Universidade de Brasília
Departamento de Ciência da Computação

VHDL

Linguagem de Descrição de Hardware (HDL)

Organização e Arquitetura de Computadores

Sumário

- Introdução
- Estrutura de um programa VHDL
- Tipos de Dados em VHDL
- Comandos sequenciais
- *Functions e Procedures*
- Comandos concorrentes

Recursos

- **The Designer's Guide to VHDL**
  Peter Ashenden
  Morgan Kaufmann Publishers, Inc.
  ISBN 1-55860-270-4
- **Circuit Design with VHDL**
  Volnei A. Pedroni
  The MIT Press
  ISBN 0-262-16224-5
- **VHDL Tutorial**
  Ashenden, Peter.
  Moodle

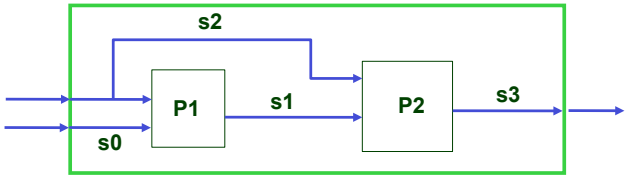
Simulação e Síntese:
Quartus II - Altera
Modelsim - Altera

Introdução

- VHDL:
  - VHSIC Hardware Description Language
    - VHSIC: Very High Speed Integrated Circuits
- Origem:
  - Departamento de Defesa EUA
    - desenvolvida entre anos 70 e 80
    - descrever e modelar circuitos complexos de forma padronizada
    - voltada inicialmente para simulação de circuitos

VHDL vs Linguagens de Programação

- VHDL provê mecanismos para modelar a concorrência e sincronização que ocorrem a nível físico no hardware
- Comunicação entre processos paralelos

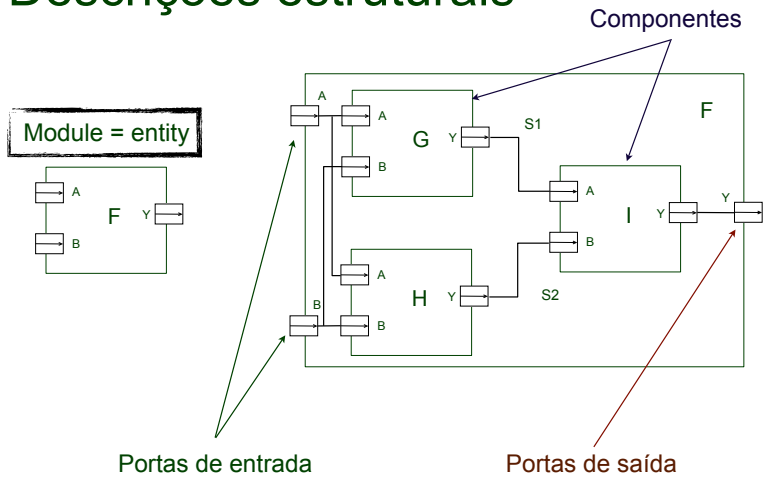


- *Processos P1 e P2 executam em paralelo (podem ser simplesmente duas portas lógicas, como dois módulos arbitrariamente complexos), e se sincronizam através de sinais*

VHDL vs Linguagens de Programação

- Atraso dos componentes
  - A <= B + C after 5.0 ns;
  - D <= A + E; -- D recebe o valor antigo de A !!
- Temporização
  - x <= y;
  - y <= x;
  - wait on clock;
  - Variáveis: sem temporização – linguagem de programação
  - Sinais: temporizados
- Código é executado em um simulador (ao invés de um compilador), não há um código executável.

