MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT AUTOMATIQUE

Mémoire

De fin d'étude en vue de l'obtention du diplôme de Master Académique
En Génie Micro-Electronique

Thème

Synthèse d'un module de cryptage RSA sur un circuit de type FPGA

Présenté par :
Mr. HOUADJ Mohamed Nazim

Proposé et dirigé par :
Mr. DAOUI Mehammed

Remerciements

Je remercie vivement mon encadreur Monsieur Mehammed DAOUI, d'avoir proposé et dirigé ce travail de ci-près, avec une qualité remarquable, j'ai grandement apprécié ses compétences, sa disponibilité et le suivi régulier a l'avancement de mon travail.

Mes sincères remerciements à tous mes enseignants qui m'ont transmit les bases de la Micro-électronique tout au long de ces deux années de master.

Un merci particulier à ma famille pour leurs encouragements et leur soutien tout au long de ce travail.

Un grand merci à mes collègues et mes amis avec qui j'ai passé de très bons moments au sein de l'université Mouloud Mammeri.

Que ceux qui se sentent oubliés trouvent ici ma profonde gratitude et mes chaleureux remerciements pour leur concours dans l'accomplissement de ce travail.

Mr HOUADJ Mohamed Nazim

A mes parents

Sommaire

Introduction générale

Le RSA est un algorithme de cryptage asymétrique, considéré actuellement comme l'un des moyens les plus sûrs pour sécuriser les transferts de données. Le RSA est un protocole garantissant une sureté proportionnelle à la taille des clés utilisées.

D'un point de vue technique ceci pose problème lorsqu'on veut implémenter ce protocole sur les systèmes embarqués actuels, dont la taille mémoire et la puissance de calcul restent insuffisantes vu les exigences sécuritaires de cet algorithme.

Pour dépasser ces limites, notre travail a consisté à trouver une solution en apportant des modifications à l'algorithme RSA en se basant sur la multiplication modulaire de Montgomery qui permet la multiplication modulaire efficace en réduisant la taille du résultat produit de deux grand nombres, par conséquent elle peut nous permettre de réduire la taille du résultat d'une puissance.

Dans la perspective d'aboutir à un protocole RSA réduit par la multiplication de Montgomery, on doit se baser sur l'algorithme Squart and Multiply. Ce dernier nous permet la réduction de la taille d'une puissance par la multiplication modulaire de Montgomery.

Le but de la réduction de la taille du RSA est d'avoir un protocole qui pourra s'implémenter sur des circuits FPGA et de les utiliser dans des systèmes embarqués afin de décharger le processeur de cette tâche qui nécessite une grande puissance de calcul.

Notre travail sera implémenté sur une carte de développement FPGA, Virtex-II (V2MB1000), de la firme Xilinx.

Organisation du mémoire :

Le travail réalisé est décomposé en 4 chapitres : le premier est dédié à la Cryptographie, dans le deuxième chapitre on s'est intéressé aux circuits logiques programmables, le troisième chapitre est consacré à la carte de développement Virtex-II et dans le dernier on a présenté notre contribution.

Chapitre 1 :

Cryptographie et étude algorithmique

Introduction

De nos jours la cryptographie est un outil très utilisé dans les domaines des télécommunications et des systèmes embarqués pour crypter des données ou des informations. Pour cela, plusieurs méthodes de cryptage existent et sont réalisées, le plus souvent, par des ordinateurs. Mais le plus grand défi de nos jours est de réduire la taille de ces méthodes en les combinant avec d'autres méthodes mathématiques afin de pouvoir les adapter aux systèmes embarqués à ressources limitées.

Dans ce chapitre, nous allons nous intéresser à la cryptographie en général, au cryptage asymétrique RSA, à l'algorithme de Montgomery pour la réduction modulaire et à l'algorithme Squart and Multiply.

II. La cryptographie

La cryptographie est l'ensemble des techniques (algorithmes, matériels, logiciels) permettant de protéger une communication au moyen d'un code secret. L'information à communiquer est modifiée par l'émetteur de façon à la rendre compréhensible uniquement par le destinataire.

De nos jours, on distingue deux types de cryptographie : la cryptographie symétrique et la cryptographie asymétrique

II.1 Cryptographie symétrique

C'est la cryptographie la plus ancienne également dite à clé secrète. Elle est basée sur l'échange préalable d'une clé secrète commune. Cette clé a une taille variable, elle peut être sous la forme d'un chiffre, d'une lettre, d'un livre de code . . . Avec cette clé commune, les communicants peuvent crypter ou décrypter un message. Aujourd'hui, de nombreux algorithmes de cryptage à clé secrète sont utilisés tels que : le DES, le triple DES, le AES. . . . Ces algorithmes sont basés sur des permutations et des transformations de blocs et n'utilisent pas d'arithmétique modulaire. La figure (II.1) illustre le principe de la cryptographie symétrique.

