MC613

Laboratório de Circuitos Lógicos

2010

Profs.:

Guido Araújo (guido @ ic….)
Mário Lúcio Cortes (cortes @ ic….)

MC613

VHDL - Introdução

Baseado em:

"The Designer´s Guide to VHDL"

e

"Fundamentals of Digital Logic with VHDL Design"

Conteúdo

- Níveis de Abstração
- Conceitos Básicos
- VHDL - uma visão geral
- Estrutura do Código VHDL
- Declarações
- Constantes
- Variáveis
- Tipos
- Operadores Aritméticos
  - rem e mod
- Exemplos de Números
- Ponto Flutuante

Conteúdo ...

- Tipos Físicos
- Tipos Enumerados
- Caracteres
- Booleans
- Bits
- Standard Logic
- Entidade
- Arquitetura
- Atribuições
  - Constante, variáveis e sinais
- Operadores Lógicos
- Operadores Relacionáis

Conteúdo ...

- Modelo VHDL Completo
- Packages
- Library e Use
- Subtipos
  - Conversão de tipos
- Processos
  - Construtores Seqüenciais
    - if, case, null, for e wait
- Vetores
  - Atribuições
- Comandos Concorrentes
  - when, select
- Atributos

Níveis de Abstração

- **Comportamental**: Descrição utilizando construções de alto nível da linguagem
- **RTL**: Nível intermediário, inclui mapeamento de portas
- ***Gate Level***: Nível de portas lógicas

VHDL - uma visão geral

- 5 tipos de unidades
  - Entity - define a interface do projeto, módulo, etc.
  - Architecture - descreve funcionalmente a entidade. Pode haver mais de uma arquitetura para uma mesma entidade.
  - Package - declarações comuns a todo o projeto. Exemplo: constantes, tipos de dados e subprogramas.
  - Packge Body - contém o corpo dos subprogramas definidos no Packge
  - Configuration - Faz a ligação de uma entidade com uma particular arquitetura, formando um componente.

VHDL - uma visão geral

- **Packages:** Assim como em linguagens de programação são utilizadas bibliotecas (de funções, procedimentos, definições de tipos, declarações de constantes etc), em VHDL isto é feito com a utilização de Packages e Bibliotecas de componentes.

- **Entidades:** Define a interface de um componente: nome, tipo dos sinais de entrada e/ou saída, ...

- **Arquiteturas:** Define a funcionalidade de um componente e sua temporização. Uma mesma entidade pode possuir múltiplas arquiteturas e para efeito de simulação e síntese é usada a última arquitetura compilada.

VHDL - uma visão geral

- **Processo:** É uma porção de código delimitada pelas palavras **Process** e **End Process** que contém comandos seqüenciais (são simulados em delta-delays que somados resultam em zero). Todos os processos de um modelo VHDL de um componente são executados concorrentemente em um time-step.

- **Componente:** É descrito pelo par Entity e Architecture. Um modelo VHDL é dito estrutural se faz uso de instanciação de componentes.

Estrutura do Código VHDL

- **Declarações**
  - Objetos que serão usados em comandos concorrentes ou seqüenciais
  - Declarados antes da clausula begin em arquiteturas, blocos, processos, procedimentos e funções
- **Comandos concorrentes**
  - block e process - comandos que serão executados em paralelo, independentemente uns dos outros
- **Comandos seqüenciais**
  - comandos que serão executados de forma seqüencial, obedecendo o fluxo de controle
  - comandos após a cláusula begin em processos

Declarações

- **Constante:** nome assinalado a um valor fixo
  - `CONSTANT vdd: real := 4.5;`
  - `CONSTANT cinco: integer := 3 +2;`

- **Variável:** nome assinalado a um valor que muda de acordo com um determinado processo
  - `VARIABLE largura_pulso: time range 1ns to 15ns := 3ns;`
  - `VARIABLE memoria: bit_vector (0 to 7);`

- **Sinal:** conectam entidades e transmitem mudanças de valores entre os processos (todo `port` é um sinal).
  - `SIGNAL contador : integer range 0 to 63;`
  - `SIGNAL condicao : boolean := TRUE;`

Constantes

```
constant number_of_bytes : integer := 4;

constant number_of_bits : integer := 8 *
  number_of_bytes;

constant e : real := 2.718281828;

constant prop_delay : time := 3 ns;

constant size_limit, count_limit : integer := 255;
```

Variáveis

```
variable index : integer := 0;

variable sum, average, largest : real;

variable start, finish : time := 0 ns;
```

