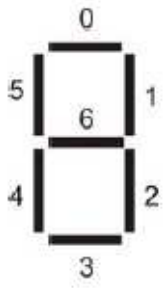
Exemple de code VHDL pour un compteur connecte a un afficheur 7-segments

Dans cet exemple, nous allons faire un compteur qui compte de 0 a 9 et qui envoie la valeur de son compte a un afficheur a 7 segments. La valeur du compteur augmente a chaque seconde et recommence a 0 après que la valeur atteigne 9.

Sur la plaquette DE2, il y a plusieurs afficheurs a 7-segments. Chaque afficheur est contrôle par 7 signaux qui proviennent du FPGA. Ces 7 signaux, qui indiquent si un segment est allume ou non, sont représentés dans la figure suivante comme étant les chiffres allant de 0 a 6. Pour représenter le chiffre 4, par exemple, il faudrait envoyer les bons signaux pour que les segments 1, 2, 5 et 6 soient allumes et que les autres ne le soient pas.



Un détail à savoir de ces afficheurs sur le DE2 c'est qu'ils ont besoin d'un '0' pour allumer. Donc, l'envoi d'un '1' a un segment l'éteindrait. Pour l'afficheur a 7 segments le plus a droite, les ports de sortie sont énumérés dans le tableau ci-dessous. Les connexions pour les autres afficheurs se trouvent dans la fiche technique du DE2.

Signal Name	FPGA Pin No.	Description
HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]

A ce stade, ce serait un bonne idée de savoir comment représenter chaque chiffre qu'on va vouloir afficher. Par exemple, si on voulait afficher le chiffre 1, il faudrait allumer les segments 1 et 2 seulement. On voudra énumérer les segments a allumer pour **tous** les chiffres allant de 0 a 9. Ceci facilitera notre travail plus tard.

Le but du design est de se faire un compteur allant de 0 à 9 qui incrémente à chaque seconde. Sachant que l'horloge qui entre dans notre système fonctionne à 50MHz, on devrait le diviser pour faire un compteur opérant à 1Hz. Pour ce faire, on se fait un compteur allant de 0 a 50000000 avant de recommencer. Quand le compte est entre 0 et 25000000, on génère 0 tandis qu'on génère 1 si le compte est entre 25000000 et 50000000. Notez en passant que mes valeurs ne sont pas précises puisque 50MHz ne représente pas exactement 50000000 cycles par seconde.

```
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        IF s_clk_compte > 50000000 THEN
            s_clk_compte <= (OTHERS => '0');
        ELSE
           s_clk_compte <= s_clk_compte + 1;
        END IF;

        IF s_clk_compte < 25000000 THEN
            s_clk_lent <= '0';
        ELSE
            s_clk_lent <= '1';
        END IF;
    END IF;
END PROCESS;
```

Nous obtenons une horloge du nom de s_clk_lent qui oscille à peu près à 1Hz. Nous allons maintenant utiliser cette horloge pour contrôler notre compteur décimal qui va de 0 à 9 avant de recommencer. Cette partie du code ressemble a ceci :

```
PROCESS (s_clk_lent)
BEGIN
    IF s_clk_lent'EVENT AND s_clk_lent = '1' THEN
        IF s_decimal > 8 THEN
            s_decimal <= (OTHERS => '0');
        ELSE
            s_decimal <= s_decimal + 1;
        END IF;
    END IF;
END PROCESS;
```

Pour afficher la valeur du compteur, on utilise un process combinatoire qui observe la valeur du compteur et qui génère les 7 signaux qui sont connectes a l'afficheur. Si le compte était égal a 0, les segments 0, 1, 2, 3, 4 et 5 seront allumes. Ceci est décrit dans le morceau de code suivant :

```
PROCESS (s_decimal)
BEGIN
   CASE s_decimal IS
       WHEN "0000" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '0';
            HEX06 <= '1';
       WHEN "0001" =>
            ...
```

Dans un process séquentiel, on va vouloir énumérer TOUS les cas possibles. Cependant, dans notre cas, nous n'avons besoin de spécifier que 10 cas allant de 0 à 9. Sachant qu'on a besoin de 4 bits, qu'arrive-t-il si le compte était égal a 12 ? Evidemment, cette situation ne devrait pas arriver, mais le VHDL veut quand même qu'on spécifie

toutes les situations. Une façon de le faire est de spécifier les sorties pour toutes les valeurs allant de 0 à 8. Ensuite, on va dire « pour tous les autres cas, utilisez la même sortie que pour le chiffre 9. Ceci se fait de la façon suivante :

```
      ...
      WHEN "1000" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '0';
            HEX06 <= '0';
       WHEN OTHERS =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '1';
            HEX05 <= '0';
            HEX06 <= '0';
   END CASE;
```

Le code final ressemblera à ceci :

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY countled IS
    PORT (
          clk   : IN  STD_LOGIC;
          HEX00 : OUT STD_LOGIC;
          HEX01 : OUT STD_LOGIC;
          HEX02 : OUT STD_LOGIC;
          HEX03 : OUT STD_LOGIC;
          HEX04 : OUT STD_LOGIC;
          HEX05 : OUT STD_LOGIC;
          HEX06 : OUT STD_LOGIC
         );
END countled;

ARCHITECTURE rtl OF countled IS

SIGNAL s_clk_compte : STD_LOGIC_VECTOR(25 DOWNTO 0);
SIGNAL s_clk_lent : STD_LOGIC;
SIGNAL s_decimal : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN

PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        IF s_clk_compte > 50000000 THEN
            s_clk_compte <= (OTHERS => '0');
        ELSE
           s_clk_compte <= s_clk_compte + 1;
```

```
        END IF;

        IF s_clk_compte < 25000000 THEN
            s_clk_lent <= '0';
        ELSE
            s_clk_lent <= '1';
        END IF;
    END IF;
END PROCESS;

PROCESS (s_clk_lent)
BEGIN
    IF s_clk_lent'EVENT AND s_clk_lent = '1' THEN
        IF s_decimal > 8 THEN
            s_decimal <= (OTHERS => '0');
        ELSE
            s_decimal <= s_decimal + 1;
        END IF;
    END IF;
END PROCESS;

PROCESS (s_decimal)
BEGIN
   CASE s_decimal IS
       WHEN "0000" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '0';
            HEX06 <= '1';
       WHEN "0001" =>
            HEX00 <= '1';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '1';
            HEX04 <= '1';
            HEX05 <= '1';
            HEX06 <= '1';
       WHEN "0010" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '1';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '1';
            HEX06 <= '0';
       WHEN "0011" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '1';
            HEX05 <= '1';
            HEX06 <= '0';
       WHEN "0100" =>
```

```
            HEX00 <= '1';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '1';
            HEX04 <= '1';
            HEX05 <= '0';
            HEX06 <= '0';
      WHEN "0101" =>
            HEX00 <= '0';
            HEX01 <= '1';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '1';
            HEX05 <= '0';
            HEX06 <= '0';
      WHEN "0110" =>
            HEX00 <= '0';
            HEX01 <= '1';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '0';
            HEX06 <= '0';
      WHEN "0111" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '1';
            HEX04 <= '1';
            HEX05 <= '1';
            HEX06 <= '1';
      WHEN "1000" =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '0';
            HEX05 <= '0';
            HEX06 <= '0';
      WHEN OTHERS =>
            HEX00 <= '0';
            HEX01 <= '0';
            HEX02 <= '0';
            HEX03 <= '0';
            HEX04 <= '1';
            HEX05 <= '0';
            HEX06 <= '0';
   END CASE;
END PROCESS;

END;
```