
Field Programmable Gate Arrays

FPGA_s



ENSP, YAOUNDE

YUHALA PETERSON JR, DEKOU WILLIAM

4GI

AUGUST 2016

SOUS LA SUPERVISION DE
MME A.M.CHANA

Abstract

The increasing integration density according to Moore's law is being slowed due to economic and physical limitations. This technological evolution leads to a higher number of physical defects after manufacturing a circuit. As yield goes down, one of the future challenges is to find a way to use a maximum of fabricated circuits while tolerating physical defects spread all over the chip. Field Programmable Gate Arrays (FPGAs) are integrated circuits that contain logic blocks and configurable interconnects. Their ability to integrate more complex applications, their flexibility and good performance make FPGAs the perfect target architecture. The aim of this report is to describe the architecture and functioning of FPGAs.

The first part of the report focuses on the theoretical part of Programmable Logic technology ie the physical architecture and functioning of programmable logic devices (PLDs).

The second part of this report describes the steps involved in the conception and simulation of logic circuits using the Altera Development and Education Board using the Quartus software. The latter part is divided into two sections, one devoted to graphical conception, and the last section for conception using the VHDL language.

Table des matières

I THEORIE	1
1 Introduction	1
1.1 PLDs-Circuits Logiques Programmable	1
2 FPGA	2
2.1 Présentation rapide	2
2.2 Architecture matérielle d'un FPGA	2
2.3 Fonctionnement d'un FPGA	4
2.4 Conception et Simulation des circuits logiques avec un FPGA	7
2.5 Structure d'un programme VHDL	7
II PRATIQUE	11
3 Tutoriel sur l'utilisation du logiciel Altera Quartus II pour la conception et simulation des circuits logiques	11
3.1 Conception Schématique	11
3.2 Conception des circuits logiques en VHDL	37

Première partie

THEORIE

1 Introduction

Comme vous pouvez l'imaginer , la conception des centaines de différents circuits logiques pour répondre à tous les possibles exigences du monde numérique complexe est devenu très difficile. En plus d'avoir des circuits logiques très complexe, un autre problème est la quantité excessive de la zone sur une carte de circuit imprimé qui est consommé pendant la production des IC contenant des circuits logiques. Puis vint "logique programmable " -l'idée que la mise en œuvre de tous les modèles logiques en utilisant les CI7400¹ ou CI4000 en séries ne sont plus nécessaires . Au lieu de cela , une entreprise achètera plusieurs circuits intégrés configurables par l'utilisateur qui seront personnalisés (à savoir , programmé) pour effectuer l'opération logique spécifique qui est requise . Ces CI sont appelés PLDs(Programmable Logic Devices).

1.1 PLDs-Circuits Logiques Programmable

Un circuit est dit configurable (ou programmable) lorsque sa fonctionnalité n'est pas prédéfini lors de sa fabrication mais peut être spécifié plus tard par une configuration (un programme). La figure suivant représente la famille des PLDs les plus utilisés.

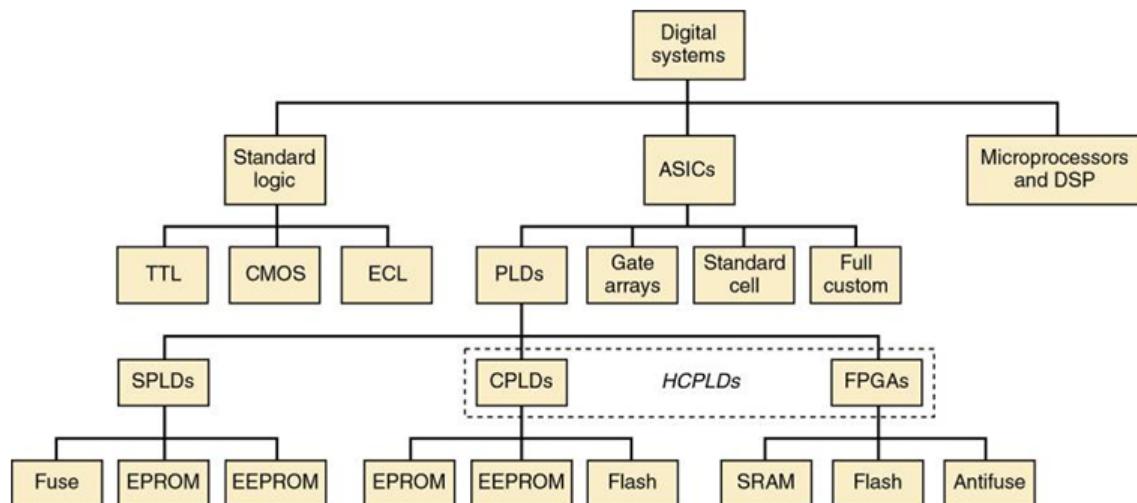


FIGURE 1 – Famille des PLDs

1. Modèle de circuit intégré

2 FPGA

2.1 Présentation rapide

Les FPGAs (Field Programmable Gate Arrays) sont des circuits intégrés qui contiennent des blocs de logique configurables (programmables), ainsi que des interconnexions configurables entre ces blocs. Ceci permet de les reprogrammer à volonté afin d'accélérer notamment certaines phases de calculs. L'avantage de ces genres de circuits est leur grande souplesse qui permet de les réutiliser à volonté dans des algorithmes différents en un temps très court.

Le FPGA diffère des autres PLDs en ce que, au lieu de résoudre la conception logique en interconnectant des portes logiques, il utilise un LTU²(table de transcodage) pour résoudre l'exigence d'un circuit logique particulière. Cela permet aux fabricants des PLDs à former une conception plus simple, la création d'un PLD beaucoup plus dense et plus rapide. En plus d'avoir des milliers d'éléments logiques internes, les FPGA ont des centaines de broches E/S avec les interconnexions programmable et les registres de stockage interne. La série Altera Cyclone est un exemple d'une famille FPGA.

2.2 Architecture matérielle d'un FPGA

Un FPGA est composé d'une matrice d'éléments logiques appelés Configurable Logic Block (CLB) connectés entre eux par des ressources d'interconnexions (Figure 2).

Chaque CLB est composé de BLEs (Basic Logic Element) utilisés pour implémenter la partie logique du circuit. Un CLB est donc caractérisé par le nombre de ses entrées I et par le nombre N de BLE qu'il contient. Dans les FPGAs modernes ce nombre de BLE peut varier de 3 à 12 et chaque BLE peut être connecté à n'importe quelle entrée I du CLB ou à un autre BLE comme indiqué par la figure 3. Un BLE est composé d'un ensemble de tables de transcodage ainsi que d'une bascule D pour implémenter les fonctions de bases grâce au bloc mémoire (SRAM), suivie d'un multiplexeur. Un BLE est composé de k LUTs ce qui permet d'implémenter une fonction logique à k entrées et une sortie et nécessite 2k blocs mémoire pour la configuration. La bascule D quant à elle permet de choisir entre un fonctionnement logique ou séquentiel. Ensemble les LUT et la bascule D forme un Basic Logic Element (Figure 3).

Les éléments logiques sont connectés entre eux et aux entrées/sorties grâce au réseau d'interconnexion. Ce réseau est programmé de la même manière que les éléments logiques avec des RAMs statiques. L'ensemble permet de configurer complètement le FPGA. En théorie, il est donc possible de charger n'importe quelle fonction logique sur un FPGA. Enfin, la reprogrammation de l'ensemble des SRAM permet de reconfigurer entièrement le circuit et donc de changer l'application ou la fonction du FPGA. Pour finir, lors de la configuration du FPGA, la *netlist* contenant l'ensemble des connexions est envoyée au FPGA. Le placement des blocs logiques et la manière dont ils sont interconnectés est alors défini par un algorithme de placement et de routage.

2. LUT-Look Up Table

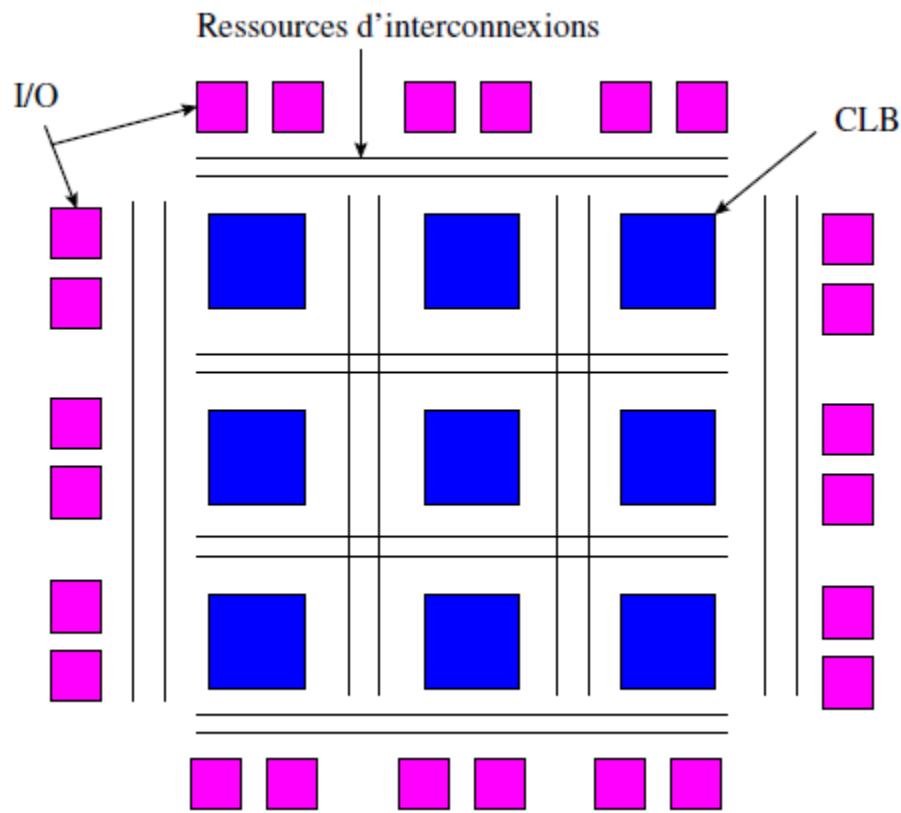


FIGURE 2 – Architecture d'un FPGA

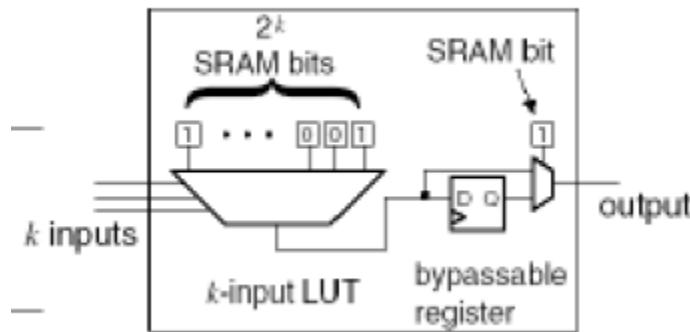


FIGURE 3 – Architecture d'un CLB

2.3 Fonctionnement d'un FPGA

Pour voir comment une table de transcodage fonctionne, reportez-vous aux figure 5. Dans la figure 4, la logique classique de l'équation $X = ABCD + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ est mis en œuvre en utilisant des circuits intégrés 7400. Dans ce cas, $X = 1$ pour trois combinaisons différentes des quatre entrées ($X = 1$ quand $ABCD = 1111$ ou 1010 ou 0000). La figure 5 montre le même circuit logique mise en œuvre dans un LUT FPGA. Le LUT fonctionne de manière similaire à une table de vérité en ce qu'il prévoit pour toutes les combinaisons possibles d'entrées et produit 1 lorsque les combinaisons souhaitées de 1 et 0 sont fournies aux entrées. Dans la figure 5, l'acheminement des niveaux logiques est contrôlé par 15 sélecteurs connectés en cascade (symboles trapézoïdaux), qui sont en fait des multiplexeurs.

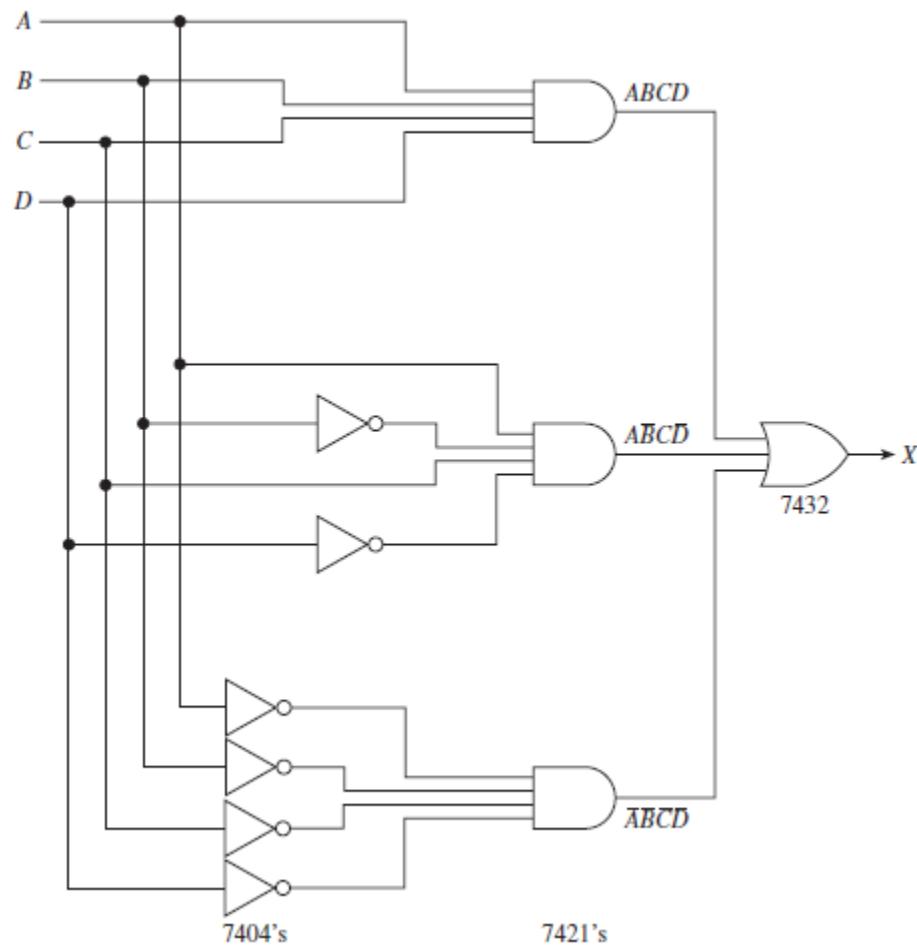


FIGURE 4 – L'équation $X = ABCD + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ implementé en utilisant un CI série 7400

Dans la figure 5 tout ce que nous devons comprendre est que lorsque l'entrée de commande A, B, C ou D est élevée, le niveau logique sur l'entrée VRAI est traversé de gauche à droite. Lorsqu'il est bas, le niveau logique sur l'entrée du complément est traversé. La commande externe de l'entrée A contrôle en fait huit sélecteurs de données : B contrôle quatre, C contrôle deux, et D contrôle un. Cette illustration d'un LUT montre le flux de la logique lorsqu'on a $ABCD = 1010$ et dans ce cas, comme on a $A = 1$, tous les niveaux logiques connectés aux huit A_s VRAIES sont traversés. Par conséquent, simplement en regardant le chemin de données surbrillé sur la figure, un 1 traverse le sélecteur de données B. Maintenant, étant donné que la commande de sélection de l'entrée B est 0, les données passent par l'entrée du complément \bar{B} au sélecteur de données C et Le résultat final de ce trajet est qu'un 1 passe jusqu'à X lorsqu'on a $ABCD = 1010$. Pour confirmer que vous comprenez cette logique, suivez la logique pour les entrées $ABCD = 1111$ et $ABCD = 0000$ pour voir que ces conditions sont également satisfaites.