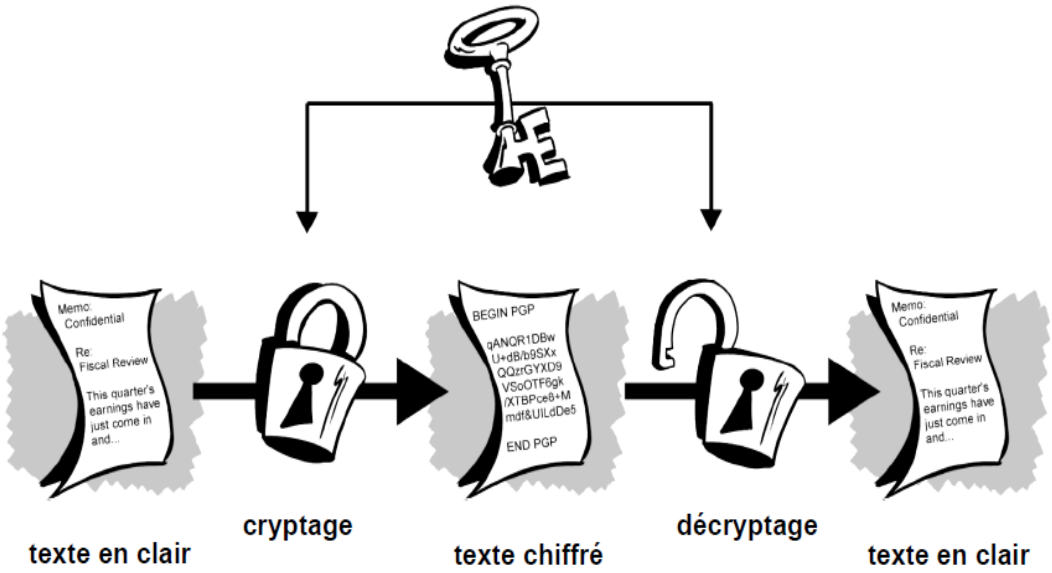


Figure (II.1) principe de la cryptographie symétrique

II.2 Cryptographie asymétrique

C'est une cryptographie moderne qui a vu le jour dans les années soixante-dix. Au lieu d'une seule clé, il y a deux clés différentes. Une clé publique (qui sera diffusée) pour crypter et une clé privée (gardée secrète) pour décrypter. Chaque récepteur peut à l'aide de la clé publique du destinataire crypter un message et seule le destinataire en possession de la clé privée peut le décrypter. Le principe de la cryptographie à clé publique, est illustré dans la figure (II.2).

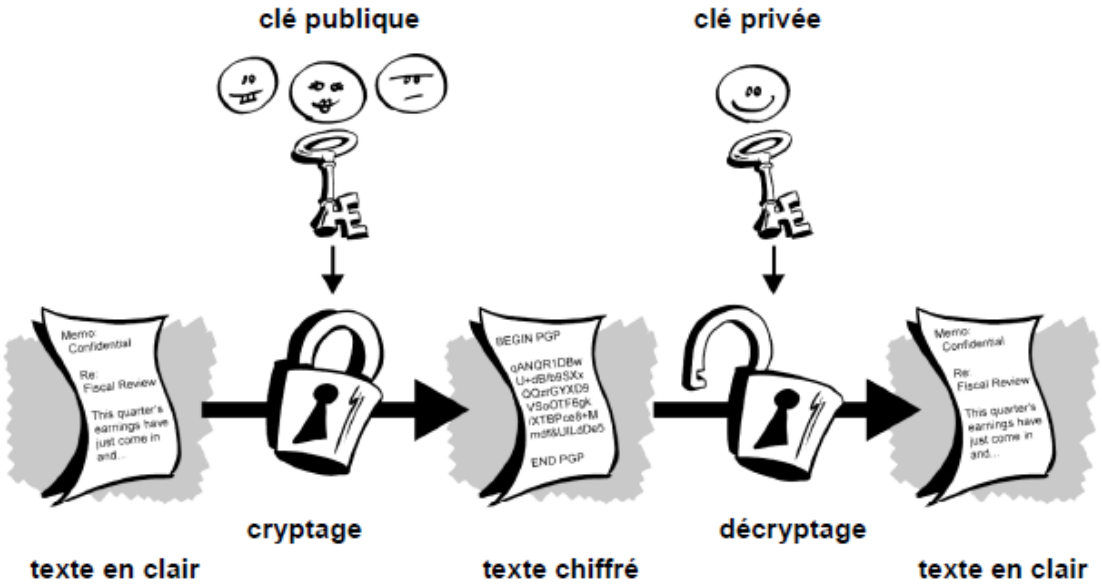


Figure II.2 principe de la cryptographie asymétrique

III. Le cryptage asymétrique 'RSA'

RSA est sûrement le plus connu des algorithmes cryptographiques à clé publique. En proposant ce protocole en 1978, Rivest Shamir et Adelman ouvrent un nouveau pan de la cryptographie (la cryptographie asymétrique). La cryptographie asymétrique se nomme ainsi car elle est à sens unique. La cryptographie asymétrique utilise une clé (publique) pour crypter et une autre clé (secrète) pour décrypter.

L'idée est qu'une personne ''A'' crée deux clés qui lui sont associées. Une clé publique est utilisée par toute personne qui désire lui envoyer un message. Cette clé est associée à un protocole et, puisqu'elle est publique, toute personne peut crypter un message pour l'envoyer à ''A''. Une seconde clé, secrète et connue uniquement par ''A'', permet à ''A'' d'utiliser une fonction trappe pour décrypter les messages qu'il a reçus. En fait, après avoir crypté un message avec la clé publique, l'émetteur n'est pas capable de décrypter le message qu'il a crypté. C'est ce qui rend sûr la cryptographie asymétrique et c'est ce qui permet à ''A'' de publier sa clé publique, elle ne permet pas de décrypter le message.

Rivest, Shamir et Adelman ont proposé un protocole basé sur ce principe de cryptographie à clé publique. Ils utilisent comme fonction trappe la factorisation, il est

“facile” de faire la multiplication de deux nombres p et q mais “difficile” de factoriser un produit p*q pour récupérer les entiers p et q.

III.1 Création des clés RSA

Algorithme :

- La personne ‘’A’’ choisit deux grands nombres premiers p et q.
- Calculer n = p*q et Ø(n) = (p-1)*(q-1).
- Choisir un entier e premier avec Ø(n).
- Calculer b = (inverse de e) mod Ø(n).

La paire (n, e) est la clé publique du RSA.
La paire (n, b) est la clé privée du RSA.

Maintenant la personne ‘’A’’ peut proposer sa clé publique afin de recevoir des messages sécurisés.

III.2 Protocole RSA

Le protocole RSA va permettre à n’importe quelle personne qui connait la clé publique, de crypter un message afin de l’envoyer à la personne ‘’A’’, et à son tour la personne ‘’A’’ peut décrypter le message.

III.2.1 Le chiffrement avec le RSA

La fonction de chiffrement RSA a pour entrées la clé publique du RSA (n, e) et un texte en clair M. Le texte va être découpé en blocs de K bits, tel que $2^K \leq n$. ($M = m_1, m_2, \ldots m_l$) et chaque bloc est traité comme un entier m.

La sortie de la fonction de chiffrement est un message crypté C ($C = c_1, c_{2, \ldots\ldots} c_l$), avec $c_i = m_i^e \bmod n$.

III.2.2 Le déchiffrement avec le RSA

La fonction de déchiffrement a pour entrées la clé privée (n, b) et le texte chiffré C ($C = c_1, c_{2, \ldots\ldots,} c_l$). Donc pour retrouver le message M. ($M = m_1, m_2, \ldots. m_{l)}$ en clair on calcule $m_i = c_i^b \bmod n$.

