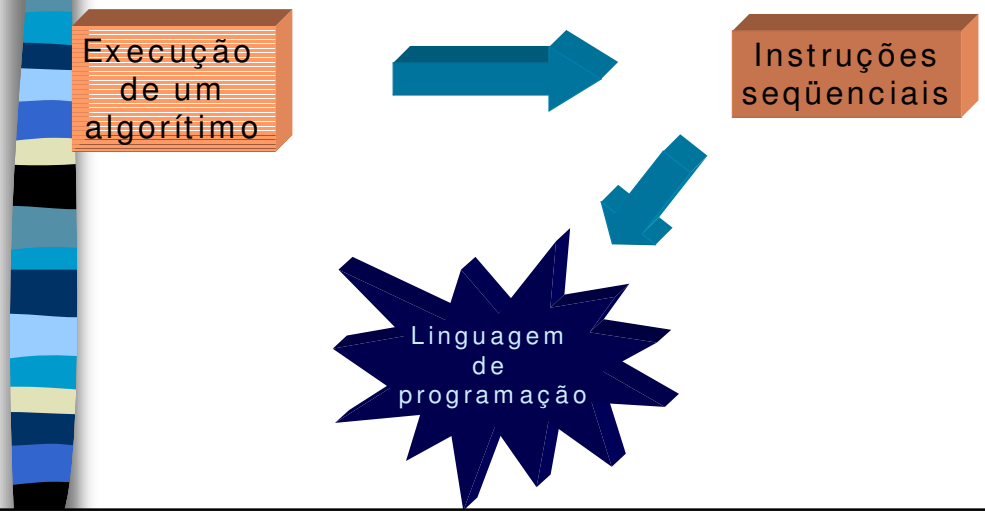
VHDL

(Very High Speed Integrated Circuit HDL (VHSIC HDL))

CIn-UFPE

GRECO

Objetivo das Linguagens

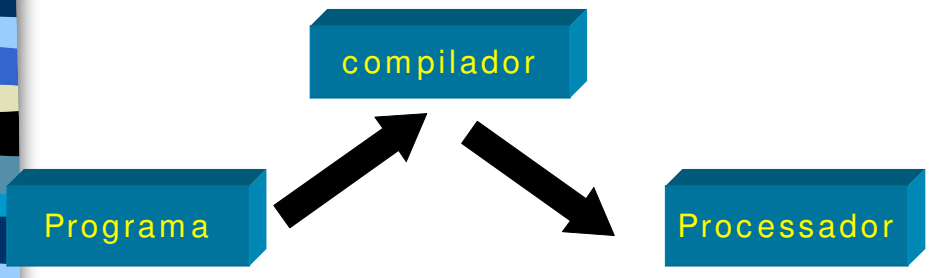


Objetivo das Linguagens

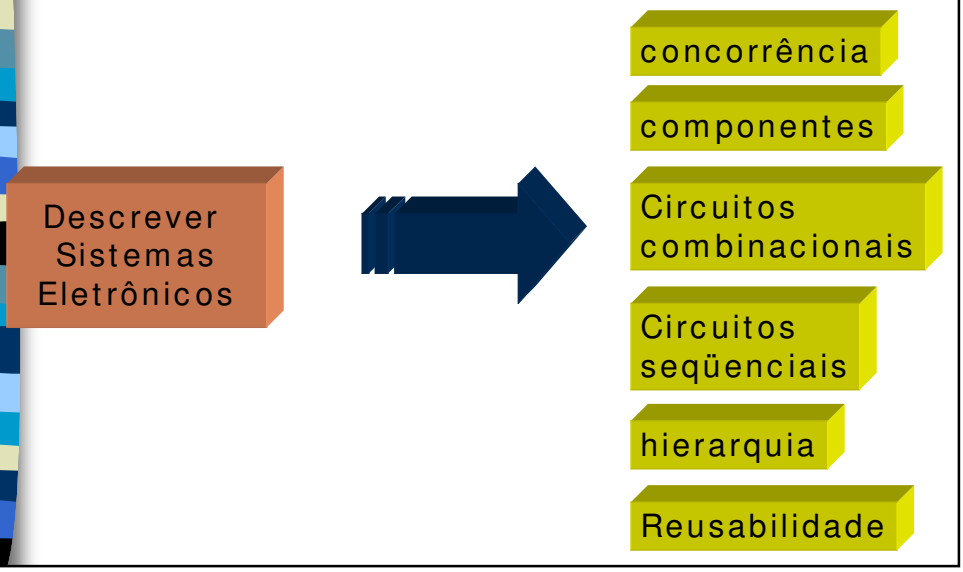
- Características Gerais
  - linguagem procedural
    - estruturada em funções e procedimentos

```
int soma (int a, int b) {
        return (a + b);
        }
```

