Sistemas Digitais

Aula 05 - VHDL – Parte II

Apresentação

Nesta aula, vamos continuar o nosso estudo sobre VHDL. Na aula anterior vimos uma breve introdução, a construção e estrutura característica do VHDL, algumas bibliotecas, operadores e tipos de dados. Agora, estudaremos os tipos de instruções mais utilizados.

Objetivos

Ao final desta aula, você será capaz de:

- Reconhecer os componentes básicos de programação em VHDL, mais especificamente as instruções.
- Elaborar um programa simples em VHDL.

Instruções

O VHDL tem similaridades com outras linguagens de programação, mas o VHDL está descrevendo um hardware enquanto, as linguagens de programação descrevem um software executado de forma sequencial. O VHDL é uma linguagem de descrição de hardware, ou seja, descreve, exemplifica o funcionamento de um hardware. Então, a programação não é sequencial e sim concorrente (Quando duas ou mais coisas acontecem ao mesmo tempo, em paralelo.).

Por exemplo, vamos analisar o seguinte código VHDL (Figura 1a), cujo circuito está representado na Figura 1b, onde temos 3 portas (duas AND e uma OR), o código em VHDL em vermelho e o código convencional em preto. No caso do VHDL, temos A recebendo (x4 AND x5) e B recebendo (x6 AND x7). Como estamos descrevendo o circuito mostrado na Figura 1b, percebemos que estas duas ações ocorrem de maneira paralela. Logo depois, a saída das portas AND vão para uma porta OR. Em VHDL dizemos que a ação de A e B estão ocorrendo paralelamente ou concorrentemente.

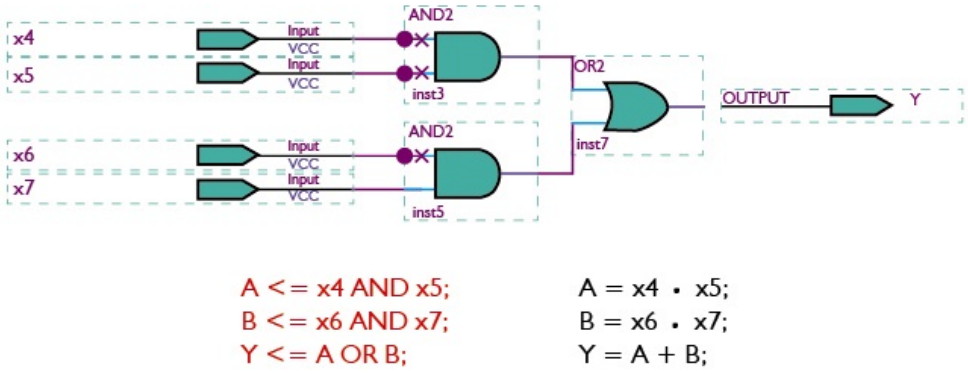
Em VHDL também podem existir ações e rotinas sequenciais, mas teremos que determinar explicitamente no código o que deve ser executado sequencialmente.

Figura 01 (a) - Código VHDL do circuito da Figura 1(b)

```
entity circuito is
   PORT(x4, x5, x6, x7: in std_logic;
      Y: out std_logic);
end circuito;

architecture arc_circuito of circuito is
signal A, B : std_logic;
begin
   A <= x4 and x5;
   B <= x6 and x7;
   Y <= A or B;
end arc_circuito;
```

Figura 01 (b) - Programação concorrente (VHDL) e sequencial



Atividade 01

1. Qual a diferença de programação de uma linguagem de descrição de hardware e uma linguagem como o Java, em respeito às instruções?

As instruções em VHDL podem ser divididas em alguns tipos:

- Atribuição de sinais concorrentes.
- Atribuição de sinais condicionais.
- Atribuição de sinais de seleção.
- Atribuição de processos.

