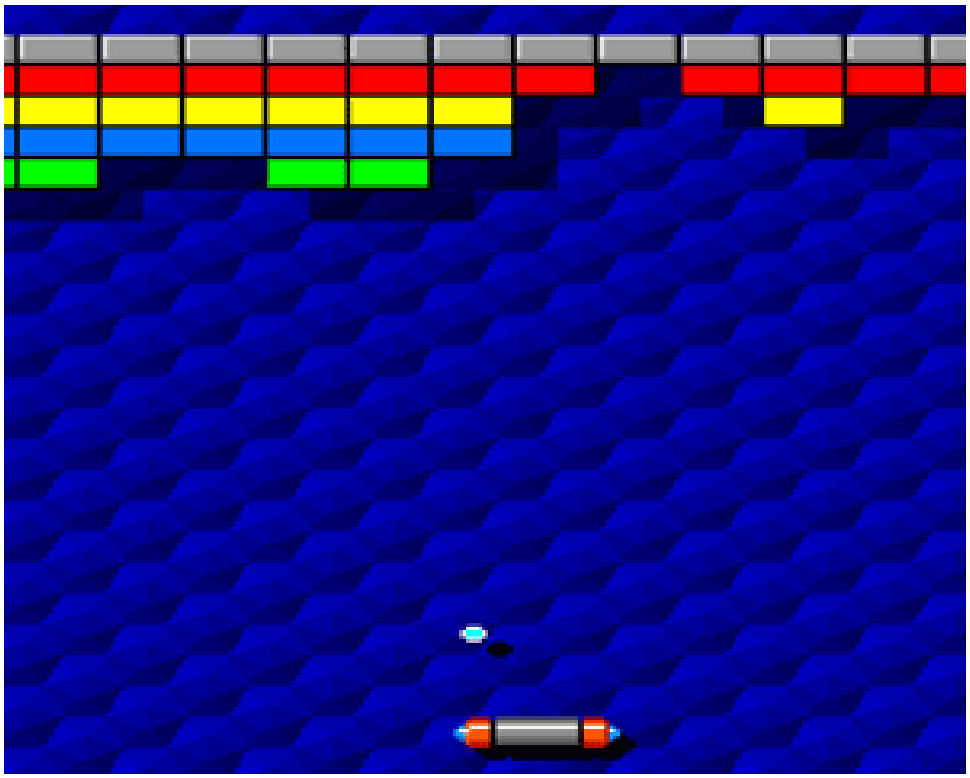
Licence de Sciences et Technologies
Mention Electronique, Energie électrique, Automatique (EEA)

UPMC SORBONNE UNIVERSITÉS

Travaux Pratiques d'Electronique Numérique

Licence de Sciences et Technologie
mention Electronique, Energie électrique et Automatique

hiver 2017



1 Séance 1 : le compteur

Vous devez rendre à la fin de la séance un compte-rendu rédigé dans lequel figure notamment les réponses aux différentes questions posées dans l'énoncé.

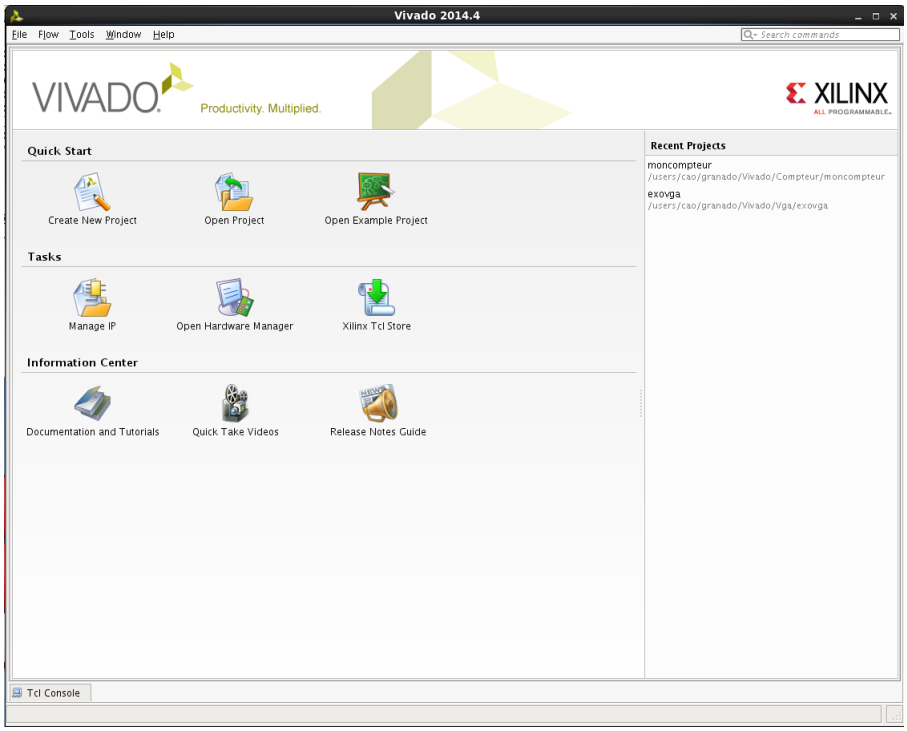
1.1 Préparation

- Télécharger le fichier `seances1et2.zip` sur le site http://bertrand.granado.free.fr/LE201/LE201/Travaux_Pratiques.html
- Copiez le fichier `seances1et2.zip` sur le disque C:/Users/1234567 (si 1234567 est votre numéro d'étudiant)
- Décompresser le fichier `seances1et2.zip`

1.2 Outils pour le VHDL : Une première simulation : le compteur

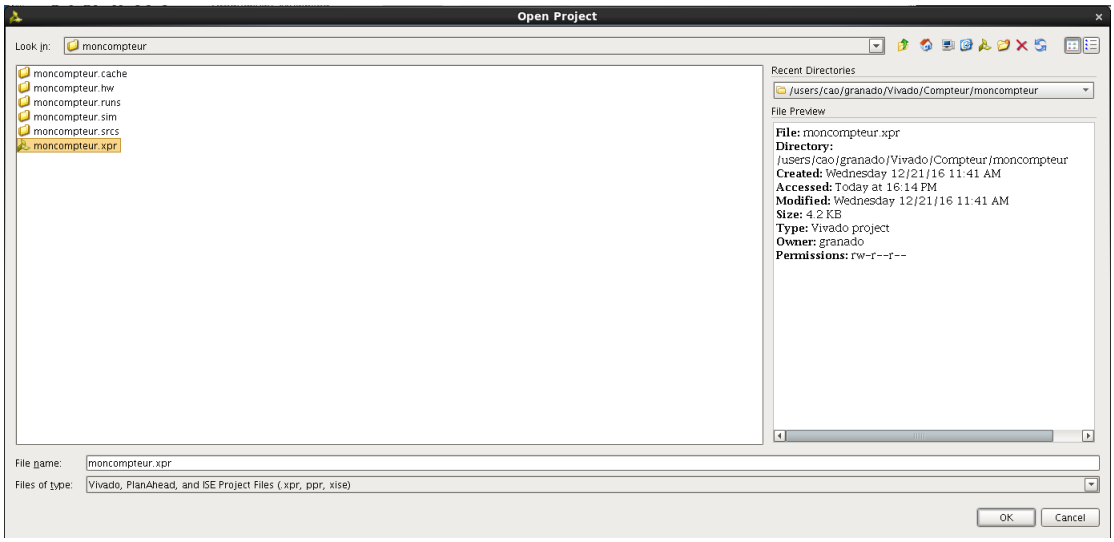
Dans cet exercice nous allons réaliser la simulation d'un compteur, l'énoncé qui suit va pas à pas vous indiquer les différentes étapes à suivre pour réaliser cette simulation.

1. Lancer le logiciel Vivado

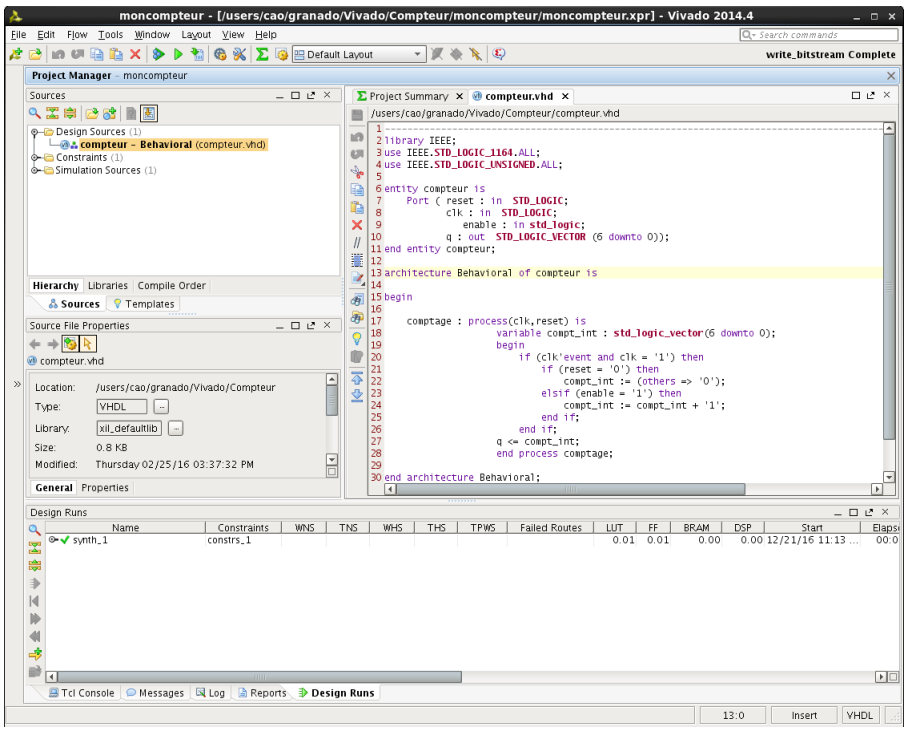


2. Ouvrez le projet `compteur.xpr` en cliquant sur `Open Project` et en allant dans le dossier

`C:/Users/1234567/seances1et2/compteurNexys4` ou `C:/Users/1234567/seances1et2/compteurNexys4DDR` suivant la carte dont vous disposez.



3. Double cliquez sur le fichier `compteur.vhd` dans la fenêtre `Sources`, ce fichier est rangé dans le dossier `Design Sources`.



4. **Q1:** Une fenêtre s'ouvre avec le code VHDL du compteur, analyser ce code et expliquer le.

5. Avant de configurer un FPGA avec un système numérique préalablement décrit en VHDL, il est nécessaire d'en faire une simulation afin de détecter les erreurs. C'est ce que nous allons faire ici.

6. Dans le dossier `Simulation Sources` double-cliquez sur le fichier `TestCompteur.vhd` Ce fichier contient un code VHDL permettant de tester le compteur décrit dans le fichier `compteur.vhd`. Vous allez analyser le contenu de ce fichier :

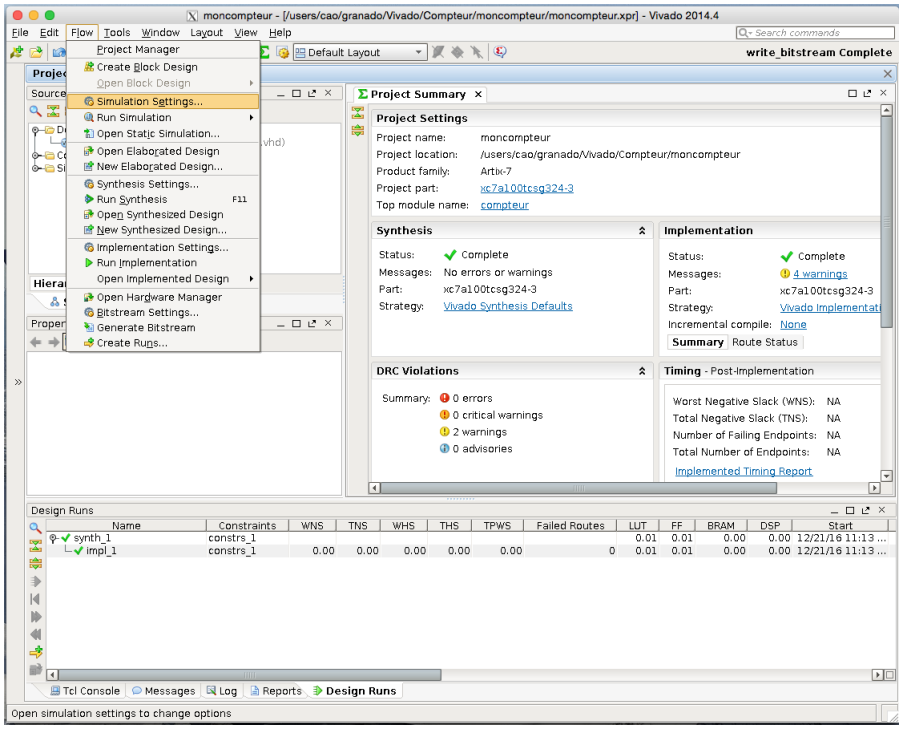
   (a) **Q2:** Identifiez les lignes où le `compteur` à tester est instancié.

   (b) Pour générer des stimuli envoyés au `compteur`, nous utilisons la construction `process`. Un `process` permet de décrire un processus séquentiel.