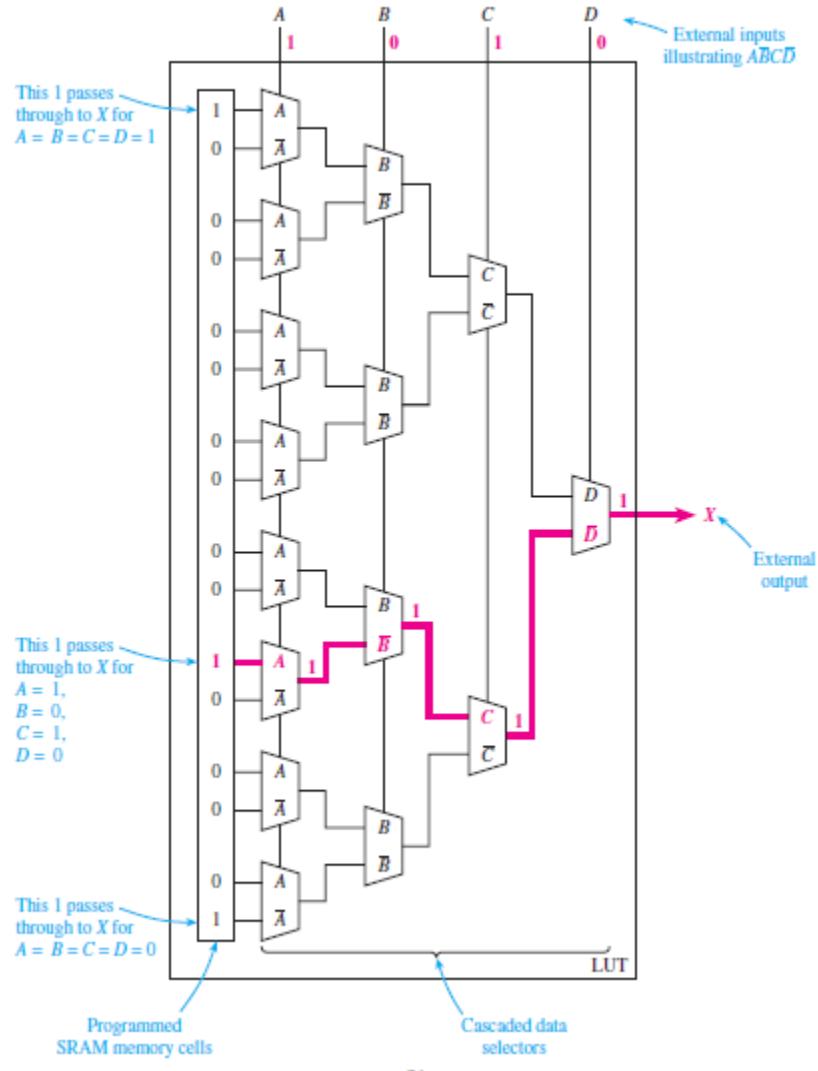


FIGURE 5 – L'équation $X = ABCD + A\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D}$ implementé en utilisant un LUT

Comme vous pouvez le voir, la sortie X dépend des niveaux logiques programmés dans le SRAM³ (mémoire vive statique) des cellules de mémoire du FPGA. Ces cellules de mémoire sont volatils et devront être réinitialisée ainsi que les interconnexions internes et registres, chaque fois que le FPGA est sous tension. Le FPGA qui est sur la carte Altera DE-2 Développement illustré dans la figure 3 est le EP2C35F672C6N . Il contient 33,216 tables de consultation et a 475 broches dédiés à l'entrée / sortie. Pour plus d'information sur les différents cartes Altera, vous pourrez visiter le site altera www.altera.com

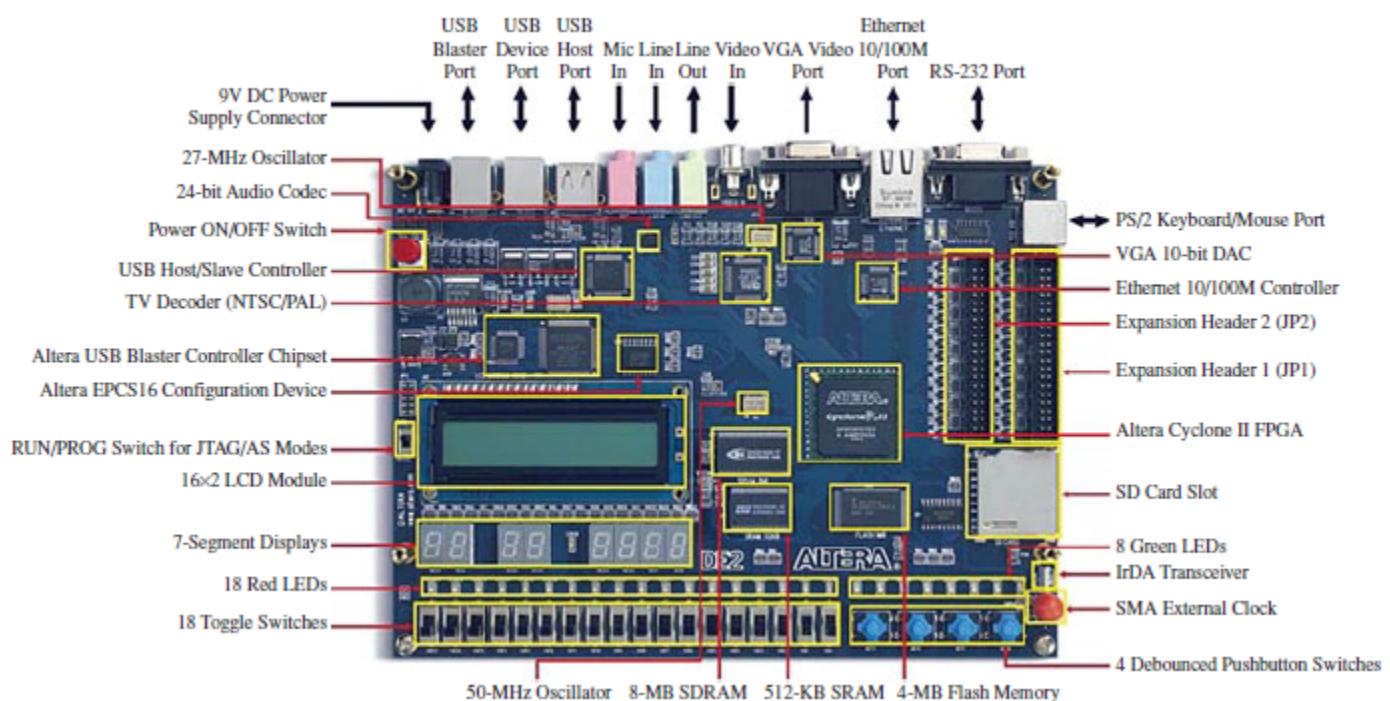


FIGURE 6 – La carte de développement Altera DE2 (Courtesy of Altera Corporation)

3. Static Random Access Memory

2.4 Conception et Simulation des circuits logiques avec un FPGA

Donc, la prochaine question évidente est : "Comment peut-on concevoir des circuits logiques avec un FPGA ?" Pour cela, nous allons utiliser le logiciel Quartus II pour concevoir et simuler nos circuits logiques. Ensuite, nous allons utiliser une carte Altera FPGA comme celui représentée dans la figure 6 pour tester le fonctionnement réel du circuit logique avec des commutateurs et des LEDs. La figure 7 montre le flux d'opérations nécessaires pour concevoir, simuler, et programmer un FPGA. Plusieurs méthodes sont actuellement disponibles pour la conception des circuits logiques, mais nous allons voir les deux méthodes les plus courantes : graphique, et VHDL⁴.

L'éditeur schématique nous permet de connecter des symboles logiques prédéfinis (AND, NAND, OR, etc.) ainsi que les entrées et sorties pour définir l'opération logique que nous allons concevoir. L'éditeur de VHDL est un éditeur de texte qui nous permet de définir la logique dans un environnement de programmation. C'est-à-dire, dans une forme textuelle, nous spécifions les entrées, sorties, et les opérations logiques que nous allons implémenter.

L'étape suivante effectuée est la compilation du programme VHDL en langage machine. Le compilateur interprète les symboles logiques et les déclarations en VHDL et les traduit en un fichier binaire qui peut être utilisé pour synthétiser, puis simuler et programmer le circuit logique dans le FPGA. Le compilateur utilise plusieurs fichiers et bibliothèques VHDL pour obtenir les informations nécessaires pour la compilation. Les rapports sont ensuite générés qui décrivent des choses telles que les affectations d'E/S des broches, le routage interne des signaux dans le FPGA, et les messages d'erreurs. Le logiciel synthétise ensuite notre projet. C'est-à-dire la modélisation de connexions électriques internes du FPGA, qui produira le circuit logique réelle qui sera programmé sur la carte FPGA.

Le simulateur logiciel (Wave Form Simulator) fournit un moyen pour vérifier le fonctionnement de notre circuit logique. Enfin nous allons programmer la carte FPGA et le tester avec des entrées et sorties réelles.

2.5 Structure d'un programme VHDL

Le langage VHDL est, comme son nom l'indique, un langage de description matériel. Le programme VHDL est divisé en trois sections : la [déclaration des bibliothèques](#), la [partie entité](#), et la [partie architecture](#).

Comme dans la plupart des langages informatiques, la première étape du programme est la **déclaration des bibliothèques** qui sont utilisés par notre programme. En VHDL, la bibliothèque standard IEEE (ieee.std logic 1164.ALL) est utilisée le plus souvent par le compilateur VHDL pour traduire les références aux entrées, sorties, et les états logiques utilisées dans le programme.

La **partie entité** définit les ports d'entrées et sorties. En VHDL, il faut noter que le nom d'entité doit correspondre au nom du fichier .vhd.

La **partie architecture** définit les opérations logiques internes qui seront effectuées sur ces ports. Le nom de l'architecture est arbitraire. Pour faciliter la lecture des programmes VHDL, une convention de mise en forme a été mise en place. Fondamentalement, tous les mots en majuscules sont des mots clés VHDL

4. VHDL-Very High Speed Circuit Hardware Description Language

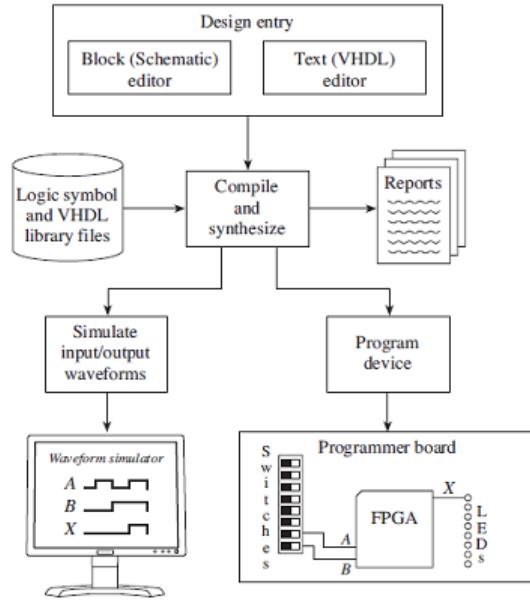


FIGURE 7 – Processus de conception

réservés, et tous les mots/lettres en minuscules sont des variables. Les figures 8 et 9 suivant nous montre respectivement un programme simple en VHDL et le même programme sous forme schématique, représentant l'équation logique $X = A \cdot B$

fig4_13.vhd

```

Library Declaration { LIBRARY ieee;
USE ieee.std_logic_1164.all; } Declare which VHDL library to use

Entity declaration { ENTITY fig4_13 IS
PORT(
    a, b: IN std_logic;
    x: OUT std_logic);
END fig4_13; } Entity name

Architecture body { ARCHITECTURE arc OF fig4_13 IS
BEGIN
    x<-a AND b; } Define the logic
END arc; } Architecture name

Line 17 | Col 1 | INS |
```

FIGURE 8 – Programme VHDL correspondant à l'équation logique $X = A \cdot B$

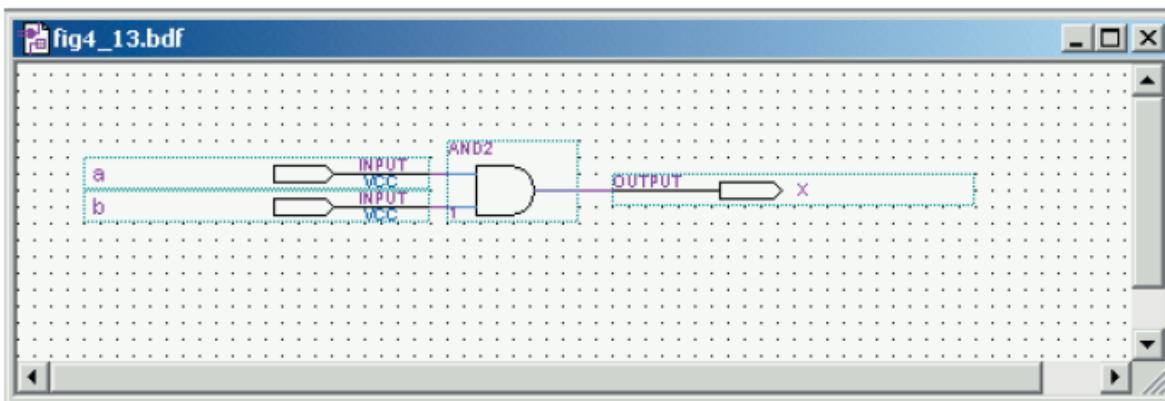


FIGURE 9 – Forme schématique correspondant à l'équation logique $X = A \cdot B$

Vous avez probablement deviné que pour définir l'action d'une simple porte ET, le programme VHDL est plus volumineux que l'entrée graphique, mais nous verrons dans la partie suivant qu'il est beaucoup plus facile d'utiliser le VHDL lorsque les circuits deviennent plus complexes. La figure 10 représente la simulation du circuit précédent avec le logiciel de simulation. Pendant la simulation, le logiciel détermine l'état logique de X pour les combinaisons différentes des entrées et nous montre la sortie(X) dans une forme ondulaire.

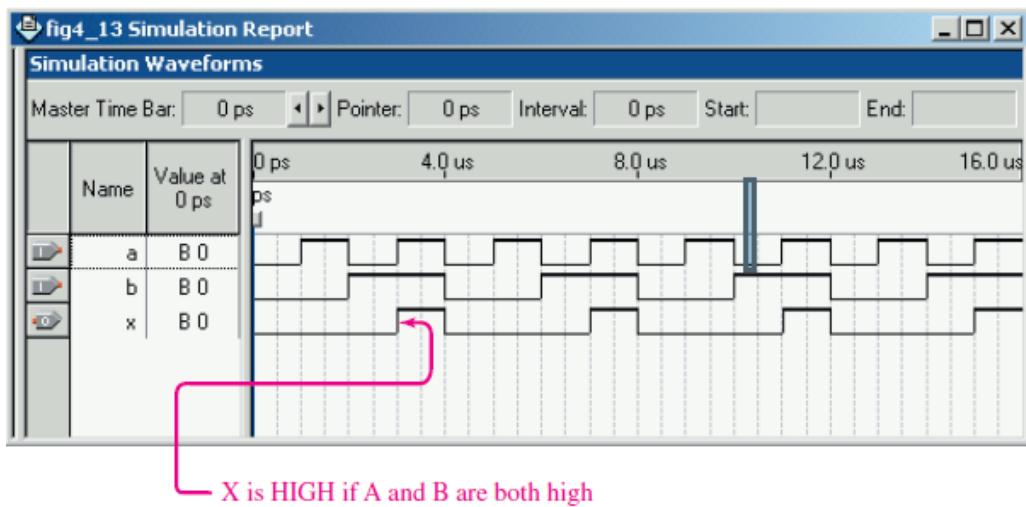


FIGURE 10 – Simulation de l'équation logique $X = A \cdot B$ avec le logiciel Quartus

Deuxième partie

PRATIQUE

3 Tutoriel sur l'utilisation du logiciel Altera Quartus II pour la conception et simulation des circuits logiques

Pour commencer, vous devez d'abord télécharger la version gratuite du logiciel Quartus II . Il existe plusieurs versions disponibles pour téléchargement . La version la plus appropriée (et celui utilisé tout au long de ce tutoriel) est la version 9.1. sp2. La raison d'utiliser cette version est que lorsque Altera a migré de la version 9.1 sp2 à la version 10 , elle a laisser tomber la capacité de créer des fichiers de simulations (fichiers .vWF) . Ces fichiers sont utilisés pour produire les simulations sous forme d'onde à partie du logiciel Quartus II . La raison principale q'un concepteur devrait utiliser la version 10 (et au-delà) est s'il a besoin d'utiliser les CPLD et FPGAs qui ne sont pas pris en charge par les versions antérieures du logiciel . Si vous avez besoin de ce niveau élevé de développement , la version la plus récente du logiciel sera nécessaire. Dans ce cas cependant, Altera recommande l'utilisation d'un autre programme appelé ModelSim pour la simulation des programmes logiques. Tous les versions ultérieures à la version 10 aura cependant une version intégré du logiciel ModelSim. La version 9.1 du logiciel Quartus II peut être téléchargé ici www.altera.com/downloads/download-center.html

Dans ce tutoriel, nous allons implementé une équation booléenne simple($X = A \cdot B + C \cdot D$) pour illustrer les différents étapes nécessaires pour concevoir, simuler et programmer un FPGA en utilisant le logiciel Altera Quartus II.

3.1 Conception Schématique

1. Démarrez le logiciel Altera Quartus II . L'écran principal est illustré dans la figure suivant.

Créer un Nouveau Projet

Toutes nos fichiers seront contenues dans un " projet ". Dans le cadre du projet, nous allons concevoir notre circuit logique en utilisant l'éditeur schématique pour dessiner un schéma ou l'éditeur de texte pour écrire un programme VHDL. Nous allons également créer un fichier de simulation pour le projet pour tester le fonctionnement avant de graver notre programme sur le FPGA.

