

VHDL - Introdução

MO801/MC912

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

Definições

- RTL: Register Transfer Level → Nível de transferência de registradores
- VHDL RTL: VHDL sintetizável
- Behavioral VHDL: VHDL comportamental → Síntese dependente da forma de escrita e da ferramenta utilizada
- **Definição comum: VHDL RTL é tudo que pode ser sintetizável**

Ferramentas

- ghdl (Linux)
 - Front-end do gcc para VHDL
- Altera Quartus II (Windows/Linux?)
 - Síntese para FPGA da Altera
- Xilinx ISE (Windows/Linux?)
 - Síntese para FPGA da Xilinx
- Modelsim (Windows/Linux?)
 - Simulador
- Mentor Leonardo ou Precision (Windows/Linux)
 - Síntese para diversos fabricantes

A linguagem

- Baseada em ADA
- Desenvolvida a pedido do departamento de defesa americano
- Finalidade original: Especificar os circuitos de forma não ambígua
- Padrão IEEE 1076 (1987)
 - Revisão em 1993
 - Revisão em 2000

Construções da Linguagem

- Foco apenas na parte sintetizável
- Não serão gastas muitas aulas nessa parte
 - Algum tempo nas construções básicas da linguagem
 - Para maiores detalhes, consultar o livro
- Abordagem por exemplos de código e descrição de funcionalidades extras

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 declarar?

```
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;
- O valor padrão é o mais a esquerda do intervalo

Operadores Aritméticos

+ → soma ou identidade

- → subtração ou negação

* → multiplicação

/ → divisão

mod → módulo

rem → resto da divisão

abs → valor absoluto

** → exponenciação

rem e mod

- $A = (A / B) * B + (A \text{ rem } B)$

$$5 \text{ rem } 3 = 2$$

$$(-5) \text{ rem } 3 = -2$$

$$5 \text{ rem } (-3) = 2$$

$$(-5) \text{ rem } (-3) = -2$$

- $A = B * N + (A \text{ mod } B)$

$$5 \text{ mod } 3 = 2$$

$$(-5) \text{ mod } 3 = 1$$

$$5 \text{ mod } (-3) = -1$$

$$(-5) \text{ mod } (-3) = -2$$

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#

Ponto Flutuante

- VHDL 2000 exige, no mínimo, 64 bits
- VHDL 87 e VHDL 93 exigem, no mínimo, 32 bits

type input_level **is range** -10.0 **to** +10.0;

variable x : input_level;

variable y : real;

- O valor padrão é o mais a esquerda do intervalo

Tipos Físicos

- Um número + uma unidade
type length is range 0 to 1E9
units
um;
mm = 1000 um;
m = 1000 mm;
inch = 25400 um;
foot = 12 inch;
end units length;

Tipos Físicos

- Atenção para as unidades
 - Só é possível somar e subtrair tipos da mesma unidade
 - Multiplicação e divisão por inteiro ou real mantém a unidade
- Uso
 - variable** distance : length := 100 m;

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
- 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)

Construtores seqüenciais

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

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 =>
- 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];

- 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;
```

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;

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) := "ZZZZZZZZZZZ----";

Estrutura Básica de um Arquivo VHDL

entity and2 **is**

port (a, b : **in** bit;
 c : **out** bit);

end entity and2;

architecture behavior **of** and2 **is**

begin

 c <= a **and** b;

end architecture behavior;

Estrutura Básica de um Arquivo VHDL

```
entity and2 is  
  port (a, b : in bit;  
         c : out bit);  
end entity and2;  
architecture behavior of and2 is  
begin  
  comb: process (a, b) is  
    begin  
      c <= a and b;  
    end process comb;  
end architecture behavior;
```

Flip-Flop D (entidade)

```
entity FlipFlopD is  
    port (d, clk : in bit;  
          q, notq : out bit);  
end entity FlipFlopD;
```

Flip-Flop D (arquitetura)

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;

Comandos 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;

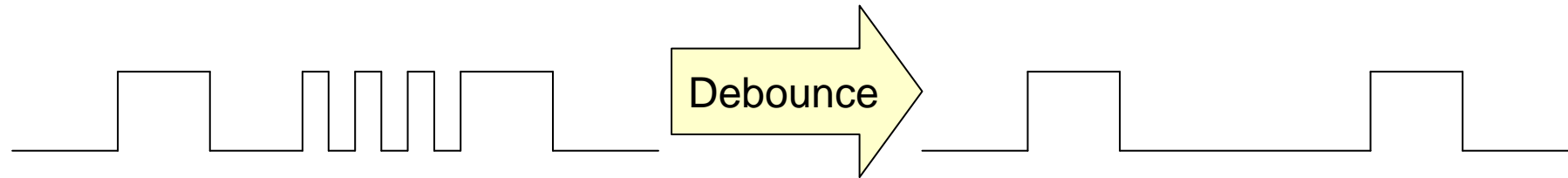
select (case)

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;

Exemplo 1

- Circuito para fazer debounce
 - Remove oscilações do circuito



- Assuma que as oscilações durem menos de 5 ciclos de clock

Exemplo 2

- Fazer um contador com 2 botões de entrada, um para contar para cima e outro para contar para baixo. Use o componente de debounce do exemplo anterior.
- Se o botão ficar apertado, o contador não deve contar de novo.

Atributos

- **Atributos**

- A'left(N)
- A'rirht(N)
- A'low(N)
- A'high(N)
- A'range(N)
- A'reverse_range(N)
- A'length(N)
- A'ascending(N)

type A is array (1 to 4 , 31 downto 0) of boolean;

- A'left(1) = 1
- A'rirht(2) = 0
- A'low(1) = 1
- A'high(2) = 31
- A'range(1) is 1 to 4
- A'reverse_range(2) is 0 to 31
- A'length(2) = 32
- A'ascending(1) = true
- A'ascending(2) = false

Atributos

```
type resistance is range 0  
  to 1E9  
  units  
    ohm;  
    kohm = 1000 ohm;  
    Mohm = 1000 kohm;  
end units resistance;
```

```
resistance'left = 0 ohm;  
resistance'right = 1E9 ohm;  
resistance'low = 0 ohm;  
resistance'high = 1E9 ohm;  
resistance'ascending = true;  
resistance'image(2 kohm) =  
  "2000 ohm";  
resistance'value("5 Mohm")  
  = 5_000_000 ohm;
```

Atributos

```
type set_index_range is range 21  
  downto 11;
```

```
type logic_level is (unknown, low,  
  undriven, high);
```

- set_index_range'left = 21;
- set_index_range'right = 11;
- set_index_range'low = 11;
- set_index_range'high = 21;
- set_index_range'ascending = false;
- set_index_range'image(14) = "14";
- set_index_range'value("20") = 20;

- logic_level'left = unknown;
- logic_level'right = high;
- logic_level'low = unknown;
- logic_level'high = high;
- logic_level'ascending = true;
- logic_level'image(undriven) = "undriven";
- logic_level'value("Low") = low;
- logic_level'pos(unknown) = 0;
- logic_level'val(3) = high;
- logic_level'succ(unknown) = low;
- logic_level'pred(undriven) = low;