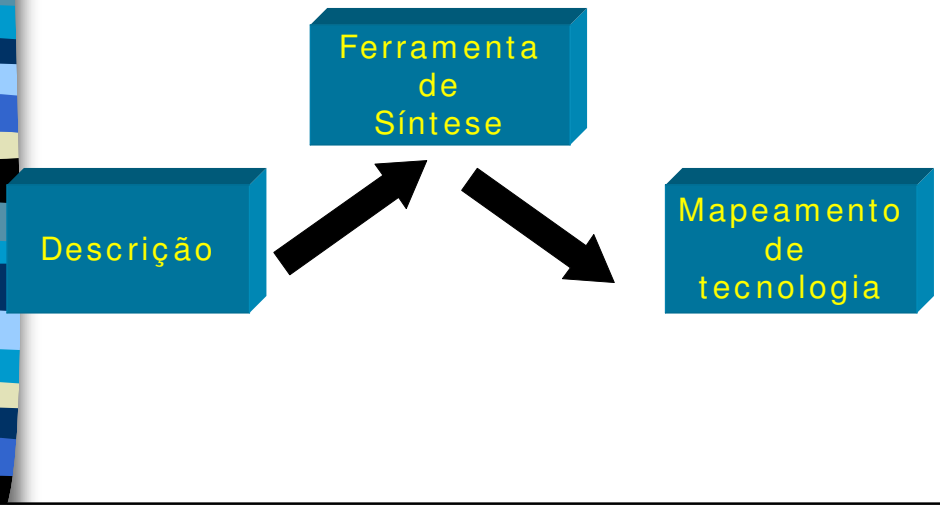
Linguagem de Programação (HLL)



Objetivo das HDL's



Linguagem de Descrição de Hardware (HDL)

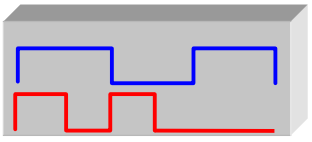


HDL's

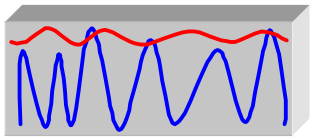
- Algumas Linguagens de Descrição de Hardware
  - VHDL
  - Verilog (indústria americana, Motorola, ...)
  - HardwareC
  - Handel-C
  - C++
  - Java
  - etc.

Introdução a VHDL

- **VHDL - Very High Speed Integrated Circuit HDL (VHSIC HDL)**
  - *É uma linguagem para descrição de hardware que permite ao usuário descrever circuitos de maneira estrutural (RTL) ou comportamental.*

