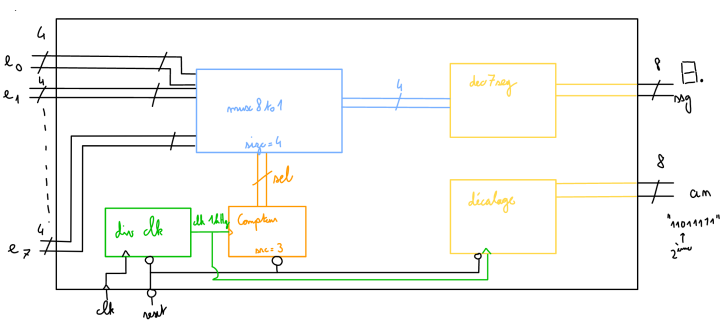
Architecture des ordinateurs - TD 3 : Afficher les 8 Sept-Segments

Concevez, développez et testez (en TP) un composant qui permet d'afficher 8 valeurs différentes sur les 8 Sept-Segments.
Le principe est d'afficher de manière cyclique chacune des valeurs en sélectionnant chaque 7-Segments pendant une période suffisamment rapide pour duper l'oeil.
Une horloge à 1 kHz permet d'avoir un affichage correct.
Trois versions sont à développer :

1. une vue structurelle (que des sous-composants reliés les uns aux autres)




```
entity All_sept_segments is
    port (
        clk, reset: in std_logic;
        e0, e1, e2, e3, e4, e5, e6, e7: in std_logic_vector(3 downto 0);
        ssg, an: out std_logic_vector(7 downto 0);
    )
end All_sept_segments;

architecture structural of All_sept_segments is
    -- Rappel des composants

    component diviseurClk
        generic (facteur : natural);
        port (
            clk,reset : in std_logic;
            nclk : out std_logic
        );
    end component;

    component dec7seg
        port (
            v : in std_logic_vector(3 downto 0);
            seg : out std_logic_vector(7 downto 0)
        );
    end component;

    component compteur
        generic (size : natural := 4);
        port (
            clk : in std_logic;
            reset : in std_logic;
            cpt : out std_logic_vector(size=1 downto 0)
        );
    end component;

    component mux8_to_1
        generic (size: natural := 4) ;
        port ( e0 , e1 , e2 , e3 ,
            e4 , e5 , e6 , e7 : in std_logic_vector(size=1 downto 0);
            sel : in std_logic_vector(2 downto 0);
            s : out std_logic_vector (size=1 downto 0)
        );
    end component;

    component decalage
        port (
            clk : in std_logic;
            reset : in std_logic;
            v : out std_logic_vector(7 downto 0)
        );
    end component;

    -- Signaux internes
    signal iclk : std_logic;
    signal selVal : std_logic_vector(3 downto 0);
```

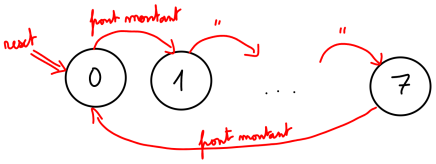
```
begin

    DivClk : diviseurClk
        generic map (facteur => 100000) -- pas sûr de celle là pour le coup
        port map (clk => clk, reset => reset, nclk => iclk);

    Cpt : compteur
        generic map (size => 3)
        port map (clk, reset, c);

    Mux : mux8_to_1
        generic map(size => 4)
        port map (hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7, sel, selVal);

    Deco : dec7seg
        port map (selVal, ssg);

    Deca : decalage
        port map(clk, reset, an);
end structural;
```

