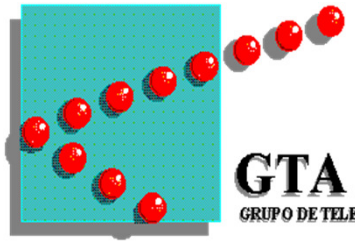

Sistemas Digitais
EEL 480

Introdução à Linguagem VHDL

Luís Henrique M. K. Costa

luish@gta.ufrj.br

UFRJ – DEL/Poli e PEE/COPPE
P.O. Box 68504 - CEP 21941-972 - Rio de Janeiro - RJ
Brasil - http://www.gta.ufrj.br

Introdução

- VHDL
  - VHSIC (*Very High Speed Integrated Circuits*) Hardware Description Language
  - Desenvolvida pelo Departmento de Defesa americano
    - VHDL 87, 93, 2002, 2008 (IEEE 1076-2008)

- Objetivos
  - Descrição por software do projeto (*design*) de um sistema digital
  - Simulação
  - Síntese

Observações Iniciais

- A linguagem não é *case-sensitive*
  - mas freqüentemente são usadas maiúsculas para as palavras reservadas

- Comentários
  - Iniciados por “- -”
  - Terminados pelo fim de linha

Comandos Básicos

- Atribuição de sinal
  - A <= B;
- Comparação
  - “=”, “>”, “<”, etc.
- Operações Booleanas
  - AND, OR, NOT, XOR
- Declarações Sequenciais
  - CASE, IF, FOR
- Declarações Concorrentes
  - WHEN-ELSE

Elementos Básicos de um Modelo VHDL

- Declaração ENTITY
  - Descreve a *interface* do modelo: entradas e saídas
- Corpo ARCHITECTURE
  - Descreve o *comportamento* do modelo
    - Podem existir várias ARCHITECTURE para uma mesma ENTITY

Objetos de Manipulação de Valores

- CONSTANT
  - Definição de valores constantes
- SIGNAL
  - Passagem de valores de dentro para fora, ou entre unidades internas do circuito (~fios)
- VARIABLE
  - Armazenamento de valores na parte sequencial do circuito
    - Válida apenas dentro de um **process**

Exemplos de Constantes

CONSTANT dez: INTEGER := 10;

- GENERIC
  - similar a CONSTANT
  - definido na entidade, constante para a arquitetura
  - pode ser mapeado para outro valor, quando importado como componente

```
ENTITY exemplo is
    generic (N: integer := 4);
    port(
        ...
        )
```

Exemplo – Contador de 4 bits

```
ENTITY counter_4 IS PORT(
    clk, reset, load_counter:       IN BIT;
    data:                           IN BIT_VECTOR( 3 DOWNTO 0 );
    count_zero:                     OUT BIT;
    count:                          BUFFER BIT_VECTOR( 3 DOWNTO 0 )
    );
END counter_4;
```

- Cada sinal possui um **modo** (IN, OUT, BUFFER) e um **tipo** (BIT, BIT_VECTOR)

Modos do Sinal PORT

- **IN**: dados fluem para dentro da Entidade, que não pode escrever estes sinais
  - Ex. Clock, entradas de controle, entradas unidirecionais de dados
- **OUT**: dados fluem para fora da Entidade, que não pode ler estes sinais
  - O modo OUT é usado quando a Entidade nunca lê estes dados
- **BUFFER**: dados fluem para fora da Entidade, que *pode* ler estes sinais, permitindo realimentação interna
  - No entanto, o BUFFER não pode ser usado para *entrada* de dados
- **INOUT**: dados podem fluir para dentro ou para fora da Entidade
  - Só deve ser usado se necessário
    - Ex. Barramento de dados bidirecional
  - Design menos compreensível

Tipos do VHDL

- BIT, BIT_VECTOR
  - Valores: “0” ou “1”
  - Atribuição de valor: bit_signal <= '0';
  - Nativos da linguagem VHDL, não precisam de declaração de biblioteca

- STD_LOGIC, STD_LOGIC_VECTOR
  - Valores: “0”, “1”, “-” (don’t care), “Z” (alta impedância), “X” (indeterminado)
  - Biblioteca ieee
  - Declarações necessárias
    - LIBRARY
    - USE

