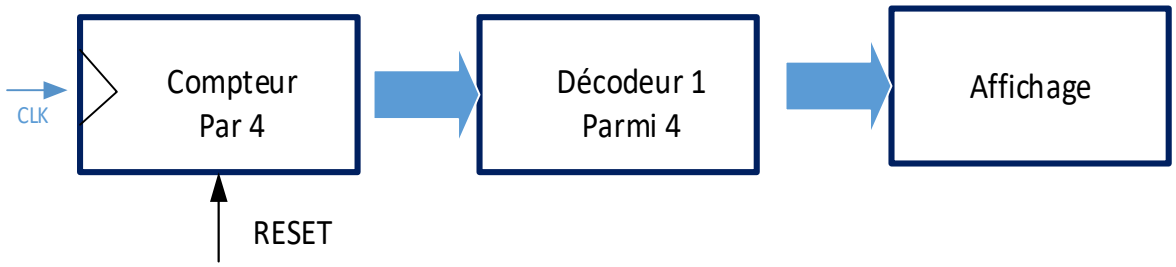
Utilisation de la logique programmable de la carte ZYBO

1) Problématique

Comme on a pu le réaliser auparavant avec la carte SPARTAN 6, on souhaite réaliser un chenillard. Cette séance nous permettra de nous familiariser avec la suite logiciel VIVADO 2014.4. Cette suite est la plateforme de développement logiciel privilégiée pour travailler avec l'environnement ZYNQ. On ne s'intéressera ici qu'à la partie logique programmable (PL) du ZYNQ, qui intègre un FPGA ARTIX7. La simulation VHDL Conclura cette séance.

Le bloc fonctionnel du chenillard est donné ci-dessous



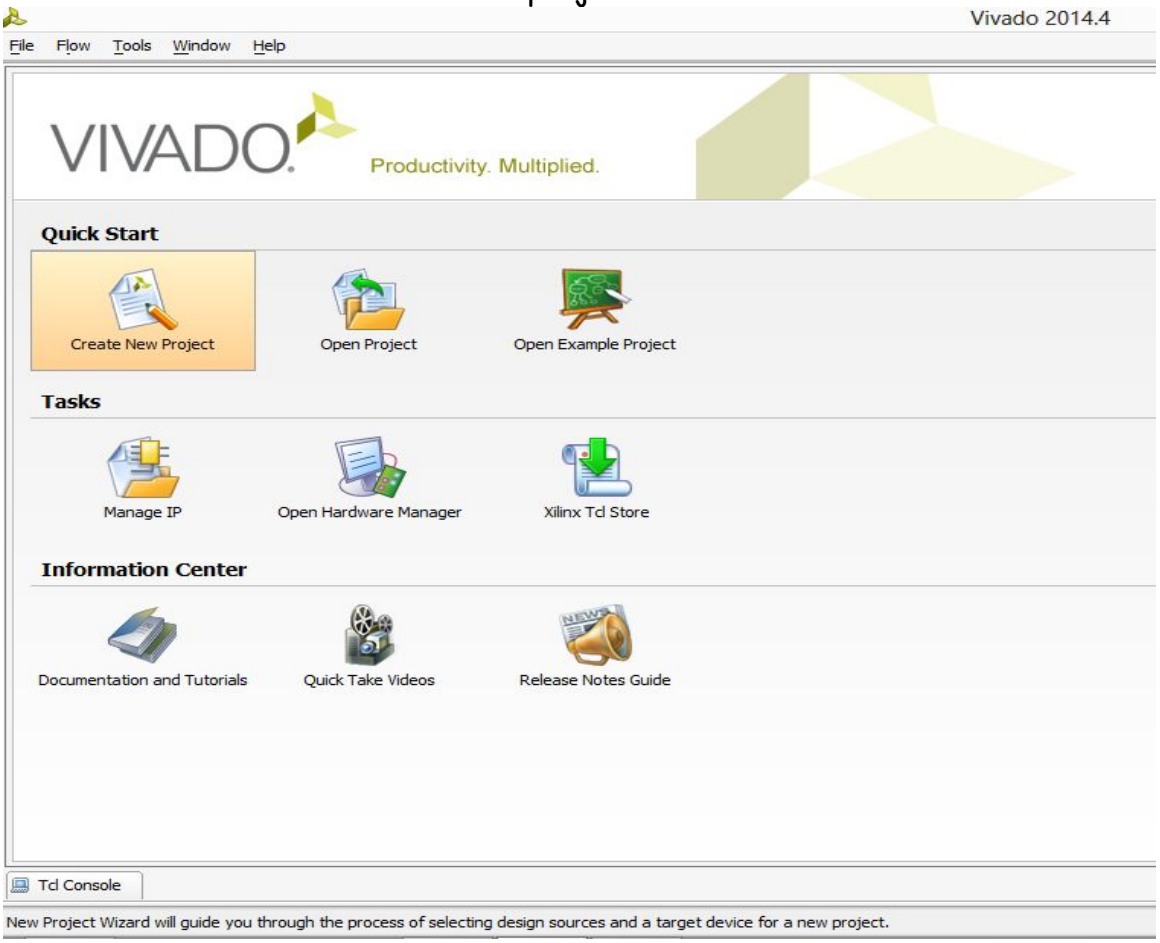
Chaque bloc fonctionnel sera décrit par un IPCORE. La consigne RESET sera prise en compte grâce à l'interrupteur SW0 de la carte ZYBO. L'affichage s'effectuera
Sur les 4 LEDS de la carte. On devra insérer un diviseur de fréquence afin de diminuer la fréquence du signal CLK

2) Création du bloc design sous VIVADO

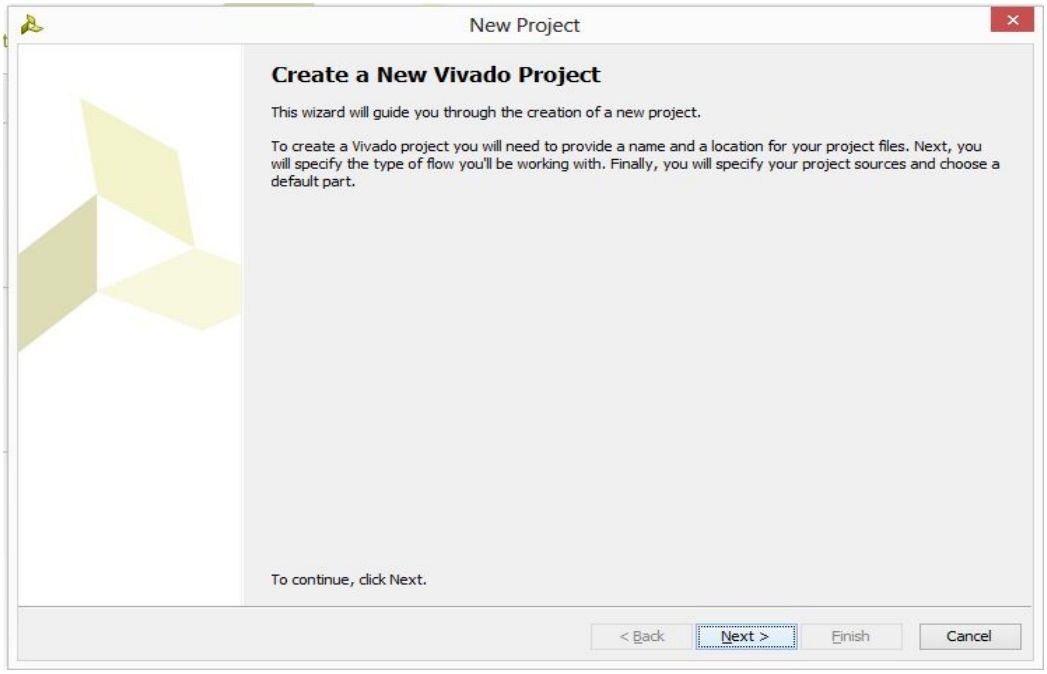
Ouvrir le logiciel Vivado



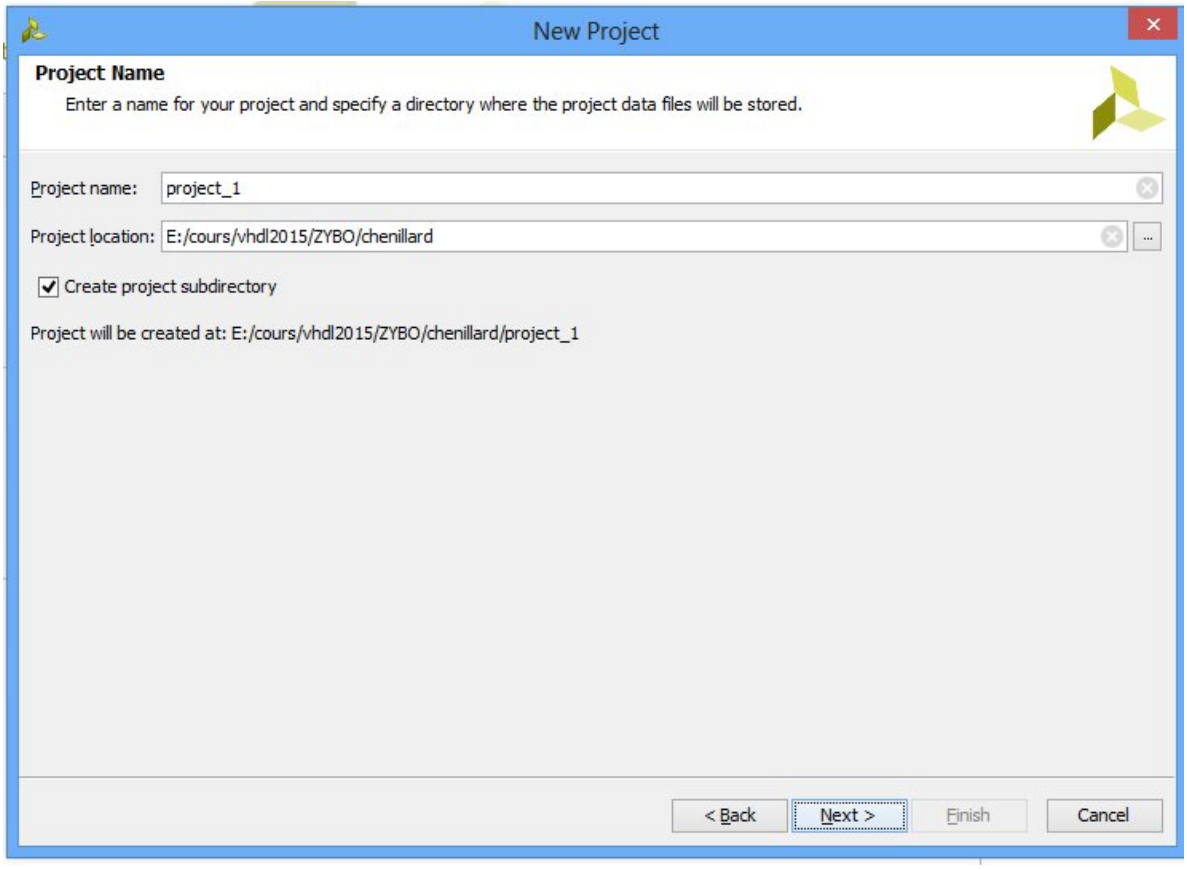
Lancez Vivado et créez un nouveau projet



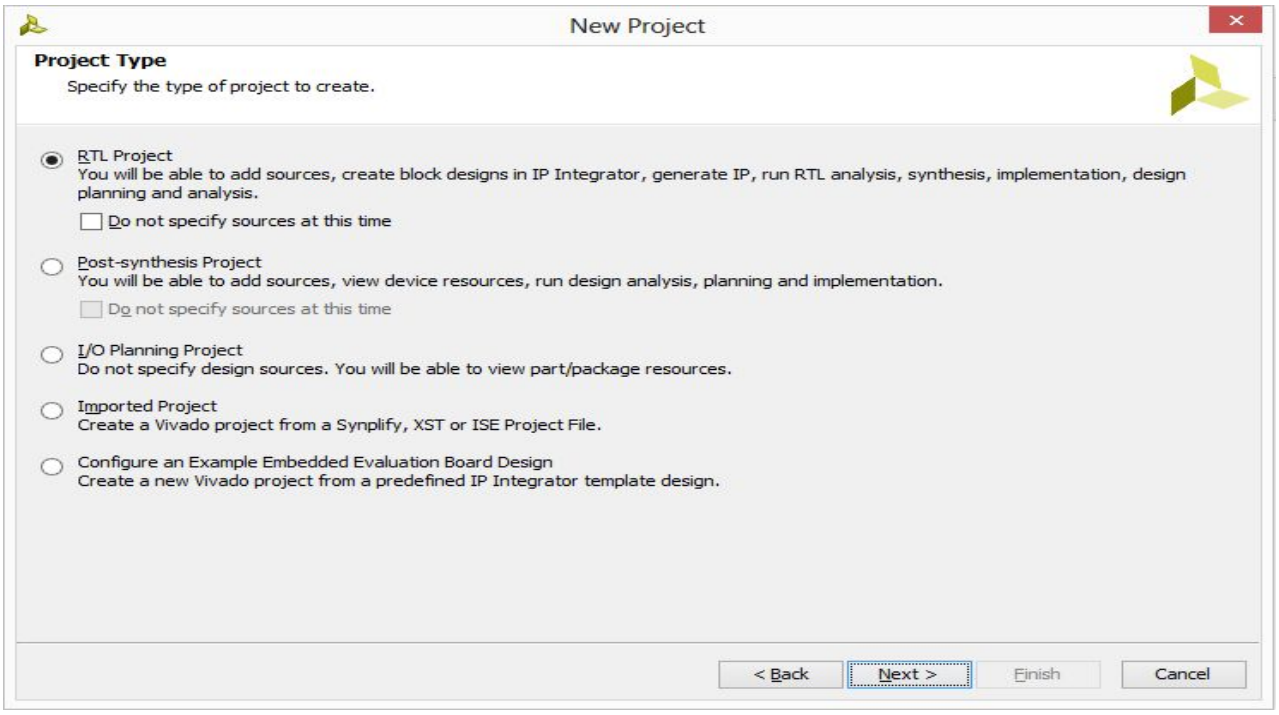
Cliquez sur next



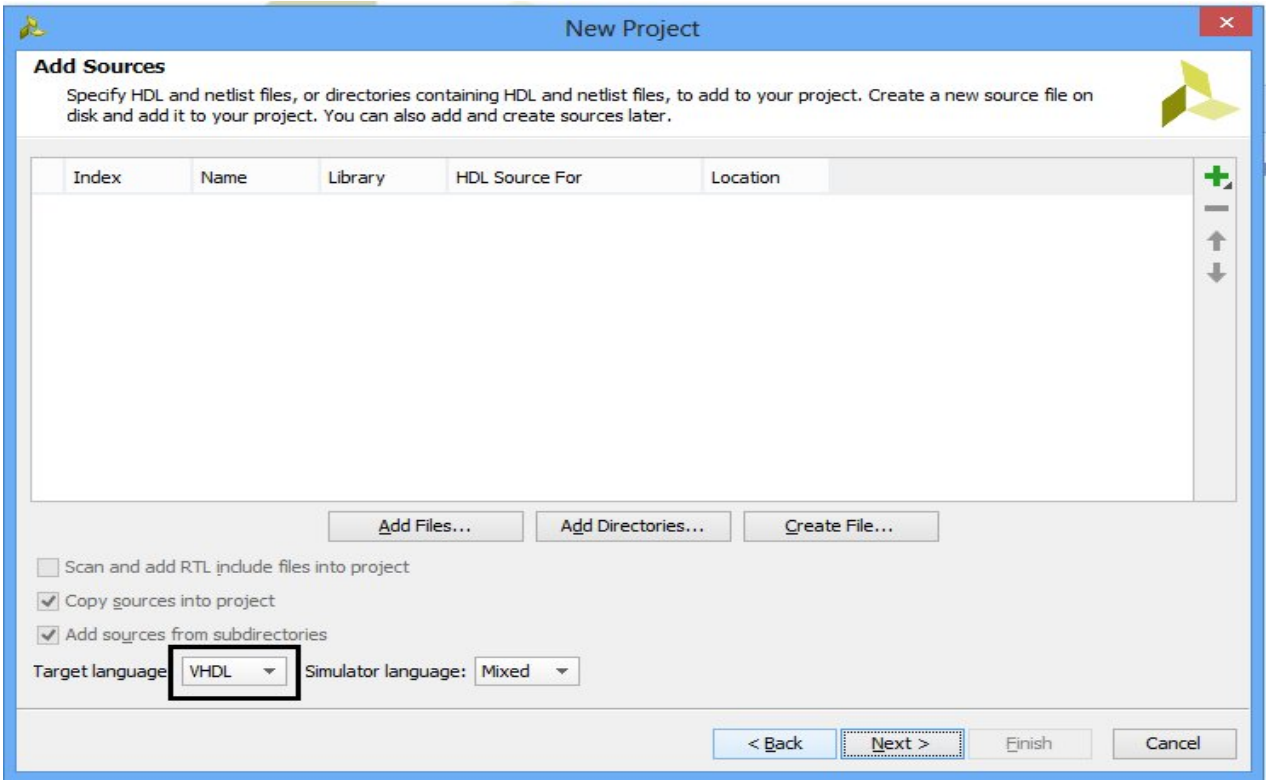
Nommez votre projet et choisir un répertoire pour ce projet



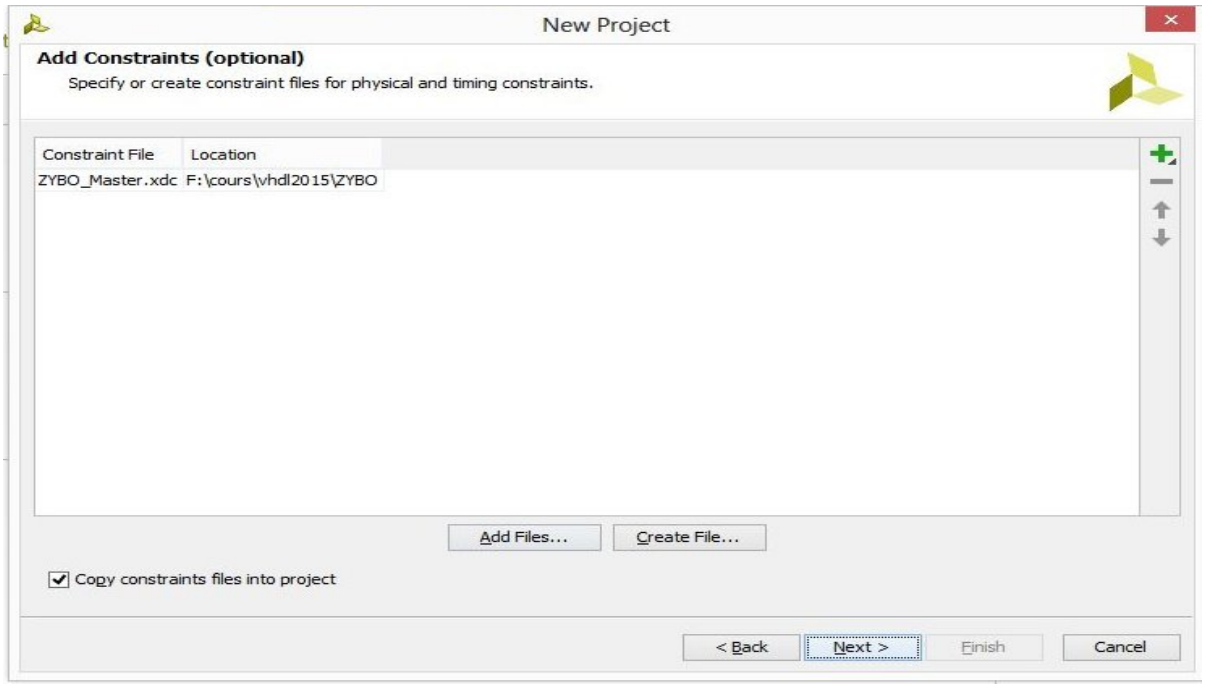
Choisir un projet du type RTL



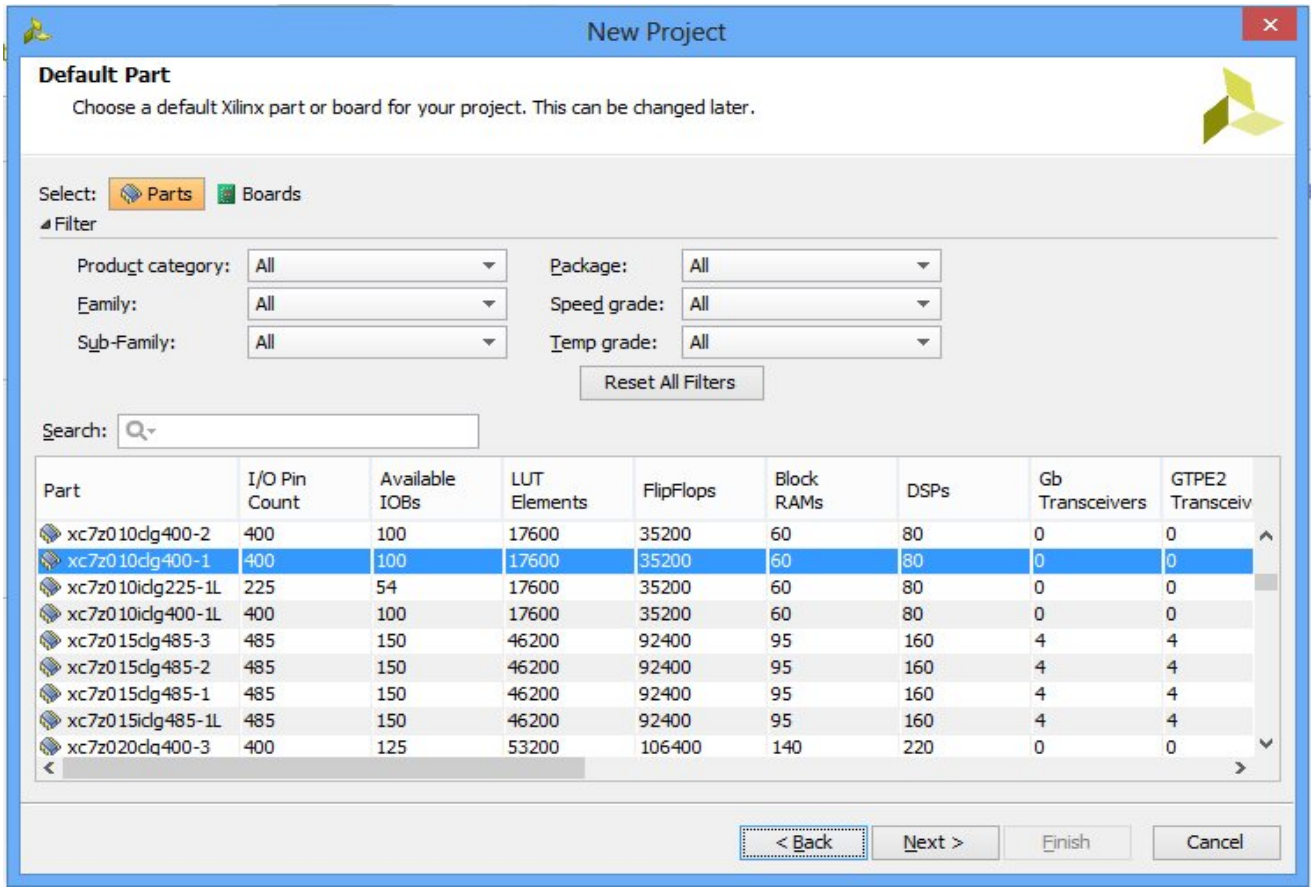
Si vous possédez des fichiers VHDL à inclure dans votre projet



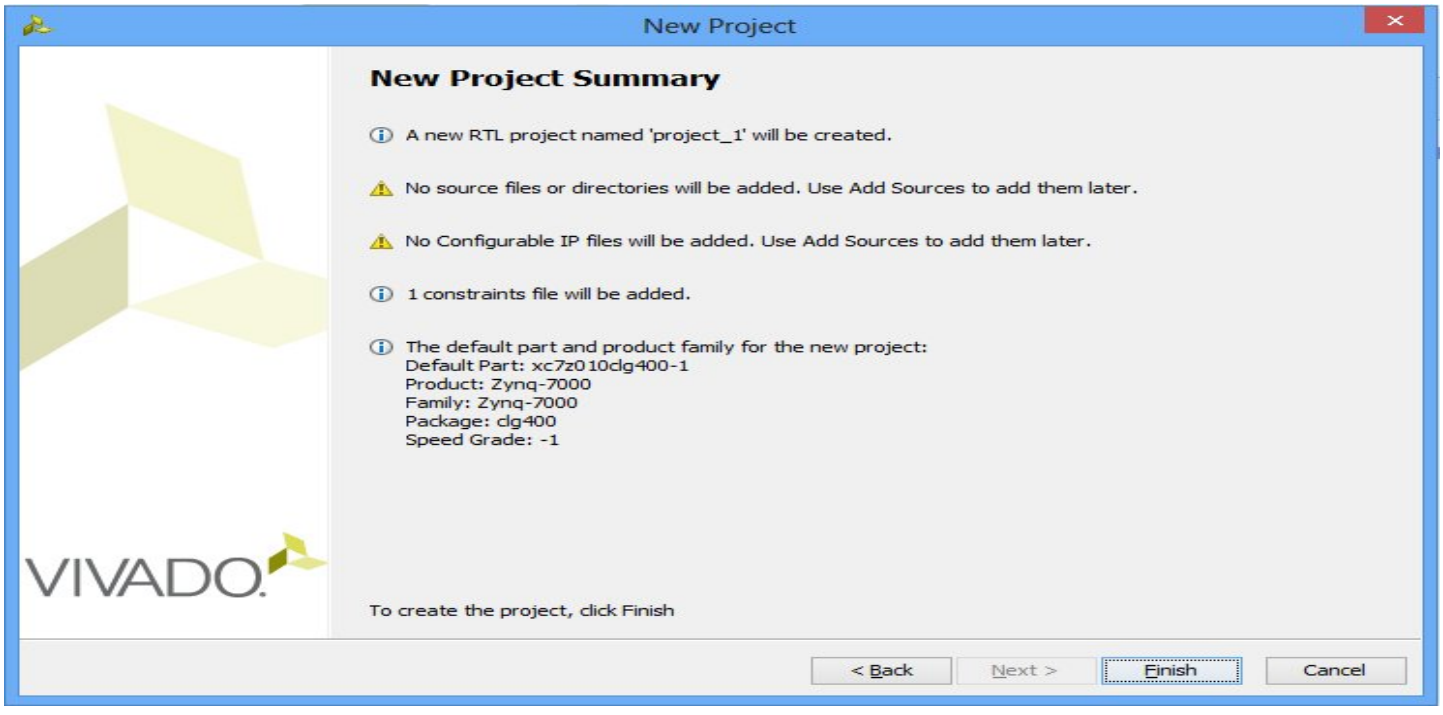
Inclure le fichier de contrainte pour votre carte de développement (dans notre cas particulier, la carte ZYBO)



