Rapport complet Chenillard

TALENT Julien et RUIZ Evan Gr3

Cahier des charges :

L'objectif est de créer un chenillard qui permet d'allumer successivement des leds et/ou des afficheurs selon le modes sélectionné et la vitesse de défilement choisie.
Il posséde :

- Différents modes de fonctionnement sélectionnable par des switchs de la carte ;
- Une vitesse de défilement variable et réglable par des switchs de la carte ;
- Une fonction "pause" à l'aide d'un bouton poussoir de la carte ;
- Une fonction "marche/arrêt" à l'aide d'un switch de la carte ;
- Une fonction "réinitialisation" à l'aide d'un bouton poussoir de la carte.

Spécifications :

4 vitesses de défilement sont à réaliser :

- 1Hz ,
- 3Hz ,
- 7Hz ,
- 12Hz .

et 8 modes de fonctionnements doivent être exécutable :

Modes LEDs :

- Mode 1.1 : les LEDs rouges s’allument successivement de la 0 à la 7, puis cela recommence de 0 à 7… ;
- Mode 1.3 : les LEDs rouges s’allument de la 0 à la 7 pendant que les leds vertes s’allument de la 7 à la 0 et le cycle recommence ;
- Mode 1.7 : les LEDs vertes s’allument successivement par 3 : 0, 1, 2 puis 1, 2, 3 puis 2, 3, 4… jusqu’à 5, 6, 7 puis recommence à 0, 1, 2… ;

Modes Afficheurs :

- Mode 2.1 : les segments d’un afficheur s’allument successivement, du segment a jusqu’au segment g… ;

- Mode 2.4 : les afficheurs clignotent en affichant 2025 ;
- Mode 2.7 : les segments e, f de HEX3 s’allume puis les segments b, c de HEX3 puis les segments e,f de HEX2… jusqu’à l’allumage des segments b, c de HEX0 puis en recommence à partir de HEX3 ;

Mode combiné et mode défilement :

- Mode 3 : mode combiné. 1 mode LED au choix doit fonctionner avec un mode Afficheur au choix ;
- Mode 4 : mode défilement. Faire défiler un mot de 6 lettres sur les 4 afficheurs .
- RAZ : le bouton poussoir KEY(2) permet de réinitialiser le mode de défilement c'est-à-dire de reprendre le défilement à partir de sa position initiale;
- PAUSE : le bouton poussoir KEY(0) permet de mettre en pause le système (l’allumage s’arrête sur la position actuelle) ;
- M/A : le switch SW(9) permet de mettre en marche ou d’arrêter le système (à l’arrêt toutes les LEDs et les afficheurs sont éteints).
-

Vitesse :

Le module "vitesse" permet de générer une horloge avec une fréquence ajustable en fonction de la sélection de vitesse.

Emplacement et fonctionnement

Emplacement

Ce module se situe au début, avant les modes, et permet de fournir une horloge à fréquence réduite en fonction d'un signal de sélection.

Fonctionnement

Le module reçoit une horloge d'entrée de 50 MHz et divise cette fréquence en fonction de la valeur de `selecSpeed` .

- "00" : 1 Hz
- "01" : 3 Hz
- "10" : 7 Hz
- "11" : 12 Hz

Le signal de sortie `clk_out` oscille en fonction de cette fréquence ajustée. Lorsque `onOff` est à '1', le signal est activé et fonctionne normalement. Le signal `onOff` permet également

de mettre en pause l'horloge et donc de stopper temporairement l'exécution des modes, selon la demande de l'utilisateur.

Le code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity vitesse is
port(
    clk_in : in std_logic; -- l'horloge de 50MHz
    clk_out : buffer std_logic; -- la sortie de la fréquence d'horloge
abaissée
    selecSpeed : in std_logic_vector(1 downto 0);
    onOff : in std_logic
);
end vitesse;

architecture compteur of vitesse is
signal etatCompteur : std_logic_vector(25 downto 0); -- Utilisation de 26
bits pour compter jusqu'à 50 MHz, nécessaire pour diviser la fréquence
d'horloge
signal vitesse : integer; -- permet de mettre les valeurs en décimal sans
définir le nombre de bits
begin
        process(selecSpeed)
        begin
            case selecSpeed is -- Sélectionne la valeur de division pour
obtenir la fréquence souhaitée
                when "00" => vitesse <= 24999999; -- 1 Hz
                when "01" => vitesse <= 8333332; -- 3 Hz
                when "10" => vitesse <= 3571428; -- 7 Hz
                when "11" => vitesse <= 2083332; -- 12 Hz
                when others => vitesse <= 24999999; -- 1 Hz
            end case;
        end process;

        process(clk_in)
        begin
            if rising_edge(clk_in) and onOff = '1' then -- Désactive
l'horloge lorsque onOff est à '0'
                if etatCompteur = vitesse then
                    etatCompteur <= conv_std_logic_vector(0,26); --
Réinitialisation du compteur
                    clk_out <= not clk_out; -- Inversion du signal de sortie
                else
                    etatCompteur <= etatCompteur + 1; -- Incrémentation
```

