

# Guide pratique d'utilisation de Xilinx Vivado

## 1. Introduction

### 1.1. Survol

Le logiciel Xilinx Vivado est un outil de conception de circuits pour FPGA de Xilinx. Ce logiciel permet essentiellement d'effectuer les différentes étapes de la conception et de la réalisation de circuits numériques sur FPGA. Il permet entre autres de faire la description, la simulation, la synthèse et l'implémentation d'un circuit, puis la programmation d'une puce d'une des différentes familles de FPGA de Xilinx.

### 1.2. Objectif de ce guide

L'objectif de ce guide est d'énumérer et de décrire les étapes pour :

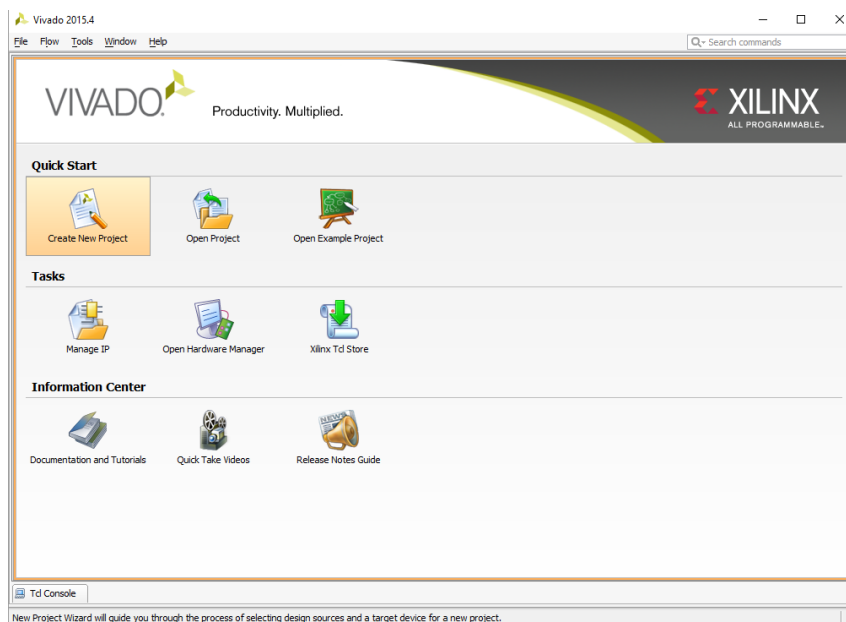
- créer un projet dans Xilinx Vivado ;
- décrire un circuit numérique à l'aide d'un code VHDL ;
- décrire un circuit numérique à l'aide d'une description hiérarchique basée sur un schéma et contenant plusieurs modules décrits en VHDL dans des fichiers séparés ;

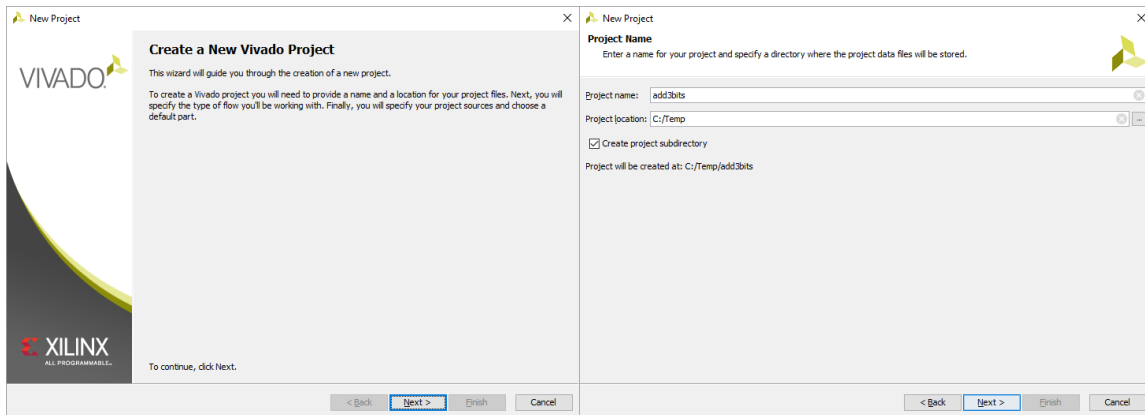
### 1.3. Lancement de Xilinx Vivado

Lancez Xilinx Vivado en choisissant la commande correspondante dans le menu **Démarrer** → **Xilinx Design Tools** → **Vivado 2015** → **Vivado 2015.x**.

## 2. Création d'un projet

Un projet regroupe plusieurs fichiers sources pour un laboratoire ou un module en particulier. Après avoir lancé le programme, cliquez sur le bouton New Project... qui se situe à gauche (ou faites File > New Project...).



