Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

TP 1 Entrées / Sorties

Systèmes Numériques

Auteurs :
Marc ROTEN
Vincent VONLANTHEN

Professeur :
Nicolas SCHROETER

0,1

30 octobre 2018

Table des matières

Laboratoire 1 : Entrées / Sorties

Noms des étudiants :	Marc Roten	Vincent Vonlanthen
Date du laboratoire :	26.09.18	

Objectifs

- Familiarisation avec le matériel et les instruments du laboratoire
- Familiarisation avec les entrées / sorties connectées à un système numérique
- Rédaction d'un rapport

Travail

Pour les travaux pratiques du cours systèmes numériques, nous allons utiliser

- une carte dénommée MiniZed,
- l'environnement de développement VHDL de Xilinx dénommé Vivado,
- des composants d'entrée / sortie tels que LED, interrupteurs, affichage LCD, etc.
- et différents capteurs (température, humidité, infrarouge, etc.)

Ce premier labo vise la prise en main des outils Vivado de Xilinx pour la synthèse et la simulation VHDL. Le deuxième objectif correspond à la mise en œuvre d'un affichage 4x7 segments avec point décimal sachant que les signaux de pilotage de chaque LED sont multiplexés.

Théorie

La carte MiniZed embarque un circuit Zynq qui est un System on Chip (SoC) : il est composé d'un processeur ARM appelé Processing System (PS), d'une série d'interfaces de communication ainsi que d'une partie FPGA programmable dénommée Programmable Logic (PL).
Nous allons nous concentrer durant ce semestre sur la partie FPGA du SoC.

La figure suivante présente les broches et leur nom utilisables par la partie PL pour y connecter des entrées/sorties ou des périphériques :

FIGURE 1 – Donnée du Labo

Etape 1

a) Déterminer la valeur minimum de la résistance à mettre en série avec les LEDs (voir datasheet A-5461AS.pdf)
b) Dessiner un schéma de connexion des broches et de l'affichage 4*7segments.
Faire valider ce schéma par le professeur.

Remarque :
Le courant va passer au travers des broches d'entrée/sortie du Zynq. Chaque broche ne supporte qu'un courant limité de 10mA.

Etape 2

c) Développer en VHDL un décodeur 4x7 qui permet d'afficher les chiffres BCD de 0 à 9 sur les 7 segments

Etape 3

L'affichage dispose de signaux d'entrée multiplexés/partagés (broches 11, 7, 4, 2, 1, 10, 5, 3) qui permettent d'allumer les LEDs des différents digits.

Pour allumer les LEDs d'un digit en particulier, il est nécessaire de piloter sa sortie. Les sorties sont les broches 12, 9, 8 et 6.
Ainsi pour allumer des segments d'un digit, il faut mettre à l'état haut les signaux d'entrée correspondants aux segments à allumer et mettre le signal de sortie correspondant au digit à 0. Dans ces conditions, le courant circulera dans les diodes entre l'anode et la cathode. Si la sortie est à 1, alors aucun courant ne pourra circuler.

Pour gérer les 4 digits, il est nécessaire de traiter chaque digit les uns après les autres en appliquant les opérations suivantes :

- Mettre à l'état haut les signaux d'entrée des segments à allumer et mettre à l'état bas la sortie correspondante du digit ; les autres sorties des autres digits sont mises à l'état haut
- Passer au prochain digit

Résultat demandé

- Fournir un rapport par groupe dans le délai maximum de 3 semaines
  Ce rapport doit comprendre :
  - cette feuille de donnée avec les noms des étudiants et la date du TP
  - une courte introduction (avec vos mots !)
  - un mode d'emploi de l'utilisation des outils Vivado
  - une description de l'approche de résolution du travail
  - la présentation des résultats du travail demandé
  - une courte conclusion
  - les signatures des étudiants

FIGURE 2 – Donnée du Labo en détail

1 Introduction

Ce travail va nous servir de base pour la prise en main de la nouvelle machine virtuelle sour Ubuntu, ainsi que l'utilisation de Vivado pour la création de code VHDL en vue de la simulation et de l'implémentation sur notre FPGA, la miniZed. ça va en outre être un bon refresh en VHDL.

+ développer driver 4x7 seg--

2 Mode d'emploi

← intro

2.1 Création du projet

Mettre du texte + indiquer comment arriver à la fenêtre File → ... + évoquer les options à cocher/sélectionner