```
normale
                end if;
            end if;
        end process;
end compteur;
```

Pour trouvé le nombre de bit nécessaire pour la variable `etatCompteur` on fait le calcul suivant :

$$\frac{\ln(50 \times 10^6)}{\ln(2)} = 25.6$$

On arrondis donc au supérieur ce qui donne 26.

Pour trouvé la valeur de `vitesse` on fait les calcules suivants :

$$\frac{clk\_in}{2 \times clk\_voulue} - 1$$

Conclusion

Le module "vitesse" permet de générer une horloge à fréquence ajustable en fonction d'un signal de sélection, avec une plage de 1 Hz à 12 Hz. Cela permet d'adapter la vitesse d'affichage ou d'autres processus nécessitant une fréquence réduite. De plus, la présence du signal `onOff` permet de mettre en pause ces modes sur demande de l'utilisateur.

Marche / Arrêts

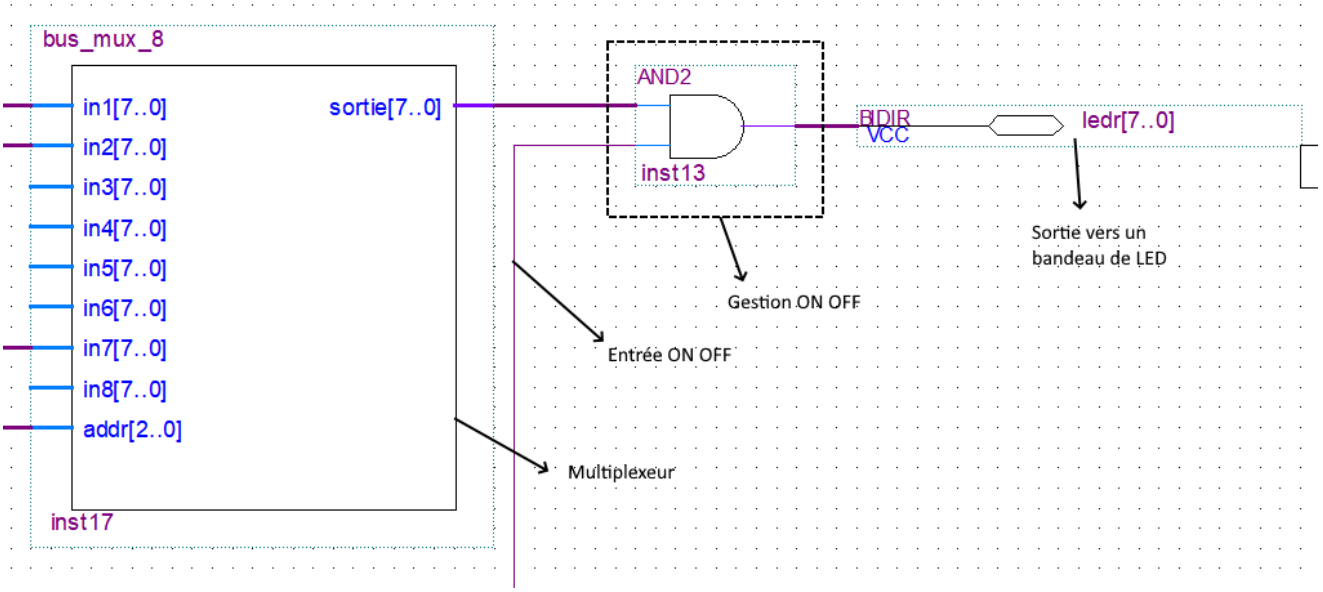
But

La fonction ON/OFF a pour devoirs d'étreindre les Diodes Electroluminescentes (LED), des bandeaux et des afficheur 7 segments lorsque l'entré ON OFF est a 0.

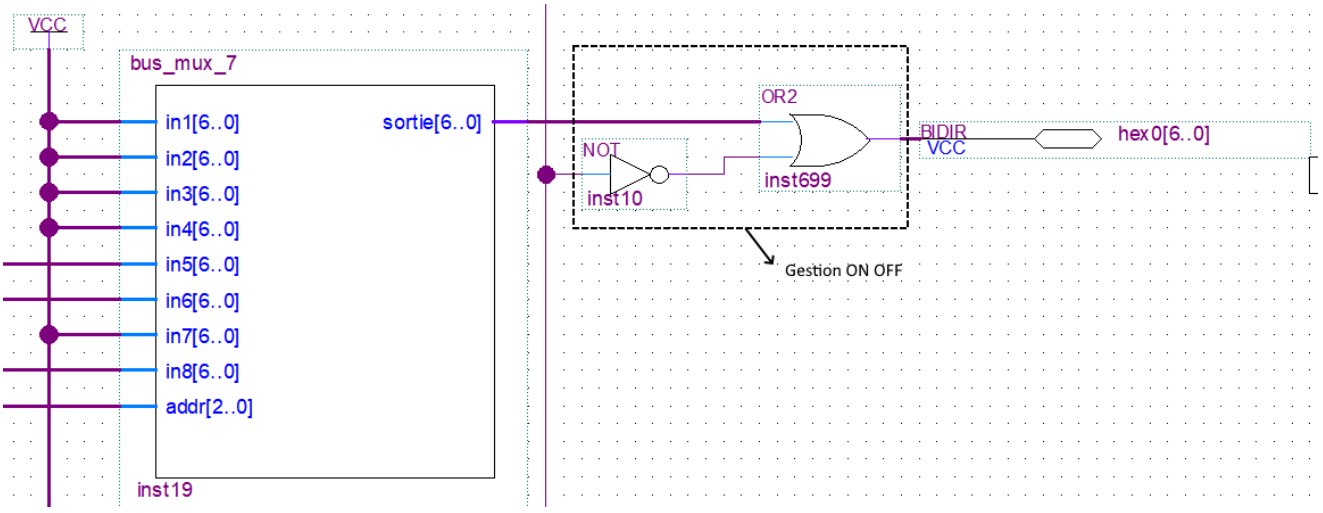
Emplacement et fonctionnement

Emplacement

Pour un bandeau LED :



Pour un afficheur 7 segments :



Fonctionnement

Pour un bandeau LED :

Chaque bit de la sortie est passé dans un masque ET avec la table de vérité suivante :

Sortie	ON/OFF	Résultat
0	0	0
1	0	0
0	1	1
1	1	1

On peut voir que tant que ON/OFF est a 0 la sortie restera éteinte.

Pour un 7 segments

On veut que la sortie soit a 1 et non a 0 pour éteindre les segment pour cela on crée un tableau de Carnot :

Sortie \ ON/OFF	0	1
0	1	0
1	1	1

On en déduit l'équation suivante :

$$résultat = \overline{sortie} + Sortie$$

On retrouve bien l'assemblage de porte logiques dans le schéma 2.

Problèmes rencontrés

lorsque l'on avait une fonction qu'y n'utilisait pas les afficheur ils était allumé car le bus était a 0 ce qui correspond a l'état allumé, nous avons donc connecté ces entré au Vcc pour que les sorties non utilisée soit éteintes.

Conclusion

La fonction ON/OFF permet d'éteindre toutes les LED.
Suites aux dernier testes de la carte il n'y a pas eu d'autres problème de fonctionnement ou de bugs.

RAZ :

> RAZ : le bouton poussoir KEY(2) permet de réinitialiser le mode de défilement c'est-à-dire de reprendre le défilement à partir de sa position initiale

Cette fonction est directement intégré dans les programmes des différents modes par un IF:

```
if key2 = '0' then--réinitialisation
            ledv <= "00000000";

        elsif rising_edge(clk_in)then
```

Process :

Dans chaque programmes on doit intégré un Process dans la partie architecture lorsque l'on utilise un case ou un if.

Le process permet de mettre à jour les variables contenue dans le multiplexeur.
En quelque sorte cela rajoute une entrée spécifique utilisé comme horloge.

Les modes LEDs :

Mode 1.1 :

> Les LEDs rouges s'allument successivement de la 0 à la 7, puis cela recommence de 0 à 7… ;

Code :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity fonction_1_1 is
port(
    clk_in : in std_logic;
    ledr : buffer std_logic_vector(7 downto 0);
    key2 : in std_logic
);
end fonction_1_1;

architecture chenilleDGDG of fonction_1_1 is

begin
    process(clk_in, key2)
    begin
        if key2 = '0' then
                ledr <= "00000000";

            elsif rising_edge(clk_in)then
            case ledr is --permet de décalé la led allumé
                when "00000001" => ledr <= "00000010";
                when "00000010" => ledr <= "00000100";
                when "00000100" => ledr <= "00001000";
                when "00001000" => ledr <= "00010000";
                when "00010000" => ledr <= "00100000";
```

```
                when "00100000" => ledr <= "01000000";
                when "01000000" => ledr <= "10000000";
                when "10000000" => ledr <= "00000001";
                when others => ledr <= "00000001";--permet
l'initialisation
            end case;
        end if;
    end process;
end chenilleDGDG;
```

L'utilisation d'un case est beaucoup plus adapté car la sortie dépend d'un grand nombre de conditions. La lisibilité est meilleurs donc en cas d'erreur on auras plus de facilité à s'y retrouver.