2. une vue comportementale avec un process qui va permettre de compter,

```
architecture behavioral of All_sept_segments is
    -- Signaux
    signal clk1kHz : std_logic;
    signal s : std_logic_vector(3 downto 0);
    signal cpt : natural;

begin
        dclk : diviseurClk
            generic map (100000)
            port map (clk, reset, clk1kHz);

        process (clk1kHz, reset)
            -- Variables
            -- Constantes
        begin
            if reset = '0' then
                -- RESET : on reset les compteurs, on réinitialise l'écran
                ssg <= (others => '0');
                an <= (others => '1');
                cpt <= 0;
            elsif (rising_edge(clk1kHz)) then
                cpt <= (cpt + 1) mod 8;
                -- ne marche que parce que 8 est une puissance de 2 !
                case cpt is
                    when 0 =>
                        s <= e0;
                        -- an <= "11111110";
                    when 1 =>
                        s <= e1;
                        -- an <= "11111101";
                    when 2 =>
                        s <= e2;
                        -- an <= "11111011";
                    when 3 =>
                        s <= e3;
                        -- an <= "11110111";
                    when 4 =>
                        s <= e4;
                        -- an <= "11101111";
                    when 5 =>
                        s <= e5;
                        -- an <= "11011111";
                    when 6 =>
                        s <= e6;
                        -- an <= "10111111";
                    when 7 =>
                        s <= e7;
                        -- an <= "01111111";
                    when others => noop; -- ????
                end case;

                -- ou mieux, on fait après le switch case :
                an <= (others => '1');
                an(cpt) <= '0';
            end if;
```

```
        end process;
end behavioral;
```

3. une vue comportementale avec un process qui va implémenter un automate à états (fsm - finite state machine).




```
            end case;
        end if;
    end process;
end finite_state_machine;
```

On dispose des composants en annexe. On peut en rajouter d'autres dont on précisera les interfaces.

2 Annexes

2.1 diviseur d'horloge

```
entity diviseurClk is
    -- facteur : ratio entre la fréquence de l'horloge origine et celle
    -- de l'horloge générée
    -- ex : 100 MHz => 1Hz : 100 000 000
    -- ex : 100 MHz => 1kHz : 100 000
    generic (facteur : natural);
    port (
        clk,reset : in stdlogic;
        nclk : out stdlogic
    );
end diviseurClk;
```

2.2 Afficheur-7segments

```
entity dec7seg is
    port (
        v : in std_logic_vector(3 downto 0);
        seg : out std_logic_vector(7 downto 0)
    );
end dec7seg;
```

2.3 compteur

```
entity compteur is
    generic (size : natural := 4);
    port (
        clk : in std logic;
        reset : in std logic;
        cpt : out std logic vector(size=1 downto 0)
    );
end compteur;
```

2.4 mux8_to_1

```
architecture finite_state_machine of All_sept_segments is
    -- Types
    type t_etat is (zero, un, deux, trois, quatre, cinq, six, sept);

    -- Signaux
    signal etat : t_etat;
    signal clk1kHz : std_logic;
    signal s : std_logic_vector(3 downto 0);

begin
    dclk : diviseurClk
        generic map (100000)
        port map (clk, reset, clk1kHz);

    process(clk1kHz, reset)
    begin
        if(reset = '0') then
            etat <= zero;
            s <= (others => '0');
            an <= (others => '1');
        elsif (rising_edge(clk1kHz)) then
            case etat is
                when zero =>
                    s <= e0;
                    an <= "11111110";
                    etat <= un;
                when un =>
                    s <= e1;
                    an <= "11111101";
                    etat <= deux;
                when deux =>
                    s <= e2;
                    an <= "11111011";
                    etat <= trois;
                when trois =>
                    s <= e3;
                    an <= "11110111";
                    etat <= quatre;
                when quatre =>
                    s <= e4;
                    an <= "11101111";
                    etat <= cinq;
                when cinq =>
                    s <= e5;
                    an <= "11011111";
                    etat <= six;
                when six =>
                    s <= e6;
                    an <= "10111111";
                    etat <= sept;
                when sept =>
                    s <= e7;
                    an <= "01111111";
                    etat <= zero;
                when others => noop; -- ????
```

```
entity mux8_to_1 is
    generic (size : natural := 4) ;
    port ( e0 , e1 , e2 , e3 ,
        e4 , e5 , e6 , e7 : in std_logic_vector(size=1 downto 0);
        sel : in std_logic_vector(2 downto 0);
        s : out std_logic_vector (size=1 downto 0)
    );
end mux8_to_1;
```

2.5 decalage

```
entity decalage is
    -- v contient un seul '0' (utilisation : anode == segment allumé)
    -- à chaque front montant de l'horloge,
    -- la valeur de vest décalée cycliquement d'une position vers la gauche
    -- "11101111" => "11011111"
    -- "01111111" => "11111110"
    port (
        clk : in std_logic;
        reset : in std_logic;
        v : out std_logic_vector(7 downto 0)
    );
end decalage;
```