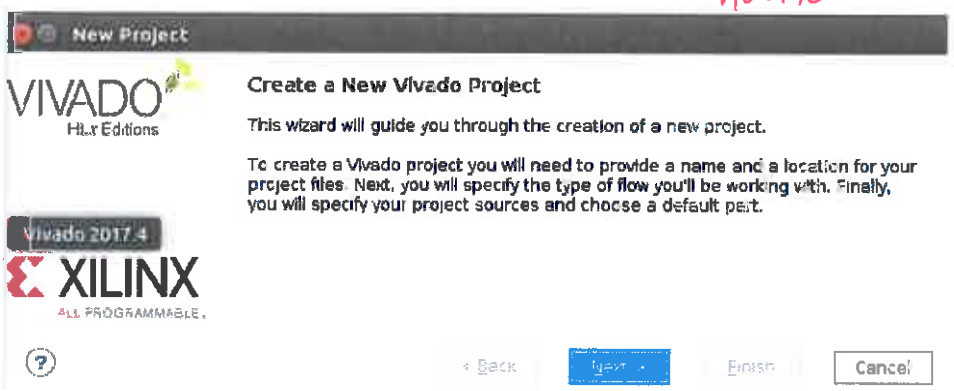


FIGURE 3 – Création du projet

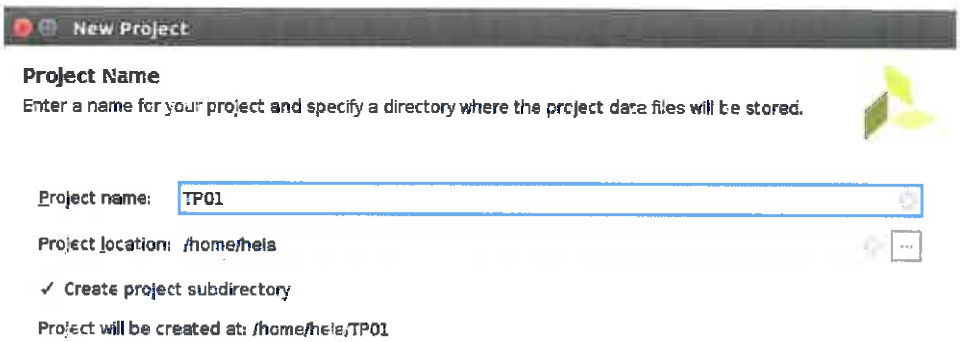


FIGURE 4 – Dénomination du projet

FIGURE 5 – Choix du type de projet

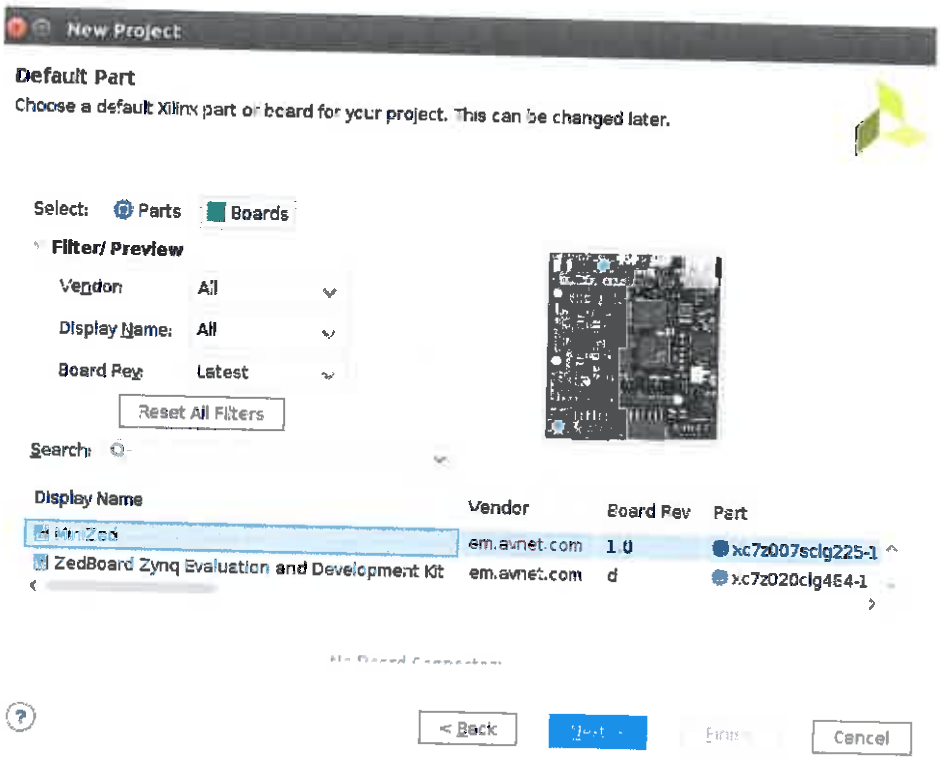


FIGURE 6 – Séléction de la bonne board

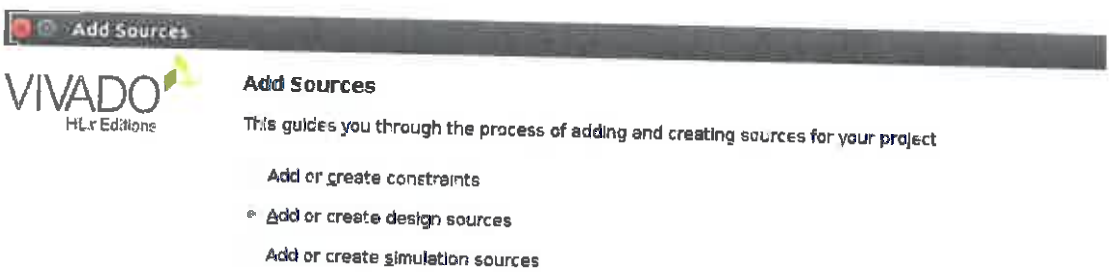


FIGURE 7 – Séléctionner add or creat design sources

2.2 Rajouter des composants à notre architecture

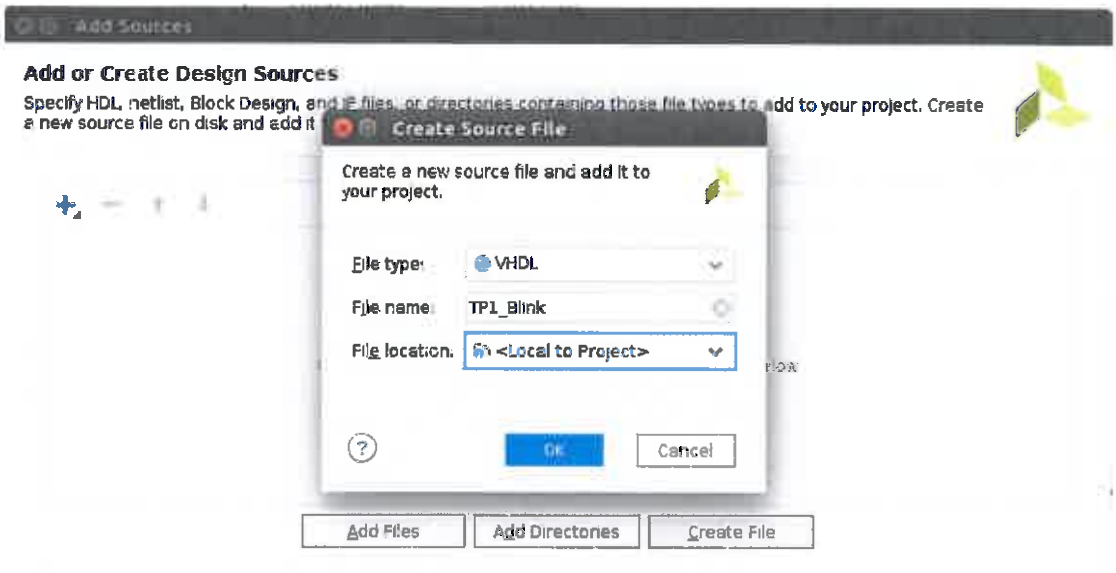


FIGURE 8 – Add sources, type VHDL

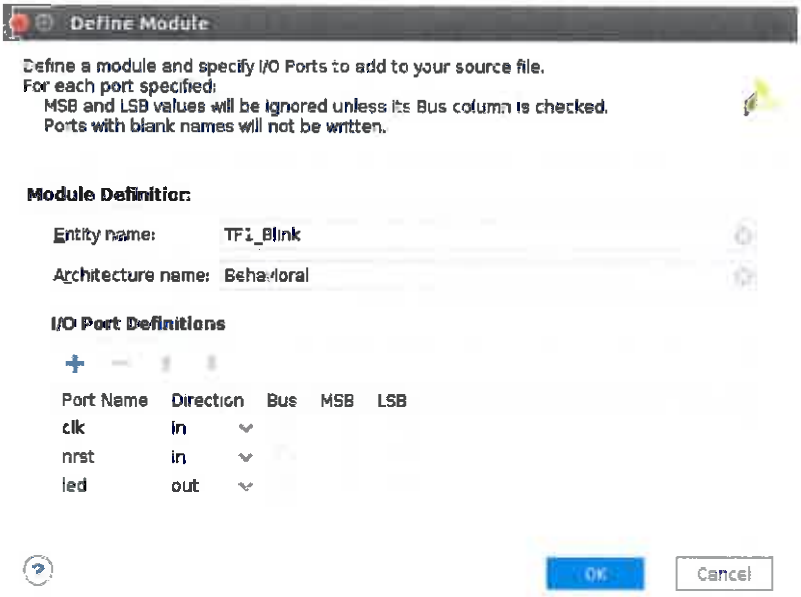


FIGURE 9 – Nommer notre composant et ajouter les I/O

2.3 Codage VHDL

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity TP1_Blink is
35      Port ( clk : in STD_LOGIC;
36             nrst : in STD_LOGIC;
37             led : out STD_LOGIC);
38  end TP1_Blink;
39
40  architecture Behavioral of TP1_Blink is
41      signal compteur : unsigned(31 downto 0);
42  signal rst, fin_compteur : std_logic;
43  constant freq_horloge : integer := 50_000_000;
44  constant delai_0_5s : integer := freq_horloge/2;
45  signal led_s : std_logic;
46  begin
47  led <= led_s;
48  rst <= not nrst;
49
50  fin_compteur <= '1' when compteur = to_unsigned(delai_0_5s, compteur'length) else '0';
51
52  process(clk, rst)
53  begin
54      if rst = '1' then
55          compteur <= (others => '0');
56          led_s <= '1';
57      elsif rising_edge(clk) then
58          if fin_compteur = '1' then
59              compteur <= (others => '0');
60              led_s <= not(led_s);
61          else
62              compteur <= compteur + 1;
63          end if;
64      end if;
65  end process;
66
67  end Behavioral;
```