Onde colocar as declarações?

```
architecture exemplo of entidade is
   constant pi : real := 3.14159;
begin
   process is
        variable contador : integer;
   begin
        ... -- uso de pi e contador
   end process;
end architecture exemplo;
```

Tipos

- Declaração de tipos

```
type apples is range 0 to 100;
type oranges is range 0 to 100;
type grapes is range 100 downto 0;
```

- O tipo *apples* é incompatível com *oranges*

```
constant number_of_bits : integer := 32;
type bit_index is range 0 to number_of_bits-1;
```

- OBS.: O valor padrão é o mais a esquerda do intervalo

Operadores Aritméticos

+ → soma ou identidade

- → subtração ou negação

* → multiplicação

/ → divisão

mod → módulo

abs → valor absoluto

** → exponenciação

Exemplos de Números

23		0		146
23.1		0.0		3.14159
46E5		1E+12		19e00
1.234E09		98.6E+21		34.0e-08
2#11111101#	=	16#FD#	=	16#0fd#
2#0.100#	=	8#0.4#	=	12#0.6#
2#1#E10	=	16#4#E2	=	10#1024#E+00
123_456		3.131_592_6		2#1111_1100_0000_0000#

8# → base 8

Tipos Enumerados

- **Exemplo**

```
type alu_funcion is (disable, pass, add, subtract,
  multiply, divide);
type octal_digit is ('0', '1', '2', '3', '4', '5', '6', '7');
type control is (stop, pass);
```

- **Uso**

```
variable command : alu_function := pass;
variable status : control := pass;
```

Caracteres

- VHDL 97 só aceitava ASCII padrão (128 valores)
  - Gerava problemas, inclusive, com os comentários

- VHDL 2000 usa ISO 8859 Latin 1, representado com 8 bits

```
variable cmd_char, terminator : character;
cmd_char := 'P';
terminator := cr;
```

Booleans

type boolean is (false, true);

- Operadores
  - and, or, nand, nor, xor, xnor, not

- OBS.: and, or, nand e nor fazem *lazy evaluation* (ou *short circuit*)
  - O segundo operando não será avaliado se o primeiro já indicar a resposta

Bits

type bit is ('0', '1');

- Todos os operadores lógicos valem para bits
- Valores entre aspas simples

```
variable switch : bit := '0';
```

Standard Logic

- Pacote std_logic_1164

```
type std_ulogic is (
        'U', -- não iniciado (unitialized)
        'X', -- desconhecido (unknow) forte
        '0', -- zero forte
        '1', -- um forte
        'Z', -- alta impedância ou desconectado (tri-state)
        'W', -- desconhecido fraco
        'L', -- zero fraco
        'H', -- um fraco
        '-'); -- indiferente (don't care)
```

OBS.: -- inicia um comentário que vai até o fim da linha

Entidade

- Define a interface do componente com o restante do sistema

```
Entity nome IS
   GENERIC (lista_dos_genericos);
    PORT (lista_dos_ports);
END nome;
```

Port

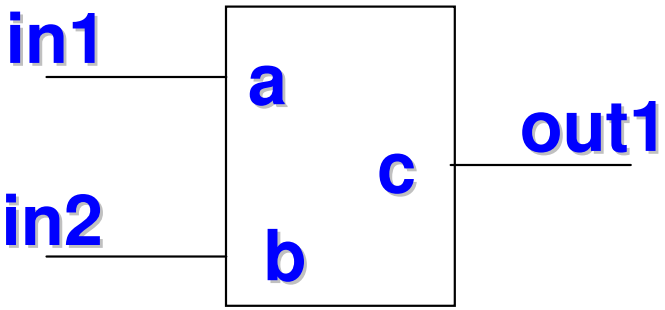
- Formato:

Port (nome: MODO tipo);

  - Modo: in, out, inout, buffer, linkage

- Exemplo:

Port (a, b : in bit;
c: out bit);



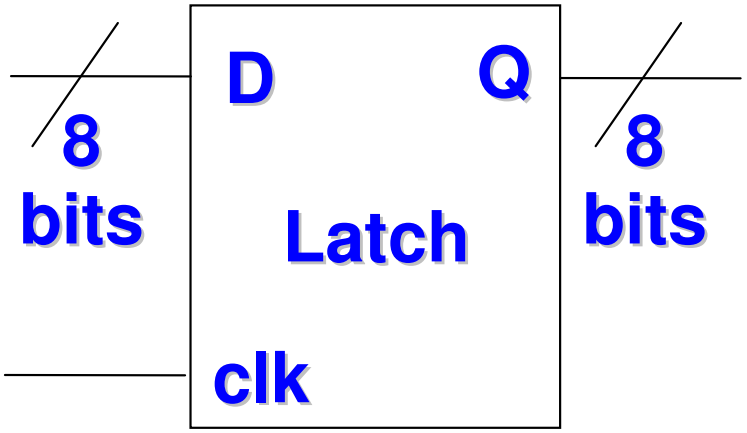
Exemplo

```
Entity porta_and IS
   GENERIC (numero_de_entradas: integer := 4);
   PORT (entradas : in bit_vector (1 to numero_de_entradas);
                saida    : out bit);
END porta_end;
```