Descrições estruturais



Descrições estruturais

```
entity contador is
        generic(prop_delay : Time := 10 ns);
        port( clock : in bit;  q1, q0 : out bit);
end contador;

architecture estrutural of contador is
        component Tflip_flop
                port( ck: in bit; q: out bit);
        end component;
        component inversor
                port( a: in bit; y: out bit);
        end component;
        signal ff0, ff1, inv_ff0 : bit;
begin
        bit_0: Tflip_flop port map( ck=> clock, q => ff0);
        inv: inversor port map( a=> ff0, y => inv_ff0);
        bit_1: Tflip_flop port map( ck=> inv_ff0, q => ff1);
        q0 <= ff0;
        q1 <= ff1;
end estrutural;
```

Descrição Estrutural do Módulo

Simulação

- **Circuito de teste: test_bench**
  - Contém um processo "gerador de teste" e uma instância do projeto
  - O test_bench **não** contém portas de entrada/saída

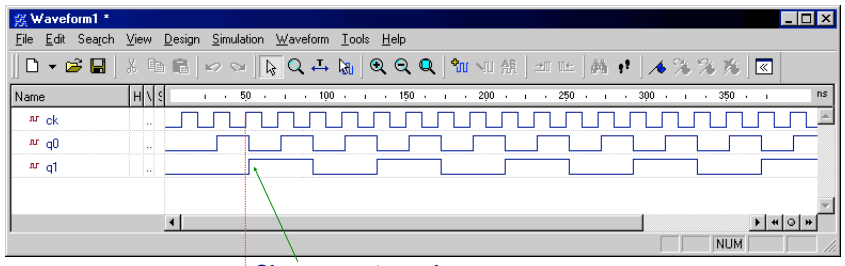
Simulação - *test bench*

```
entity test_bench is
end test_bench;

architecture estrutural of test_bench is
    component contador
        port( clock : in bit;  q1, q0 : out bit );
    end component;
    signal ck, q1, q0 : bit;
begin
    cont1:  contador port map(clock=>ck, q0=>q0, q1=>q1);
    process
        begin
            ck <= '1' after 10ns, '0' after 20ns;
            wait for 20ns;
    end process;
end estrutural;
```


Simulação - visualização

- **Resultado da simulação:**



Observar o atraso de q0 e q1 em relação ao clock

Estrutura de um programa VHDL

Projeto VHDL

Arquivos VHDL

Package: Declara constantes, tipos de dados, subprogramas. Objetivo: reutilização de código

Architecture: define a(s) implementação (ões) do projeto

Entity: declara as interfaces do projeto (pinos de entrada/ saída)

Configuration: declara qual das arquiteturas será utilizada

Estrutura de uma descrição VHDL

Libraries	Declaração e uso de *libraries* ocorre no início da descrição. *Libraries* são conjuntos de *packages*
Entidade	Descreve a interface do módulo, seus sinais de entrada e saída
Arquitetura	Descreve a implementação do módulo

Exemplo

library ieee; **use** ieee.std_logic_1164.all; **use** ieee.numeric_std.all;	Declaração e uso das libraries
entity signed_adder **is** **generic** (SIZE: **natural** := 8); -- opcional **port** (a: **in signed** ((SIZE-1) **downto** 0); b: **in signed** ((SIZE-1) **downto** 0); result: **out signed** ((SIZE-1) **downto** 0)); **end** entity;	Declaração da entidade signed_adder e suas portas de entrada e saída com largura parametrizada por SIZE

architecture rtl **of** signed_adder **is**
begin
result <= a + b;
end rtl;
	Arquitetura da entidade signed_adder

VHDL Packages

- std_logic_1164
  - define tipos de dados e operações: std_logic, std_logic_vector, std_ulogic...
- std_logic_signed
  - processamento de std_logic como inteiros
- std_logic_unsigned
  - processamento de std_logic como naturais
- std_logic_arith
  - define tipos unsigned e signed e funções aritméticas

Entity

ENTITY <entity_name> IS
GENERIC (<constant_id> : <const type> := <valor>; <constant_id> : <const type> := <valor>; …);
PORT (<port_id> : <modo> <tipo>; <port_id> : <modo> <tipo>; …);
END <entity_name>;

Arquitetura

Architecture
Declarações
signal - sinais que comunicam-se entre processos concorrentes sinais que comunicam-se entre processos concorrentes e os pinos de E/S
type - novos tipos
constant - constantes
component - componentes (para descrever estrutura)
function - Subprogramas (*apenas a declaração destes)*
Begin (declarações concorrentes)
Blocos: conjunto de declarações concorrentes (encapsulamento)
Atribuição a sinais
Chamadas a "functions" e a "procedures"
Instanciação de Componentes
Processos: descrição de algoritmo
End

