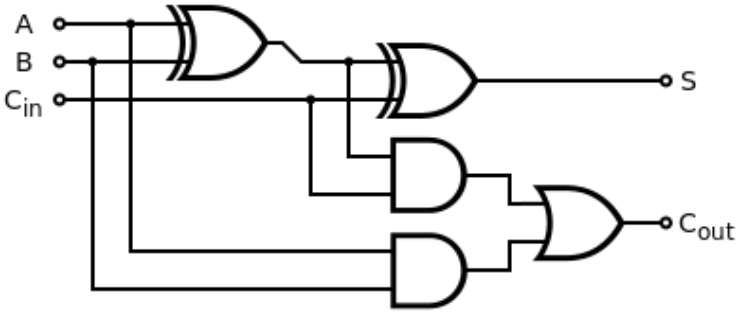
PROJET D'ELECTRONIQUE NUMÉRIQUE
EN103

Département Electronique 1ère année — Semestre 6

Tutoriel Vivado et projet



1. Tutoriel Vivado

1.1 Objectifs

L'objectif de ce tutoriel est de donner les principales clefs pour utiliser la suite d'outils Vivado de la société Xilinx. Vous allez, dans un premier temps, réaliser un simple additionneur complet (full adder) en utilisant un language de description matérielle: le VHDL. Nous allons ensuite simuler le fonctionnement du full adder et le tester sur une carte de développement intégrant un circuit FPGA. A la fin de ce tutoriel, vous devez être capable de :

- Lancer l'outil Vivado,
- Créer un nouveau projet,
- Décrire un circuit combinatoire simple en VHDL,
- Créer un fichier de contraintes .xdc,
- Créer un testbench pour votre description VHDL,
- Simuler le comportement de votre description,
- Faire la synthèse, le placement, le routage et la génération du bit stream,
- Reconfigurer le circuit FPGA de la carte pour tester votre système.

1.2 Lancement de l'outil Xilinx Vivado 2018

En fonction du système d'exploitation sur lequel vous travaillez, vous pouvez lancer l'outil en cliquant sur l'icône ou bien depuis une fenetêtre de commande. Après avoir lancé l'outil Vivado 2018, une fenêtre apparait. Si cette fenêtre n'est pas vide, faites :

File>Close Project

Généralement, l'outil s'ouvre avec le dernier projet ouvert.

1.3 Création d'un nouveau projet

Durant toute la durée des TP, vous allez travailler sur un seul projet dans lequel vous ajouterez les éléments des différents TP au fur et à mesure. Créez un projet :

File>Project>New

La fenêtre **New Project** apparaît:

- **Cliquez sur Next**
- **Project Name** : nom de votre projet : `TP_1` (attention pas d'espace)
- **Project Location** : chemin de sauvegarde du projet et des fichiers associés
- **Cliquez sur Next**
- **Sélectionnez RTL Project**
- **Cliquez sur Next**
- **Cliquez sur Next**
- **Cliquez sur Next**
- A l'aide des filtres, sélectionnez la référence du circuit FPGA présent sur votre carte de développement : par exemple pour la carte Nexys A7 : **xc7a100tcsg324-1**
- **Cliquez sur Next**
- **Cliquez sur Finish**

NB : A tout moment il est possible de changer ces informations, et bien d'autres encore, dans le menu **Tools** > **Settings**.

Le projet vide est maintenant crée. Nous allons à présent créer un fichier source VHDL qui contriendra la description du full adder, et l'ajouter au projet.

1.4 Ajouter des fichiers sources

Pour ajouter une nouvelle source faites :

- **File** > **Add Sources**.
- Sélectionnez **Add or Create Design Sources**.
- Cliquez sur **Next**
- Cliquez sur **Create File**
- **File Type:** VHDL
- **File name:** full_adder
- **File location:** Local to Project
- Cliquez sur **Ok**
- Cliquez sur **Finish**
- Definissez les port d'entrée / sortie :
  - A : in 1 bit
  - B : in 1 bit

  - C : in 1 bit
  - SUM : out 1 bit
  - CARRY : out 1 bit
- Cliquez sur **OK**

Une fois le fichier VHDL crée, celui-ci apparait dans l'arboresence de la fenêtre **Sources**. Il se trouve à la fois dans le repertoire **Design Sources** et dans le répertoire **Simulation Sources**

Double cliquez sur le fichier full_adder.vhd.