- Port MODE:
  - IN: sinal é somente de entrada
  - OUT: sinal é somente de saída
  - Buffer: sinal é de entrada e saída (um de cada vez)
  - Inout: sinal é bidirecional, implica em um BUS
  - Linkage: direção do sinal é desconhecida

Exemplo

- Qual é a Entidade?




```
Entity latch is
  port (d    : in   bit_vector(7 downto 0);
        clk :  in   bit;
        q    :  out out bit_vector(7 downto 0);
End latch;
```

Exemplo

- Usando *generic*

```
Entity latch is
  generic(w : integer := 8);
  port (d     : in   bit_vector(w-1 downto 0);
        clk : in   bit;
        q     : out bit_vector(w-1 downto 0);
End latch;
```

Arquitetura

- Estabelece a relação entre entradas e saídas
  - Funções
  - Procedimentos
  - Execução paralela de processos
  - Instanciação de componentes

- Arquiteturas múltiplas
  - Utiliza a última compilada

Arquitetura

```
ARCHITECTURE  label OF nome_entidade IS
  --  parte declarativa (declarações de tipos, subtipos, sinais,
  --  funções, procedimentos, ...)

BEGIN
     --  comandos concorrentes
END label;
```

```
ARCHITECTURE  rtl OF porta_and IS
  constant atraso : time := 5 ns;

BEGIN
     y <=  a AND b AFTER atraso;
END rtl;
```

Qual a entidade?

Arquitetura

```
Entity porta_and is
  port (a, b  : in   bit;
        y     :  out bit);
End porta_and;
```

```
Entity porta_and is
  generic(w : integer := 8);
  port (a, b   :  in   bit_vector(w-1 downto 0);
        y      :  out bit_vector(w-1 downto 0);
End porta_and;
```

Atribuições

- **Atribuição a sinal:**

<=

- **Atribuição a variável:**

:=

- **Inicialização** (constante, sinal e variável)**:**

:=

Atribuições de Dados

- **Constante:**

```
Constant cnt_reset : bit_vector(0 to 3) := “1001”;
```

- **Variável:**

```
var1 := 2005;
```

- **Sinal:**

```
dado <= ‘1’;
d <= ‘1’, ‘0’ AFTER 5 ns;
```

Atribuições e Operadores Aritméticos

Operador	Operação	Exemplo
+	Adição	I := i + 2;
-	Subtração	J <= j -10;
*	Multiplicação	M := fator * n;
/	Divisão	K := i / 2;
**	Potenciação	I := i ** 3;
ABS	Valor absoluto	Y <= ABS(tmp)
MOD	Módulo	Z <= MOD(t);
REM	resto	R <= REM(tot);

Operadores Lógicos

- Predefinidos para os tipos: bit; std_logic, std_ulogic, boolean
  - NOT
  - AND
  - NAND
  - OR
  - NOR
  - XOR
  - XNOR

Operadores Relacionais

Operador	Operação
=	Igual
/=	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual que
<=	Menor ou igual que

Modelo Completo

- Indica as bibliotecas utilizadas
- Faz uso das definições contidas nas bibliotecas
- Descreve a entidade
- Descreve a arquitetura

Exemplo: or de 2 entradas

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity or2 is
          --  porta or de duas entradas
  port (i1, i2 :  in   bit;
          out1 : out bit);
End or2;

Architecture rtl of or2 is
Begin
    out1 <= i1 or i2;
End rtl;
```

Packages

- Coleção de declarações comuns definidas fora dos modelos (corpo + declaração)

- Uso de:
  - Tipos; Subtipos
  - Subprogramas (Funções e Procedimentos)
  - Constantes; Sinais; Aliases
  - Atributos
  - Componentes

Packages

```
PACKAGE label IS

    --  declarações;

END label;

Package BODY label IS

    --  Corpo de subprogramas;

END label;
```

Library e USE

- **LIBRARY nome_da_biblioteca;**

```
Library IEEE;
```

- **A cláusula USE torna os pacotes visíveis em entidades e arquiteturas**
  - **USE nome_biblioteca.nome_package;**
  - **USE nome_package.identificador;**

```
USE IEEE.std_ulogic_1164.all;
USE math.all;
USE textio.all;
```

Subtipos

- **Formato:**

```
SUBTYPE natural IS INTEGER RANGE 0 to
2147483647;

SUBTYPE positive IS INTEGER RANGE 1 to
2147483647;
```

Conversão de Tipos

- Possível para conversão de alguns tipos
  - Inteiro --> Ponto Flutuante

- Formato: tipo(objeto)
  - Exemplo:

```
soma := REAL(N1) + N2;
```

- OBS.: Funções de Conversão definidas nos pacotes.

VHDL - Exemplo

- Qual o circuito implementado (em termos de portas lógicas)?