2. Pour créer un nouveau projet : Cliquez sur **Create a New Project** et ensuite sur **Next**

La première page de l'Assistant Nouveau projet nous demande le dossier, le nom, et l'entité principale de notre projet. La figure suivante nous montre le même onglet mais avec un sous- répertoire nommé alterafiles et un répertoire de travail nommé boolean1. Il est conseillé de mettre chaque nouveau projet dans un nouveau répertoire de travail.

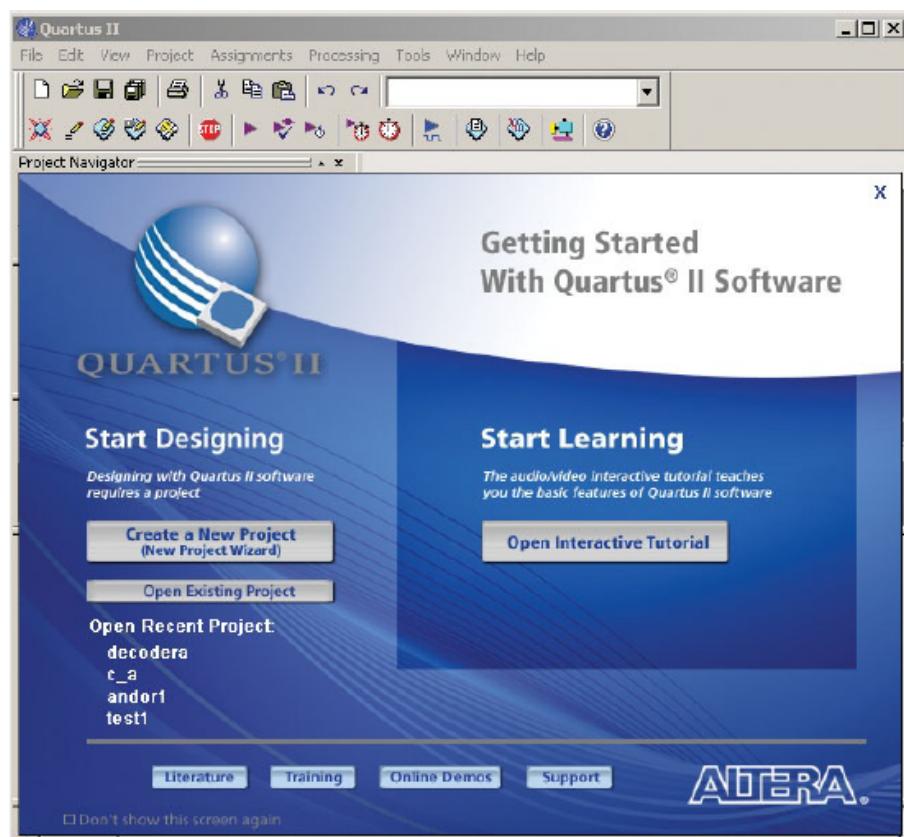


FIGURE i – L'écran principal du logiciel Quartus

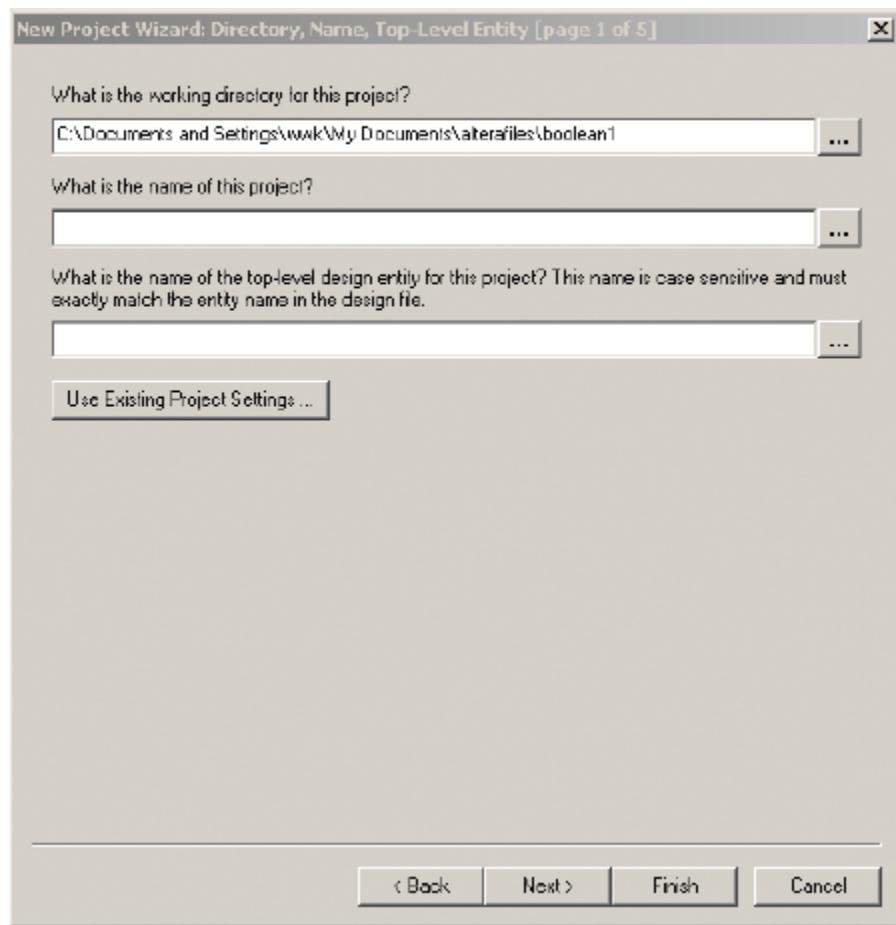


FIGURE ii – L'écran Assistant Nouveau projet-1/5

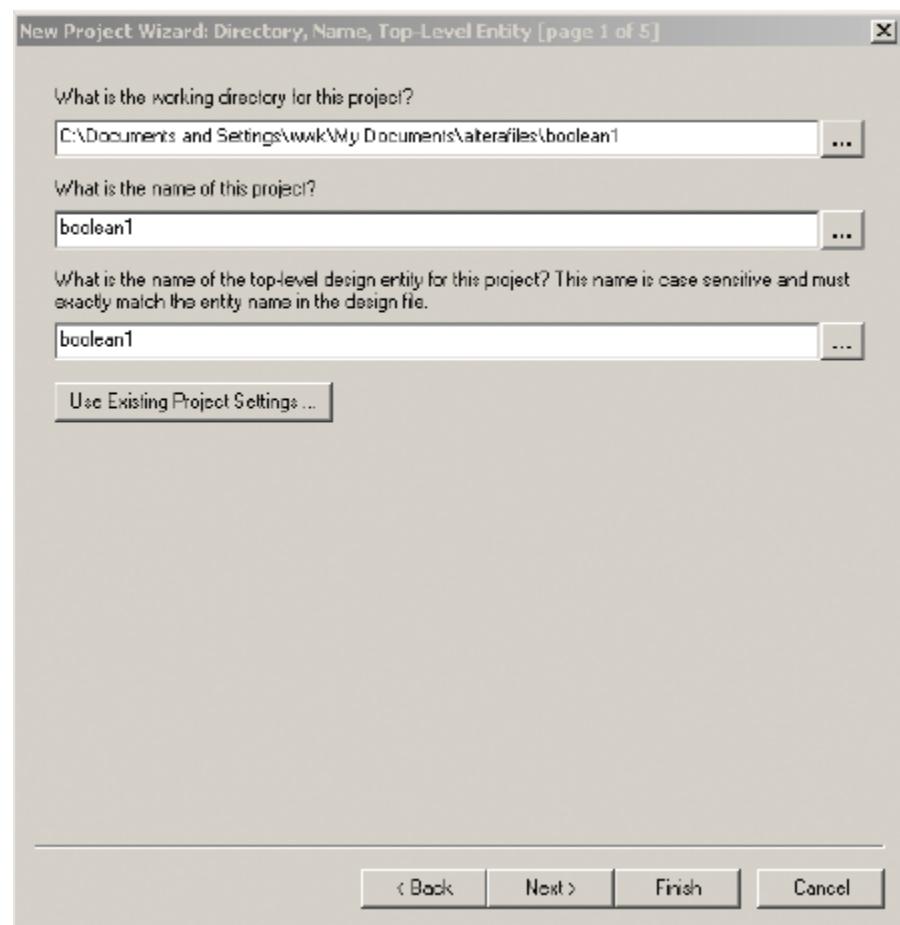


FIGURE iii – L'écran Assistant Nouveau projet-1/5

3. Ensuite, vous devez nommer votre projet et l'entité principale de votre projet. J'ai choisi boolean1 dans ce cas. Notez : le nom boolean1 apparaît sur les trois lignes.Cliquez sur **Next** et **Yes** pour créer la nouvelle sous-répertoire pour notre projet.

4. Le deuxième écran de l'assistant est illustré dans la figure iv. Nous n'avons pas des fichiers supplémentaires pour ajouter, donc cliquez sur **Next**.

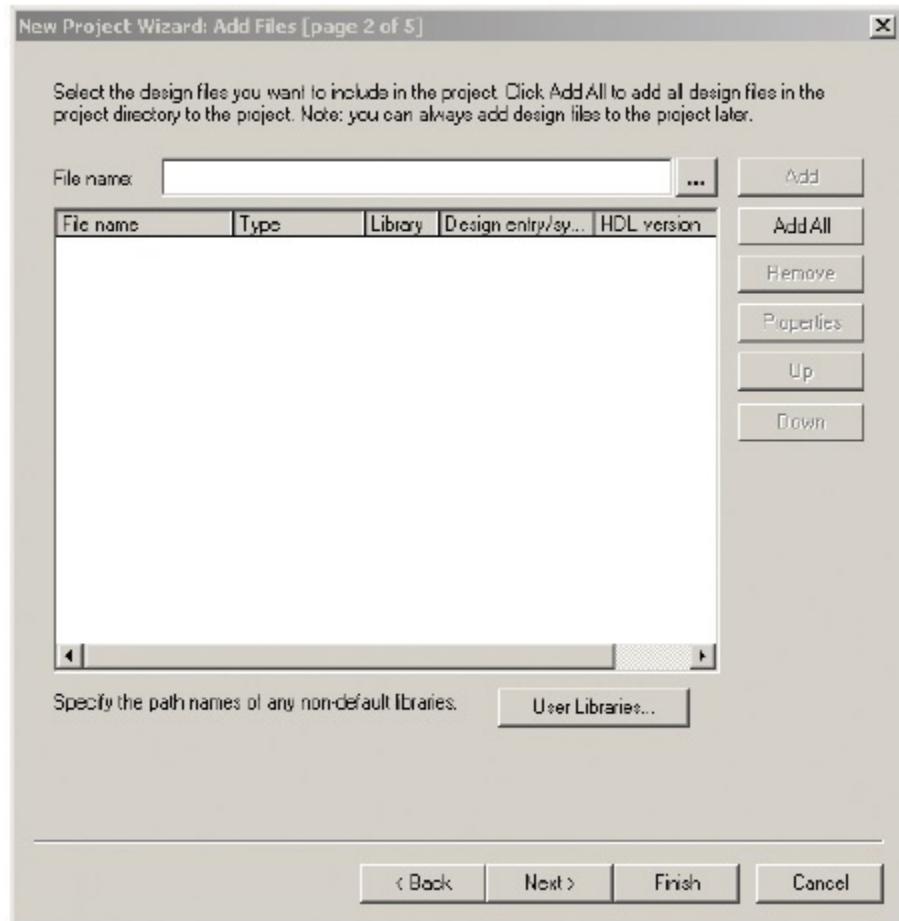


FIGURE iv – L'écran Assistant Nouveau projet-2/5

5. Le troisième écran de l'assistant est illustré dans la figure v. Cet écran nous permettra de spécifier la famille de FPGA ciblé et l'appareil spécifique pour notre projet. La famille utilisé ici est le Cyclone II et l'appareil spécifique est le EP2C35F672C6. Après avoir choisi, cliquez sur **Next**.
6. Le quatrième écran de l'assistant est illustré dans la figure suivante . Nous n'avons pas d'outils EDA supplémentaires à utiliser, donc cliquez sur **Next** pour passer aux cinquième écran.
7. Le cinquième écran de l'assistant est illustré à la figure vii . Cela montre un résumé des tous les choix que nous avons fait. Cliquez sur **Finish** pour terminer l'assistant Nouveau Projet.

