Sinais versus Variáveis

Prof. MSc. André Macário Barros

15/04/2019

Sinais versus variáveis dentro de código sequencial sequencial

- Este material tem por objetivo demonstrar a você a diferença prática entre sinais e variáveis.
- Para tal um exemplo com multiplexadores será abordado neste material.

Exemplo 1
Multiplexador "2x1"

- Consideremos aqui um multiplexador "2x1" que opere de acordo com a Figura 1 e a Tabela 1.

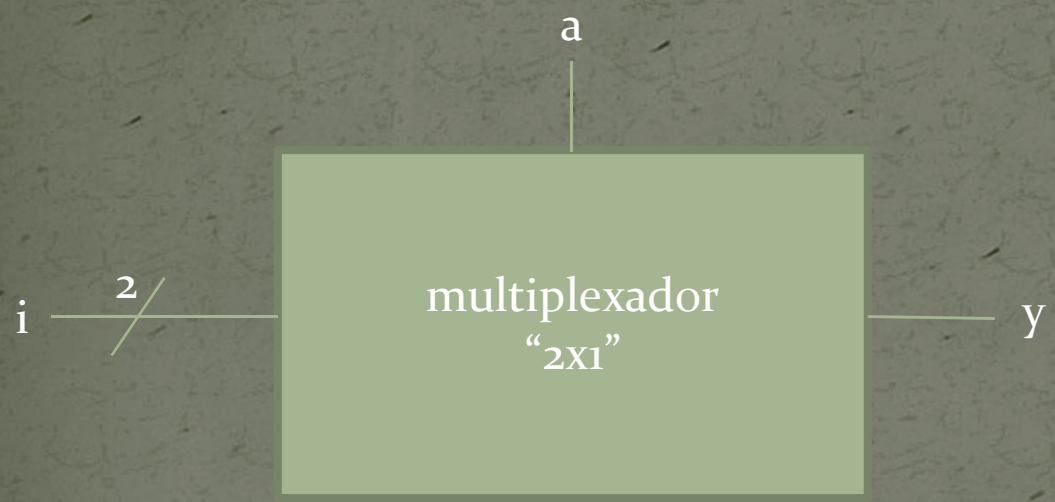


Figura 1

Tabela 1

a	*y*
0	i_0
1	i_1

Exemplo 1
Multiplexador "2x1"

- O código pode ser desenvolvido por meio de declarações condicionais concorrentes ou através dos conceitos da eletrônica digital clássica.
- As duas versões estão apresentadas no código a seguir, sendo que a versão tradicional está com a linha comentada (linha 10).

Exemplo 1
Decodificador "1-hot" de 2 bits – versão 1

```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-entity ccto is
04-port(a: in std_logic;
05-     i: in std_logic_vector(1 downto 0);
06-     y: out std_logic);
07-end entity;
08-architecture arq of ccto is
09-begin
10-   --y <= (not(a) and i(0)) or (a and i(1));
11-   y <= i(0) when a='0' else
12-        i(1);
13-end arq;
```

Exemplo 1
Multiplexador "2x1"

- Apresentaremos o código de *testbench* para este código por dois motivos:
  - Como lidar com um escalar a ser simulado dentro de um loop do process
  - E vamos demostrar os efeitos de uma simulação em que um sinal receba um valor dentro do escopo de um código sequencial (process):
    - de uma variável
    - e de outro sinal
- Segue nos dois slides a seguir o referido código.

Exemplo 1 – Multiplexador "2x1" *testbench* com variável – parte #1/2

```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-use ieee.numeric_std.all;
04-entity tb_ccto is
05-end tb_ccto;
06-architecture arq of tb_ccto is
07-signal ta, ty: std_logic;
08-signal ti: std_logic_vector(1 downto 0);
09-begin
10-   uut: entity work.ccto(arq)
11-   port map(a => ta, i => ti, y => ty);
```

Exemplo 1 – Multiplexador "2x1" *testbench* com variável – parte #2/2

```
12-process
13- variable i, j: integer;
14- variable ta_vec: std_logic_vector(0 downto 0);
15-begin
16- for i in 0 to 1 loop
17-  for j in 0 to 3 loop
18-   ta_vec := std_logic_vector(to_unsigned(i, 1));
19-   ta <= ta_vec(0);
20-   ti <= std_logic_vector(to_unsigned(j, 2));
21-   wait for 10 ns;
22-  end loop;
23- end loop;
24-end process; end arq;
```

