica o pacote e o objeto da biblioteca que foi especifica pela cláusula *library*

Exemplo

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

```
entity cmpl_sig is
  port (
    a, b, sel : in bit;
    x, y, z   : out bit );
end entity cmpl_sig;
```

```
architecture logic of cmpl_sig
    is
  begin
    x <= (a and not sel) or (b
        and sel);
    y <= a when sel='0' else b;

    with sel select
      z <= a when '0',
      b when '1',
      '0' when others;
```

- **libray** <nome>, <nome>;
  - <nome> é simbólico e definido pelo compilador
  - **work** e **std** não precisam ser referenciados
- **use** lib_name. pack_name.object;
  - **all** é uma palavra reservada para o objeto <nome>
- Colocando as cláusulas libray/use permite-se acesso ao os módulos seguintes

Biblioteca STD

- Contém os seguintes pacotes:
  - **standard**
    - Tipos pré-definidos
    - Funções operadoras para dar suporte tipos pré-definidos
  - **textio**
    - Operação em arquivos
    - **Útil para simulação**
  - **STD** é implícita, logo não é necessário referenciá-la no projeto VHDL

Tipos definidos no pacote **standard**

- Tipo **BIT**:
  - Dois valores lógicos: (’0’, ’1’)

```
signal a_temp : bit;  -- único bit
-- vetores
signal temp : bit_vector(3 downto 0);
signal temp : bit_vector(0 to 3);
```

- Tipo **BOOLEAN**
  - (false, true)
- Tipo **INTEGER**
  - Valores positivos e negativos em **decimal**

```
-- número de 32 bits
signal int_tmp : integer;
-- número de 8 bits
signal int_tmp1 : integer range 0 to 255;
```

Tipos definidos no pacote **standard** (cont.)

- Tipo **NATURAL**
  - Inteiros de 0 a 2^{32}
- Tipo **POSITIVE**
  - Inteiros de 1 a 2^{32}
- Tipo **CHARACTER**
  - Caracteres ASCII
- Tipo **STRING**
  - Array de caracteres
- Tipo **TIME**
  - Tempo (ex.: ps, us, ns, ms, sec, min, hr)
- Tipo **REAL**
  - Precisão dupla em ponto flutuante

Biblioteca IEEE

- Contém os seguintes pacotes:
  - **STD_LOGIC_1164**
    - Tipos e funções **STD_LOGIC**
  - **NUMERIC_STD**
    - *Unsigned arithmetic functions using standard logic vectors defined as SIGNED and UNSIGNED data type*
  - **STD_LOGIC_ARITH**
    - *Arithmetic functions using standard logic vectors as SIGNED or UNSIGNED*
  - **STD_LOGIC_SIGNED**
    - *Signed arithmetic functions directly using standard logic vectors*
  - **STD_LOGIC_UNSIGNED**
    - *Unsigned arithmetic functions directly using standard logic vectors*
  - **STD_LOGIC_TEXTIO**
    - Operações em arquivos utilizando **std_logic**

Tipos definidos no pacote **std_logic_1164**

- Tipo **STD_LOGIC**:
  - Nove valores lógicos: (‘U’, ‘X’, ‘0’, ‘1’, ‘Z’, ‘W’, ‘L’, ‘H’, ‘-’)
    - ’1’: Nível alto
    - ’0’: Nível baixo
    - ’X’: Desconhecido
    - ’Z’: alta-impedância (tri-state)
    - ’-’: *Don’t care*
    - ’H’: Nível alto fraco
    - ’L’: Nível baixo fraco
    - ’W’: Desconhecido fraco
  - Tipo resolvido: suporta sinais com múltiplas fontes
- Tipo **STD_ULOGIC**:
  - Mesmo valores que **STD_LOGIC**
  - Tipo não resolvido: sinais com múltiplas fontes resulta em erro

Pacotes/Bibliotecas definidos pelo usuário

- Utilizando o mesmo diretório do projeto:

```
library work; --opcional
use work.<nome do pacote>.all;
```

- Pacotes em diretórios diferentes:
  - Nome da biblioteca deve ser mapeado para o diretório nas configurações do compilador

```
library <qualquer_nome>;
use <qualquer_nome>.<nome do pacote>.all;
```

Tópico

Referências

- *Introduction to Altera Devices and Design Software*

- PEDRONI, Volnei A. Eletrônica digital moderna e VHDL. Rio de Janeiro: Elsevier, 2010. 619 p., ISBN 9788535234657.