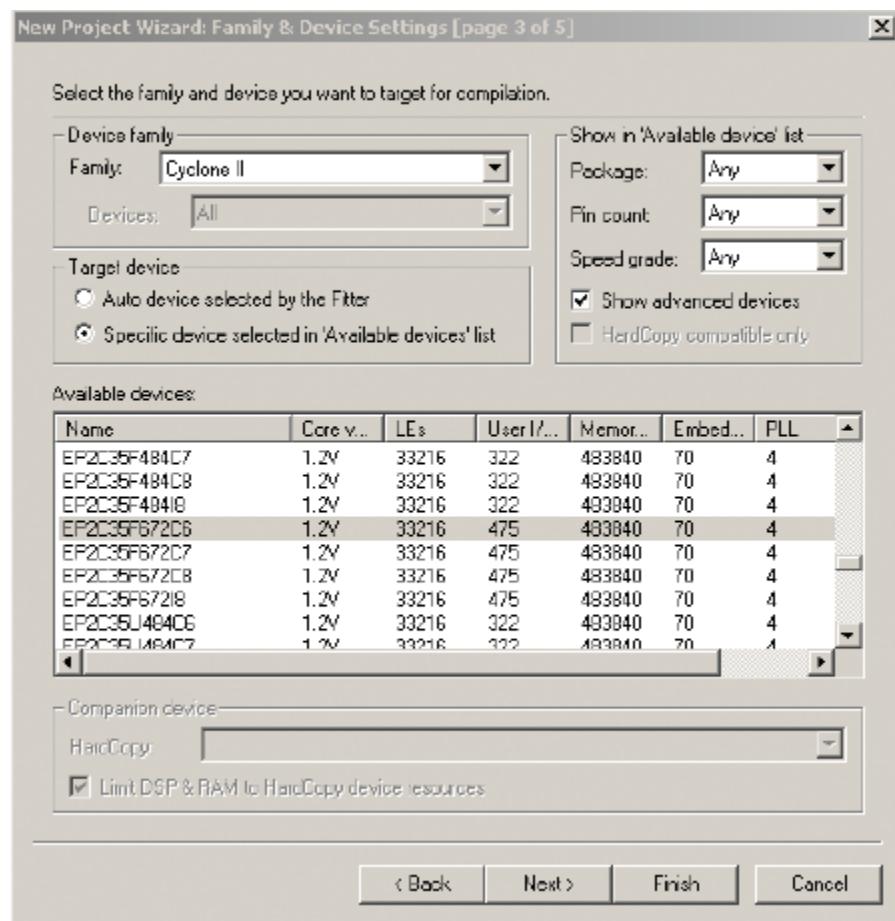


FIGURE v – L'écran Assistant Nouveau projet-3/5

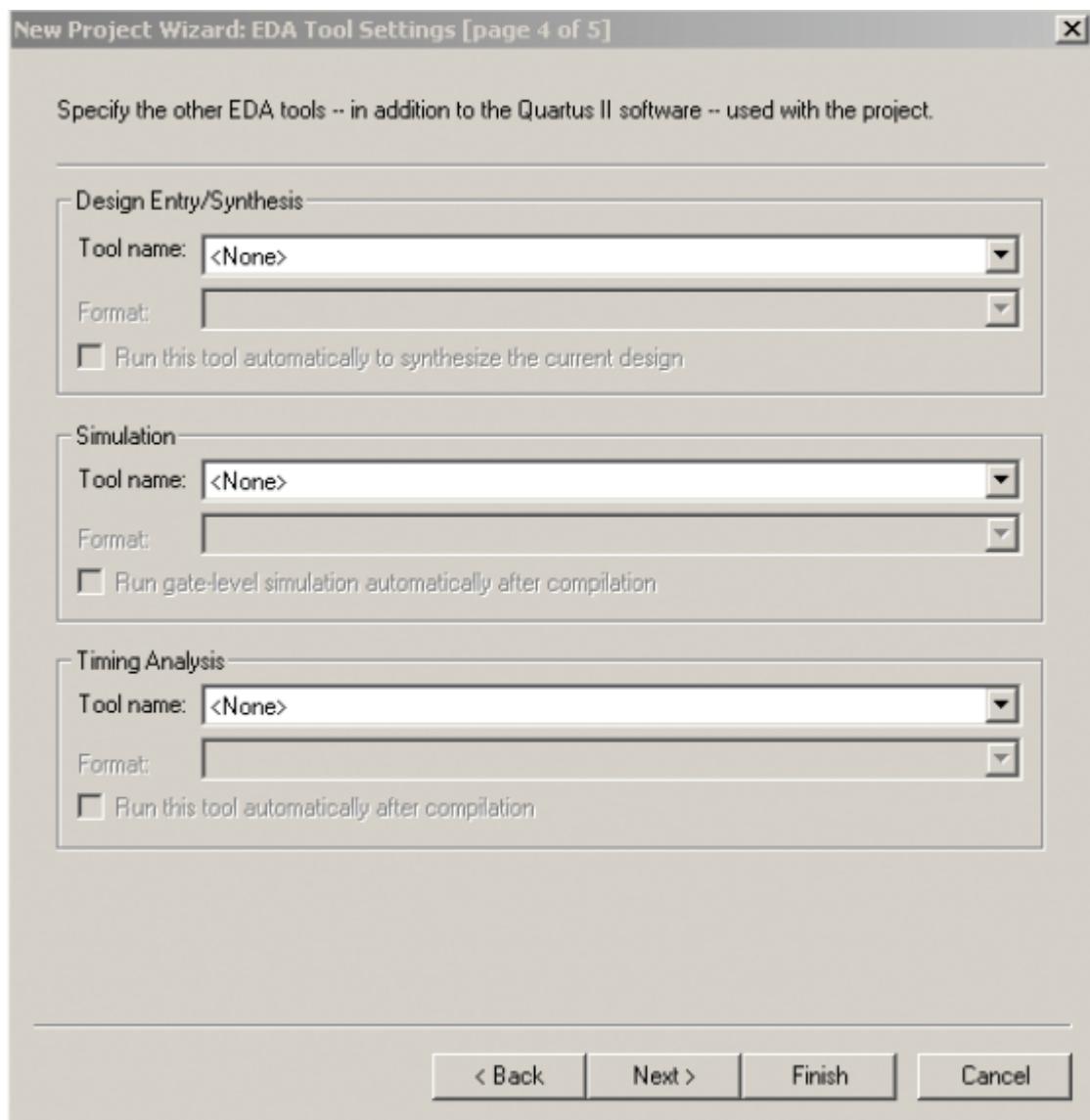


FIGURE vi – L'écran Assistant Nouveau projet-4/5

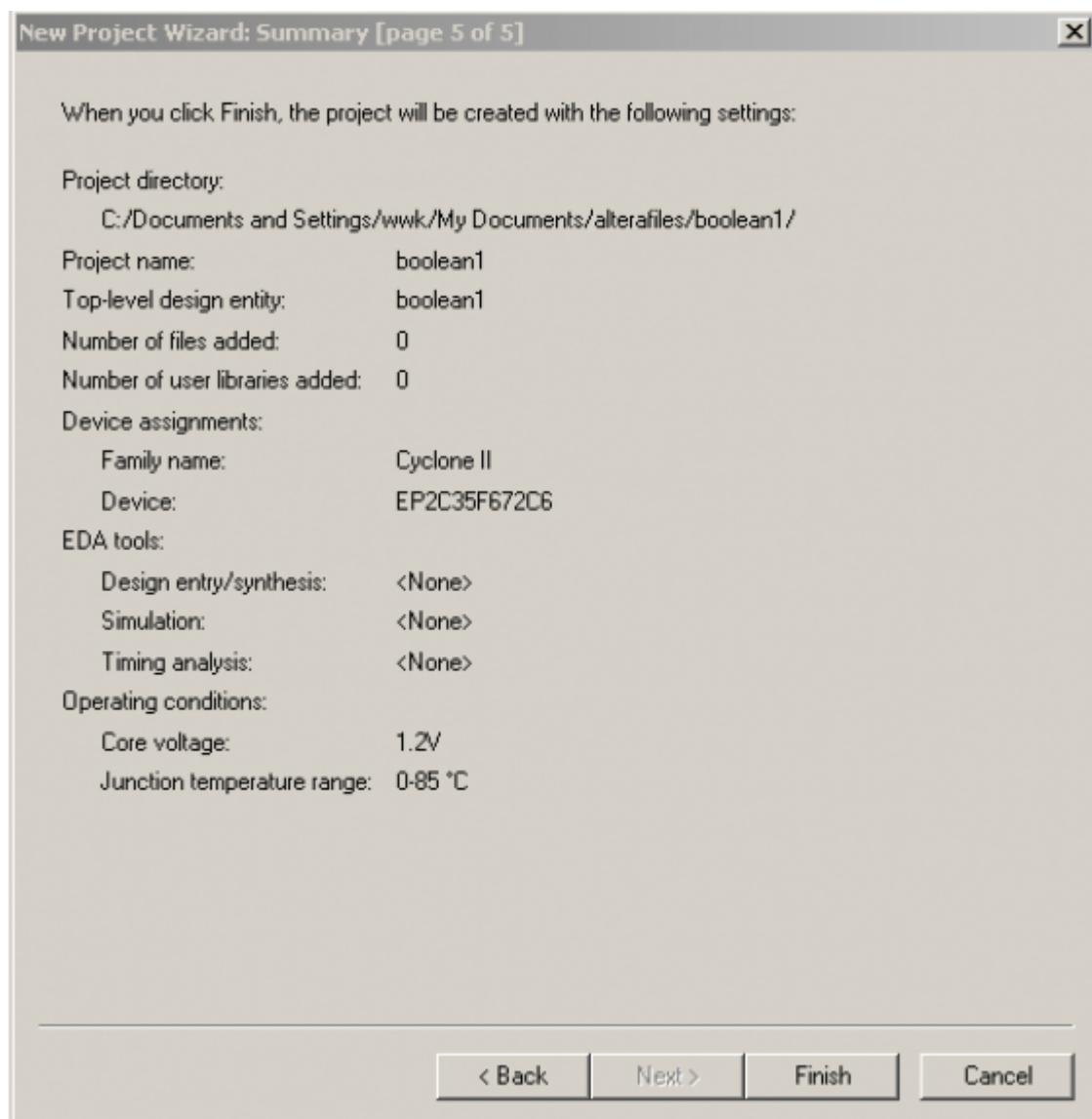


FIGURE vii – L'écran Assistant Nouveau projet-5/5

Créer un Nouveau Fichier bdf⁵

8. Pour dessiner le circuit logique pour notre équation booléenne , nous allons utiliser l'éditeur schématique pour créer un fichier bdf qui va contenir le diagramme logique de notre circuit. Cliquez sur **File → New** (Figure viii).
9. Sélectionnez **Block Diagram/Schematic File** et appuyez sur **OK**. Un espace de travail vierge apparaît . Nous allons dessiner notre circuit logique numérique dans cet espace de travail.
10. Avant de dessiner le circuit logique , nous devons nommer le fichier bdf et l'enregistrer. Choisissez **File → Save As** et entrez le nom du fichier comme boolean1. Ajouter ensuite ce fichier au projet courant avec l'option **Add file to current project**. Appuyez sur **Save** pour enregistrer votre projet (Figure ix).

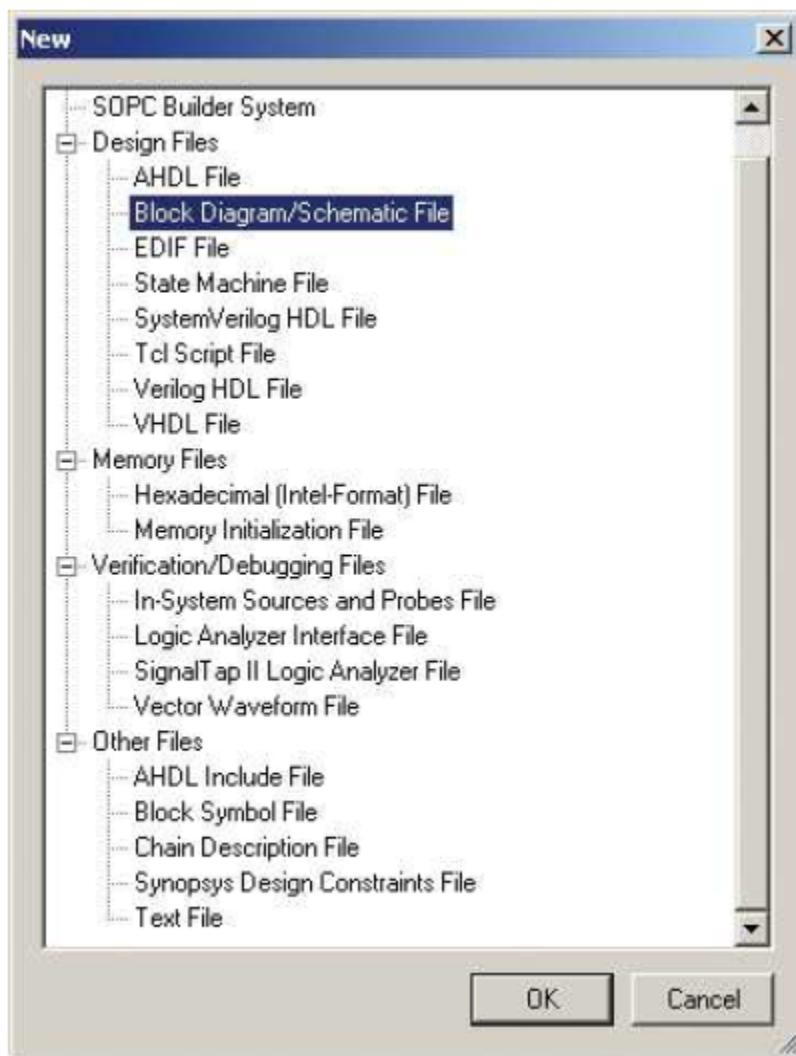


FIGURE viii – Crédit d'un nouveau fichier schématique

5. bdf : Block Diagram File

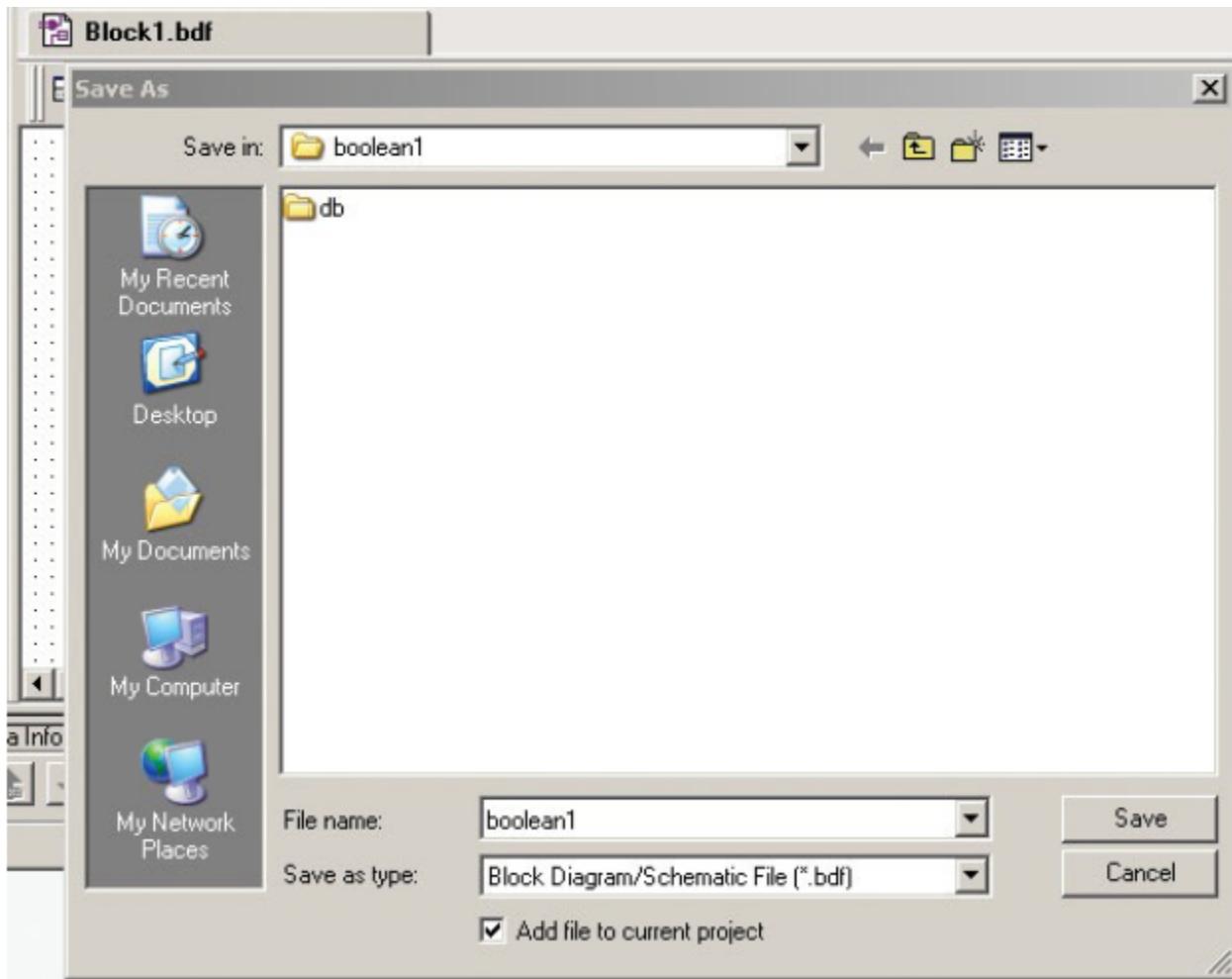


FIGURE ix – Ajouter le fichier bdf au projet courant et enregistrer

Dessiner le Circuit Logique correspondant à l'équation logique

11. Faites un clic droit dans l'espace de travail vide. Choisissez **Insert** → **Symbol** pour insérer un symbole et tapez **and2** dans le champ **Name** et appuyez sur **OK**.
12. Déposez la porte **and2** dans l'espace de travail du fichier *bdf* en déplaçant votre souris à une emplacement appropriée et en appuyant sur le bouton gauche de la souris.

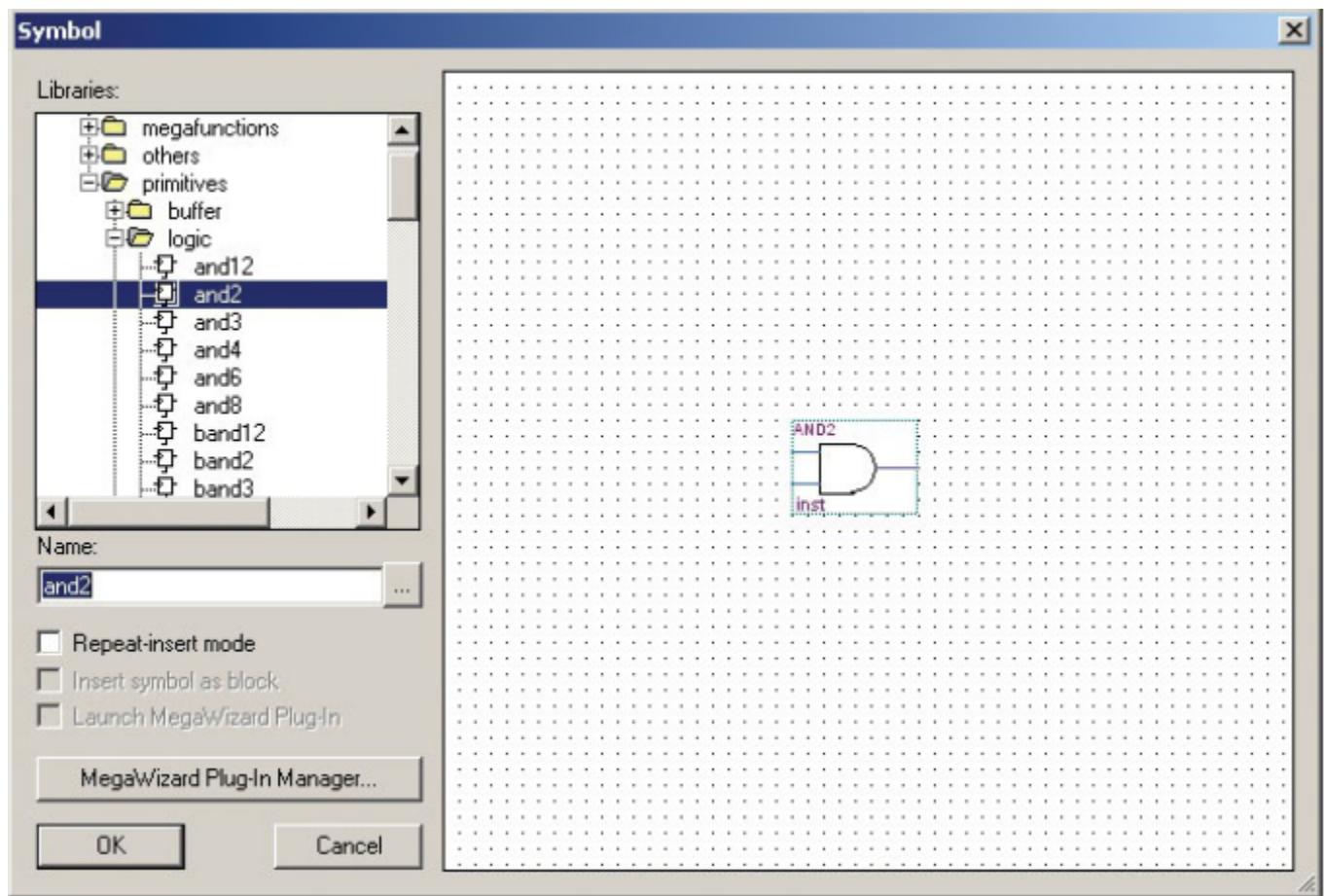


FIGURE x – Ajoute d'une porte logique ET à deux entrées

13. Pour implementer l'équation logique $X = AB + CD$ nous aurons besoin d'un total de deux portes ET et une porte OU. Répétez les étapes 11 et 12 pour les autres entrées. Nous devons également fournir quatre broches d'entrée pour A , B , C et D et une broche de sortie pour X. Répétez les étapes 11 et 12 pour quatre broches d'entrée (input) et une broche de sortie (output). L'espace de travail bdf devrait maintenant ressembler à la figure xi.