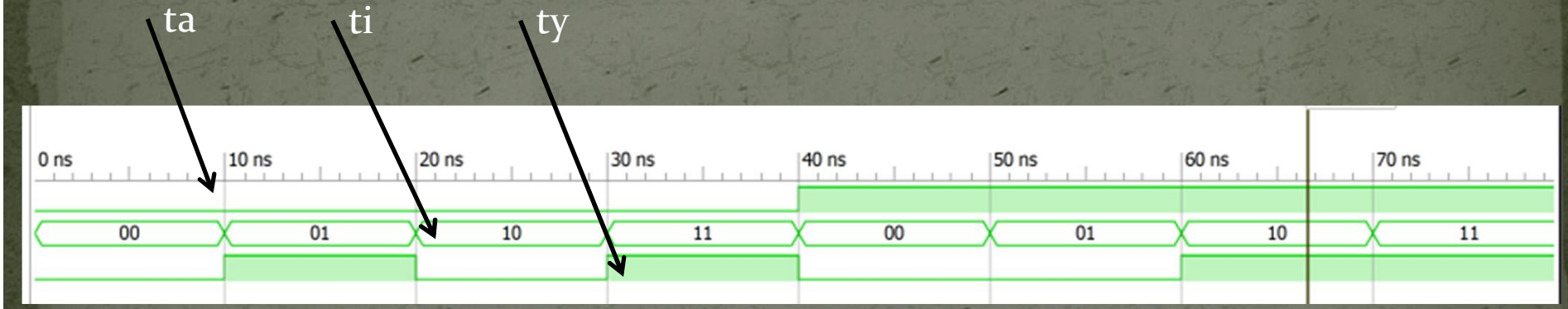
Exemplo 1 – Multiplexador "2x1" *testbench* com variável

- Observe que este código precisa simular valores que um escalar (**`std_logic`**) possa receber, que é o **`ta`**. O sinal **`ta`** é responsável por simular a entrada de endereçamento **`a`**, também escalar.
- Para que isto ocorra, ta precisa receber o valor que a função **`std_logic_vector`** retorna. Porém o VHDL não permite por serem tipos de dados incompatíveis.
- Por este motivo precisaremos de um vetor de uma dimensão **`(0 downto 0)`**, representado por **`ta_vec`**.

Exemplo 1 – Multiplexador "2x1" *testbench* com variável

- Nesta versão com variável, o arquivo de *testbench* empregou **`ta_vec`**, declarado como variável, na linha 14.
- Desta forma, é possível atribuir-se o valor que a função **`std_logic_vector`** retorna na linha 18 para **`ta_vec`**.
- Em seguida, o escalar da posição **`0`** de **`ta_vec`** é atribuído finalmente para **`ta`** na linha 19 e o loop prossegue sem problemas.
- Observe que, na linha 18 a variável **`ta_vec`** recebe o valor que deva ser atribuído à mesma com **`' :=`**
- O resultado da simulação empregando-se variável está mostrado a seguir.

Exemplo 1 – Multiplexador "2x1" *testbench* com variável



- Observe nesta simulação que **`ta`** apresenta imediatamente o valor '0' no período que vai de 0 a 10 ns. Tal valor foi obtido por meio da atribuição do conteúdo da variável **`ta_vec`**.

Exemplo 1 – Multiplexador "2x1" *testbench* com sinal

- Passemos agora à versão do *testbench* com o vetor auxiliar **`ta_vec`** empregado na forma de sinal.
- O código será apresentado nos dois slides a seguir.

Exemplo 1 – Multiplexador "2x1" *testbench* com sinal – parte #1/2

```
01-library ieee;
02-use ieee.std_logic_1164.all;
03-use ieee.numeric_std.all;
04-entity tb_ccto is
05-end tb_ccto;
06-architecture arq of tb_ccto is
07-signal ta, ty: std_logic;
08-signal ti: std_logic_vector(1 downto 0);
09-signal ta_vec: std_logic_vector(0 downto 0);
10-begin
11-   uut: entity work.ccto(arq)
12-   port map(a => ta, i => ti, y => ty);
```