IV Algorithme de Montgomery pour la multiplication modulaire

En 1985, Peter Montgomery franchit un cap dans la réduction modulaire. Il transforme la réduction modulo n en une division par une puissance de la base. En pratique, une division par une puissance de la base revient à un décalage, qui ne coûte rien.

IV.2 Présentation du problème

Etant donnés trois nombres entiers a, b, n, le problème est de calculer le plus rapidement possible a*b mod n en un minimum d'espace.

IV.3 Transformation de Montgomery

- **Définition de la transformation**

Soit n le modulo intervenant dans l'opération. On supposera désormais que n est un nombre impair. Le nombre de bits de n est l'entier k tel que : $2^{k-1} \leq n < 2^k$. On appellera r le nombre 2^k. Comme n est impair, r est premier avec n et donc inversible modulo n. On notera r^{-1} l'inverse de r modulo n.

Soit mont (transformation de Montgomery) donc :

$$mont(a) = a*r \bmod n.$$

Cette application mont (multiplication par r modulo n) est une bijection puisque r est inversible modulo n et on peut écrire :

$$a = mont(a)*r^{-1} \bmod n.$$

- **Comment calculer la transformation d'un produit ?**

On calcule une fois pour toute r^{-1}, tel que :

$$r\ r^{-1} = 1 \bmod n.$$

Et on calculera aussi v tel que :

$$r\ r^{-1} - v\ n = 1.$$

Une fois ces coefficients calculés on suit les étapes suivantes :

- s = mont(a) * mont(b).
- t = (s*v) mod r.
- res_mont = (s + t*n) / r.

Les résultats de la multiplication obtenus sont dans le domaine de Montgomery. Pour avoir le résultat dans le domaine naturel on fait la transformation suivante :

$$res = res\_mont * r^{-1} \bmod n.$$

V. Algorithme Square and Multiply

La première façon de calculer une puissance n^e, est de multiplier n par lui-même e fois. Cependant, il existe des méthodes bien plus efficaces, où le nombre d'opérations nécessaires n'est plus de l'ordre de e.

L'algorithme square and multiply est l'une des méthodes qui calculent une puissance $S = X^E \bmod n$, en un minimum d'opérations.

V.1 Description de l'algorithme

- On initialise les paramètres d'entrée et de sortie :

$$S(0) = 1 ;\ (\text{élément neutre}).$$

$$Z(0) = X.$$

$$E = \sum_{i=0}^{n-1} e_i 2^i$$

- On fait une boucle allant de 0 jusqu'à (n-1) et on calcule :

$$Z\,(i+1) = Z_i^2 \bmod n.$$

- On fait un test sur chaque bit de e et si e (i) = 1 on calcule :

$$S\,(i+1) = S\,(i) * Z\,(i) \bmod n.$$

- Si non on calcule :

$$S\,(i+1) = S\,(i).$$

VI. Conclusion

En cryptographie asymétrique, de grands nombres sont utilisés (plusieurs centaines de bits pour le RSA). Dans certains systèmes embarqués ou l'espace est limité, le calcul du produit de deux grand nombres est problématique.

Dans ce contexte nous allons proposer un nouveau protocole RSA qui permet de crypter un message tout en réduisant la taille de ce dernier afin de pouvoir l'implémenter sur un circuit FPGA.

Le chapitre suivant sera consacré aux circuits logiques programmables.

Chapitre 2 :

Circuits logiques programmables

Introduction

Durant les dernières années, le domaine de l'intégration des circuits électroniques a connu des progrès considérables dans la recherche des circuits capables de réaliser des fonctions de plus en plus complexes. Ces progrès ont permis la miniaturisation des composants, la réduction de leur coût et de leur consommation ainsi que l'augmentation de leurs performances. Pour cela, le rôle des fabricants des semi-conducteurs, pendant ces trois dernières décennies, ne s'est pas limité à faire progresser la technologie et par conséquent à augmenter le nombre de transistors par unité de surface, mais aussi à définir les types de circuits à créer afin de satisfaire leurs clients.

Dans les années soixante on distinguait :

- Les portes logiques.
- Les circuits intégrés SSI (Small Scale Integration).
- Les registres à décalages, les compteurs …etc.

Dans les années soixante-dix :

- Les circuits intégrés LSI (Large Scale Intégration).
- Les PIA (Adaptateur Interface Périphérique).
- Les ACIA (Adaptateur Interface Asynchrone de Communication).
- Les microprocesseurs.

Les composants programmables, apparus au début des années 70, ont dû attendre les années 80 pour que l'évolution de la technologie permette l'intégration d'architectures complexes et performantes.

Dans les années quatre-vingts, un virage a été marqué dans la conception des circuits intégrés avec le développement des microprocesseurs et l'arrivée de la VLSI (Very Large Scale Integration).

Notons bien que cette évolution a apporté beaucoup de solutions à la majeure partie des problèmes logiques, par une solution programmée en utilisant des microcontrôleurs, mais elle n'était pas la solution idéale car elle s'est heurtée à plusieurs butoirs :

- D'une part, la gamme des produits standards offerts aux utilisateurs, ne répond pas tout le temps à leurs exigences.
- D'autre part, la difficulté des fabricants de poursuivre l'innovation.
- Et enfin, les outils de conception tardent à venir.

Une autre solution est proposée aux utilisateurs, c'est la conception de circuits intégrés sur mesure. Elle permet :

- D'optimiser les dimensions, autrement dit, réduire la surface du silicium.
- D'optimiser les performances fonctionnelles et électriques (rapidité et consommation d'énergie).

Cependant, cette solution est très coûteuse en temps de conception et en investissement. Elle est envisageable pour un volume de production très élevé. A partir des années 80 une autre voie a vu le jour, l'utilisation des ASIC (Application Specific Integrated Circuit), qui a été réalisable grâce aux progrès des outils C.A.O (conception assistée par ordinateur) qui permettent aux ingénieurs de concevoir des circuits intégrés de manière rapide et sûre et dans divers domaines de l'électronique.

II. Les ASIC (Application Specific Integrated Circuit)

Dans un premier temps, nous allons rappeler quelques concepts autour des circuits intégrés pour applications spécifiques ASIC. Les circuits ASIC constituent la troisième génération de circuits intégrés qui a vu le jour au début des années 80. En comparaison avec les circuits intégrés standards et figés proposés par les fabricants, l'ASIC présente une personnalisation de son fonctionnement, selon l'utilisation, accompagnée d'une réduction de temps de développement, d'une augmentation de la densité d'intégration et de la vitesse de fonctionnement. En outre sa personnalisation lui confère un autre avantage industriel, c'est évidemment la confidentialité. Ce concept d'abord développé autour du silicium s'est ensuite étendu aux autres matériaux pour les applications micro-ondes ou très rapides.