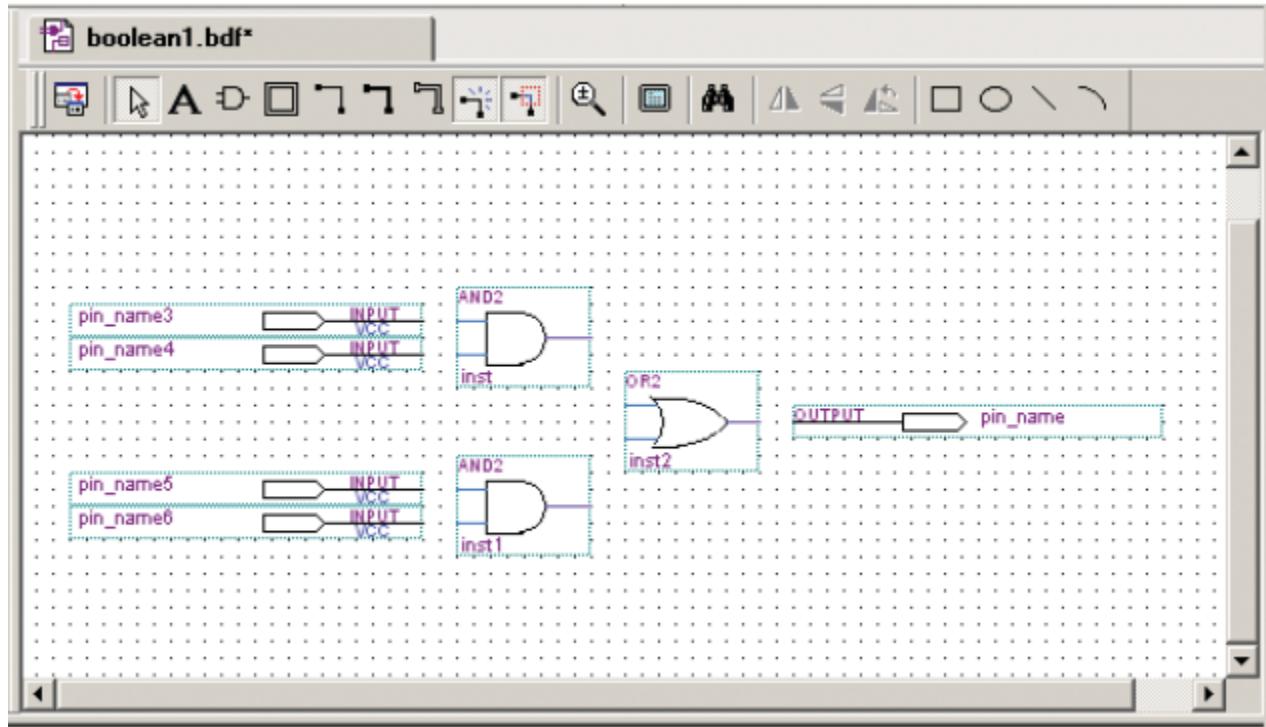


FIGURE xi – Fichier bdf avec les broches d'entrée/sortie insérés

Faire les Connexions pour le Circuit Logique

14. Avant de faire les interconnexions, les noms des broches doivent être assignées. Pour le faire, double-cliquez sur le **pin-name** et entrer le nom de la broche. (Note : Nous utilisons des lettres minuscules a,b,c,d et x pour les noms des broches d'entrée/sortie ce qui est en cohérence avec la convention utilisée dans le langage VHDL).
15. Nous allons maintenant faire les interconnexions du circuit. Lorsque vous déplacez le pointeur de la souris près du point de toute symbole d'entrée ou de sortie, le pointeur devient automatiquement une croix. Appuyez et maintenez le bouton gauche de la souris et glisser la souris pour créer une connexion entre deux symboles. Après avoir fait cela, votre figure doit ressembler à celui de la figure xii.

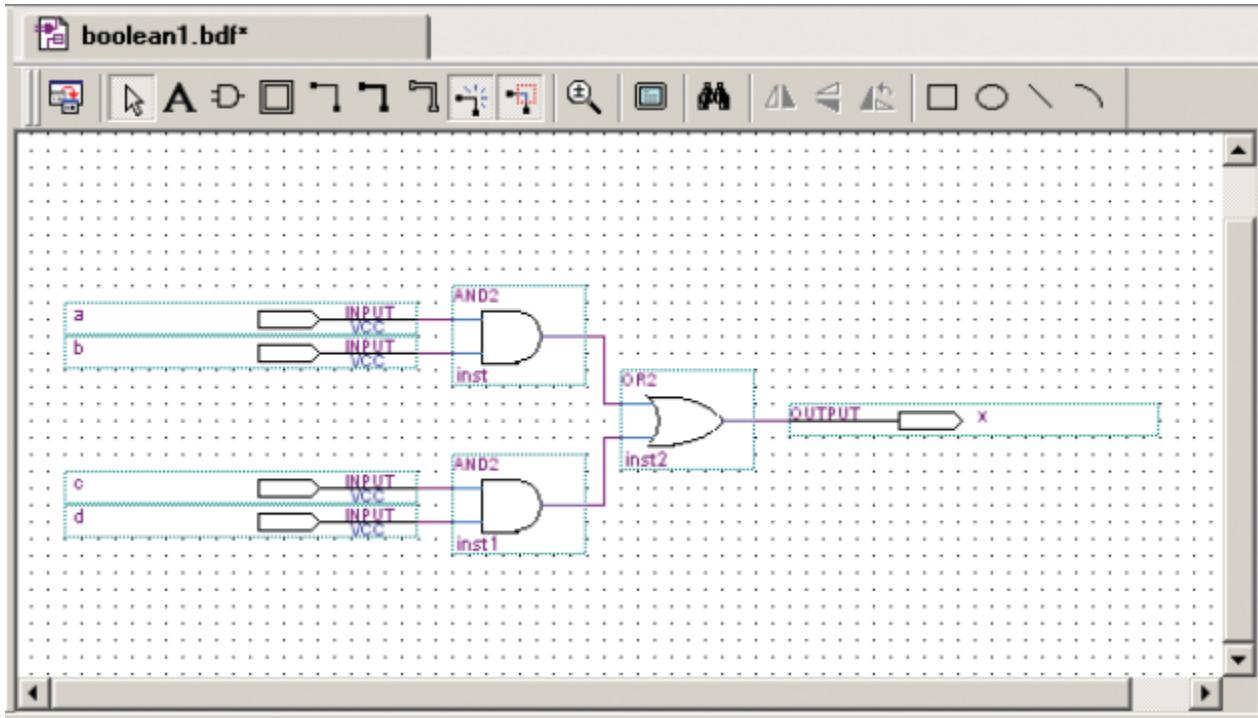


FIGURE xii – Le fichier bdf terminé

16. Pour enregistrer notre fichier *bdf*, cliquer sur **File** → **Save**. (Remarquez que l’astérisque est retiré du nom de fichier.)

Compilation du Projet

17. Maintenant, nous allons compiler le projet. Dans cette étape, le logiciel Quartus II effectue une analyse et une synthèse du fichier *bdf* pour assurer qu’il n’y a pas d’erreurs dans notre circuit logique. Le logiciel adapte ensuite notre circuit à un modèle FPGA-EP2C35F672C6. Pour démarrer la compilation : Cliquer sur **Processing** → **Start Compilation**. La compilation prend quelques secondes. A la fin de la compilation, le compilateur nous envoie un message : "Full Compilation Successful" si la compilation a réussi, et les messages d’erreurs sinon. (Les avertissements seront résolus plus tard, quand nous allons définir les numéros de broches d’entrée/sortie. (Voir la figure xxvi)). Cliquer sur **OK**.

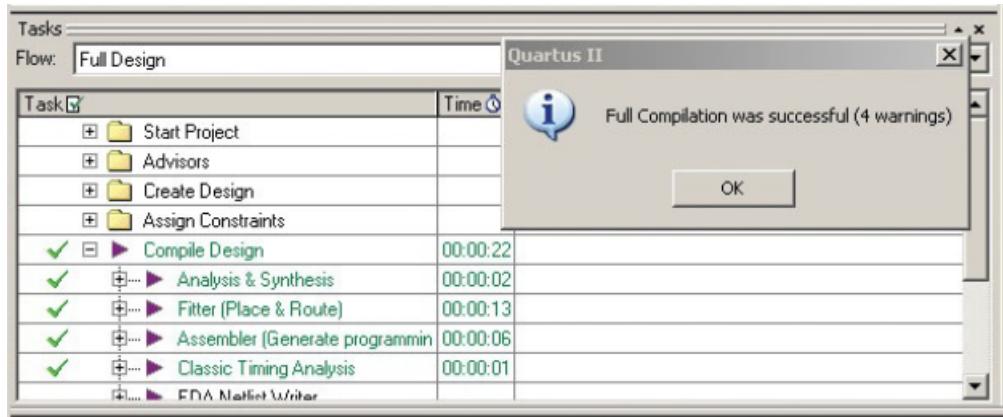


FIGURE xiii – Résultats de Compilation

Creation du fichier de Simulation

18. Le fichier de simulation fournit un moyen de simuler notre circuit logique en une forme ondulatoire en utilisant toutes les combinaisons possibles des entrées pour a, b, c et d pour produire la sortie x. Pour créer un fichier de simulation : Cliquer sur **File** → **New** → **Verification/Debugging Files** → **Vector WaveForm File** → **OK**. (Voir la figure xiv)
19. Avant de simuler le circuit, nous devons nommer le fichier **vwf** et l'enregistrez comme un fichier de notre projet. Cliquez sur **File** → **Save As** et entrez le nom du fichier, *boolean1* dans mon cas. Ajoutez le fichier à notre projet en cochant la case marqué : **Add file to current project**. Cliquez sur **Save** pour enregistrer le fichier. (Voir la figure xv)

5. Tous les fichiers de simulation dans ce tutoriel ont été créés avec la version 9.1 de Quartus sp2 . Une autre alternative est d'utiliser le logiciel **ModelSim**. Cela nécessiterait la création d'un fichier appelé un testbench en VHDL, qui est possible si vous avez une bonne compréhension du langage VHDL

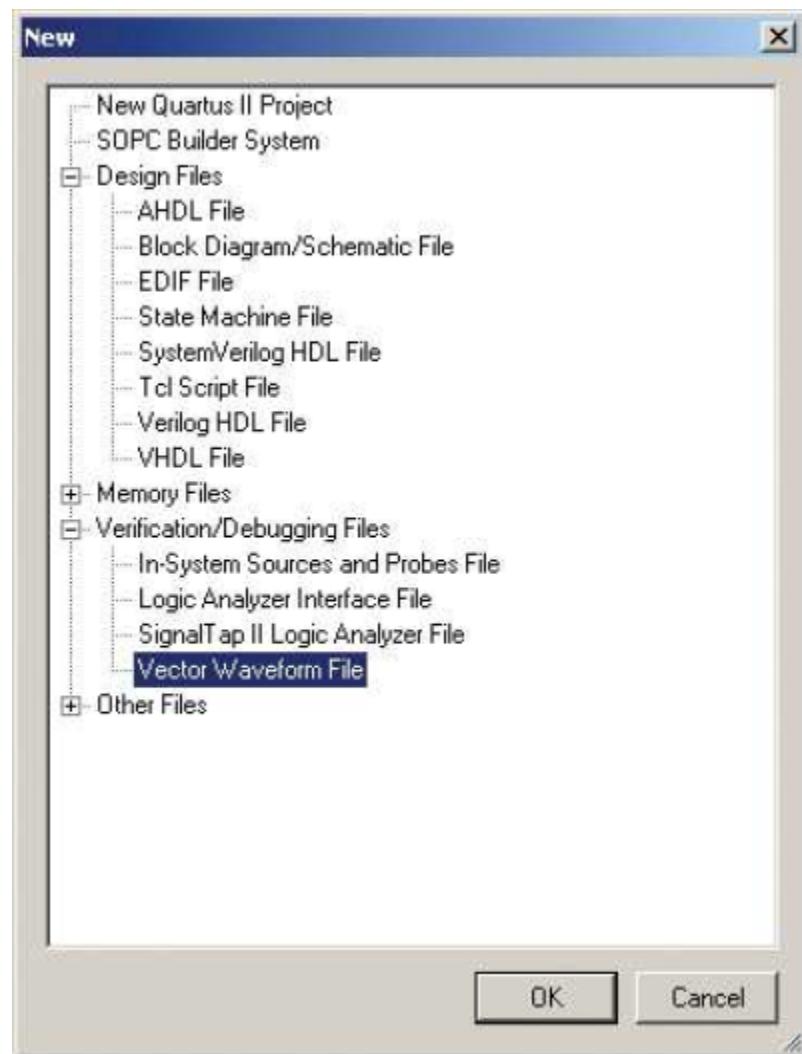


FIGURE xiv – Création d'un fichier de simulation vwf

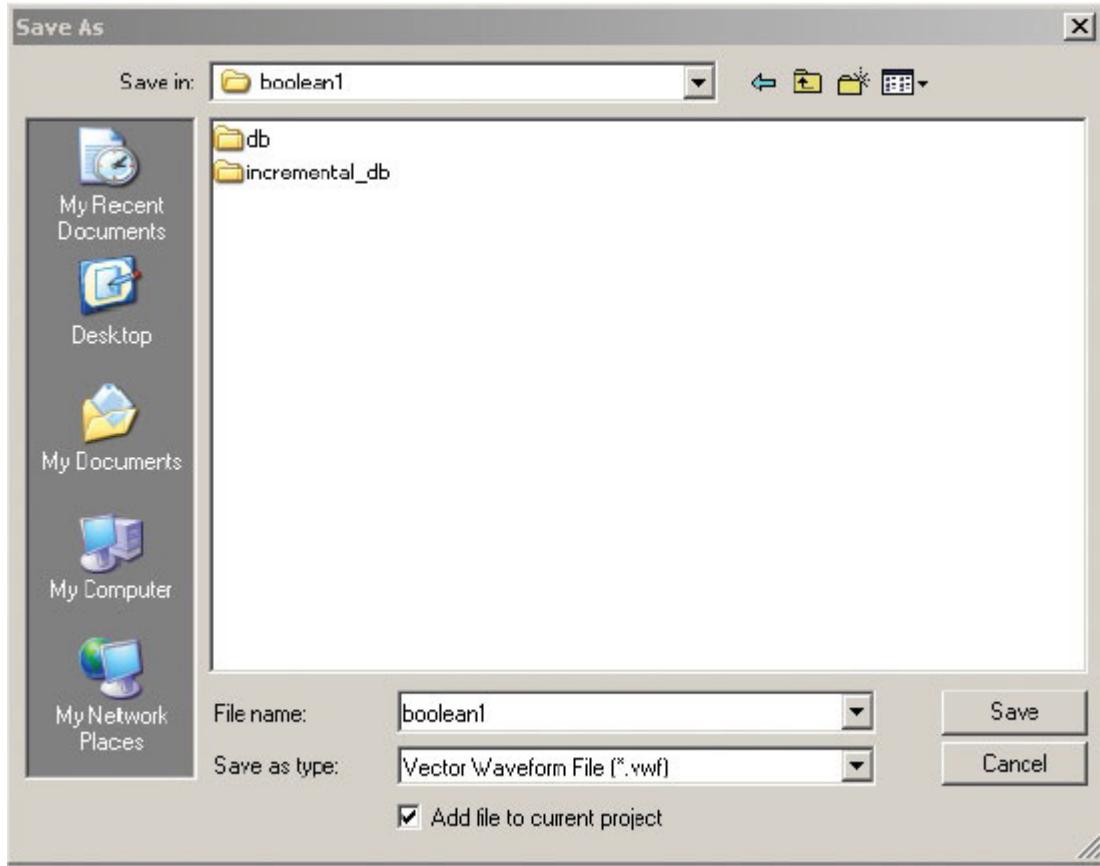


FIGURE xv – Enregistrer le fichier de simulation comme un .vmf

20. Pour créer le fichier de simulation nous devons d'abord spécifier la durée de simulation ($16\mu s$ dans mon cas) et la période des ondes de simulation ($1\mu s$ dans mon cas). Pour faire cela : Cliquez sur **Edit** → **End time** → **16** → **us** → **OK**. Ensuite : Cliquez sur **Edit** → **Grid Size** → **Period** → **1** → **us** → **OK** (Voir les figures vi et vii.)