FIGURE 10 – Réaliser le code en VHDL

2.4 Vérification de fonctionnement

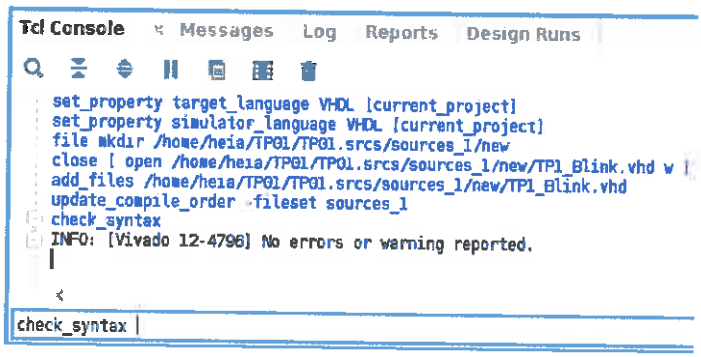
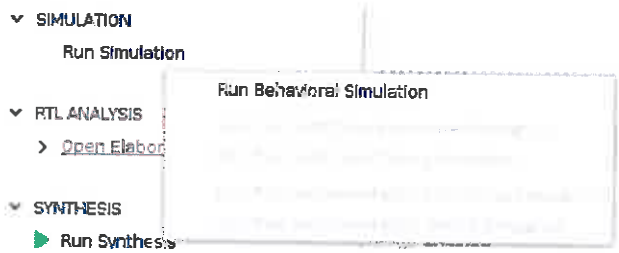


FIGURE 11 – Réaliser le check syntax



Choix d'une autre déclaration pour la Simulation

FIGURE 12 – Run Behavioral Simulation

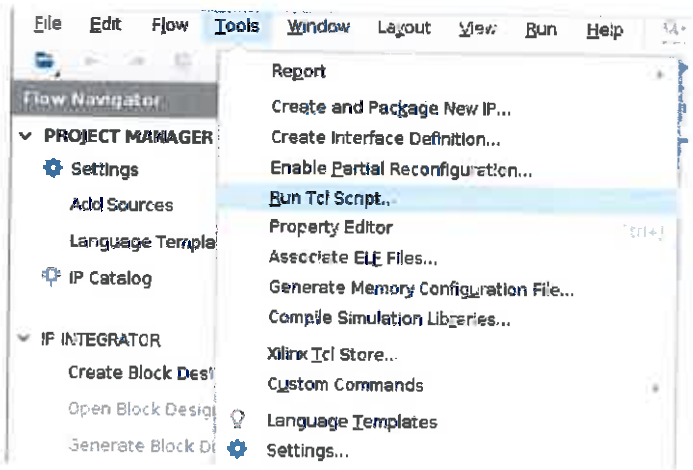


FIGURE 13 – Run TCL Script

ajouter au début d'un script TCL de simulation

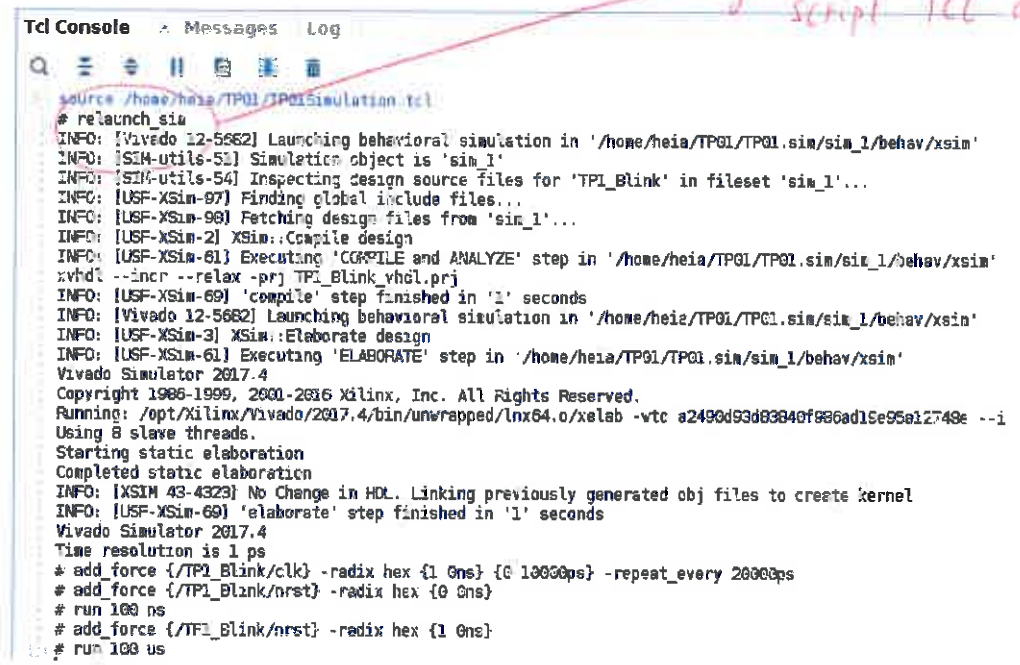


FIGURE 14 – Exécution du script TCL

2.5 Création du Design

FIGURE 15 – Create block design

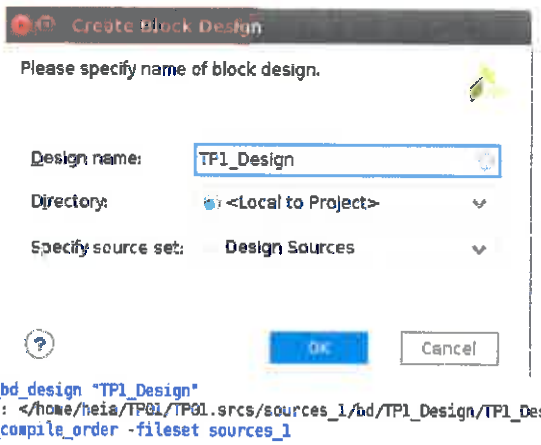


FIGURE 16 – Create block design, Design's name

FIGURE 17 – Ajouter des composants à notre architecture

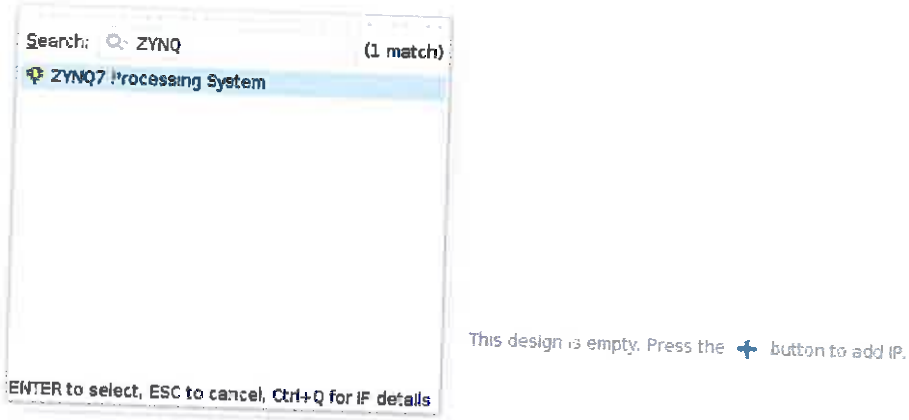


FIGURE 18 – Selectionner ZYNQ7

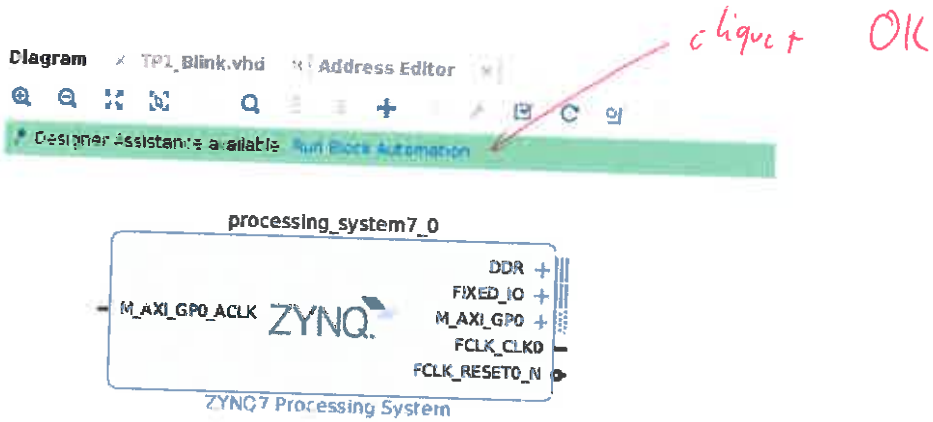


FIGURE 19 – Résultat Graphique de notre design

2.6 Connexion entre nos composants

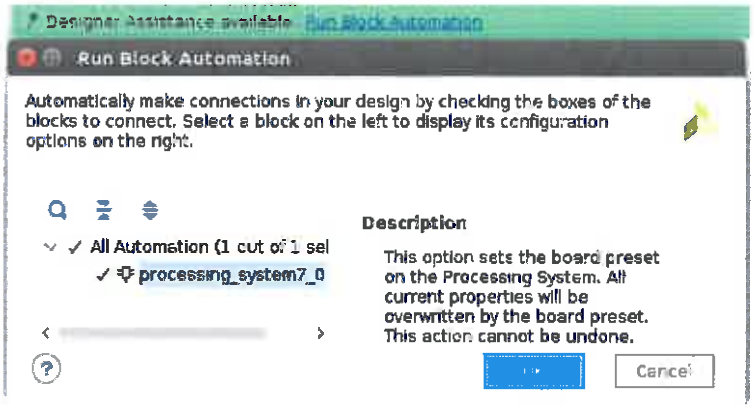


FIGURE 20 – activer 'run block automation'

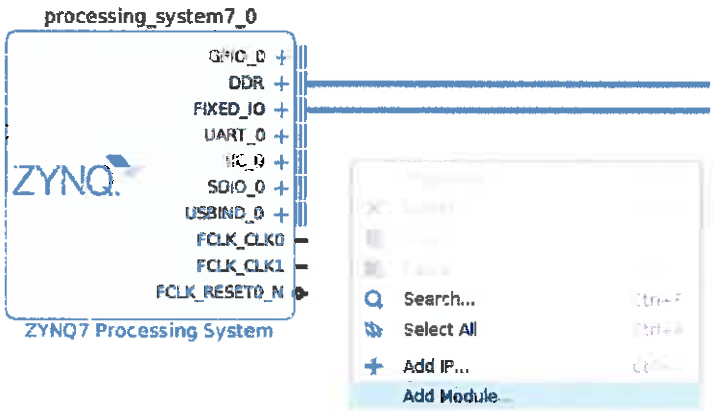


FIGURE 21 – Right click into Add module

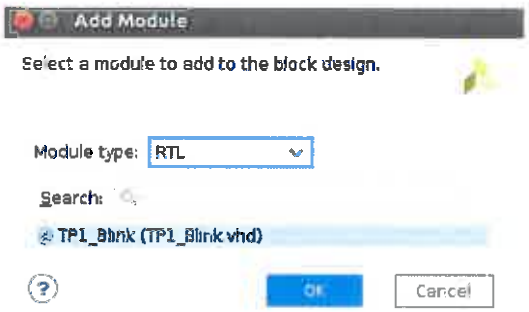


FIGURE 22 – Select your module

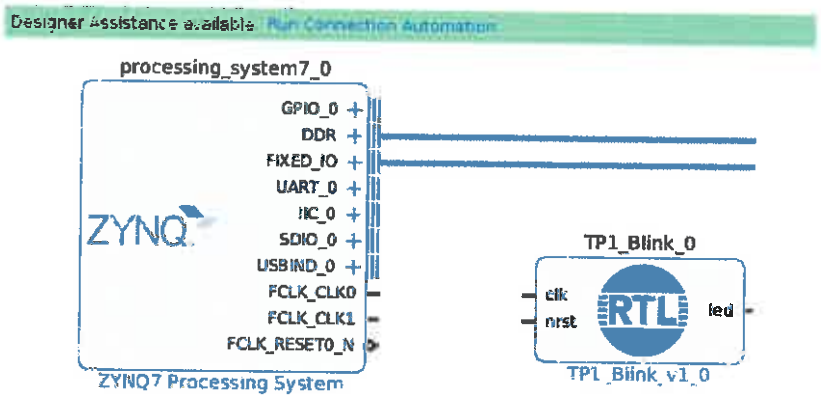


FIGURE 23 – Click Run connection Automation

Connecter E/S

2.7 Configurer une Sortie

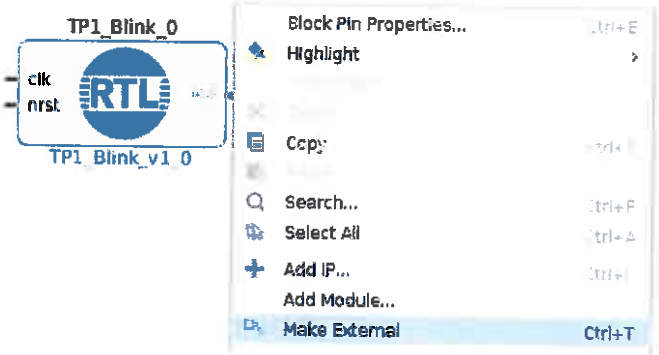


FIGURE 24 – Right click Led to make it as an Output

2.8 Création du wrapper HDL

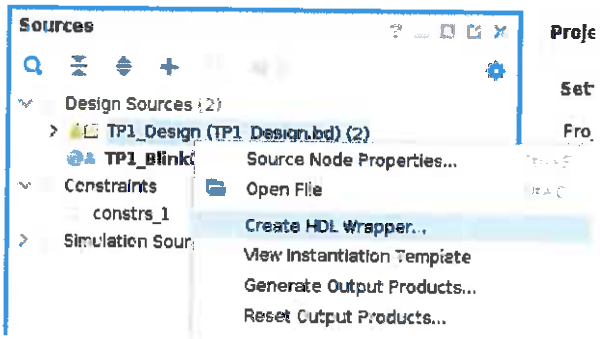


FIGURE 25 – Create HDL Wrapper

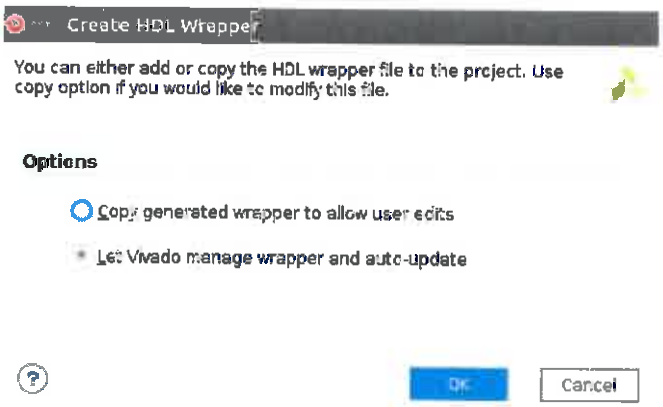


FIGURE 26 – Let vivado Manage

2.9 Lancement de la synthèse

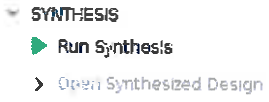


FIGURE 27 – Run synthesis

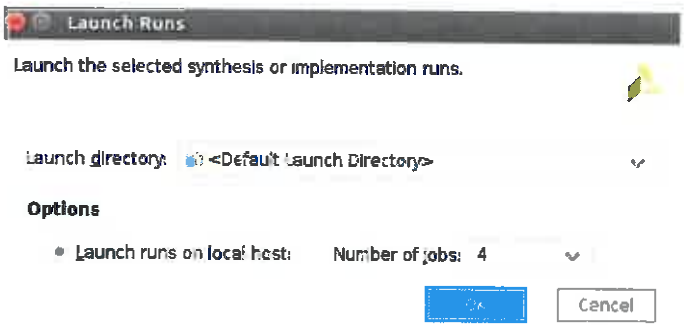