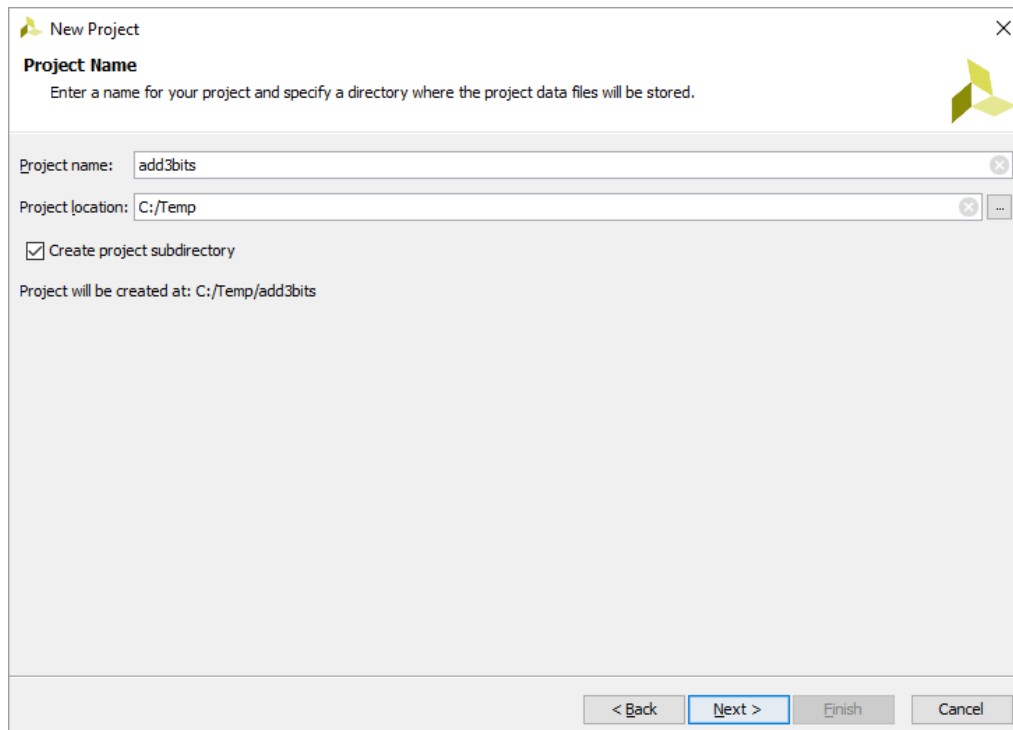


Cliquez sur **Next** pour passer à la fenêtre suivante.

Choisissez un nom représentatif pour votre projet. Il faut aussi spécifier un répertoire où le projet sera sauvegardé. Il est important que le chemin de ce répertoire ne contienne pas d'espaces, parce que certains outils invoqués peuvent ne pas les accepter. Dans tous les cas, essayer de toujours respecter cette pratique.

NOTE : Dans les laboratoires du GIGL, il est recommandé de travailler dans un répertoire personnel sur C:\temp\ votrenom. *Une fois le travail terminé, il est important d'archiver ce répertoire et d'en emporter une copie avec vous. Le répertoire C:\temp\ des ordinateurs des laboratoires est régulièrement effacé.*

Une fois le nom et le répertoire du projet choisis, cliquez sur **Next** et choisissez un **projet RTL (RTL Project)** dans la fenêtre suivante.



Vous devez remplir certains champs correspondants au composant FPGA utilisé. Cette puce est celle qui est montée sur la carte FPGA disponible au laboratoire. Reproduisez les choix de la figure suivante (pour le cours INF3500), puis cliquez sur **Next**, puis sur **Finish**.

**Product category** : General Purpose      **Speed grade** : -1  
**Family** : Artix-7      **Temp grade** : C  
**Package** : CSG324

**New Project**

**Default Part**  
Choose a default Xilinx part or board for your project. This can be changed later.

Select: **Parts** ☐ **Boards** ☐

Filter

Product category: General Purpose      Speed grade: -1  
Family: Artix-7      Temp grade: C  
Package: csg324

Reset All Filters

Search:

Part	I/O Pin Count	Block RAMs	DSPs	FlipFlops	GTPE2 Transceivers	Gb Transceivers	Available IOBs	LU Ele
xc7a15tcsg324-1	324	25	45	20800	0	0	210	10
xc7a35tcsg324-1	324	50	90	41600	0	0	210	20
xc7a50tcsg324-1	324	75	120	65200	0	0	210	32
xc7a75tcsg324-1	324	105	180	94400	0	0	210	47
xc7a100tcsg324-1	324	135	240	126800	0	0	210	63

<  >

< Back    **Next >**    Finish    Cancel

### 3. Description d'un circuit numérique en VHDL

#### 3.1. Circuit en exemple

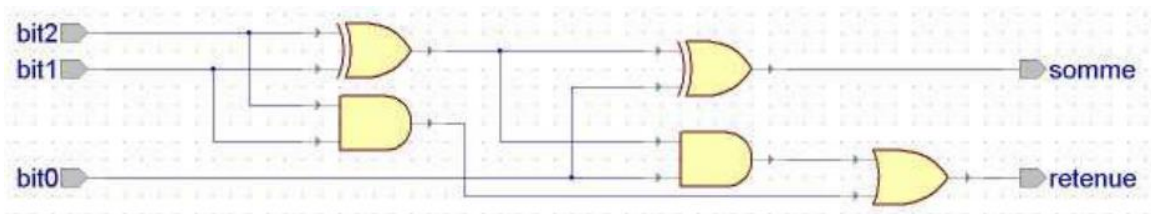
Dans les instructions qui suivent, on construit un circuit arithmétique de base : un additionneur à 3 bits. Ce circuit possède une entrée de 3 bits, et deux sorties : une retenue ainsi que la somme des 3 bits d'entrée. Les sorties possibles sont donc (retenue, sortie)  $\in \{(0, 0), (0, 1), (1, 0), (1, 1)\}$ , correspondant respectivement aux cas où les trois bits d'entrée sont 0, un seul bit est 1, deux bits sont 1, et trois bits sont 1. Ce circuit indique donc le nombre de bits d'entrée qui valent 1.

Le tableau de vérité de ce circuit est donné ci-dessous :

bit0	bit1	bit2	retenue	somme
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

On peut obtenir les équations pour les deux sorties - retenue et somme - grâce à un tableau de Karnaugh.