Par définition, les circuits ASIC regroupent tous les circuits dont la fonction peut être personnalisée d'une manière ou d'une autre en vue d'une application spécifique, par opposition aux circuits standards dont la fonction est définie et parfaitement décrite dans le catalogue de composants. Les ASIC peuvent être classés en plusieurs structures de base

(réseau programmable, cellule de base, matrice,… etc.). Sous le terme ASIC deux familles sont regroupées, les semi-personnalisées et les personnalisées comme le représente la figure (1).

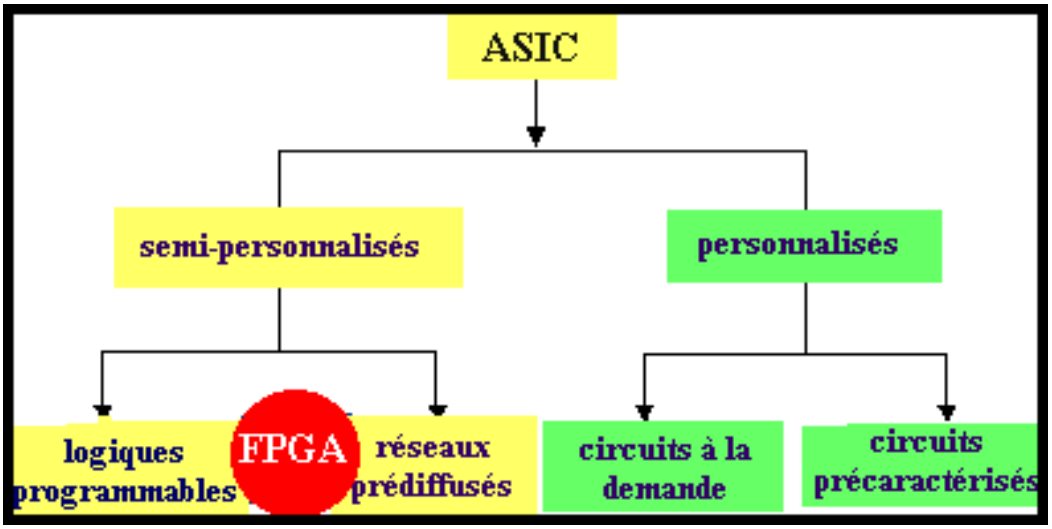


Figure (1) les différents circuits ASIC

II.1 Les circuits semi-personnalisés

Les semi-personnalisables sont des réseaux de transistors et/ou de portes logiques. L'utilisateur doit personnaliser les interconnexions pour réaliser des fonctions spécifiques bien définies. Cette famille comprend :

- Les réseaux logiques programmables
- Les réseaux pré-diffusés

II.1.1 Les réseaux logiques programmables

Ce sont des circuits standards programmables par l'utilisateur (programmation des interconnexions). Ces composants ont une architecture basée sur des portes logiques 'OR' et 'AND'. On trouve dans la même catégorie :

- PAL (Programmable Aray Logic).
- EPLD (Erasable PLD) effaçable par rayons UV.

- EEPLD (Electrically Erasable PLD) effaçable électriquement.

II.1.2 Circuits pré-diffusés

Ces composants se présentent sous forme de tableau de portes logiques bien distribuées et pré-caractérisées, fourni avec une carte détaillée de l'architecture interne et l'emplacement exact de chaque porte, pour que l'utilisateur arrive à définir une carte des interconnexions de ces portes pour avoir la fonction souhaitée tout en essayant d'optimiser les interconnexions et l'emplacement des portes utilisées.

Cette technologie a un avantage de rapidité de conception et elle est à faible coût, mais elle présente l'inconvénient d'avoir une faible densité d'intégration.

II.2 Circuits personnalisés

Ce sont des circuits non préfabriqués. Pour chaque application on conçoit un circuit intégré le plus optimisé possible, ce qui conduit à la création de son propre composant. Cette famille comprend :

- Les circuits à la demande.
- Les circuits pré-caractérisés.

II.2.1 Circuits à la demande

Ce sont des circuits qui sont conçus pour répondre aux exigences d'un utilisateur (une fonction bien définie). La conception de ce type de circuits démarre des spécifications, de l'architecture voulue et à l'aide des outils C.A.O faire les dessins de mask et passer à la fonderie pour obtenir le produit final.

Toutes les opérations, de la conception à la fabrication sont effectuées de façon spécifique, adaptée aux exigences de l'utilisateur. C'est l'ASIC le plus optimisé car aucune contrainte ne lui est imposée, cependant son temps de fabrication est élevé ainsi que son coût de revient.

II.2.2 Les circuits pré-caractérisés

Ce sont des circuits basés sur des cellules standard pré-caractérisées et d'autres cellules qui réalisent des fonctions spécifiques. La conception de ces circuits consiste à trouver le meilleur moyen de les disposer et de les interconnecter afin d'optimiser le circuit. Il existe trois types de cellules :

- Cellules standards (logique classique).
- Méga cellules (microprocesseur, périphériques).
- Cellules compilées (RAM, ROM, …etc.).

Ces circuits présentent des avantages tels que la rapidité de conception et de fabrication (environ 8 semaines), pas de perte de silicium et un grand investissement des outils C.A.O. Parmi ces inconvénients, le choix limité des cellules et un début de concurrence avec les FPGA.

II.3 Les avantages de l'utilisation des ASIC

D'une manière générale l'utilisation d'un ASIC conduit à de nombreux avantages provenant essentiellement de la réduction de la taille des systèmes. Il en ressort :

- Réduction du nombre de composants sur le circuit imprimé. La consommation et l'encombrement s'en trouvent considérablement réduits.
- Le concept ASIC par définition assure une optimisation maximale du circuit à réaliser. Nous disposons alors d'un circuit intégré correspondant réellement à nos propres besoins.
- La personnalisation du circuit procure une confidentialité au concepteur et une protection industrielle.
- Enfin, ce type de composant augmente la complexité du circuit, sa vitesse de fonctionnement et sa fiabilité.