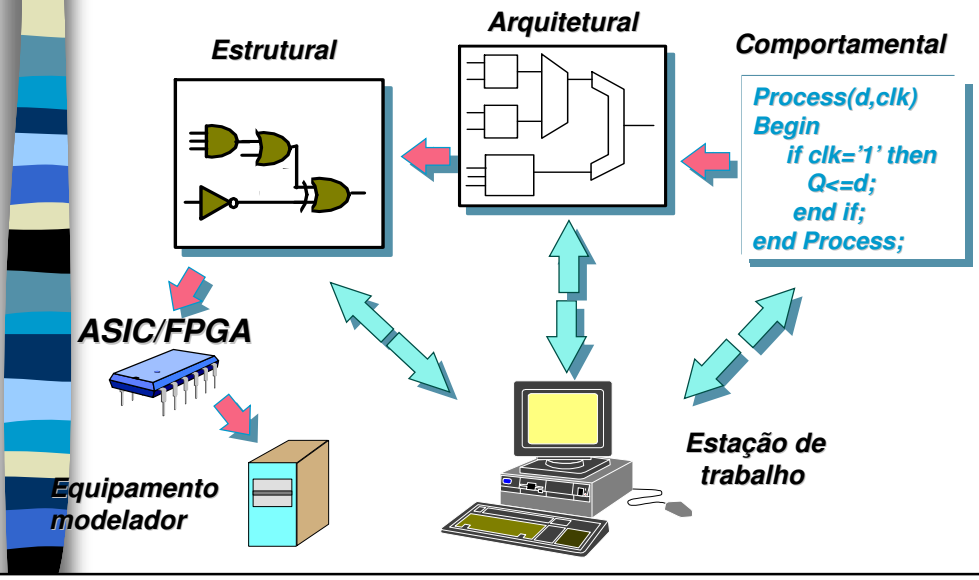


Circuitos digitais

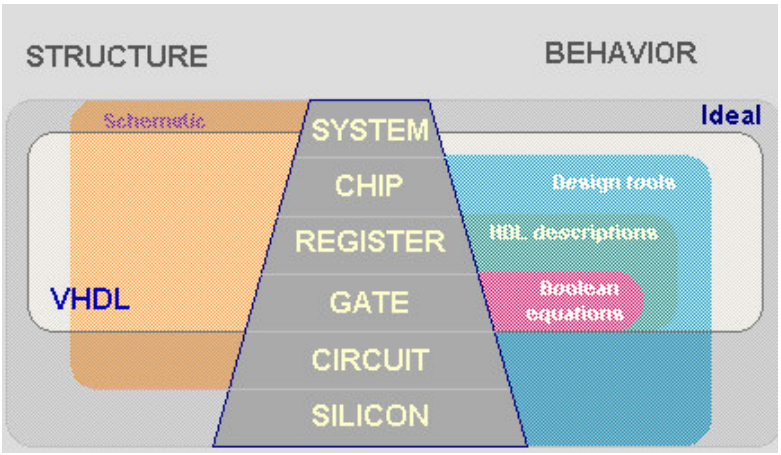


Circuitos Analógicos (AVHDL)

Modelamento de Circuitos Integrados Digitais

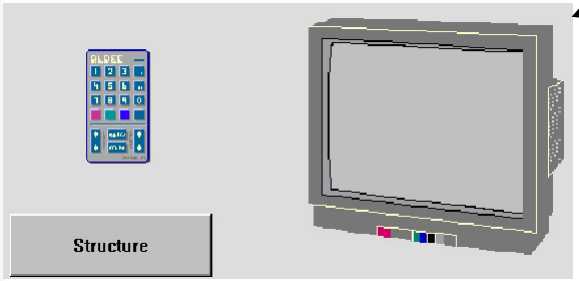


Níveis de especificação



Exemplo

O que ?



Como?

```
entity TVset is
    port  (

          );
end entity TVset;
```

```
architecture TV2000 of TVset is

...

end architecture TV2000;
```

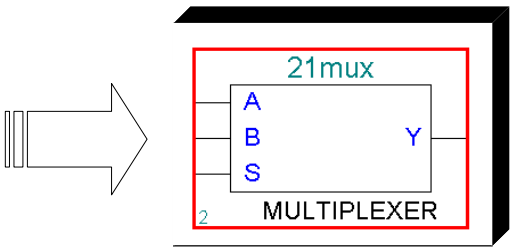
Estrutura de VHDL

- **Componentes de Projeto**
  - Entidades (entities)
  - Arquiteturas (architectures)
  - Configurações (configurations) (opcional)
  - Bibliotecas de funções/ procedimentos (Packages) (opcional)

Entidade

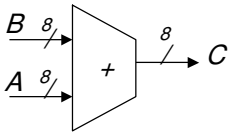
- **Descrição VHDL especifica:**
  - **Entity**
    - **Fornece uma visão externa do circuito. Equivale a um símbolo no modelo esquemático.**
      - **nome**
      - **interface**

```
Entity 21mux is
port (A : in bit;
      B : in bit;
      S : in bit;
      Y : out bit
     );
end 21mux;
```



