Architecture des ordinateurs - TD 2

Objectifs

Mise en oeuvre de composants cadencés par une horloge.

1. Retour sur les process synchrones du cours et leur intégration dans un composant Compteur

1.1 Process synchone sur une horloge

```
process (clk)
    variable cpt_aux : std_logic_vector(3 downto 0) := (others => '0');
begin
    if (clk = '1') then
        cpt_aux := cpt_aux + 1;
        cpt <= cpt_aux;
    end if;
end process;
```

1.2 Process synchrone sur une horloge avec reset asynchrone

```
process (clk, reset)
    variable cpt_aux : std_logic_vector(3 downto 0) := (others => '0');
begin
    if (reset = '0') then
        cpt_aux := (others => '0');
        cpt <= cpt_aux;
    elsif (rising_edge(clk)) then
        cpt_aux := cpt_aux + 1;
        cpt <= cpt_aux;
    end if;
end process;
```

1.3 Composant Compteur avec process synchrone

2. Adaptatation du composant Compteur

Modifiez l'interface et le fonctionnement du compteur en rajoutant un signal en sortie qui indique quand le compteur passe à zéro.

Solution avec variable

```
entity compteur is
    port ( clk, reset : in std_logic;
           cpt : out std_logic_vector(3 downto 0);
           carry : out std_logic;
    );
end compteur;

architecture Behavioral of compteur is
    signal cpt_aux : std_logic_vector(3 downto 0);
begin
    carry <= '1' when (cpt_aux = 0) else '0';
    process (clk, reset)
    begin
        if (reset = '0') then
            cpt_aux <= (others => '0');
            cpt <= cpt_aux;
        elsif (rising_edge(clk)) then
            cpt_aux <= cpt_aux + 1;
            cpt <= cpt_aux;
        end if;
    end process;

end compteur_behavioural;
```

Cependant, `cpt` a un front montant de l'horloge de retard par rapport à `cpt_aux`.
Pour régler ce problème, on effectue les modifications suivantes (lignes 17 et 19) :

```
- cpt <= cpt_aux;
+ cpt <= (others => '0')

- cpt <= cpt_aux;
+ cpt <= cpt_aux + 1;
```

Mais on peut encore faire mieux !

```
entity compteur is
    port ( clk, reset : in std_logic;
           cpt : out std_logic_vector(3 downto 0);
           carry : out std_logic;
    );
end compteur;

architecture Behavioral of compteur is
begin
    process (clk, reset)
        variable cpt_aux : std_logic_vector(3 downto 0) := (others => '0');
    begin
        if (reset = '0') then
            carry <= '0';
            cpt_aux := (others => '0');
            cpt <= cpt_aux;
        elsif (rising_edge(clk)) then
            cpt_aux := cpt_aux + 1;
            cpt <= cpt_aux;
            if (cpt_aux = 0) then
                carry <= '1';
            else
                carry <= '0';
            end if;
        end if;
    end process;

end compteur_behavioural;
```

Solution sans variable
On pourrait aussi faire comme suit :

```
entity compteur is
    port ( clk, reset : in std_logic;
           cpt : out std_logic_vector(3 downto 0);
           carry : out std_logic;
    );
end compteur;

architecture Behavioral of compteur is
    signal cpt_aux : std_logic_vector(3 downto 0);
begin
    carry <= '1' when (cpt_aux = 0) and (reset = '1') else '0';
    cpt <= cpt_aux;
    process (clk, reset)
    begin
        if (reset = '0') then
            cpt_aux <= (others => '0');
        elsif (rising_edge(clk)) then
            cpt_aux <= cpt_aux + 1;
        end if;
    end process;

end compteur_behavioural;
```

3. Un Compteur synchrone avec commande

Un compteur donne en sortie une valeur entière à chaque front montant d'une horloge.
Les entiers sont représentés par un nombre fixé de bits, nombre donné par un paramètre générique.
Une commande en entrée permet de modifier le comportement du compteur ; cette commande a cinq valeurs :

1. *STAY* maintient la valeur
2. *LOAD* permet de charger une valeur (donnée en entrée) dans le compteur
3. *INC* incrémente la valeur du compteur en recommançant à zéro quand la valeur maximale a été atteinte
4. *DEC* décrémente la valeur en recommançant à la valeur maximale quand zéro a été atteint
5. *RAZ* remet à zéro la valeur du compteur

La valeur de la commande est prise en compte sur chaque front montant de l'horloge (compteur synchrone). L'état du compteur correspond en fait à cette valeur de la commande.
Quand la commande est *DEC* ou *INC* et que la valeur en sortie atteint zéro, une retenue est mise à '1' en sortie du compteur puis repasse à '0' dès que la valeur redevient non nulle.

Question : Ecrivez le code correspondant à l'interface et à une architecture de ce compteur.

Utilisation d'un package pour définir un type utilisable dans l'entité

Fichier `pack_compteur.vhd` :

```
package pack_compteur is
    type t_cmd is (RAZ, LOAD, INC, DEC, STAY);
end package;```


**Fichier `compteur.vhd` :**
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use work.pack_compteur.all;

entity compteur is
    generic (
        cpt_width : natural := 4;
    );

    port ( clk: in std_logic;
           cmd: in t_cmd;
           cpt: out std_logic_vector(cpt_width-1 downto 0);
           incpt: in std_logic_vector(cpt_width-1 downto 0);;
           carry: out std_logic;
    );
end compteur;


architecture Behavioral of compteur is
begin

    process (clk)
        variable cpt_aux : std_logic_vector(cpt_width downto 0) :=
            (others => '0');
        variable changed : boolean;
    begin
        changed := false;

        if (rising_edge(clk)) then
            if cmd == RAZ then
                cpt_aux := (others => '0'); -- RAZ
            else if cmd == LOAD then
                cpt_aux := incpt;           -- LOAD
            else if cmd == INC then
                cpt_aux := cpt_aux + 1;     -- INC
                changed := true;
            else if cmd == DEC then
                cpt_aux := cpt_aux - 1;     -- DEC
                changed := true;
            else if cmd == STAY then
                cpt_aux := cpt_aux;         -- STAY
            end if;

            cpt <= cpt_aux;
        end if;

        if changed and cpt_aux == 0 then
            carry <= '1';
        else
            carry <= '0';
        end if;
    end process;

end Behavioral;
```
```