FIGURE xvi – Spécifier la durée de simulation

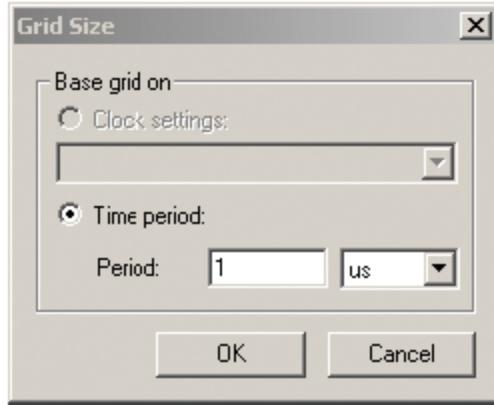


FIGURE xvii – Spécifier la période des ondes de simulation

- Pour voir tout l'intégralité de l'affichage, cliquer sur : **View → Fit In Window**. Ton écran *vwf* doit maintenant ressembler à celui de la figure xviii.

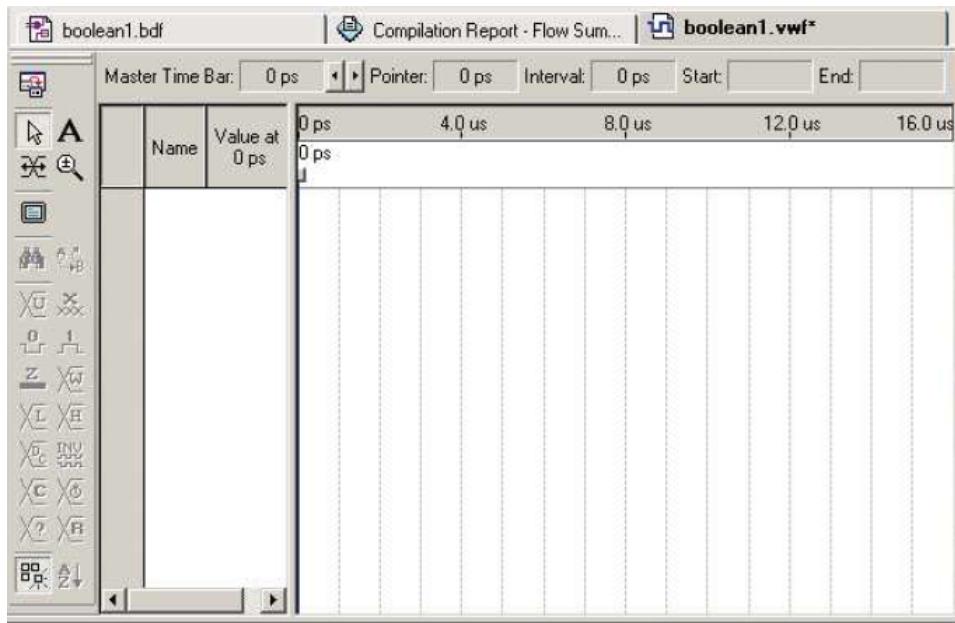


FIGURE xviii – L'écran vwf avec une période de 1us et un temp de simulation de 16us

Ajout des Entrées et Sorties à l'affichage

- Nous devons maintenant ajouter les entrées et les sorties de notre circuit à l'affichage. Le logiciel Quartus II fournit un utilitaire pour faire cela ; cet outil est appelée le **Node Finder**. Cliquer sur : **View → Utility Windows → Node Finder**. (Voir la figure xix.)

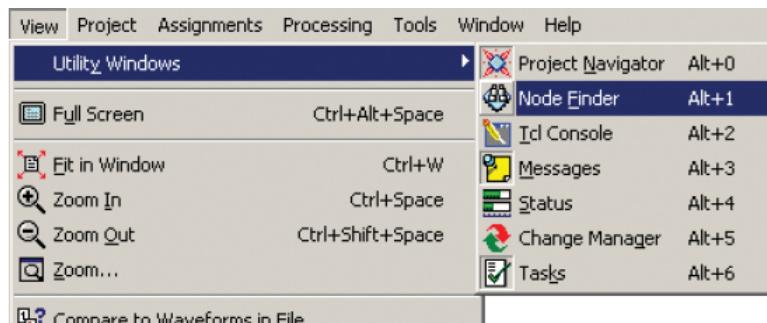


FIGURE xix – L'utilisation du Node Finder pour lister les entrées et sorties du circuit

23. Dans l'écran du Node Finder, choisissez **Filter : Design Entry (All Names)**. Appuyez sur **List**.
L'affichage doit ressembler à celui dans la figure xx

The screenshot shows the 'Node Finder' dialog box. At the top, there are fields for 'Named:' (set to '*'), 'Filter:' (set to 'Design Entry (all names)'), and a 'List' button. Below these are fields for 'Look in:' (set to 'boolean1') and 'Include subentities' (checked). A magnifying glass icon is also present. The main area is titled 'Nodes Found:' and contains a table with the following data:

Name	Assignments	Type	Creator
a	Unassigned	Input	User entered
b	Unassigned	Input	User entered
c	Unassigned	Input	User entered
d	Unassigned	Input	User entered
inst	Unassigned	Combinational	User entered
inst1	Unassigned	Combinational	User entered
inst2	Unassigned	Combinational	User entered
x	Unassigned	Output	User entered

FIGURE xx – L'écran du Node Finder énumérant toutes les entrées et sorties du projet

24. Ensuite, nous allons utiliser la souris pour glisser et déposer les entrées et de sorties de l'écran du Node Finder à l'écran boolean1.vwf. Vous pouvez sélectionner tout les entrées et sorties en utilisant la touche **CTRL** et le **bouton gauche** de la souris, pour glisser tous à la fois. (Voir la figure xxi).

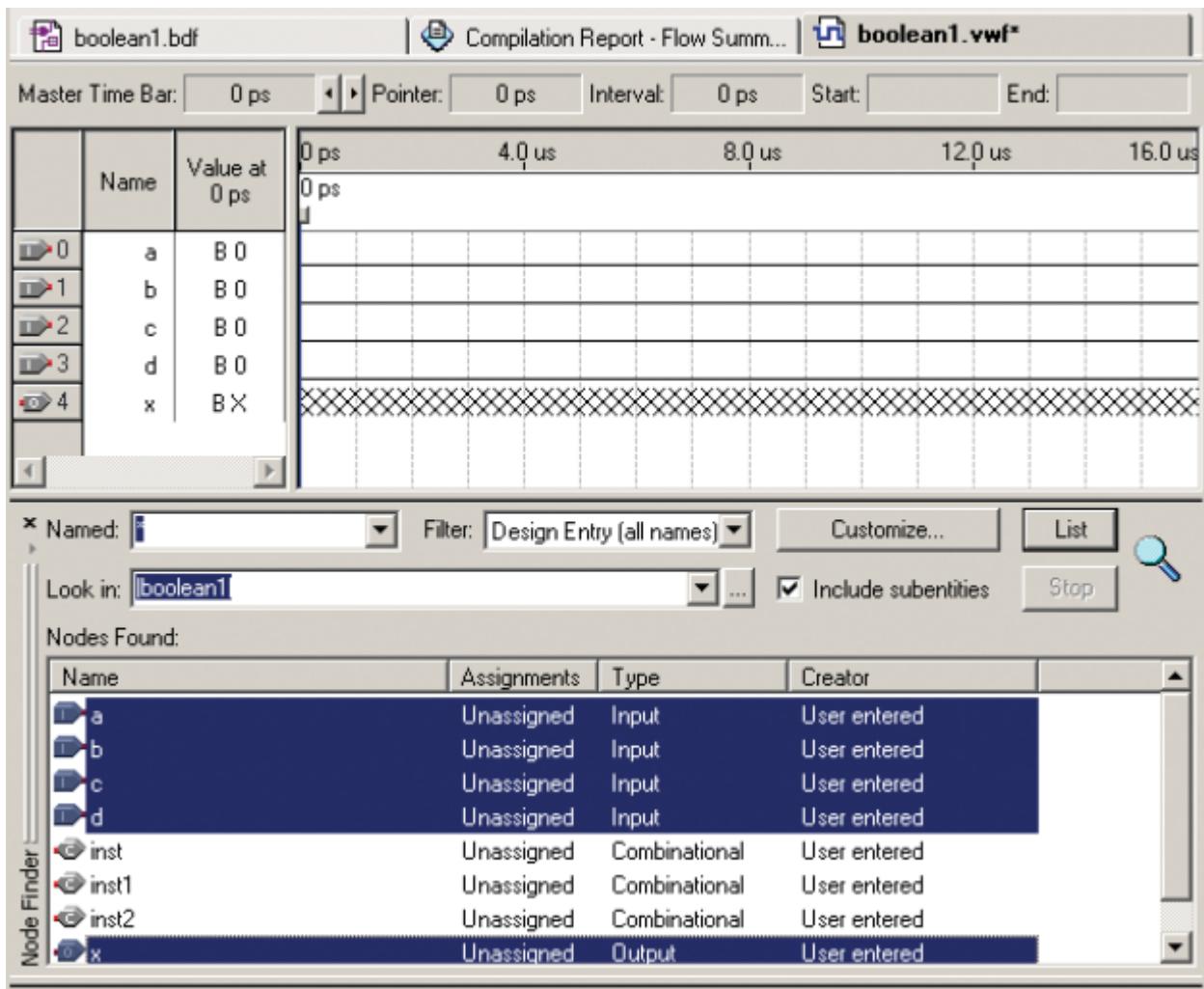


FIGURE xxi – Ajout des entrées et sorties à l'écran vwf

Création des formes d'onde pour les Entrées et Sorties

25. Pour tester le fonctionnement du circuit logique, nous devons créer des formes d'onde pour les 16 combinaisons différents des entrées. La meilleure façon de le faire est de créer un compteur binaire qui compte de 0000 à 1111 (ie 0 à 15 en décimal).

Dans l'écran *vwf*, fait un clic gauche sur la première entrée, a.

Cliquez sur **Edit → Value → Clock**. Entrer une période de 2 μ s. Cliquez sur *OK*. La forme d'onde de a doit ressembler à celui de la figure xxii.

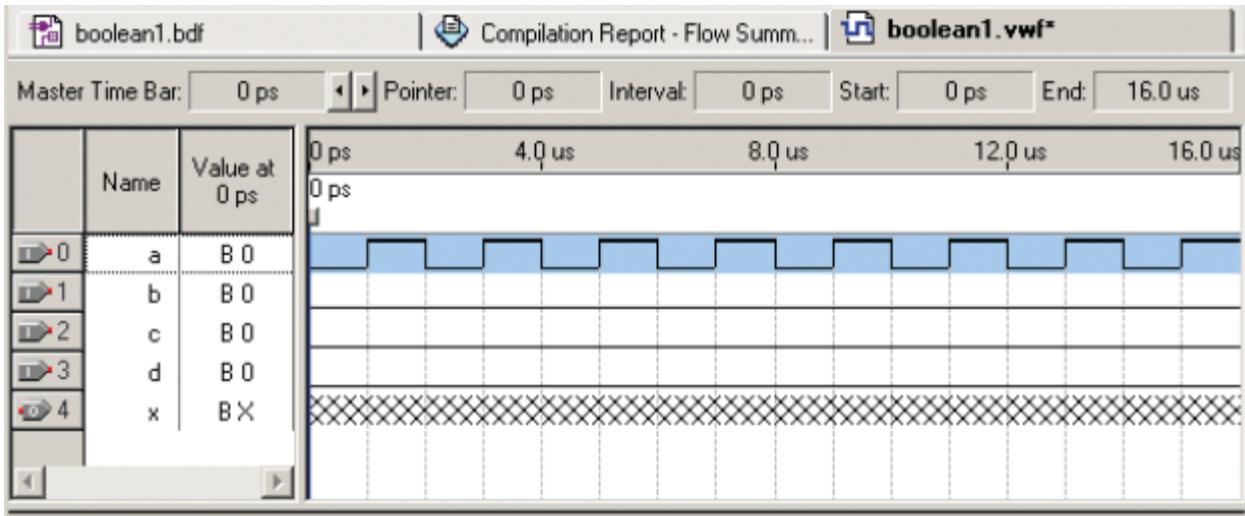


FIGURE xxii – Forme d’onde de l’entrée a

26. Pour créer la forme d’onde de l’entrée b, fait un clic gauche sur l’entrée b.

Cliquez sur **Edit** → **Value** → **Clock**. Entrez une période de $4 \mu\text{s}$ et cliquez sur *OK*.

27. Répétez cette procédure pour les entrées c et d mais en utilisant des périodes de 8 et 16 μs respectivement. L’écran *vwf* final doit ressembler à celui de la figure xxiii.

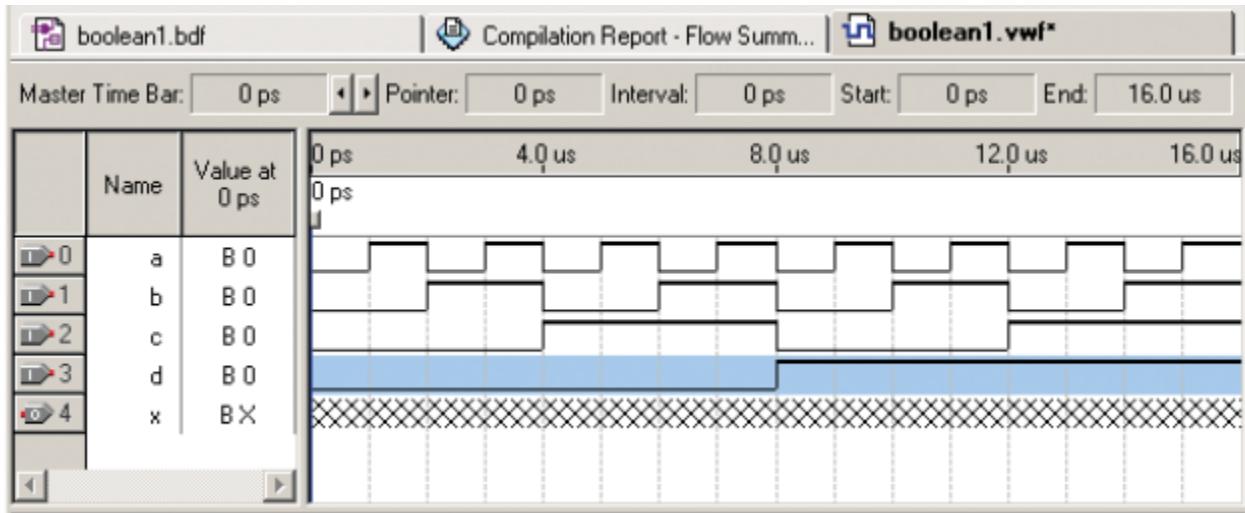


FIGURE xxiii – L’écran vwf final

28. Enregistrez le fichier *vwf* :

Cliquez sur **File** → **Save**. (Remarquez l’astérisque est retiré du nom du fichier *vwf*.)

Effectuer une simulation fonctionnelle du la sortie x.

29. Maintenant qu’on a les niveaux logiques des entrées défini, le logiciel Quartus II peut les utiliser pour déterminer le niveau logique x pour chaque combinaison d’entrées. Pour visioner la simulation :

Cliquer sur **Assignments** → **Settings**.

Ensuite, sur le côté gauche de la fenêtre dans la figure xxiv, choisissez **Simulator Settings**, et pour le mode de simulation (**Simulation Mode**), choisissez **Functional** → **OK**.