Dans notre travail nous nous sommes intéressés à une classe d'ASIC que sont les FPGA.

III. FPGA (Fiel Programmable Gate Arrays)

Les FPGA sont des circuits pré-diffusés programmables. Contrairement aux circuits pré-diffusés conventionnels, on n'a pas à faire de fabrication spéciale en usine, ni de système de développement coûteux.

Inventé par XILINX, dans la famille des ASIC, se situe entre les réseaux logiques programmables et les pré-diffusés, c'est donc un composant standard qui combine densité et performances d'un pré-diffusé avec la souplesse due à la reprogrammation d'un PLD.

Les FPGA sont des circuits VLSI à haute intégration (3 000 000 de ports en 2010), qui emploient une technologie CMOS performante, qui est passée de 2µm avec 2 niveaux de métallisation en 1986 à 0.13µm avec 9 niveaux de métallisation en 2002, et à 13nm en 2010.

III.1 Principaux fondeurs des FPGA

Les fabricants des circuits FPGA ne cessent d'améliorer leur produit en efficacité et en puissance. L'ensemble des principaux fondeurs qui conçoivent ce type de circuits sont : Actel, Altera, Atmel, Xilinx, Lattice et d'autres.

III.2 Architecture des FPGA

Les FPGA sont des circuits intégrés composés d'un réseau de cellules élémentaires ou d'éléments logiques programmables, répartis régulièrement et reliés entre eux grâce à des interconnexions qui forment une matrice de routage programmable pour obtenir un comportement du circuit dans sa globalité. Les différents éléments d'un FPGA (figure (2)) sont : cellule de base, slice, CLB, IOB et matrice d'interconnexion.

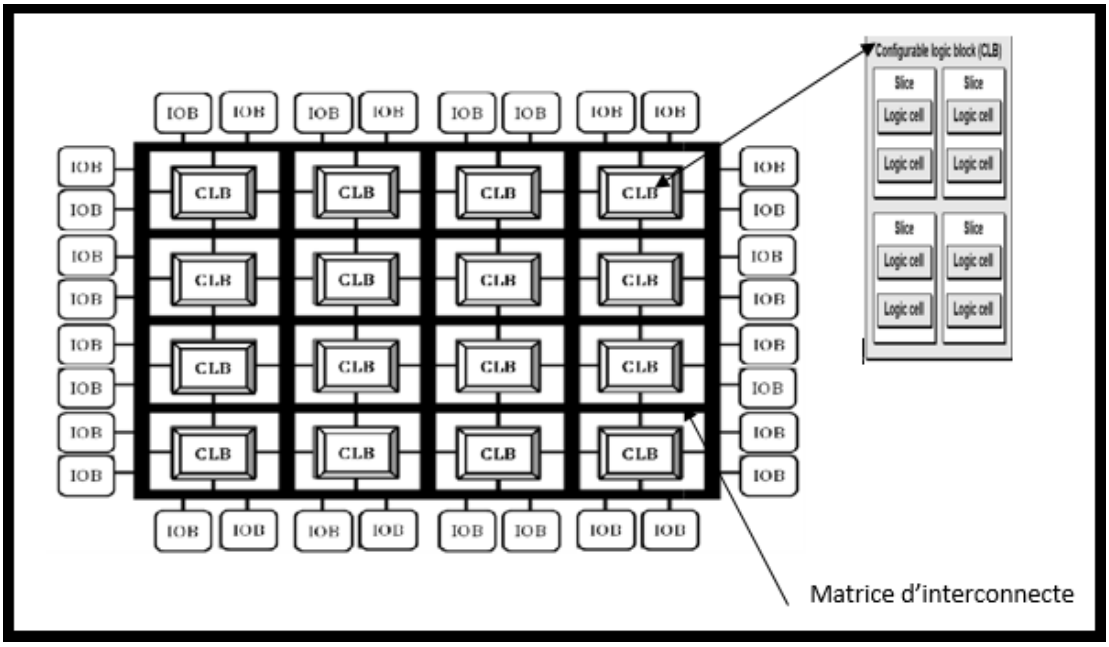


Figure (2) : Architecture interne d'un FPGA

- **Cellule de base :** ce sont des éléments de base d'un FPGA, avec lesquels on peut réaliser toute opération de logique. Ils contiennent des LUT (look up table) qui sauvegardent la table de vérité de la fonction programmée, des générateurs de fonctions ou aussi des blocs de mémorisation.
- **Slice :** un slice est constitué d'un ensemble de cellules de base, qu'on va interconnecter pour avoir la fonction voulue.
- **CLB (Configurable Logic Bloc)** : c'est un ensemble de slice interconnecté pour obtenir une fonction déterminé.
- **IOB (input output bloc) :** les blocs IOB servent d'interface entre les broches du composant FPGA et la logique interne développée à l'intérieur du composant. Ils sont sur toute la périphérie du circuit intégré.
- **Matrice d'interconnexions :** elle est sous forme de grille de segments métalliques verticaux et horizontaux positionnée entre les lignes et les colonnes des CLB et IOB.

III.3 Les différents types d'FPGA

Les FPGA ont différentes architectures et on distingue deux types principaux qui sont :

- Les FPGA à anti fusibles.
- Les FPGA de types SRAM.

III.3.1 Les FPGA de type anti fusible

Les anti-fusibles sont des composants généralement constitués de deux couches conductrices entourant un diélectrique. L'application d'une tension élevée fait claquer le diélectrique créant une liaison entre les deux couches conductrices du composant, donc elles peuvent être considérées comme des circuits ouverts dans leur état initial, et des circuits fermés une fois programmées.

L'avantage de cette technologie est sa faible taille, son haut niveau de confidentialité due aux difficultés de relecture de l'état du fusible, et sa faible sensibilité aux radiations et autres perturbations environnementales, induisant une bonne fiabilité. Cependant elle présente l'inconvénient d'être configurable une seule fois.

III.3.2 Les FPGAS de type SRAM

La structure de base d'un FPGA de type SRAM est complexe, le point de connexion entre les différentes cellules est un ensemble de transistors MOS de commutation commandés par des cellules de mémoire vive (RAM).

Les mémoires SRAM sont également utilisées pour la conception de réseaux programmables du fait de leurs performances (rapidité d'accès notamment) et leur faible coût. On trouve deux grandes architectures pour ces points mémoires, les 6T-SRAM et 5T-SRAM, respectivement à six et cinq transistors. La cellule à six transistors est constituée de deux inverseurs montés tête-bêche et deux transistors d'accès, voir la Figure (3). La figure (4) représente une cellule à cinq transistors.