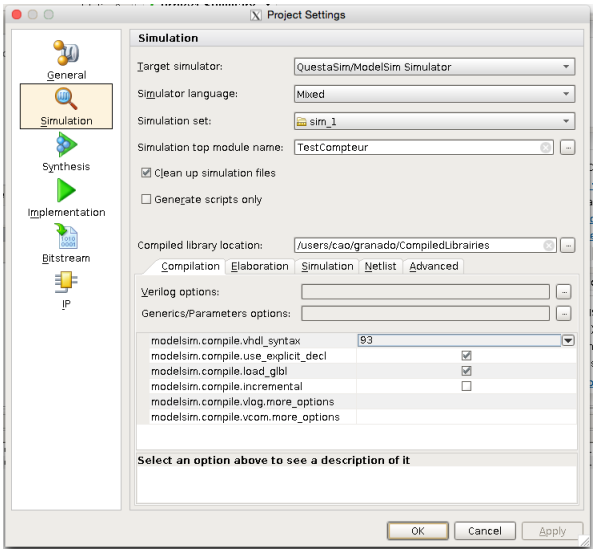
      i. **Q3:** Combien de process sont utilisés ? Quel est le rôle de chaque process ?

      ii. **Q4:** Quelle est la clause VHDL utilisée pour mettre des points d'arrêt dans les process ?

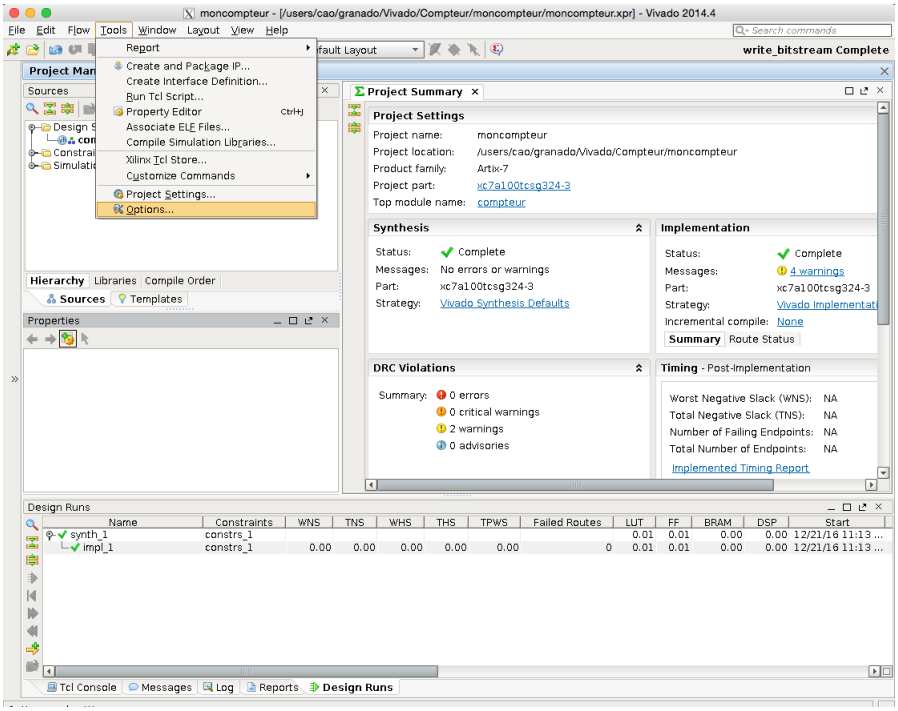
7. Pour réaliser la simulation, nous allons utiliser les simulateur `Modelsim`. Allez dans le menu `Flow` et ouvrez la fenêtre `Simulation Settings...`



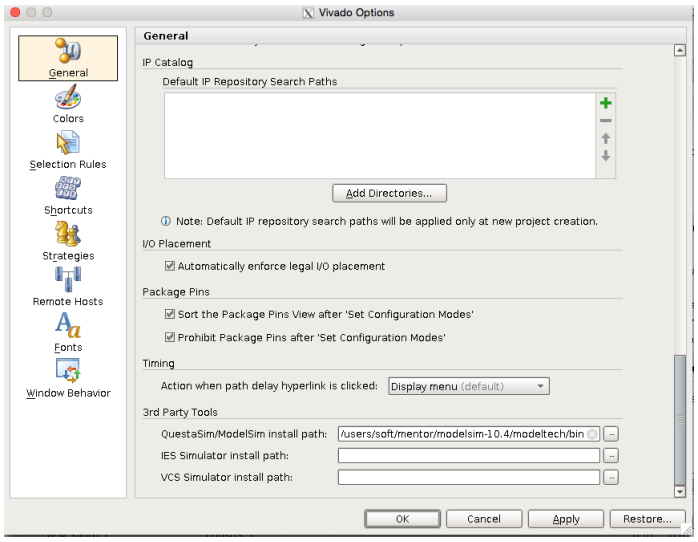
Vérifiez que le simulateur choisi (`Target Simulator`) est bien `QuestaSim/ModelSim Simulator` et que le chemin pour `Compiled library location` est égal à `I:/bgranado/2E200`



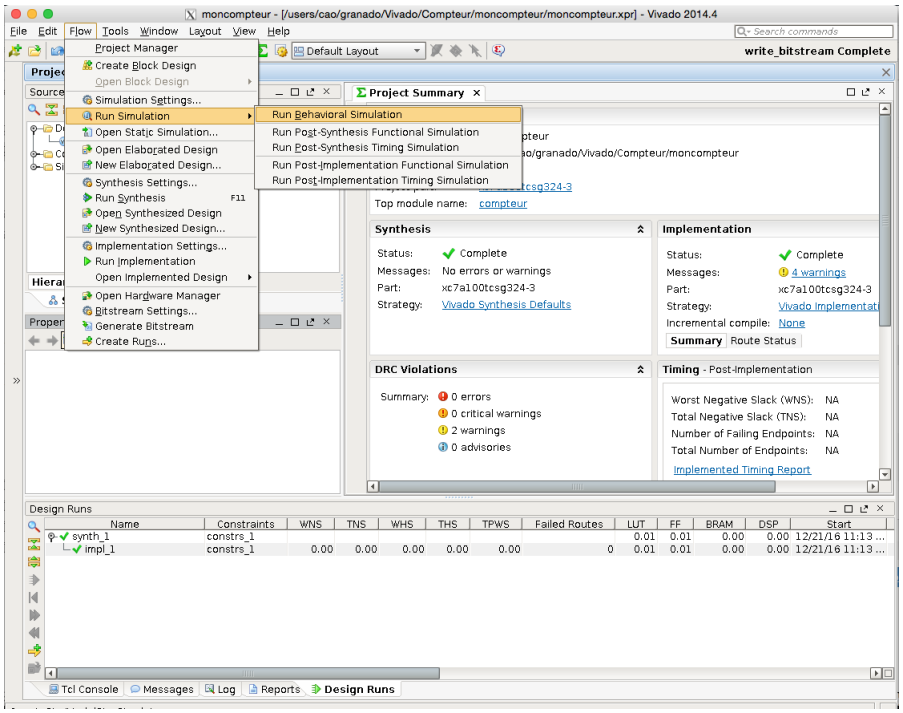
Ensuite allez dans le menu `Tools` et ouvrez la fenêtre `Options`.



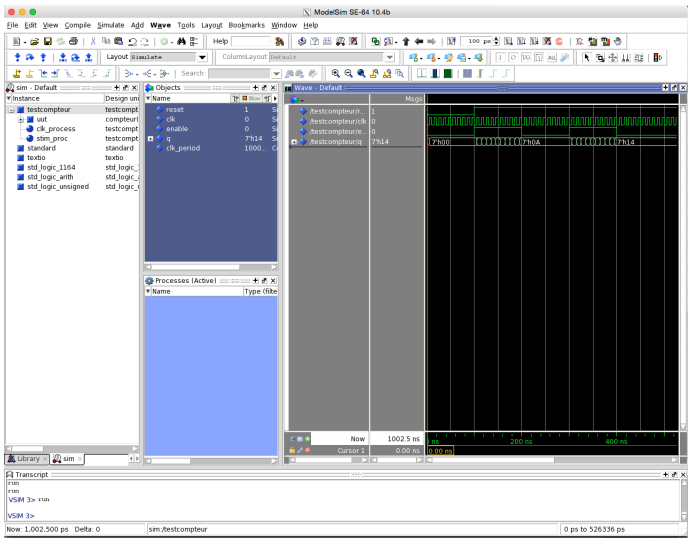
Vérifiez que le chemin pour accèder à Modelsim est bien renseigné, ce chemin est défini dans `QuestaSim/ModelSim install path` il doit être positionné à `c:/modeltech_6.6f/win32`.



8. Aller dans le menu `Flow` et cliquez sur `Run Simulation` puis sur `Run Behavioral Simulation`.



Une fenêtre de simulation provenant de l'outil `ModelSim` s'ouvre.



9. Vérifiez que dans la fenêtre `Wave` l'ensemble de signaux "`reset, clk, enable, q`" apparaissent. Si cela n'est pas le cas, sélectionnez les signaux manquants dans la fenêtre `Objects` et en cliquant sur le bouton droit de la souris ajouter les dans la fenêtre `Wave`.

10. en vous aidant des zoom 

    (a) Vérifiez que le compteur compte bien de 1 en 1 à partir de 0.

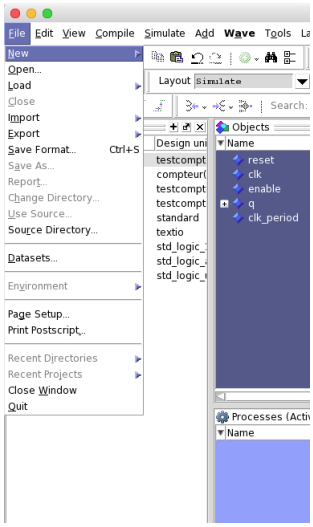
    (b) **Q5 :** Indiquez l'instant en seconde auquel le compteur commence à compter.

    (c) **Q6 :** Indiquez les instants en seconde auxquels le compteur arrête de compter et indiquez la valeur du compteur pour chaque instant.

    (d) **Q7 :** Déduisez le rôle du signal `enable`.

11. Il est possible de relancer une simulation en cliquant sur le bouton  ou bien de faire un pas de simulation en cliquant sur le bouton  , la durée du pas est indiquée dans le champs  , ce champs est modifiable.

12. Fermez le simulateur à l'aide du menu `File` et l'item `Quit`


File
Edit
View
Compile
Simulate
Add
Wave
Tools
New
Open...
Load
Close
Import
Export
Save Format...
Ctrl+S
Save As...
Report...
Change Directory...
Use Source...
Source Directory...
Datasets...
Environment
Page Setup...
Print Postscript...
Recent Directories
Recent Projects
Close Window
Quit
Layout
Simulate
Search:
Objects
Name
reset
clk
enable
q
clk_period
Processes
Name

1.3 Outils pour le VHDL : Une première synthèse : le compteur

Nous venons de simuler le compteur décrit dans le fichier `compteur.vhd`, nous allons maintenant synthétiser ce compteur et télécharger sur la carte FPGA `Nexys4` (ou `Nexys4DDR`) le résultat de cette synthèse. Ce résultat est un fichier binaire, appelé `bitstream`, qui contient l'ensemble des informations nécessaires pour configurer le FPGA afin que ce dernier réalise la fonctionnalité décrite dans le fichier `compteur.vhd`.

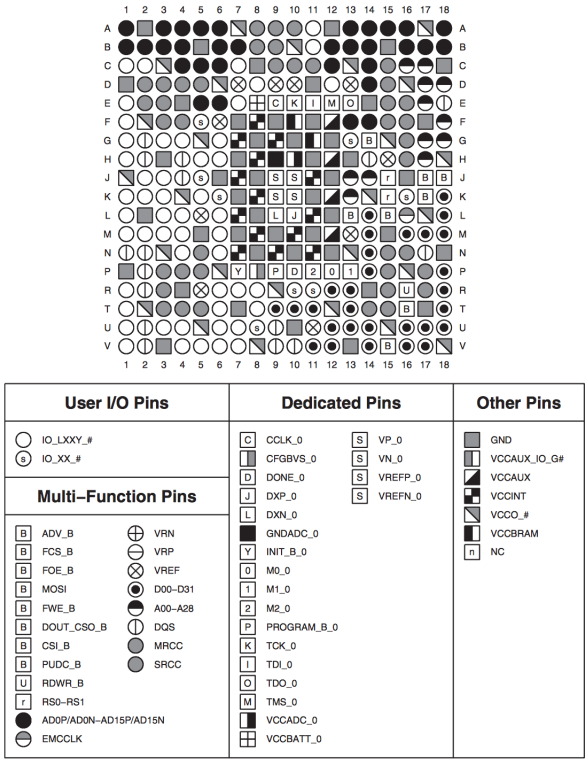
La synthèse logique que nous allons effectuer, consiste à transformer une description VHDL pouvant contenir des constructions de haut niveau, comme des `process` par exemple, en une description de plus bas-niveau proche des équations Booléennes.

1. Avant de réaliser la synthèse, il est nécessaire de connecter les entrées-sorties du compteur aux entrées-sorties de la carte `Nexys4` (ou `Nexys4DDR`), ce que nous allons réaliser ici.