Introdução a VHDL

entity *COMPARE* ***is***
port *(A,B: in bit;*
C: out bit);
end *COMPARE ;*



entity *SOMADOR* ***is***
port *(A,B: in bit_vector(7 downto 0);*
C: out bit_vector(7 downto 0));
end *SOMADOR;*

Arquitetura

- Architecture
  - Define os componentes ou o comportamento da entidade (entity) e suas conexões.
  - Architecture está sempre conectada a uma entidade.
  - A arquitetura pode conter componentes, Lógica Combinacional ou Seqüencial.

Tipos de Descrição em VHDL

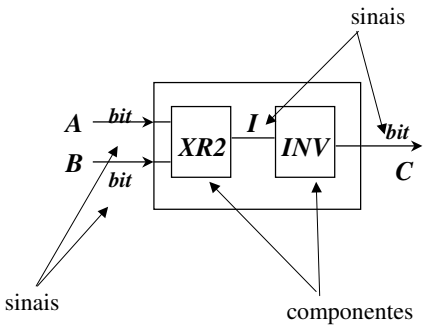
Três tipos de descrição, ou arquiteturas, podem ser utilizados em VHDL

Estrutural	Comportamental Data Flow	Comportamental Processos

VHDL - Estrutural

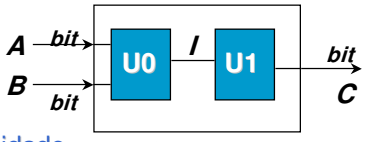
- Sinais:
  - interconexão entre componentes
- Componentes:
  - devem ter sido especificados anteriormente (hierarquia)
- Uso dos componentes:
  - Podem ser usados mais de uma vez *

*REUSABILIDADE de COMPONENTES

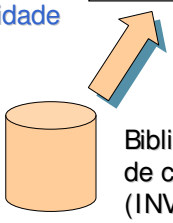


VHDL - Estrutural

```
entity COMPARE is
port (A,B: in bit;
      C: out bit);
end COMPARE;
architecture STRUCT of COMPARE is
    signal I : bit;
    component XR2
        port(x,y: in bit; z:out bit);
    end component;
    component INV
       port(x:in bit; z:out bit);
    end component;
begin
    U0: XR2 port map(A,B,I);
    U1: INV port map(I,C);
end STRUCT;
```



Entidade

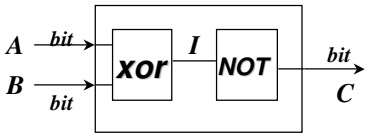


Biblioteca de componentes (INV, XR2, AND ...)

Arquitetura

VHDL - Comportamental(data flow)

- **Data flow (eq. booleanas):**
  - concorrência




```
entity COMPARE is
    port(A, B: in bit; C : out bit);
end COMPARE,
architecture DFLOW of COMPARE is
signal I : bit;
begin
    I <= A XOR B;
    C<= NOT I;
end DFLOW;
```

VHDL - Comportamental (processo)

- **Algoritmo (Processo)**
  - seqüencial
- **Lista de sensibilidade**
  - sinais de entrada
- **Região declarativa**
  - Região entre o fim da lista de sensibilidade e a palavra chave begin.
  - Usada para declarar variáveis ou constantes dentro do processo.
- **Campo de atribuições**
  - Campo entre a chave begin e end ALG;


```
entity COMPARE is
   port(A,B: in bit; C : out bit);
end COMPARE,
architecture ALG of COMPARE is
begin
   process (A,B)
   begin
      if (A = B) then
         C <= '1';
      else
         C <= '0';
      end if;
   end process;
end ALG;
```

Atribuições concorrentes...