FIGURE 28 – Run synthesis

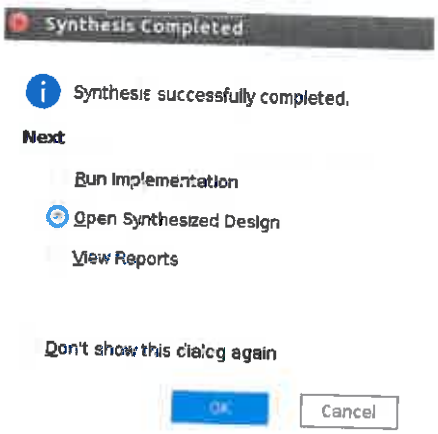


FIGURE 29 – Run synthesis

2.10 Mapping I/O

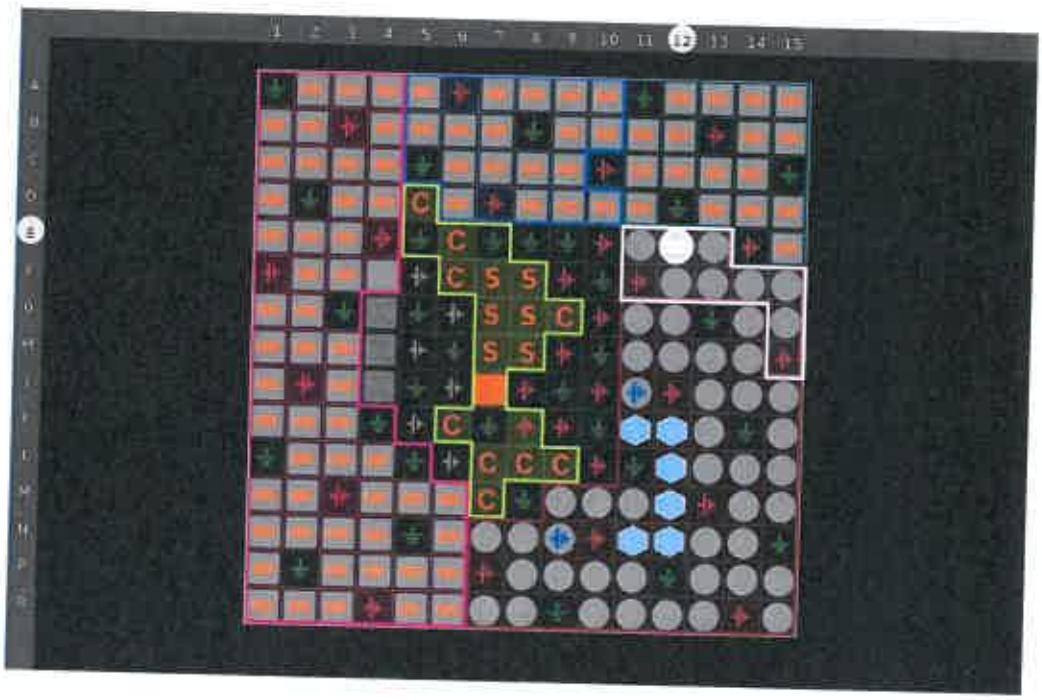


FIGURE 30 – Placer ses éléments

Montrer signal #broches Tension

Fichier contrainte

2.11 Génération du bitstream

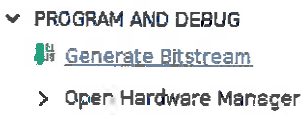


FIGURE 31 – Génération du bitstream

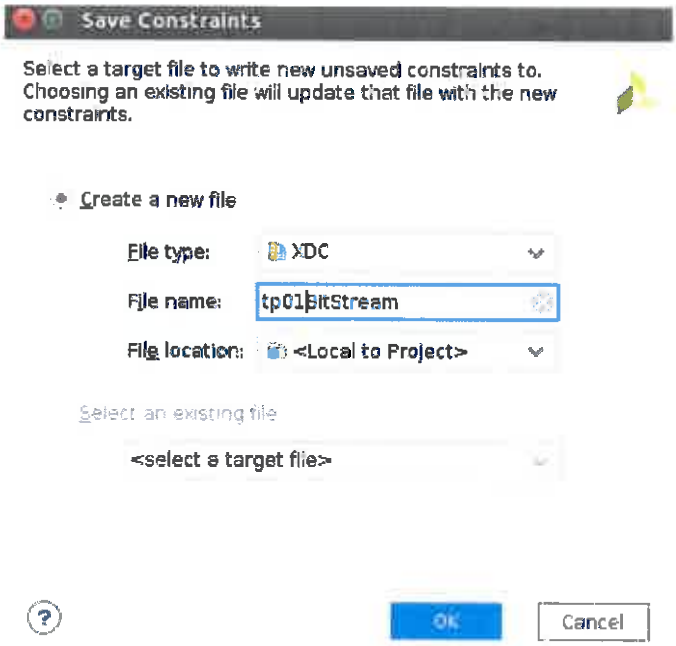


FIGURE 32 – Génération du bitstream

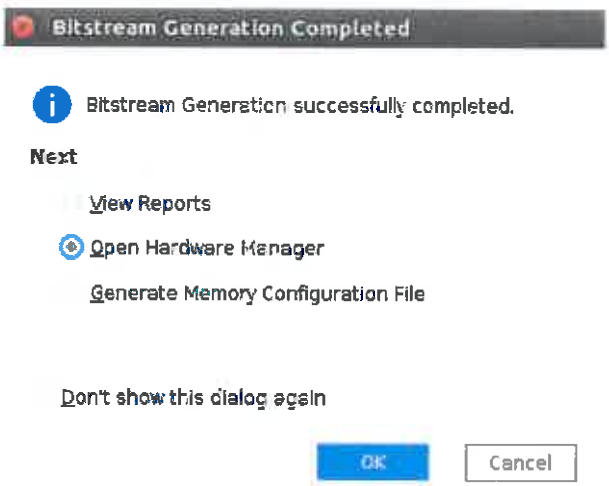


FIGURE 33 – Hardware manager

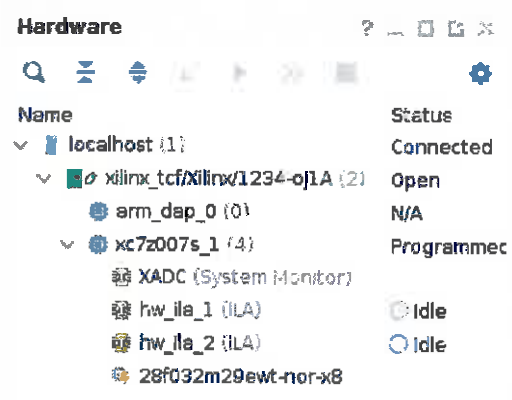


FIGURE 34 – Vérifier la détéction de notre board

FIGURE 35 – Cliquer sur Program Device

3 Etape 1 : préparation

3.1 Partie A

Déterminer la valeur minimum de la résistance à mettre en série avec les LEDs (voir datasheet A-5461AS.pdf)

On voit en figure 36 au point 2 que le courant souhaité est de 20mA. On voit aussi, au point 1, que le voltage souhaité est de 2.1V.

Forward voltage IF = 20mA	(Min.)	V_F	1.8	V
	(Typ.)	V_F	1 2.1	V
	(Max.)	V_F	2.4	V
Reverse current VR = 5V	(Max.)	I_R	20	µA
Optical efficiency IF = 20mA 2	(Typ.)	η_{OPT}	-	lm/W

FIGURE 36 – Courant optimal

Maximum Ratings

Parameter	Symbol	Value	Unit
Operating temperature	T_{OP}	-35 ~ 85	℃
Storage temperature	T_{STG}	-35 ~ 85	℃
Forward current (TA=25 ℃)	I_F	30	mA per seg
Peak forward current (T_A=25 ℃) *1	I_{PF}	120	mA per seg
Reverse voltage (TA=25 ℃)	V_R	5	V per seg
Power consumption (TA=25 ℃)	P	80	mW per seg

*1 at 1/10 Duty Cycle

FIGURE 37 – Forward current

On voit en figure 37, que le Forward Current est de 30mA. Mais comme indiqué dans la consigne **Le courant va passer au travers des broches d'entrée/sortie du Zynq. Chaque broche ne supporte qu'un courant limité de 10mA.** On se limitera donc à un FWd Voltage de 10mA.