Tipos do VHDL

- INTEGER
  - Valores: - ($2^{31} - 1$) até $2^{31} - 1$
  - Atribuição de valor: integer_signal <= 19;
- NATURAL
  - Valores: 0 até $2^{31} - 1$
  - Atribuição de valor: natural_signal <= 19;
- CHARACTER
  - Valores: caracteres ISO 8859-1
  - Atribuição de valor: char_signal <= ‘a’;

Vetores

- Declaração

  bit_vector_signal : BIT_VECTOR(maximum_index DOWNTO 0);

  bit_vector_hum, bit_vector_dois : BIT_VECTOR(3 DOWNTO 0);
  bit_sozinho : BIT;

- Atribuição
  - bit_vector_hum(0) <= ‘1’;
  - bit_vector_hum <= bit_vector_dois;
  - bit_vector_hum(0) <= bit_sozinho;
  - bit_vector_hum(0) <= bit_vector_dois(3);
  - bit_vector_hum <= "0001";

Exemplo de Entidade: Contador de 4 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY counter_4 IS PORT(
   clock, reset, load_counter: IN std_logic;
   data:                       IN std_logic_vector( 3 DOWNTO 0 );
   reset_alert:                   OUT std_logic;
   count:                      BUFFER std_logic_vector( 3 DOWNTO 0 )
   );
END counter_4;
```

- Evitar misturar BIT com STD_LOGIC
  - Existem funções de conversão mas o código se torna mais complexo

Exemplo Completo: Maioria de 3

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY majconc IS
        PORT ( A, B, C :      IN std_logic;
                         Y:      OUT std_logic
        );
END majconc;

ARCHITECTURE arq_majconc OF majconc IS
BEGIN
        Y <= (A and B) or (A and C) or (B and C);
END arq_majconc;
```

Exemplo: *Full-Adder*

```
ENTITY full_adder IS
        PORT (        a, b, carry_in :        IN BIT;
                      sum, carry_out:         OUT  BIT
              );
END full_adder;
```

ARCHITECTURE 1

- Descrição de fluxo de dados (*dataflow*) ou concorrente
  - Atribuições ocorrem simultaneamente
    - Geralmente descrevem o fluxo de dados no sistema

```
ARCHITECTURE dataflow OF full_adder IS
   SIGNAL x1, x2, x3, x4, y1 : BIT;
BEGIN
   x1 <= a AND b;
   x2 <= a AND carry_in;
   x3 <= b AND carry_in;
   x4 <= x1 OR x2;
   carry_out <= x3 OR x4;
   y1 <= a XOR b;
   sum <= y1 XOR carry_in;
END dataflow;
```

ARCHITECTURE 1

- Pode-se eventualmente eliminar os sinais internos adicionais…

```
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   carry_out <= ( a AND b ) OR ( a AND carry_in ) OR ( b AND carry_in );
   sum <= a XOR b XOR carry_in;
END dataflow;
```

ARCHITECTURE 1

- Pode-se usar comandos condicionais…

```
output_vector <=      "00" WHEN ( a = b )
                  ELSE "01" WHEN ( a = c )       - - and a != b
                  ELSE "10" WHEN ( a = d )       - - and a != b and a != c
                  ELSE "11";
```

```
WITH selecting_vector SELECT
   output_vector <= "0001" WHEN "00",
                    "0010" WHEN "01",
                    "0100" WHEN "10",
                    "1000" WHEN "11";
```

ARCHITECTURE 2

- Descrição Estrutural
  - As atribuições de sinais são feitas através do mapeamento de entradas e saídas de *componentes*

```
ENTITY full_adder IS PORT (
   a, b, carry_in   : IN  BIT;
   sum, carry_out   : OUT BIT
   );
END full_adder;
```

```
ARCHITECTURE structural OF full_adder IS
        SIGNAL x1, x2, x3, x4, y1 : BIT;
        COMPONENT and_gate PORT (
                a, b      :   IN  BIT;
                a_and_b   :   OUT BIT
                );
        END COMPONENT and_gate;
        COMPONENT or_gate PORT (
                a, b      :   IN  BIT;
                a_or_b    :   OUT BIT
                );
        END COMPONENT or_gate;
        COMPONENT xor_gate PORT (
                a, b      :   IN  BIT;
                a_xor_b   :   OUT BIT
                );
        END COMPONENT xor_gate;
        BEGIN
                and0 : and_gate PORT MAP( a, b, x1 );
                and1 : and_gate PORT MAP( a, carry_in, x2 );
                and2 : and_gate PORT MAP( b, carry_in, x3 );
                or0:   or_gate  PORT MAP( x1, x2, x4 );
                or1:   or_gate  PORT MAP( x3, x4, carry_out );
                xor0:  xor_gate PORT MAP( a, b, y1 );
                xor1:  xor_gate PORT MAP( y1, carry_in, sum );
END structural;
```