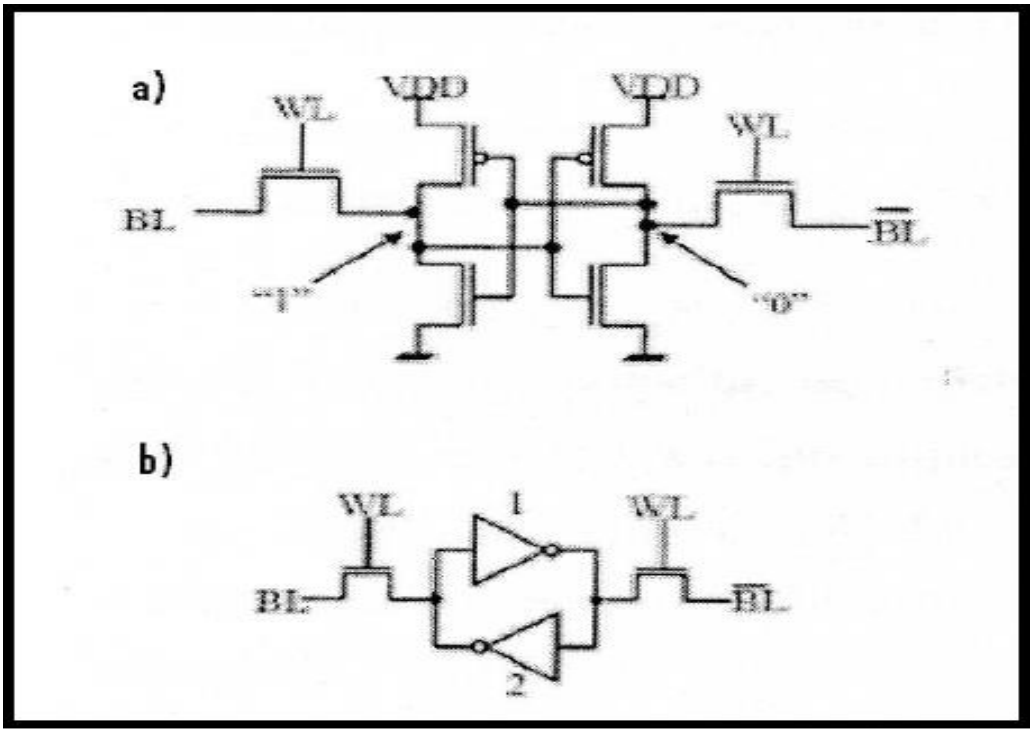


Figure (3) : architecture 6T-SRAM

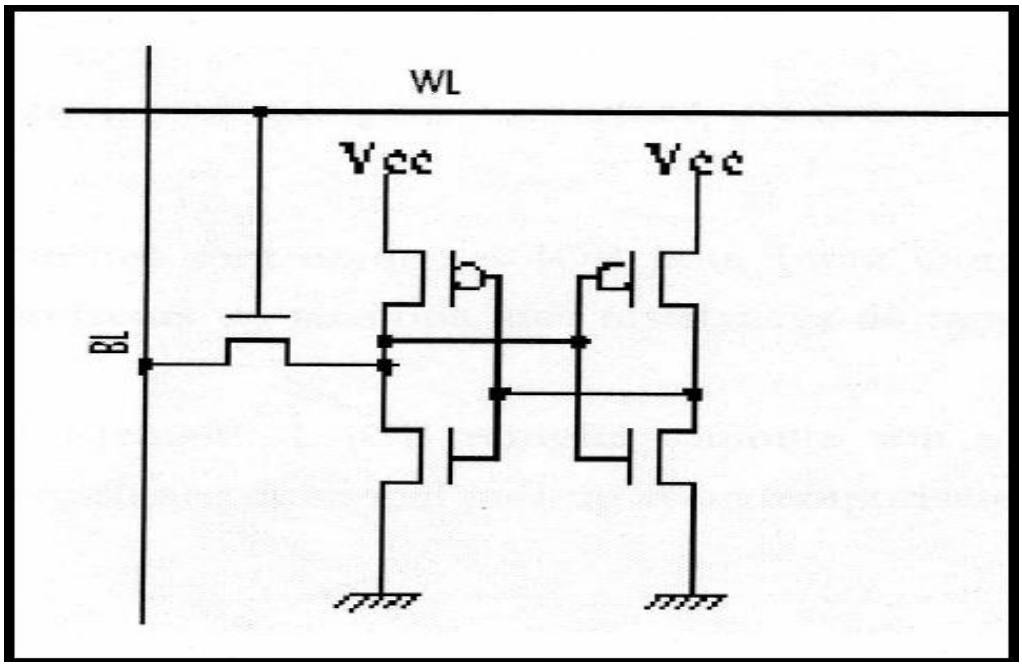


Figure (4) : architecture 5T-SRAM

III.4 Domaine d'application des FPGA

Les FPGA dépassent désormais leur rôle d'interfaçage d'appoint, qui leur permettait d'apporter une solution à ce que la logique des produits ne parvenait pas à résoudre. Ils occupent en effet aujourd'hui le devant de la scène grâce à leur capacité à s'intégrer dans des systèmes sur puce entièrement personnalisés, conçus spécifiquement pour une application particulière. Parmi ces applications : la télécommunication, la cryptographie, les systèmes embarqués, les applications de teste et de contrôle …etc.

III.5 Critères de choix des FPGA

Les FPGA étant adaptables à différentes tâches, ils peuvent facilement répondre à des exigences poussées, comme par exemple le conditionnement de signaux, l'utilisation de nouvelles interfaces ou l'accélération de parties de modèles. Les FPGA sont particulièrement utiles, vu qu'on peut les configurer en un temps relativement court et à bas coût, et le fait de connaitre ces caractéristiques on pourra faire des applications sur ces circuits qui seront plus performants et qui consommeront un minimum de puissance.

III.6 Avantage et inconvénients des FPGA

En nous intéressant aux FPGA on a constaté qu'ils ont beaucoup d'avantages comme ils ont aussi leurs inconvénients. On va citer quelques-uns dans le tableau suivant :

Avantages	Inconvénients
• Technologie facile à maitriser. • Haute performance de calcul. • Temps de mise sur le marché réduit. • Faible coût de revient. • Très fiables. • Reconfiguration parfois en temps réel.	• Performances non optimisées. • Temps de réponse long par rapport aux ASIC.