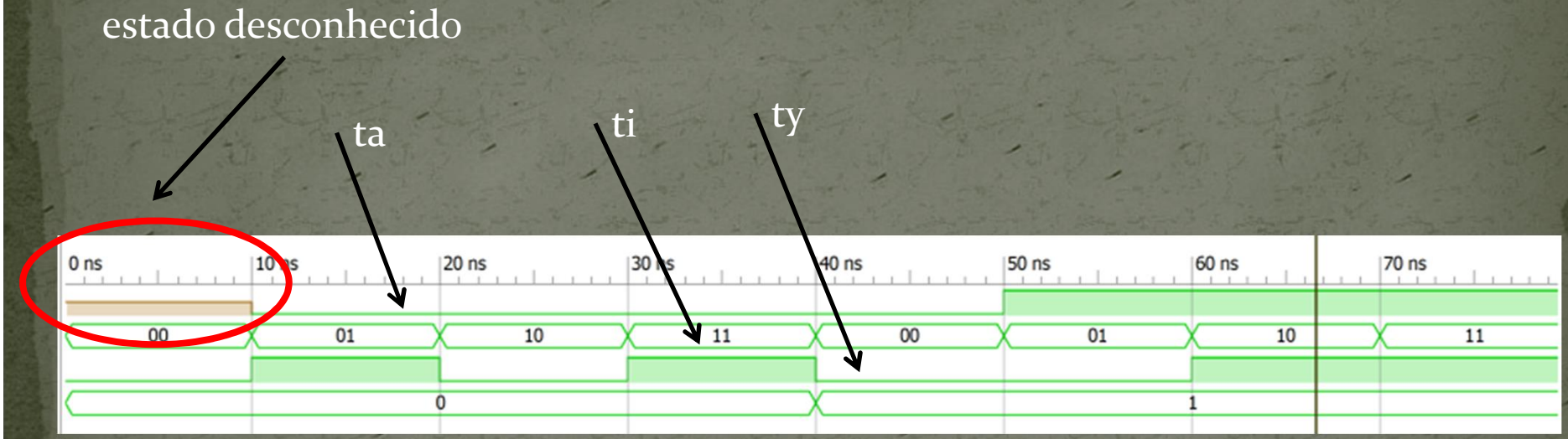
Exemplo 1 – Multiplexador "2x1" *testbench* com sinal – parte #2/2

```
13-process
14- variable i, j: integer;
15-begin
16- for i in 0 to 1 loop
17-  for j in 0 to 3 loop
18-   ta_vec <= std_logic_vector(to_unsigned(i, 1));
19-   ta <= ta_vec(0);
20-   ti <= std_logic_vector(to_unsigned(j, 2));
21-   wait for 10 ns;
22-  end loop;
23- end loop;
24-end process; end arq;
```

Exemplo 1 – Multiplexador "2x1" *testbench* com sinal

- Nesta versão com variável, o arquivo de *testbench* empregou **`ta_vec`**, declarado como sinal, na linha 09.
- Desta forma, é possível atribuir-se o valor que a função **`std_logic_vector`** retorna na linha 18 para **`ta_vec`**.
- Em seguida, o escalar da posição **`0`** de **`ta_vec`** é atribuído finalmente para **`ta`** na linha 19 e o loop prossegue sem problemas.
- Observe que, na linha 18 o sinal **`ta_vec`** recebe o valor que deva ser atribuído ao mesmo com **`'<='`**
- O resultado da simulação empregando-se variável está mostrado a seguir.

Exemplo 1 Multiplexador "2x1" *testbench* com sinal



- O sinal **`ta_vec`** recebeu um valor e, logo em seguida, precisou passar tal valor para **`ta`**
- O estado desconhecido em **`ta`** de 0 a 10 ns deve-se ao fato de **`ta`** só receber do sinal **`ta_vec`** algum valor no final da primeira iteração do loop.

Sinal versus variável dentro de código sequencial

- Através dos resultados destas duas simulações, esperamos que você tenha concluído que:
  - Um sinal que receba um conteúdo proveniente de outro sinal estando ambos dentro de um escopo sequencial delimitado por process, só receberá a atualização de tal atribuição ao final da iteração
  - Caso você não queira tal tipo de comportamento por parte do circuito, você deverá utilizar variáveis para atribuírem seus conteúdos aos sinais de destino para que recebam atualizações imediatas.

Referências

- Volnei Pedroni. **Eletrônica digital moderna e VHDL**. Elsevier, Rio de Janeiro, 2010.
  - Capítulos 19, 20 e 21
  - Há 12 exemplares na biblioteca
  - Número de chamada: 621.392 P372e