   - Pour l'entrée `horloge` du compteur nous utiliserons un bouton poussoir, le bouton BTNC de la carte `Nexys4` (ou `Nexys4DDR`).
   - Pour l'entrée `reset` du compteur nous utiliserons l'interrupteur SW0 de la carte `Nexys4` (ou `Nexys4DDR`).
   - Pour l'entrée `enable` du compteur nous utiliserons l'interrupteur SW1 de la carte `Nexys4` (ou `Nexys4DDR`).
   - Pour la sortie `q` du compteur nous utiliserons les leds LD0 à LD6 de la carte `Nexys4` (ou `Nexys4DDR`).

   Ces différentes entrées-sorties sont connectées aux broches du FPGA `Artix7` de la carte `Nexys4` (ou `Nexys4DDR`). Le brochage de ce FPGA contient 324 broches organisées matriciellement, certaines de ces broches ont une utilisation spécifique (alimentation VCC et GND ou Horloge notamment), d'autres sont génériques.

   La matrice contient 18 lignes indentifiées par les lettres de A à V et 18 colonnes numérotées de 1 à 18.

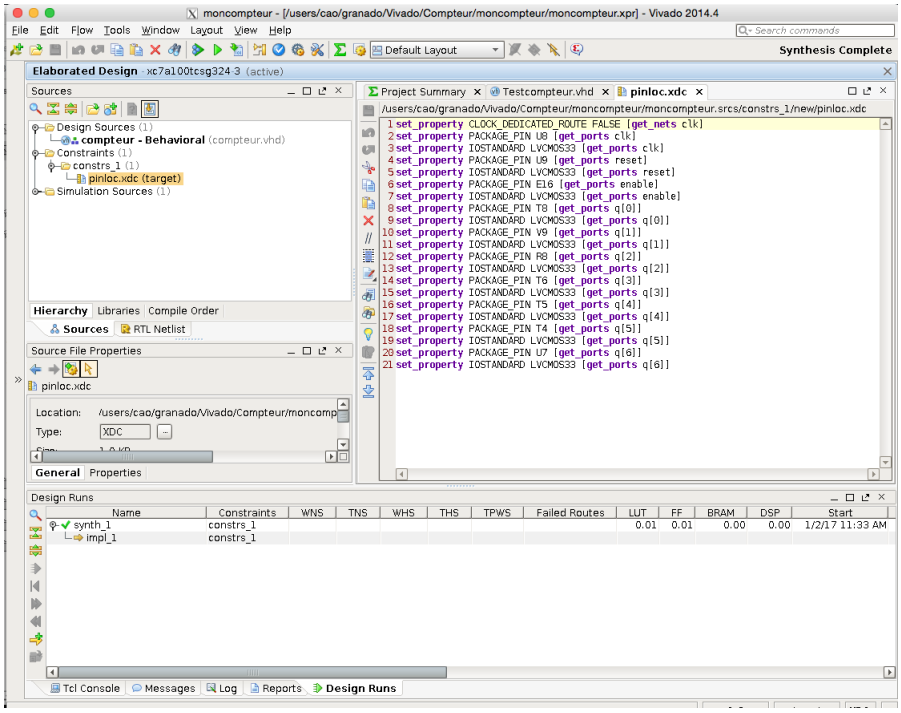


Pour associer les entrées sorties du compteur aux broches du FPGA, il est nécessaire d'avoir un fichier qui contient les informations d'interface.

Ce fichier a pour extension, l'extension `.xdc`. Double-cliquez sur le fichier `pinloc.xdc` dans le répertoire `Constraints/constrs_1`, ce fichier contient les informations d'interface entre les entrées-sorties de la carte `Nexys4` (ou `Nexys4DDR`) et les FPGA `Artix7`. Une fenêtre similaire à celle qui suit doit s'être ouverte.

Dans ce fichier il y a deux types d'informations:

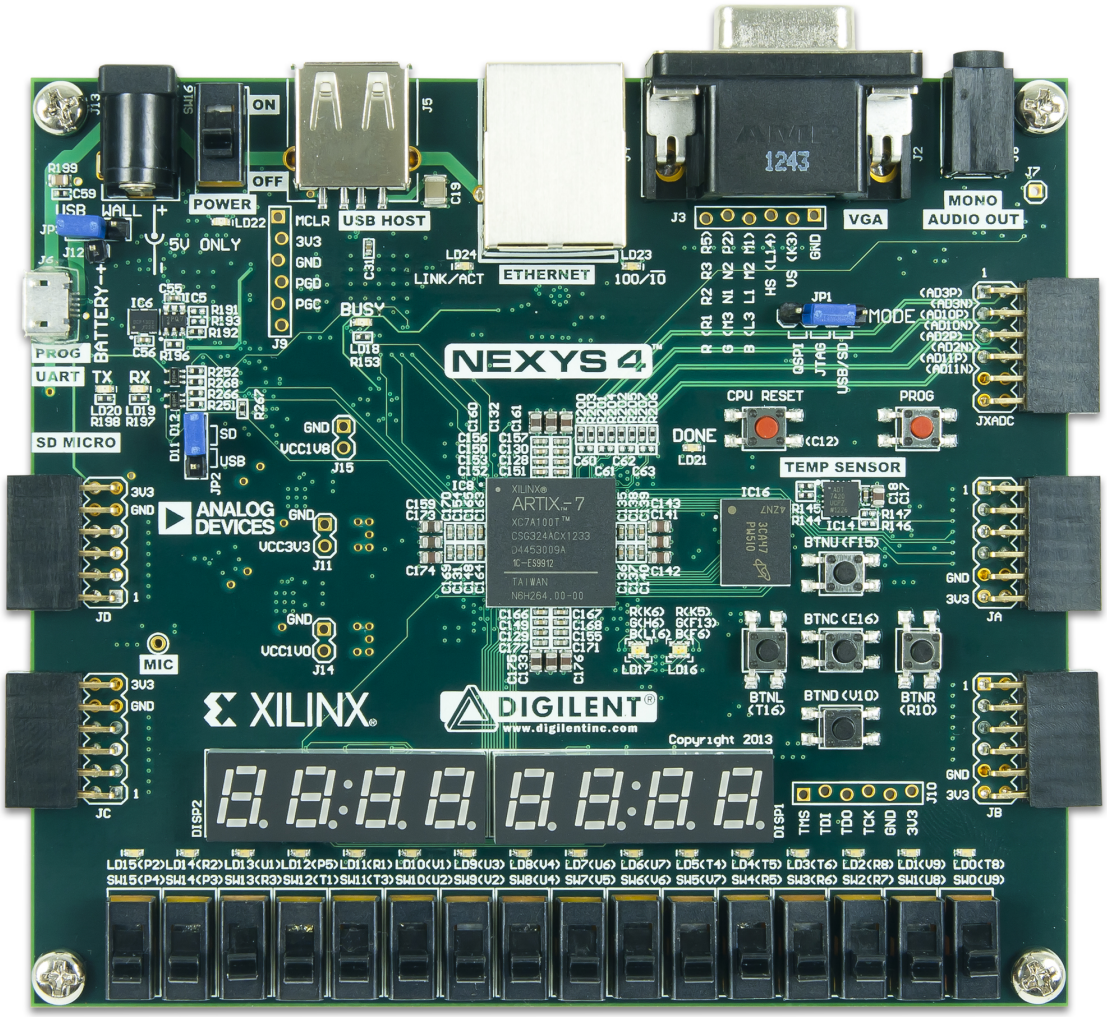
- La localisation des entrées-sorties à l'aide de la commande `set_property PACKAGE_PIN`
- La définition du type électrique des entrées-sorties à l'aide de la commande `set_property IOSTANDARD`. Dans le cadre de l'UE2E200 les entrées-sorties ont tous le même type électrique le LVCMOS33.



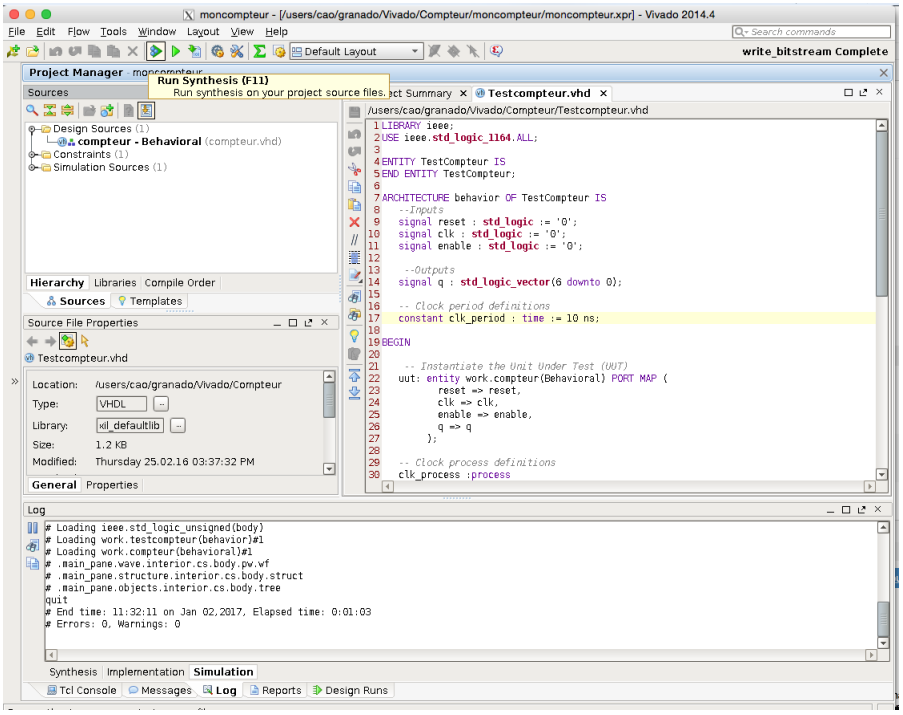
(a) **Q8 :** Identifiez les coordonnées des LED utilisées pour afficher la sortie du compteur

(b) **Q9 :** Identifiez les coordonnées des interrupteurs utilisés pour les signaux `RESET` et `ENABLE`

(c) **Q10 :** Vérifiez sur la carte (image ci-après) que les localisations correspondent bien aux souhaits énoncés plus haut.

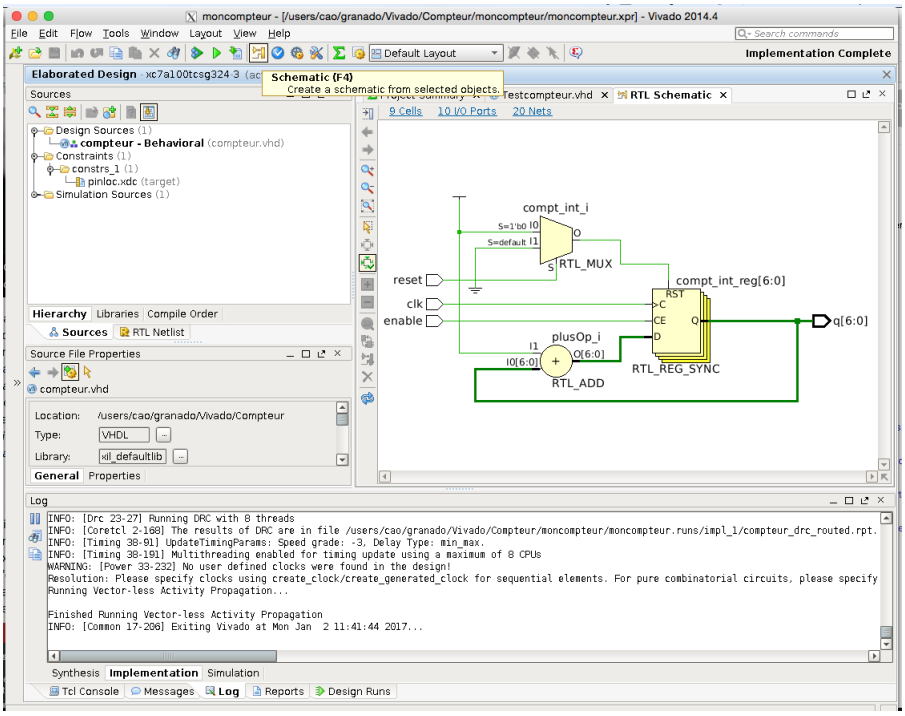


2. Pour réaliser la synthèse, il suffit de cliquer sur l'icône représentant un triangle vert contenant 3 boules, comme on le voit ci-dessous



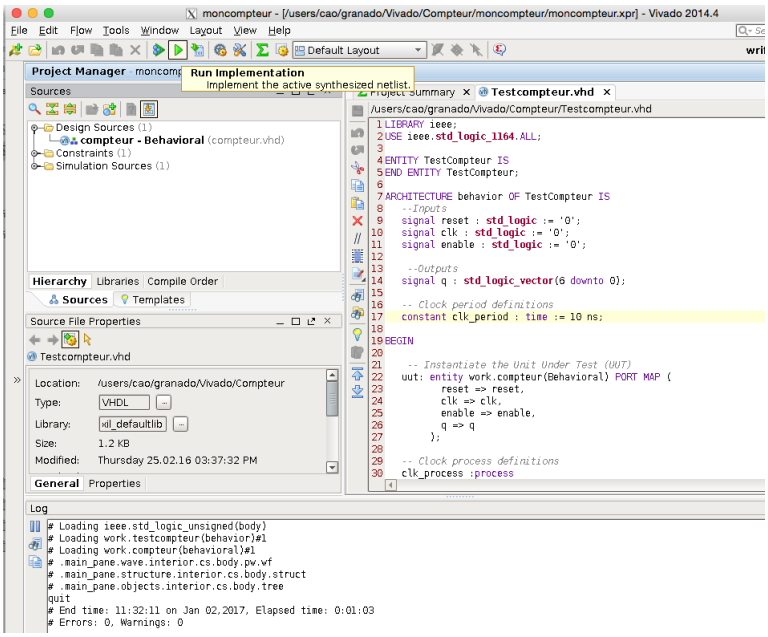
A la fin de la synthèse, une fenêtre s'ouvre cliquez sur `Cancel` pour la fermer.