ARCHITECTURE 3

- Descrição Comportamental
  - Usada na descrição de sistemas seqüenciais
- Elemento fundamental: PROCESS
  - *label* (opcional), a palavra **PROCESS**, e uma *lista de sensibilidade*

```
process_name: PROCESS( sensitivity_list_signal_1, ... )
BEGIN
   -- comandos do processo
END PROCESS process_name;
```

Codificador de prioridade

- 7 entradas
- Y7 mais prioritária
- Saída: 3 bits
  - Indica entrada mais prioritária em 1
  - 0 se nenhuma entrada em 1

```
library ieee;
use ieee.std_logic_1164.all;
entity priority is
   port ( y1, y2, y3, y4, y5, y6, y7 : in std_logic;
            dout: out std_logic_vector(2 downto 0)
          );
end priority;
```

Codificador de prioridade

- Com comandos IF / ELSIF

```
architecture ifels of priority is
begin
  process (y1, y2,y3, y4, y5, y6, y7)
  begin
            if (y7 = '1') then dout <= "111";
            elsif (y6 = '1') then dout <= "110";
            elsif (y5 = '1') then dout <= "101";
            elsif (y4 = '1') then dout <= "100";
            elsif (y3 = '1') then dout <= "011";
            elsif (y2 = '1') then dout <= "010";
            elsif (y1 = '1') then dout <= "001";
            else dout <= "000";
            end if;
  end process;
end ifels;
```

Codificador de prioridade

- Com comandos IF
- No PROCESS, o último comando executado é o que conta
  - Por isso a ordem das atribuições foi invertida

```
architecture so_if of priority is
begin
  process (y1, y2,y3, y4, y5, y6, y7)
  begin
            dout <= "000;
            if (y1 = '1') then dout <= "001"; end if;
            if (y2 = '1') then dout <= "010"; end if;
            if (y3 = '1') then dout <= "011"; end if;
            if (y4 = '1') then dout <= "100"; end if;
            if (y5 = '1') then dout <= "101"; end if;
            if (y6 = '1') then dout <= "110"; end if;
            if (y7 = '1') then dout <= "111"; end if;
  end process;
end so_if;
```

Codificador de prioridade

- Com apenas um comando

  WHEN / ELSE

- Sem PROCESS

```
architecture whenelse of priority is
begin
        dout <= "111" when (y7 = '1') else
                "110" when (y6 = '1') else
                "101" when (y5 = '1') else
                "100" when (y4 = '1') else
                "011" when (y3 = '1') else
                "010" when (y2 = '1') else
                "001" when (y1 = '1') else
                "000";
end whenelse;
```

MUX 4:1 com vetores de 8 bits

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4to1_8 is
        port (    a,b,c,d : in std_logic_vector(7 downto 0);
                  sel: in std_logic_vector (1 downto 0);
                  dout: out std_logic_vector(7 downto 0)
                  );
end mux4to1_8;
architecture whenelse of mux4to1_8 is
begin
        dout <= b when (sel = "01") else
                  c when (sel = "10") else
                  d when (sel = "11") else
                  a;                                    -- default
end whenelse;
```

Circuito seqüencial: Contador de 4 bits

- A entrada clock determina quando o estado do circuito muda

```
ENTITY counter_4 IS PORT(
   clock, reset, load_counter: IN BIT;
   data:                IN BIT_VECTOR( 3 DOWNTO 0 );
   reset_alert:         OUT BIT;
   count:               BUFFER BIT_VECTOR( 3 DOWNTO 0 )
   );
END counter_4;
```