Atribuição de Sinais Concorrentes

A sintaxe dos sinais concorrentes é a mais comum. O valor da expressão é transferido para o sinal alvo pelo operador “<=”. Pode-se descrever como:

Exemplos:

```
1 Soma <= (A xor B) xor Cin;
2 VaiUm <= (A and B);
3 Z <= (not X) or Y;
```

No código da Figura 2 vemos um exemplo que descreve uma porta NAND e uma OR utilizando atribuição de sinal concorrente.

Figura 02 - Exemplo de código com atribuição de sinal concorrente

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

ENTITY entidade_exemplo IS
PORT (
   A,B: IN STD_LOGIC;
   X,Y: OUT STD_LOGIC
);
END entidade_exemplo;

ARCHITECTURE arquitetura_exemplo OF entidade_exemplo IS
BEGIN
   X <= A NAND B;
   Y <= A OR B;
END arquitetura_exemplo;
```

Atribuição de Sinais Condicionais

Sobre a atribuição de um sinal condicional, podemos dizer que seria quando um sinal ou uma variável assume certo valor se por acaso a condição descrita for satisfeita. A sintaxe pode ser descrita como colocado no quadro abaixo:

```
1 sinal/variável <= expressão1 when condição1 else
2 expressão2 when condição2 else
3 expressão3 when condição3 else
4 expressão4;
```

O sinal ou variável recebe uma **expressão1** se uma **condição1** for satisfeita; uma **expressão2**, quando uma **condição2** é satisfeita; ou uma **expressão3** quando uma **condição3** é satisfeita. Pode-se ter quantas condições forem necessárias, não necessariamente apenas três como mostrado na sintaxe geral.

Vamos observar um exemplo (Figura 3) de um circuito Multiplexador 4x1 (seletor de 4 entradas para uma saída) que estudaremos com mais detalhes na Aula 06. Temos como saída "f" e como entradas "a", "b", "c", "d" e "sel", sendo este último um vetor de duas entradas. Quando o SEL é "11", a saída será o dado que está em "a"; quando o SEL é "10", a saída será o dado que está em "b"; quando o SEL é "01", a saída será o dado que está em "c" e quando o SEL é "00", a saída será o dado que está em "d". Caso nenhuma das possibilidades anteriores for alcançada, a saída será '0'.

Figura 03 - Exemplo de código com atribuição de sinal condicional

```
Entity mux4 is
port (a,b,c,d: in std_logic;
      sel: in std_logic_vector(1 downto 0);
      f: out std_logic);
end mux4;

architecture arc_mux4 of mux4 is
begin
        f   <=  a when (sel = "11") else
                b when (sel = "10") else
                c when (sel = "01") else
                d when (sel = "00") else
                '0';
end arc_mux4;
```

Atribuição de Sinais de Seleção

A sintaxe pode ser descrita por:

```
with sinal/variável select
   S <= expressão1 when condição1,
   expressão2 when condição2,
   expressão3 when others;
```

Parecido com a atribuição de um sinal condicional. A saída **S** receberá expressão1 se **sinal/variável** for igual a **condição1**; S receberá **expressão2** se **sinal/variável** for igual a **condição2**; para qualquer outro valor de **sinal/variável**, **S** receberá **expressão3**.

Vamos analisar o exemplo da Figura 4 que também está descrevendo um circuito MUX4:

Figura 04 - Exemplo de código com atribuição de sinais de seleção

```
Entity mux4 is
port (a,b,c,d: in std_logic;
      sel: in std_logic_vector(1 downto 0);
      f: out std_logic);
end mux4;

architecture arc_mux4 of mux4 is
begin
        with  sel  select
                f   <=  a when "11" ,
                        b when "10",
                        c when "01",
                        d when "00",
                        '0' when others;
end arc_mux4 ;
```

A entidade da Figura 4 está descrita da mesma maneira do exemplo anterior (Figura 3). A arquitetura terá uma estrutura diferente, na qual utilizaremos a sintaxe de sinais de seleção. Com (**with**) o SEL selecionado, a saída (f) recebe os dados de “a” quando (**when**) SEL = “11”, de “b” quando SEL = “10”, de “c” quando SEL = “01” e de “d” quando SEL = “00”. Caso contrário, a saída recebe ‘0‘ (função do **catch all**, ou seja, recebe ‘0’ para **qualquer outra condição que não foi descrita**).

Atribuição de Processos

Vimos que a execução de um código em VHDL é concorrente, mas podemos ter partes do código que são executadas de forma sequencial. Para isso, teremos que declarar dentro da arquitetura que aquela parte do código deve ser entendida como instruções sequenciais. Assim, temos que abrir o que chamamos de **processo**, e dentro de um processo teremos instruções sequenciais como laços (**loops**) de condição.

O código VHDL de um processo teria a seguinte construção:

```
1 nome_processo: process (lista- sensível de variáveis)
2   begin
3     {algoritmo - sequencial}
4   end process nome_processo;
```

Cada processo deve possuir um nome diferente. Seu nome estará em **nome_processo**. O processo também possuirá uma **lista sensível de variáveis**. Qualquer mudança de valor das variáveis declaradas nesta lista acarretará a execução do processo. Desta forma, é aconselhado que todas as variáveis de entrada utilizadas no processo devam estar declaradas.

Vamos analisar o exemplo da Figura 5 descrita logo abaixo.

Figura 05 - Exemplo de código utilizando processo

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

ENTITY entidade_exemplo IS
PORT (
   A,B,C: IN STD_LOGIC;
   X, Y : OUT STD_LOGIC
);
END entidade_exemplo;

ARCHITECTURE arquitetura_exemplo OF entidade_exemplo IS
BEGIN

   Processo1: process (A,B,C)
   begin
      X <= A NOR B;
      Y <= B AND C;
   end process Processo1;

END arquitetura_exemplo;
```

O circuito da Figura 5 possui 3 entradas A, B e C e duas saídas X, Y. Nele é criado um processo chamado **Processo1** cuja lista sensível de variáveis é **A,B,C**. O processo executa a operação NOR e AND de forma sequencial, ou seja, primeiro o NOR é executado e posteriormente o AND. Por fim, o processo é finalizado.

Importante!

Um código pode ter mais de um processo. A execução **entre processos** ocorre de forma paralela, enquanto que a execução dos comandos **dentro de um processo** ocorre de forma sequencial.

Dica!

Nunca esqueça de colocar nomes diferentes a processos. Também nunca esqueça de finalizar o processo escrevendo seu nome. Isto facilitará sua leitura e interpretação do código quando estiver estudando.

Atividade 02

1. Quais os tipos de sinais de atribuições que você aprendeu? Tente descrevê-los sucintamente.
2. O código que será executado dentro de um processo será sequencial ou concorrente?

Estudando o IF..THEN

Vamos estudar duas estruturas de dados condicionais muito encontradas também em outras linguagens. São elas o (1) If.. then, e (2) Case.

No If..Then, uma ação deve ser tomada baseado em algumas condições. A sintaxe da expressão é:

Figura 06 - Sintaxe do If..Then

```
If (condição 01) then
  {declaração 01 a ser executada}
elseif (condição 02) then
  {declaração 02 a ser executada}
else
  {declaração 03 a ser executada}
end if;
```

A *declaração 01* será executada se *condição 01* for verdadeira. Caso não seja, será verificada a *condição 02*. Se ela for verdadeira, *declaração 02* será executada. Caso nenhuma das duas condições anteriormente ditas foram verdadeiras, a *declaração 03* será executada.

Por exemplo, olhando para o exemplo da Figura 7, podemos deduzir que: se **sel** é igual a **0**, então **resultado** recebe **A**, se não **resultado** receberá **B**.

Figura 07 - Exemplo de trecho de um código VHDL usando a estrutura IF...THEN

```
if sel = 0 then
        resultado<= A;    -- executado se 'sel' for igual a 0
else
        resultado <= B;   -- executado se 'sel' for diferente de 0
end if;
```

Precisamos saber ainda que relacionado a expressão IF...THEN:

- Os parênteses são opcionais.
- Todo o **if** está associado ao **then**, com exceção do último **else**.
- A parte do **else** tem a função de **catch all**.
- As condições são avaliadas sequencialmente de cima para baixo.
- A condição satisfeita permite que a ação correspondente seja realizada.