Nous devons maintenant créer un fichier *netlist* pour activer la simulation :

Cliquez sur **Processing** → **Generate Functional Simulation Netlist** → **OK**

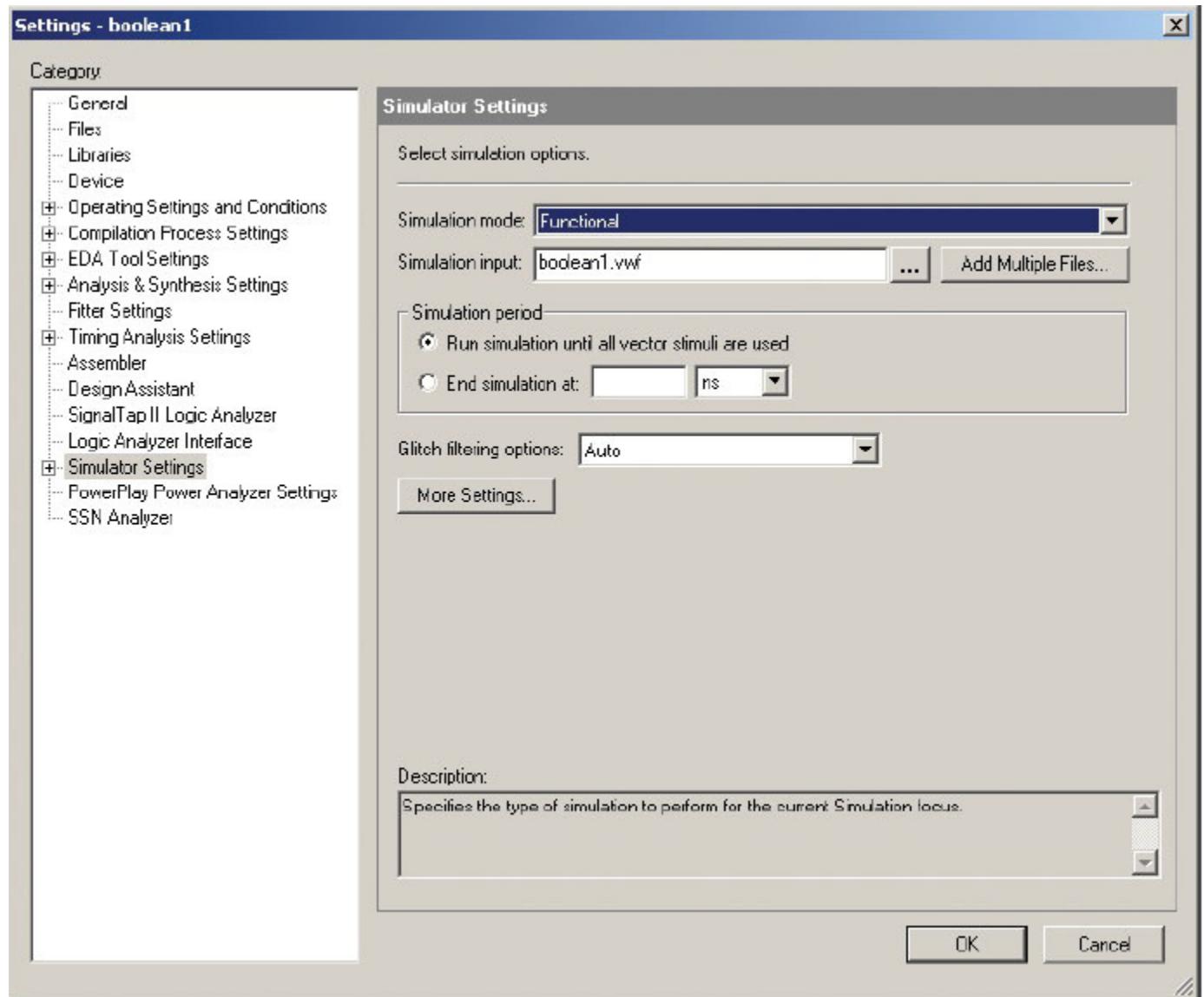


FIGURE xxiv – Spécification du mode de simulation

30. Pour lancer la simulation :

Cliquer sur **Processing** → **Start Simulation**. Après quelques secondes, le message **Simulation Successful** devrait apparaître. Appuyer sur **OK**. Selon l'équation booléenne, X devrait être 1 si A ET B sont à la fois 1 ou si C ET D sont à la fois 1. Étudiez les formes d'onde pour vous assurer que la simulation montre un résultat valide.

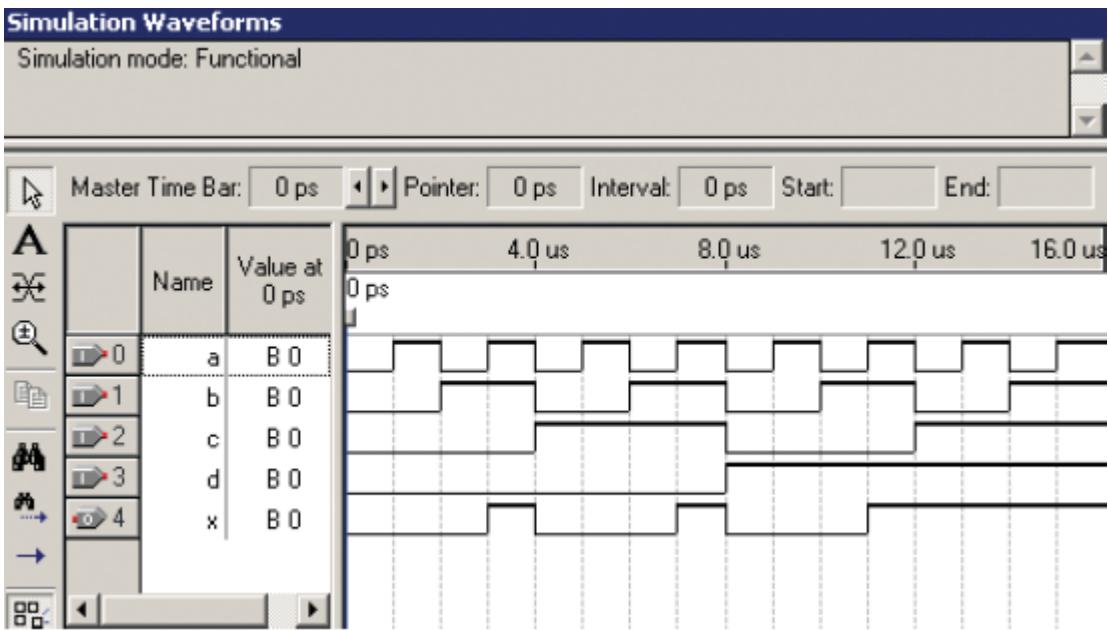


FIGURE xxv – Résultats finaux de simulation

Programmation de la carte Altera-FPGA.

La prochaine étape dans notre processus de développement est de graver notre circuit logique sur un FPGA réelle et tester son fonctionnement en utilisant les commutateurs comme les entrées et un LED comme la sortie. La carte utilisée ici est la carte Altera DE2. Cette carte contient un FPGA Altera EPC2C35F672C6 et d'autres périphériques d'E/S.

L'attribution des Broches :

- Auparavant, notre compilateur a attribué des broches arbitraires à nos entrées a, b, c et d, et à la sortie x. Notre carte DE2 a plusieurs interrupteurs, boutons, et LEDs connectés aux broches spécifiques sur le FPGA. Par conséquent, nous devons attribuer des numéros de broches spécifiques à nos entrées et sortie. Le tableau suivant présente une liste partielle des connexions de broches sur le FPGA qui sont connectés directement aux E/S de la carte DE2 . (Une liste complète est fournit dans le manuel utilisateur de la carte DE2 en tant que fichier .csv (Excel)).

TABLE 4-1 EPC2C35F672C6 FPGA Pin Assignments to the DE2 Board (Partial List)

Input Switches		Output LEDs	
Switch Name	FPGA Pin Number	LED Number	FPGA Pin Number
SW0 <i>A</i>	N25	LEDR0 <i>X</i>	AE23
SW1 <i>B</i>	N26	LEDR1	AF23
SW2 <i>C</i>	P25	LEDR2	AB21
SW3 <i>D</i>	AE14	LEDR3	AC22
SW4	AF14	LEDR4	AD22
SW5	AD13	LEDR5	AD23
SW6	AC13	LEDR6	AD21
SW7	C13	LEDR7	AC21

FIGURE xxvi – L'attribution des broches

La figure xxvii nous montre les interrupteurs et LEDs qu'on va utiliser pour nos entrées et sortie.

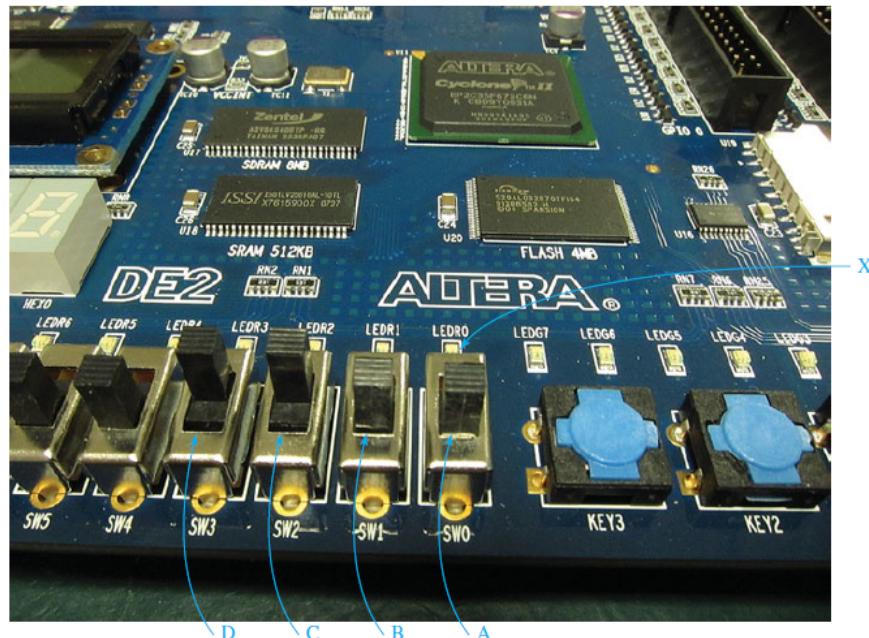


FIGURE xxvii – La carte Altera DE2 avec les entrées et sortie spécifiées

L'attribution des broches est fait avec un éditeur, le **Assignment Editor**. Cliquez sur **Assignments** → **Pins**. Voir la figure xxviii. Dans la colonne **Location**, entrez les numéros des broches comme indiqué dans la figure xxvi. (Raccourci : . Il suffit de taper N25 , N26 , etc. dans chaque emplacement). La table complet est affichée dans la partie inférieure de la figure xxix. La partie supérieure de la figure xxix montre que les affectations des broches sont indiqués automatiquement dans le fichier schématique.

Re-compiler le Projet :

32. Maintenant que nous avons attribué les broches nous devons re-compiler le projet pour interconnecter les entrées/sortie de notre circuit logique aux broches appropriés du FPGA. Choisissez **Processing** → **Start Compilation**.

Après une compilation réussie, nous sommes prêts à programmer le FPGA.

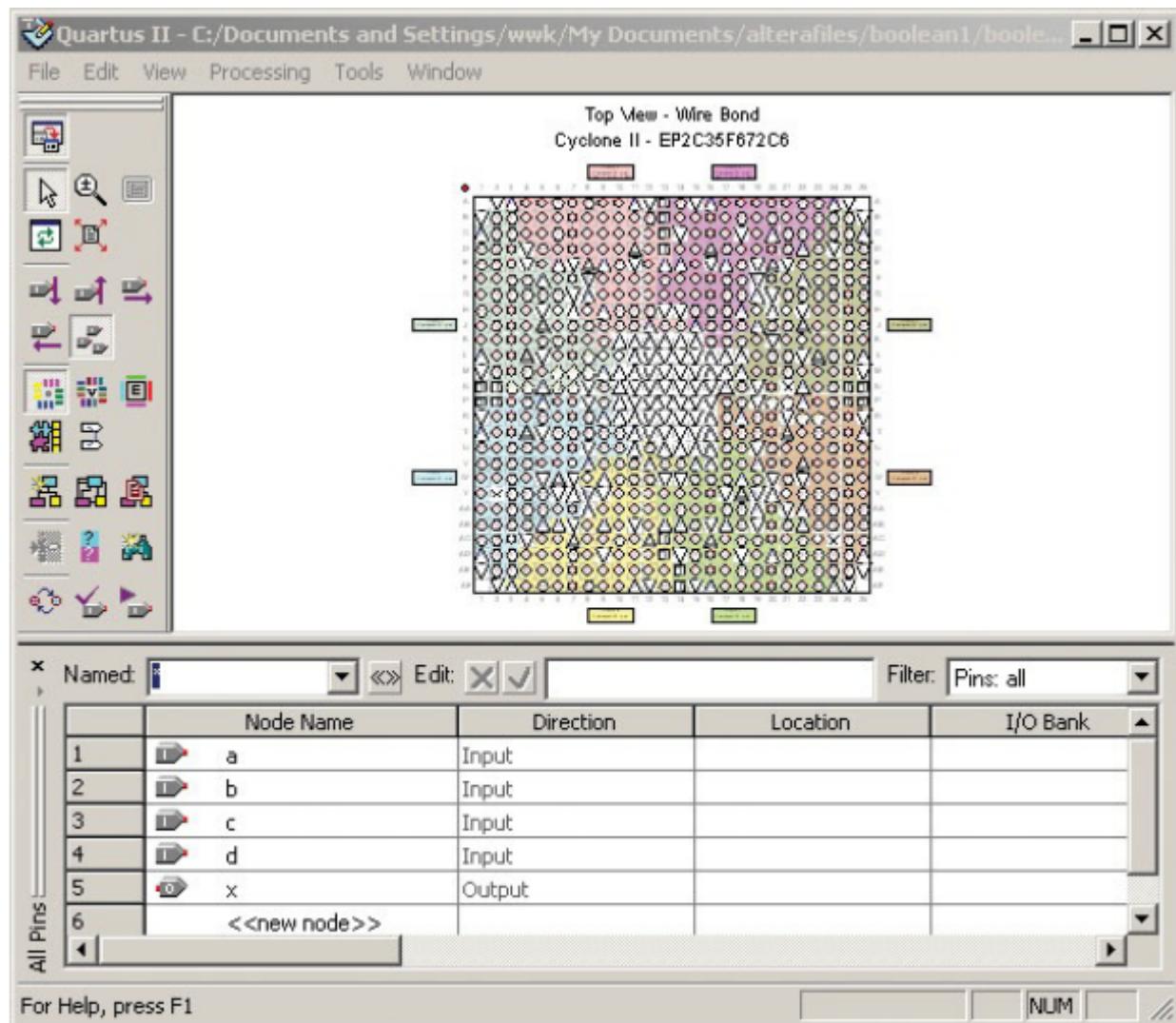


FIGURE xxviii – L'écran d'attribution des broches

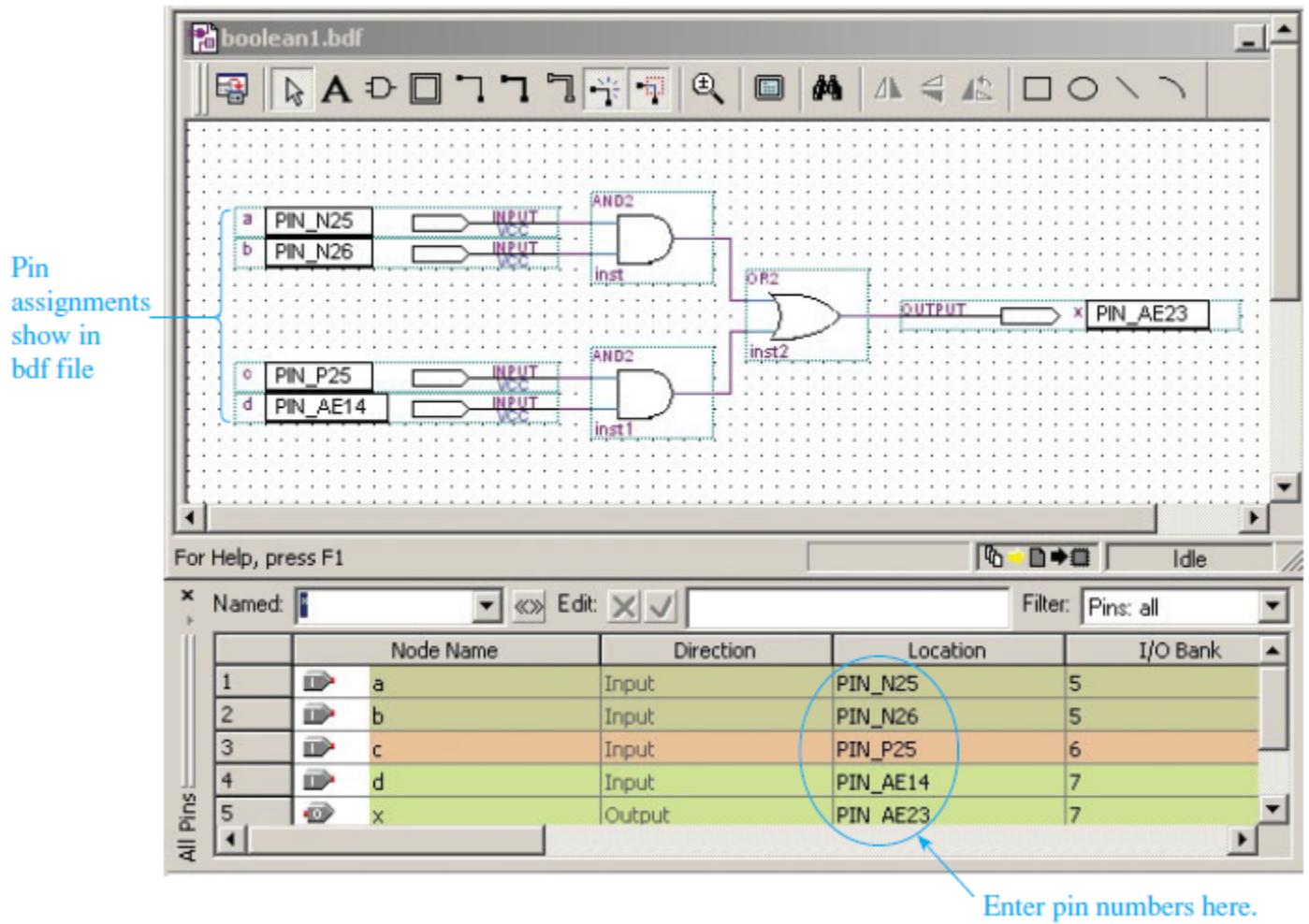


FIGURE xxix – Attribution complet

Graver le circuit sur le FPGA :

33. La dernière étape consiste à programmer le FPGA qui est sur notre carte Altera DE2. Si c'est la première fois que vous programmez une carte sur votre ordinateur, il faut installer le pilote USB pour votre carte DE2. Lisez votre manuel utilisateur pour savoir comment procéder. Le pilote facilite la communication entre votre compilateur et l'interface JTAG⁶ sur la carte Altera.