Package

- Permite a reutilização de código já escrito.
- Armazena:
  - Declaração de subprogramas
  - Declaração de tipos
  - Declaração de constantes
  - Declaração de arquivos
  - Declaração de "alias" (sinônimos, por exemplo, para mneumônicos)

```
package minhas_definições is
        function max(L, R: INTEGER) return INTEGER;
        type UNSIGNED is array (NATURAL range <>) of STD_ULOGIC;
        constant unit_delay :  time := 1ns;
        file outfile :Text is Out "SIMOUT.DAT";
        alias C : Std_Ulogic is grayff (2);
end minhas_definições
```


Package

- Um *package* pode ser dividido em duas partes: declaração e corpo.
- Corpo: opcional, detalha especificações incompletas na definição.
- Exemplo completo:

```
package data_types is
    subtype address is bit_vector(24 downto 0);
    subtype data is bit_vector(15 downto 0);
    constant vector_table_loc : address;
    function data_to_int(value : data) return integer;
    function int_to_data(value : integer) return data;
end data_types;
package body data_types is
    constant vector_table_loc : address := X"FFFF00";
    function data_to_int(value : data) return integer is
        body of data_to_int
    end data_to_int;
    function int_to_data(value : integer) return data is
        body of int_to_data
    end int_to_data;
end data_types;
```


Package

- Utilização do *package* no programa que contém o projeto:
  - Via utilização do prefixo do package
    - variable PC : data_types.address;
    - int_vector_loc := data_types.vector_table_loc + 4*int_level;
    - offset := data_types.data_to_int(offset_reg);
  - Via declaração, antes da iniciar a seção "entity", indicação para utilizar todos os tipos declarados em determinado "package"
    - use data_types.all;
- Praticamente todos os módulos escritos em VHDL iniciam com:

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

  - utilizar a biblioteca IEEE, que contém a definição de funções básicas, subtipos, constantes; e todas as definições dos packages incluídos nesta biblioteca.

Tipos de Dados em VHDL

- VHDL é uma linguagem fortemente “tipada” (integer 1 ≠ real 1.0 ≠ bit ‘1’)
  - auxilia para detectar erros no início do projeto
  - exemplo: conectar um barramento de 4 bits a um barramento de 8 bits
- Tópicos
  - Escalares
  - Objetos
  - Expressões

Escalares

- Escalar define um tipo indivisível
  - character / bit / boolean / real / integer / physical_unit
  - std_logic (IEEE)
- Bit
  - Assume valores ‘0’ e ‘1’
  - Declaração explícita: bit’(‘1’), pois neste caso ‘1’ também pode ser ‘character’.
  - bit não tem relação com o tipo boolean.
  - bit_vector: tipo que designa um conjunto de bits. Exemplo: “001100” ou x”00FF”.
- Boolean
  - Assume valores true e false.
  - Útil apenas para descrições abstratas, onde um sinal só pode assumir dois valores

Escalares

- Real
  - Utilizado durante desenvolvimento da especificação
  - Sempre com o ponto decimal
  - Exemplos: -1.0 / +2.35 / 37.0 / -1.5E+23
- Inteiros
  - Exemplos: +1 / 1232 / -1234
  - NÃO é possível realizar operações lógicas sobre inteiros (realizar conversão)
  - Vendedores provêem versões próprias: signed, natural, unsigned, bit_vector (este tipo permite operações lógicas e aritméticas)

Escalares

- Character
  - VHDL não é “case sensitive”, exceto para caracteres.
  - valor entre apóstrofes: ‘a’, ‘x’, ‘0’, ‘1’, …
  - declaração explícita: character’(‘1’), pois neste caso ‘1’ também pode ser ‘bit’.
  - string: tipo que designa um conjunto de caracteres. Exemplo: “xuxu”.
- Physical
  - Representam uma medida: voltagem, capacitância, tempo
  - Tipos pré-definidos: fs, ps, ns, um, ms, sec, min, hr

Escalares

- Intervalos (range)
  - sintaxe: **range** valor_baixo **to** valor_alto
    range valor_alto **downto** valor_baixo
  - integer **range** 1 **to** 10 e NÃO integer **range** 10 **to** 1
  - real **range** 10.0 **downto** 1.0 e NÃO real **range** 1.0 **downto** 10.0
  - declaração sem range declara todo o intervalo
  - declaração range<> : declaração postergada do intervalo
- Enumerações
  - Conjunto ordenando de nomes ou caracteres.
  - Exemplos:
    - type logic_level is (‘0’, ‘1’, ‘X’, ‘Z’);
    - type octal is (‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’);