IV. Configuration des FPGA par outils CAO

Les techniques de conception CAO (Conception Assistée par Ordinateur) sont aujourd'hui largement employées afin de concevoir des circuits électroniques nécessaires à mettre en pratique les connaissances algorithmiques. L'approche moderne pour la conception des circuits et la manière d'introduire une fonctionnalité sur un support physique sont confiées aux outils CAO. Les outils CAO sont utilisés pour générer le fichier de configuration des FPGA qui s'appelle bit-Stream, à partir d'une description de haut niveau.

IV.1 Méthode de conception par CAO

Le flot de conception d'un système sur puce regroupe plusieurs niveaux d'abstraction. Dans chaque niveau le concepteur s'intéresse à la résolution d'un problème. Les outils CAO sont utilisés intensivement afin d'assurer la transition entre les différents niveaux d'abstraction.

Le développement d'une application sur FPGA par des outils CAO, suit l'enchaînement des étapes suivantes (figure (5)) : spécification du design, développement du design, la synthèse, le placement et le routage, intégration et implémentation.

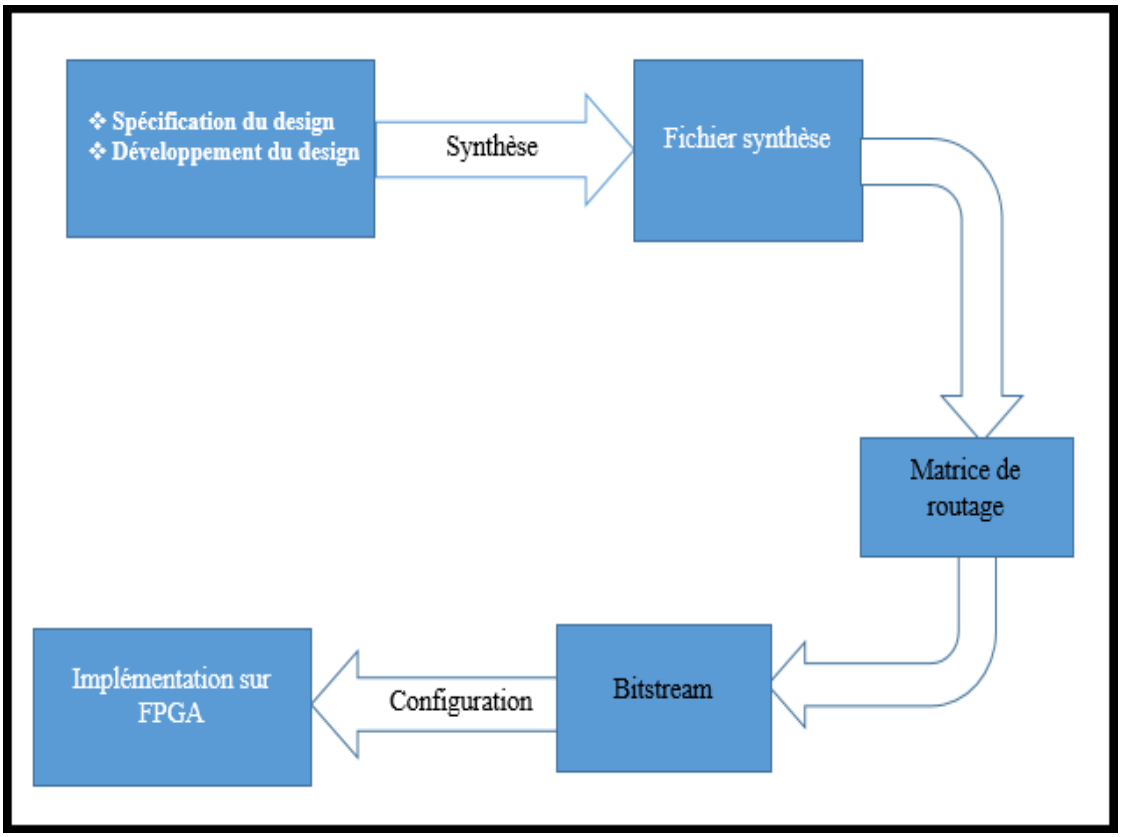


Figure (5) : étapes de développement par outils CAO

IV.1.1 Spécification du design

Cette étape se fait sur un outil de développement tel que le VHDL, VERILOG,… Elle consiste en la spécification de nombres de broches et leur localisation sur le circuit

FPGA, ainsi que la spécification de la fréquence d'horloge du système et la mémoire requise pour l'application.

IV.1.2 Développement du design

Une fois l'outil de développement choisi, il faut respecter certaines règles propres à ce dernier, pour que le code soit synthétisable. Vient alors la saisie du code et là on a deux choix, soit le graphique (Machine à états) ou la saisie HDL (Hardware Description Language). Après ça, on passe à la simulation ensuite à la synthèse.

IV.1.3 La synthèse

La synthèse est le processus qui convertit le code HDL ou le graphique pour avoir une représentation au niveau portes logiques la mieux adaptée. Le résultat de cette étape est un fichier représentant les différentes portes à utiliser et leurs interconnexions, représentant le circuit électronique, appelé fichier synthèse.

IV.1.4 Placement et routage

A partir du fichier synthèse, l'outil de conception procède au placement et routage. Un algorithme de routage est sensé faire l'aiguillage des données qu'il reçoit vers leur destination par action sur les nœuds de routage, ce qui est équivalent à définir les chemins qui relient l'ensemble des CLB contenus dans la fonction désirée, toute en essayant d'optimiser ces chemins au maximum.

Cette étape va générer un ensemble de valeurs numériques appelées bit-Stream, qui seront chargées sur le composant FPGA.

IV.1.5 Intégration et implémentation

L'implémentation est la réalisation proprement dite, qui consiste à mettre en œuvre l'algorithme cible, c'est à dire à compiler, charger, puis lancer l'exécution sur un

ordinateur ou calculateur. C'est une étape de programmation physique et de tests électriques qui clôturent la réalisation du circuit.