$$R = \frac{Vcc - FwdVoltage}{ForwardCurrent}$$

$$R = \frac{3.3 - 2.1}{10 * 10^{-3}} = 120\Omega$$

3.2 Partie B

Dessiner un schéma de connexion des broches et de l'affichage 4*7segments. Faire valider ce schéma par le professeur.

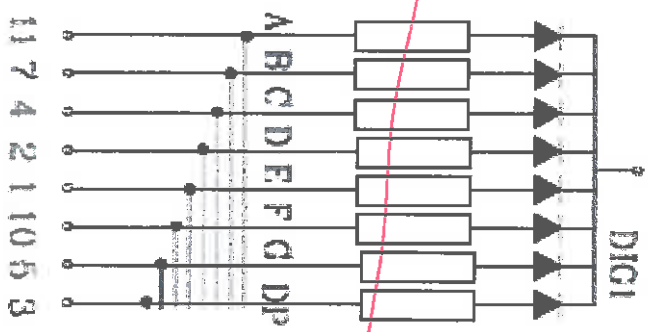


FIGURE 38 – Schéma résistance diode

Pour que le courant soit assez fort, mais pas trop non plus pour ne pas détruire la LED, on place une résistance de 105 Ω en série avec la sortie et celà sur chaques sorties

4 Etape 2 : code VHDL

4.1 Développer en VHDL un décodeur 4x7 qui permet d'afficher les chiffres BCD de 0 à 9 sur les 7 segments

Tout d'abord un chiffre BCD , ou "Binary Coded Decimal" en anglais signifiant décimal codé binaire, est un chiffre utilisé en électronique qui est codé sur quatre bits.

Ci-dessous, en figure 39, on retrouve un tableau des différentes représentations binaires des chiffres allant de 0 à 9.

		Binaire			
chiffre décimal	0	0	0	0	0
	1	0	0	0	1
	2	0	0	1	0
	3	0	0	1	1
	4	0	1	0	0
	5	0	1	0	1
	6	0	1	1	0
	7	0	1	1	1
	8	1	0	0	0
	9	1	0	0	1

FIGURE 39 – Représentation binaire 0 à 9

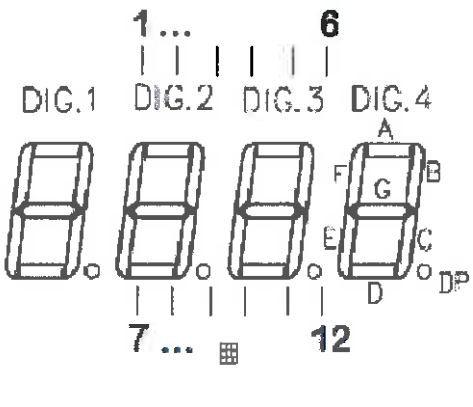
Non décodeur
BCD → 7 Seg

4.2 Code de notre Tester

Voir Annexe

5 Etape 3 : Travail en lui même

En introduction, on s'interesse à la disposition des différentes pin qui permettra d'avoir un câblage correct pour alimenter les 4x7 segments et afficher les valeurs envoyées au bon endroit. Voir figure 40



Pin	topic
1	DIG1
2	A
3	F
4	DIG2
5	DIG3
6	B
7	E
8	D
9	Dp
10	C
11	G
12	DIG4