- Especificam o comportamento paralelo dos módulos (hardware)
- Concorrência:
  - atribuição de sinal
  - atribuição de sinal condicional/ selecionada
  - processos
  - procedimentos
  - blocos

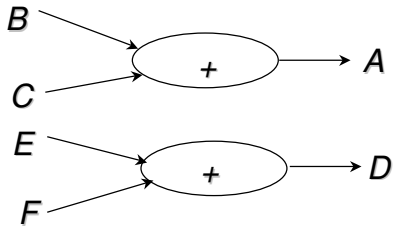
Sinais

- Sinais são elementos que conectam componentes dentro de um circuito.
  - Tipos de sinais
    - escalar
      - signal S: bit := '1';
    - vetor
      - signal S_BUS: bit_vector(0 to 7)
    - Inteiro, Real, ...
  - Localidade
    - globais
      - entity
    - locais
      - architecture

Atribuição de sinal concorrente

- Usada em descrições estruturais e dataflow (circuitos combinacionais)
- Não possuem ordem fixa de execução

```
A <= B + C;
D <= E + F
```



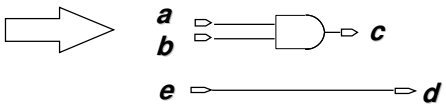
Atribuição de sinais

Exemplo: (arquivo simpsig.vhd)

- Exemplo

```
ENTITY simpsig IS
  PORT
  (
    a, b, e : IN BIT;
       c, d: OUT BIT
  );
END simpsig;
ARCHITECTURE struct OF simpsig IS
BEGIN
  c <= a AND B;
  d <= e;
END struct;
```

Equivalente Gráfico (simpsig.gdf)



Atribuição de sinal condicional

- **Sintaxe:**
  - **target <= {expression when condition else} expression**
- **Só uma expressão será atribuída**

```
Z <= A when (x > 3) else
     B when (x < 3) else
     C
```

- *Não pode ser usada em processos*

Atribuição de sinal selecionada

- **Sintaxe:**

with expression select
target <= {expression when choices};

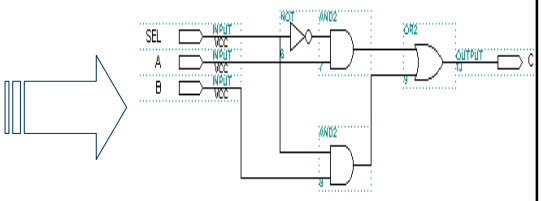
```
with MYSIG select
   Z <= A when 15,
        B when 22,
        C when others;
```

- Equivale ao CASE
- Não pode ser usado em processos

Atribuição de sinais condicionais

- Exemplo de um multiplexador 2:1

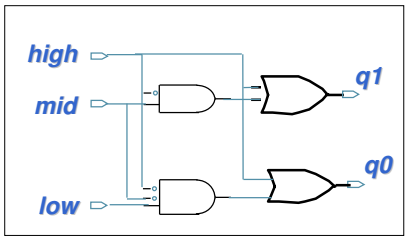
```
ENTITY condsig IS
  PORT
  (
    A, B, sel  : IN BIT;
    C : OUT BIT
  );
END condsig;
ARCHITECTURE maxpld of condsig IS
BEGIN
  C <= A when sel = '0' ELSE B;
END maxpld;
```



Atribuição de sinais condicionais

Exemplo

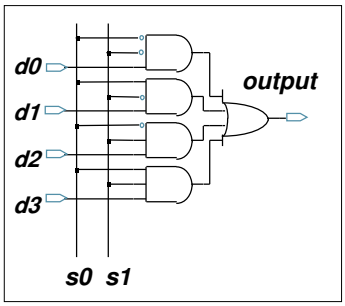
```
ENTITY condsigm IS
   PORT
   (
     high, mid, low : IN BIT;
     q : OUT INTEGER
   );
END condsigm;
ARCHITECTURE maxpld of condsigm IS
BEGIN
   q <= 3 WHEN high = '1' ELSE  - - high
        2 WHEN mid  = '1' ELSE  - - mid e não high
        1 WHEM low  = '1' ELSE  - - low  e não mid ou high
          0;
END maxpld;
```



Atribuição de sinais condicionais

- *Exemplo (Multiplexador 4:1)*

```
ENTITY selsig IS
   PORT
   (
     do, d1, d2, d3  : IN BIT;
     s : IN INTEGER RANGE 0 to 3;
     output : OUT BIT
   );
END selsig;
ARCHITECTURE maxpld of selsig IS
BEGIN
   WITH  S SELECT-- cria um multiplexador
     output <=  d0 WHEN 0,
                d1 WHEN 1,
                d2 WHEN 2,
               d3 WHEN 3;
END maxpld;
```