Choisir le type de carte de développement ou le type de FPGA implanté sur votre carte :



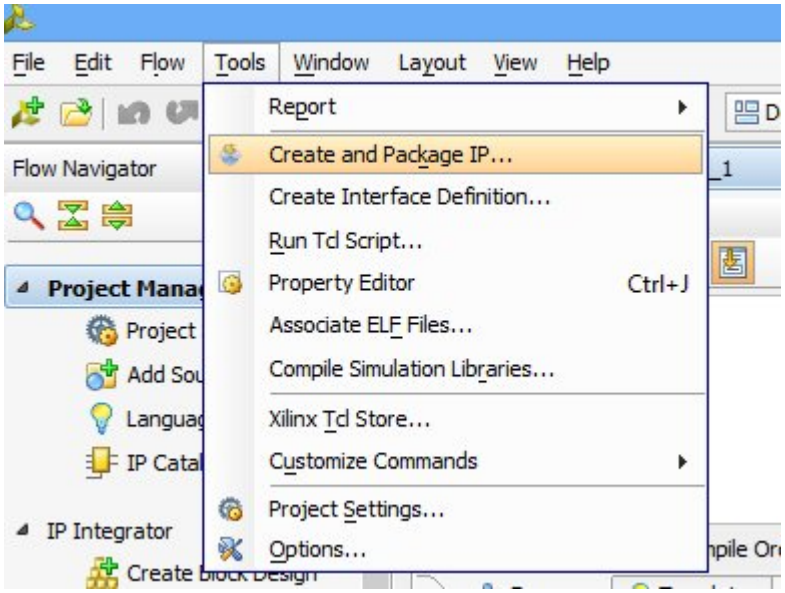
Cliquez sur finish dans le panneau récapitulatif

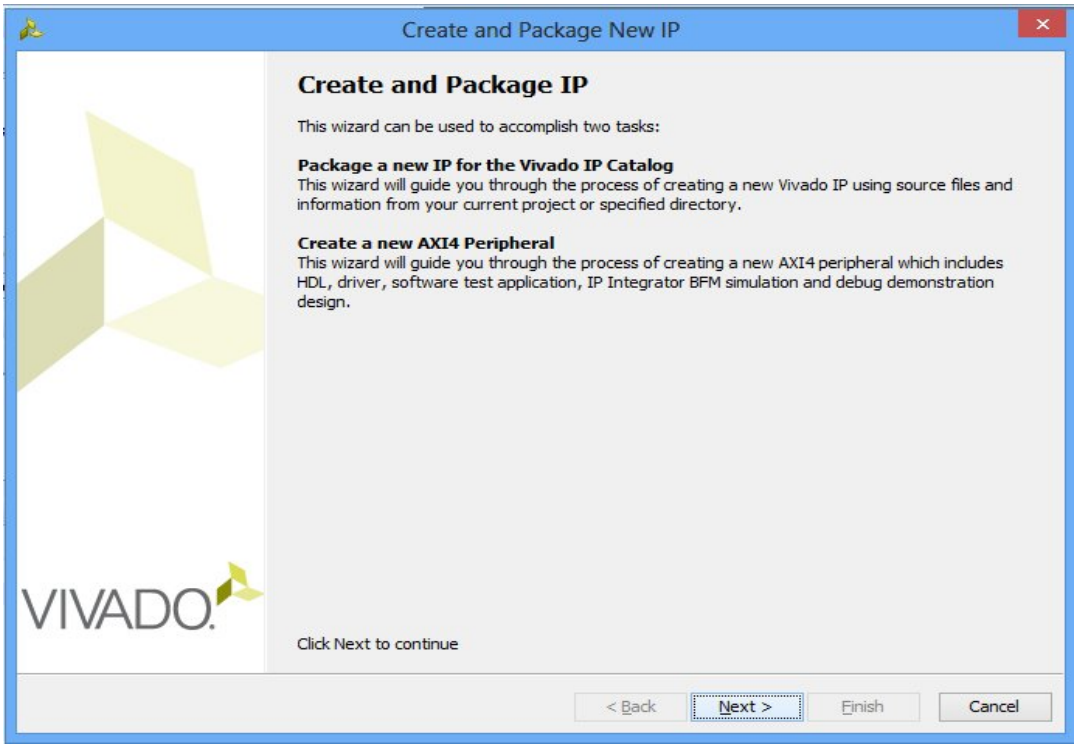


Dans le cas du chenillard, il nous faut inclure un compteur, un décodeur et un diviseur de fréquence. Créez un répertoire pour chacun des éléments et y placer le fichier VHDL associé à chacun. Cette dernière étape nous permettra l'intégration de chaque élément dans un IPCORE



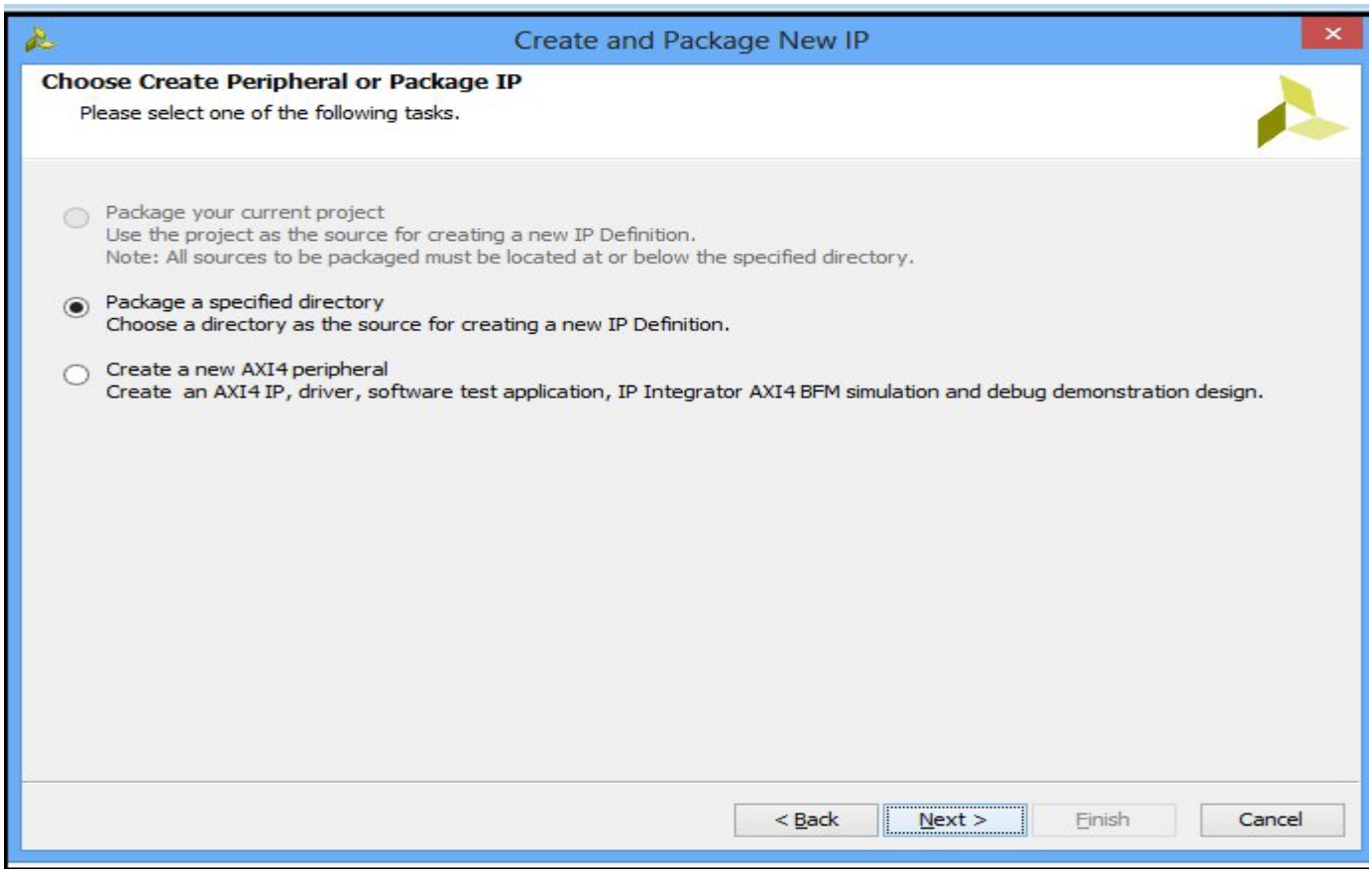
On va par la suite créer un package IP pour chaque élément :



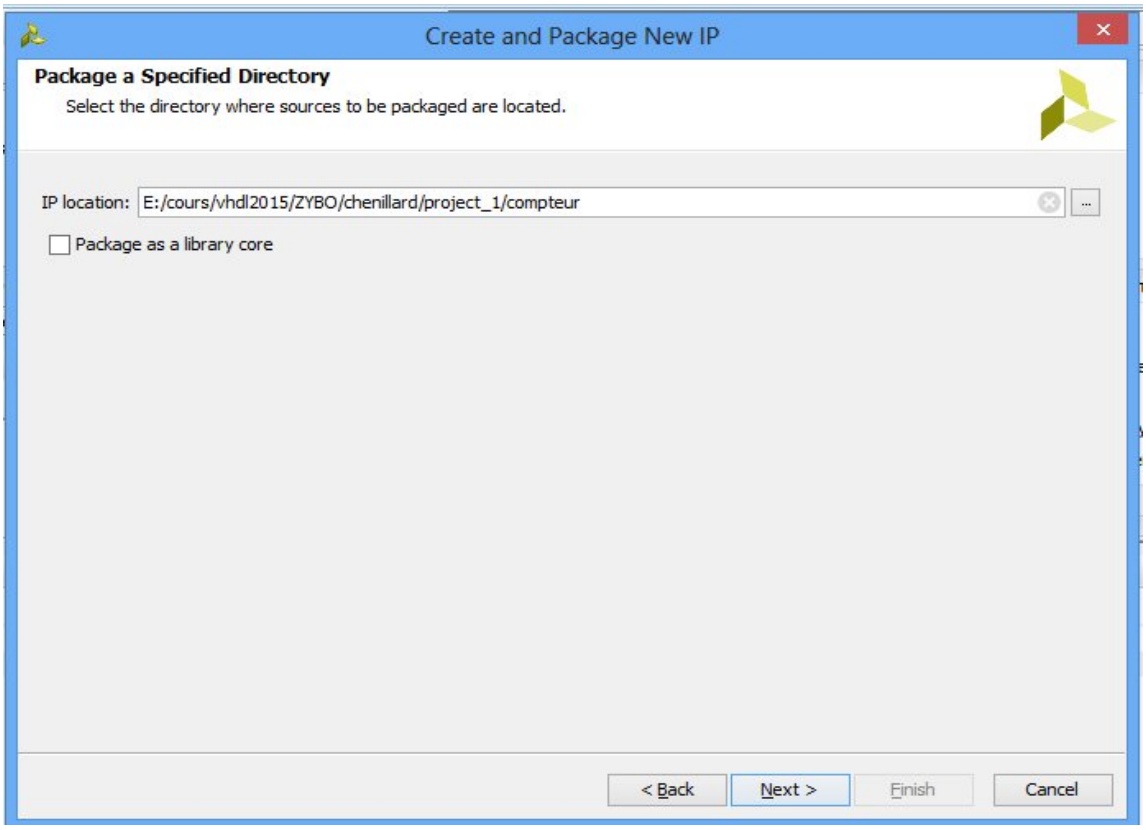


Cliquez sur next

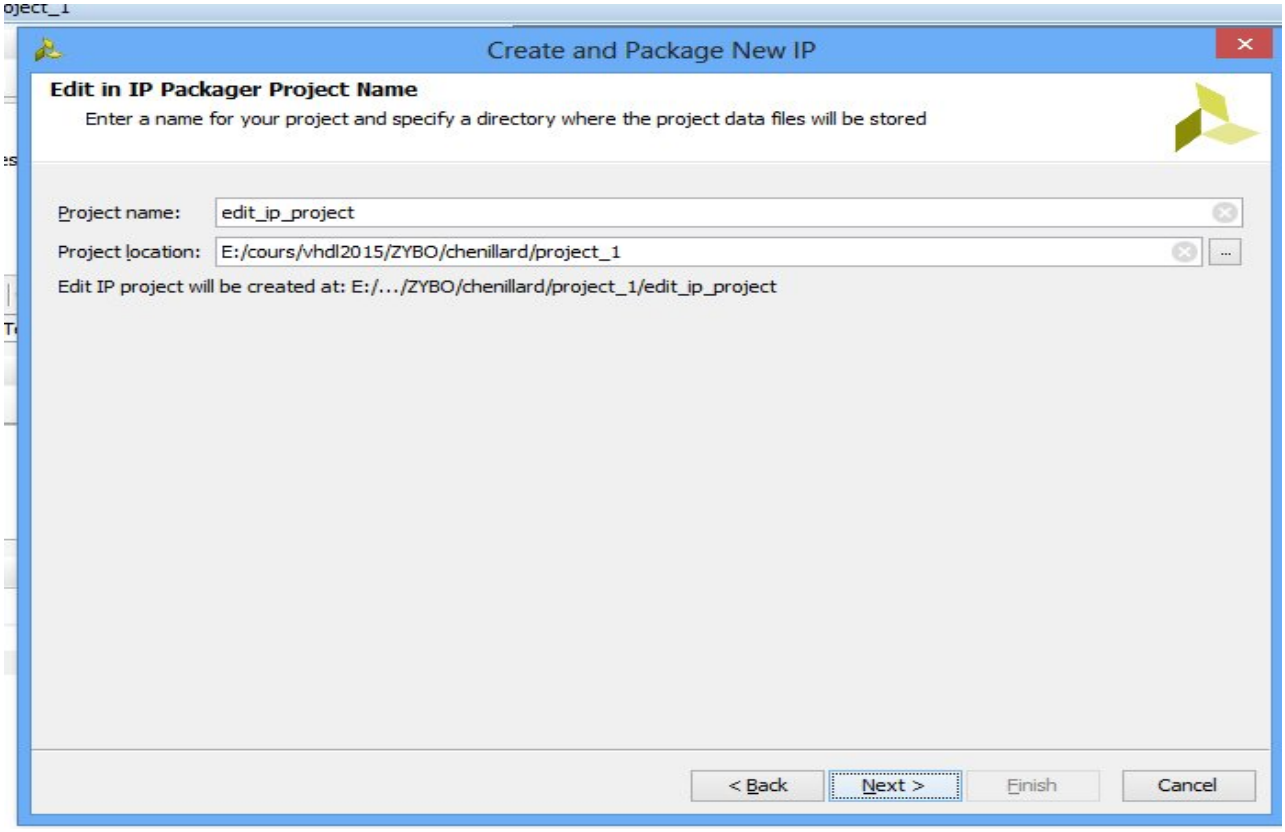
Sélectionnez package dans un répertoire



Fournir le chemin du répertoire d'un des trois éléments. Le répertoire doit contenir le fichier VHDL de l'IPCORE à implanter



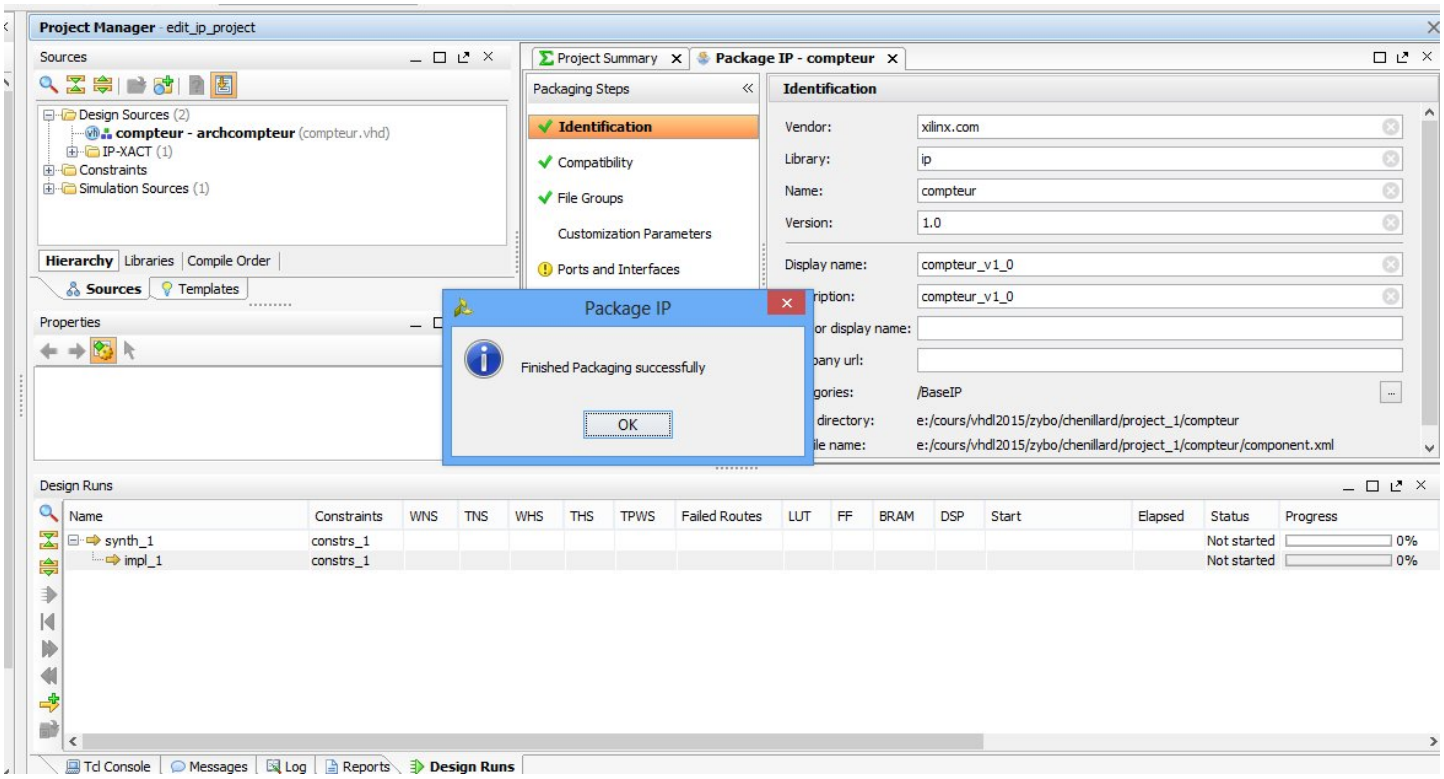
Choisir un nom de projet pour l'intégration de cet élément comme IPCORE



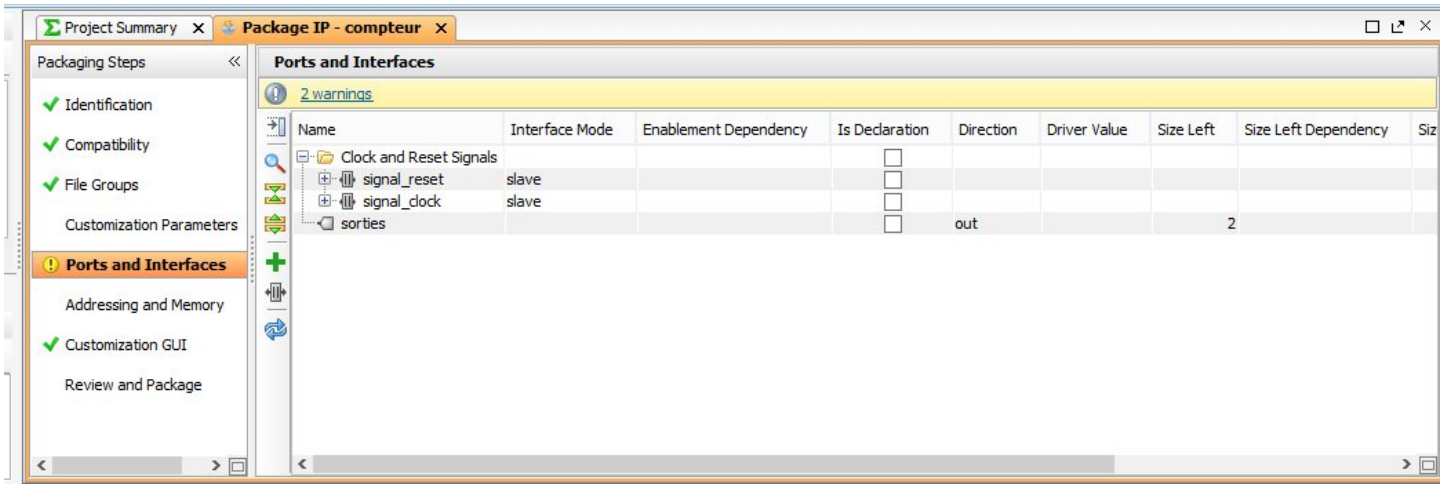
Cliquez sur finish dans le panneau récapitulatif



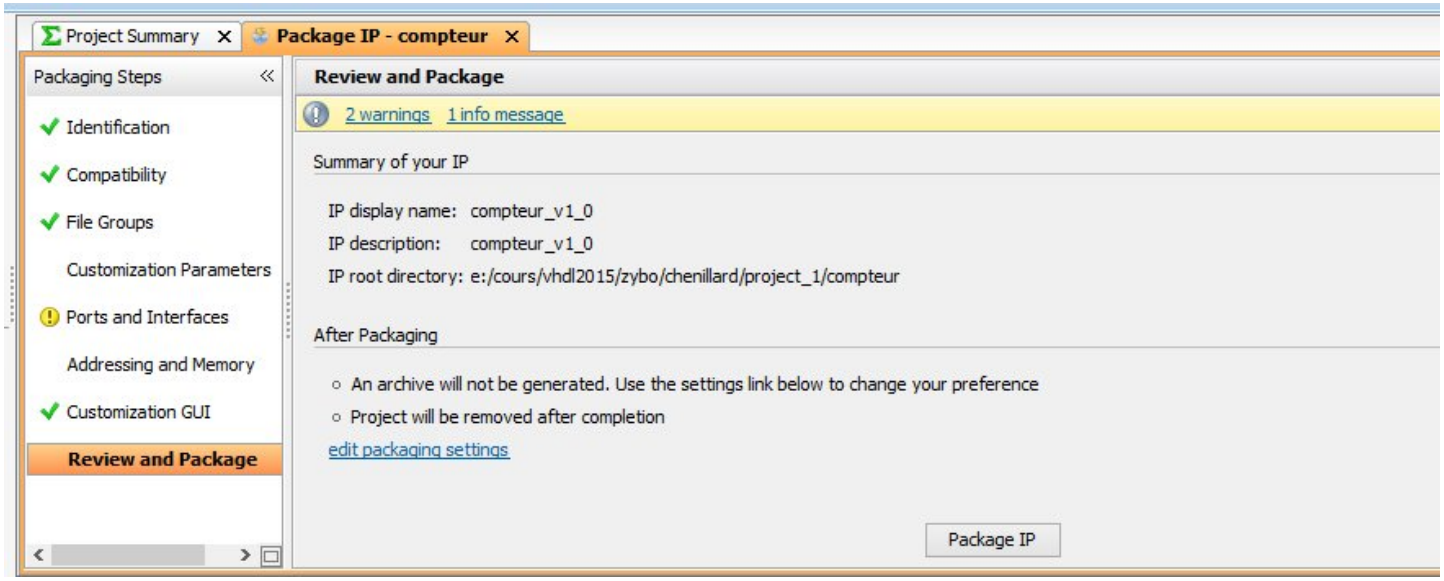
Le logiciel Vivado se lance pour l'intégration de cet IPCORE



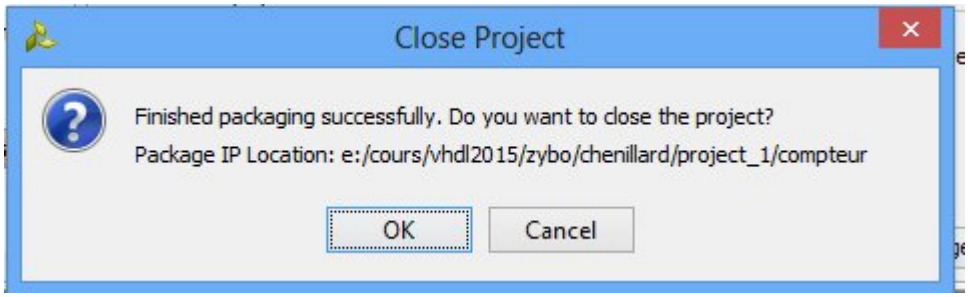
Vérifiez sur Ports et interfaces nos différents signaux