FIGURE 40 – Disposition des segments de A à G

5.1 Concevoir et de décrire une architecture (par exemple : machines d'états, compteurs, etc.) qui réalise le pilote de l'affichage 4x7segments avec le point décimal

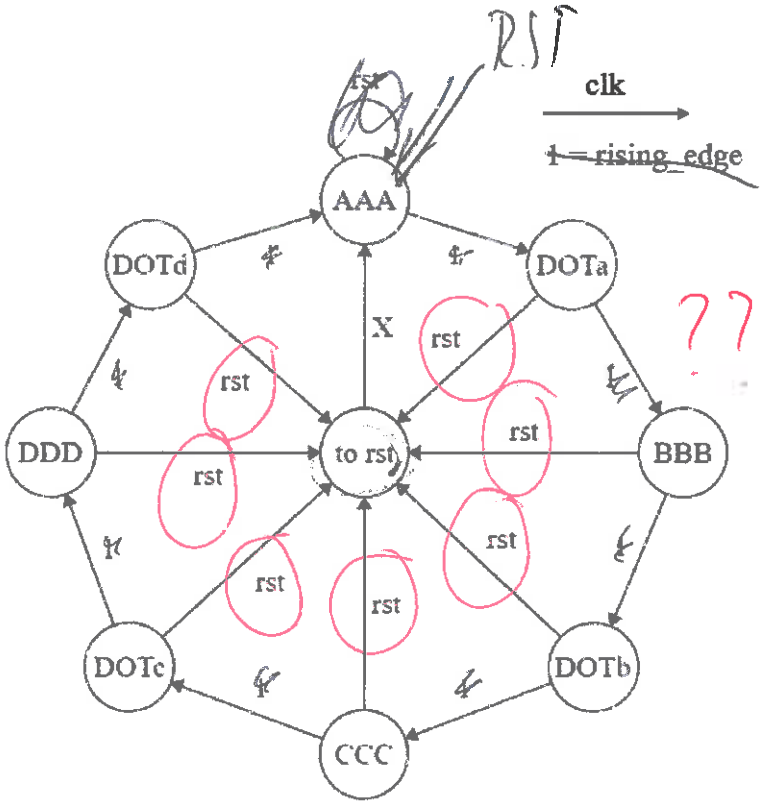


FIGURE 41 – Machine d'état du balayage

dot est piloté en même temps que les autres

Segment avec la sortie ⇒ pas d'état spécial pour dot

La machine d'état ci-dessus représente le passage dans les différents états du balayage de notre afficheur 4x7 segements. L'état AAA correspond au premier digit de notre afficheur, l'état BBB au deuxième, l'état CCC au troisième et l'état DDD au quatrième. Les états intermédiaires correspondent aux dots de l'afficheur, respectivement DOTa pour le premier dot, DOTb pour le deuxième dot et ainsi de suite jusqu'à DOTd. Lorsque le rst est appelé, on retourne directement à l'état AAA peu importe à quel état on se trouve.

C'est pas ce que vous avez dessiné

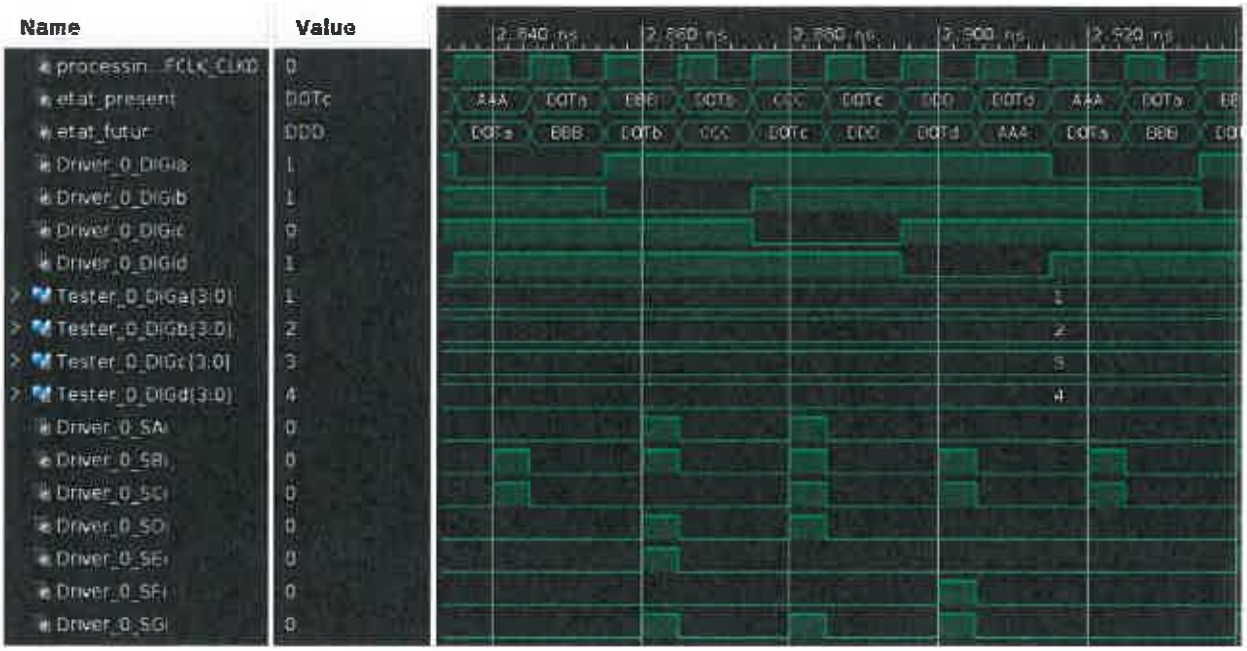


FIGURE 42 – Datagramme de notre architecture

Sur la figure 42, chaque Digit, de DIGia à DIGid, correspond, s'il est zéro, à un Digit éteint. Et les sorties de SAi jusqu'à SGi correspondent aux digits de notre 4x 7 segments.

Décrire les autres blocs { entrées / sorties ; fonctions réalisés

5.2 Développer le code VHDL

Voir Annexe.

5.3 Valider le bon fonctionnement

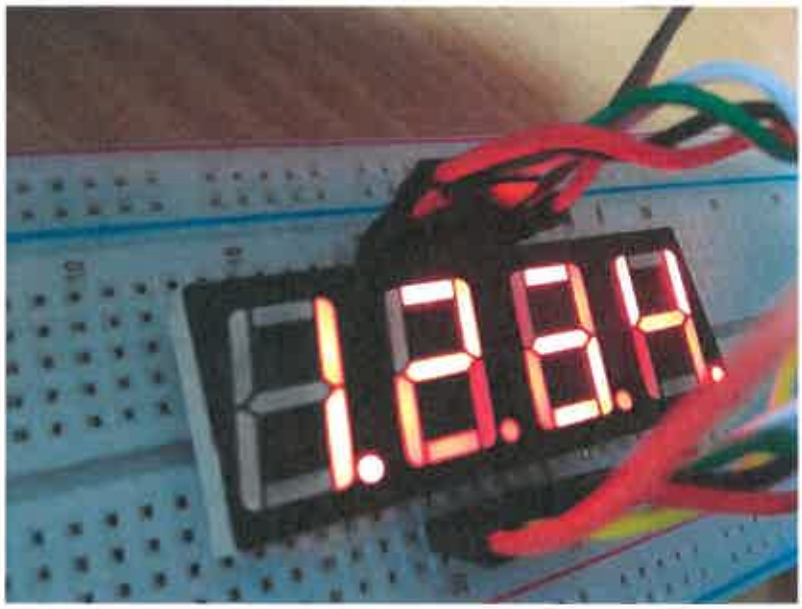


FIGURE 43 – Affichage des digits et des dots

Le but final de notre code est d'afficher les digits 1 2 3 4 et les dots 1 4 sur notre afficheur 4x7 segments. Mais comme on peut l'apercevoir sur la figure 43, il y a des effets rémanents sur notre afficheur ce qui biaise la lecture du résultat.