Construtores seqüenciais

- if-then-else

```
if condition then sequential_statements
[elsif condition then sequential_statements]
[else condition then sequential_statements]
end if;
```

- case

```
case expression is
   when choice1 = >  sequential_statements
   ...
   when choice n = > sequential_statements
   [when others = >  sequential_statements]
end case;
```

Processos - Lógica Combinacional

- Declarações em processos incluem um conjunto de declarações seqüenciais que atribuem valores à sinais. Estas declarações permitem a execução passo-a-passo da computação. Declarações em processos que descrevem apenas comportamento combinacional podem ser usados para criar lógica combinacional.
- Para assegurar que um processo é combinacional, sua "sensitive list" deve conter todos os sinais que são lidos no processo.
- "Uma sensitive list contém todos os sinais que causam as declarações do processo serem executados se seus valores mudam".

Processo: estrutura

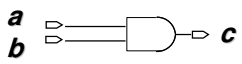
- **Algoritmo (Processo)**
  - seqüencial
- **Lista de sensibilidade**
  - sinais de entrada
- **Região declarativa**
  - Região entre o fim da lista de sensibilidade e a palavra chave begin.
  - Usada para declarar variáveis ou constantes dentro do processo.
- **Campo de atribuições**
  - Campo entre a chave begin e end ALG;


```
architecture ALG of COMPARE is
begin
  process (A,B)
  begin
    if (A = B) then
      C <= '1';
    else
      C <= '0';
    end if;
  end process;
end ALG;
```

Processo - Lógica combinacional

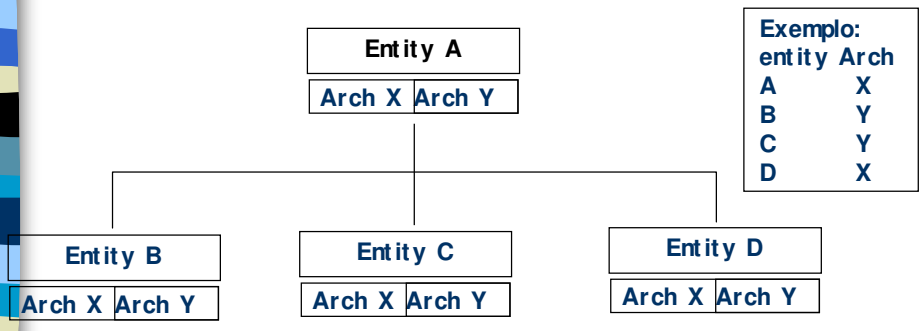
Exemplo

```
ENTITY proc IS -- Porta AND
   PORT
   (
   a,b  : IN BIT;
   c : OUT  BIT
   );
END proc;
ARCHITECTURE maxpld  OF proc IS
BEGIN
   PROCESS(a, b) - - Processo
    BEGIN
        q <= a AND b;
    END PROCESS;
END maxpld;
```



Configuração

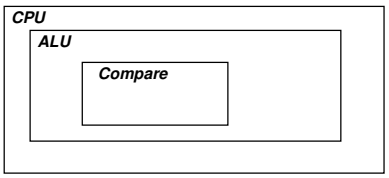
- **É uma lista de "componentes" que especifica qual arquitetura deve ser usada para cada entidade em um determinada projeto.**



Exemplo:	
entity	Arch
A	X
B	Y
C	Y
D	X

Escolhendo uma arquitetura...

- **Configurações**
  - **permite a seleção de diversas arquiteturas em diferentes níveis de hierarquia.**




```
configuration fast_one of ALU is
for first
for U0: COMPARE use entity WORK.COMPARE(dataflow);
```

```
entity ALU is
   port( opcode:...
end ALU;

architecture first of ALU is
  component COMPARE
  port (a,b:in bit; c:out : bit);
  end component;
  ...
begin
  U0: COMPARE port map (s,d,q);
  ....
end first
```

Package

- **Parte Declarativa**
  - **Declaração de constantes**
  - **Declaração de subprogramas**
  - **Declarações de tipos**
  - **Declaração de componentes**
- Corpo do Pacote
  - corpo de funções
  - corpo de procedimentos

Package

- É uma unidade que agrupa várias declarações, as quais, podem ser compartilhadas entre vários projetos.
- Packages são em geral armazenas em bibliotecas para maior conveniência.
- Packages são usadas para declarar itens compartilhados, tais como: types, subtypes, components, signals, files, aliases, attributes e groups.