3. Afin d'avoir une vue du résultat de synthèse, cliquez sur le bouton représentant un inverseur et une bascule tous les deux en jaune. Si le bouton n'apparaît pas, allez dans `Flow Navigator` puis dans `Synthesis` et dans `Synthesized Design` et là cliquez sur le bouton qui a du apparaître.



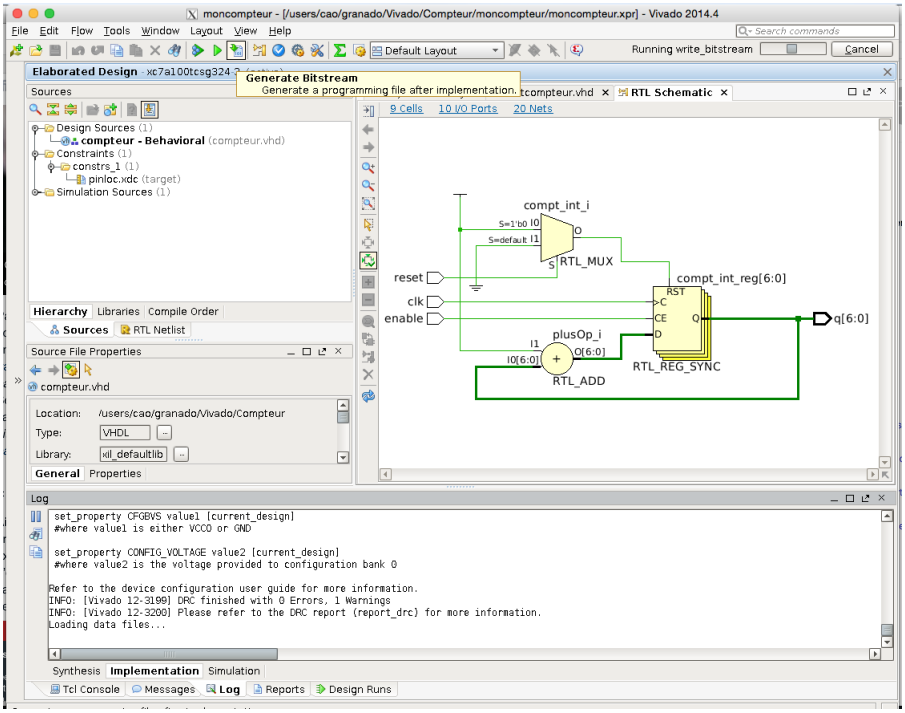
Le schéma que vous voyiez correspond à la transformation du modèle VHDL via la synthèse.

4. Une fois la synthèse effectuée, il faut faire l'implémentation de cette synthèse sur le composant. La synthèse a créé une architecture générique, l'implémentation va la rendre spécifique au composant, ici le composant `Artix7`. Pour cela cliquez sur l'icône représentant un triangle vert.



A la fin de l'implémentation, une fenêtre s'ouvre cliquez sur `Cancel` pour la fermer.

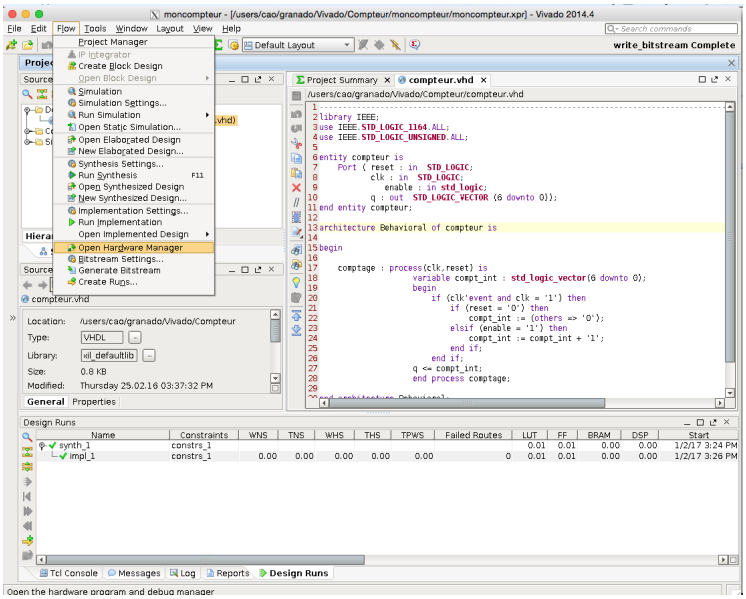
5. Nous sommes prêt pour générer le fichier de configuration en binaire, ou `bitstream`. Il suffit pour cela de cliquez sur l'icône représentant un fichier binaire.



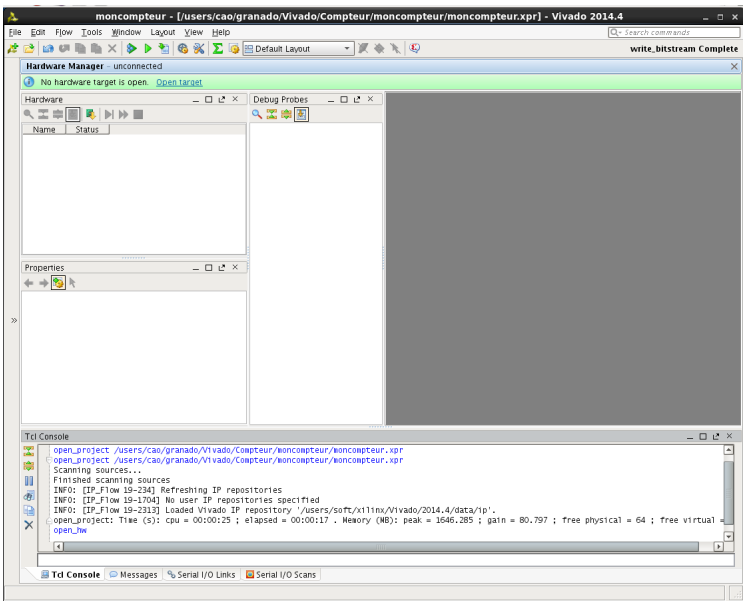
A la fin de la génération, une fenêtre s'ouvre cliquez sur `Cancel` pour la fermer.

6. Le fichier de configuration étant prêt, nous allons pouvoir configurer le FPGA de la carte. Avant toute chose, vérifier que votre carte `Nexys4` (ou `Nexys4DDR`) est alimentée, pour cela il suffit de connecter le câble USB de programmation à l'ordinateur et à la carte.

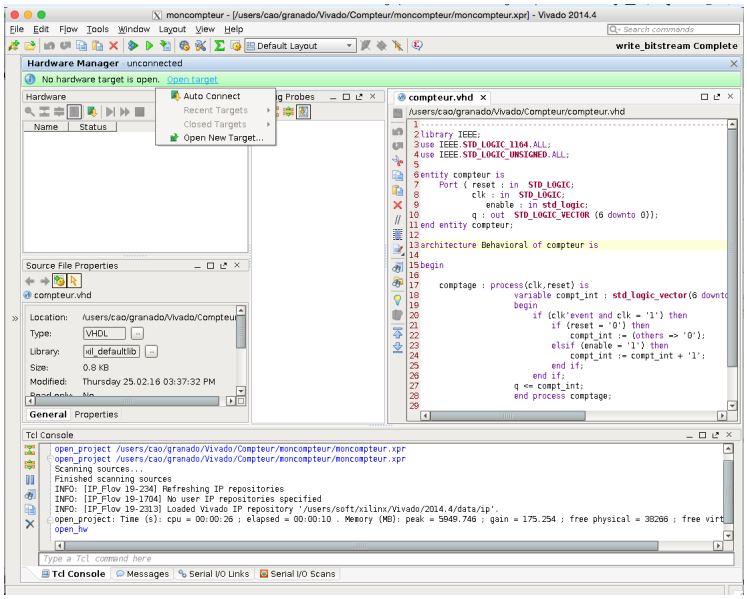
7. Ouvrez le `Hardware Manager` dans le menu `Flow`



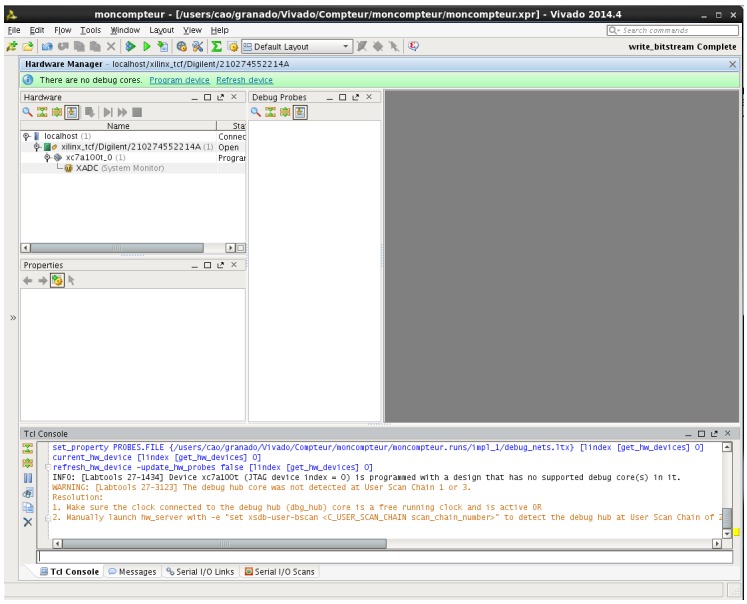
8. Cliquez dans le `Hardware Manager` sur `open target`



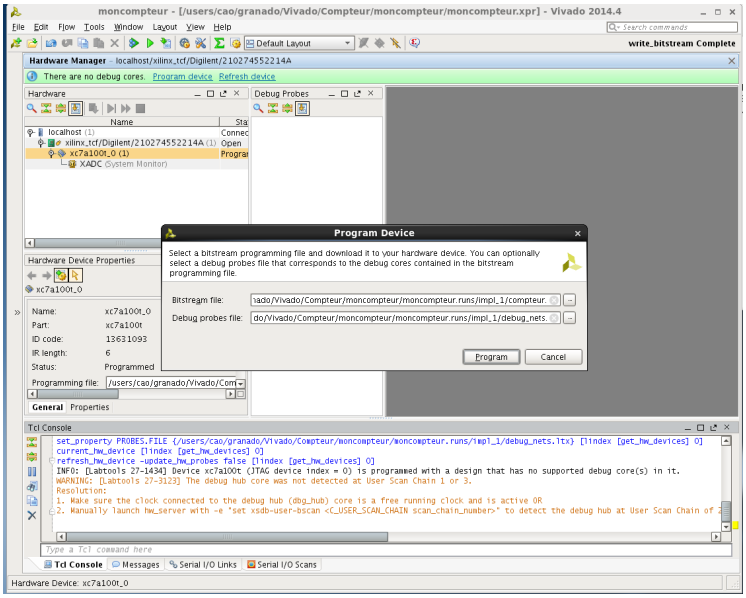
9. Cliquez sur `Auto Connect` ou sur une la ligne débutant par `localhost:60001.....`



vous obtenez une configuration similaire à l'image suivante



10. Cliquez sur `Program device` , choisissez `Device1 (FPGA xc7a100t)`
11. Une fenêtre s'ouvre, cliquez sur `Program`



12. Vous pouvez maintenant testez sur la carte, et vérifiez le bon comportement du compteur.

2 Séance 2 : VGA

2.1 IP VGA

Vous devez rendre à la fin de la séance un compte-rendu rédigé qui indique notamment les réponses aux différentes questions, vous devez y mettre vos sources VHDL commentées.

Dans cette séance, vous allez travailler avec une IP [1] matérielle qui vous est fournie permettant de réaliser un affichage sur l'écran VGA une fenêtre de 480 par 640 pixels.

Une IP est, dans ce cas, un code VHDL réalisant la fonction désirée. Les paramètres à connaître de cette IP sont décrits dans la section suivante.

L'entité donnée en figure 1 représente l'interface de l'IP VGA. Cette IP fonctionne avec une horloge de 25 MHz, l'horloge de la carte étant de 100 MHz, il est nécessaire alors de diviser par 4 sa fréquence. Pour cela une IP de division vous ait fournie, la description de son interface est donnée dans la figure 2.