Le fichier VHDL apparait alors dans l'éditeur. On peut modifier la spécification des ports d'entrée/sortie dans la partie entity du fichier: Par exemple, pour un port d'entrée TOTO sur 1 bit:

```
TOTO : in  STD_LOGIC;
```

Pour un port de sortie TATA sur 3 bits:

```
TATA : out  STD_LOGIC_VECTOR(2 downto 0);
```

1.5 Description de l'architecture

Une fois les ports d'entrée/sortie définis, il nous faut décrire l'architecture de notre full_adder. Un circuit combinatoire implémentant un `full adder` est donnée sur la Figure 1.1.

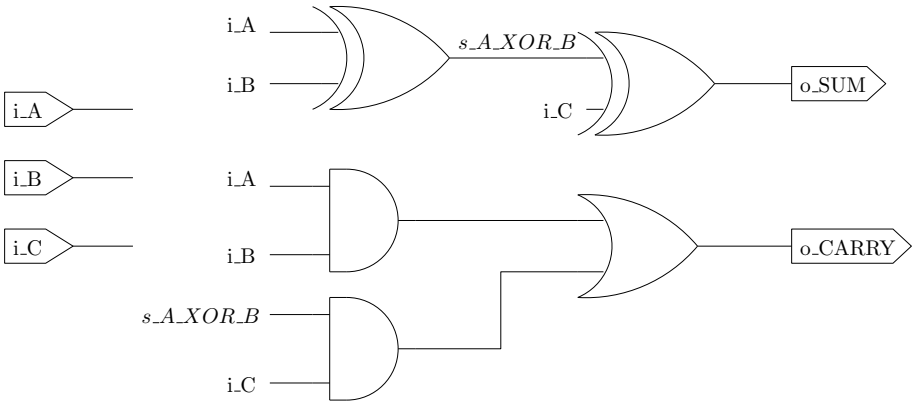


Figure 1.1: Architecture de l'additionneur complet

Pour décrire ce circuit en VHDL, nous avons besoin de portes logiques de base. En VHDL, tous les opérateurs logiques de base sont disponibles: `not`, `and`, `or`, `nand`, `nor`, `xor`, `xnor`. Par exemple pour générer $s\_AXORB = i\_A \oplus i\_B$, il suffit de taper, après le mot clef `begin`:

```
s_AXORB <= i_A xor i_B;
```

Nous avons également besoin de déclarer le fil `AXORB`. Pour cela, il suffit de le déclarer avant le mot clef `begin`:

```
signal s_AXORB : std_logic;
```

Modifiez le fichier VHDL pour obtenir la description du full_adder suivante:

```vhdl
-- Déclaration des librairies
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Déclaration de l'interface (entity)
entity full_adder is
Port (  i_A     : in  STD_LOGIC;
        i_B     : in  STD_LOGIC;
        i_C     : in  STD_LOGIC;
        o_SUM   : out STD_LOGIC;
        o_CARRY : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is

-- déclaration des fils (signal) et d'éventuels composants (component)
signal AXORB : std_logic;

-- début de l'architecture
begin

   -- description du contenu de l'architecture
   s_AXORB <= i_A xor i_B;
   o_SUM   <= s_AXORB xor i_C;
   o_CARRY <= (i_A and i_B) or (i_C and s_AXORB);

-- fin de l'architecture
end Behavioral;
```

1.6 Simulation du Full Adder à l'aide d'un test bench

Avant d'implémenter le Full adder sur le circuit FPGA, vous allez simuler le comportement de la description VHDL. Pour cela, nous allons utiliser un autre fichier VHDL particulier : le *test bench*. Celui-ci n'a pas d'entrées/sorties, il instancie le composant à tester (ici le top level qui contient lui-même le full adder) et permet de générer des stimuli logiques qui seront envoyés sur les entrées. La simulation consistera alors à observer l'état des sorties en fonction des stimuli appliqués en entrée. Par habitude/convention, on nomme le fichier test bench du nom du composant à tester avec le préfixe `tb_`. Téléchargez le test bench `cleroux.vvv.enseirb-matmeca.fr/EN103/tb_full_adder.vhd`. Pour ajouter le test bench à votre projet :

- **File** > **Add Sources**.
- Sélectionnez **Add or Create Design Sources**.
- Cliquez sur **Next**
- Cliquez sur **Add Files**
- Allez chercher le testbench là où vous l'avez téléchargé.
- Cliquez sur **Finish**

Vérifiez que **tb_full_adder.vhd** apparait en gras dans la fenêtre **Sources>Simulation Sources>Sim_1**. Si ce n'est pas le cas, faites **Clique droit>Set as Top**. Ouvrez ce fichier et repérez où se trouve la génération des stimuli logiques. Dans le **Flow Navigator** sur la gauche, lancez la simulation en cliquant sur **Run Simulation>Run Behavioral Simulation**. Après quelques secondes, la simulation se lance et un chronogramme apparait. Utilisez les outils de zoom et vérifiez que le full_adder fonctionne. Si certains signaux n'apparaissent pas dans le chronogramme, il faut relancer la simulation pour que l'outil re-simule et affiche ces signaux : **Run > Relaunch Simulation**.