- Les LEDs rouges sont des buffers car on lis leurs valeurs pour ensuite changer leurs état donc ils reçoivent des informations.
- Le `if key2 = '0' then` permet d'éteindre les leds si on appuie sur `KEY2`. C'est la fonction `RAZ`.
- Dans ce cas le `case` est simple à comprendre, si la `ledr0` est allumé alors on allume la `ledr1`, si la `ledr1` est allumé on allume la `ledr2` et ainsi de suite.
- Le `elsif rising_edge(clk_in)` permet de passer à l'étape suivante sur le front montant de l'horloge interne.
- A la fin on met un `when others` qui permet de revenir au points de départ.

Problèmes :

Au début nous n'avons pas pensez à intégrer la fonction `RAZ` au programme

Mode 1.3

> Les LEDs rouges s'allument de la 0 à la 7 pendant que les leds vertes s'allument de la 7 à la 0 et le cycle recommence

On repart sur la base du mode 1.1 mais on rajoute un deuxième case pour les LEDs vertes qui vont défiler de la 7 à la 0.

Code :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

```
entity fonction_1_3 is
port(
    clk_in : in std_logic;
    ledr : buffer std_logic_vector(7 downto 0);
    ledg : buffer std_logic_vector(7 downto 0);
    key2 : in std_logic
);
end fonction_1_3;

architecture chenilleDGDG of fonction_1_3 is

begin
    process(clk_in, key2)
    begin
        if key2 = '0' then
            ledr <= "00000000";
            ledg <= "00000000";

        elsif rising_edge(clk_in)then
            case ledr is --permet de décalé la led allumé
                when "00000001" => ledr <= "00000010";
                when "00000010" => ledr <= "00000100";
                when "00000100" => ledr <= "00001000";
                when "00001000" => ledr <= "00010000";
                when "00010000" => ledr <= "00100000";
                when "00100000" => ledr <= "01000000";
                when "01000000" => ledr <= "10000000";
                when "10000000" => ledr <= "00000001";
                when others => ledr <= "00000001";--permet l'initialisation
            end case;
            case ledg is --permet de décalé la led allumé
                when "10000000" => ledg <= "01000000";
                when "01000000" => ledg <= "00100100";
                when "00100000" => ledg <= "00010000";
                when "00010000" => ledg <= "00001000";
                when "00001000" => ledg <= "00000100";
                when "00000100" => ledg <= "00000010";
                when "00000010" => ledg <= "00000001";
                when "00000001" => ledg <= "10000000";
                when others => ledg <= "10000000";--permet l'initialisation
            end case;
        end if;
    end process;

end chenilleDGDG;
```

- Nous avons pensez à intégrer la RAZ des LEDs vertes.
- Les LEDs vertes sont elles aussi des buffer.

Problèmes :

Aucun, car les problèmes rencontré au MODE 1.1 ont été évité au MODE 1.3.

Mode 1.7:

> Les LEDs vertes s'allument successivement par 3 : 0, 1, 2 puis 1, 2, 3 puis 2, 3, 4… jusqu'à 5, 6, 7 puis recommence à 0, 1, 2… ;

On repart encore une fois sur la base du MODE 1.1 sauf que cet fois si on utilise les LEDs vertes et ont les fait défiler 3 par 3 donc il y auras moins de conditions dans le case :

Code :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity fonction_1_7 is
port(
    clk_in : in std_logic;
    ledv : buffer std_logic_vector(7 downto 0);
    key2 : in std_logic

);
end fonction_1_7;

architecture chenilleDGDG of fonction_1_7 is

begin
    process(clk_in, key2)
    begin
        if key2 = '0' then--réinitialisation
            ledv <= "00000000";

        elsif rising_edge(clk_in)then
            case ledv is --permet de décalé la led allumé
                when "00000111" => ledv <= "00001110";
                when "00001110" => ledv <= "00011100";
                when "00011100" => ledv <= "00111000";
                when "00111000" => ledv <= "01110000";
                when "01110000" => ledv <= "11100000";
                when "11100000" => ledv <= "00000111";
                when others => ledv <= "00000111";--permet l'initialisation
            end case;
```