Dans l'IP Division, il y a deux entrée :

- une horloge qui doit être de 100 MHz, cette entrée s'appelle `clk100`;
- une entrée de remise à zéro, cette entrée est asynchrone, active à 0 et s'appelle `reset`.

il y a une sortie :

[1] IP signifie ici Intellectual Property et représente un composant décrit sous un format électronique permettant son implantation matérielle soit dans un FPGA soit dans un circuit silicium

```
entity VGA is
port (
 clk25,reset: in std_logic;                          -- Horloge, Reset Asynchrone
 r,g,b: in std_logic;                                -- Couleur Envoyee par le Controleur de Jeu
 red,green,blue: out std_logic;                      -- Affichage Couleur vers Ecran VGA
 hsync,vsync: out std_logic;                         -- Synchro Ligne (H) et Trame (V)
 visible: out std_logic;                             -- Partie Visible de l'Image
 endframe: out std_logic;                            -- Dernier Pixel Visible d'une Trame
 xpos,ypos: out std_logic_vector(9 downto 0)         -- Coordonnees du Pixel Courant
);
end VGA;
```

Figure 1: IP VGA

```
entity ClkDiv is
Port ( clk100,reset : in  STD_LOGIC;    -- Horloge 100 MHz et Reset Asynchrone
 clk25 : out  STD_LOGIC                 -- Horloge 25 MHz
);
end ClkDiv;
```

Figure 2: IP Division de fréquence

- une sortie `clk25` qui est une horloge fonctionnant à 25 MHz si l'entrée `clk100` a été connectée à une horloge de 100MHz.

Muni de cette IP de division vous pouvez maintenant utiliser l'IP VGA, dans cette IP il y a 5 entrées :

- une entrée d'horloge, surlaquelle doit être connectée une horloge à 25 MHz, cette entrée s'appelle `clk25`;
- une entrée asynchrone de remise à zéro, active à 0, qui s'appelle `reset`;
- trois entrées `r`, `g` et `b` qui permettent d'indiquer quelle couleur on désire afficher sur l'écran VGA. Les couleurs possibles sont indiquées dans le tableau suivant

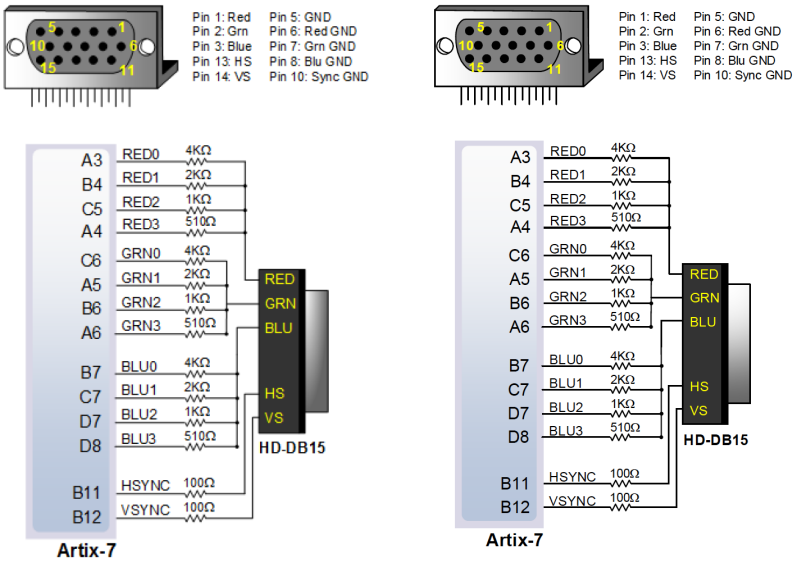
r	g	b	couleur
0	0	0	écran éteint
0	0	1	bleu
0	1	0	vert
0	1	1	cyan
1	0	0	rouge
1	0	1	mangeta
1	1	0	jaune
1	1	1	blanc

L'IP VGA dispose aussi de 9 sorties, qui sont :

- trois sorties `red`, `green` et `blue` qui une fois connectée à la prise VGA indique à l'écran quelle couleur afficher;
- deux sorties `hsync` et `vsync` permettant d'assurer la synchronisation horizontale et verticale de l'écran;
- une sortie `visible` qui indique si le pixel courant est dans la zone d'affichage de l'écran ou non. Si `visible` est à 1 alors le pixel est dans la zone visible, sinon il est en dehors. Il faut absolument que les sorties `red`, `green` et `blue` soient à zéro lorsque le pixel est en dehors de la zone visible;
- une sortie `endframe` qui indique si le pixel est le dernier de la fenêtre courante
- deux sorties `xpos` et `ypos` qui indique les coordonnnées du pixel courant

2.2 Gestion du VGA sur la carte `Nexys4` (`Nexys4DDR`)

Sur la carte Nexys4 il est possible de gérer jusqu'à 4096 couleurs puisque chaque canal couleur est codé sur 4 bits, comme on peut le voir sur la figure suivante :



Nexys4 Nexys4DDR

Nous n'utiliserons que 8 couleurs, et vous veillerez à ce que les bits 0 à 2 de chaque couleur soient à 0 et que les bits 3 de chaque couleur soient connectés aux sorties correspondantes de l'IP VGA.

2.3 Exercice A : Affichage 1 couleur parmi 8

A l'aide de la documentation de l'IP VGA, écrivez un code VHDL utilisant l'IP `VGA` et l'IP `ClkDiv` permettant d'afficher une couleur parmi les 8 possibles sur l'écran.

Pour cet exercice vous écrirez une entité nommée `unecouleur` qui aura les entrées suivantes

- une entrée `clk100` qui devra être connecté à une horloge de 100 MHz;
- une entrée `reset` permettant de faire la remise à zéro du système, cette entrée sera asynchrone et active à la valeur 0, elle sera reliée à un bouton poussoir de la carte;
- trois entrées `bouton1`, `bouton2` et `bouton3` qui seront connectée aux 3 premiers interrupteurs de la carte et qui permettrons de choisir une couleur parmi 8;
- trois sorties `red`, `blue` et `green` permettant d'indiquer à l'écran quelle couleur afficher, elle seront reliées à la prise VGA aux entrées éponymes;
- deux sorties `hsync` et `vsync` permettant de transmettre à l'écran VGA la synchronisation horizontale et la synchronisation verticale.

Utiliser le simulateur MODELSIM pour vérifier votre IP.
Une fois la simulation correcte testez votre IP sur la carte `Nexys4`.

2.4 Exercice B : Affichage Damier

A l'aide de la documentation de l'IP VGA, écrivez un code VHDL utilisant l'IP `VGA` et l'IP `ClkDiv` permettant d'afficher une couleur dans 4 parties de l'écran. Pour cela vous diviserez en 4 l'écran, et vous y afficherez 4 couleurs différentes parmi les 8 possibles. Pour choisir les couleurs vous utiliserez 3 interrupteurs. Votre écran aura quatre parties qui seront :

- une partie haut-gauche;
- une partie haut-droit;
- une partie bas-gauche;
- une partie bas-droit.

Pour cet exercice vous écrirez une entité nommée `damier` qui aura les entrées suivantes

- une entrée `clk100` qui devra être connecté à une horloge de 100 MHz;
- une entrée `reset` permettant de faire la remise à zéro du système, cette entrée sera asynchrone et active à la valeur 0, elle sera reliée à un bouton poussoir de la carte;
- trois entrées `bouton1`, `bouton2` et `bouton3` qui seront connectée aux 3 premiers interrupteurs de la carte et qui permettrons de choisir une couleur parmi 8;
- trois sorties `red`, `blue` et `green` permettant d'indiquer à l'écran quelle couleur afficher, elle seront reliées à la prise VGA aux entrées éponymes;

- deux sorties `hsync` et `vsync` permettant de transmettre à l'écran VGA la synchronisation horizontale et la synchronisation verticale.

Utiliser le simulateur MODELSIM pour vérifier votre IP.
Une fois la simulation correcte testez votre IP sur la carte `Nexys4`.

2.5 Exercice C : Affichage d'une Brique

A l'aide de la documentation de l'IP VGA, écrivez un code VHDL utilisant l'IP `VGA` et l'IP `ClkDiv` permettant d'afficher une brique de taille 100 par 100 pixel d'une couleur parmi les 8 possibles.

Pour cet exercice vous écrirez une entité nommée `brique` qui aura les entrées suivantes

- une entrée `clk100` qui devra être connecté à une horloge de 100 MHz;
- une entrée `reset` permettant de faire la remise à zéro du système, cette entrée sera asynchrone et active à la valeur 0, elle sera reliée à un bouton poussoir de la carte;
- trois entrées `bouton1`, `bouton2` et `bouton3` qui seront connectée aux 3 premiers interrupteurs de la carte et qui permettrons de choisir une couleur parmi 8;
- trois sorties `red`, `blue` et `green` permettant d'indiquer à l'écran quelle couleur afficher, elle seront reliées à la prise VGA aux entrées éponymes;
- deux sorties `hsync` et `vsync` permettant de transmettre à l'écran VGA la synchronisation horizontale et la synchronisation verticale.

Utiliser le simulateur MODELSIM pour vérifier votre IP.
Une fois la simulation correcte testez votre IP sur la carte `Nexys4`.

2.6 Exercice D : Affichage de plusieurs Briques

A l'aide de la documentation de l'IP VGA, écrivez un code VHDL utilisant l'IP `VGA` et l'IP `ClkDiv` permettant d'afficher trois briques de taille 50 par 50 pixels sur une même ligne.

Vous créerez trois lignes différentes, toutes les briques d'une même ligne auront la même couleur et vous utiliserez 3 couleurs différentes pour chaque ligne.

Pour cet exercice vous écrirez une entité nommée `briqueligne` qui aura les entrées suivantes

- une entrée `clk100` qui devra être connecté à une horloge de 100 MHz;
- une entrée `reset` permettant de faire la remise à zéro du système, cette entrée sera asynchrone et active à la valeur 0, elle sera reliée à un bouton poussoir de la carte;

- trois entrées `bouton1`, `bouton2` et `bouton3` qui seront connectée aux 3 premiers interrupteurs de la carte et qui permettrons de choisir une couleur parmi 8;
- trois sorties `red`, `blue` et `green` permettant d'indiquer à l'écran quelle couleur afficher, elle seront reliées à la prise VGA aux entrées éponymes;
- deux sorties `hsync` et `vsync` permettant de transmettre à l'écran VGA la synchronisation horizontale et la synchronisation verticale.

Utiliser le simulateur MODELSIM pour vérifier votre IP.
Une fois la simulation correcte testez votre IP sur la carte `Nexys4`.

3 Documentation Outils de Développement

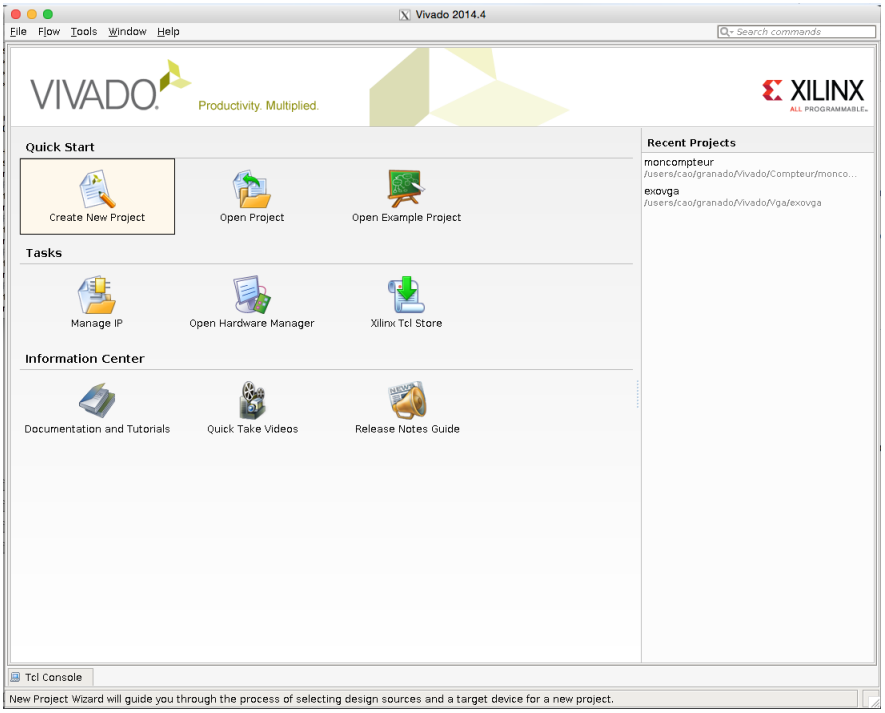
4 Utilisation du logiciel Vivado

Nous allons décrire ici les étapes complémentaires à celles déjà vues lors de la séance 1 sur le compteur pour réaliser votre propre projet VHDL.