Puis cliquez sur Review and Package pour l'intégration finale. Cliquez à nouveau sur le bouton Package IP



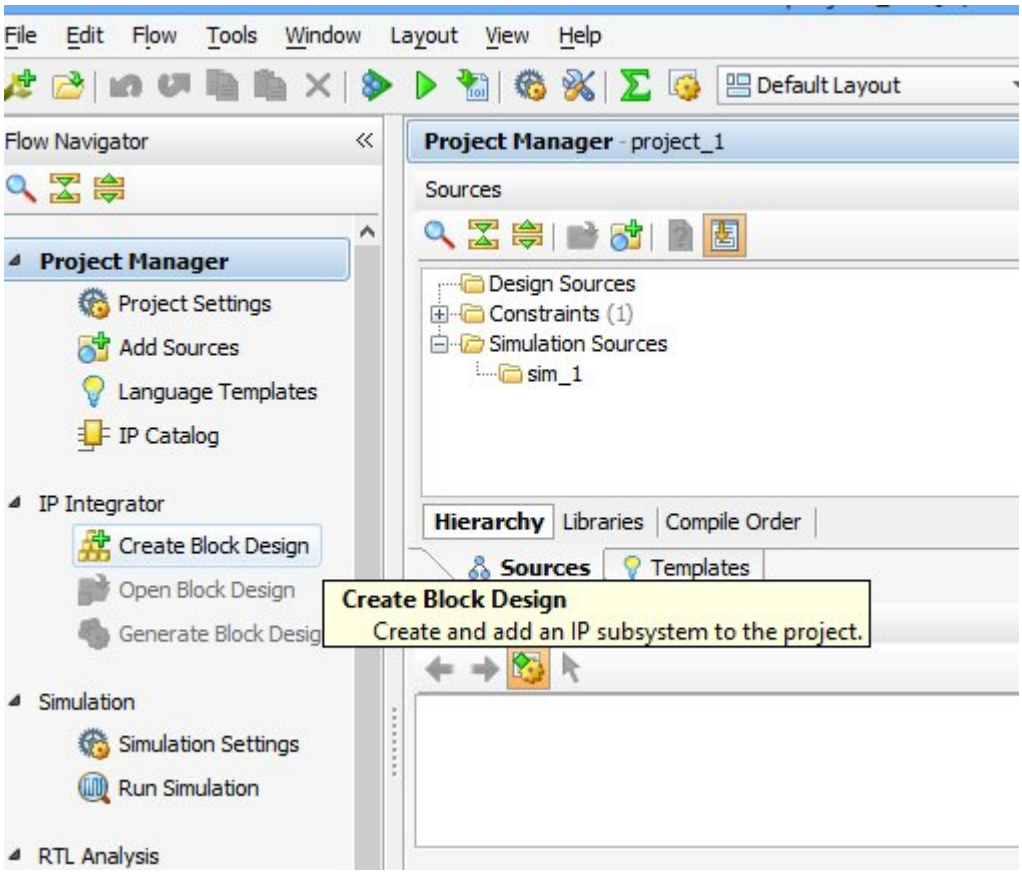
La fenêtre de réussite



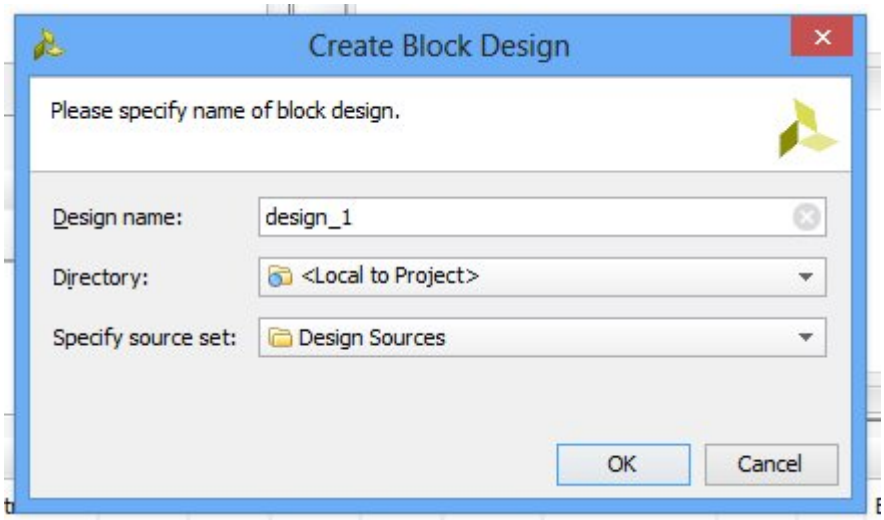
Reprendre la même procédure pour les deux autres éléments.

Une fois les trois éléments intégrés on va créer un block design afin d'associer ces trois éléments entre eux et réaliser ainsi un chenillard.

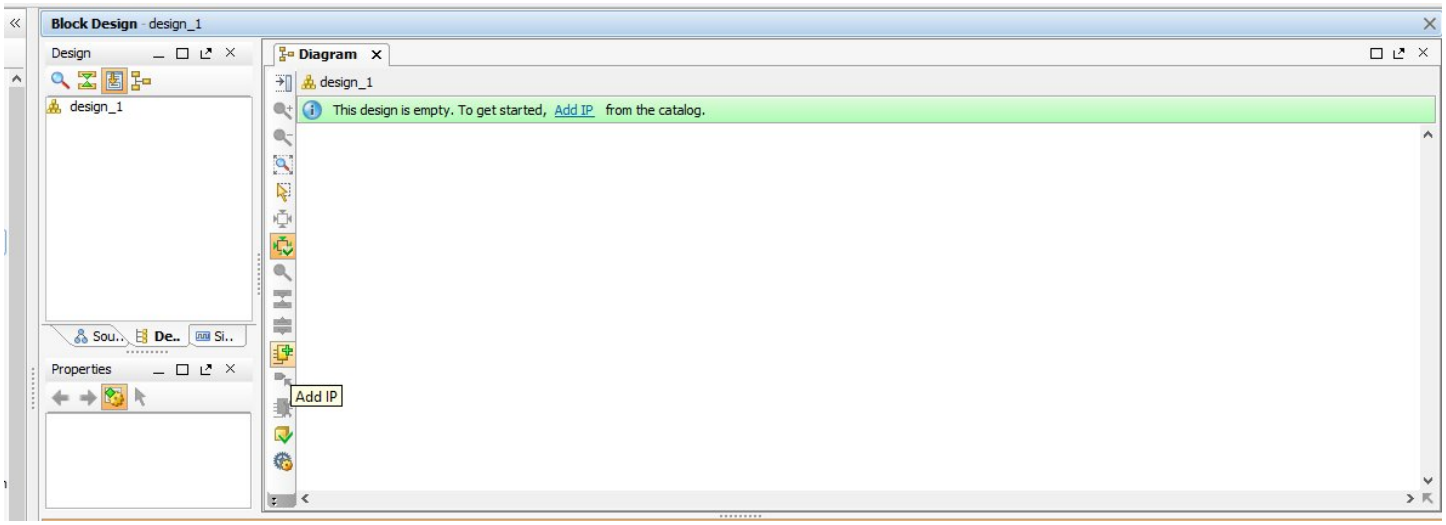
Cliquez sur create block design



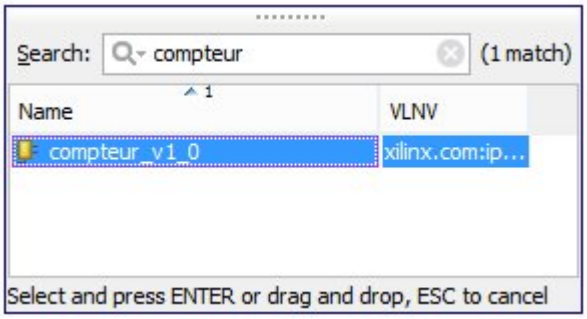
Choisir un nom pour notre block design

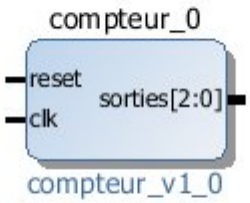


La fenêtre de construction du block design apparait
Cliquez sur add ip pour rechercher un des trois éléments



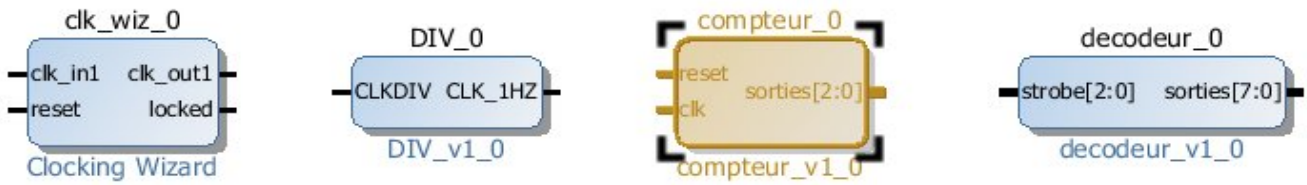
Le premier élément le compteur





Procédez de même pour les deux éléments suivants

On obtient alors le schéma suivant en intégrant aussi un clock wizard

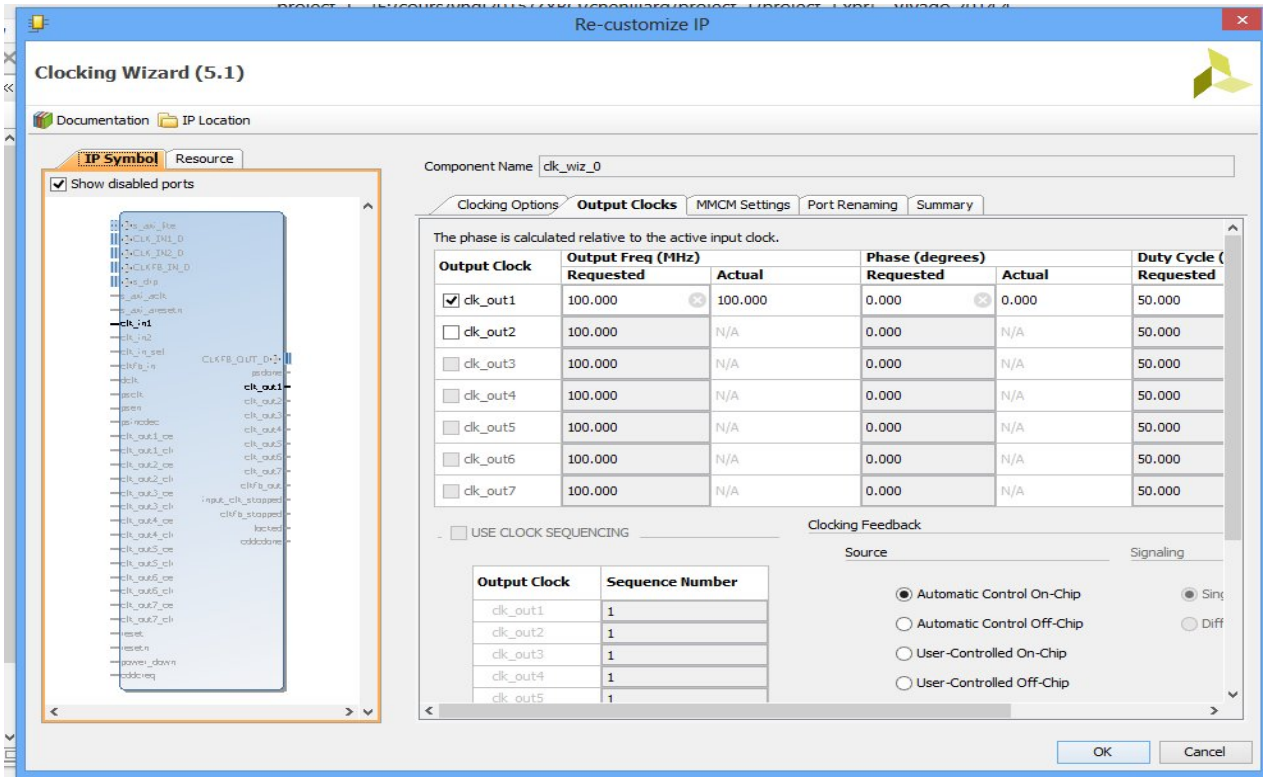


On peut faire apparaitre les caractéristiques d'un ipcore en double cliquant dessus



Double cliquez

On obtient la fenêtre suivante :

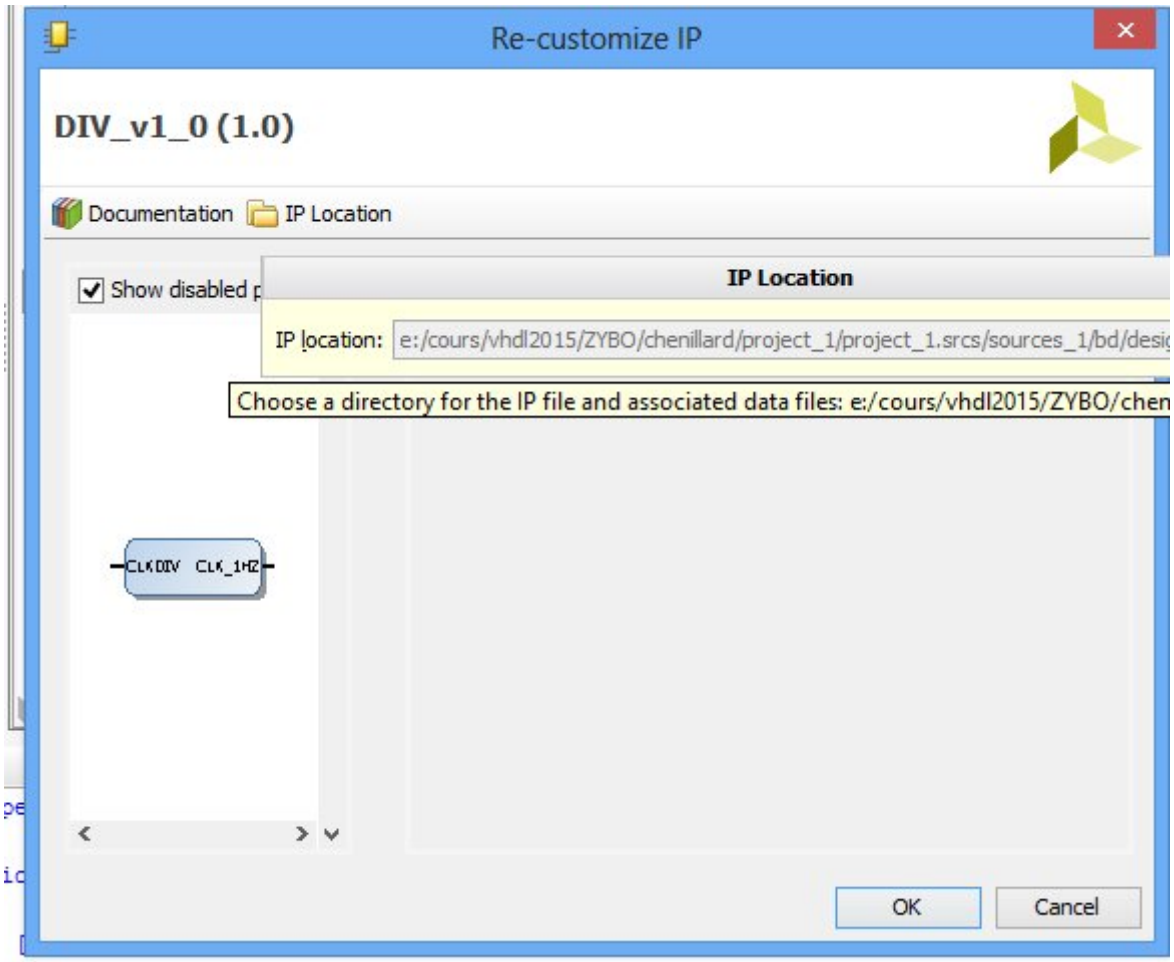


Supprimez la broche reset et locked de l'ipcore clocking wizard.

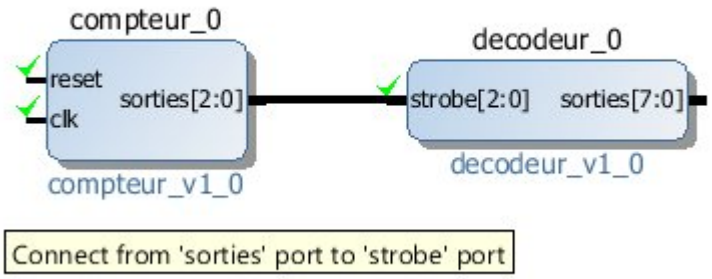
Ci-dessous on retrouve les caractéristiques de l'ipcore diviseur de fréquence.



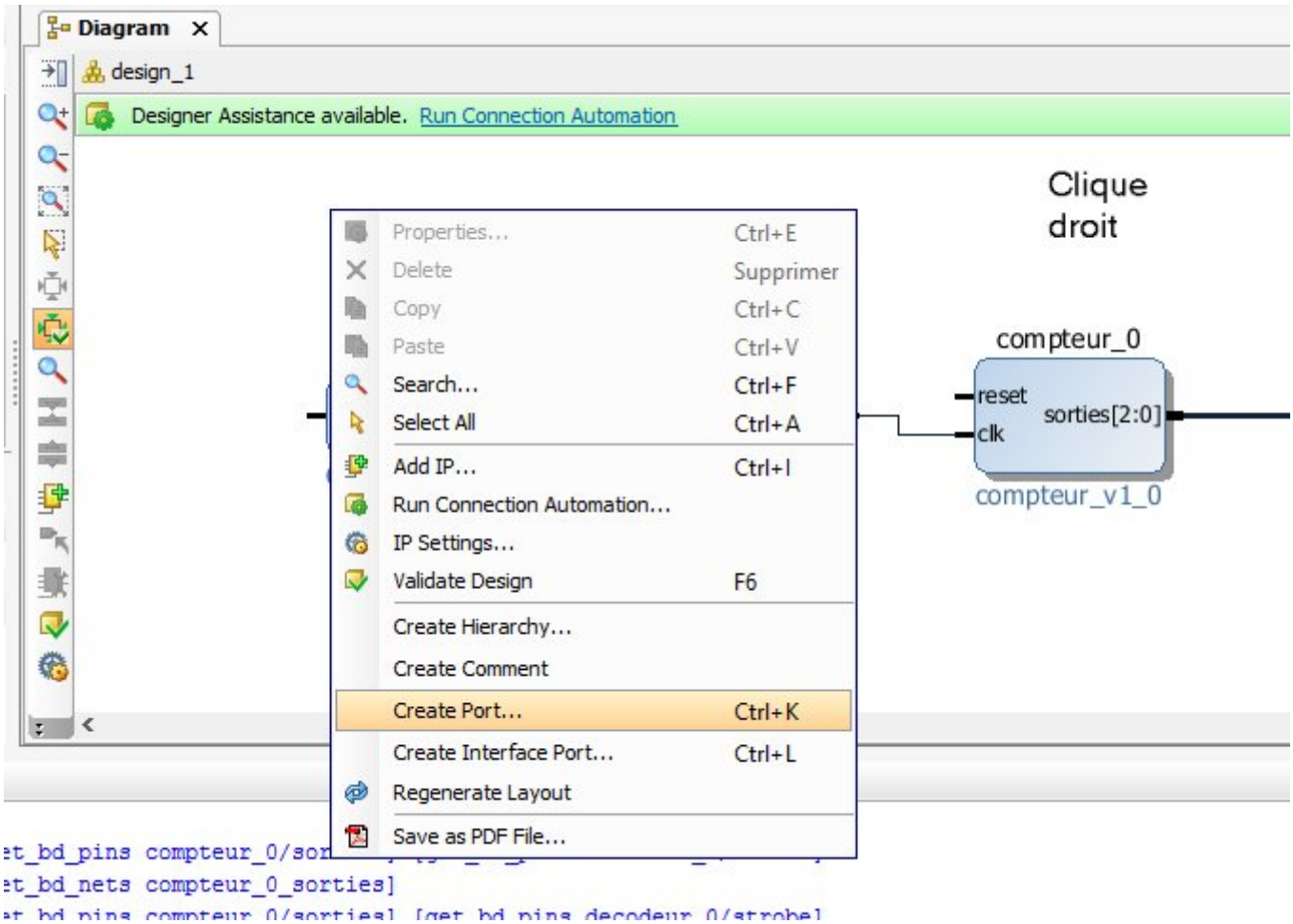
Double cliquez



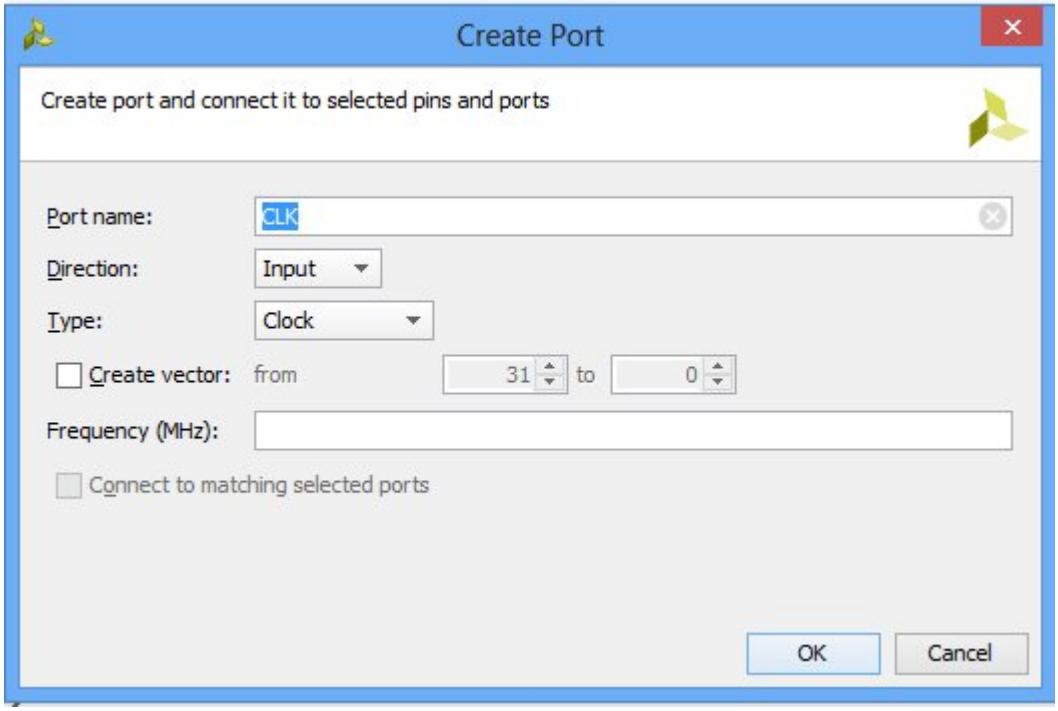
Associez grâce à l'outil fil le câblage entre les différents éléments du montage.