```
        end if;
    end process;
end chenilleDGDG;
```

Attention : On constate que l'on utilise plus ledg mais ledv, cela ne change rien car ce sont juste des nom d'entré et sorties. Plus tard dans le BUS MUX ont définir les bonnes entrées et sorties lors de la création d'un block.

- On garde toujours la fonction `RAZ` et rien ne change par rapport aux deux modes précédents.

Problèmes :

Aucun, les erreurs faites précédemment nous on permis de ne plus les refaire.

Les modes afficheurs

Particularité des 7 segments :

Les afficheurs 7 segments que l'on utilise ont la particularité d'être des 7 segment inversé c'est a dire que l'on met à `1` un segment pour qu'il s'éteigne et inversement on met à `0` un segment pour qu'il s'allume.

Ce sont des afficheurs 7 segment à anodes communes.

Mode 2.1 :

> Les segments d’un afficheur s’allument successivement, du segment a jusqu’au segment g… ;

Emplacement et fonctionnement

Emplacement

Il se situe après la gestion de l'horloge et avant les multiplexeurs.

Fonctionnement

Le mode reçoit le signal d'horloge et allume les segments de l'afficheur un par un, en partant du segment a jusqu'au segment g.

Le code

```
-- Importations des librairies
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- Definition des entrées
entity fonction_2_1 is
port(
    clk_in : in std_logic;
    hex3 : buffer std_logic_vector(6 downto 0);
    key2 : in std_logic
);
end fonction_2_1;

-- Le programme
architecture chenilleDGDG of fonction_2_1 is
begin
    process(clk_in, key2) -- En cas de changement de l'une de ces entrées
nous allons procéder à l'actualisation du code suivant
    begin
        if key2 = '0' then -- Remise à 0
            hex3 <= "0000000";
        elsif rising_edge(clk_in) then -- On changera l'affichage uniquement
si l'on est sur un front montant de l'horloge
            case hex3 is -- Permet de décaler la LED allumée
                when "1111110" => hex3 <= "1111101"; -- Segment a
                when "1111101" => hex3 <= "1111011"; -- Segment b
                when "1111011" => hex3 <= "1110111"; -- Segment c
                when "1110111" => hex3 <= "1101111"; -- Segment d
                when "1101111" => hex3 <= "1011111"; -- Segment e
                when "1011111" => hex3 <= "0111111"; -- Segment f
                when others => hex3 <= "1111110"; -- Segment g, permet
l'initialisation
            end case;
        end if;
    end process;
end chenilleDGDG;
```

Conclusion

Le mode 2.1 allume successivement les segments de l'afficheur, en partant du segment a jusqu'au segment g, à la vitesse de l'horloge.

Mode 2.4

> Les afficheurs clignotent en affichant 2025 ;

Emplacement et fonctionnement

Emplacement

Il se situe après la gestion de l'horloge et avant les multiplexeurs.

Fonctionnement

le mode reçoit le signal d'horloge et passe de l'affichage de rien à 2025.

Le code

```
-- Importations des librairies
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- Definition des entrées
entity fonction_2_4 is
port(
    clk_in : in std_logic;
    hex0 : buffer std_logic_vector(6 downto 0);
    hex1 : buffer std_logic_vector(6 downto 0);
    hex2 : buffer std_logic_vector(6 downto 0);
    hex3 : buffer std_logic_vector(6 downto 0);
    key2 : in std_logic

);
end fonction_2_4;

-- Le programme
architecture Didier of fonction_2_4 is
begin

    process(clk_in, key2)-- En cas de changement de l'une de ces entrées
nous allons procédé a l'actualisation du code suivant
    begin
        if key2 = '0' then-- Remise a 0
            hex0 <= "0000000";
            hex1 <= "0000000";
            hex2 <= "0000000";
            hex3 <= "0000000";

            -- Gestion de l'affichage
```

```
        elsif rising_edge(clk_in)then-- On changera l'affichage uniquement
si l'on est sur un front montant de l'horloge
            case hex0 is-- On se base sur l'état de hex0
                when "0010111" => -- Etein les segments
                    hex0 <= "1111111";
                    hex1 <= "1111111";
                    hex2 <= "1111111";
                    hex3 <= "1111111";
                when others => -- Permet l'initialisation et l'affichage de
2025
                    hex0 <= "0010111";
                    hex1 <= "0100100";
                    hex2 <= "1000000";
                    hex3 <= "0100100";
            end case;
        end if;
    end process;