Arrays

- coleção de elementos de mesmo tipo
  - type word is array (31 downto 0) of bit;
  - type memory is array (address) of word;
  - type transform is array (1 to 4, 1 to 4) of real;
  - type register_bank is array (byte range 0 to 132) of integer;
- array sem definição de tamanho
  - type vector is array (integer range <>) of real;
- exemplos de arrays pré definidos:
  - type string is array (positive range <>) of character;
  - type bit_vector is array (natural range <>) of bit;
- preenchimento de um array: posicional ou por nome
  - type a is array (1 to 4) of character;
  - posicional: ('f', 'o', 'o', 'd')
  - por nome: (1 => 'f', 3 => 'o', 4 => 'd', 2 => 'o')
  - valores default: ('f', 4 => 'd', others => 'o')

Records

- estruturas semelhantes a “struct” em linguagem C
- coleção de elementos com tipos diferentes

```
type instruction is
  record
    op_code : processor_op;
    address_mode : mode;
    operand1, operand2: integer range 0 to 15;
  end record;
```

- declaração: signal instrução : instruction;
- referência a um campo: instrução.operando1

Objetos

- Objetos podem ser escalares ou vetores (arrays)
- Referência em vetores:
  - vet é o vetor;
  - vet(3) é o elemento 3 no vetor;
  - vet(1 to 4) é um pedaço do vetor.
- Devem obrigatoriamente iniciar por uma letra, depois podem ser seguidos de letras e dígitos (o caracter “_” pode ser utilizado). Não são case sensitive.
- Constantes / Variáveis / Sinais

Constantes

- nome dado a um valor fixo
- consiste de um nome, do tipo, e de um valor (opcional, com declaração posterior)
- sintaxe:
  - `constant identificador : tipo [:=expressão];`
- correto:

```
constant gnd: real := 0.0;
```

- incorreto:

```
gnd := 4.5;
```

- constantes podem ser declaradas em qualquer parte, porém é aconselhável declarar constantes frequentemente utilizadas em um package

Variáveis

- Utilizada em processos, sem temporização. Atribuição imediata.
- Sintaxe:

```
variable var_id : tipo [restrição][:=expressão];
```

- Exemplos

```
variable  índice : integer range 1 to 50 := 50;
variable  ciclo : time range 10 ns to 50 ns := 10ns;
variable  memória : bit_vector (0 to 7)
variable  x, y : integer;
```


Sinais

- Comunicação entre módulos.
- Temporizados.
- Podem ser declarados em *entity*, *architecture* ou em *package*.
- Não podem ser declarados em processos, podendo serem utilizados no interior destes.
- sintaxe:

  signal identificador : tipo [restrição] [:=expressão];
- exemplo
  - `signal cont : integer range 50 downto 1;`
  - `signal ground : bit := '0';`
  - `signal bus : bit_vector;`

Expressões

- Expressões são fórmulas que realizam operações sobre objetos de mesmo tipo.
  - Operações lógicas: and, or, nand, nor, xor, not
  - Operações relacionais: =, /=, <, <=, >, >=
  - Operações aritméticas: - (unária), abs
  - Operações aritméticas: +, -
  - Operações aritméticas: *, /
  - Operações aritméticas: mod, rem, **
  - Concatenação

Menor

PRIORIDADE

Maior

- Questão: o que a seguinte linha de VHDL realiza:
  X <= A <= B ?