Un schéma d'un circuit réalisant les deux fonctions 'retenue' et 'somme' est donné ici :



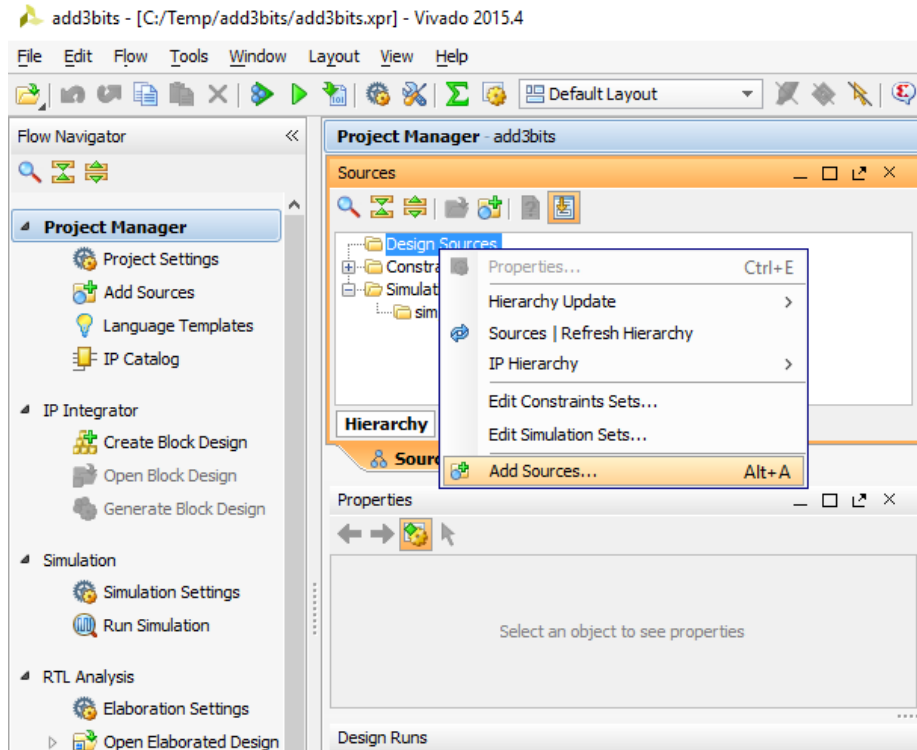
Le circuit est composé des composantes suivantes :

- deux portes OU-exclusif à deux entrées ;
- deux portes ET à deux entrées ;
- une porte OU à deux entrées ; et,
- trois ports d'entrée et deux ports de sortie.

Il est aussi possible de décrire le schéma directement avec des énoncés en VHDL.

### 3.2. Procédure

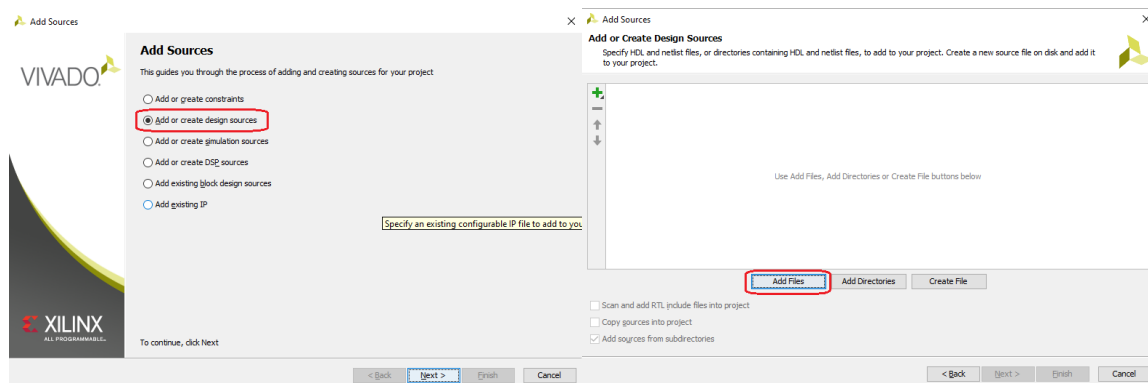
Dans la fenêtre **Project Manager**, vous pouvez voir votre projet. Afin d'y ajouter des fichiers, faites un clic droit sur **Design sources** et cliquez sur **Add Sources ...**



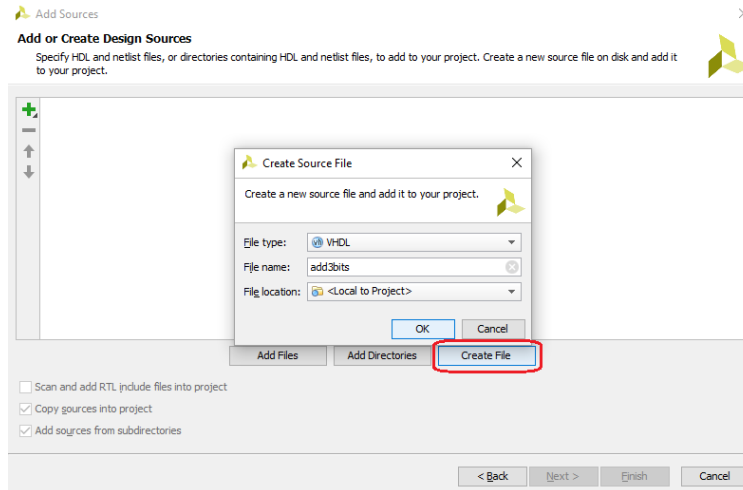
Dans la fenêtre qui apparaît, choisissez **Add or Create Design Sources**, puis cliquez sur **Next**.