1.7 Implantation du Full Adder sur le FPGA

1.7.1 Ajout du fichier de contraintes

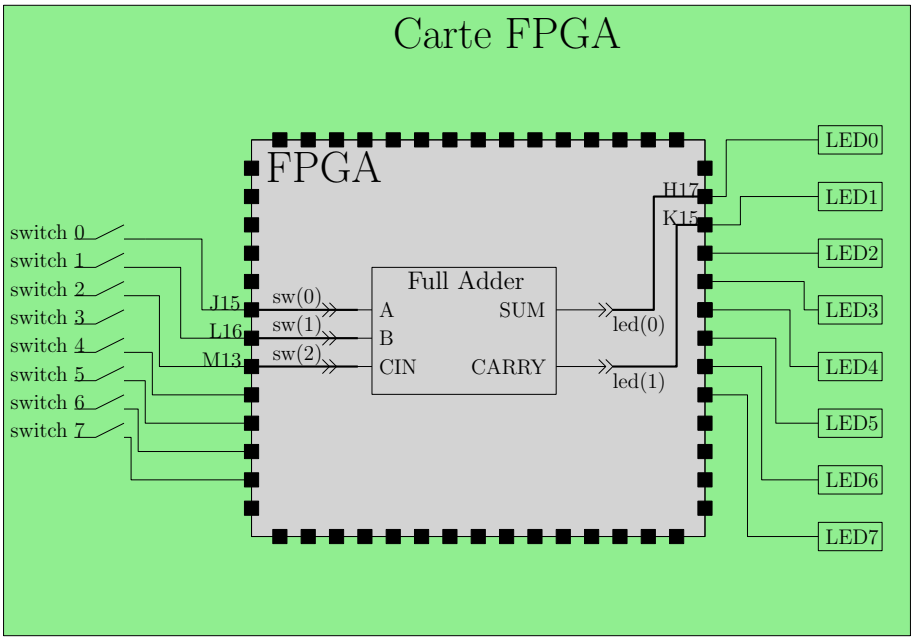


Figure 1.2: Exemple de connexion des I/O du module Full Adder aux I/O d'un FPGA sur une carte de développement Basys

Maintenant que l'architecture du Full Adder est décrite en VHDL et validée en simulation, vous allez l'implanter sur le circuit FPGA. Avant cela, il faut définir un fichier de contraintes (fichier `.xdc`). En effet, notre circuit décrit dans l'architecture `full_adder` va être implanté à l'intérieur du FPGA. Il faut donc indiquer à l'outil à quelles entrées/sorties du FPGA il faut connecter les entrées/sorties de notre système (Figure 1.2). Cela est fait dans le fichier `.xdc`. Téléchargez le fichier de contraintes `.xdc` correspondant à votre carte de développement : `cleroux.vvv.enseirb-matmeca.fr/EN103/Nexys_A7.xdc`. Ajoutez le fichier de contrainte à votre projet:

- **File > Add Sources**
- Sélectionnez **Add or Create Constraints**
- Cliquez sur **Add Files**
- Allez chercher le fichier `.xdc` là où vous l'avez téléchargé

- Cliquez sur **Finish**

Après l'ajout, votre fichier de contrainte apparait dans l'arborescence **Constraints** de la fenêtre Sources. Ouvrez-le fichier de contrainte et vérifiez que les lignes spécifiant les connexions à toutes les entrées sorties de votre top level sont bien décommentées. A quelles broches sont connectées les 5 entrées / sorties ?

1.7.2 Elaboration RTL

Définissez le fichier `top_full_adder.vhd` comme top level en faisant **Clique droit>Set As Top**, le nom du fichier doit alors apparaitre en gras dans l'arboresence **Design Sources**. Attention, le top_level pour la simulation (`tb_top_full_adder.vhd`) et pour l'implémentation (`full_adder.vhd`) ne sont pas les mêmes.
Dans le Flow Navigator, cliquez sur **Open Elaborated Design**. Après quelques secondes, vous devez voir apparaitre le circuit constitué de portes logiques.