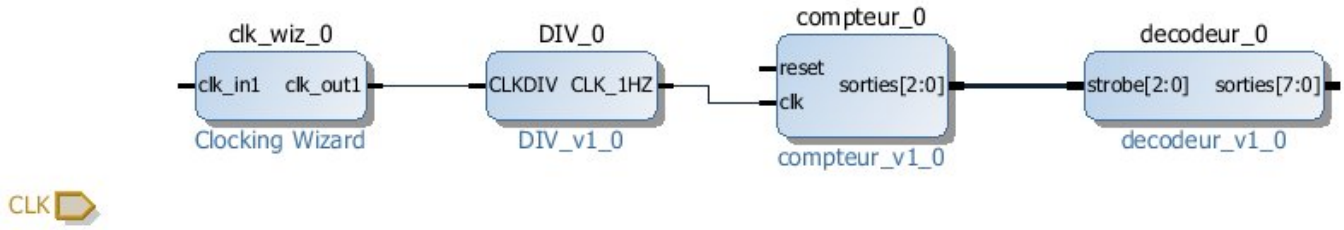
Pour créer un port cliquez droit sur le design block puis sélectionner create port



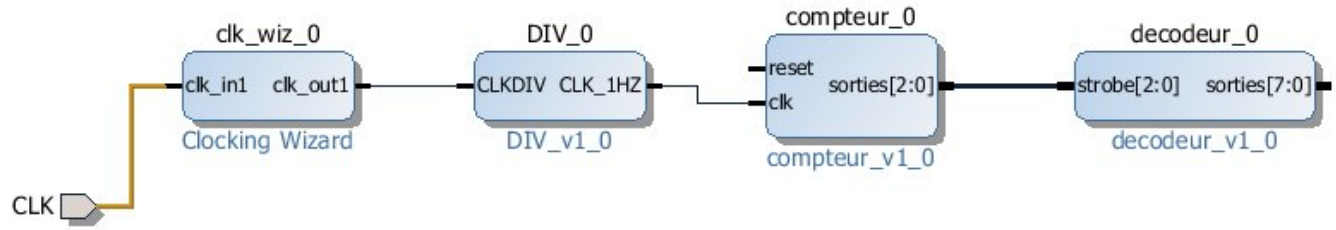
Donnez un nom au port (ici CLK) et choisir sa direction et son type



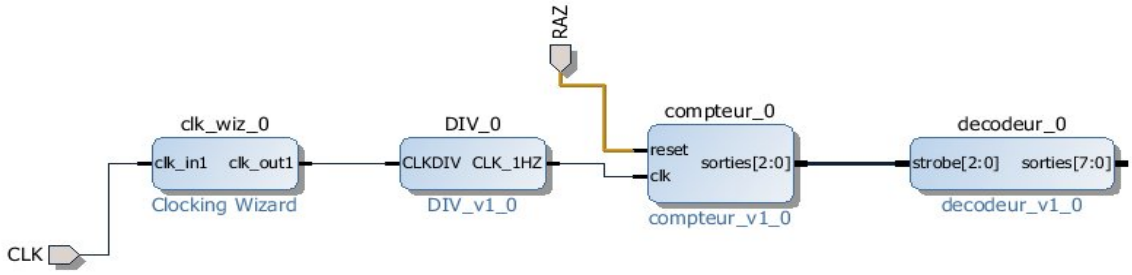
Le schéma devient le suivant :



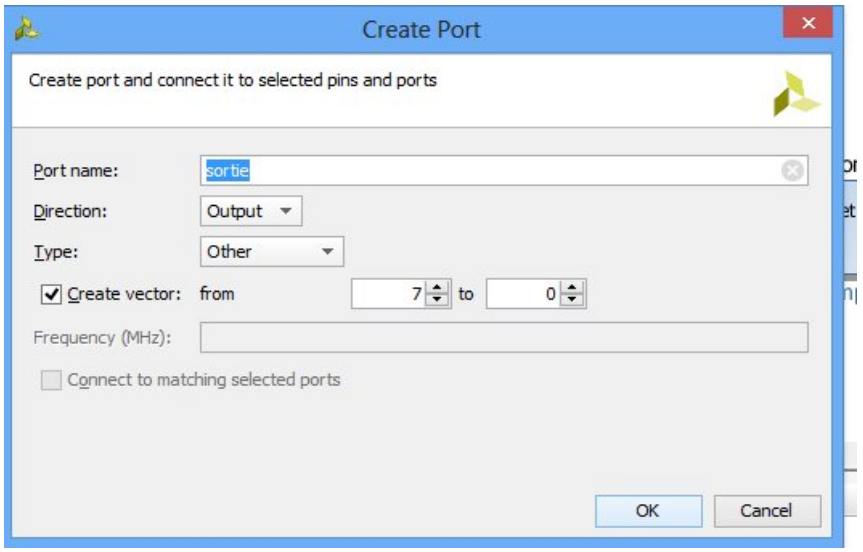
Reliez grâce à l'outil fil la broche clk_int au port d'entrée CLK



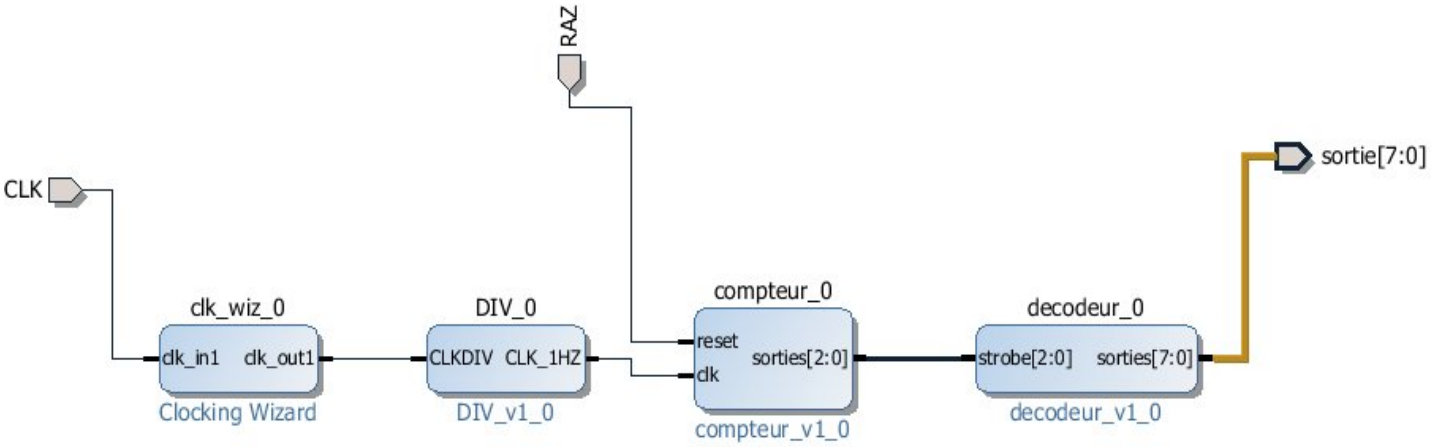
Même procédure pour le port RAZ



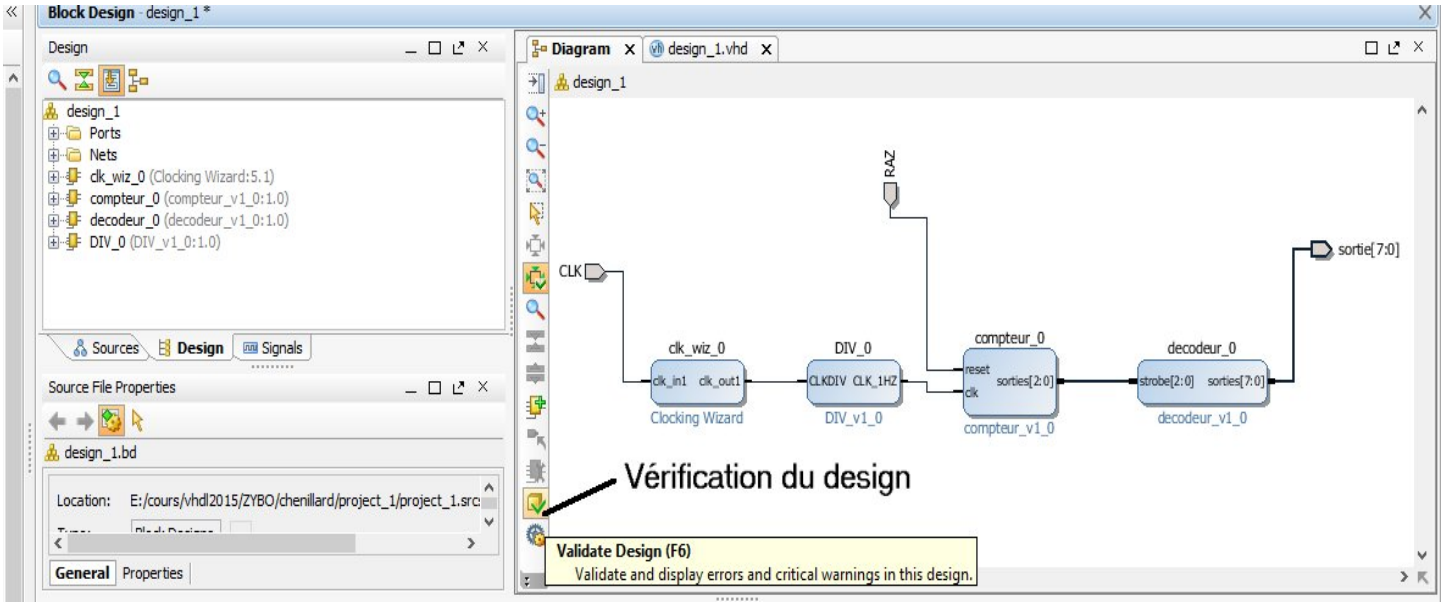
On doit déclarer un bus de huit bits pour le port de sortie



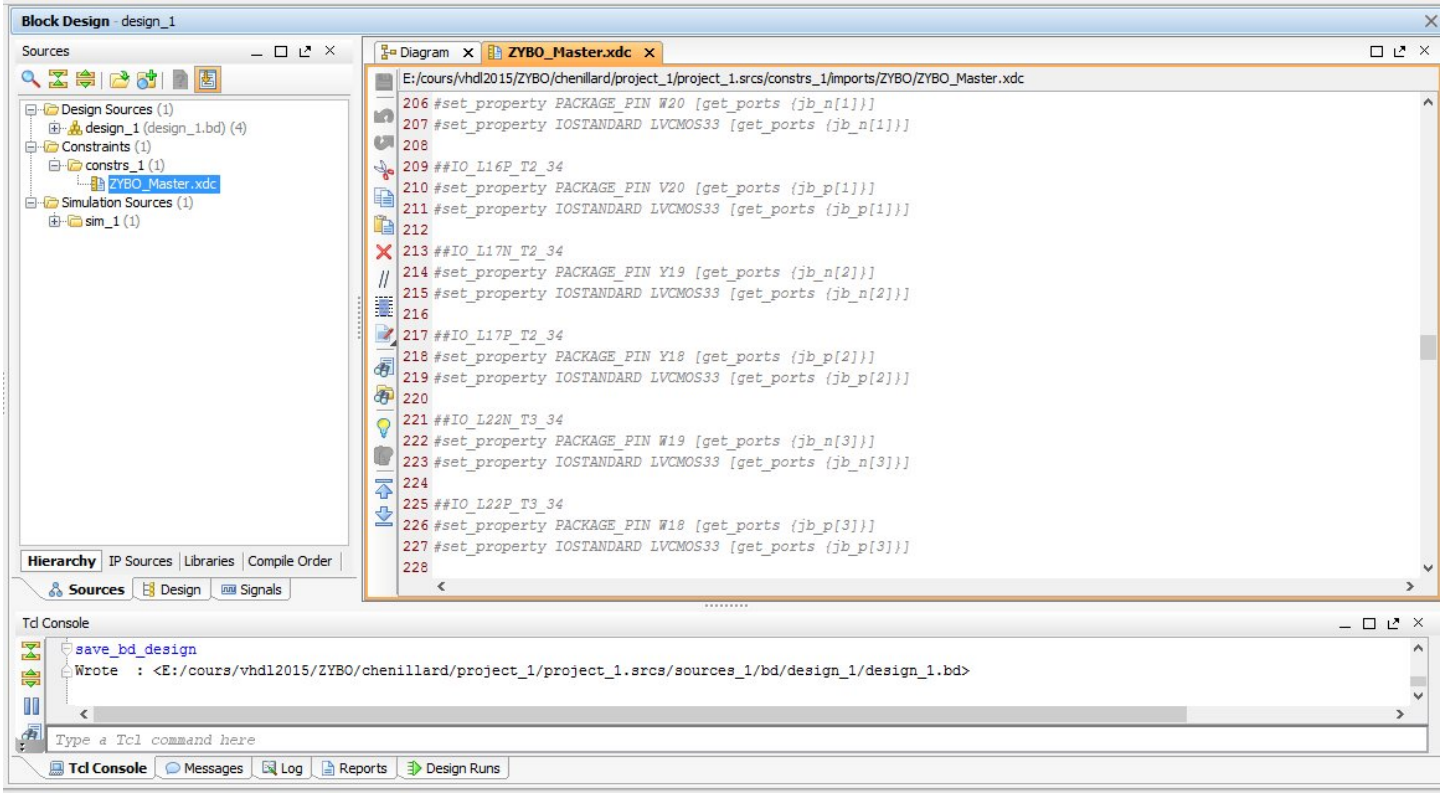
Finalisez le câblage :

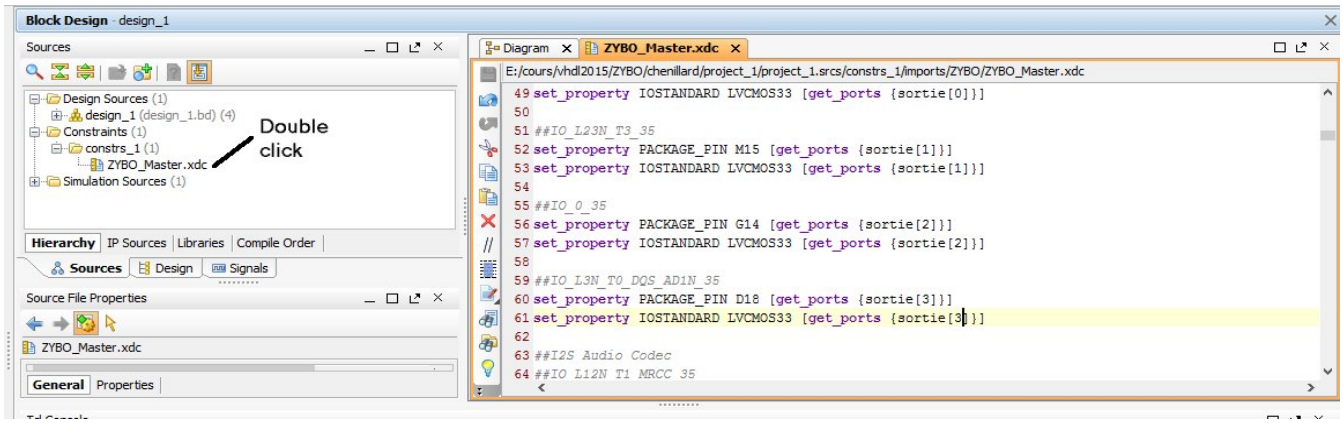


On peut vérifier la conformité du block design en cliquant sur validate design

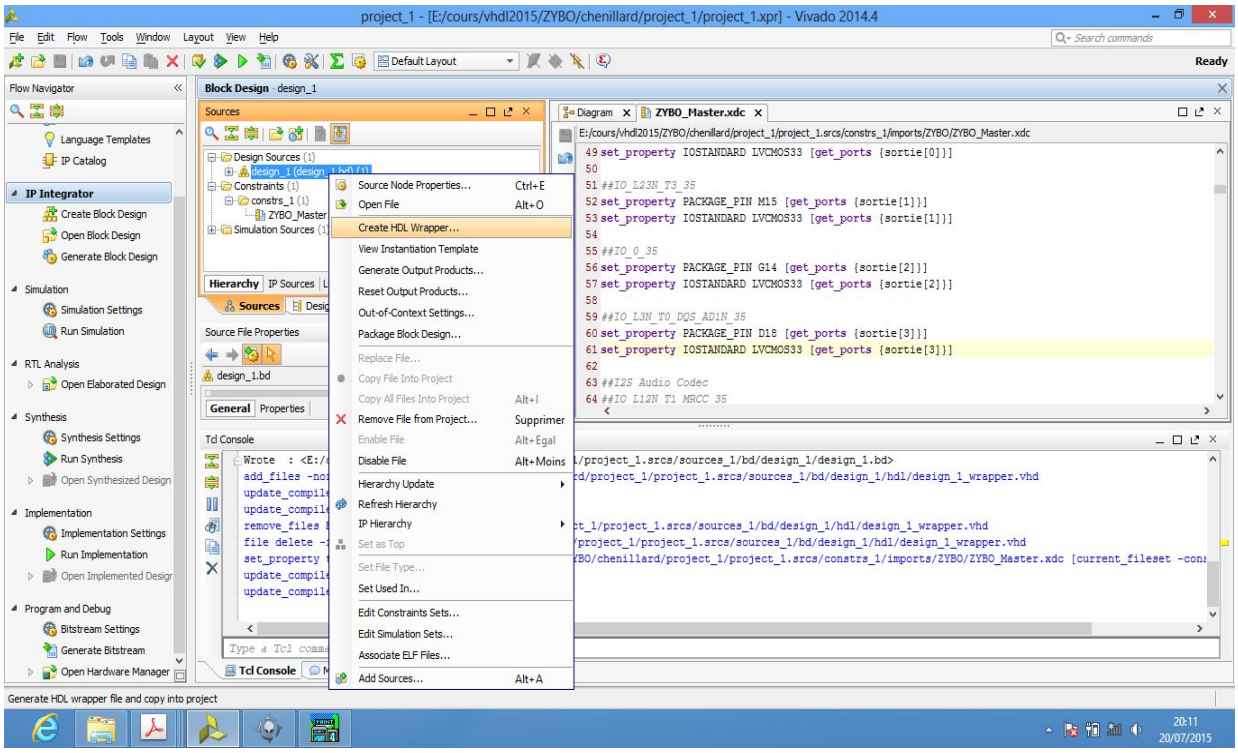


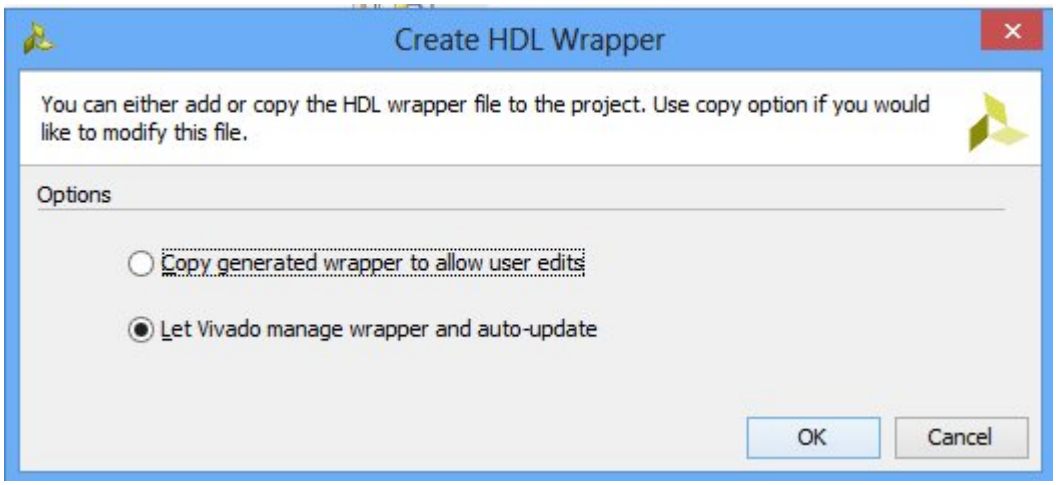
Chaque port du block design doit être déclaré dans le fichier de contrainte de la carte. Double cliquez sur le fichier de contrainte et faire les déclarations nécessaires en associant à un port une broche du FPGA



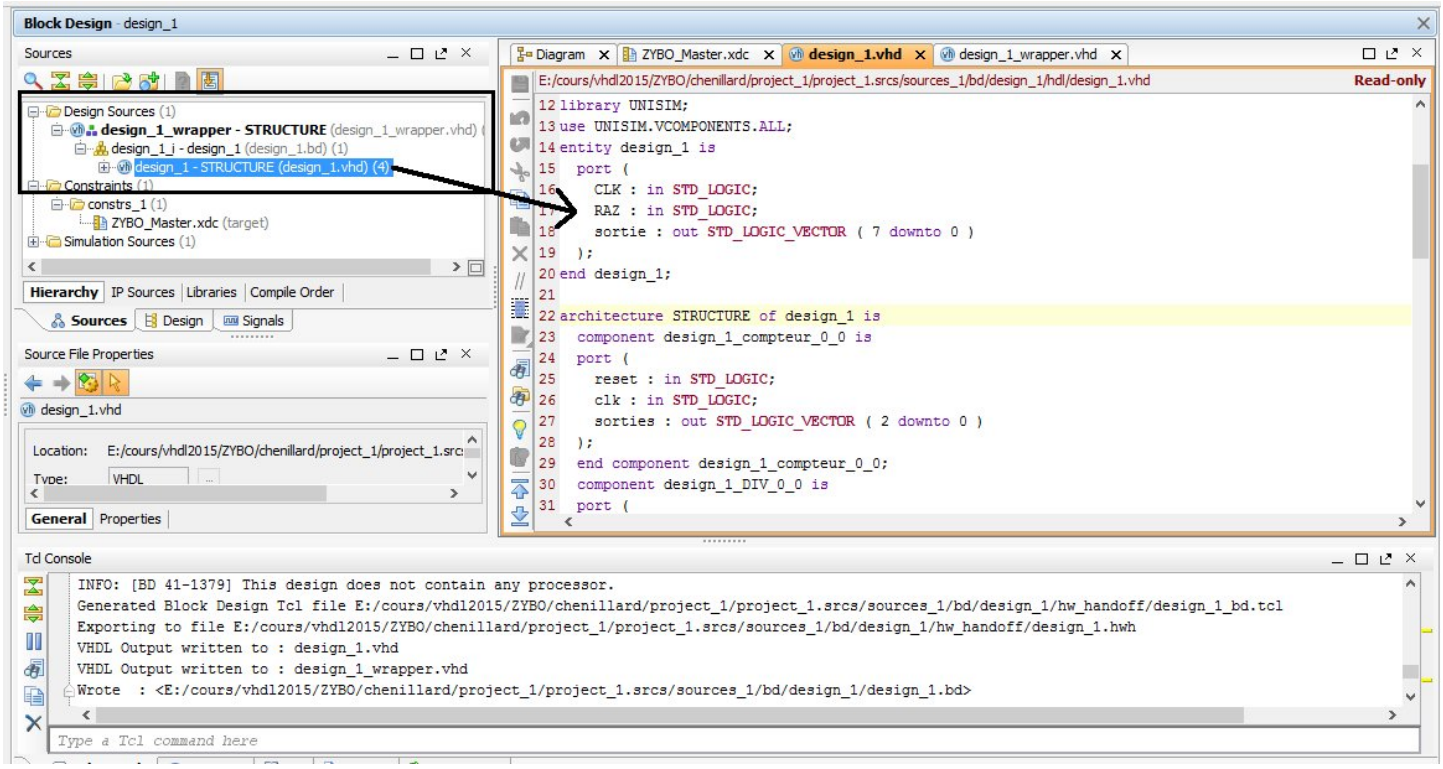


On doit créer un fichier VHDL de hiérarchie supérieure pour le block design. Cliquez droit sur le block design puis sélectionnez create HDL wrapper

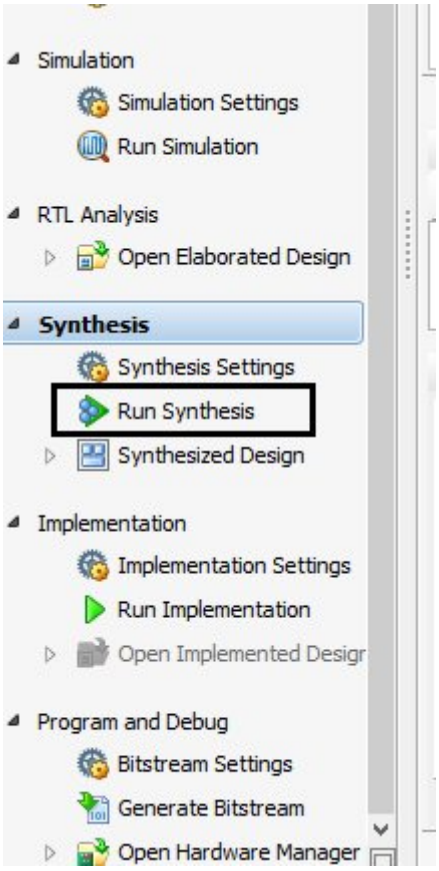


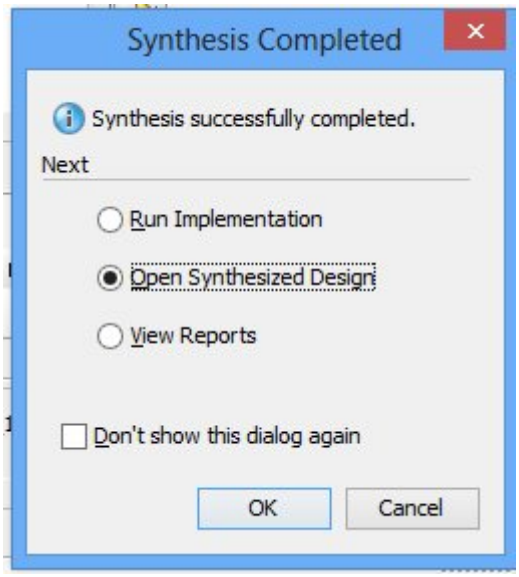


Notre fichier VHDL apparait

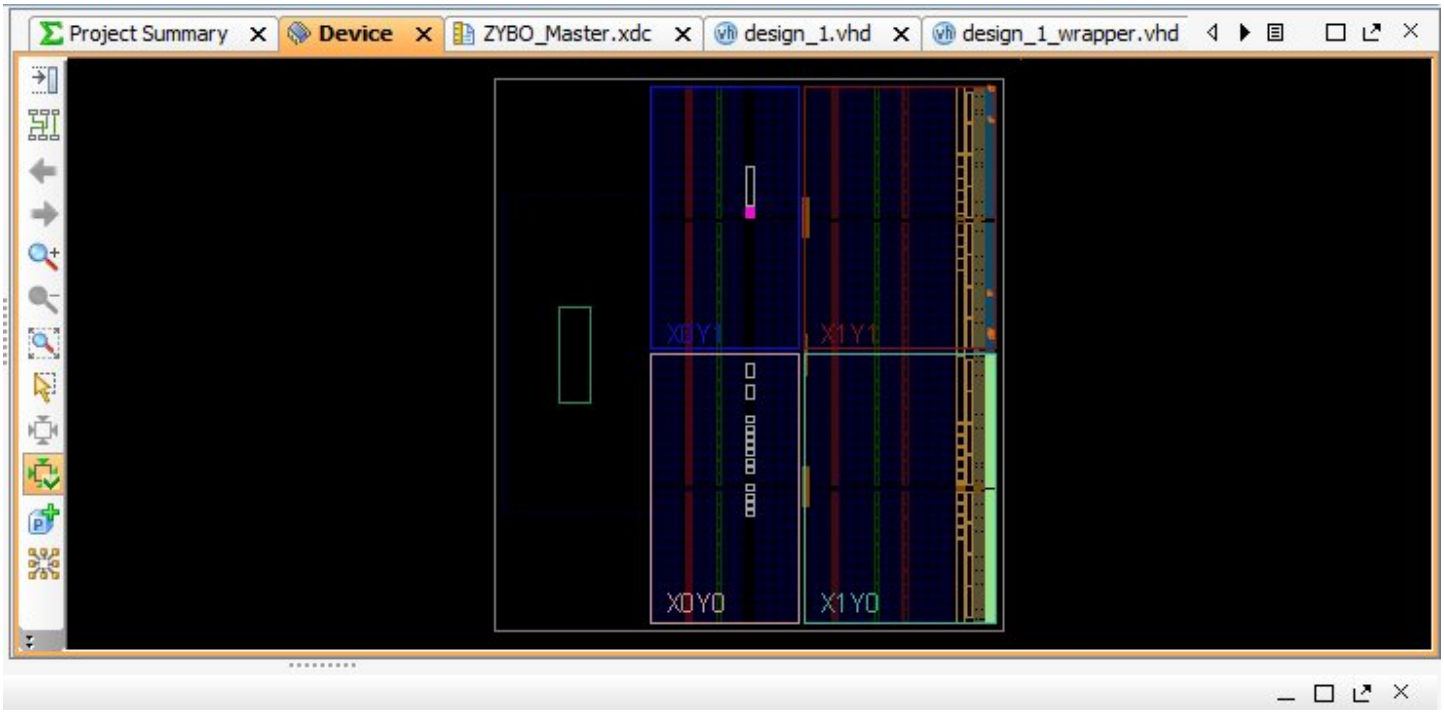


Lancez la phase de synthèse :