Dans la fenêtre suivante, cliquez sur **Add Files...** et ajoutez votre fichier VHDL exemple **add3bits.vhd**, puis cliquez sur **Finish**. N'oubliez pas de cocher la case **Copy sources into projet**.

Le contenu du fichier add3bits.vhd décrit le circuit d'additionneur à 3 bits. Vous trouverez avec ce guide les fichiers add3bits.vhd et add3bits\_tb.vhd à télécharger [ici](#).



Une deuxième possibilité : Vous pouvez aussi cliquer sur **Create File...** pour créer votre fichier vhd comme le montre la figure suivante.



Cliquez sur **Finish**. Vous pouvez maintenant donner les entrées et sorties de votre module puis cliquez sur **OK**.

Vous verrez alors un nouveau fichier VHDL dans votre projet qui contient déjà votre description matérielle partielle. Remplacez ce qui est déjà présent par le code VHDL du fichier add3bits.vhd.

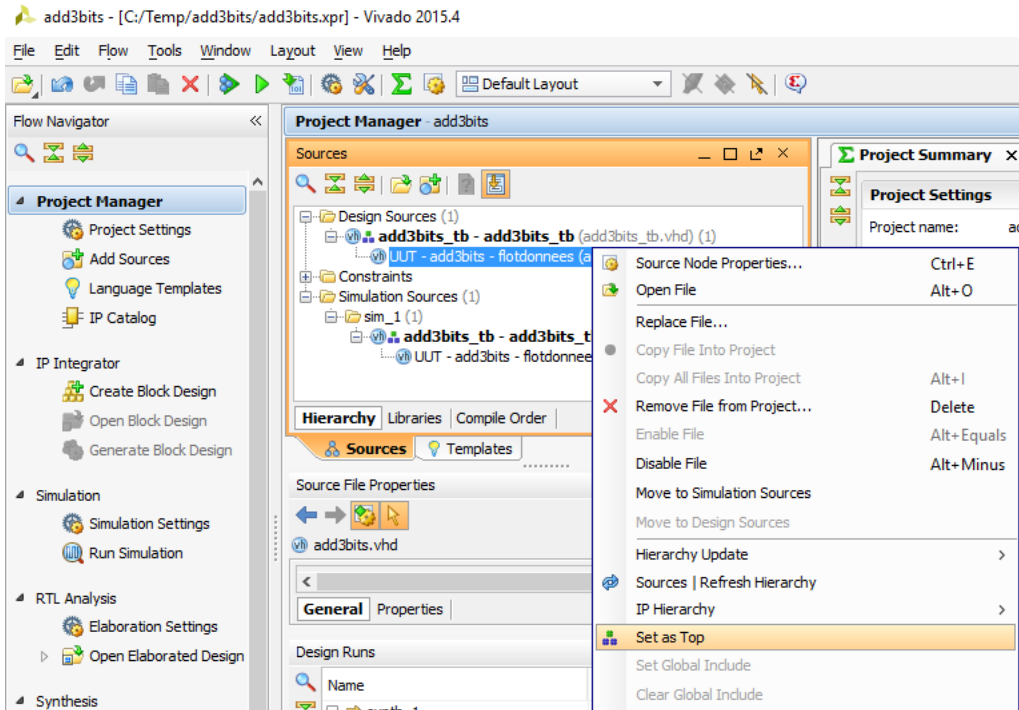
#### 4. Simulation comportementale

La simulation est une étape essentielle pour vérifier la description correcte d'un circuit logique. Les étapes suivantes décriront comment simuler un code VHDL avec Vivado.

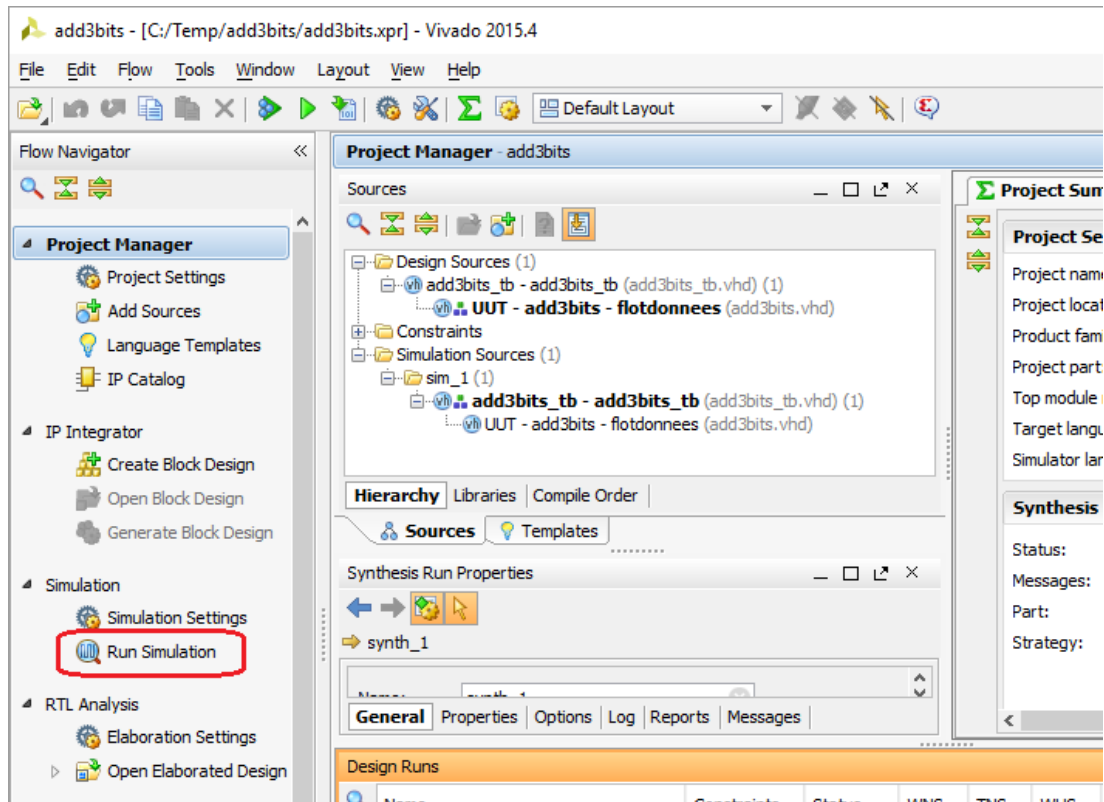
Dans Vivado Project Manager, vous trouverez deux types de sources : *Design Sources* et *Simulation Sources*.