1.7.3 Synthèse logique

Dans le Flow Navigator, cliquez sur **Run Synthesis**. Le processus de synthèse se lance. Une fois terminé, vous pouvez observer le schéma logique composé de LUT et de buffer en cliquant sur Schematic. Vous pouvez vérifiez le contenu de chaque LUT :

- **Clique droit sur une LUT**
- **Cell properties**

1.7.4 Placement et routage

Une fois la synthèse logique effectuée, l'outil va affecter (placer) les 2 LUTs synthétisées sur les LUTs disponibles sur le FPGA (ce FPGA en contient plusieurs dizaines de milliers). Il va ensuite configurer les interconnexions pour relier les entrées du circuit FPGA aux LUTs et les sorties des LUTs aux sorties du circuit. Pour lancer cette phase de placement/routage, dans le Flow Navigator, cliquez sur Run Implementation. Une fois le placement routage terminé, vous pouvez voir le circuit implémenté dans le FPGA en cliquant sur Schematic.
Une fenêtre device apparait. En zoomant sur le circuit essayez de retrouver les LUTs, les entrées `i_A`, `i_B` et `i_C` et les sorties `o_CARRY` et `o_SUM`.

1.7.5 Génération du bitstream et configuration du FPGA

La dernière étape du processus d'implantation consiste à générer un *bit stream*, c'est à dire un fichier binaire qui va être transféré sur le FPGA et qui va permettre de configurer le contenu de chaque LUT et de chaque interconnexion du FPGA pour que celui-ci implémente l'architecture que vous avez visualisée après la phase de placement routage.
Pour générer ce bit stream:

- Branchez la carte à votre PC, et allumez votre carte (interrupteur à coté du cable mini USB)
- Dans la rubrique **Program and Debug**, cliquez sur **Generate Bitstream**
- Après quelques secondes, une fenêtre apparait, sélectionnez **Open Hardware Manager**, puis **OK**

- Clique droit sur la cible, puis **Program Device**

Le bitstream est transféré sur le circuit FPGA via le cable USB. Une fois le circuit configuré, vous pouvez changer la valeur des interupteurs et vérifier que votre full adder fonctionne.
NB: Il est possible de réaliser toutes les étapes de l'implémentation jusqu'à la génération du bitstream en cliquant directement sur **Generate Bitstream**.
Avant de faire le projet, vous devez à présent, être en mesure de :

- Créer une nouvelle source VHDL
- Ajouter une source VHDL existante
- Décrire un circuit combinatoire simple
- Instancier un component dans une description VHDL
- Créer un test bench avec quelques stimuli simples
- Mettre à jour un fichier de contraintes .xdc
- Lancer une simulation logique
- Implanter et tester un système sur carte FPGA

Dans la suite, n'hésitez pas à revenir vers ce tutoriel si certains concepts ne sont pas encore clairs.