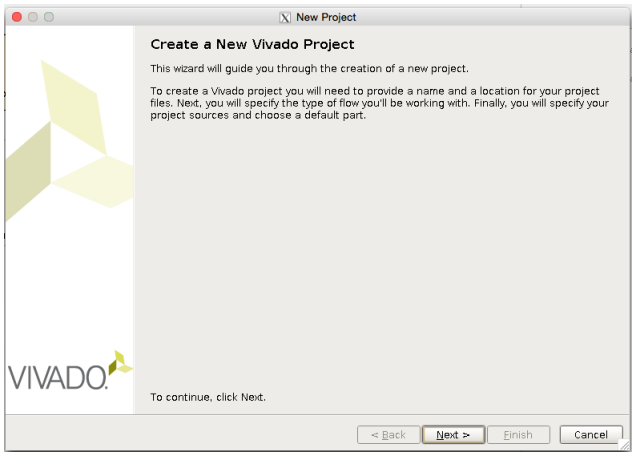
4.1 Création d'un nouveau projet

Lancer Vivado à partir du menu `Démarrer / Tous les Programmes / Xilinx Design Tools / Vivado 2014.2 / Vivado2014.2`

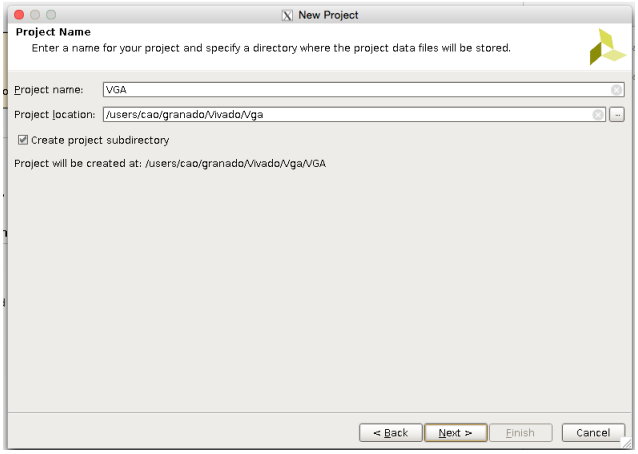
Créez un nouveau projet en cliquant sur l'icône `Create New Project`



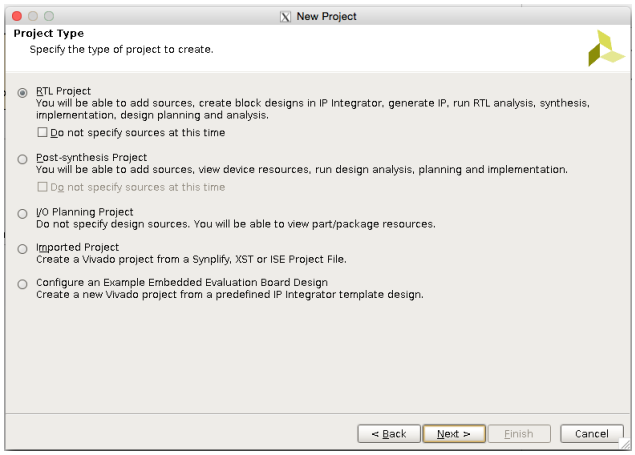
A la fenêtre suivante cliquez sur `Next`



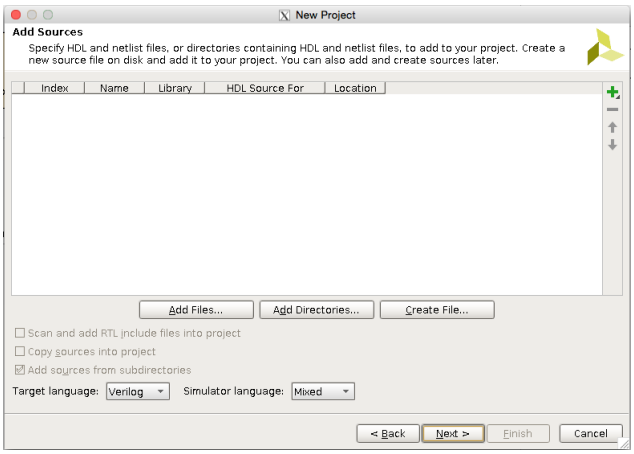
Dans la fenêtre suivante indiquez le nom de votre projet, par exemple `VGA` et la localisation de ce projet (il doit être dans votre répertoire H)



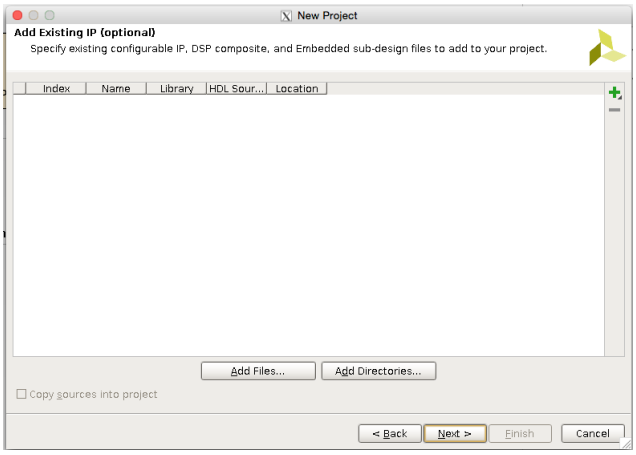
Dans la fenêtre suivante, assurez-vous que `RTL Project` est bien coché.



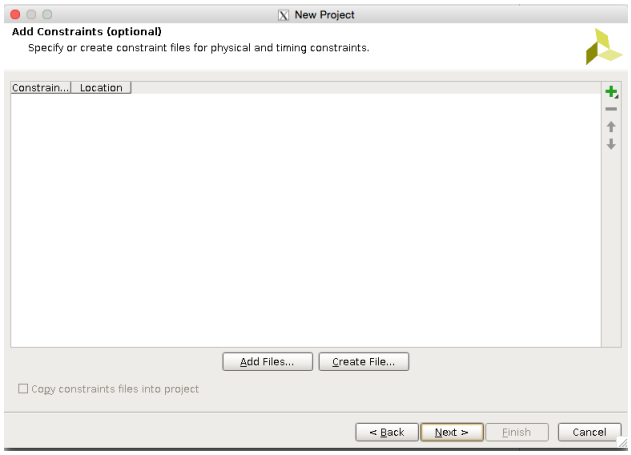
Dans la fenêtre suivante (`Add Sources`) cliquez sur `Next`, nous ajouterons les sources après.



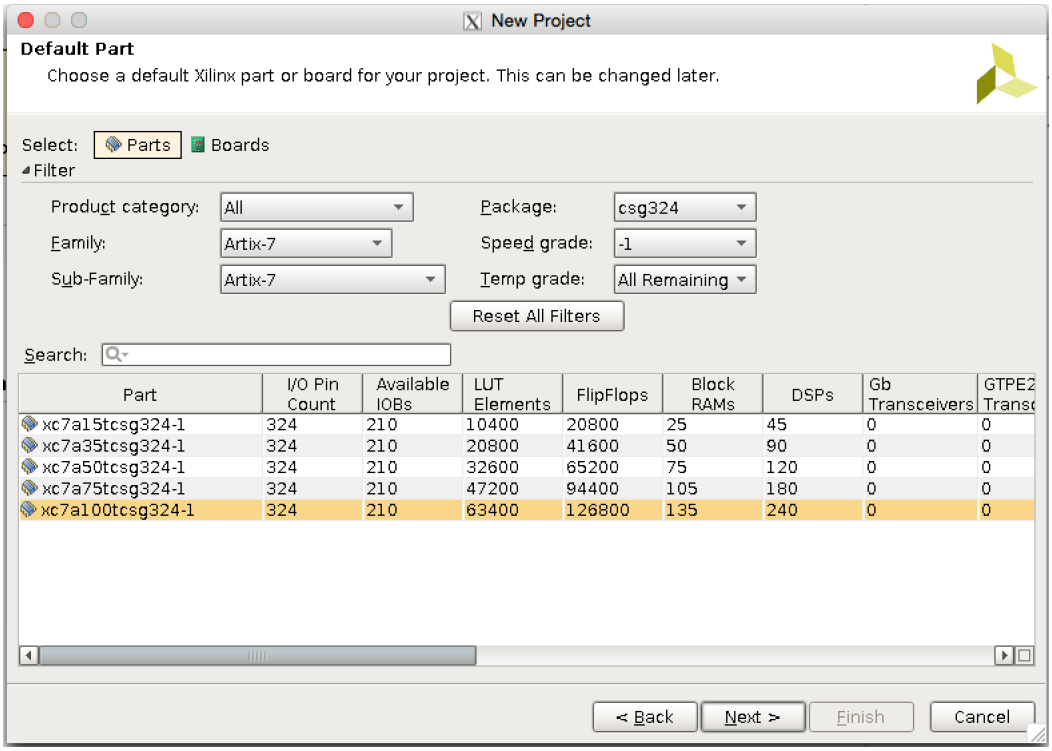
Dans la fenêtre suivante (`Add Existing IP`) cliquez sur `Next`



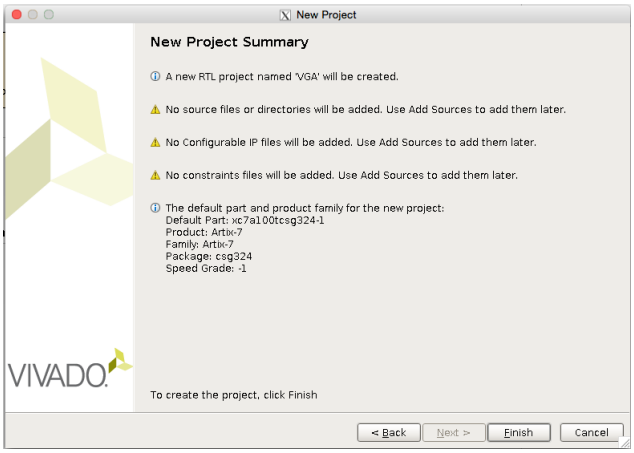
Dans la fenêtre suivante (`Add Constraints`) cliquez sur `Next`, nous ajouterons les contraintes après



Dans la fenêtre suivante (`Default Part`) assurez-vous de bien sélectionner le FPGA `xc7a100tcsg324-1`

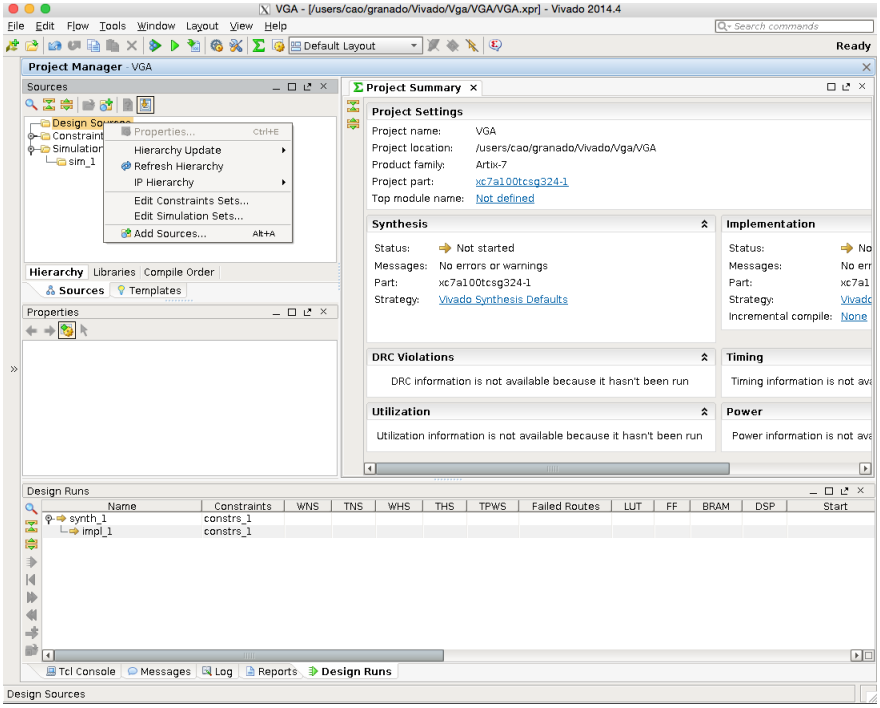


La fenêtre qui s'ouvre résume vos choix, cliquez sur `Finish`

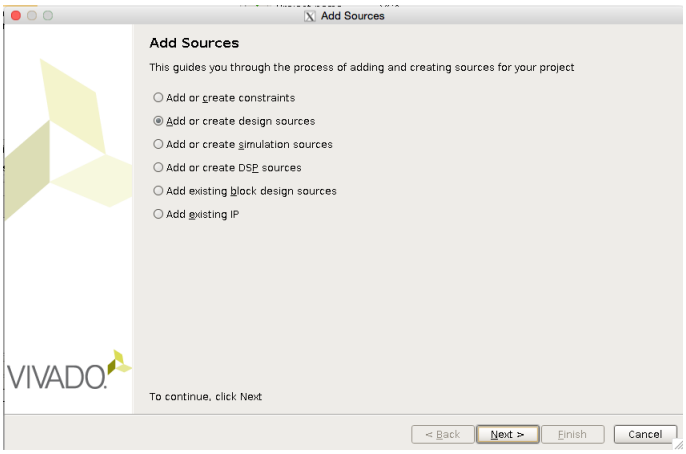


4.2 Ajout d'un fichier source

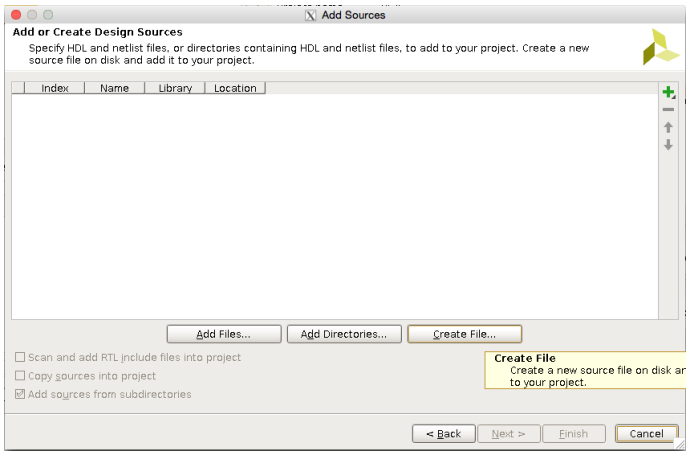
Cliquez sur le bouton droit de la souris dans la fenêtre `Sources` et choisissez `Add Sources`.