Afin d'effectuer la simulation de votre code, une approche puissante consiste à décrire le simulateur à l'aide d'un banc d'essai, lui-même codé en VHDL. Le fichier VHDL qui contient le banc d'essai doit être identifié comme module principal, ou *top* dans la terminologie de Vivado.

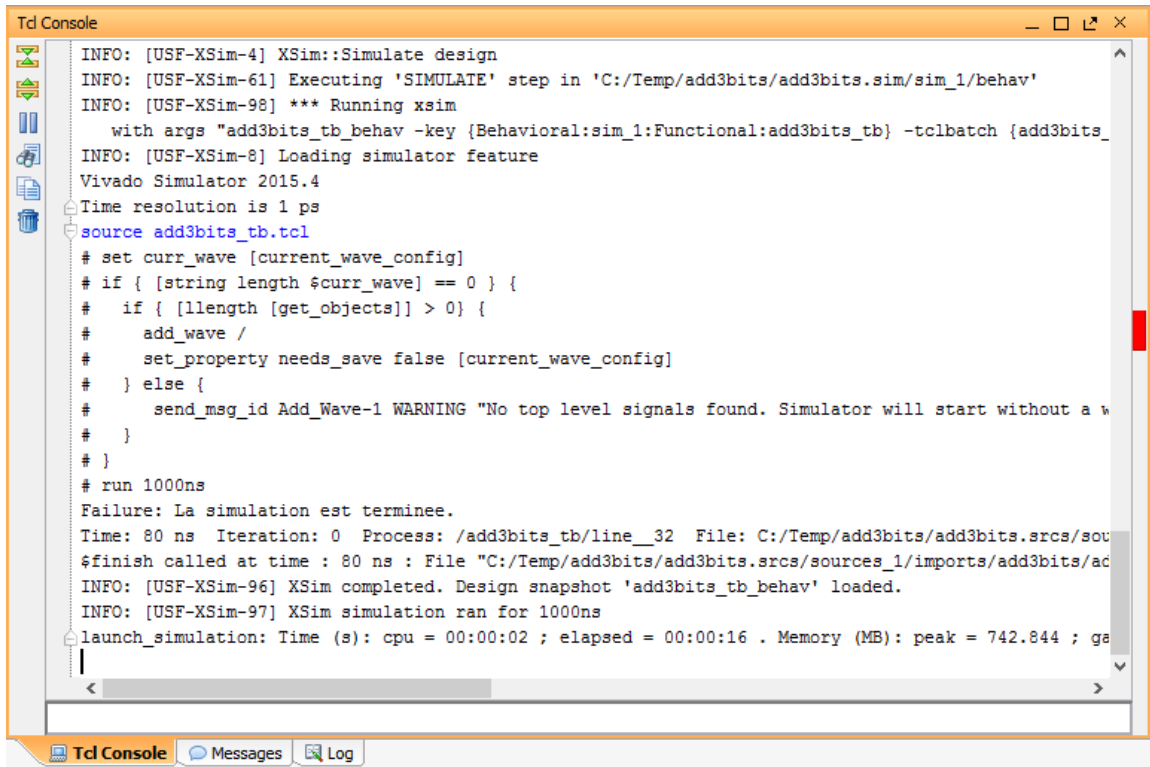
Vérifiez si votre banc d'essai est configuré comme *top* en *Simulation Sources*. Sinon, faites-le comme illustré dans la figure suivante.



Pour simuler, cliquez sur *Run Simulation* dans le panneau *Flow Navigator* → *Simulation*.



Les étapes de la simulation sont affichées dans la *TCL console*. Vous pouvez y observer les erreurs de compilation ainsi que les messages de type assert-report insérés dans votre code, et qui ont pour but de générer des messages d'erreur et des messages informatifs.