2. Projet

Jeu de conversion

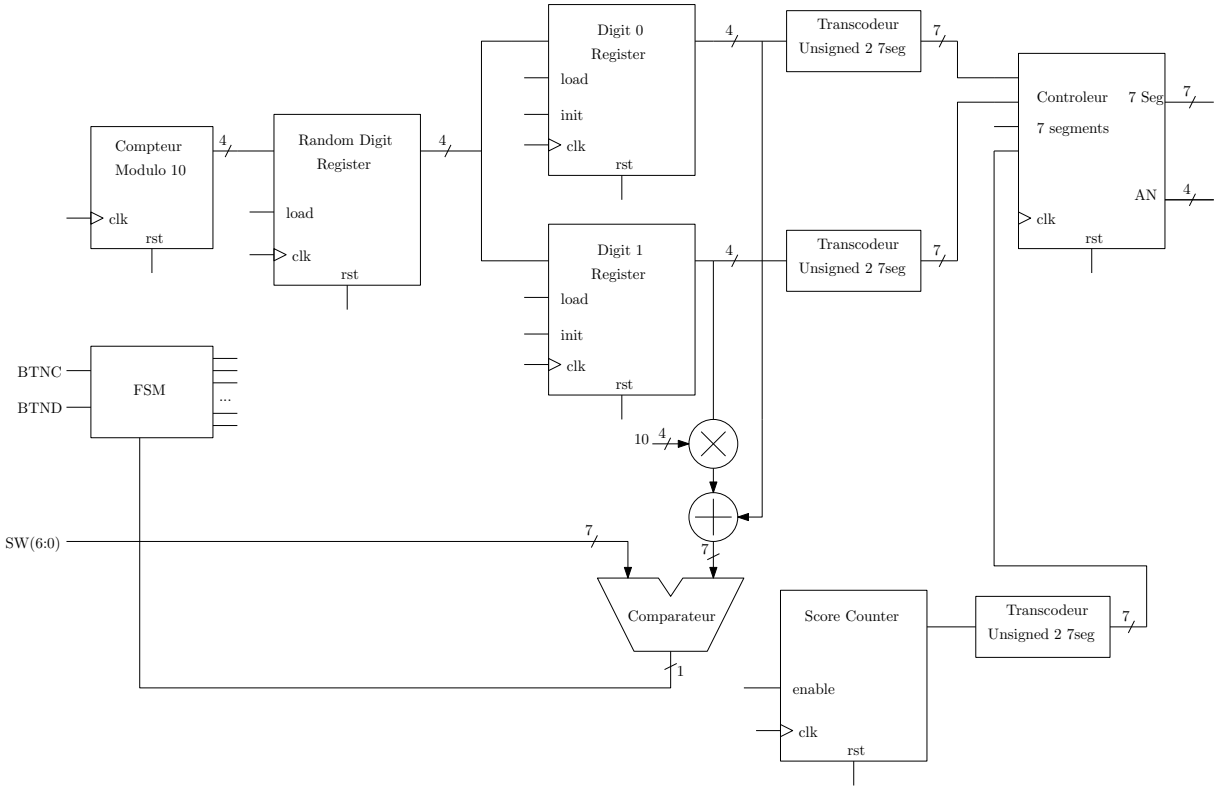


Figure 2.1: Découpe architecturale.

Jeu de conversion binaire

L'objectif du projet est de concevoir un jeu pour tester sa maîtrise de la conversion d'un nombre décimal (sur 2 chiffres) en un nombre binaire (7 bits). Le système génère deux chiffres de manière aléatoire, les affiche sur les afficheurs 7 segments. L'utilisateur doit alors convertir la valeur affichée (nombre entre 0 et 99) en binaire puis l'indiquer sur les interrupteurs. Le système vérifie alors l'équivalence entre la valeur binaire proposée par l'utilisateur et le nombre affiché. Le système incrémente le score de 1 uniquement si la réponse est correcte et affiche le score mis à jour sur un afficheur.

Fonctionnement

A l'initialisation, le système affiche le score (0) sur l'afficheur 3, ainsi que la valeur 00 sur les afficheurs 0 et 1. Un premier appui sur BTNC affiche le premier nombre tiré aléatoirement sur l'afficheur 0. Un deuxième appui sur BTNC affiche le deuxième nombre sur l'afficheur 1. A partir de là, l'utilisateur positionne les switchs à la valeur binaire qu'il pense être équivalente à la valeur décimale sur 2 chiffres affichée puis il appuie sur BTND afin de lancer la vérification. Si les deux valeurs binaires (calculée et proposée) sont identiques, alors on incrémente le score de 1, sinon le score ne bouge pas.

Découpe architecturale

La découpe architecturale est présentée sur la Figure 2.1.

- Pour tirer un chiffre aléatoirement, on fait fonctionner un compteur modulo 10 à 100MHz. Lorsqu'un appui bouton est détecté, on vient charger la valeur courante du compteur dans un registre (Random Digit Register).
- Deux registres sont ainsi nécessaires pour stocker le nombre à deux chiffres qu'il faudra essayer de convertir. L'affichage de ce nombre est assuré par deux transcodeurs non-signés => 7 segments suivi du controleur d'affichage 7 segments.
- Pour convertir le nombre décimal en binaire, on multiplie les dizaines par 10 et on ajoute les unités.
- Cette valeur est comparée à la valeur présente sur les switches.
- La machine d'état a pour entrée les boutons ainsi que le résultat de la comparaison. Elle permet de contrôler tous les composants séquentiels du système (signaux de chargement, initialisation, ...).
- Le score est stocké et mis à jour dans un compteur qui n'est incrémenté que lorsque la réponse est correcte.

Améliorations

Une fois que votre projet fonctionne, il est possible d'améliorer le système de diverses manières:

- Donner la possibilité de jouer avec un seul chiffre, deux chiffres ou bien même trois chiffres.
- Rajouter la conversion héxadécimale => binaire
- Indiquer avec des LEDs si la réponse est correcte ou non.
- Donner un nombre limite d'essais global.
- Ajouter une limite de temps en ajoutant un chronomètre.
- Afficher le meilleur score.
- etc ...

A vous de personnaliser votre système avec les fonctionalités de votre choix.