Manque tests

5.4 En maintenant le même fonctionnement, est-ce possible de diminuer le nombre de signaux entre la FPGA et l'affichage ? Si oui, comment ?

Non, notre code est optimisé dans le cas de son utilisation lors de ce travail pratique. Mais une piste d'amélioration possible serait de multiplexer nos différents signaux de sortie et d'en faire un bus. Mais dans le cadre de ce TP, nous n'avons pas jugé cette alternative optimale.

Si

Mais pas réponse attendue

6 Conclusion

Par ce travail pratique, nouv avons pu prendre en main les nouveaux outils par la création du mode d'emploi de Vivado. Nous avons aussi pris en main la nouvelle Board mise à disposition, ainsi que les différents éléments tels que les résistances ou les afficheurs 7 segments. Ce TP fut très intéressant dans le sens où il nous a permis de reprendre, et consolider, les bases du VHDL vues précédemment.

Marc Roten

Vincent Vonlanthen

Approved :

Nicolas Schroeter
Professeur HEIA-FR

7 Annexe

7.1 Code du Tester

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Tester is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           dp1 : out STD_LOGIC;
           dp2 : out STD_LOGIC;
           dp3 : out STD_LOGIC;
           dp4 : out STD_LOGIC;
           DIGa : out STD_LOGIC_VECTOR(3 downto 0);
           DIGb : out STD_LOGIC_VECTOR(3 downto 0);
           DIGc : out STD_LOGIC_VECTOR(3 downto 0);
           DIGd : out STD_LOGIC_VECTOR(3 downto 0));
end Tester;

architecture Behavioral of Tester is

begin

DIGd<="0100";DIGc<="0011";DIGb<="0010";DIGa<="0001";
dp4<='1';dp3<='0';dp2<='0';dp1<='1';

end Behavioral;
```