Branchez le cable USB de votre carte à votre ordinateur.

Cliquez sur **Tools → Programmer**.

La fenêtre de programmation est montré dans la figure xxx. Si c'est la première fois que vous programmez un FPGA, vous devez spécifier l'option **USB-Blaster** dans le **Hardware Setup**. Choisissez le **Mode : JTAG**.

Cliquez sur **Start** pour lancer la programmation du FPGA. Lorsque la programmation est terminée (ie 100 pourcent), vous pouvez maintenant tester votre circuit logique avec le FPGA.

6. JTAG : Joint Test Action Group, est une norme IEEE qui définit une méthode pour tester et transférer des données dans les circuits numériques.

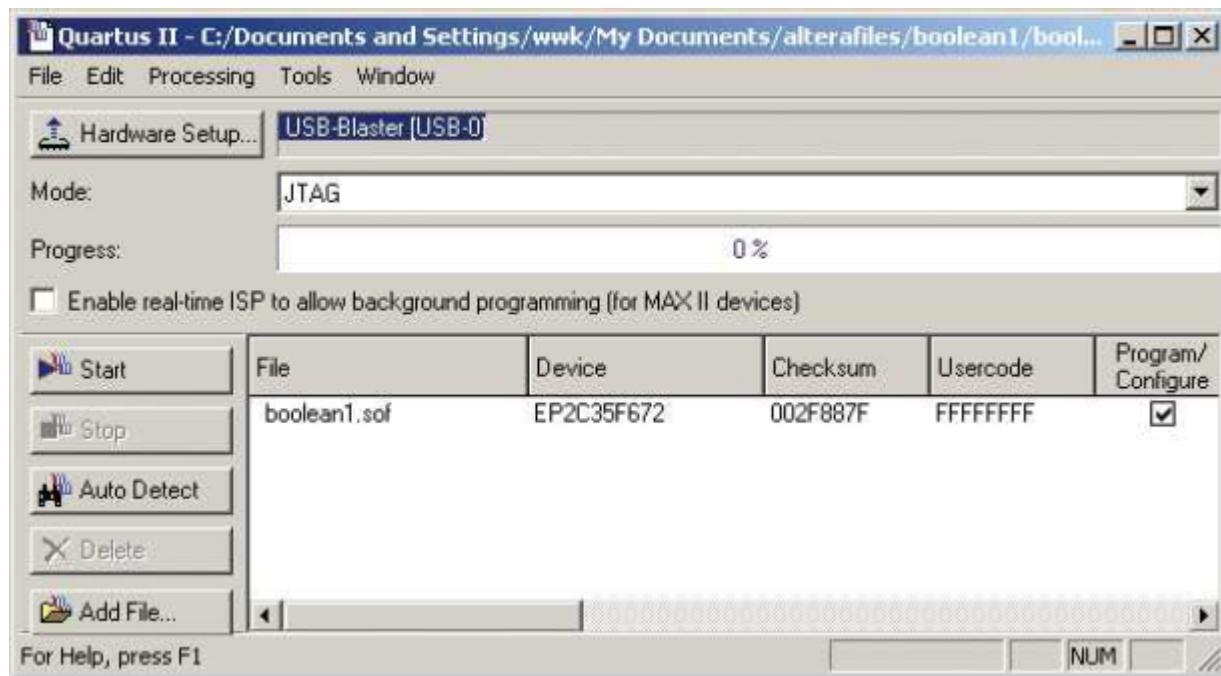


FIGURE xxx – L'écran de programmation du FPGA

Tester votre circuit logique sur le FPGA

34. Rappelons l'équation logique de notre circuit $X = AB + CD$. Tester le circuit logique en glissant les différents interrupteurs d'une façon appropriée pour tester les différents combinaisons des entrées et l'état de la sortie.

3.2 Conception des circuits logiques en VHDL

Dans cette section, nous allons implémenter le même circuit qu'avant (ie $X = AB + CD$) dans le langage VHDL. Après avoir défini les entrées et sorties, et l'équation booléen en VHDL, nous allons recompiler et simuler notre projet comme avant pour s'assurer qu'on a le même résultat.

Pour commencer, assurez vous que vous travaillez dans le projet *boolean1*. Sinon ouvrez-le, ou créez un nouveau projet (Voir la partie sur la création d'un nouveau projet).

35. Pour créer un nouveau fichier VHDL :

Cliquez sur **File → New → VHDL File → OK**. (Voir la figure xxxi)

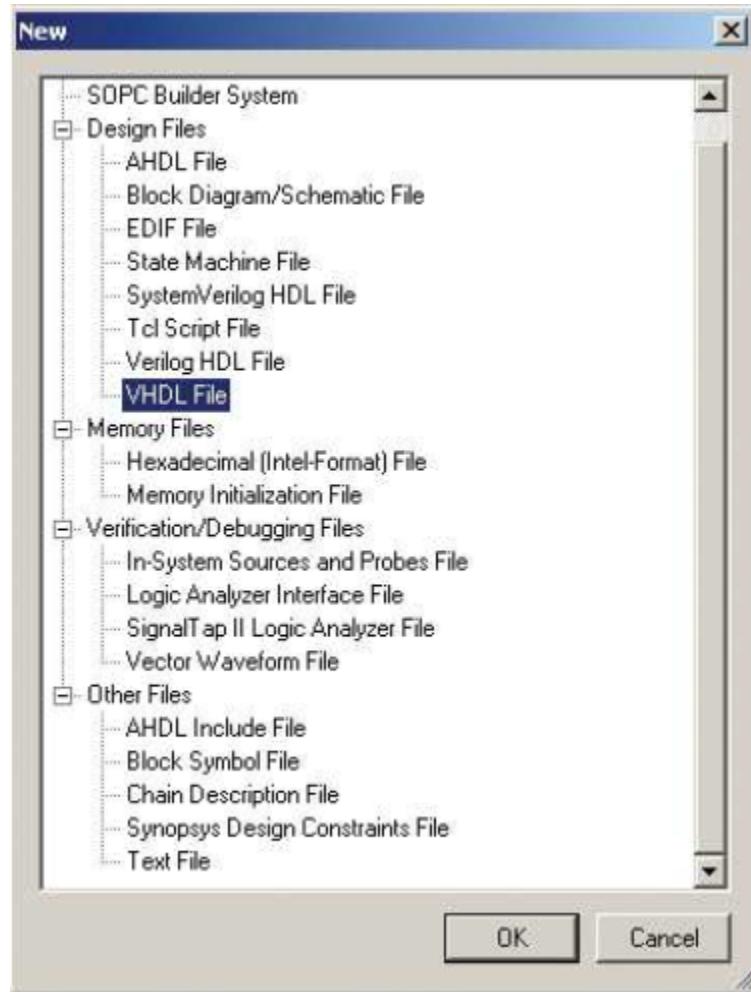
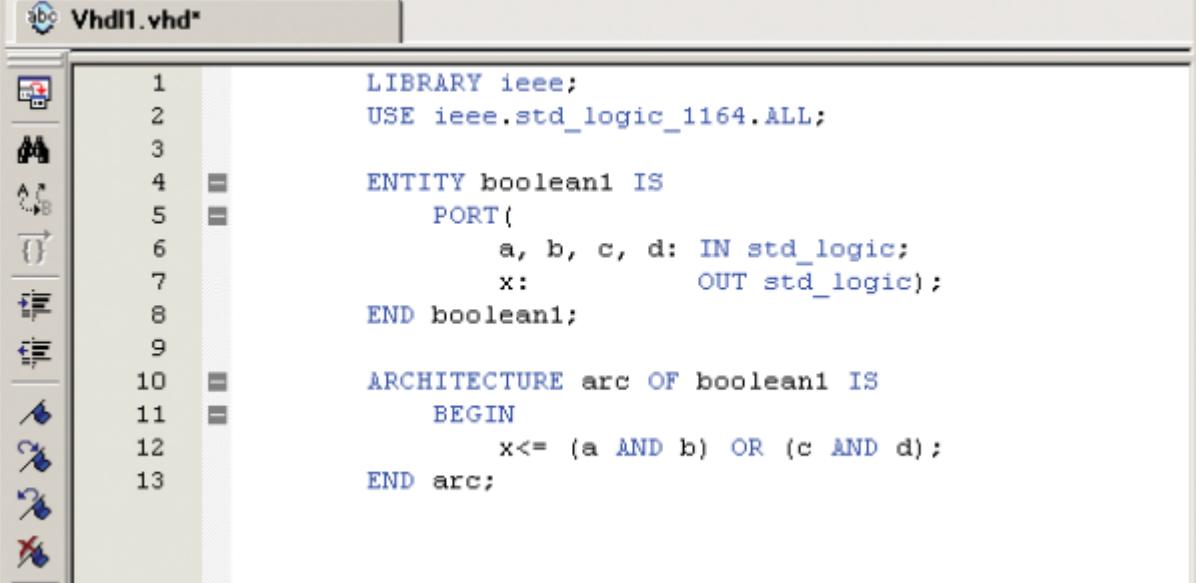


FIGURE xxxi – Crédit d'un nouveau fichier VHDL

36. Entrez le programme VHDL de l'équation logique $X = AB + CD$.(Voir la figure xxxii)



```

1      LIBRARY ieee;
2      USE ieee.std_logic_1164.ALL;
3
4      ENTITY boolean1 IS
5          PORT(
6              a, b, c, d: IN std_logic;
7              x:          OUT std_logic);
8      END boolean1;
9
10     ARCHITECTURE arc OF boolean1 IS
11         BEGIN
12             x<= (a AND b) OR (c AND d);
13         END arc;

```

FIGURE xxxii – Le code VHDL de notre équation logique

37. Enregistrez ce fichier et ajoutez-le au projet courant :

Cliquez sur **File** → **Save As** → **File name : boolean1**. Choisissez **Add File to current project** et cliquez sur **Save**. (Voir la figure xxxiii.)



FIGURE xxxiii – Enregistrez le fichier VHDL

38. Maintenant, nous allons re-compiler notre programme. Cependant, puisque nous avons déjà compilé un fichier *bdf*, nous devons le retirer du projet en cours, sinon il y aura des conflits car le projet ne saura pas qu'el fichier à utiliser comme fichier principale. Pour enlever le fichier *bdf* du projet :

Cliquer sur : **Assignments** → **Settings**.

Choisissez **Category Files**. Choisissez ensuite le fichier *boolean1.bdf* et cliquez sur **Remove** → **OK**.

(Voir la figure xxxiv) (Noter : Ceci ne supprime pas le fichier .bdf de notre projet. Nous pouvons ajouter le fichier .bdf au projet en suivant la même procédure.)

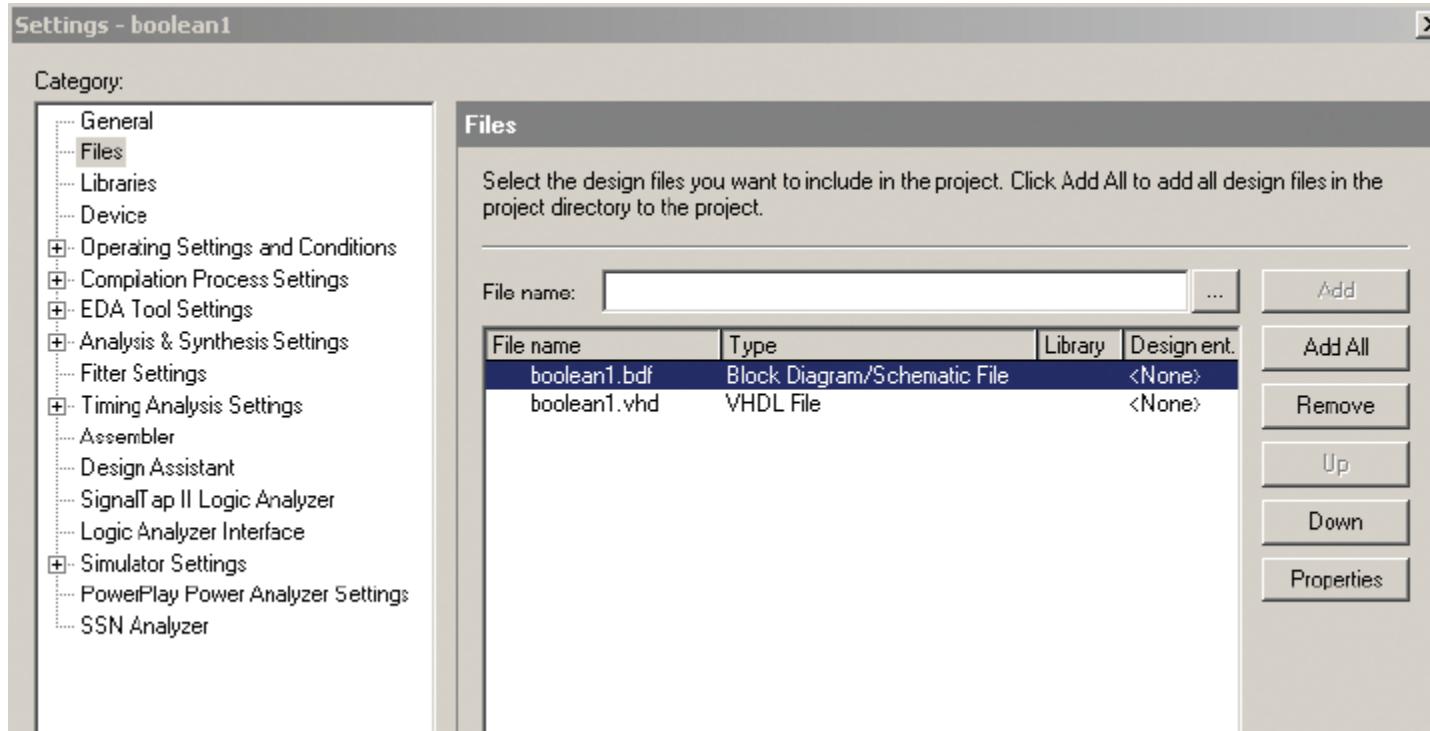


FIGURE xxxiv – Enlever le fichier bdf du projet courant

39. Recompilez le fichier :

Cliquez sur **Processing → Start Compilation**

40. Nous pouvons maintenant suivre la même procédure qu'avant pour simuler le circuit et programmer le FPGA. (Pour ouvrir le fichier de simulation créé précédemment :
Cliquez sur : **File → Open → File Name : boolean1.vwf → Open**). Suivez les mêmes étapes qu'avant pour faire votre simulation.

[2] [1]

Références

- [1] Adrien Blanchardon. *HAL, Synthèse d'architectures de circuits FPGA*. January 2016.
- [2] William Kleitz. *Digital Electronics - A Practical approach with VHDL*. PEARSON, 9 edition, 2012.