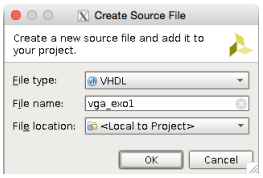
Dans la fenêtre qui s'ouvre (`Add Sources`) assurez-vous que `Add or create design sources` est bien coché.



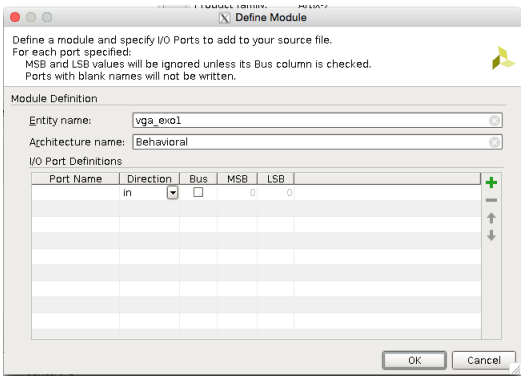
Dans la fenêtre suivante cliquer sur `Create File`



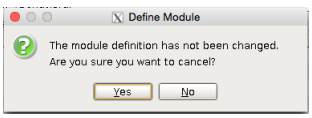
Dans la fenêtre suivante, indiquez le nom de votre fichier, par exemple `vga_exo1` et sélectionnez son type (ici VHDL), cliquez sur `OK`.



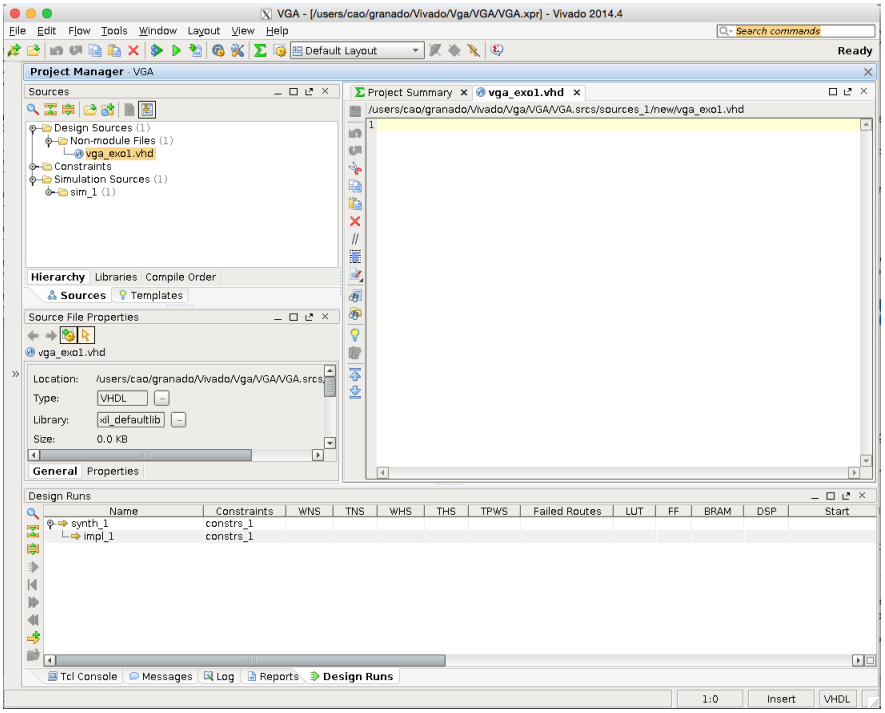
Dans la fenêtre suivante (`Define Module`) cliquez sur `Cancel`.



Dans la fenêtre suivante cliquez `Yes`

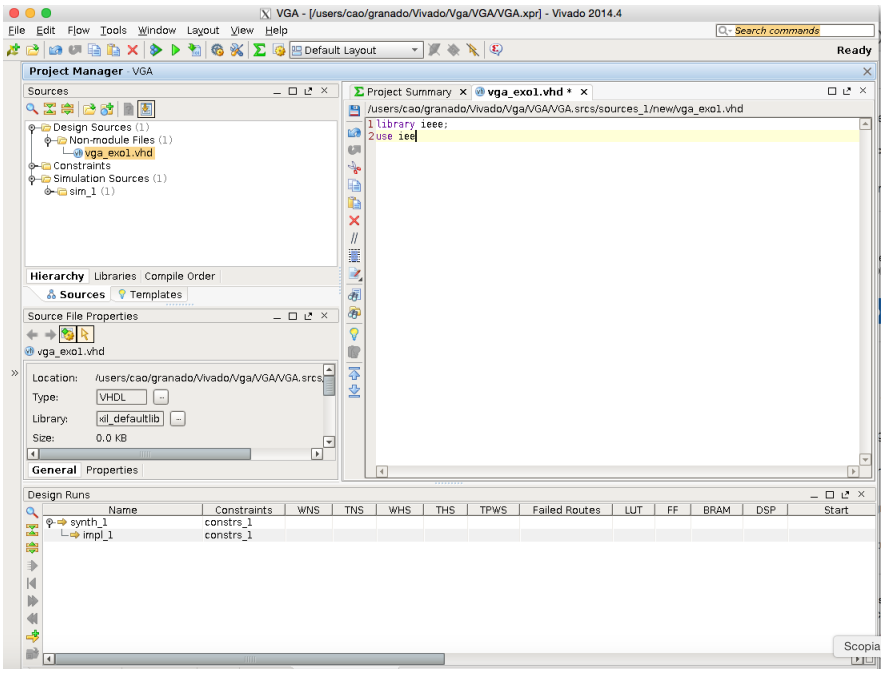


Le fichier apparaît maintenant sous le répertoire `Design Sources/Non-module files`

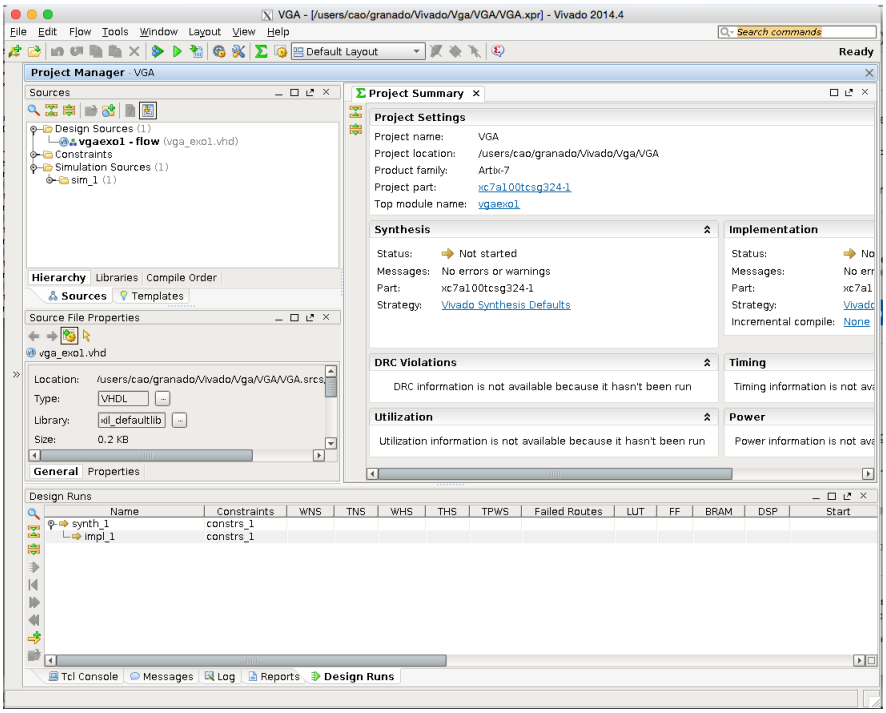


Ecrivez votre code dans le fichier en utilisant un des trois noms suivants pour l'architecture de votre composant :

- `Dataflow` pour une description flot de donnée
- `Struct` pour une description structurelle
- `Behavioral` pour une description comportementale

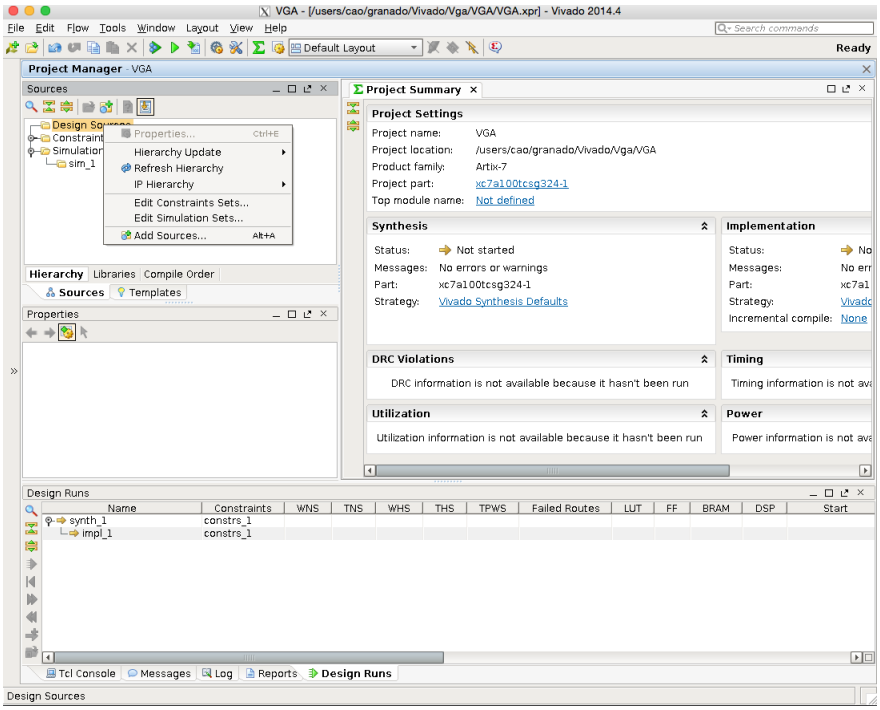


Une fois le code écrit et validez, le fichier devrait se trouver dans le répertoire `Design Sources`

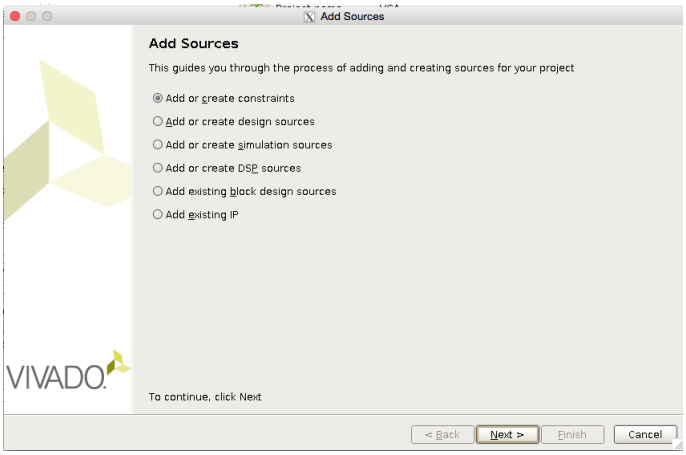


4.3 Ajout d'un fichier de contraintes

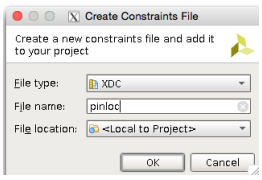
Cliquez sur le bouton droit de la souris dans la fenêtre `Sources` et choisissez `Add Sources`.