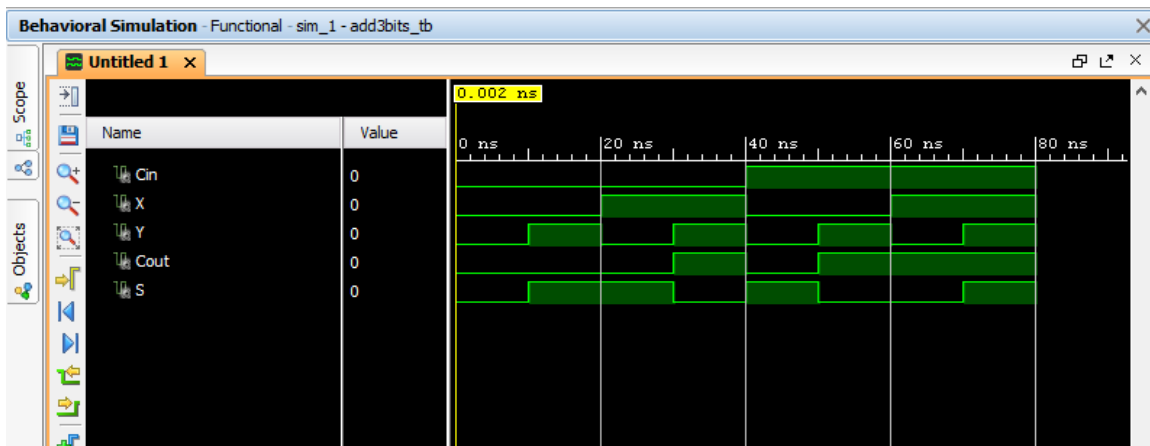


```

INFO: [USF-XSim-4] XSim::Simulate design
INFO: [USF-XSim-61] Executing 'SIMULATE' step in 'C:/Temp/add3bits/add3bits.sim/sim_1/behav'
INFO: [USF-XSim-98] *** Running xsim
    with args "add3bits_tb_behav -key {Behavioral:sim_1:Functional:add3bits_tb} -tclbatch {add3bits_
INFO: [USF-XSim-8] Loading simulator feature
Vivado Simulator 2015.4
Time resolution is 1 ps
source add3bits_tb.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0 } {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#     send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a w
#   }
# }
# run 1000ns
Failure: La simulation est terminée.
Time: 80 ns Iteration: 0 Process: /add3bits_tb/line_32 File: C:/Temp/add3bits/add3bits.srcs/sou
$finish called at time : 80 ns : File "C:/Temp/add3bits/add3bits.srcs/sources_1/imports/add3bits/ac
INFO: [USF-XSim-96] XSim completed. Design snapshot 'add3bits_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:16 . Memory (MB): peak = 742.844 ; ga

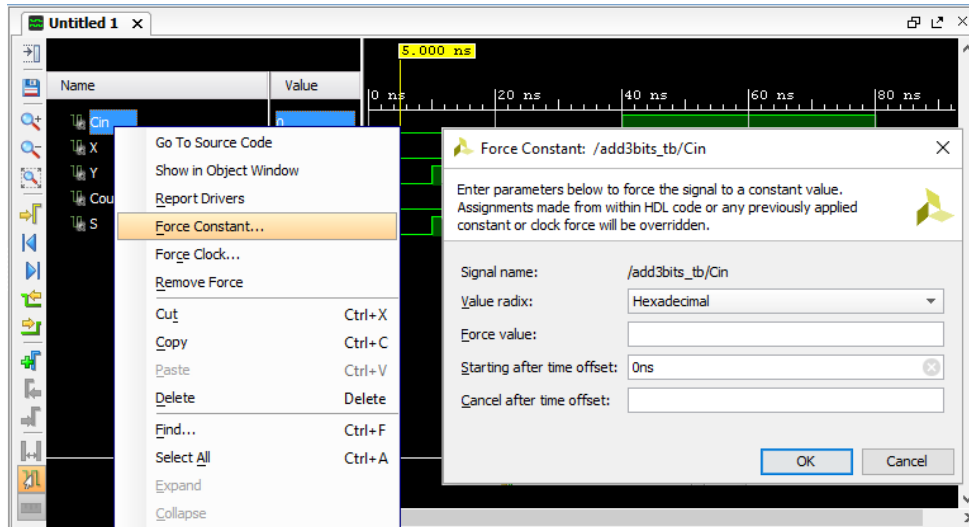
```

Ensuite, le résultat de la simulation peut être visualisé dans un chronogramme.



Le simulateur Vivado permet aussi de forcer manuellement la valeur des signaux, c'est à dire sans passer par un banc d'essai. Cette fonctionnalité peut être utile pour effectuer un test rapide pour vérifier le comportement de votre circuit. Cependant, en général il est toujours préférable d'utiliser un banc d'essai, ce qui documente votre procédure de test et facilite l'évolution de vos tests au fur et à mesure que vous développez la fonctionnalité de votre code.





## 5. Synthèse et implémentation du projet

### 5.1. Description

La synthèse d'un circuit consiste à traduire sa description en blocs disponibles dans la technologie utilisée. Par exemple, pour un circuit décrit avec un schéma et qui doit être réalisé sur un FPGA, le processus de synthèse convertit et regroupe les portes logiques du schéma en composantes réalisables sur le FPGA choisi. Pour un circuit décrit en VHDL, la synthèse analyse le code et infère des composantes logiques correspondantes à son comportement.

L'implémentation du circuit est divisée en quatre sous étapes :

- la transformation (*mapping*) : regrouper les composantes obtenues lors de la synthèse dans des blocs spécifiques du FPGA ;
- la disposition (*placement*) : choisir des endroits spécifiques sur le FPGA où disposer les blocs utilisés, et choisir les pattes du FPGA correspondant aux ports d'entrée et de sortie ;
- le routage (*routing*) : établir des connexions électriques entre les blocs utilisés ; et,
- la configuration (*configuration*) : convertir toute cette information en un fichier pouvant être téléchargé sur le FPGA pour le programmer.