Vamos analisar outro exemplo (Figura 8) que implementa a equação $S = A\overline{BC} + B$.

Figura 08 - Exemplo de um código VHDL usando a estrutura IF...THEN

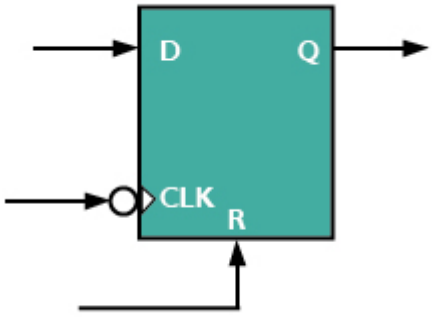
```
if (A='1' and B='0' and C='0')  then
   S <= '1'; -- executado se A=1, B=0 e C=0;
else
   if (B = '1') then
      S <= '1'; -- executado se B=1;
   else
      S <= '0';
   end if;
end if;
```

Para a condição de $A = 1$, $B = 0$ e $C = 0$, a saída recebe o valor 1, bem como para a condição de $B = 1$. Caso contrário (que é o último **else**) a saída recebe 0.

Perceba que o IF..THEN é intrinsecamente sequencial. As condições são sequencialmente verificadas uma após a outra e uma declaração só pode ser executada após uma condição verdadeira. Por conta de sua estrutura sequencial, um If..Then deve estar dentro de uma estrutura que explicite a leitura sequencial do código. Existem outras estruturas sequenciais em VHDL, mas até agora só estudamos o **process**. Desta forma, para até o momento, o If..Then deve estar dentro de um **process**. Vamos ver o código de um *flip-flop* D.

O *flip-flop* D é um circuito sequencial que você estudará nas aulas que ainda estão por vir. Teremos na saída o dado da entrada a cada vez que o relógio de entrada subir ou descer. Olhando para a Figura 9, temos como saída Q e como entradas: o D, o relógio (CLK) e o habilitador, R, que quando estiver em 1 o *flip-flop* D estará funcionando, caso contrário teremos a saída Q igual a zero.

Figura 09 - Flip-Flop D



Fonte: Bryan Mealy (2004).

Na Figura 10 vemos o código em VHDL do flip-flop D.

Figura 10 - Exemplo de código em VHDL de um circuito sequencial (flip-flop D) usando processo

```
library ieee;                           -- Declaramos as bibliotecas
use ieee.std_logic_1164.all;

entity FlipFlopD is                     -- Abrimos a entidade
port (D, R, clk: in std_logic;          -- Declaramos as portas de entrada e saída
      Q: out std_logic);
end FlipFlopD;                          -- Finalizamos a entidade

architecture exemplo of FlipFlopD is
begin                                   -- Abre a arquitetura com begin
        P2: process (D,clk)             -- Abre um processo, nome do processo P2
        begin                           -- Abre o processo com begin e descreve a expressão
                if (R = '0') then       -- Se R for igual a "0"
                        Q <= '0';       -- Então Q é igual a "0"
                elsif (falling_edge(clk)) then  -- Caso contrário na descida do clk
                        Q <= D;         -- Q vai receber o dado da entrada D
                else                    -- Senão
                        Q <= 0;         -- Q vai receber "0"
                end if;
        end process P2;                 -- Finaliza o processo
end exemplo;                            -- Finaliza a arquitetura
```

Atividade 03

1. Olhando para o código abaixo, qual será a expressão que resultou nesse código?

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

Entity ExemploIF is
        port (A, B: in std_logic;
              s: out std_logic);
end ExemploIF;

architecture exemplo of ExemploIF is
begin
        P3: process (A,B)
        begin
                if (A = '1' and B = '1') then
                        s <= '1';
                elsif (B = '0') then
                        s <= '0';
                else
                        s <= '0';
                end if;
        end process P3;
end exemplo;
```

Estudando o CASE

A sintaxe da expressão é esta que vemos na Figura 11:

```
case (expressão) is
   when condição 01 =>
      {declaração 01}
   when condição 02 =>
      {declaração 02}
   when others =>
      {declaração 03}
end case;
```

Figura 11 - Exemplo de trecho de um código VHDL usando a estrutura CASE

No caso do CASE, todas as expressões são avaliadas e a que for satisfeita, a declaração relacionada a condição, será executada. O último **when others** (quando outros) não é obrigado estar na sintaxe da expressão. Perceba que o CASE também tem sua estrutura intrisicamente sequencial e, portanto, deve estar dentro de um

process. Nada melhor do que um exemplo para analisarmos o funcionamento dessa expressão, não é?. Vamos analisar com a equação implementada para o CASE, $out1 = A\overline{BC} + ABC + \overline{A}BC$:

Figura 12 - Exemplo de um código VHDL usando a estrutura CASE

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

entity ExemploCase is
port (
   ABC: in std_logic_vector(2 downto 0);
   S  : out std_logic );
end ExemploCase;

architecture exemplo of ExemploCase is
begin
      P4: process (ABC)
      begin
         case (ABC) is
               when "100" => S <= '1';     -- Quando A.B̄.C̄
               when "011" => S <= '1';     -- Quando Ā.B.C
               when "111" => S <= '1';     -- Quando A.B.C
               when others => S <= '0';
         end case;
      end process P4;
end exemplo;
```

Atividade 04

1. Que estrutura de controle decide executar ou não executar uma instrução?
2. Que estrutura de controle decide qual de várias ações diferentes executar?

Construções de Laços

Depois de analisarmos várias estruturas de atribuições, os casos condicionais, agora, iremos ver uma estrutura também bastante utilizada para realizarmos iterações, ou seja, os *loops*.

Existem dois tipos de laços em VHDL: **for** e **while**. O comando **while** condicional funciona de forma similar ao comando **if then**. O laço é executado enquanto a condição presente no seu inicio for válida. A condição é verificada e se for satisfeita,

os comandos presentes no laço são executados, caso contrário, o laço é considerado completo e o comando passa para a instrução seguinte ao laço.

A sintaxe do *while loop* pode ser descrita como a seguir:

```
r1: while (condição) loop
   {declaração}
end loop r1;
```

Vamos analisar o exemplo a seguir: abrimos um processo (chamado "**proc**"), inicializamos uma variável (*variable*) "conta" do tipo inteiro inicializada com '0'. Quando o Nível for igual a '1', realizará a expressão (conta = conta + 1), quando Nível for diferente de '1', sairá do loop.

```
proc: process
   variable conta : integer := 0;
   begin
      while Nivel = '1' loop
            conta := conta + 1;
      end loop;
end process proc;
```

Veja que o comando ***while loop*** deve estar dentro de um **process** por ser uma operação sequencial. Veja também que foi criado uma variável dentro de **process** chamada *conta*. Esta é uma forma de declarar variáveis temporárias dentro de um processo. Você pode utilizar outros tipos de variáveis, como std_logic, bit, vetores, etc. É importante saber que esta variável pode ser lida e escrita, ou seja, pode receber valores (como variável de saída) e pode ser lida (como variável de entrada). Como ela foi declarada dentro do process, somente dentro dele ela existe.

O comando ***while loop*** repete-se enquanto a condição apresentada no seu início for satisfeita. Algumas vezes pode ser necessário repetir o laço por um número específico de vezes. Isso pode ser feito de forma mais conveniente com o comando **for**. Esse comando não usa nenhuma expressão *booleana*, mas sim um contador, e desde que o valor do contador esteja em certa faixa, o laço é executado. Ao término de cada execução do laço, o contador recebe o novo valor. O contador não precisa ser declarado e é tratado como uma constante que só existe dentro do laço.