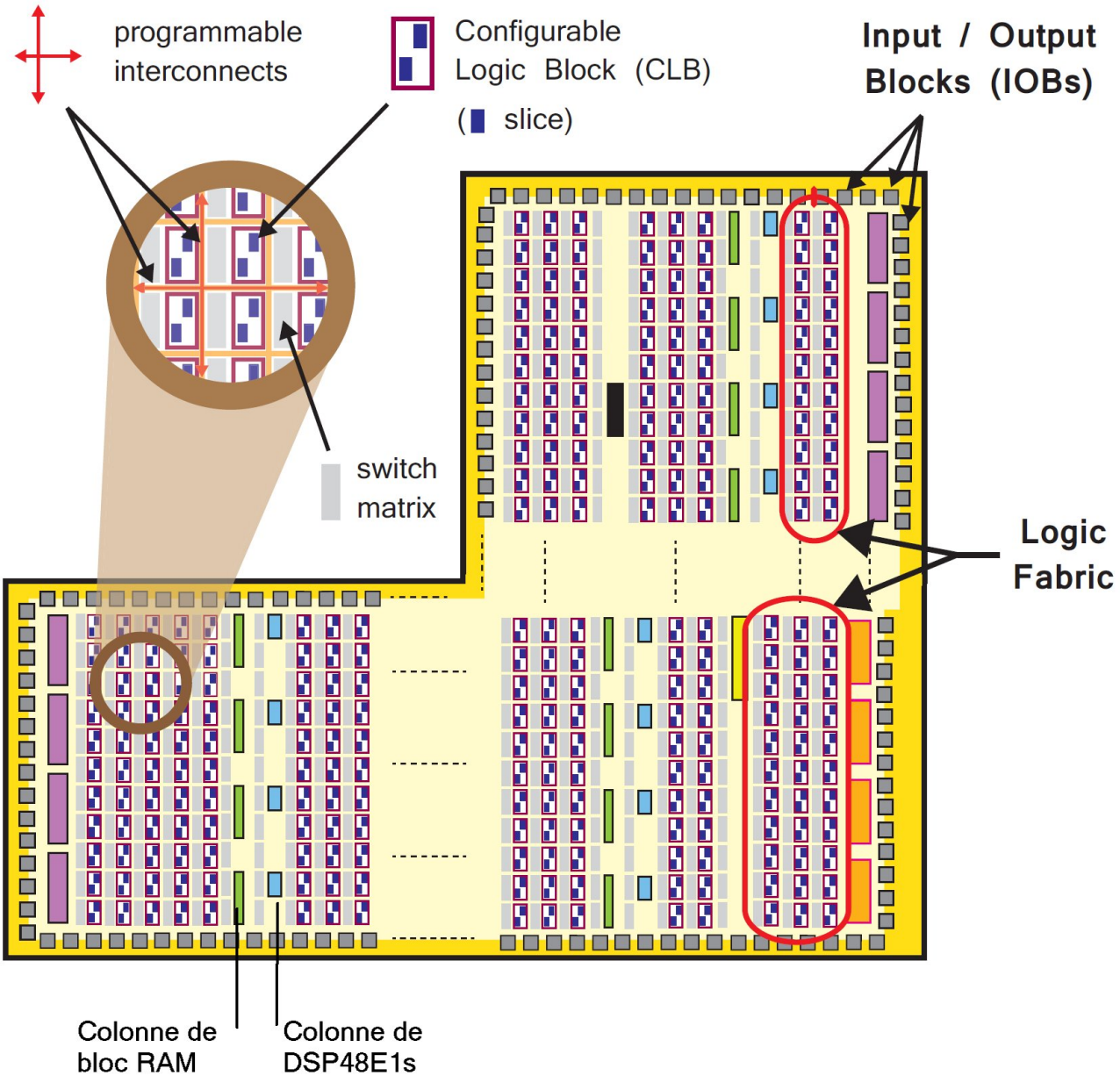




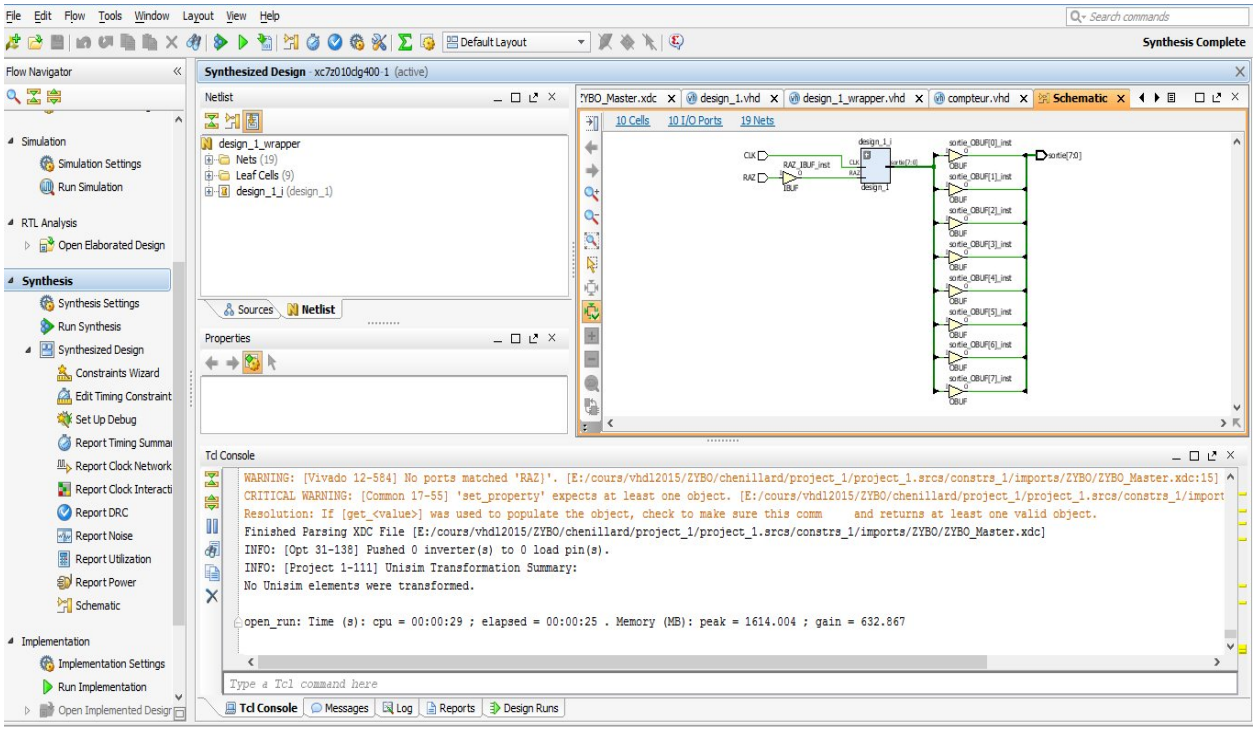
On peut visualiser l'intégration de notre chenillard au sein du FPGA :



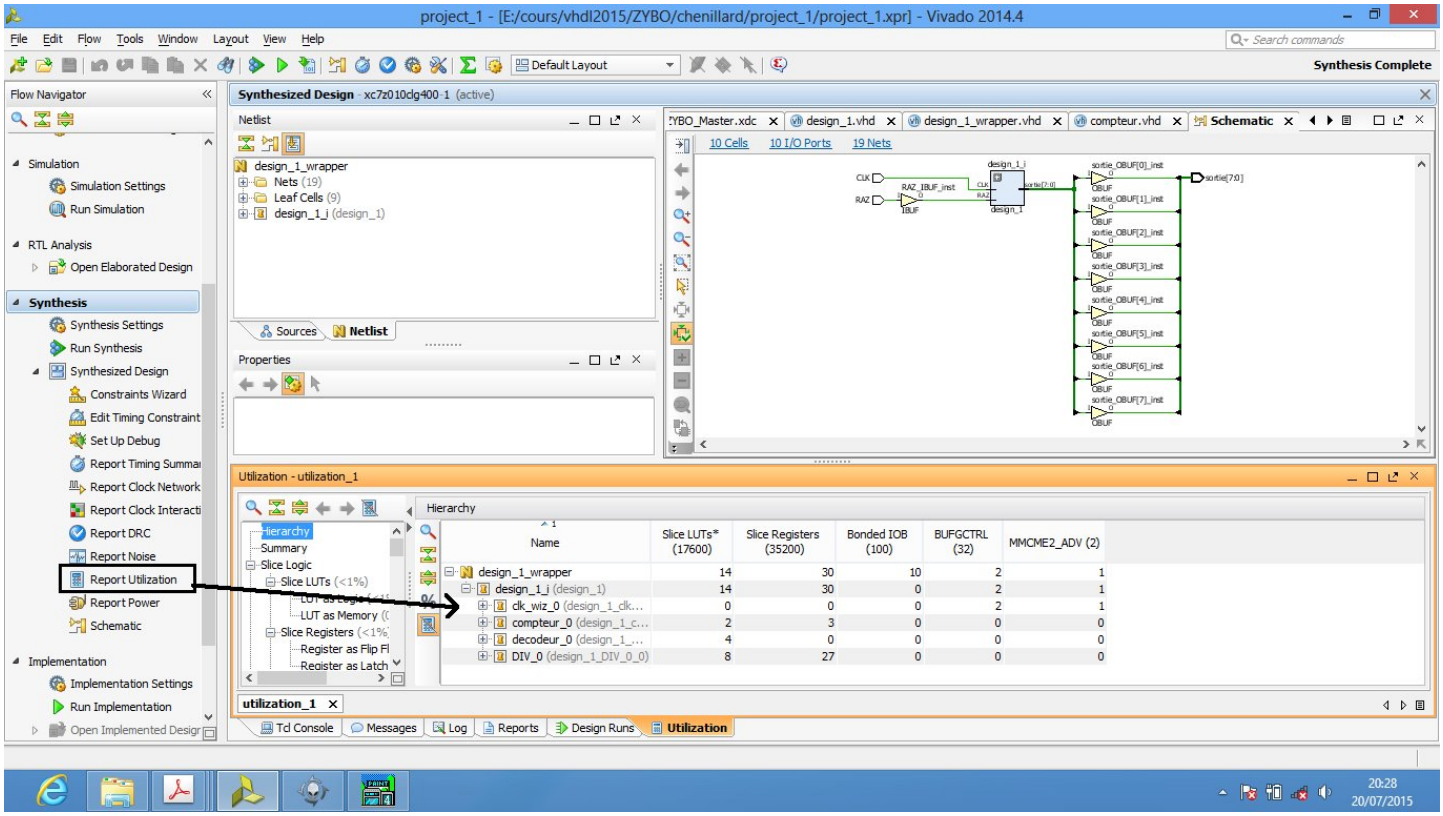
Il faut savoir que la structure du FPGA est composée des éléments suivants :



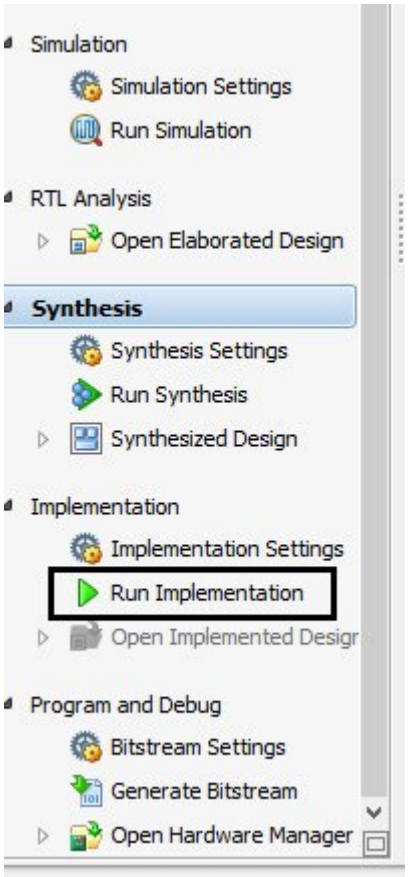
En cliquant sur schematic on fait apparaitre les principaux éléments du FPGA qui ont été sélectionné pour répondre à cette application

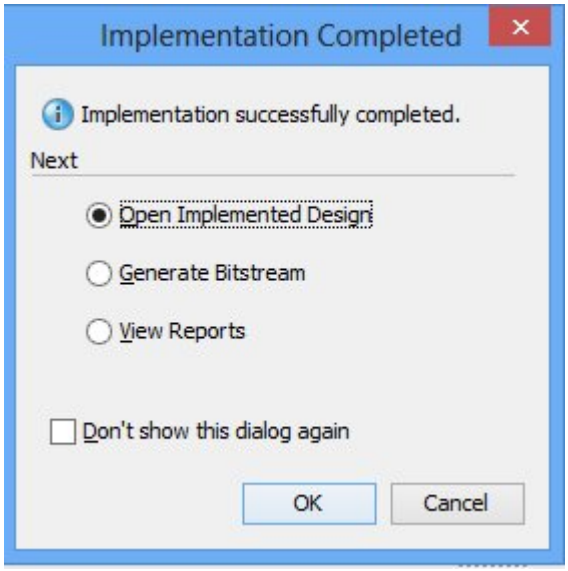


En cliquant sur report utilisation on fait apparaitre les ressources monopolisées afin de répondre à l'application.

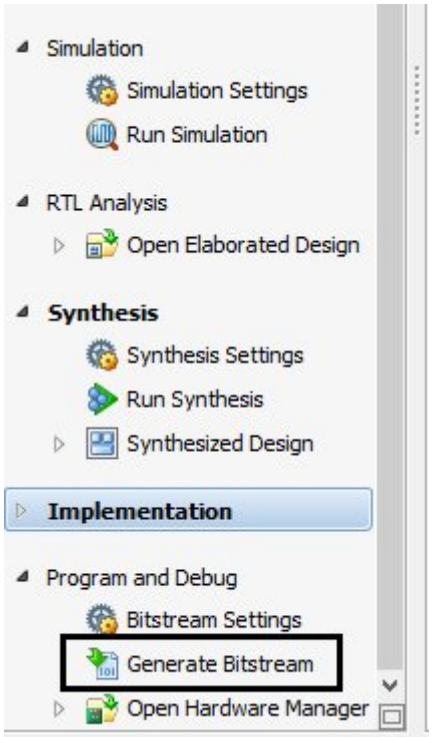


Commençons l'implémentation du FPGA

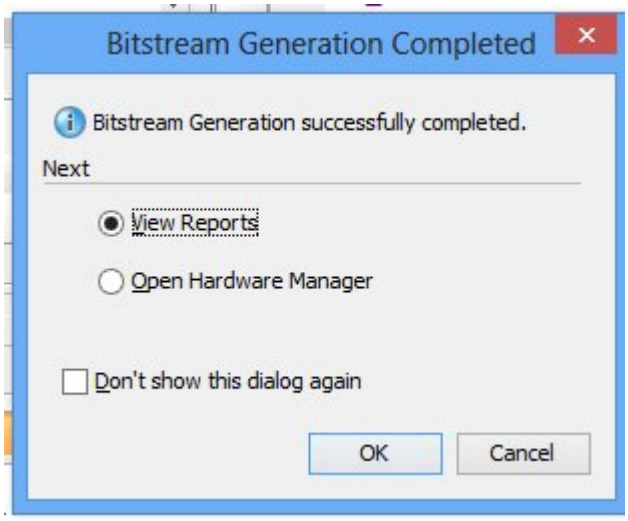




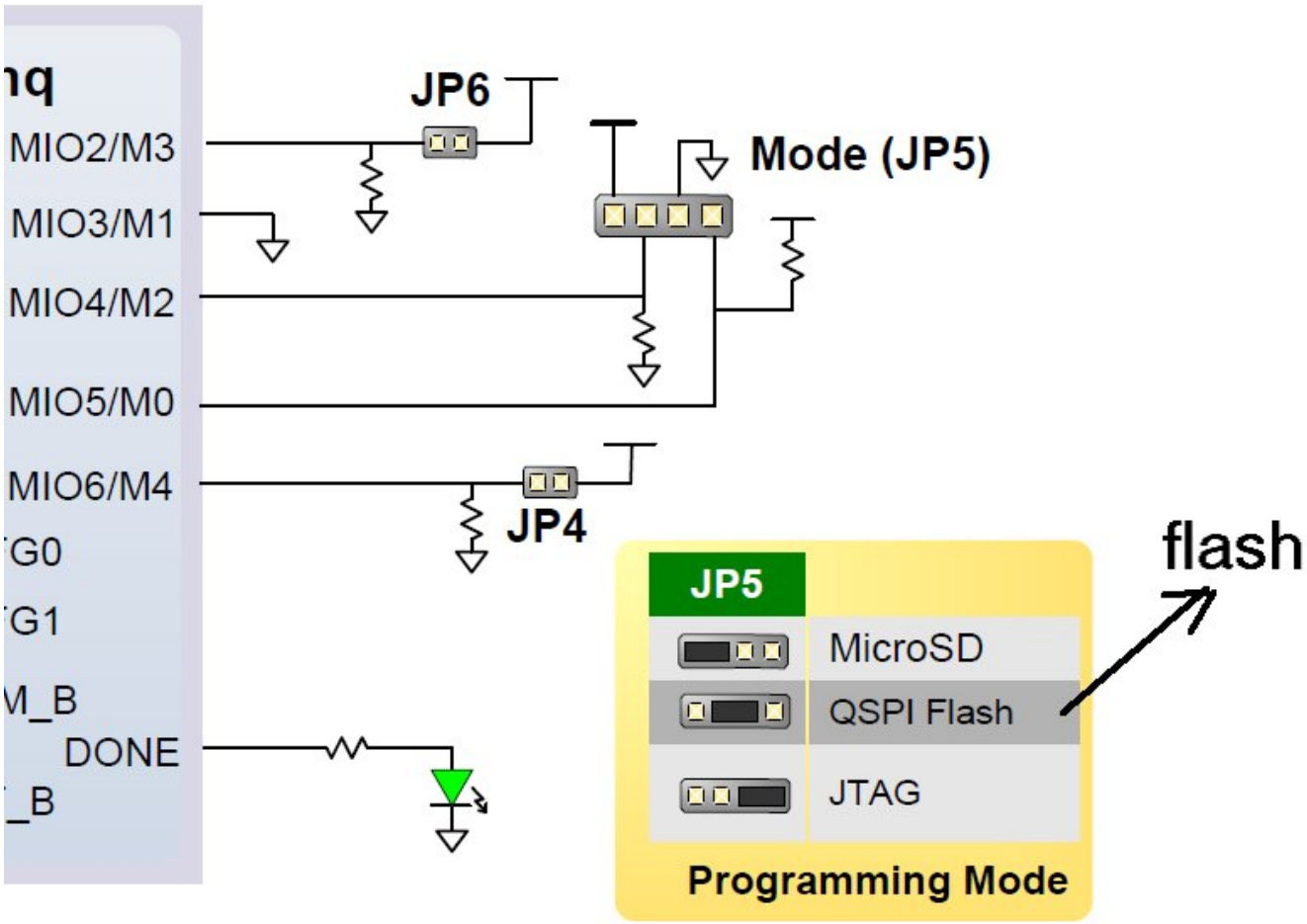
Nous reste à générer le bitstream



La fenêtre de succès

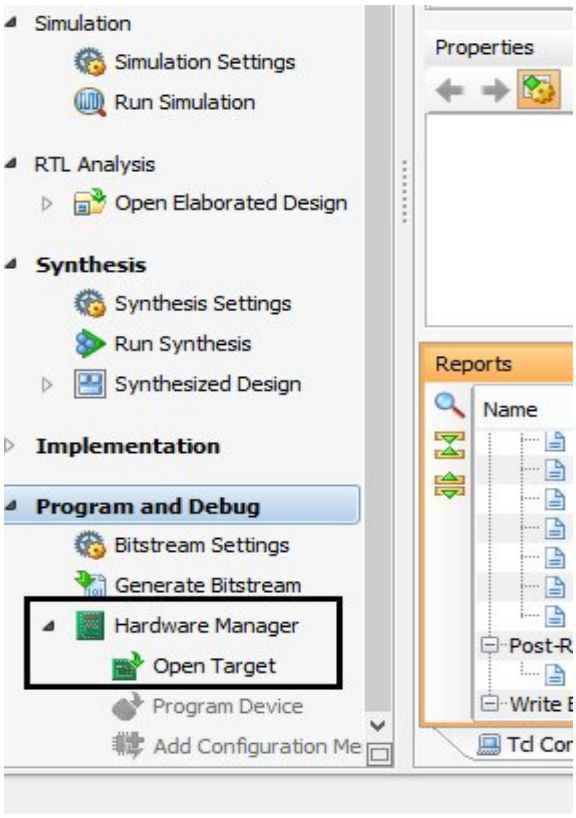


On va programmer notre FPGA par la suite en utilisant le port micro-usb de la carte qu'il faudra relier au PC