end Didier;
```

Conclusion

Le mode 2.4 affiche la date 2025 en clignotant à la vitesse de l'horloge.

Mode 2.7 :

> Les segments e, f de HEX3 s'allume puis les segments b, c de HEX3 puis les segments e,f de HEX2… jusqu'à l'allumage des segments b, c de HEX0 puis en recommence à partir de HEX3.

Emplacement et fonctionnement

Emplacement

Il se situe après la gestion de l'horloge et avant les multiplexeurs.

Fonctionnement

le mode reçoit le signal d'horloge et décale les segments allumé de l'HEX3 à l'HEX0 en boucle.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity fonction_2_7 is
port(
    clk_in : in std_logic;
    hex0 : buffer std_logic_vector(6 downto 0);
    hex1 : buffer std_logic_vector(6 downto 0);
    hex2 : buffer std_logic_vector(6 downto 0);
    hex3 : buffer std_logic_vector(6 downto 0);
    key2 : in std_logic

);
end fonction_2_7;

architecture chenille of fonction_2_7 is

begin



    process(clk_in, key2)
    begin
        if key2 = '0' then
            hex0 <= "0000000";
            hex1 <= "0000000";
            hex2 <= "0000000";
            hex3 <= "0000000";

        elsif rising_edge(clk_in)then
            case hex3 is -- barre de gauche : 1001111 barre de droite :
1111001
                when "1001111" => -- hex3 gauche
                    hex0 <= "1111111";
                    hex1 <= "1111111";
                    hex2 <= "1111111";
                    hex3 <= "1111001";-- hex3 droite
                when "1111001" => -- hex3 droite
                    hex0 <= "1111111";--
                    hex1 <= "1111111";--
                    hex2 <= "1001111";-- hex2 gauche
                    hex3 <= "1111111";
                when "1001111" => -- hex2 gauche
                    hex0 <= "1111111";--
                    hex1 <= "1111111";--
                    hex2 <= "1111001";-- hex2 droite
                    hex3 <= "1111111";--
                when "1111001" => -- hex2 droite
                    hex0 <= "1111111";--
                    hex1 <= "1001111";--hex1 gauche
```

```
                    hex2 <= "1111111";--
                    hex3 <= "1111111";--
                when "1001111" => -- hex1 gauche
                    hex0 <= "1111111";--
                    hex1 <= "1111001";-- hex1 droite
                    hex2 <= "1111111";--
                    hex3 <= "1111111";--
                when "1111001" => -- hex1 droite
                    hex0 <= "1001111";-- hex0 gauche
                    hex1 <= "1111111";--
                    hex2 <= "1111111";--
                    hex3 <= "1111111";--
                when "1001111" => -- hex0 gauche
                    hex0 <= "1111001";-- hex0 droite
                    hex1 <= "1111111";--
                    hex2 <= "1111111";--
                    hex3 <= "1111111";--
                when others => -- hex0 droite
                    hex0 <= "1111111";--
                    hex1 <= "1111111";--
                    hex2 <= "1111111";--
                    hex3 <= "1111001";-- hex3 gauche

            end case;
        end if;
    end process;

end chenille;
```

Le principe est le même, on utilise un case car on as beaucoup de conditions.

Conclusion

Le mode 2.7 affiche les segments e,f et b,c de l'HEX3 à l'HEX0 à la vitesse de l'horloge.

Mode combiné et mode défilement :

Mode 3 :

> Mode combiné. 1 mode LED au choix doit fonctionner avec un mode Afficheur au choix
> On choisie d'afficher le mode 1.1 avec le mode 2.1

- Mode 1.1 : les LEDs rouges s'allument successivement de la 0 à la 7, puis cela recommence de 0 à 7… ;
- Mode 2.1 : les segments d'un afficheur s'allument successivement, du segment a jusqu'au segment g… ;

Code :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity fonction_3 is
port(
    clk_in : in std_logic;
    hex3 : buffer std_logic_vector(6 downto 0);
    ledr : buffer std_logic_vector(7 downto 0);
    key2 : in std_logic

);
end fonction_3;

architecture chenilleDGDG of fonction_3 is

begin
    process(clk_in, key2)
    begin
        if key2 = '0' then
                hex3 <= "0000000";
                ledr <= "00000000";

            elsif rising_edge(clk_in)then

            case hex3 is --permet de décalé la led allumé
                when "1111110" => hex3 <= "1111101";
                when "1111101" => hex3 <= "1111011";
                when "1111011" => hex3 <= "1110111";
                when "1110111" => hex3 <= "1101111";
                when "1101111" => hex3 <= "1011111";
                when "1011111" => hex3 <= "0111111";
                when others => hex3 <= "1111110";--permet l'initialisation
            end case;


            case ledr is --permet de décalé la led allumé
                when "00000001" => ledr <= "00000010";
                when "00000010" => ledr <= "00000100";
                when "00000100" => ledr <= "00001000";
```

