VHDL

Sequencial (process)

Rafael Corsi Ferrão - IMT

rafael.corsi@maua.br
http://www.maua.br

12 de outubro de 2014

Conteúdo

Conteúdo

Definição

Comandos sequencias são avaliados de maneira sequencial, na ordem que aparecem no código.

A descrição de uma lógica concorrente é alocado em uma região específica de código, chamada **PROCESS**. Os comandos sequenciais são:

- Atribuição de valor (sinal e **variável**)
- IF
- CASE
- WAIT

Definição

A região definida como sendo um processo (process) permite a execução de comandos sequencias.

A região de processo é definida por duas partes:

- lista de sensibilidade
- comandos sequenciais

A lista de sensibilidade é usada para indicar que o processo deve ser executado somente quando algum dos sinais sofreu alteração no seu valor.

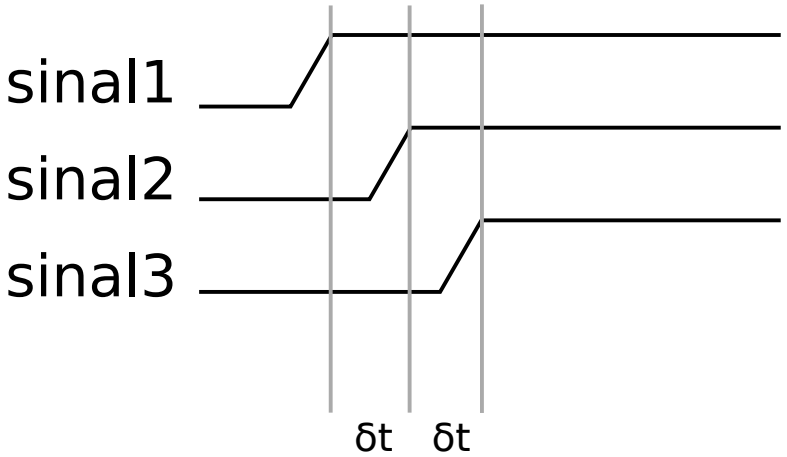
```
rotulo : PROCESS (lista de sensibilidade)
    -- Declaracao de sinais e variaveis
BEGIN
    -- Parte referente aos comandos sequencias
    -- comando
    -- comando
END PROCESS rotulo;
```

Exemplo:

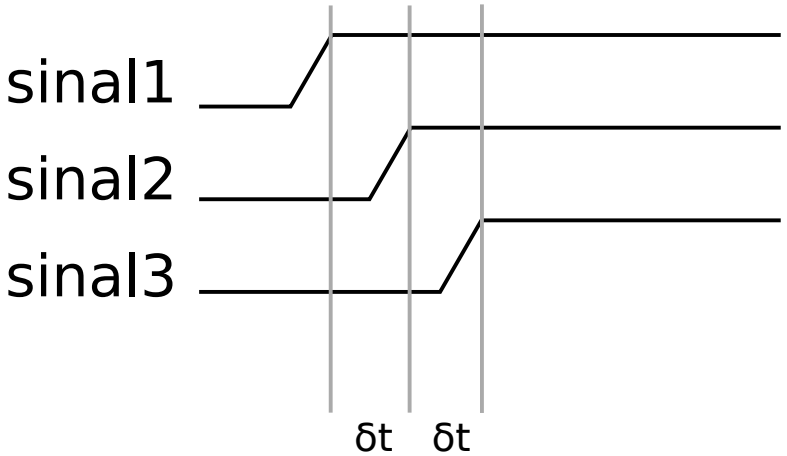
```
ex1 : PROCESS (sinal1, sinal2, sinal3)
BEGIN
    sinal2 <= sinal1;
    sinal3 <= sinal2;
END PROCESS ex1;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity sequencial is
end sequencial;

architecture bhv of sequencial is

    signal sinal1, sinal2, sinal3 : std_logic;

begin
ex1 : PROCESS (sinal1, sinal2, sinal3)
BEGIN
   sinal2 <= sinal1;
   sinal3 <= sinal2;
END PROCESS ex1;

end bhv;
```

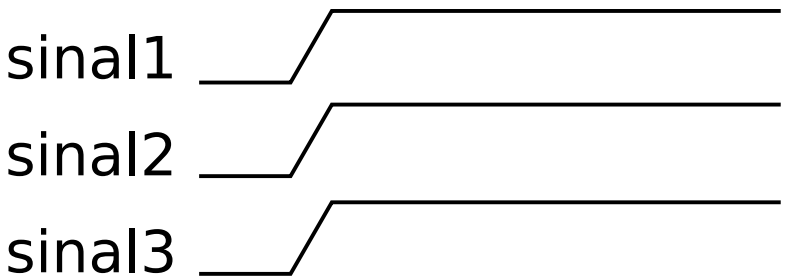
Com o exemplo, obtemos a seguinte carta de tempo:



Com o exemplo, obtemos a seguinte carta de tempo:



Mas $\delta t = 0$:

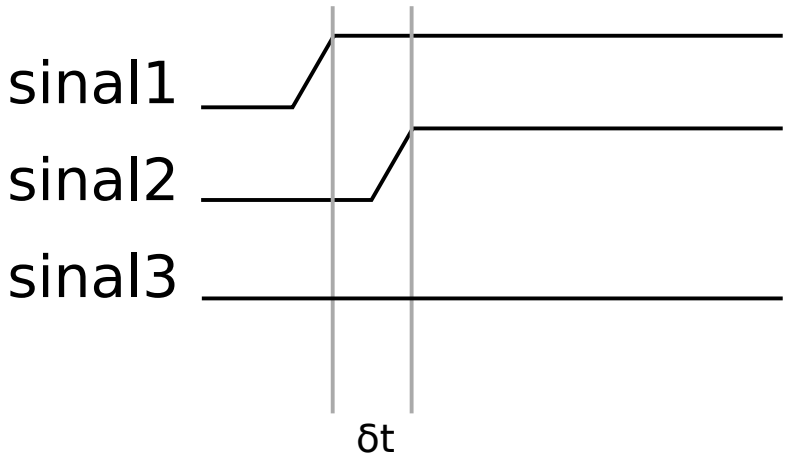


Importante os sinais que fazem parte da lista de sensibilidade, imagine a situação em que o sinal2 não é listado:

```
ex2 : PROCESS (sinal1, sinal3)
BEGIN
   sinal2 <= sinal1;
   sinal3 <= sinal2;
END PROCESS ex2;
```

Importante os sinais que fazem parte da lista de sensibilidade, imagine a situação em que o sinal2 não é listado:

```
ex2 : PROCESS (sinal1, sinal3)
BEGIN
   sinal2 <= sinal1;
   sinal3 <= sinal2;
END PROCESS ex2;
```



Em um processo, somente a última atribuição é valida.

```
ex2 : PROCESS (sinal1, sinal2)
BEGIN
   sinal1 <= '1';
   sinal2 <= sinal1;
   sinal2 <= '0';
END PROCESS ex2;
```

Implica que o **sinal1 = '1'** e **sinal2 = '0'**.

Como evitar que um valor seja atribuído só no final ?

Definição

Variável é um objeto (armazena valor) empregado em regiões de código sequencial (processo) porém a atribuição do valor acontece de forma imediata no código.

Pontos importantes :

- A declaração de variáveis ocorrem antes do **BEGIN** e são acessíveis somente dentro do processo.
- diferente do sinal, para atribuir um valor a uma variável usamos := ao invés de <=
- uma variável pode ser de qualquer tipo (inteiro, std_logic, . . .)
- o sinal só é visível dentro do processo que foi declarado
- um sinal pode receber uma variável e uma variável pode receber um sinal

```
label : PROCESS (lista_de_sensibilidade)
-- Declaracao de variaveis
  variable var1 : std_logic;
  variable var2 : integer range 0 to 100;
  variable var3 : unsigned(8 downto 0) := (others => '0');

  signal sig1 : unsigned(10 downto 0) := (others => '0');
BEGIN
-- seccao de codigo sequencial

    var1 := '1';
    var2 := 54;
    var3 := to_unsigned(var2, 8);

    sig1(8 downto 0) <= var3;

END PROCESS label;
```

```
entity mux is
port(
  i0, i1, i2, i3 : in std_logic;
  s0, s1         : in std_logic;
  ox             : out std_logic );
end mux;

architecture bhv of mux is
begin

 PROCESS (io, i1, i2 , i3)
   -- Declaracao de variaveis
   variable int0, int1, int2, int3 : std_logic;

 BEGIN
   -- seccao de codigo sequencial
   int0 := i0 AND NOT s1 AND NOT s0;
   int1 := i1 AND NOT s1 AND s0;
   int2 := i2 AND s1 AND NOT s0;
   int3 := i3 AND s1 AND  s0;

   ox   := int0 or int1 or int2 or int3;
 END PROCESS;
end mux;
```

```
entity mux is
port(
  i0, i1, i2, i3 : in std_logic;
  s0, s1         : in std_logic;
  ox             : out std_logic );
end mux;

architecture bhv of mux is
begin

 PROCESS (io, i1, i2 , i3)
   -- Declaracao de variaveis
   variable int0, int1, int2, int3 : std_logic;

 BEGIN
   -- Erro, as variaveis ainda nao foram definidas
   ox   := int0 or int1 or int2 or int3;

   int0 := i0 AND NOT s1 AND NOT s0;
   int1 := i1 AND NOT s1 AND s0;
   int2 := i2 AND s1 AND NOT s0;
   int3 := i3 AND s1 AND  s0;

 END PROCESS;
end mux;
```