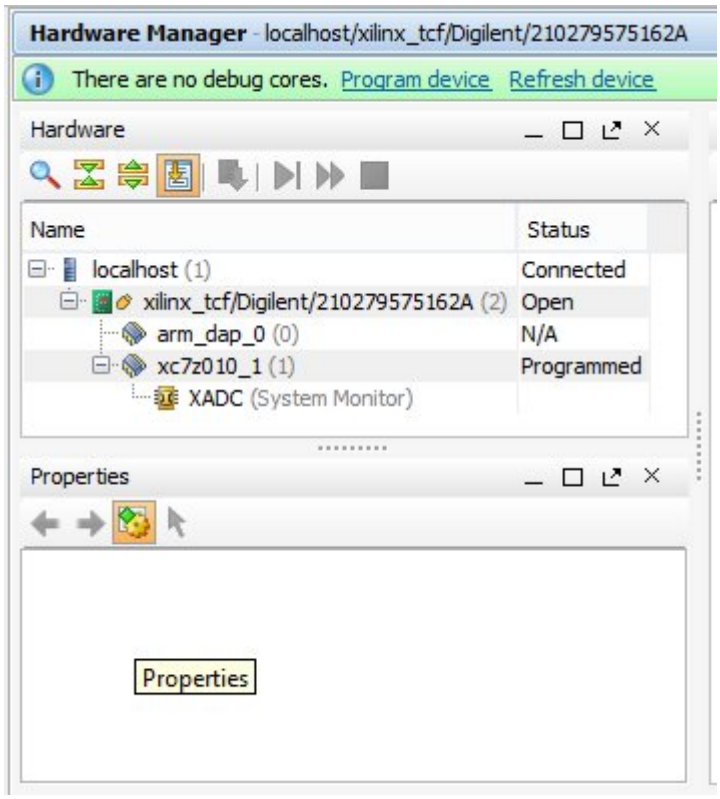


3Q configuration pins

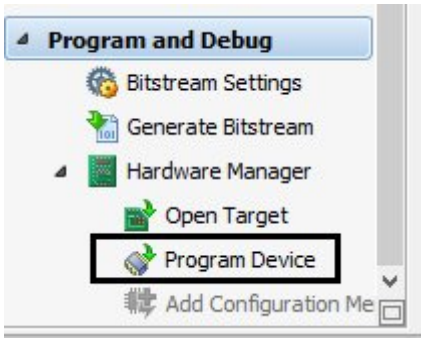
Cliquez sur open target



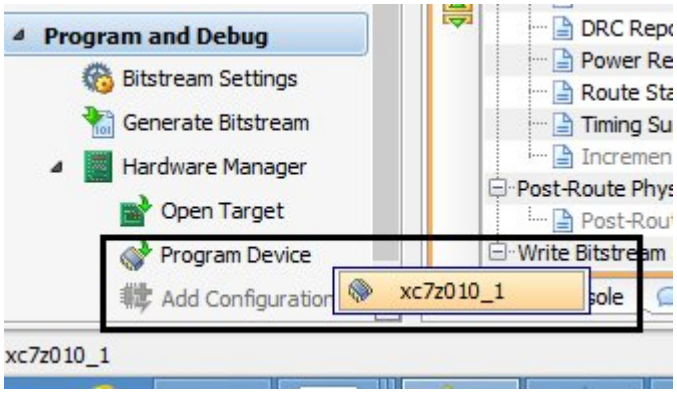
La présence du SOC est indiquée



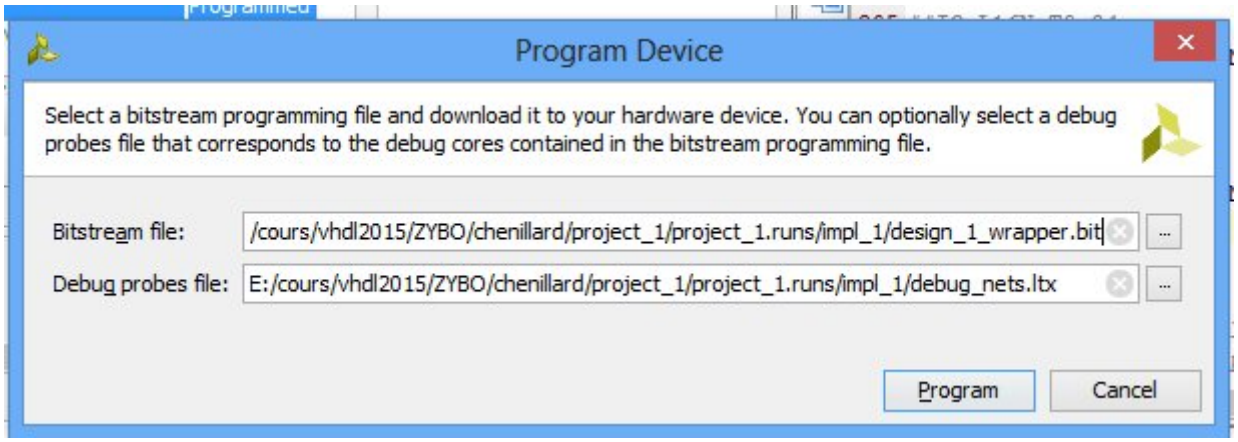
Cliquez sur program device



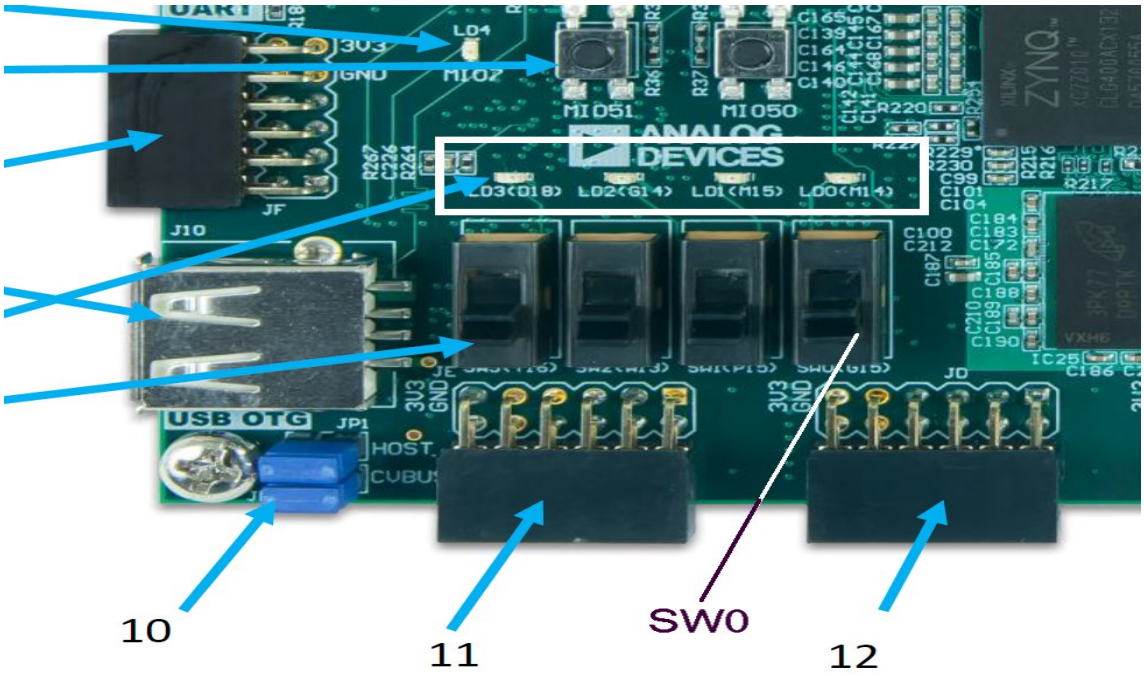
Et sélectionnez XC7010_1



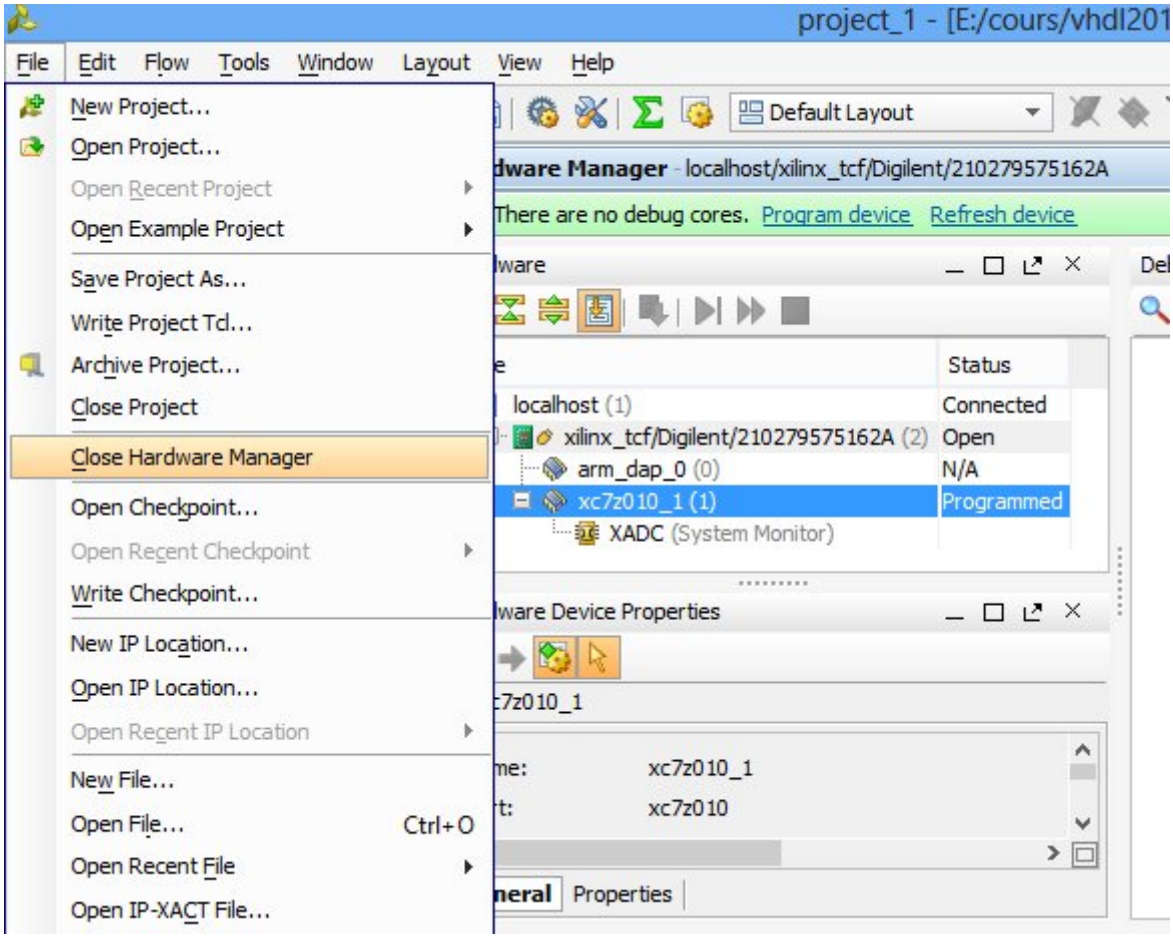
Par défaut le fichier bitstream est sélectionné



Le FPGA est programmé et l'application tourne



Pour retourner sur les sources on peut refermer le manageur de matériel



Compteur .vhd

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

entity compteur is
   Port ( reset : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          sorties : out  STD_LOGIC_VECTOR (2 downto 0));
end compteur;

architecture archcompteur of compteur is
signal counter : integer range 0 to 7;
begin
compteur: process (reset, clk)
begin
if (reset='1') then counter <= 0;
elsif ( clk'event and clk='1') then counter <= counter+1;
end if;
end process compteur;
sorties <= std_logic_vector(to_unsigned(counter,3));
end archcompteur;
```

Div .vhd

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


entity DIV is
   Port ( CLKDIV : in  STD_LOGIC;
          CLK_1HZ : out  STD_LOGIC);
end DIV;

architecture Behavioral of DIV is
signal cnt1hz : integer range 0 to 50e6:= 0;
signal div1hz_temp : std_logic := '0';
begin
process (CLKDIV) begin
if (CLKDIV'event and CLKDIV = '1') then
-- Diviseur par 50e6
```

```
if cnt1hz >= 50e6 then
div1hz_temp <= not(div1hz_temp);
cnt1hz <= 0;
else
div1hz_temp <= div1hz_temp;
cnt1hz <= cnt1hz + 1;
end if;
end if;
CLK_1HZ <= div1hz_temp; -- horloge à 2 Hz

end process;
end Behavioral;
```

decodeur.vhd

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

entity decodeur is
   Port ( strobe : in  STD_LOGIC_VECTOR (2 downto 0);
          sorties : out  STD_LOGIC_VECTOR (7 downto 0));
end decodeur;