```
                when "00001000" => ledr <= "00010000";
                when "00010000" => ledr <= "00100000";
                when "00100000" => ledr <= "01000000";
                when "01000000" => ledr <= "10000000";
                when "10000000" => ledr <= "00000001";
                when others => ledr <= "00000001";--permet l'initialisation
            end case;
        end if;
    end process;

end chenilleDGDG;
```

Ce code ressemble au mode 1.3 sauf que on remplace les LEDs vertes par un afficheur HEX3.

- On n'oublie toujours pas la fonction `RAZ`
- On met deux case pour faire fonctionner les LEDs rouges en même temps que l'afficheur HEX3.

Problèmes :

Aucun car ce code ressemble beaucoup au MODE 1.3 qui lui est ressemblant au MODE 1.1 donc toutes les erreurs faites précédemment nous ont permis de ne plus les faires.

Mode 4 :

> Mode 4 : mode défilement. Faire défiler un mot de 6 lettres sur les 4 afficheurs .
>
> - On décide de faire défiler le mot Github.

Code:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity fonction_4 is
port(
    clk_in : in std_logic;
    hex0 : buffer std_logic_vector(6 downto 0);
    hex1 : buffer std_logic_vector(6 downto 0);
    hex2 : buffer std_logic_vector(6 downto 0);
    hex3 : buffer std_logic_vector(6 downto 0);
    key2 : in std_logic
```

```
);
end fonction_4;

architecture chenille of fonction_4 is

begin

    process(clk_in, key2)
    begin
        if key2 = '0' then
            hex0 <= "0000000";
            hex1 <= "0000000";
            hex2 <= "0000000";
            hex3 <= "0000000";

        elsif rising_edge(clk_in)then
            case hex0 is
                when "1000010" => -- G sur hex0
                    hex0 <= "1111001"; -- I
                    hex1 <= "1000010"; -- G
                    hex2 <= "1111111";
                    hex3 <= "0000011";
                when "1111001" => -- I sur hex0
                    hex0 <= "0000111";-- t
                    hex1 <= "1111001";-- I
                    hex2 <= "1000010";-- G
                    hex3 <= "1111111";
                when "0000111" => -- t sur hex0
                    hex0 <= "0001011";-- h
                    hex1 <= "0000111";-- t
                    hex2 <= "1111001";-- I
                    hex3 <= "1000010";-- G
                when "0001011" => -- h sur hex0
                    hex0 <= "1000001";-- U
                    hex1 <= "0001011";-- h
                    hex2 <= "0000111";-- T
                    hex3 <= "1111001";-- I
                when "1000001" => -- U sur hex0
                    hex0 <= "0000011";-- b
                    hex1 <= "1000001";-- u
                    hex2 <= "0001011";-- h
                    hex3 <= "0000111";-- t
                when "0000011" => -- b sur hex0
                    hex0 <= "1111111";-- rien
                    hex1 <= "0000011";-- b
                    hex2 <= "1000001";-- u
                    hex3 <= "0001011";-- h
                when others => -- rien sur hex0
                    hex0 <= "1000010";-- rien
```

```
                    hex1 <= "1111111";-- rien
                    hex2 <= "0000011";-- b
                    hex3 <= "1000001";-- U

            end case;
        end if;
    end process;

end chenille;
```

- Chaque afficheur est un buffer car a chaque conditions on va lire la lettre afficher sur l'HEX0 pour changer d'état les 4 afficheurs.
- Quand une lettre est afficher sur l'HEX0 ont fait décaler les lettres de un vers afficheur vers la gauche.

Problèmes :

Ce code nous as posé problème car on voulait que au tout début le mot apparaisse ainsi :

				G
			G	I
		G	I	T
	G	I	T	H
G	I	T	H	U
I	T	H	U	B
T	H	U	B	
H	U	B		
U	B			
B				
				G

Mais lors du codage on avait :

```
when => --rien sur HEX0
    HEX0 <= -- rien
    HEX1 <= -- b
    HEX2 <= -- u
    HEX3 <= -- h
when => -- rien sur HEX 0
    HEX0 <= -- rien
    HEX1 <= -- rien
    HEX2 <= -- b
```

```
    HEX3 <= -- u
when => -- rien sur HEX0
    HEX0 <= -- rien
    HEX1 <= -- rien
    HEX2 <= -- rien
    HEX3 <= -- b
```

On avait plusieurs fois `when "000000" =>` cependant la carte faisait que buger, elle ne devait pas savoir quoi faire.