Existe também a estrutura do comando *for*, que é dada por:

```
nome: for index in a_range loop
   {declaração}
end loop nome;
```

No exemplo a seguir, começamos um processo chamado “contar”, inicializamos uma variável do tipo inteiro chamado “Numeros” com o valor “0”. Começamos o laço com uma variável “temp” que irá contar de 0 até 3, incrementando a variável “Numeros”.

```
Contar : process (Dados)
   variable Numeros : integer := 0;
   begin
      for temp in 0 to 3 loop
      Numeros := Numeros +1;
   end loop;
end process Contar;
```

Atividade 05

1. Quais são as duas estruturas de laço?
2. Qual é a diferença entre as duas estruturas de laço?

Conclusão

Chegamos ao fim de nossa aula. Com essa aula você já é capaz de entender um código simples em VHDL, e também pode elaborar algumas construções. Na aula prática serão passados exercícios que lhe darão confiança em escrever algumas funções básicas como estas que foram apresentadas nos exemplos. Vamos lá!

Leitura Complementar

Se você quiser estudar mais um pouco e saber mais sobre a linguagem de VHDL, existem mais exemplos e informações sobre o VHDL nestes links.

- <www.vhdl.com.br>
- <https://www.youtube.com/watch?v=uxF_fSDH-YY>
- <http://esd.cs.ucr.edu/labs/tutorial/>.

Resumo

Nesta aula, você estudou noções básicas da linguagem de descrição de hardware, VHDL. Você viu as estruturas básicas, construções de instruções, processos e laços. Com a próxima aula prática, você terá a oportunidade de praticar os conceitos aprendidos, afinal programar só se aprende programando. Assim, para um melhor entendimento das aulas de VHDL, você terá que praticar.

Autoavaliação

1. Observe o circuito abaixo e responda qual é a resposta que melhor descreve o circuito:

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

Entity ExemploIF is
        port (A, B, C: in std_logic;
              s: out std_logic);
end ExemploIF;

architecture exemplo of ExemploIF is
begin
        P5: process (A,B,C)
        begin
                if (A = '1' and B = '0' and C = '0' ) then
                        s <= '1';
                elsif (B = '0') then
                        s <= '0';
                else
                        s <= 0;
                end if;
        end process P5;
end exemplo;
```

a. $S = \overline{\overline{ABC}} + B$

b. $S = \overline{\overline{BAC}} + B$

c. $S = ABC + B$

d. $S = \overline{\overline{BAC}} + \overline{B}$

2. Responda se é verdadeiro ou falso:

( ) As estruturas de laços são definidas pela expressão **for** e **while**.

( ) Existem duas estruturas básicas em VHDL, a entidade e a arquitetura.

( ) VHDL é uma linguagem de descrição de hardware e tem uma programação sequencial.

( ) Com sinais concorrentes utiliza-se “>=”.

a. (V)(V)(F)(F)

b. (V)(V)(V)(F)

c. (V)(F)(F)(F)

d. (F)(V)(F)(F)

3. Quais as estruturas de controle que você aprendeu?

4. Qual a necessidade de abrir um **process** no código em VHDL?

5. Dê um exemplo da utilização da estrutura “IF....THEN”.

Referências

ASHENDEN, Peter J. **VHDL Tutorial**. EDA CONSULTANT, ASHENDEN DESIGNS PTY. LTD. Disponível em: <www.ashenden.com.au>. Acesso em: 26 jun. 2012.

CASILIO, Leonardo; SARAIVA, Ivan. **VHDL aula 02**: Semântica de VHDL. [20-?].

GUERREIRO, Ana M. G. **Aulas de Circuitos Digitais**. Universidade Federal do Rio Grande do Norte, 2008

SPIEGEL, Jan Van der. **VDHL tutorial**. University of Pennsylvania. Department of Electrical and Systems Engineering. Disponível em: <http://www.seas.upenn.edu/~ese201/vhdl/vhdl_primer.html#_Toc526061341>. Acesso em: 26 jun. 2012.

TOCCI, Ronald; WIDNER, Neal S.; MOSS, Gregory L. **Sistemas Digitais**: princípios e aplicações. 10. ed. São Paulo: Prentice Hall,2007.

UNIVERSIDADE FEDERAL DE ITAJUBÁ. **Tutorial de VHDL**. Grupo de microeletrônica. [20-?].