architecture decodarch of decodeur is

begin
with strobe select

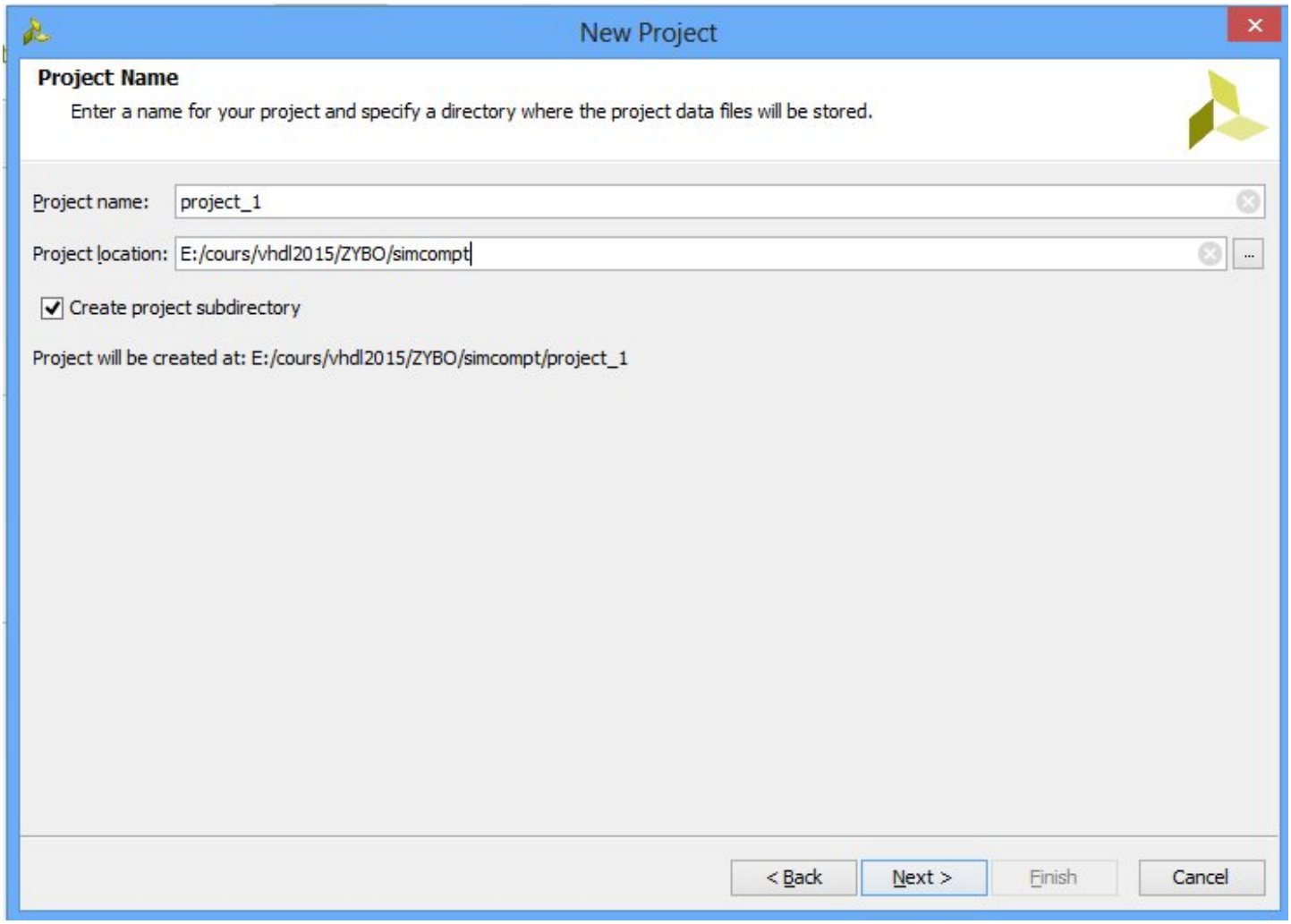
   sorties <= "00000001" when "000",
          "00000010" when "001",
          "00000100" when "010",
          "00001000" when "011",
          "00010000" when "100",
          "00100000" when "101",
          "01000000" when "110",
          "10000000" when "111",
          "00000000" when others;
end decodarch;
```

Simulation avec modelsim

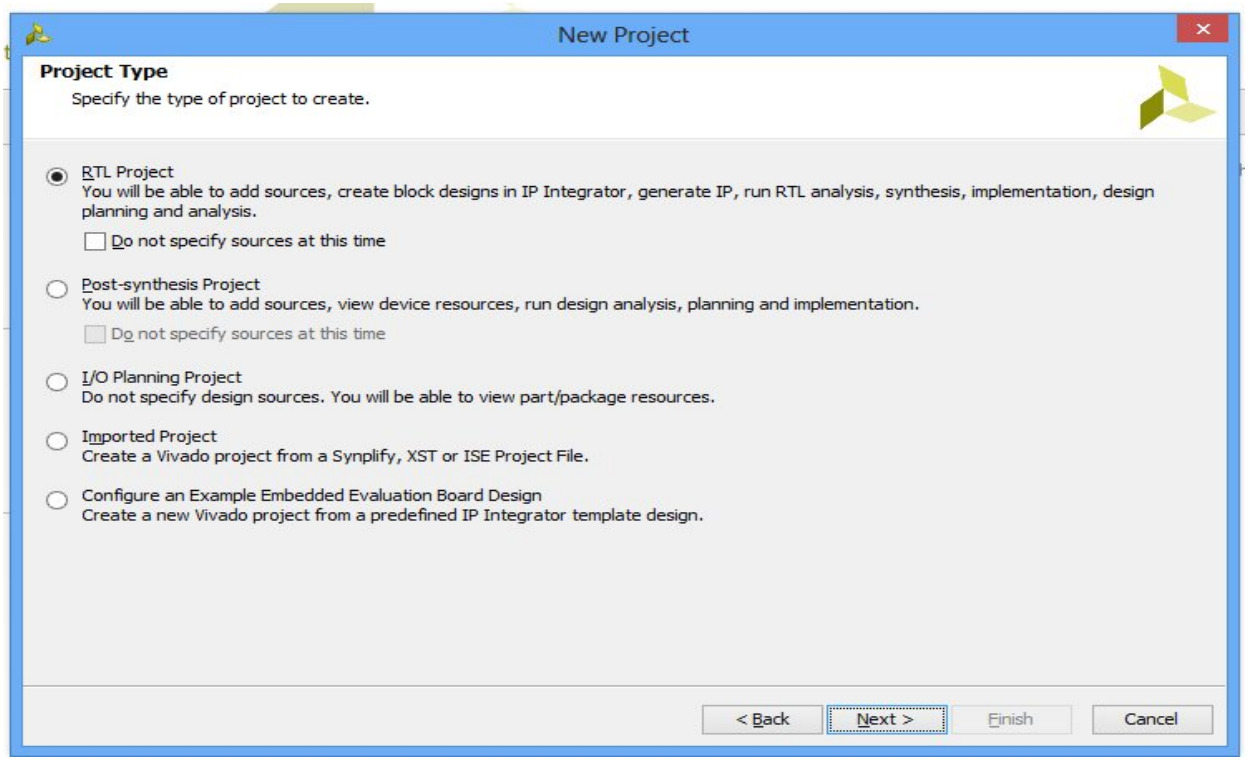
Cette séance nous permettra de s'initier aux outils de développement Vivado et notamment l'outil de simulation vivado simulation ou Modelsim.
L'exemple choisi restera anecdotique car celui-ci sera un simple compteur à modéliser en code VHDL.

Mise en place du matériel

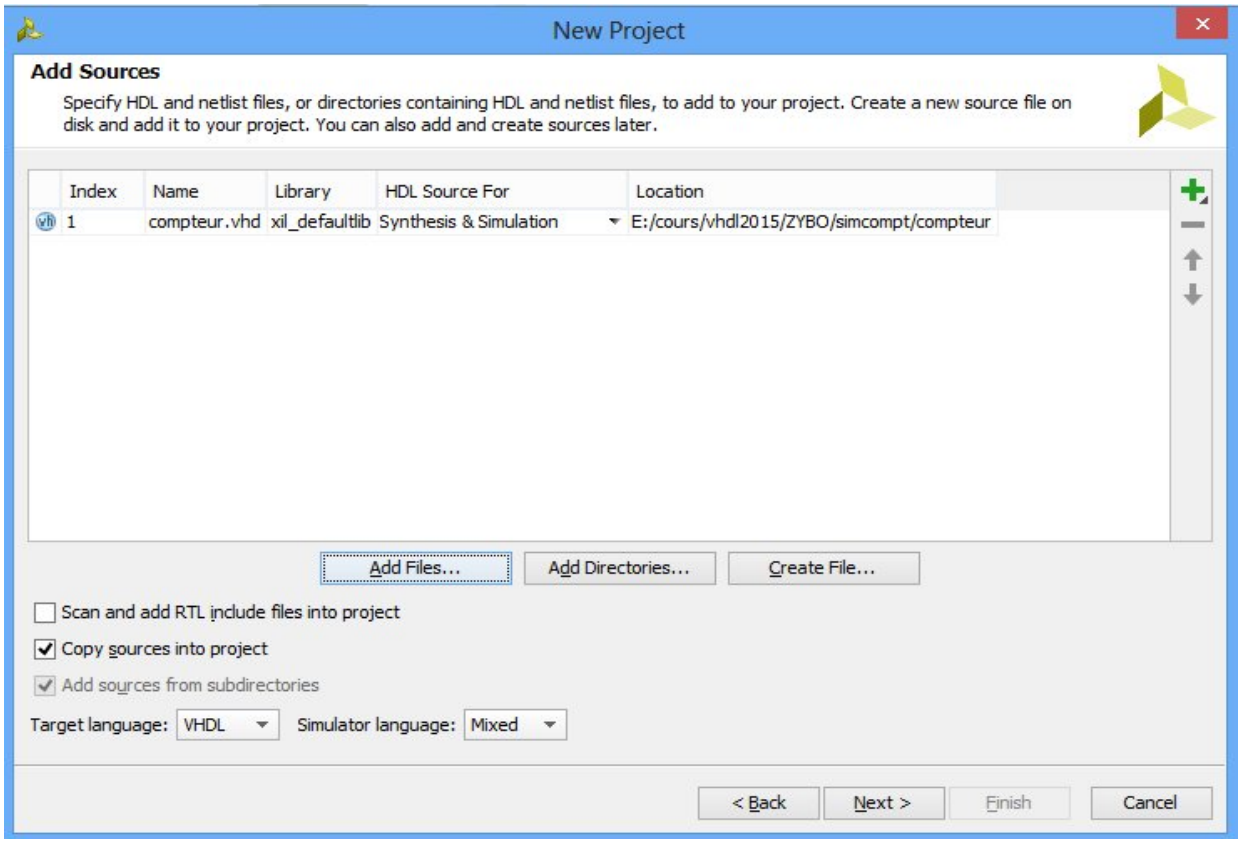
Créez un nouveau projet sur vivado



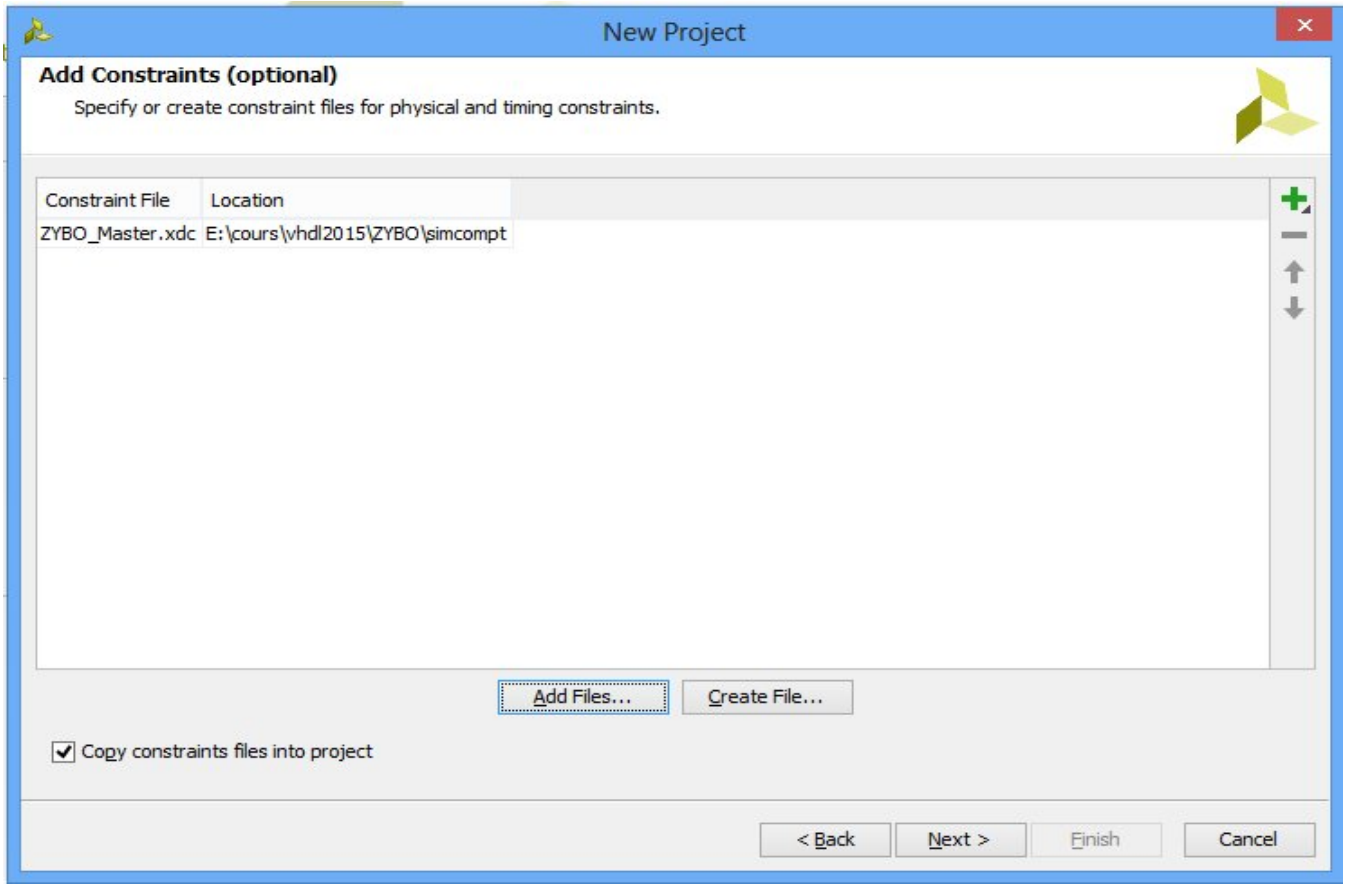
Du type RTL

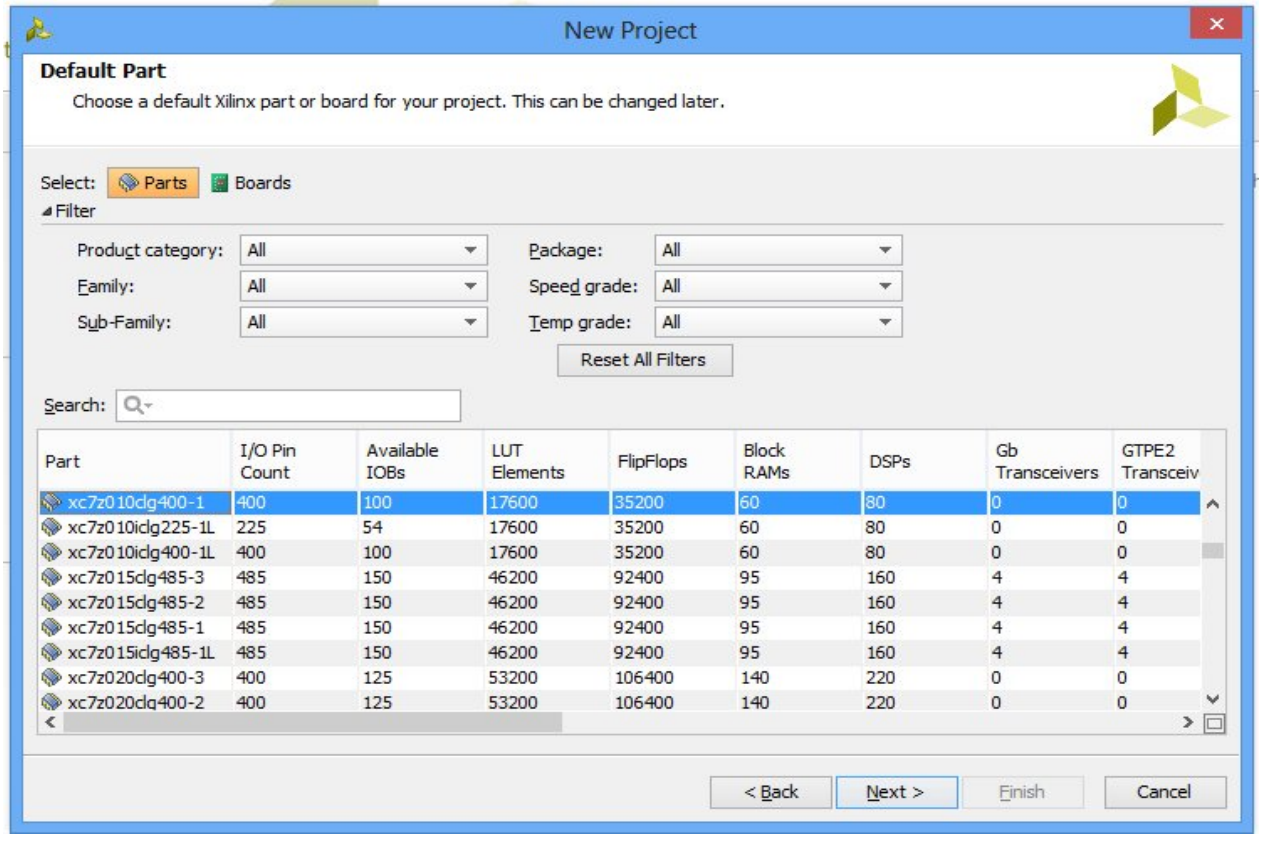


Additionnez le code VHDL de notre compteur

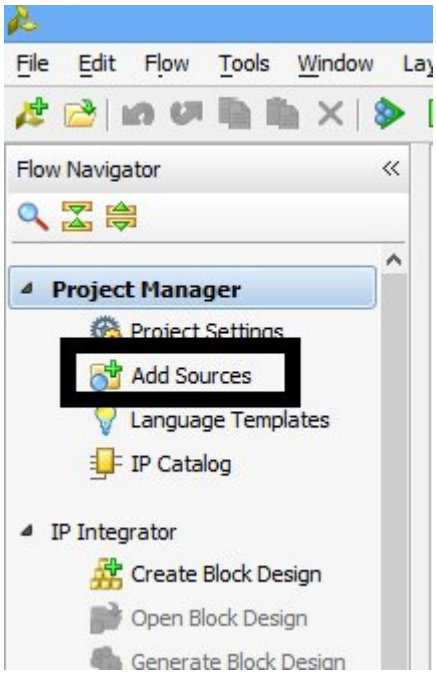


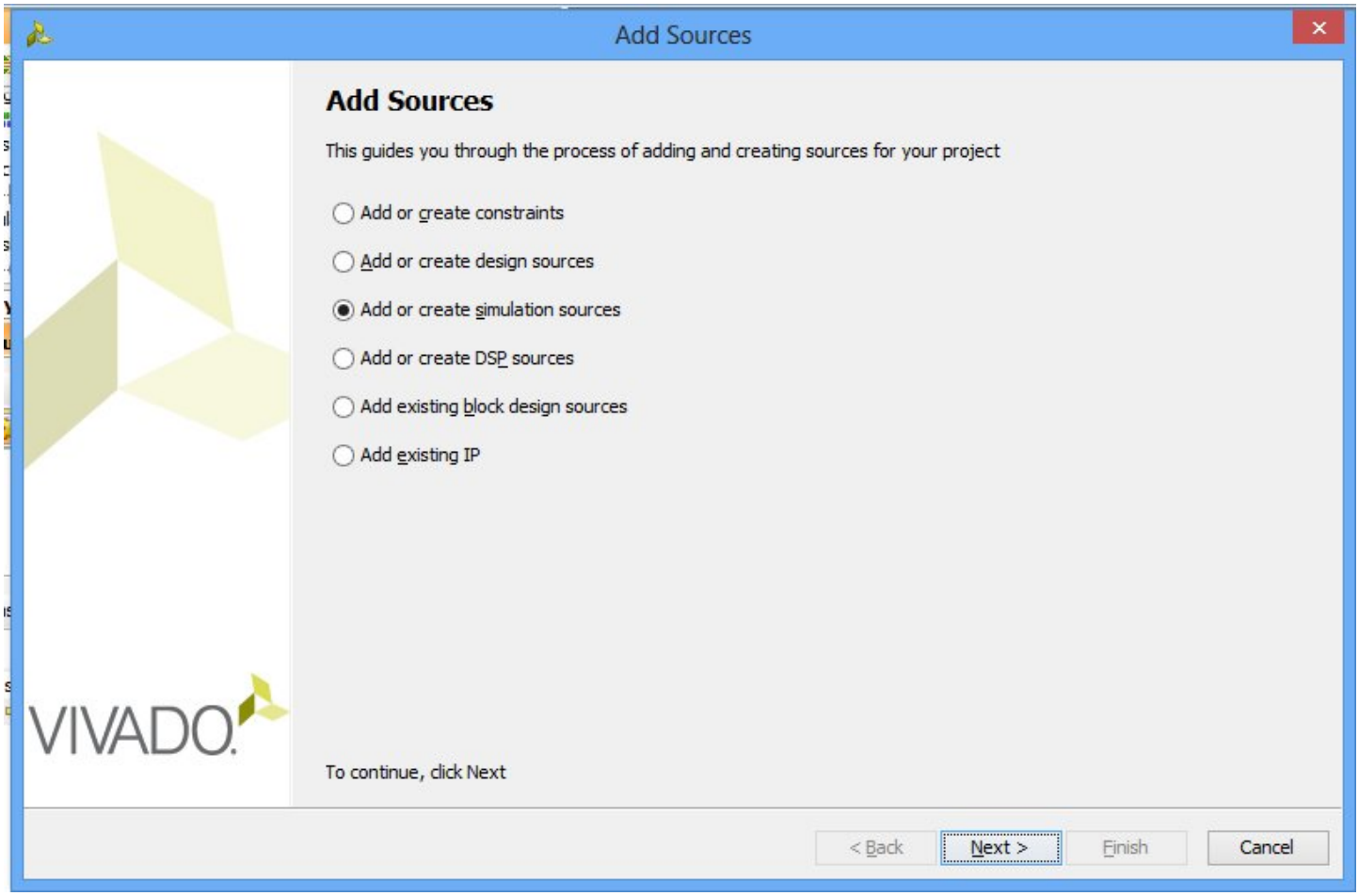
Associez le fichier de contrainte pour la carte zybo



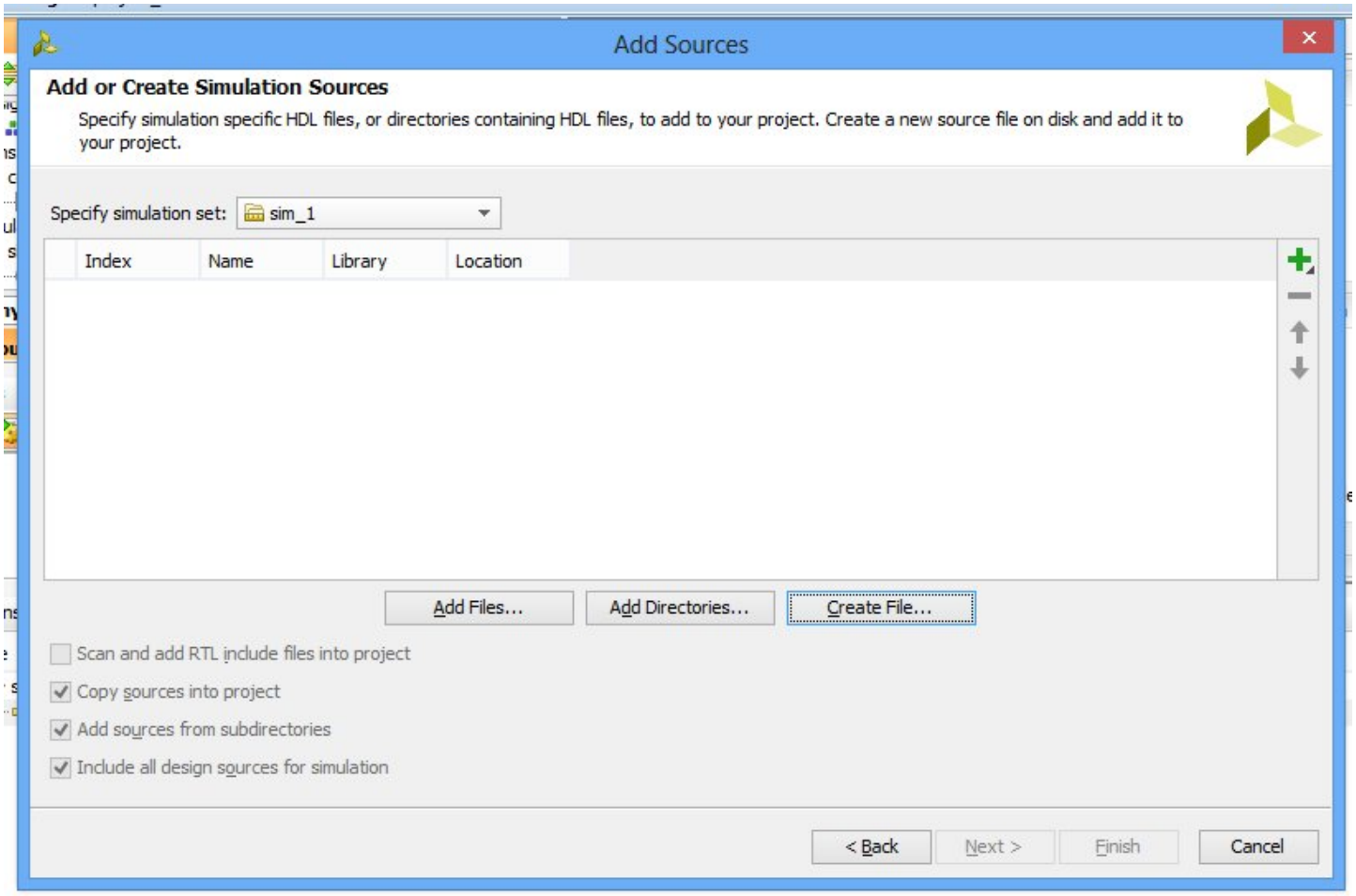


On va créer un fichier de simulation du type testbench

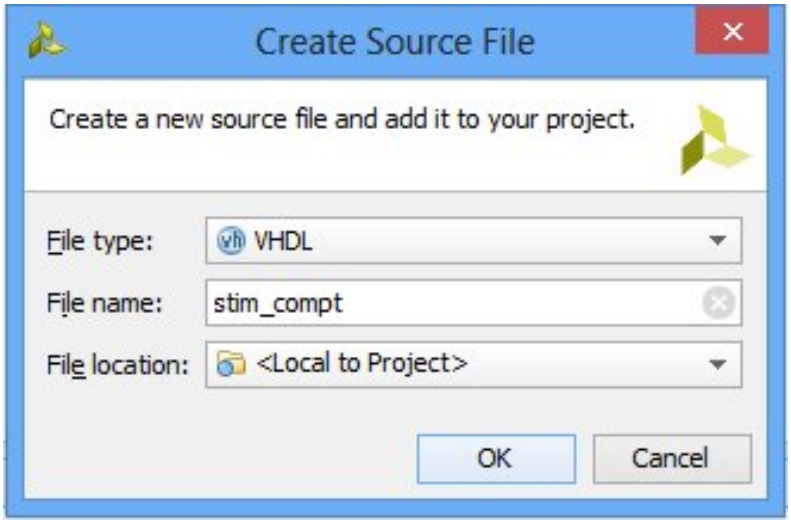


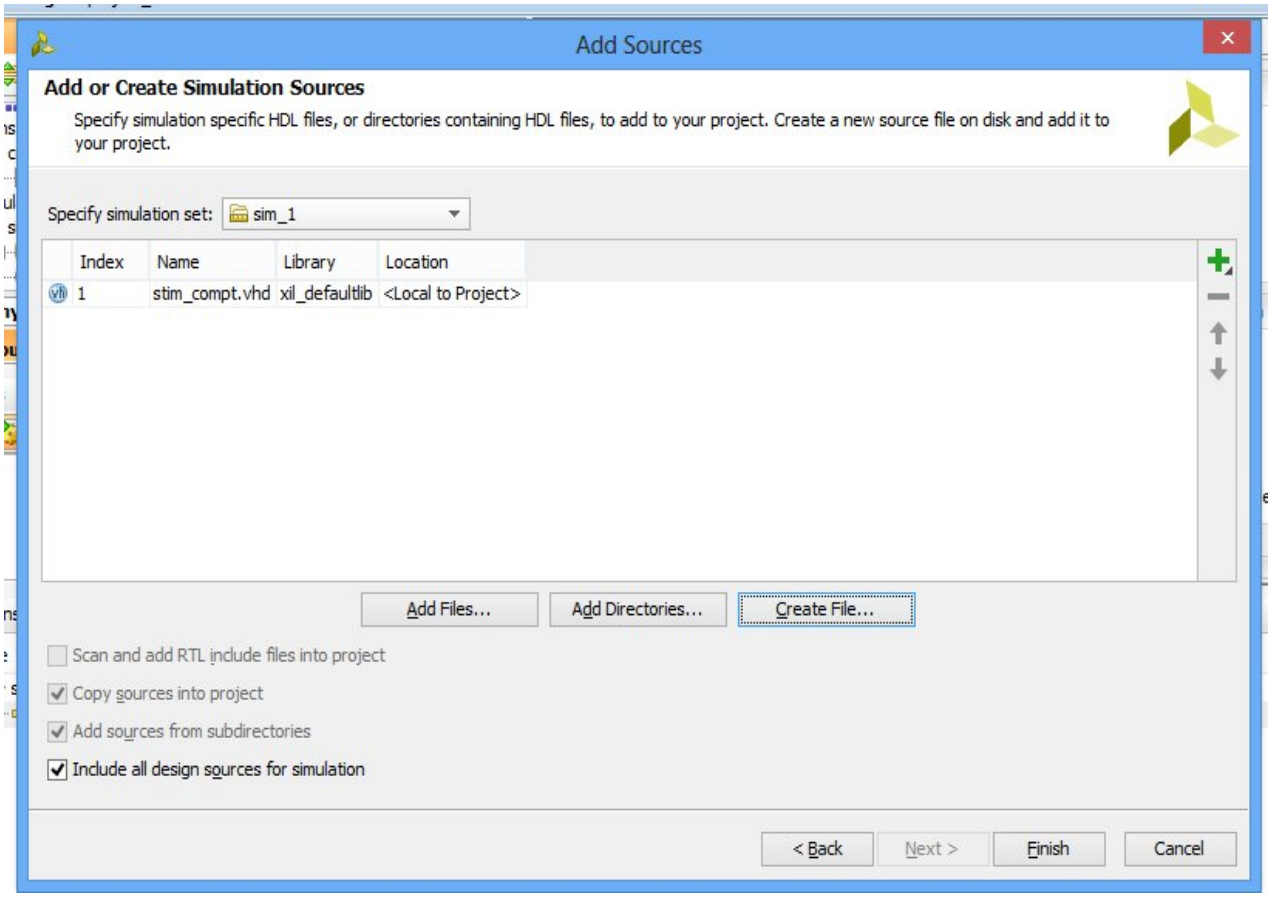


Cliquez sur create file

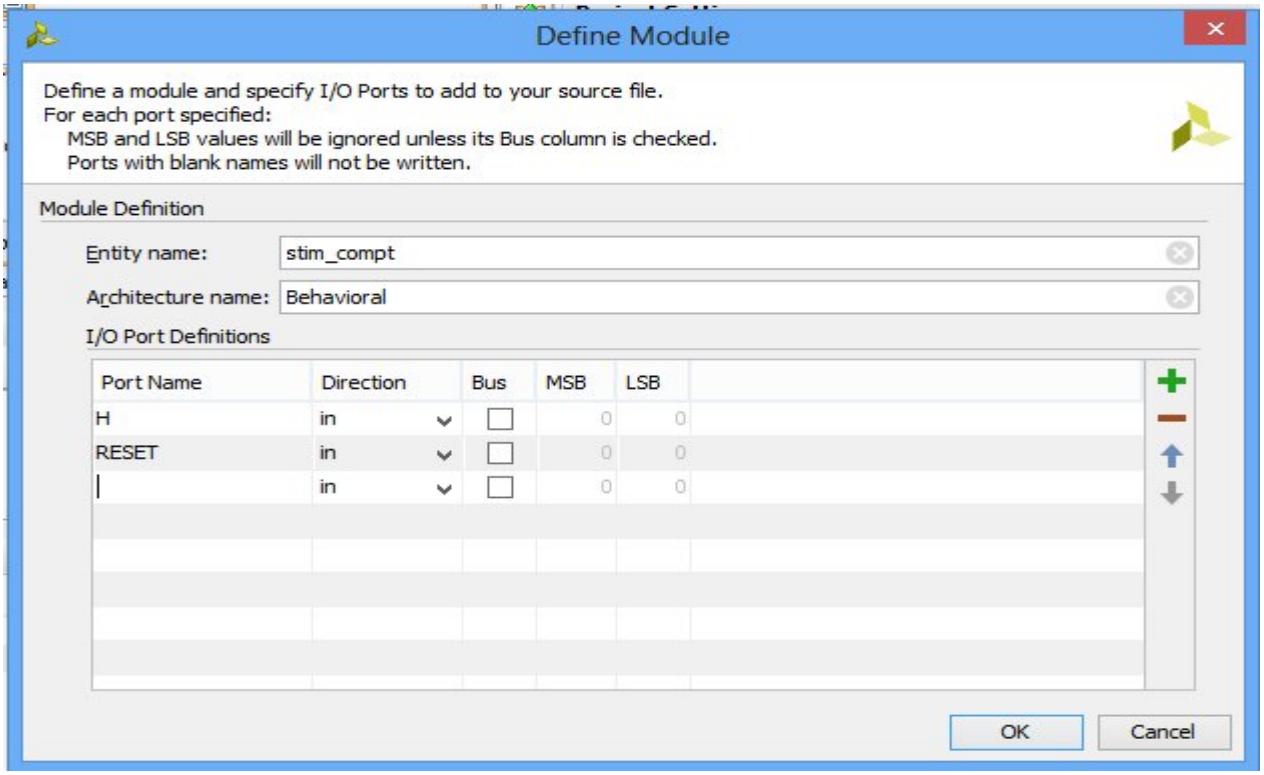


Nommez le fichier stim_compt

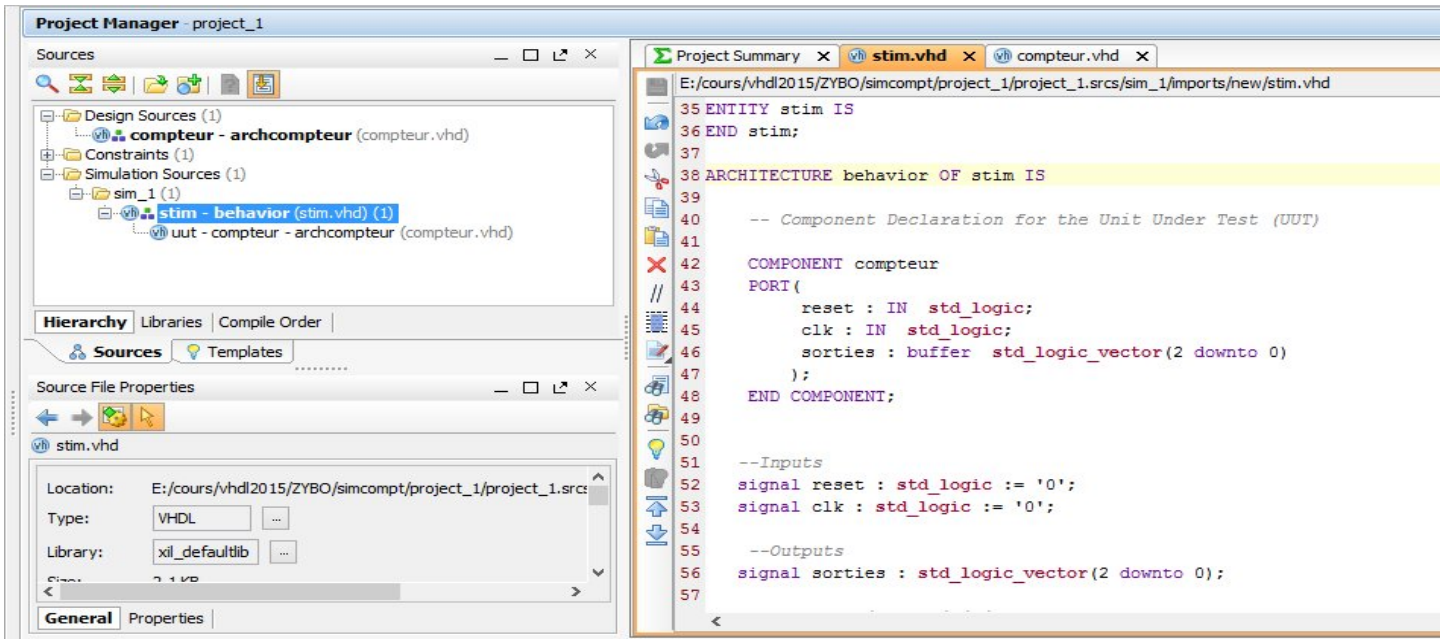




Complétez le nom des signaux pour les stimulis

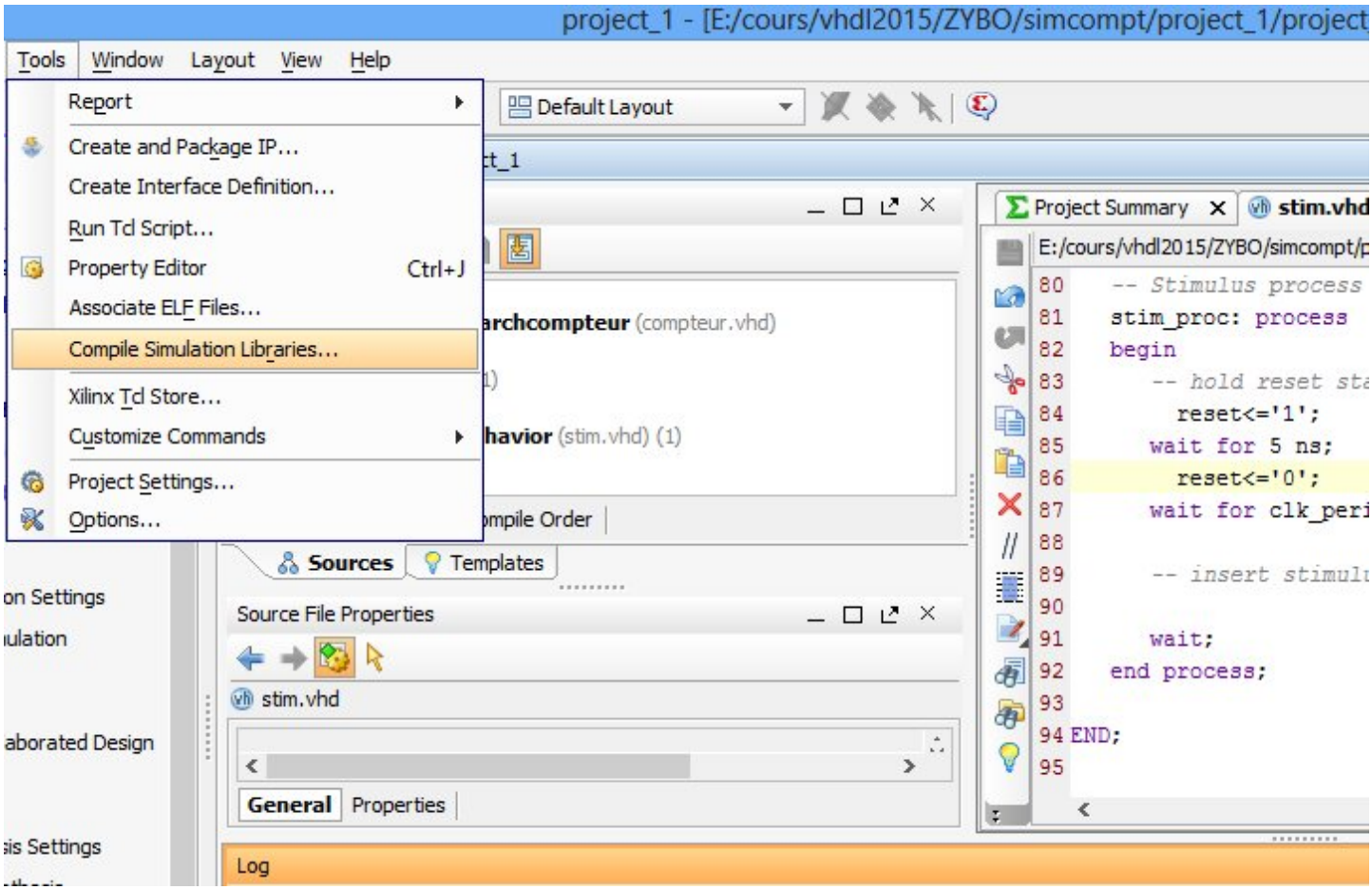


Complétez le fichier de stimuli

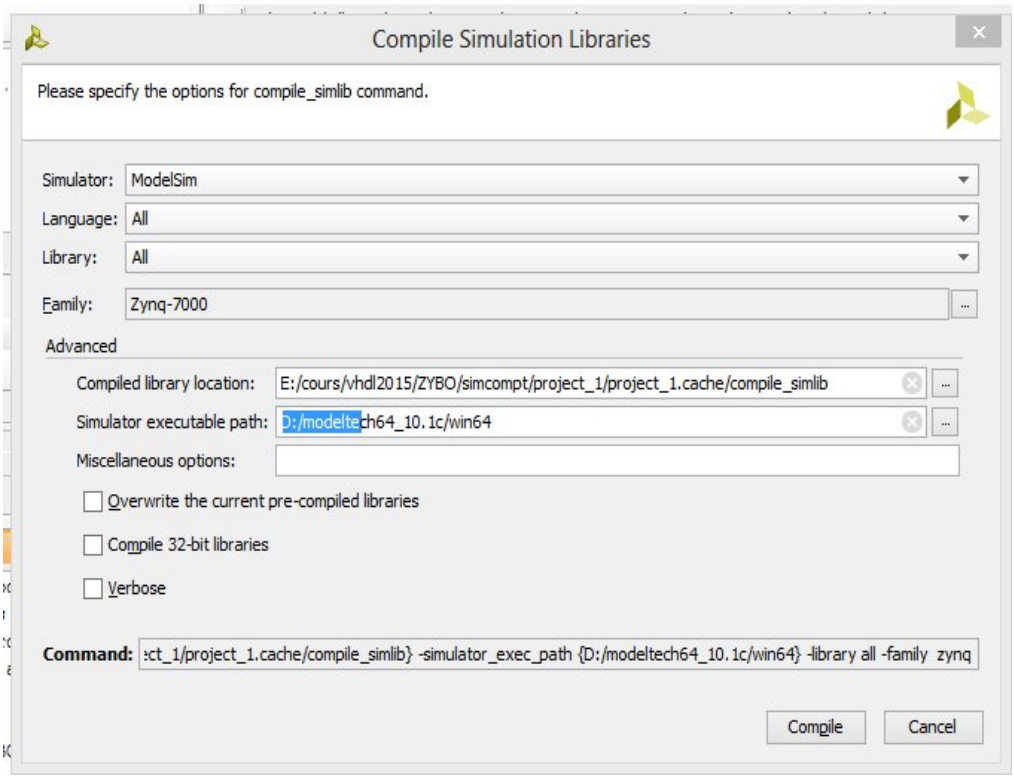


Simulation sous MODELSIM

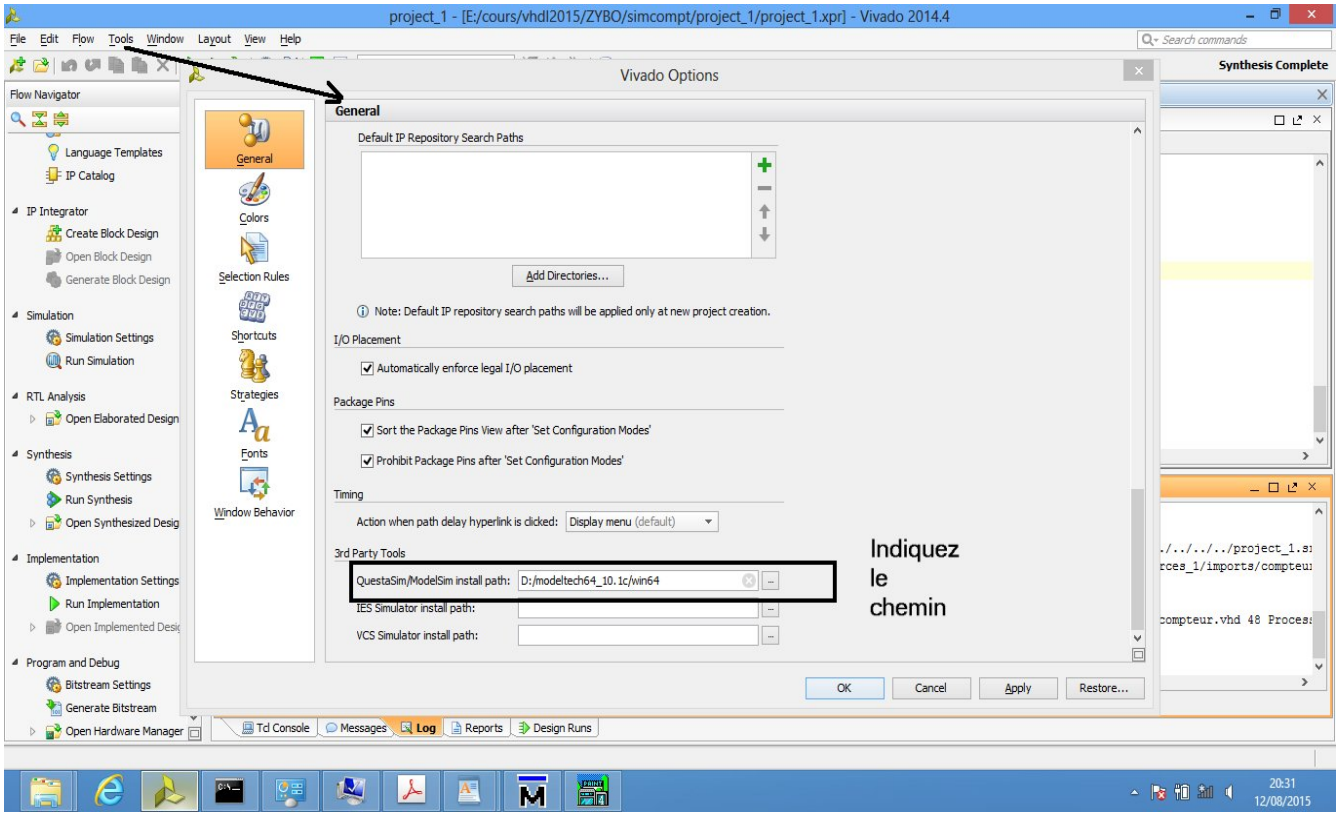
Pour le simulateur MODELSIM, quelques paramétrages sont nécessaires



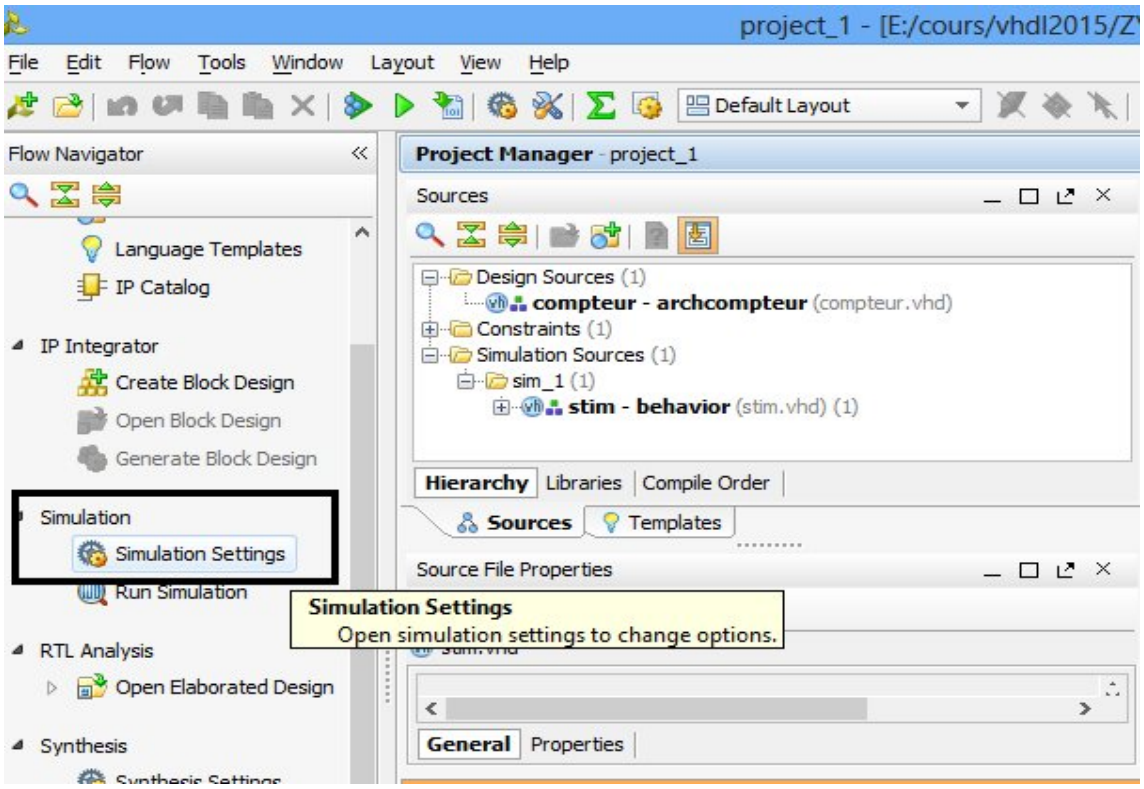
Complétez les champs de la manière suivante

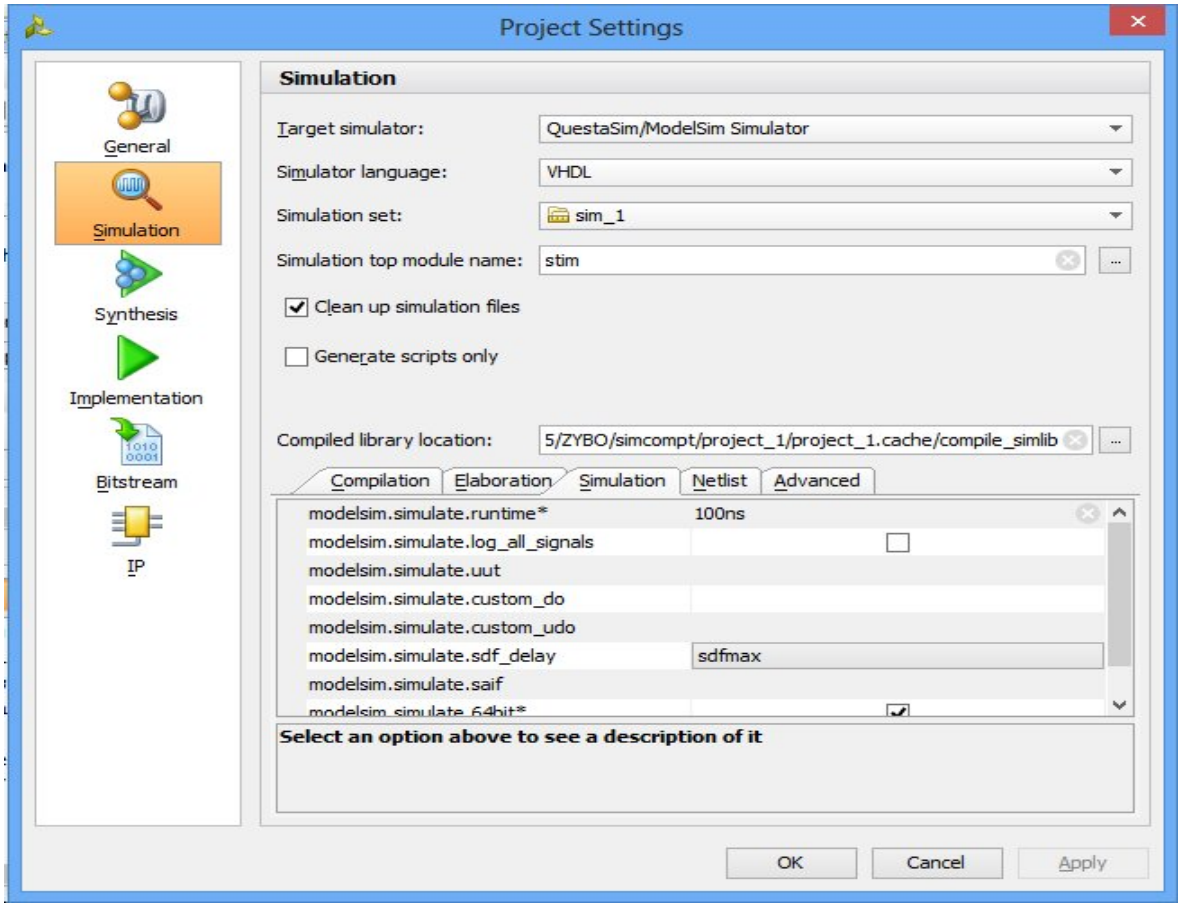


Dans tools→ options et général complétez les champs manquants

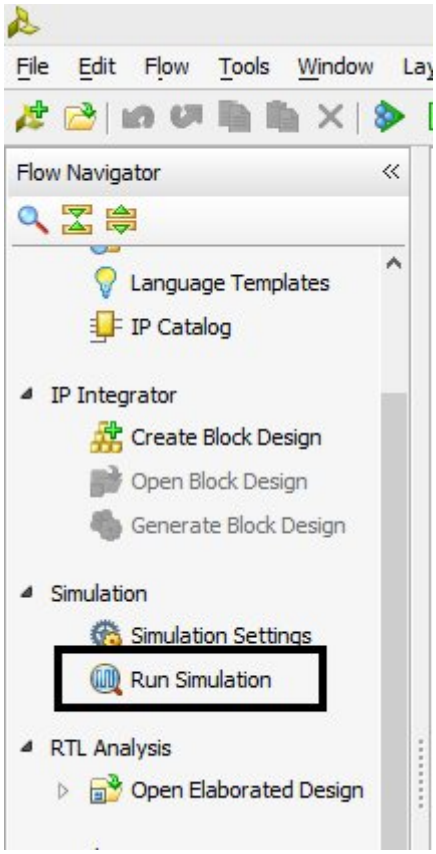