Alors on as décider que lorsque le programme commence on allait afficher directement :

H	**U**	**B**		**G**
U	B		G	I
B		G	I	T
	G	I	T	H
G	I	T	H	U
I	T	H	U	B
T	H	U	B	
H	U	B		G

Cela nous as éviter d'avoir plusieurs fois `when "000000" =>`

Cependant cela peut paraitre bizarre de commencer directement en voyant la fin du mot a gauche et la première lettre du mot a droite.

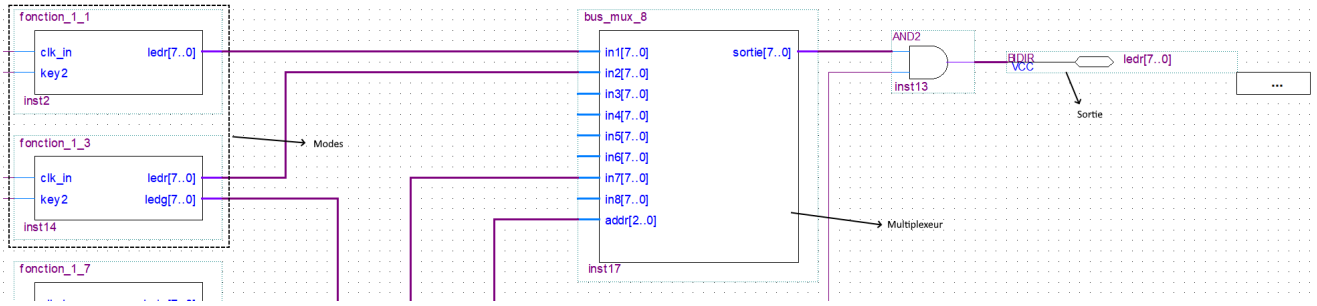
Mais c'était la manière la plus simple que nous avions trouvé.

Bus Multiplexeur :

> le but du multiplexeur est de pouvoir choisir le mode d'affichage en reliant la sortie d'un mode avec la sortie.

Emplacement et fonctionnement

Emplacement



Ils se situe entre les modes et la sortie.

Fonctionnement

En fonction de l'entré d'adresse `addr` une des entré sera affiché sur la sortie, comme dans cette exemple :
Si dans l'entré `addr` on trouve `000` alors en sortie on trouveras la valeur de l'entré `in1`, si on met `001` ce sera la valeur de l'entré `in2`.

Le code

```
-- Importations des librairies
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

-- Definition des entrées
entity bus_mux_7 is
port(
    sortie : out std_logic_vector(6 downto 0);
    in1 : in std_logic_vector(6 downto 0);
    in2 : in std_logic_vector(6 downto 0);
    in3 : in std_logic_vector(6 downto 0);
    in4 : in std_logic_vector(6 downto 0);
    in5 : in std_logic_vector(6 downto 0);
    in6 : in std_logic_vector(6 downto 0);
    in7 : in std_logic_vector(6 downto 0);
    in8 : in std_logic_vector(6 downto 0);
    addr : in std_logic_vector(2 downto 0)

);
end bus_mux_7;

-- Le programme
architecture multiplexeur of bus_mux_7 is
begin

    process (addr, in1, in2, in3, in4, in5, in6, in7, in8)-- En cas de changement de l'une de ces entrées nous allons procédé a l'actualisation du code suivant
```

```
    begin
        case addr is-- On se base sur la valeur d'adresse
            when "000" => sortie <= in1;-- Si la valeur d'adresse est 000
alors la sortie sera a l'état de in1
            when "001" => sortie <= in2;
            when "010" => sortie <= in3;
            when "011" => sortie <= in4;
            when "100" => sortie <= in5;
            when "101" => sortie <= in6;
            when "110" => sortie <= in7;
            when "111" => sortie <= in8;
            when others => sortie <= in1;
        end case;

    end process;
end multiplexeur;
```

Dans le code on retrouve bien la même chose que dans l'exemple cité précédemment.

Problèmes rencontrés

Lors de la conception de ce multiplexeur nous avons rencontré des problème avec le modèle Busmux étant inclus dans Quartus II, en effet il n'était doté que de 2 entré et donc d'une seul entré d'adresse, nous avons donc décidé de les enchainé en pyramide mais lors du teste de sont fonctionnent peut importe la valeur d'entré et d'adresse la valeur de sortie était à 1.

Conclusion

Le multiplexeur sert à relier les modes a la sortie avec une possibilité du choix de l'affichage. Suites aux testes de la carte il n'y a pas eu de problème de fonctionnement ou de bugs.