Laboratório de Lógica Digital

Prática VI

Comandos VHDL:

Comando IF-ELSE

Este comando permite a execução condicional de um ou mais comandos seqüenciais. O comando IF inicia a lista de condições, e pode ser seguido do comando ELSIF contendo também, condições a serem verificadas. Se nenhuma das condições forem verdadeiras e existir uma cláusula ELSE, o conjunto de comandos que segue será executado. E se não houverem nenhuma condição verdadeira e nem a cláusula ELSE, nenhum comando será executado. Em uma cadeia de IF ELSEs, as condições são dispostas em uma prioridade onde o primeiro IF define a condição de maior prioridade. No código 1, temos um IF que testa se o sinal "a" está em uma borda de subida (a='1' and a'event), caso a condição seja verdadeira, é atribuído o sinal de "a" em "ta1" e o sinal de "b" em "tb1", se a condição não for satisfeita o ELSE é executado e "ta1" recebe o valor de "a". Ou seja, "ta1" sempre receberá o sinal de "a", mas "tb1" só receberá o sinal de "b" quando "a" estiver em uma borda de subida como foi dito na página sobre PROCESS.

```
1  entity sequencia01 is
2  port (a,b : IN BIT;
3        sa, sb: OUT BIT);
4  end sequencia01;
5
6  architecture exemplo of sequencia01 is
7  signal ta1, tb1: bit;
8  begin
9      PROCESS (a)
10     begin
11         if (a='1' and a'event) then
12             ta1 <= a;
13             tb1 <= b;
14             else ta1 <= a;
15         end if;
16     end PROCESS;
17
18     sa <= ta1;
19     sb <= tb1;
20 end exemplo;
```

Comando PROCESS:

Um processo permite definir uma área de código contendo comandos sequenciais. O comando **PROCESS** apesar de ser um comando concorrente, ele delimita uma região de código seqüencial. Este comando divide-se em duas partes: a parte da declaração e a parte dos comandos sequenciais. A primeira parte é onde ocorre a declaração de constantes, variáveis, sinais, lista de sensibilidade, etc. A segunda parte, entre as palavras reservadas **BEGIN** e **END**, inicia a área contendo comandos sequenciais que representam o comportamento da descrição.
No Código 1 temos um exemplo do uso do **PROCESS**, onde o sinal de entrada "a" é a sensibilidade, ou seja, em cada mudança em "a" ocorre a execução de todos os comandos do processo. De modo geral,

a saída "sa" recebe o valor de "a" e a saída "sb" recebe o valor de “b” apenas quando ocorre uma mudança em “a” para ativar o processo.

```
entity sequencia01 is
port (a,b : IN BIT;
      sa, sb: OUT BIT);
end sequencia01;

architecture exemplo of sequencia01 is
signal ta1, tb1: bit;
begin
    PROCESS (a)
    begin
        if (a='1' and a'event) then
            ta1 <= a;
            tb1 <= b;
            else ta1 <= a;
        end if;
    end PROCESS;

    sa <= ta1;
    sb <= tb1;
end exemplo;
```

Pacotes:

Os pacotes são o único mecanismo de linguagem para compartilhar objetos entre unidades de design diferentes. Normalmente, eles são concebidos para proporcionar soluções padrão para problemas específicos, por exemplo tipos de dados e subprogramas correspondentes, como funções de conversão de tipos para diferentes tipos de dados, procedimentos e funções para fins de processamento de sinal, etc. Um pacote é declarado no seguinte formato:

```
package package_name is
    -- Declaration of
        -- types and subtypes
        -- subprograms
        -- constants, signals etc.
end package_name;


package body package_name is
    -- Definition of previously declared
       -- constants
       -- subprograms
    -- Declaration/definition of additional
       -- types and subtypes
       -- subprograms
       -- constants, signals and shared variables
end package_name;
```

Exemplo de programa utilizando pacotes:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

package test_pkg is
type t1 is  --a data type(totally 32 bits contains 3 different fields)
    record
        a :std_logic_vector(11 downto 0);
        b :std_logic_vector(15 downto 0);
        c :std_logic_vector(3 downto 0);
    end record;

 --function declaration.
function add (a2 : t1; b2: t1) return t1;
end test_pkg;   --end of package.

package body test_pkg is  --start of package body
--definition of function
function  add (a2 : t1; b2: t1) return t1 is
variable sum : t1;
begin -- Just name the fields in order...
sum.a:=a2.a xoR b2.a;
sum.b:=a2.b xoR b2.b;
sum.c:=a2.c xoR b2.c;
return sum;
end add;
--end function
end test_pkg;  --end of the package body
use work.test_pkg.all;

entity test is
port (clk : in std_logic;
      a1 : in t1;
      b1 : in t1;
      c1: out t1
    );
end test;

architecture Behavioral of test is
begin
process(clk)
begin
if(clk'event and clk='1') then
c1<=add(a1,b1);
end if;
end process;
end Behavioral;
```

Exercício:

Parte 1: Implementar o programa acima que utiliza pacotes, analisar, gerar o netlist no quartus

Parte 2: O algoritmo abaixo em VHDL apresenta um divisor. Neste código são apresentadas:

- A função utilizada para a divisão
- A forma de se utilizar a função no código
- As bibliotecas necessárias para implementação da função

```
function  divide  (a : UNSIGNED; b : UNSIGNED) return UNSIGNED is
variable a1 : unsigned(a'length-1 downto 0):=a;
variable b1 : unsigned(b'length-1 downto 0):=b;
variable p1 : unsigned(b'length downto 0):= (others => '0');
variable i : integer:=0;

begin
for i in 0 to b'length-1 loop
p1(b'length-1 downto 1) := p1(b'length-2 downto 0);
p1(0) := a1(a'length-1);
a1(a'length-1 downto 1) := a1(a'length-2 downto 0);
p1 := p1-b1;
if(p1(b'length-1) ='1') then
a1(0) :='0';
p1 := p1+b1;
else
a1(0) :='1';
end if;
end loop;
return a1;

end divide;
```

The function can be used as follows in your main module:

```
--An example of how to use the function.
signal a : unsigned(7 downto 0) :="10011100";
signal b : unsigned(7 downto 0) :="00001010";
signal c : unsigned(7 downto 0) :=(others => '0');
c <= divide ( a , b );  --function is "called" here.
```

For using the function you have to copy the code snippet between the green lines and put it in a package.The libraries I have used are given below:

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;       -- for UNSIGNED
```

Fazer um código VHDL que utiliza a parte 2, baseado no código da parte 1.

Enviar no moodle o código da parte 2 e os netlists dos códigos das partes 1 e 2.

OBS: Salvar o código da parte 2 porque ele vai ser objeto de estudo na aula que vem.

Referências:

http://vhdl.com.br/site/vhdl/comandos-basicos/ - comandos básicos VHDL

http://vhdl.com.br/site/exemplos - exemplos de projetos em VHDL

http://vhdlguru.blogspot.com.br/2010/03/usage-of-packages-and-functions.html