Cliquez sur simulation settings choisir les bon paramètres

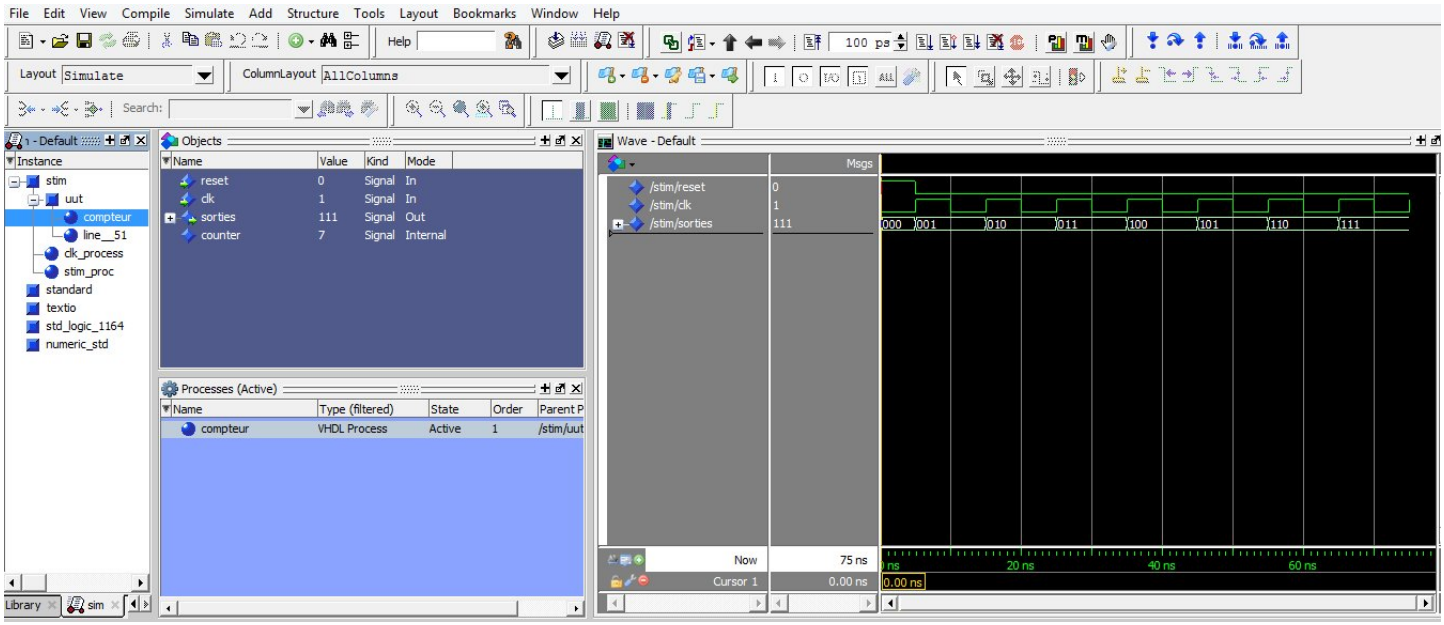




Lancez la simulation

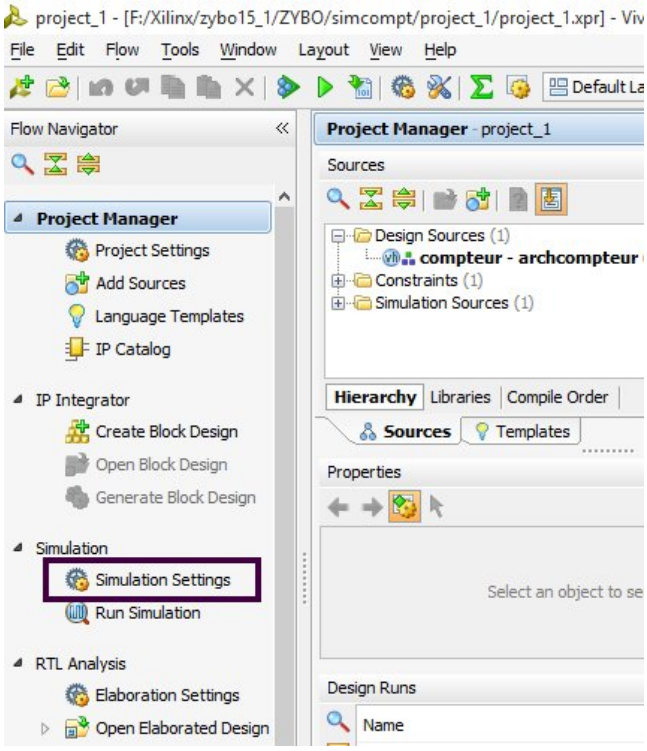


Les résultats apparaissent sous MODELSIM

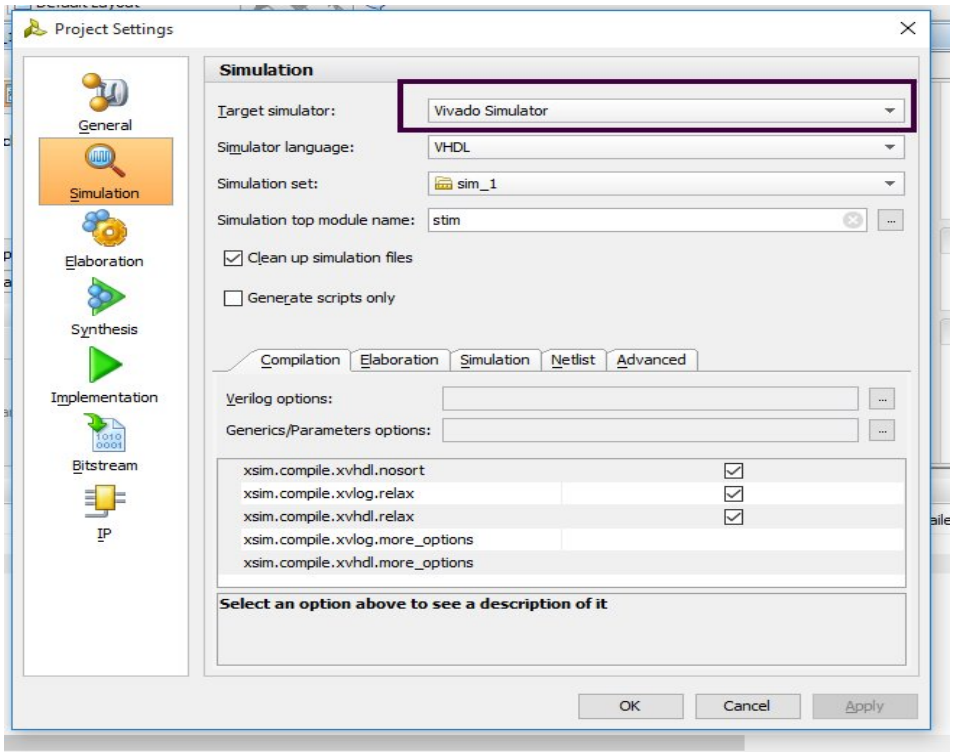


Simulation sous Vivado simulator

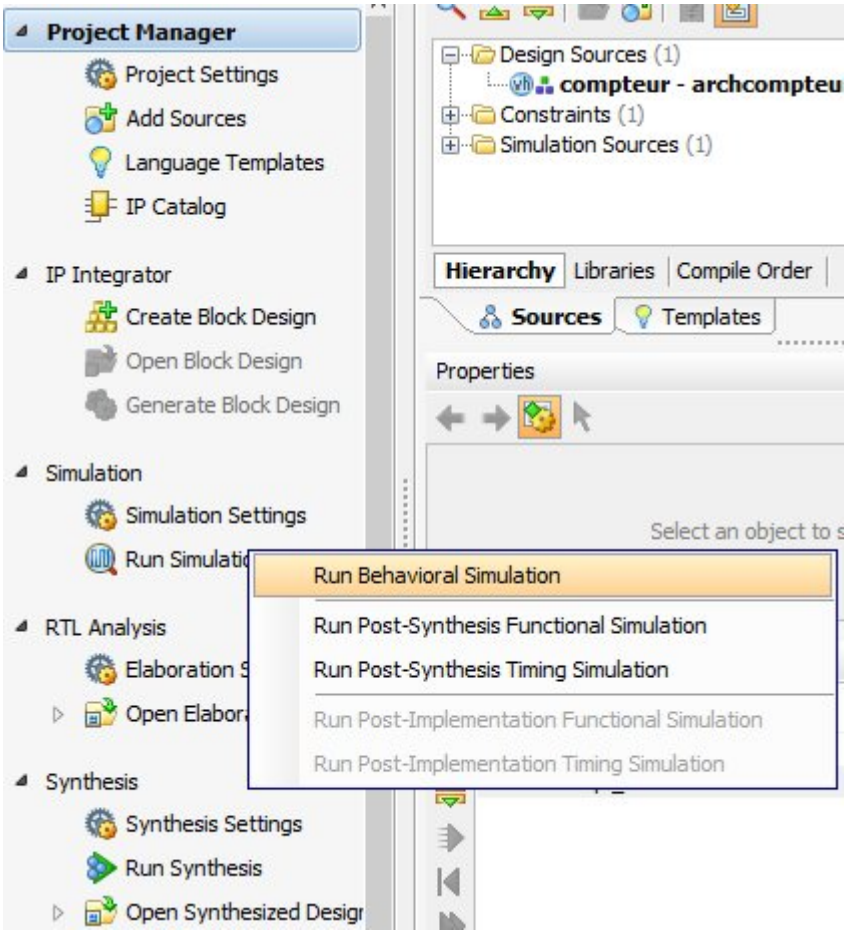
Cliquez sur simulation settings



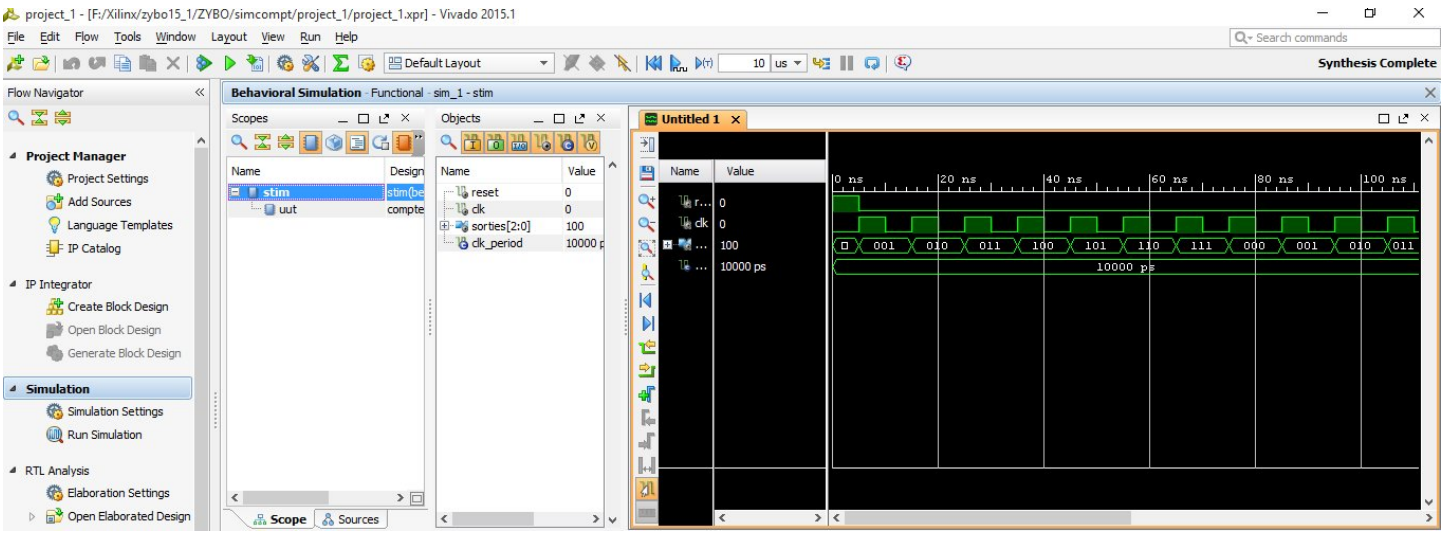
Sélectionnez le simulateur vivado



Lancez la simulation



Les résultats apparaissent sous Vivado simulator



Compteur.vhd

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


entity compteur is
   Port ( reset : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          sorties : out  STD_LOGIC_VECTOR (2 downto 0));
end compteur;

architecture archcompteur of compteur is
signal counter : integer range 0 to 7;
begin
compteur: process (reset, clk)
begin
if (reset='1') then counter <= 0;
elsif ( clk'event and clk='1') then counter <= counter+1;
end if;
end process compteur;
sorties <= std_logic_vector(to_unsigned(counter,3));
end archcompteur;
```

stim.vhd

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY stim IS
END stim;

ARCHITECTURE behavior OF stim IS

    -- Component Declaration for the Unit Under Test (UUT)



   --Inputs
   signal clk : std_logic := '0';
   signal reset : std_logic := '0';

     --Outputs
```

```
   --signal Sortie : std_logic_vector(3 downto 0);

   -- Clock period definitions
   constant CLK_period : time := 10 ns;

BEGIN

     -- Instantiate the Unit Under Test (UUT)


   -- Clock process definitions
   CLK_process :process
   begin
           clk <= '0';
           wait for CLK_period/2;
           clk <= '1';
           wait for CLK_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
           reset <= '0';
      wait for 10 ns;
                     reset <= '0';


      wait for CLK_period*40;

      -- insert stimulus here

      wait;
   end process;

END;



--end Behavioral;
```