Simulação em VHDL

Prof. MSc. André Macário Barros

25/03/2019

Agenda

- Simulações e ferramentas de EDA
- Arquivos de *testbench*
- Mapeamento entre arquivos/circuitos em simulações
- A declaração PROCESS e escopos para declarações sequenciais em simulações
- Dois exemplos
- Cinco exercícios

A Simulação

- Já aprendemos como especificar circuitos combinacionais básicos em VHDL e também como sintetizá-los em uma ferramenta de EDA (*Electronic Design Automation*), no caso o ISE da Xilinx
- No entanto, como saber se o circuito gerado pelo código executa o que se deseja?
- Veremos como resolver isto através dos arquivos de *testbench* para simulações que serão apresentados neste encontro

Arquivo de *testbench*

- O que devemos fazer para simular/testar o código VHDL de um circuito é criar um arquivo de *testbench*
- Tal arquivo, também em VHDL, por meio de uma relação explicitamente declarada com o código do circuito a ser testado, especifica estímulos a serem nele aplicados.
- E, por meio do recurso de simulação contido na ferramenta de EDA, é possível analisar-se quais são as respectivas respostas que o circuito sob simulação fornece aos estímulos impostos.

Exemplo 1

- Tomemos inicialmente o Exemplo 4 do material da aula "Introdução à VHDL": o comparador de 1 *bit*

a	*b*	*y*
0	0	1
0	1	0
1	0	0
1	1	1

$y = a'b' + ab$

Exemplo 1 – código VHDL do comparador de 1 *bit*

- Solução ($y = a'b' + ab$):


```
01-library ieee;
02-use ieee.std_logic_1164.all;

03-entity comparador is
04-port(a, b: in std_logic;
05-         y: out std_logic);
06-end comparador;

07-architecture arq_comp of comparador is
08-begin
09-    y <= (not(a) and not(b)) or (a and b);
10-end arq_comp;
```

Exemplo 1 – arquivo de *testbench* do comparador para simulação

```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-entity tb_comparador is
04-end tb_comparador;
05-architecture arq of tb_comparador is
06-signal ta, tb, ty: std_logic;
07-begin
08-   uut: entity work.comparador(arq_comp)
09-   port map(a => ta, b => tb, y => ty);
10-   process begin
11-      ta <= '0'; tb <= '0'; wait for 10 ns;
12-      ta <= '0'; tb <= '1'; wait for 10 ns;
13-      ta <= '1'; tb <= '0'; wait for 10 ns;
14-      ta <= '1'; tb <= '1'; wait for 10 ns;
15-   end process;
16-end arq;
```

Exemplo 1 – Descrição do *testbench*

- Partamos agora para a descrição de cada uma das linhas de código do arquivo de *testbench*

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
01-library ieee;
02-use ieee.std_logic_1164.all;
```

- Linhas 01 e 02: o arquivo de *testbench* também opera com sinais std_logic, além de outros valores. Portanto, necessita da biblioteca/pacote/unidade declaradas nestas linhas.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
03-entity tb_comparador is
04-end tb_comparador;
```

- Linhas 03 e 04: o arquivo de *testbench* não contém entradas e saídas. Logo, sua entity é vazia.
- Observe que o identificador, **`tb_comparador`**, é diferente do identificador do código VHDL do circuito propriamente dito (**`comparador`**).

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
03-entity tb_comparador is
04-end tb_comparador;
```

- Linhas 03 e 04: pode-se atribuir um identificador qualquer à entity do arquivo de *testbench*, da mesma forma que para o identificador de sua architecture, desde que obedeçam às regras já estudadas (26 caracteres, etc). No entanto, nomear o identificador da entity como sendo "tb_xxxx", onde xxxx corresponda ao circuito a ser testado, ajuda a identificar e relacionar tais arquivos em um sistema mais complexo sistema.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
05-architecture arq of tb_comparador is
...
07-begin
...
16-end arq;
```

- Linhas 05, 07 e 16: a architecture do arquivo de *testbench* é declarada e finalizada normalmente conforme já aprendido.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
05-architecture arq of tb_ccto is
06-signal ta, tb, ty: std_logic;
07-begin
...
16-end arq;
```

- Linha 06: os três sinais declarados nesta linha serão associados às entradas e saídas do circuito a ser testado, neste caso as entradas *a* e *b*, da mesma forma que a saída *y*.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
08-uut: entity work.comparador(arq_comp)
09-port map(a => ta, b => tb, y => ty);
```