```
Library IEEE;
use IEEE.std_logic_1164.all;
Entity cct1 IS
   Port (in1, in2,in3,in4 : IN std_logic;
         out1, out2 : OUT std_logic);
End cct1;
Architecture rtl OF cct1 IS
  signal out_i : std_logic;
Begin
   out_i  <=  in1 and in2 and in3;
   out1   <=  out_i;
   out2   <=  in4 XOR out_i;
End rtl
```

Processos

- Um processo equivale a um comando concorrente.
-
- As instruções no interior do processo são instruções seqüenciais.

- Todos os processos são executados concorrentemente.

- Cada instrução dentro de um processo leva um tempo "delta time" que somados são iguais a zero

Processo

```
Label: PROCESS (lista de sensitividade)
            cláusulas declarativas;
       BEGIN
            --  comentários
                inicializações;
                cláusulas de atribuição;
       END PROCESS label;
```

OBS.: A lista de sensibilidade funciona como um comando WAIT ON

Process - exemplo

```
Architecture bhv OF generic_decoder IS
Begin
PROCESS (sel, en)
   BEGIN
      y <= (others => '1');
      FOR i IN y'range LOOP
          IF (en = '1' and (bvtoi(To_Bitvector(sel)) = i))
          THEN
              y(i) <= '0';
          END IF;
      END LOOP;
END PROCESS;
END bhv
```

atributo

Função de conversão de tipo bit_vector para inteiro

Exemplo de processo

```
entity latch_d is
  port (dado, enable : in std_logic;
        q, notq : out std_logic);
end latch_d;

architecture rtl of latch_d is
begin
latchD: process (dado,enable)
begin
        IF enable = '1' THEN
        Q <= dado;
        notQ <= not dado
    END IF;
end process latchD;
end rtl;
```

Construtores Seqüenciais

- **Válidos quando dentro de processos**
- **Nem todos são sintetizáveis**
  - **if, case, null, for**
  - while, loop, exit, next, wait

if

```
[if_label:] if expressão_lógica then
   ...
elsif expressão_lógica then
   ...
else
   ...
end if [if_label];
```

case

```
[case_label:] case expression is
  when opção1 =>
       ...
  when opção2 =>
       ...
  when others =>
       ...
end case [case_label];
```

case (Exemplo)

```
case opcode is
  when load | add | subtract =>
       operand := memory_operand;
  when store | jump | jumpsub | branch =>
       operand := address_operand;
  when others =>
       operand := none;
end case;
```

case

- Também pode ser indicado intervalo

```
when 0 to 9 =>
```

- OBS.: Para síntese, é necessário cobrir todas as opções
  - Usar others quando em dúvida
  - std_logic tem 9 valores!

null

- Comando nulo

for

```
[for_label:] for identificador in intervalo
  loop
    ...
  end loop [for_label];
```

- **OBS.:**
  - Para hardware, significa replicação
  - *identificador* não precisa ser declarado
    - *e não pode ter valor atribuído*

for (exemplo)

```
for count_value in 0 to 127 loop
  count_out <= count_value;
  wait for 5 ns;
end loop;
```

for (exemplo)

```
type controller_state is (initial, idle, active,
  error);
…
for state in controller_state loop
  …
end loop;
```

wait

Formato

WAIT [ON lista]

espera atividade em algum sinal da lista

WAIT [UNTIL condição]

espera até que condição ocorra (condição seja verdadeira)

WAIT [FOR tempo]

espera pelo tempo especificado

Wait: exemplo

```
Process
    variable i : integer := 0;
begin
    i := i + 1;
    j <= j + 1;
    wait on k;
    wait for 100 ns;
    wait on m until j > 5; -- Condição só é testada
                           -- quando ocorre um evento em m
end process;
```

wait

wait; -- espera infinita

wait until clk = '1'; -- espera até que um evento satisfaça a condição clk = '1'

wait on clk until reset = '0'; -- espera até que ocorra um evento em clk e que se satisfaça a condição reset = '0'

wait until trigger = '1' for 1 ms; -- espera até que ocorra um evento em trigger ou se passe 1 milisegundo

Wait: exemplo