7.2 Code du Driver

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;


entity Driver is
    Port ( BCD1 :    in STD_LOGIC_VECTOR(3 downto 0);
           BCD2 :    in STD_LOGIC_VECTOR(3 downto 0);
           BCD3 :    in STD_LOGIC_VECTOR(3 downto 0);
           BCD4 :    in STD_LOGIC_VECTOR(3 downto 0);
           DOT1 :    in STD_LOGIC;
           DOT2 :    in STD_LOGIC;
           DOT3 :    in STD_LOGIC;
           DOT4 :    in STD_LOGIC;
           CLK :     in STD_LOGIC;
           RST :     in STD_LOGIC;

           DIGia:    out STD_LOGIC;
           SAi :     out STD_LOGIC;
           SFi :     out STD_LOGIC;
           DIGib:    out STD_LOGIC;
           SBi :     out STD_LOGIC;
           SEi :     out STD_LOGIC;
           SDi :     out STD_LOGIC;
           DPi :     out STD_LOGIC;
           SCi :     out STD_LOGIC;
           SGi :     out STD_LOGIC;
           DIGid :   out STD_LOGIC);
end Driver;

architecture Behavioral of Driver is

        TYPE etats is(AAA,DOTa,BBB,DOTb,CCC,DOTc,DDD,DOTd);
        signal etat_present,etat_futur:etats;
begin

MachineEtatDecoder:process(etat_present,DOT1,DOT2,DOT3,DOT4)
begin

case etat_present is
        when AAA =>
                DIGia<='0';DIGib<='1';DIGic<='1';DIGid<='1';DPi<='0';
```

Non pas machine état

Faire un process pour les sorties
Ne pas mélanger état futur avec les affectation des signaux de sortie

```
        when DOTa =>
    DIGia<='0';DIGib<='1';DIGic<='1';DIGid<='1';
        if DOT1='1' then
            DPi<='1';
            else DPi<='0';
        end if;
when BBB =>
        DIGia<='1';DIGib<='0';DIGic<='1';DIGid<='1';DPi<='0';
when DOTb =>
    DIGia<='1';DIGib<='0';DIGic<='1';DIGid<='1';
        if DOT2='1' then
            DPi<='1';
            else DPi<='0';
        end if;
when CCC =>
        DIGia<='1';DIGib<='1';DIGic<='0';DIGid<='1';DPi<='0';
when DOTc =>
    DIGia<='1';DIGib<='1';DIGic<='0';DIGid<='1';
        if DOT3='1' then
            DPi<='1';
            else DPi<='0';
    end if;
when DDD =>
        DIGia<='1';DIGib<='1';DIGic<='1';DIGid<='0';DPi<='0';
when DOTd =>
    DIGia<='1';DIGib<='1';DIGic<='1';DIGid<='0';
    if DOT4='1' then
        DPi<='1';
             else DPi<='0';
        end if;
when others=>
        DIGia<='1';DIGib<='1';DIGic<='1';DIGid<='1';DPi<='0';
end case;
end process MachineEtatDecoder;

EtatBalayage:process(clk,rst)
begin
    if rst='0' then
        etat_present<=AAA;
    elsif rising_edge(clk) then
        etat_present<=etat_futur;
    end if;
end process EtatBalayage;
```

```
combiBalayage:process(etat_present)
begin
case etat_present is
        when AAA =>
                etat_futur<=DOTa;
        when DOTa=>
                etat_futur<=BBB;
        when BBB =>
                etat_futur<=DOTb;
        when DOTb =>
                etat_futur<=CCC;
        when CCC =>
            etat_futur<=DOTc;
        when DOTc =>
            etat_futur<=DDD;
        when DDD =>
            etat_futur<=DOTd;
        when DOTd =>
            etat_futur<=AAA
        when others =>
            etat_futur<=etat_present;
END case;
end process combiBalayage;

toDisplay:process(clk)
begin
--init a 0 a chaque coup de clock
SAi<='0'; SBi<='0'; SCi<='0'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';

if falling_edge(clk)  then

    if etat_present = AAA  then
        case BCD1 is
            when "0000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='0';
            when "0001" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "0010" =>
SAi<='1'; SBi<='1'; SCi<='0'; SDi<='1'; SEi<='1'; SFi<='0';SGi<='1';
            when "0011" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='0';SGi<='1';
            when "0100" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='1';SGi<='1';
            when "0101" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
```

Non car vous restez bloqué dans cet état

Non car ailleurs rising-edge

faire 1 seul décodeur

Pourquoi faire un process séquentiel ?

```
            when "0110" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "0111" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "1000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "1001" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when others =>
SAi<='0'; SBi<='0'; SCi<='0'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
        end case;
    end if;
    if etat_present = BBB  then
        case BCD2 is
            when "0000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='0';
            when "0001" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "0010" =>
SAi<='1'; SBi<='1'; SCi<='0'; SDi<='1'; SEi<='1'; SFi<='0';SGi<='1';
            when "0011" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='0';SGi<='1';
            when "0100" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='1';SGi<='1';
            when "0101" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when "0110" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "0111" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "1000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "1001" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when others =>
SAi<='0'; SBi<='0'; SCi<='0'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
        end case;
    end if;
    if etat_present = CCC  then
        case BCD3 is
            when "0000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='0';
            when "0001" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "0010" =>
```

```
SAi<='1'; SBi<='1'; SCi<='0'; SDi<='1'; SEi<='1'; SFi<='0';SGi<='1';
            when "0011" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='0';SGi<='1';
            when "0100" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='1';SGi<='1';
            when "0101" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when "0110" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "0111" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "1000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "1001" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when others =>
SAi<='0'; SBi<='0'; SCi<='0'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
        end case;
    end if;
    if etat_present = DDD  then
        case BCD4 is
            when "0000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='0';
            when "0001" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "0010" =>
SAi<='1'; SBi<='1'; SCi<='0'; SDi<='1'; SEi<='1'; SFi<='0';SGi<='1';
            when "0011" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='0';SGi<='1';
            when "0100" =>
SAi<='0'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='1';SGi<='1';
            when "0101" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when "0110" =>
SAi<='1'; SBi<='0'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "0111" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
            when "1000" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='1'; SFi<='1';SGi<='1';
            when "1001" =>
SAi<='1'; SBi<='1'; SCi<='1'; SDi<='1'; SEi<='0'; SFi<='1';SGi<='1';
            when others =>
SAi<='0'; SBi<='0'; SCi<='0'; SDi<='0'; SEi<='0'; SFi<='0';SGi<='0';
        end case;
    end if;
```

```
end if;
end process toDisplay;
end Behavioral;
```

Macro Simulation ?