### 5.2. Ports d'entrée et de sortie

Pendant l'étape de disposition de l'implémentation, il faut assigner des pattes spécifiques du FPGA à des ports d'entrée et de sortie de son design. Pour le design présent, les ports d'entrée sont X, Y et Cin, et les ports de sortie sont Cout et S.

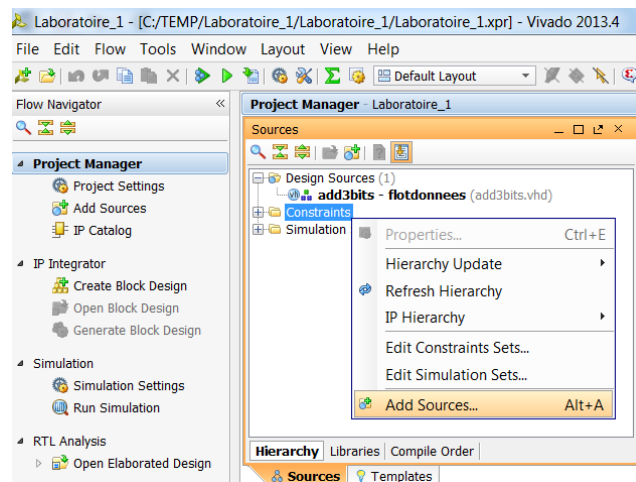
L'assignation des ports se fait par l'entremise d'un fichier de contraintes avec l'extension « .xdc » (pour *xilinx design constraints* file).

Ouvrez un éditeur de texte (comme Notepad++) et copiez-y les lignes suivantes. On suppose que vous utilisez la planchette Nexys 4 DDR.

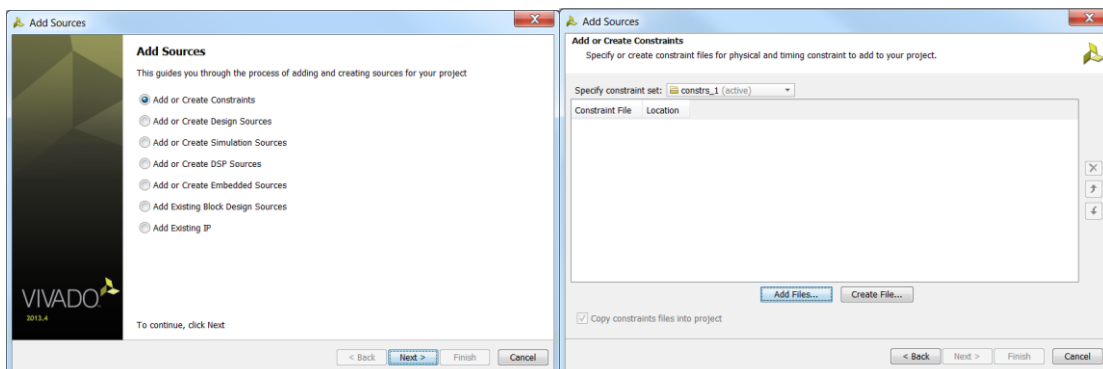
```
# add3bits.xdc
# pour planchette Nexys 4 ddr

# LEDs
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { Cout }];
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports { S }];
# commutateurs
set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { X }];
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 } [get_ports { Y }];
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 } [get_ports { Cin }];
```

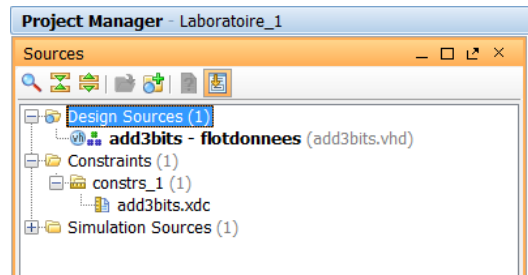
On peut noter que chaque port est identifié dans le fichier, exactement comme il apparaît dans le code VHDL. Le symbole du dièse (#) indique que le reste de la ligne est un commentaire pour les fichiers .xdc. Sauvegardez le fichier dans le même répertoire que les autres fichiers sources de votre design, sous le nom de add3bits.xdc. Puis, dans **Project Manager**, faites un clic droit sur **constraints**, cliquez sur **Add Sources...** comme le montre la figure suivante :



Cochez la case **Add or Create Constraints**, puis cliquez sur **Next** pour choisir le fichier que vous venez de créer (illustré dans les deux figures ci-dessous).



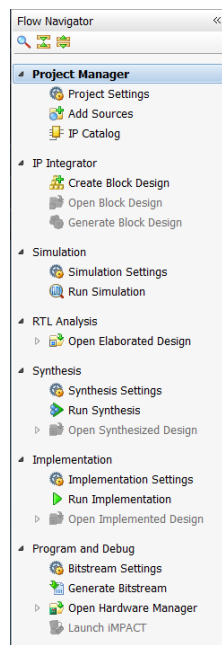
Vous devriez avoir la vue suivante dans Vivado :



Consultez le manuel de l'utilisateur de votre planchette de développement pour plus de détails.

### 5.3. Procédure

Il vous est maintenant possible de faire la synthèse de votre circuit en vue d'une implémentation sur le FPGA. Pour ce faire, il faudra utiliser l'onglet **Flow Navigator** de Vivado (Colonne à gauche):



Pour les opérations de **Synthesis**, **Implementation** et **Generate Bitstream**, il faut faire un clic-droit sur **Run Synthesis**, **Run Implementation** et **Generate Bitstream**, respectivement.