```
ARCHITECTURE behavioral OF counter_4 IS
BEGIN
   upcount: PROCESS( clock ) BEGIN
      IF( clock'event AND clock= '1' ) THEN
         IF reset = '1' THEN
            count <= "0000";
         ELSIF load_counter = '1' THEN
            count <= data;
         ELSE
            count(0) <= NOT count(0);
            count(1) <= count(0) XOR count(1);
            count(2) <= ( count(0) AND count(1) ) XOR count(2);
            count(3) <= ( count(0) AND count(1) AND count(2) ) XOR count(3);
            IF count = "0000" THEN
               reset_alert <= '1';
            ELSE
               reset_alert <= '0';
            END IF; -- IF count = "0000"
         END IF; -- IF reset = '1'
      END IF; -- IF( clock'event AND clock = '1' )
   END PROCESS upcount;
END behavioral;
```

Signal x Variable

SIGNAL

- Declarada na ENTITY
- Escopo global
- Novo valor só é considerado após a conclusão do **process**
- Atribuição: **<=**
- Apenas uma atribuição válida no código inteiro

VARIABLE

- Declarada no PROCESS
- Escopo local
- Novo valor disponível imediatamente após a atribuição
- Atribuição: **:=**
- Múltiplas atribuições no código

Outro Contador de 4 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY counter_4 IS PORT(
    clock, reset, load_counter: IN STD_LOGIC;
    data:                       IN STD_LOGIC _VECTOR( 3 DOWNTO 0 );
    reset_alert:                OUT STD_LOGIC;
    count:                      OUT STD_LOGIC _VECTOR( 3 DOWNTO 0 )
    );
END counter_4;
```

```
ARCHITECTURE com_var OF counter_4 IS
CONSTANT nb: INTEGER := 3;
BEGIN
   upcount: PROCESS( clock )
            VARIABLE contagem: UNSIGNED (nb DOWNTO 0);
   BEGIN
      IF( clock'event AND clock= '1' ) THEN
         IF reset = '1' THEN
            contagem := "0000";
         ELSIF load_counter = '1' THEN
            contagem := data;
         ELSE
            contagem := contagem + 1;
            IF count = "0000" THEN
               reset_alert <= '1';
            ELSE
               reset_alert <= '0';
            END IF; -- IF count = "0000"
         END IF; -- IF reset = '1‘
         count <= std_logic_vector(contagem);
      END IF; -- IF( clock'event AND clock = '1' )
   END PROCESS upcount;
END com_var;
```

.J

Outro Contador de 4 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY generic_counter IS
   GENERIC nb: INTEGER := 3;
   PORT(
   clock, reset, load_counter: IN STD_LOGIC;
   data:                IN STD_LOGIC _VECTOR( nb DOWNTO 0 );
   reset_alert:         OUT STD_LOGIC;
   count:               OUT STD_LOGIC _VECTOR( nb DOWNTO 0 )
   );
END generic_counter;
```

Importando como Componente

Dentro da ARCHITECTURE...

```
signal loc_contagem: STD_LOGIC_VECTOR(7 DOWNTO 0);
signal loc_clock, loc_reset, loc_load_counter: STD_LOGIC;
-- signal loc_reset_alert: STD_LOGIC;
signal loc_data: STD_LOGIC_VECTOR (7 DOWNTO 0);

...

adder_128: work.generic_counter(arch)
        GENERIC MAP (N=>7)
        PORT MAP(contagem => loc_contagem, clock => loc_clock,
reset => loc_reset, load_counter => loc_load_counter, OPEN, data =>
loc_data);
```

Repetição de Código

Comando concorrente

```
ENTITY bit_a_bit_and  IS
  PORT(
    A, B : IN STD_LOGIC _VECTOR( 4 DOWNTO 0 );
    C : OUT STD_LOGIC _VECTOR( 4 DOWNTO 0 );
);
END bit_a_bit_and;

ARCHITECTURE repetition OF bit_a_bit_and IS
SIGNAL x0, x1, x2, x3, x4 : STD_LOGIC;
BEGIN
        Gen_1 : FOR i IN 0 TO 4 GENERATE
                x(I) <= A(i) AND B(i);
        END GENERATE;
        C <= X;
END repetition;
```

Repetição de Código: N Componentes

Comando concorrente

```
ARCHITECTURE teste OF teste IS

COMPONENT func2
  PORT( a0 : IN  std_logic;
        a1 : IN  std_logic;
        y  : OUT std_logic);
 END COMPONENT func2;

BEGIN
 G1 : FOR n IN (length-1) DOWNTO 0 GENERATE
   func2_N: func2  PORT MAP( a0 => sig1(n), a1 => sig2(n),
              y  => z(n));
  END GENERATE G1;
END test;
```

Repetição de Código

Comando sequencial

```
PROCESS (A)
BEGIN
        Z <= "0000";
        FOR I IN 0 TO 3 LOOP
                IF (A = I) THEN
                        Z(I) <= '1';
                END IF;
        END LOOP;
END PROCESS;
```