Expressões

- Observações:
- Operações lógicas são realizadas sobre tipos *bit* e *boolean*.
- Operadores aritméticos trabalham sobre inteiros e reais. Incluindo-se o package da Synopsys, por exemplo, pode-se somar vetores de bits.
- Todo tipo físico pode ser multiplicado/dividido por inteiro ou ponto flutuante.
- Concatenação é aplicável sobre caracteres, strings, bits, vetores de bits e arrays.
  Exemplos:
  - `“ABC” & “xyz” = “ABCxyz”`
  - `“1001” & “0011” = “10010011”`

Comandos sequenciais

- VHDL provê facilidades de paralelismo entre diferentes processos
  e atribuição de sinais.
- Dentro dos processos pode-se especificar um conjunto de ações seqüenciais, executadas passo a passo. É um estilo de descrição semelhante a outras linguagens de programação.
- Comandos:
  - atribuição de variáveis,
  - if,
  - case,
  - for,
  - while,
  - wait

Comandos sequenciais

- Atribuição de variáveis
  - variable_assignment_statement ::= target := expression ;
  - target ::= name | aggregate
- Variáveis não passam valores fora do processo na qual foram declaradas, são locais.
- As atribuições são sequenciais, ou seja, a ordem destas importa.

Comando If

- if_statement ::=
  if condition then
  sequence_of_statements
  { elsif condition then
  sequence_of_statements }
  [else
  sequence_of_statements]
  end if ;
- IMPORTANTE
- teste de borda de subida:
  - `if clock'event and clock='1' then …`
- teste de borda de descida:
  `if clock'event and clock='0' then …`
- a sequência na qual estão definidos os ´ifs´ implica na prioridade das ações.

Exemplo de "if"

- exemplo onde a atribuição à variável T tem maior prioridade:

```
if (x) then T:=A; end if;
if (y) then T:=B; end if;
if (z) then T:=C; end if;
```

```
if (z) then T:=C;
      elseif (y) then T:=B;
      elseif (x) then T:=A;
end if;
```


Exemplo de "if"

- Qual a implementação em hardware da seguinte sequência de comandos ?

```
process(A, B, control)
   begin
    if( control='1')
          then Z <= B;
          else     Z <= A;
    end if;
   end process;
```


CASE

- É utilizado basicamente para decodificação.
- O bloco de controle é um grande case.
  - case_statement ::=
    case expression is
    case_statement_alternative
    { case_statement_alternative }
    end case ;
  - case_statement_alternative ::=
    when choices =>
    sequence_of_statements
  - choices ::= choice { | choice }
  - choice ::=
    simple_expression
    | discrete_range
    | element_simple_name
    | others

CASE

- **case** element_colour **of**
  when red => -- escolha simples
  statements for red;
  when green | blue => -- ou
  statements for green or blue;
  when orange to turquoise => -- intervalo
  statements for these colours;
  end case;

- **case** opcode **of**
  when X"00" => perform_add;
  when X"01" => perform_subtract;
  when others => signal_illegal_opcode;
  end case

CASE

- Qual a implementação em hardware da seguinte seqüência de comandos ?

```
process(A, B, C, D, escolha)
    begin
        case escolha is
            when IS_A => Z<=A;
            when IS_B => Z<=B;
            when IS_C => Z<=C;
            when IS_D => Z<=D;
        end case;
    end process;
```


Loop - FOR

- útil para descrever comportamento / estruturas regulares
- o *for* declara um objeto, o qual é alterado somente durante o laço
- internamente o objeto é tratado como uma constante e não deve ser alterado.

```
for item in 1 to last_item loop
    table(item) := 0;
end loop;
```


Loop - FOR

- next: interrompe a iteração corrente e inicia a próxima

```
outer_loop : loop
    inner_loop : loop
        do_something;
        next outer_loop when temp = 0;
        do_something_else;
    end loop inner_loop;
end loop outer_loop;
```

- exit: termina o laço

```
for i in 1 to max_str_len loop
    a(i) := buf(i);
    exit when buf(i) = NUL;
end loop;
```


Loop - FOR

- Qual a função do laço abaixo ?

```
function conv (byte : word8) return integer is
variable result : integer := 0;
variable k : integer := 1;
begin
    for index in 0 to 7 loop
     if ( std_logic'(byte(index))='1')
        then result := result + k;
     end if;
           k := k * 2;
    end loop;
    return result;
end conv ;
```

- Exercício: faça a conversão ao contrário.