- Linhas 08 e 09: é através do conteúdo destas linhas que ocorre o estabelecimento da relação entre o arquivo de *testbench* e o arquivo que corresponde ao circuito a ser testado.
- **uut** é um identificador qualquer seguido de dois pontos necessário à sintaxe do mapeamento.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
08-uut: entity work.comparador(arq_comp)
09-port map(a => ta, b => tb, y => ty);
```

- Linha 08: entity work.xxx(yyy) determina que seja associado o arquivo de *testbench* que está sendo escrito com o circuito cuja entity chama-se xxx e architecture chama-se yyy.
- **work** informa à ferramenta de EDA (no caso o ISE) que tal circuito xxx/yyy encontra-se na biblioteca de trabalho, dentro do próprio arquivo de projeto no qual está sendo desenvolvido o arquivo de *testbench*.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
08-uut: entity work.comparador(arq_comp)
09-port map(a => ta, b => tb, y => ty);
```

- Linha 09: é a linha que estabelece o mapeamento propriamente dito entre as entradas **a** e **b** com os sinais **ta** e **tb** respectivamente, da mesma forma que entre a saída **y** e o sinal **ty**.
- **port map** é palavra-chave obrigatória. E o operador "=>", além de realmente ser para a direita, pode ser lido como "a entrada **a** deve ser mapeada para o sinal **ta**" e da mesma forma para os sinais **tb** e **ty**.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
08-uut: entity work.comparador(arq_comp)
09-port map(a => ta, b => tb, y => ty);
```

- Linha 09: há um segundo tipo de mapeamento, chamado mapeamento posicional, que pode ser usado. A diferença deste mapeamento para o apresentado no código é que os sinais de entrada e saída do circuito a ser testado não são mostradas, ficando desta forma;
  - `port map(ta, tb, ty);`

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
10-process begin
11-...
...
15-end process;
```

- Linhas 10 e 15: estas duas linhas delimitam um escopo de declarações sequenciais. Portanto, todos as declarações que existirem nas linhas 11 à 14 serão executados uma após a outra, como ocorre nas linguagens de programação.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
11-ta <= '0'; tb <= '0'; wait for 10 ns;
12-ta <= '0'; tb <= '1'; wait for 10 ns;
13-ta <= '1'; tb <= '0'; wait for 10 ns;
14-ta <= '1'; tb <= '1'; wait for 10 ns;
```

- Linhas 11 a 14: estas linhas estão compreendidas dentro de um escopo de process, portanto, as declarações ocorrem uma após a outra.
- Lembrando: se tais linhas não estivessem dentro de do escopo de um process, tais declarações seriam concorrentes uma da outra e o teste não poderia ser feito.

Exemplo 1 – Descrição do *testbench*

- Explicações:

```
11-ta <= '0'; tb <= '0'; wait for 10 ns;
12-ta <= '0'; tb <= '1'; wait for 10 ns;
13-ta <= '1'; tb <= '0'; wait for 10 ns;
14-ta <= '1'; tb <= '1'; wait for 10 ns;
```

- É aqui onde ocorre o estímulo ao circuito a ser testado. Como queremos saber se o código VHDL processa de fato a tabela-verdade de um comparador, basta atribuírmos os valores lógicos “00”, “01”, “10” e “11” aos sinais ta e tb respectivamente.

Exemplo 1 – Descrição do *testbench*

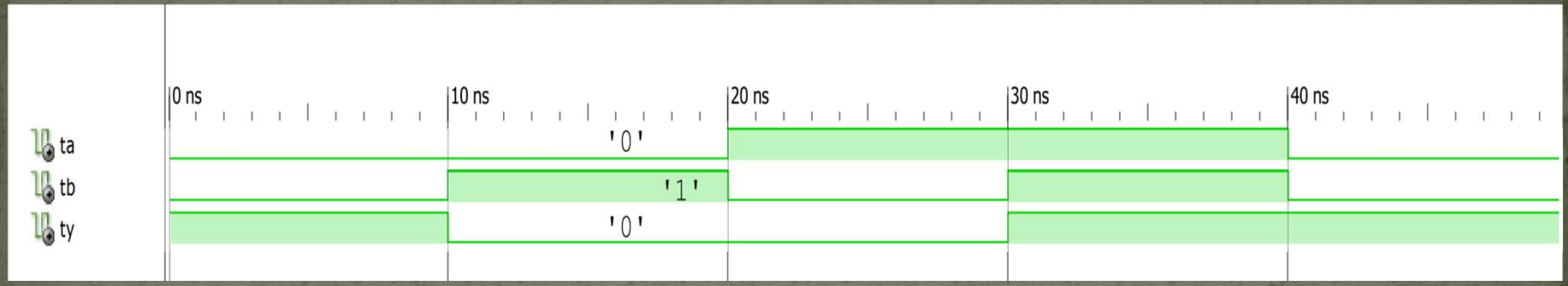
- Explicações:

```
11-ta <= '0'; tb <= '0'; wait for 10 ns;
12-ta <= '0'; tb <= '1'; wait for 10 ns;
13-ta <= '1'; tb <= '0'; wait for 10 ns;
14-ta <= '1'; tb <= '1'; wait for 10 ns;
```

- Porém, observe que após cada estímulo aos sinais ta e tb, é dado um tempo de espera através da declaração **wait for xx ns**. Isto é necessário para que o sinal passe pelo circuito e atinja a saída y. Ocorrendo isto, o sinal ty (que está mapeado para a saída y) recebe o resultado. Somente após isto é que novo estímulo aos sinais ta e tb é dado.

Exemplo 1 – A simulação

- Como resultado da simulação através do código do circuito comparador estimulado por este arquivo de *testbench*, temos as formas de onda a seguir (compare-as com a tabela-verdade e os estímulos):



a	*b*	*y*
0	0	1
0	1	0
1	0	0
1	1	1

```
11-ta <= '0'; tb <= '0'; wait for 10 ns;
12-ta <= '0'; tb <= '1'; wait for 10 ns;
13-ta <= '1'; tb <= '0'; wait for 10 ns;
14-ta <= '1'; tb <= '1'; wait for 10 ns;
```

Exemplo 2

- Façamos agora um exemplo com duas saídas: um circuito *half-adder*.

a	*b*	*s*	*c*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s = a \oplus b$$
$$c = a \ . \ b$$

Exemplo 2 – Código VHDL do *half-adder*

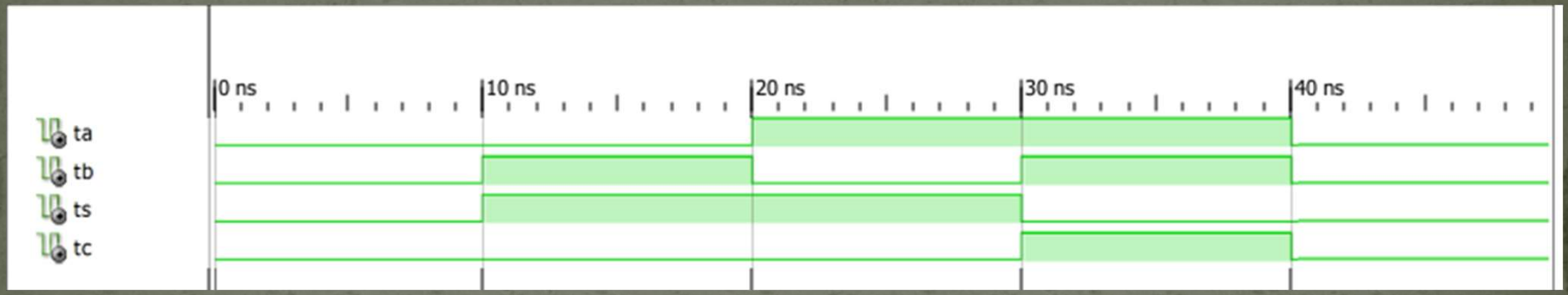
- Solução ($s = a \oplus b$; $c = a \ . \ b$):

```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-entity ccto is
04-port(
05-    a, b: in std_logic;
06-     s, c: out std_logic);
07-end ccto;
08-architecture arq of ccto is
09-begin
10-    s <= a xor b;
11-    c <= a and b;
12-end;
```

Exemplo 2 – Código VHDL do arquivo de *testbench*


```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-entity tb_ccto is
04-end tb_ccto;
05-architecture arq of tb_ccto is
06-signal ta, tb, ts, tc: std_logic;
07-begin
08-   uut: entity work.ccto(arq)
09-   port map(a => ta, b => tb, s => ts, c => tc);
10-   process begin
11-      ta <= '0'; tb <= '0'; wait for 10 ns;
12-      ta <= '0'; tb <= '1'; wait for 10 ns;
13-      ta <= '1'; tb <= '0'; wait for 10 ns;
14-      ta <= '1'; tb <= '1'; wait for 10 ns;
15-   end process;
16-end arq;
```

Exemplo 2 – A simulação



a	*b*	*s*	*c*
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
11-ta <= '0'; tb <= '0'; wait for 10 ns;
12-ta <= '0'; tb <= '1'; wait for 10 ns;
13-ta <= '1'; tb <= '0'; wait for 10 ns;
14-ta <= '1'; tb <= '1'; wait for 10 ns;
```

Exercícios

- Desenvolva os cinco arquivos de *testbench* para os exercícios propostos no material da aula “Introdução à VHDL”

Referências

- CHU, Pong P. **FPGA Prototyping by VHDL examples**. New Jersey (NJ): John Wiley & Sons, 2008. Há 8 exemplares disponíveis na biblioteca. Número de chamada: 621.395 C559f