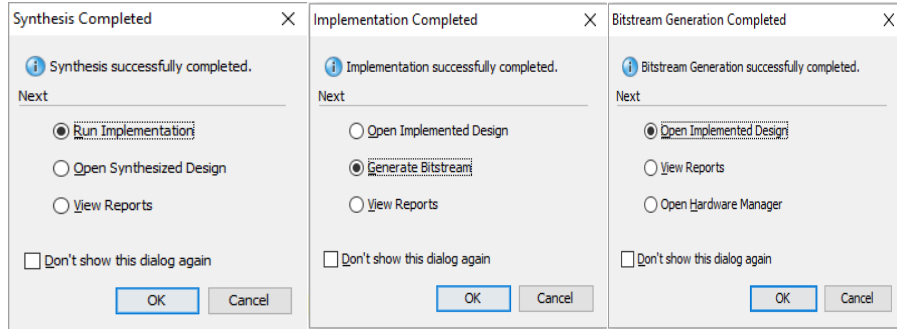
**Run Synthesis** : permet de faire la synthèse du circuit en blocs configurables du FPGA.

**Run Implementation** : permet de faire les *mapping* et routage nécessaires pour placer le tout sur le composant FPGA.

**Generate Bitstream** : génère le fichier (.bit) utilisé pour programmer le FPGA.

Après chacune des étapes, un message de succès devrait s'afficher dans la console. Dans le cas contraire, il faut corriger les erreurs ou inspecter les avertissements (warnings) pour vous assurer qu'ils ne proviennent pas d'erreurs dans votre code.

L'erreur la plus fréquente est produite par une différence entre les noms des ports de votre circuit dans votre code VHDL et dans le fichier de contraintes .xdc.

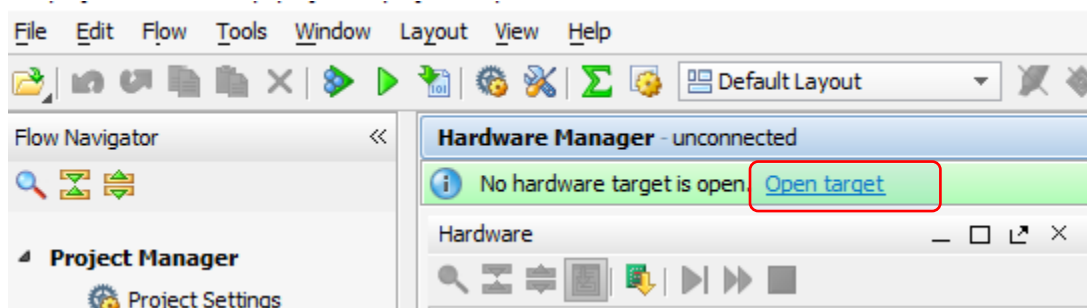
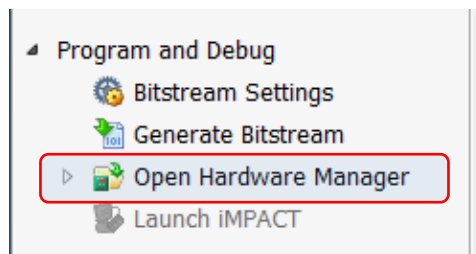


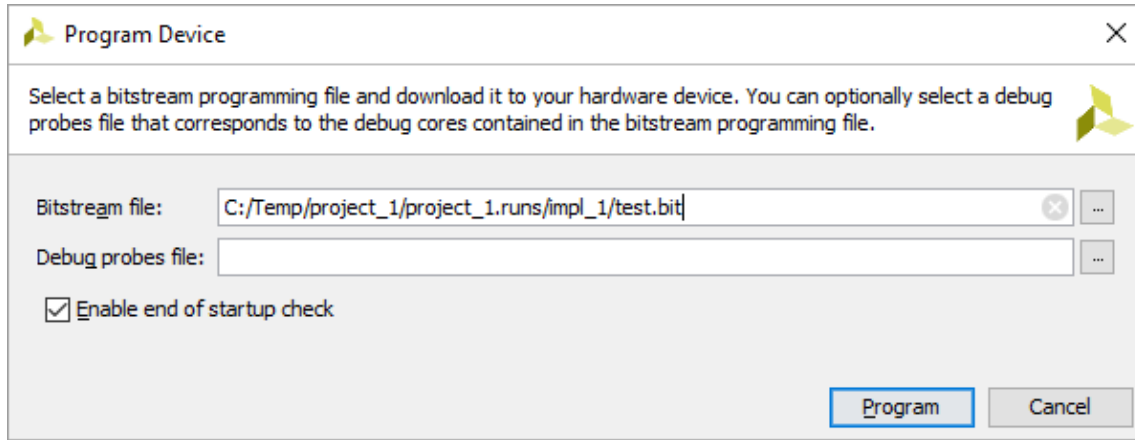
Lors de ces différentes étapes, plusieurs données deviendront disponibles dans le *Design Summary*. Vous aurez donc aisément accès, entre autres, aux ressources utilisées et aux résultats temporels (chemin critique, etc.).

## 6. Programmation du FPGA et vérification

Une fois les trois étapes (synthèse, implémentation et génération du bitsream) finies, on peut procéder à la programmation du FPGA.

Dans **Flow Navigator**, cliquez sur **Open Hardware Manager** → **Open Target** → **Program** pour programmer le FPGA tel que le montrent les figures suivantes :





## 7. Contrôle des versions

Version	Altérations	Auteur
0.0	Premier release	Hamza
1.0	Ajout du guide de simulation	Jeferson
1.1	Précisions au texte et orthographe	Pierre