Loop - While

```
while (index < length)
      and (str(index) /= ' ')
loop
     index := index + 1;
end loop;
```


NULL

- serve, por exemplo, para indicar “faça nada” em uma condição de case.

-
```
case controller_command is
  when forward => engage_motor_forward;
  when reverse => engage_motor_reverse;
  when idle => null;
end case;
```


Functions e Procedures

- Simplificam o código, pela codificação de operações muito utilizadas.
- Funções e procedures são declaradas entre a entity e o begin, ou no corpo de um determinado package.
- Utilizam os comandos seqüenciais para a execução do programa
- Procedures: permitem o retorno de vários sinais, pela passagem de parâmetros.
  - mult(A,B, produto);
- Functions: retornam apenas um valor, utilizando o comando return
  - produto <= mult(A,B);

Functions e Procedures

- Exemplo de procedure:

```
procedure mpy (signal a, b : in std_logic_vector (3 downto 0);
       signal prod : out std_ulogic_vector (7 downto 0)) is
variable p0,p1,p2,p3 : std_logic_vector(7 downto 0);-- prod. parciais
constant zero : std_logic_vector := "00000000";
begin
if b(0)='1' then p0 := ("0000" & a); else p0:=zero; end if;
if b(1)='1' then p1 := ("000" & a & '0'); else p1:=zero; end if;
if b(2)='1' then p2 := ("00" & a & "00"); else p2:=zero; end if;
if b(3)='1' then p3 := ('0' & a & "000"); else p3:=zero; end if;
prod <= ( p3 + p2 ) + ( p1 + p0 );
end mpy;
```

```
library IEEE;
use IEEE.Std_Logic_1164.all;

package calcHP is
        subtype ...
        type ...
        constant ...
        procedure somaAB  ( signal A,B: in  regsize; signal S:   out regsize);
end calcHP;

package body calcHP is
        procedure somaAB  ( signal A,B: in  regsize; signal S:   out regsize);
        is
                variable carry : STD_LOGIC;
        begin
                ....
        end procedure somaAB;
end package body calcHP;

library IEEE;
use IEEE.Std_Logic_1164.all;
use work.calcHP.all;

entity calculadora is
                port( clock : in bit;   saida : out regsize; flag :  out std_logic);
end;
architecture rtl of calculadora is
begin
        ...
        somaAB( opA, opB, cin, soma, cout);
        ...
end rtl;
```

A procedure é declarada no package

A procedure é implementada no package body

Significa: utilizar todas as declarações do package calcHP

A procedure é utilizada na architecture

Comandos concorrentes

- PROCESS
  - Conjunto de ações sequenciais
- Wait: suspende o processo, até que as condições nele incluídas sejam verdadeiras:
  - wait [sensitivity_clause] [condition_clause] [timeout_clause] ;
  - sensitivity_clause ::= on signal_name { , signal_name }
  - condition_clause ::= until condition
  - timeout_clause ::= for time_expression
- Exemplo:

```
exemplo : process
begin
  wait until a = '1' and b = '1';
  q <= '1';
  wait until a = '0' and b = '0';
  q <= '0';
end process ;
```

- Não são permitidos componentes dentro de processos.

Comandos concorrentes

- Sensitivity list: caso haja uma lista de sinais no início do processo, isto é equivalente a um *wait* no final do processo.
- Havendo *sensitivity list* no processo, nenhum *wait* é permitido no corpo do processo.

```
process (reset, clock)
    variable state : bit := false;
begin
    if reset then state := false;
    elsif clock = true then state := not state;
    end if;
    q <= state after prop_delay;
    -- implicit wait on reset, clock
end process;
```


Comandos concorrentes

- ATRIBUIÇÃO DE SINAIS

  alu_result <= op1 + op2;

- ATRIBUIÇÃO SELETIVA DE SINAIS

```
with alu_function select
  alu_result <= op1 + op2       when alu_add | alu_incr,
                op1 – op2       when alu_subtract,
                op1 and op2     when alu_and,
                op1 or op2      when alu_or,
                op1 and not op2 when alu_mask;
```

- Tem semântica similar ao comando CASE em um processo

Comandos concorrentes

- ATRIBUIÇÃO CONDICIONAL DE SINAIS
  - construção é análoga a um processo com sinais na *sensitivity list* e um *if-then-else* para determinar o valor de *mux_out*.

```
mux_out <= 'Z'    after Tpd when en =  '0' else
           in_0   after Tpd when sel = '0' else
           in_1   after Tpd;
```

"mux_out" dependente dos sinais "en" e "sel".

  - escreva a atribuição de "mux_out" em um processo com if-then-else