```
exemplo: process
        variable var : integr := 0;
        signal x, y : bit := '0';
    begin
        var := var + 1;
        var := var + var;
        x <= not y;
        x <= y;
        wait;   --  por quanto tempo?
    End process exemplo;
```

Vetores

```
type word is array (0 to 31) of bit;

type word is array (31 downto 0) of bit;

type controller_state is (initial, idle, active,
  error);

type state_counts is array (idle to error) of
  natural;

type matrix is array (1 to 3, 1 to 3) of real;
```

Atribuição de Valores para Vetores

```
subtype coeff_ram_address is integer range 0 to
  63;
type coeff_array is array (coeff_ram_address)
  of real;
variable coeff : coeff_array := (0 => 1.6, 1 =>
  2.3, 2 => 1.6, 3 to 63 => 0.0);
variable coeff: coeff_array := (0 => 1.6, 1 =>
  2.3, others => 0.0);
variable coeff : coeff_array := (0 | 2 => 1.6, 1
  => 2.3, others => 0.0);
```

Vetores

```
type bit_vector is array (natural range <>) of bit;

subtype byte is bit_vector(7 downto 0);

type std_ulogic_vector is array (natural range <>) of
  std_ulogic;

variable channel_busy_register : bit_vector(1 to 4);

variable  current_test : std_ulogic_vector(0 to 13)
  := "ZZZZZZZZZZ----";
```

Exemplos

- Escrever um modelo para um latch tipo D

```
Library ieee;
use ieee.std_logic_1164.all;

entity latch_d is
  port (dado, enable : in
   std_logic;
         q, notq : out std_logic);
end latch_d;
```

```
architecture rtl of latch_d is
begin
latch: process
begin
   wait until enable = '1'
   q <= dado;
   notq <= not dado;
end process latch;
end rtl;
```

Exemplos

- Latch D com if

```
Library ieee;
use ieee.std_logic_1164.all;
entity latch_d is
  port (dado, enable : in std_logic;
          q, notq : out std_logic);
end latch_d;
architecture rtl of latch_d is
begin
latchD: process (dado,enable)
begin
  IF enable = '1' THEN
          Q <= dado;
          notQ <= not dado
    END IF;
end process latchD;
end rtl;
```

Exemplo: Flip-Flop D

```
entity FlipFlopD is
     port (d, clk : in bit;
               q, notq : out bit);
end entity FlipFlopD;
architecture behavior of FlipFlopD is
    signal state : bit;
begin
    process (clk) is
    begin
          if (clk'event and clk = '1') then
                    state <= d;
          end if;
          q <= state;
          notq <= not state;
    end process;
end architecture behavior;
```

tem que incluir state também?

```
q <= d;
notq <= not d;
```

contador de 4 bits

```
Architecture rtl of contador is
  signal cont : unsigned(3 downto 0) := "0000";
begin
   process(reset,clk)
   begin
       if reset = '1' then
          cont <= "0000";
       elseif (clk'event and clk = '1') then
           if enable = '1' then
              if incr = '1' then cont <= cont + "0001";
              else  cont <= cont - "0001";
              end if;
           end if;
        end if;  end process;  end rtl;
```

Escreva a Entidade Correspondente

Comandos Concorrentes (paralelos)

- Executados fora dos processos
- Possuem equivalentes para serem executados dentro dos processos
- Cada um tem o seu lugar, não é permitida a troca

when (≈ if)

```
architecture bhv of M is
begin
   process (sel, d0, d1) is
        if (sel = '0') then
          z <= d0;
        else
          z <= d1;
        end if;
   end process;
end architecture bhv;
```

```
architecture bhv of M is
begin
   z <= d0 when sel = '0'
   else d1;
end architecture bhv;
```

when

```
architecture bhv of M is
begin
   z <=  d0 when sel0 = '0' and sel1 = '0' else
         d1 when sel0 = '0' and sel1 = '1' else
         unaffected when sel0 = '1' and sel1 = '0'
         else
         d2;
end architecture bhv;
```

select (≈ case)

```
with alu_function select
   result <= a + b when alu_add | alu_add_u,
             a - b when alu_sub | alu_sub_u,
             a and b when alu_and,
             a or b when alu_or
             a when alu_pass_a;
```

Resumo de comandos

- Sequenciais (dentro de processos)
  - if
  - case
  - null
  - for
  - while
  - loop
  - exit
  - next
  - wait
- Concorrentes (fora de processos)
  - when (else)
  - with select