Exemplo Package

Parte Declarativa

```
Package my_defs is
Constant my_value: integer := 10;
function my_not (a : boolean) return boolean;
end my_defs;
```

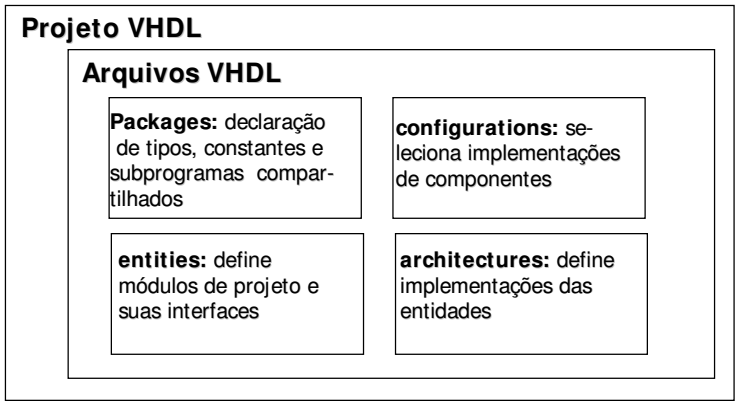
Corpo do Pacote

```
Package body my_defs is
function my_not (a : boolean) return boolean is
begin
  return not a;
end;
end my_defs;
```

Packages - MAX+PLUS II

Arquivo	*Package*	*Biblioteca*	*Conteúdo*
Maxplus2.vhd	maxplus2	altera	Max+plus II primitivas, mega e macrofunções (VHDL)
megacore.vhd	megacore	altera	MAX+PLUS II Megacore VHDL
sd1164.vhd std1164b.vhd	std_logic_1164	ieee	STD_LOGIC e STD_LOGIC_VECTOR
lpm_pack.vhd	lpm_components	lpm	Funções LPM (VHDL)
arith.vhd arithb.vhd	std_logic_arith	ieee	tipos SIGNED e UNSIGNED, comparações, conversões
signed.vhd signedb.vhd	std_logic_signed	ieee	Funções que permite uso de STD_LOGIC_VECTOR como tipos SIGNED
unsigned.vhd unsignedb.vhd	std_logic_unsigned	ieee	STD_LOGIC_VECTOR como tipos UNSIGNED

Resumindo...



VHDL - Operadores

Table 6.1 VHDL Operators.

VHDL Operator	Operation
+	Addition
-	Subtraction
*	Multiplication*
/	Division*
MOD	Modulus*
REM	Remainder*
&	Concatenation – used to combine bits
SLL**	logical shift left
SRL**	logical shift right
SLA**	arithmetic shift left
SRA**	arithmetic shift right
ROL**	rotate left
ROR**	rotate right
=	equality
/=	Inequality
<	less than
<=	less that or equal
>	greater than
>=	greater than or equal
NOT	logical NOT
AND	logical AND
OR	logical OR
NAND	logical NAND
NOR	logical NOR
XOR	logical XOR
XNOR*	logical XNOR

*Not supported in many VHDL synthesis tools. In the MAX+PLUS II tools, only multiply and divide by powers of two (shifts) are supported. Mod and Rem are not supported in MAX+PLUS II. Efficient design of multiply or divide hardware typically requires the user to specify the arithmetic algorithm and design in VHDL.

** Supported only in 1076-1993 VHDL.

VHDL

- Referências
  - Manual VHDL da Altera
  - A Guide to VHDL, Stanley Mazor, Patricia Langstrat
  - VHDL by Douglas, Perry
  - Vhdl Made Easy!
    by David Pellerin, Douglas Taylor