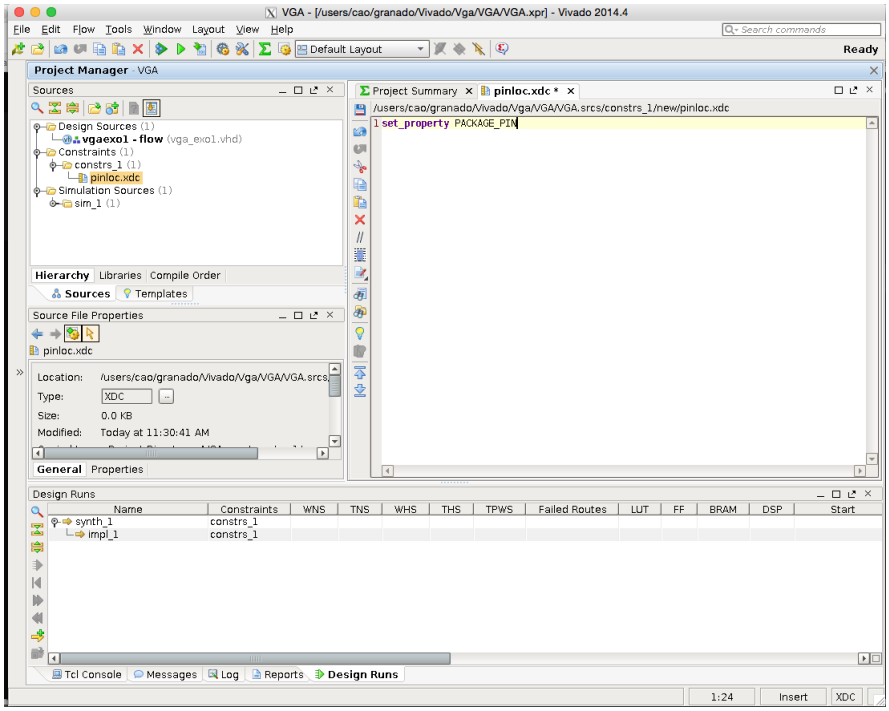
Dans la fenêtre suivante assurez-vous que `Add or create constraints` est bien coché



Dans la fenêtre suivante indiquez le nom de votre fichier de contraintes, par exemple `pinloc`, cliquez sur `OK`



Vous pouvez maintenant éditer votre fichier qui est dans le dossier `Constraints/constrs_1`



Pour la carte `Nexys4` les localisations des entrées-sorties utiles pour le VGA sont les suivantes :

- broche d'horloge clk = "E3";
- broche de reset= "C12";
- broche du port VGA red="A4";
- broche du port VGA green="A6";
- broche du port VGA blue="D8";
- broche du port VGA syncv="B12";
- broche du port VGA synch="B11";
- l'interrupteur SW0="U9";
- l'interrupteur SW1="U8";
- l'interrupteur SW2="R7";
- l'interrupteur SW3="R6";

Pour la carte `Nexys4DDR` les localisations des entrées-sorties utiles pour le VGA sont les suivantes :

- broche d'horloge clk = "E3";
- broche de reset= "C12";
- broche du port VGA red="A4";
- broche du port VGA green="A6";
- broche du port VGA blue="D8";
- broche du port VGA syncv="B12";
- broche du port VGA synch="B11";
- l'interrupteur SW0="J15";
- l'interrupteur SW1="L16";
- l'interrupteur SW2="M13";
- l'interrupteur SW3="R15";

5 Mini-Projet : Casse Brique

Le CasseBrique est un jeu inventé en 1976 par la société Atari et dont le nom original est `BreakOut`.

Le but de ce jeu, hérité du jeu `Pong` du même constructeur, est de casser des briques à l'aide d'une balle et d'une raquette. La raquette est un rectangle de taille 120x60 pixels, la balle est un carré de taille de taille 7x7 pixels et les briques des rectangles de taille 100x50.

Au départ la raquette est au centre en bas de l'écran, elle ne peut ensuite évoluer que de gauche à droite ou de droite à gauche en fonction d'une commande de mouvement qui lui est indiquée.

La balle descend à partir du milieu de l'écran en diagonale soit vers la droite, soit vers la gauche.

Le but du jeu est de faire rebondir la balle avec la raquette pour qu'elle heurte les briques. A chaque fois que la balle heurte une brique, la brique est détruite et la balle rebondie, soit vers le bas si elle a heurté la brique de bas en haut, soit vers le haut si elle a heurté la brique de haut en bas.

Le jeu est fini lorsque toutes les briques sont détruites ou que la balle n'a pas été rattrappée par la raquette alors qu'elle arrivait en bas de l'écran.

Il vous ait demandé dans ce mini-projet de réaliser un jeu de casse-brique à l'aide de la carte `Nexys4`, d'un écran VGA et d'un accéléromètre. Une documentation de la carte `Nexys4` vous ait fourni à la section **??**.

Pour réaliser ce jeu vous disposez du composant `VGA` que vous avez déjà utilisé et de 3 autres composants :

- Le composant `Objects` gère l'affichage des objets du jeu à savoir les briques, la balle, la raquette et le décor.
- Le composant `Game` gère le jeu, c'est à dire l'évolution des objets du jeu.
- Le composant `SPIcomponent` gère l'accéléromètre, il acquiert les données du capteur et fourni selon trois axes la valeur de l'accélération.

ces composants sont disponibles sur le site web du cours 2E200, vous trouverez leur description dans la section 6.

5.1 Diagramme SADT du Casse Brique

Avant de commencer la conception à l'aide du langage VHDL du Casse-Brique, vous allez décrire sous forme d'un diagramme SADT le Casse-Brique. Vous utiliserez les composants qui vous sont donnés et vous y ajouterez les composants que vous jugerez nécessaires pour les interconnecter.

1. Décrivez le niveau A-0 du Casse-Brique.

2. En utilisant les trois composants fournis, `Object`, `Game` et `SPIcomponent` décrivez le niveau A0 du Casse-Brique.

3. En utilisant les diagrammes SADT et la documentation des composants donnée à la section 6, réalisez un schéma complet et précis du jeu du Casse-Brique.

5.2 Accéléromètre

Pour jouer vous allez utiliser un accéléromètre qui mesure à la fois une accélération dynamique lors d'un mouvement, mais aussi une accélération statique du à la gravité terrestre et ceci suivant les trois axes `x`, `y` et `z`. Le principe de fonctionnement d'un accéléromètre est décrit dans la vidéo visible à l'adresse

`http://bertrand.granado.free.fr/LE201/LE201/Principe_dun_accelerometre.html`

Il vous ai donné un composant `SPIcomponent` qui gère la lecture des valeurs de l'accéléromètre. Vous devez ici utiliser ce composant afin de récupérer les valeurs en x, y et z fournies par l'accéléromètre.

1. Avant de réaliser la description VHDL permettant de lire les valeurs de l'accéléromètre, en vous aidant du schéma complet du jeu du Casse-Brique que vous avez déjà fait, dessinez un schéma permettant de visualiser l'interconnexion entre l'accéléromètre et le composant `SPIcomponent`. Vous déduirez de ce schéma les directions des différents signaux, `IN` ou `OUT`, de l'entité du composant VHDL que vous allez décrire.

2. Décrire en VHDL un composant qui instancie le composant `SPIcomponent` capable de lire les valeurs de l'accéléromètre. Attention a bien réfléchir au moment ou vous initiez un transfert des valeurs de trois axes.

3. Modifiez votre composant pour afficher sur les 8 leds de la carte `Nexys4` les 8 bits de poids forts de l'axe des `x`, en faisant évoluer de gauche à droite et de droite à gauche l'accéléromètre regarder l'évolution des leds.

4. Modifier votre composant pour afficher sur les 8 leds de la carte `Nexys4` les 8 bits de poids forts de l'axe des `y`, en faisant évoluer de gauche à droite et de droite à gauche l'accéléromètre regarder l'évolution des leds.

5. Modifier votre composant pour afficher sur les 8 leds de la carte `Nexys4` les 8 bits de poids forts de l'axe des `z`, en faisant évoluer de gauche à droite et de droite à gauche l'accéléromètre regarder l'évolution des leds.

6. A l'aide des observations que vous venez d'effectuer choisissez l'un des axes et indiquez la loi d'évolution des valeurs liées à cet axe lorsque l'on fait un mouvement de droite à gauche ou de gauche à droite.

7. Décrivez un composant VHDL capable d'indiquer le sens de déplacement de l'accéléromètre, gauche ou droit.

5.3 Mise en place du jeu

Vous disposez maintenant de l'ensemble des informations nécessaires à la réalisation du jeu du Casse-Brique.

1. Décrire en VHDL un composant qui réalise le jeu du Casse-Brique
2. Testez et validez sur la carte `Nexys4` votre jeu
3. En vous aidant de l'information fournie lors de la séance sur l'affichage VGA modifier progressivement votre composant pour y ajouter
   (a) l'affichage d'un message "Partie Terminée" lorsque la balle est perdue
   (b) l'affichage du score, c'est à dire le nombre de briques détruite
   (c) le passage à un niveau supérieur lorsque toutes les briques du niveau courant sont détruites. Le passage à un niveau supérieur correspond à une vitesse accrue de la balle.

6 Composants fournis

6.1 Objects

Le composant Objects permet de gérer l'affichage des différents objets du jeu, les entrées-sorties de ce composant sont :

- Une entrée horloge `clk25` qui doit être de 25MHz
- Une entrée horloge `clk5Hz` qui doit être de 5Hz
- Une entrée `reset` qui est une remise à zéro asynchrone, cette entrée est active à l'état 0
- Une entrée `visible` qui indique si le pixel courant est dans la zone visible de l'écran
- Une entrée `endframe` qui indique si on est à la fin de l'image affichée courante
- Une entrée `xpos` sur 10 bits qui indique la position en x du pixel courant
- Une entrée `ypos` sur 10 bits qui indique la position en y du pixel courant
- Une entrée `rot_left` qui indique si la raquette se déplace sur la gauche
- Une entrée `rot_right` qui indique si la raquette se déplace sur la droite
- Une entrée `taille` qui indique 1 taille parmi 2 de la raquette, grande ou petite

- Une entrée `speed` qui indique 1 vitesse parmi 2 de la balle, lente ou rapide
- Une entrée `freeze` qui permet de mettre le jeu en mode pause
- Une entrée `miss_time` sur 6 bits
- Une sortie `bluebox` qui indique que le pixel courant est une case bleue
- Une sortie `bottom` qui indique que le pixel courant est en bas de l'écran
- Une sortie `wall` qui indique que le pixel courant appartient au mur
- Une sortie `pad` qui indique que le pixel courant appartient à la raquette
- Une sortie `brick` qui indique que le pixel courant appartient à une brique
- Une sortie `brick_bounce` de 2 lignes de 9 bits qui indique si il y a un rebond sur l'une de 18 briques
- Une sortie `ball` qui indique que le pixel courant appartient à la balle.

Le composant `Objects` gère à la fois l'évolution et l'affichage de tous les objets nécessaires au jeu : briques, raquette, balle et décor.

Ce composant prend en entrée les paramètres liés aux objets : `visible`, `endframe`, `xpos`, `ypos`, `rot_left`, `rot_right`, `taille`, `speed`, `freeze` et `miss_time`.

Il génère les informations nécessaires au jeu : `bluebox`, `bottom`, `wall`, `pad`, `brick`, `brick_bounce` et `ball`.

Toutes les entrées-sorties sont de type `std_logic` ou `std_logic_vector` sauf `brick` et `brick_bounce` qui sont de type `tableau`. Le type tableau est défini dans un paquetage appelé `pong_pack` comme suit :

```
type tableau is array (1 downto 0) of std_logic_vector(8 downto 0);
```

6.2 Game

Le composant game permet de gérer le jeu, les entrées-sorties de ce composant sont :

- Une entrée `clk25` qui est une horloge qui doit être de 25 MHz
- Une entrée `reset` qui est la remise à zéro asynchrone du composant, elle est active à 0
- Une entrée `press` qui permet de mettre en pause le jeu
- Une entrée `endframe` qui indique la fin de l'image visible
- Une entrée `visible` qui indique si le pixel courant est dans la zone visible de l'écran

- Une entrée `wall` qui indique que le pixel courant appartient au mur
- Une entrée `bottom` qui indique que le pixel courant est en bas de l'écran
- Une entrée `bluebox` qui indique que le pixel courant appartient à une case bleue de l'écran
- Une entrée `pad` qui indique que le pixel courant appartient à la raquette
- Une entrée `ball` qui indique que le pixel courant appartien à la balle
- Une entrée `brick_bounce` qui indique qu'il y a un rebond contre une brique
- Une sortie `red` qui indique la valeur du canal rouge du pixel courant
- Une sortie `green` qui indique la valeur du canal vert du pixel courant
- Une sortie `blue` qui indique la valeur du canal bleu du pixel courant
- Une sortie `miss_timer` sur 6 bits
- Une sortie `freeze` qui indique que le jeu est en pause

Le composant `Game` a pour fonction de gérer l'évolution du jeu, il acquiert en entrée les éléments nécessaires à cette évolution, qui sont `press`, `endframe`, `visible`, `wall`, `bottom`, `bluebox`, `pad`, `ball` et `brick_bounce` et génére à la fois les couleurs du pixel à afficher via les signaux `red`, `green` et `blue` et l'arrêt ou la suspension du jeu à l'aide des signaux `miss_timer` et `freeze`.

Toutes les entrées-sorties sont de type `std_logic` ou `std_logic_vector` sauf `brick` et `brick_bounce` qui sont de type `tableau`. Le type tableau est défini dans un paquetage appelé `pong_pack` comme suit :

```
type tableau is array (1 downto 0) of std_logic_vector(8 downto 0);
```

6.3 ADXL362Ctrl

Le composant ADXL362Ctrl permet de récupérer les valeurs en sortie de l'accéléromètre pour les coordonnées en x, y et z. Les entrées-sorties de ce composant sont :

- une entrée `SYSCLK` qui est l'horloge système du composant et qui doit être de 25 MHz
- une entrée `RESET` active à '1' qui permet de faire une remise à zéro asynchrone du composant
- une sortie `ACCEL_X` qui fourni la valeur sur 12 bits de l'accélération en x
- une sortie `ACCEL_Y` qui fourni la valeur sur 12 bits de l'accélération en y

- une sortie `ACCEL_Z` qui fourni la valeur sur 12 bits de l'accélération en z
- une sortie `ACCEL_TMP` qui fourni la valeur sur 12 bits de la température
- une sortie `Data_Ready` qui indique quand un nouveau jeu de données est valide
- une sortie `SCLK` qui correspond à l'horloge du bus SPI
- une sortie `MOSI` qui correspond à la sortie série du bus SPI
- une entrée `MISO` qui correspond à la sortie série du bus SPI
- une sortie `SS` qui correspond au signal de sélection du bus SPI