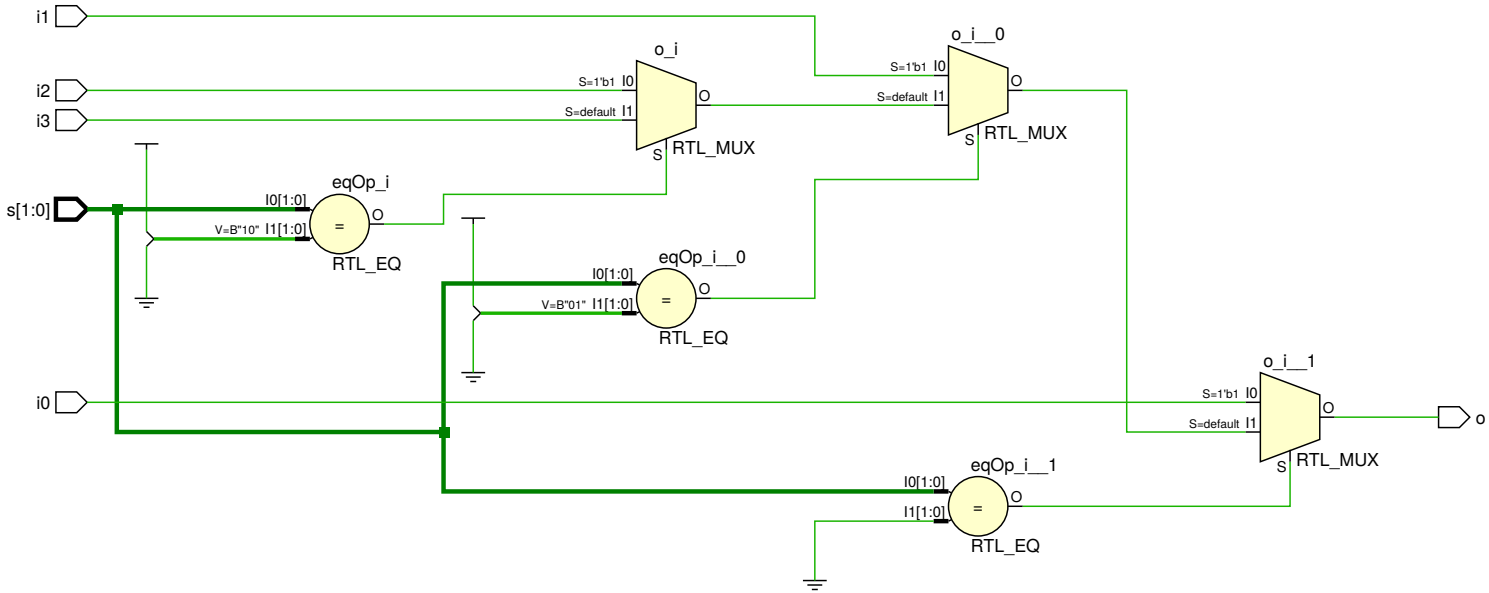
Definição

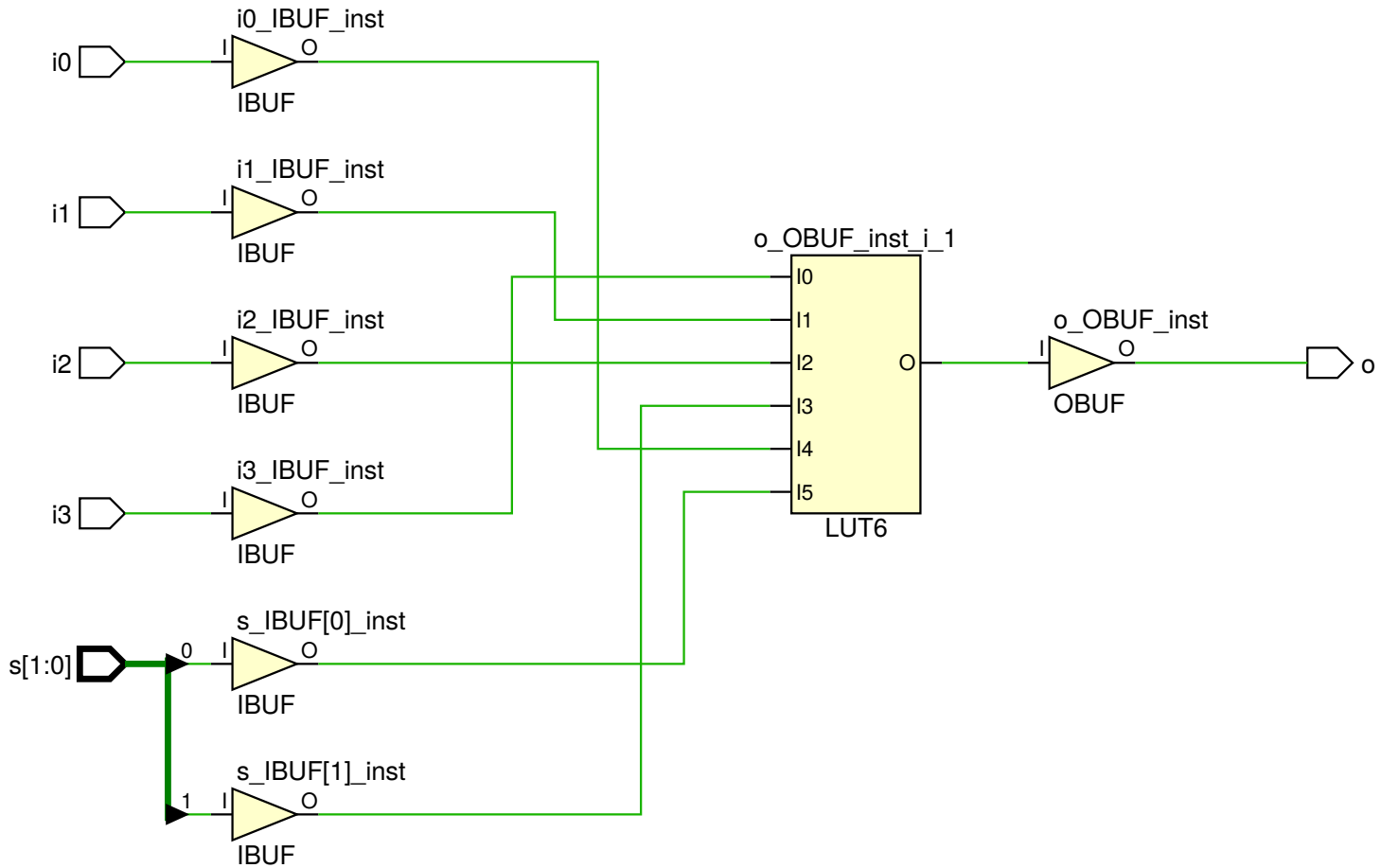
Permite a execução condicional de um ou mais comandos sequências

A ordem de execução respeita a ordem de aparição no código. A utilização é ilustrada a seguir:

```
IF condicao_1 THEN
   -- comandos 1
ELSIF condicao_2 THEN
   -- comandos 2
ELSIF condicao_3 THEN
   -- comandos 3
ELSE
   -- comandos restantes;
END IF;
```

```
ENTITY mux IS
   PORT(
      i0, i1, i2, i3 : IN STD_LOGIC; -- entradas
      s : IN STD_LOGIC_VECTOR(1 downto 0); -- selecao
      o : OUT STD_LOGIC -- saida
   );
END mux;

ARCHITECTURE bhv OF mux IS
BEGIN
   PROCESS(i0,i1,i2,i3,s)
   BEGIN
    IF s = "00" THEN
       o <= i0;
    ELSIF s="01" THEN
       o <= i1;
    ELSIF s="10" THEN
       o <= i2;
    ELSE
       o <= i3;
    END IF;
   END PROCESS;
END bhv;
```





Definição

Permite a execução condicional de um ou mais comandos sequências. Cada condição define um ou mais comandos sequências a serem executados.

As condições devem ser mutualmente exclusivas, e cada condição pode executar um ou mais comandos.

```
CASE expressao IS
   WHEN condicao_1                => comandos_a;
   WHEN condicao_2                => comandos_b;
   WHEN condicao_3 | condicao_4   => comandos_c;
   WHEN condicao_5 OR condicao_6  => comandos_d;
   WHEN OTHERS                    => comandos_e;
END CASE;
```

CASE WHEN
Exemplo - MUX

```
ENTITY mux_2 IS
   PORT(
      i0, i1, i2, i3 : IN STD_LOGIC; -- entradas
      s : IN STD_LOGIC_VECTOR(1 downto 0); -- selecao
      o : OUT STD_LOGIC -- saida
   );
END mux_2;

ARCHITECTURE bhv OF mux_2 IS
BEGIN

   PROCESS(i0,i1,i2,i3,s)
   BEGIN
       CASE s IS
            WHEN "00"   => o <= i0;
            WHEN "01"   => o <= i1;
            WHEN "10"   => o <= i2;
            WHEN OTHERS => o <= i3;
       END CASE;
   END PROCESS;

END bhv;
```