IV.2 Technique de programmation des FPGA

Les circuits FPGA ne possèdent pas de programme résident. A chaque mise sous tension, il est nécessaire de les configurer. Leur configuration permet d'établir des interconnexions entre les CLB et les IOB. Pour cela ils disposent d'une RAM interne dans laquelle sera écrit le fichier de configuration. Le format des données du fichier de configuration est produit automatiquement par le logiciel de développement sous format d'un ensemble de bits organisés en champs de données. Les FPGA disposent de quatre modes de chargement et de trois broches M0, M1, M2, lesquelles définissent les différents modes comme l'indique le tableau ci-dessous. Ces modes définissent les différentes méthodes pour envoyer le fichier de configuration vers le circuit FPGA.

M0 M1 M2	Mode sélectionné
0 0 0	Maître série
0 0 1	Maître parallèle bas
0 1 1	Maître parallèle haut
1 0 1	Périphérique
1 1 1	Esclave série

- Mode maître série :

  Le mode maître série utilise une mémoire à accès série de type registre à décalage commercialisée par le fabricant Xilinx. Le programme est préalablement chargé par le système de développement utilisé pour le circuit FPGA. Le FPGA génère tous les signaux de dialogue nécessaires pour la copie du contenu de la PROM dans la RAM interne, lorsque la copie est terminée, il bascule le signal DONE pour le signaler au circuit. Pour ce mode on peut avoir une seule PROM pour configurer plusieurs circuits FPGA avec la même configuration ou plusieurs PROM pour configurer plusieurs FPGA en chaîne ou le premier des circuits FPGA est le maître et génère l'horloge.

- On dispose aussi d'un mode maître-parallèle ou le FPGA est relié à une EPROM classique, de même qu'un mode passif type périphérique dans lequel le FPGA est considéré comme un périphérique de microprocesseur et peut-être configuré à partir de celui-ci.
- Mode esclave :

  Dans ce mode, le programme de configuration peut être envoyé à partir d'un PC, d'une station ou à partir d'un autre circuit FPGA. Le circuit FPGA maître peut être interfacé à une mémoire en mode parallèle ou série sans apporter aucune modification au niveau du câblage des circuits FPGA esclaves. C'est souvent le mode exploité pour la mise au point d'une configuration.

V. Conclusion

Dans ce chapitre nous avons présenté les différents circuits logiques programmables et concluons que la technologie FPGA s'inscrit au sommet de l'évolution des composants logiques programmables et ouvre de grandes perspectives en termes de coût et rapidité de conception.

Le chapitre suivant va être dédié à la présentation de la carte Virtex-II sur laquelle en va implémenter notre protocole.

Chapitre III :

Présentation de la carte de développement Virtex-II

Introduction

Le kit de développement Virtex-II, V2MB1000, fournit une solution complète pour développer des modèles et des applications basés sur la famille de FPGA Xilinx Virtex-II.

La carte système Virtex-II utilise le dispositif FPGA Virtex-II 1 million de portes (XC2V1000-4FG456C). Dans le paquet de réseau de grille en trouve 456 pins. La forte densité de portes et le grand nombre des E / S permet des solutions complètes des systèmes à mettre en œuvre.

La carte système comprend une mémoire de 16M x 16 DDR, deux sources d'horloge, port RS-232, des circuits de soutien. Une interface LVDS est munie d'une transmission de 16 bits en émission et en réception, ainsi qu'une horloge.

La famille Virtex-II FPGA a les fonctionnalités avancées nécessaires pour s'adapter aux exigences des applications de haute performance.

Le kit de développement FPGA Virtex-II fournit une excellente plate-forme pour explorer ces caractéristiques de sorte que vous pouvez rapidement et efficacement répondre à vos exigences.

II. La carte de développement Virtex-II

La figure (II.1) montre une photo de la carte et ces composants.

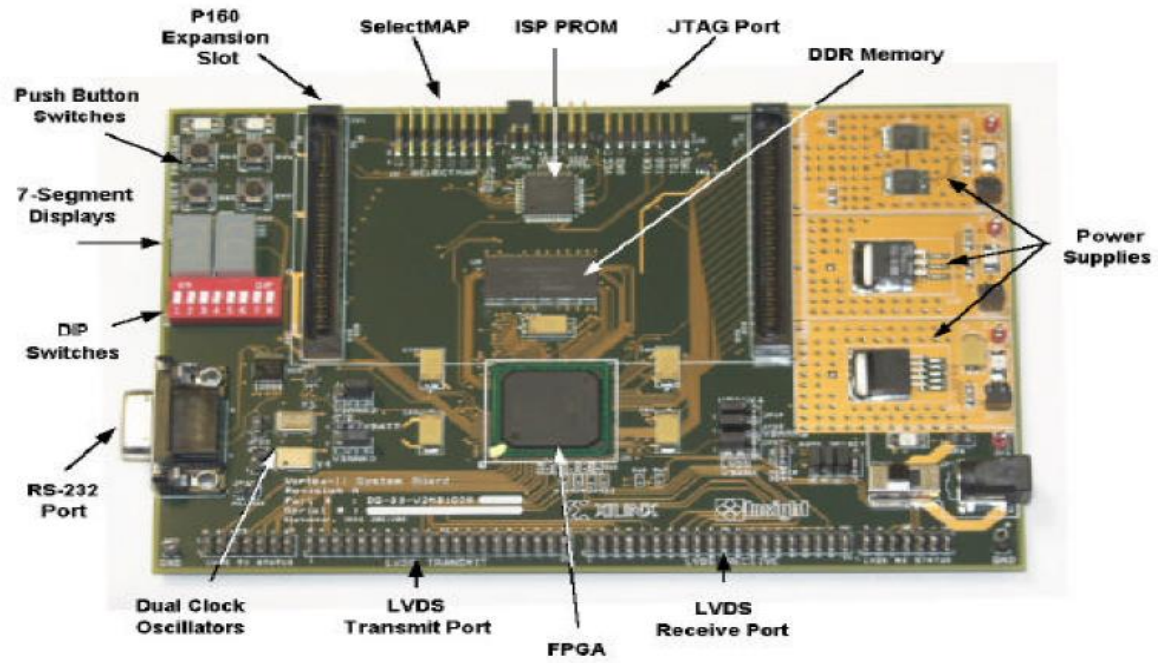


Figure (II.1) : carte de développement Virtex-II.

II.1 Description de la carte système Virtex-II

Un schéma bloc de la carte de développement FPGA Virtex-II, est représenté sur la figure (II.2), suivi par une brève description de chaque sous-section.

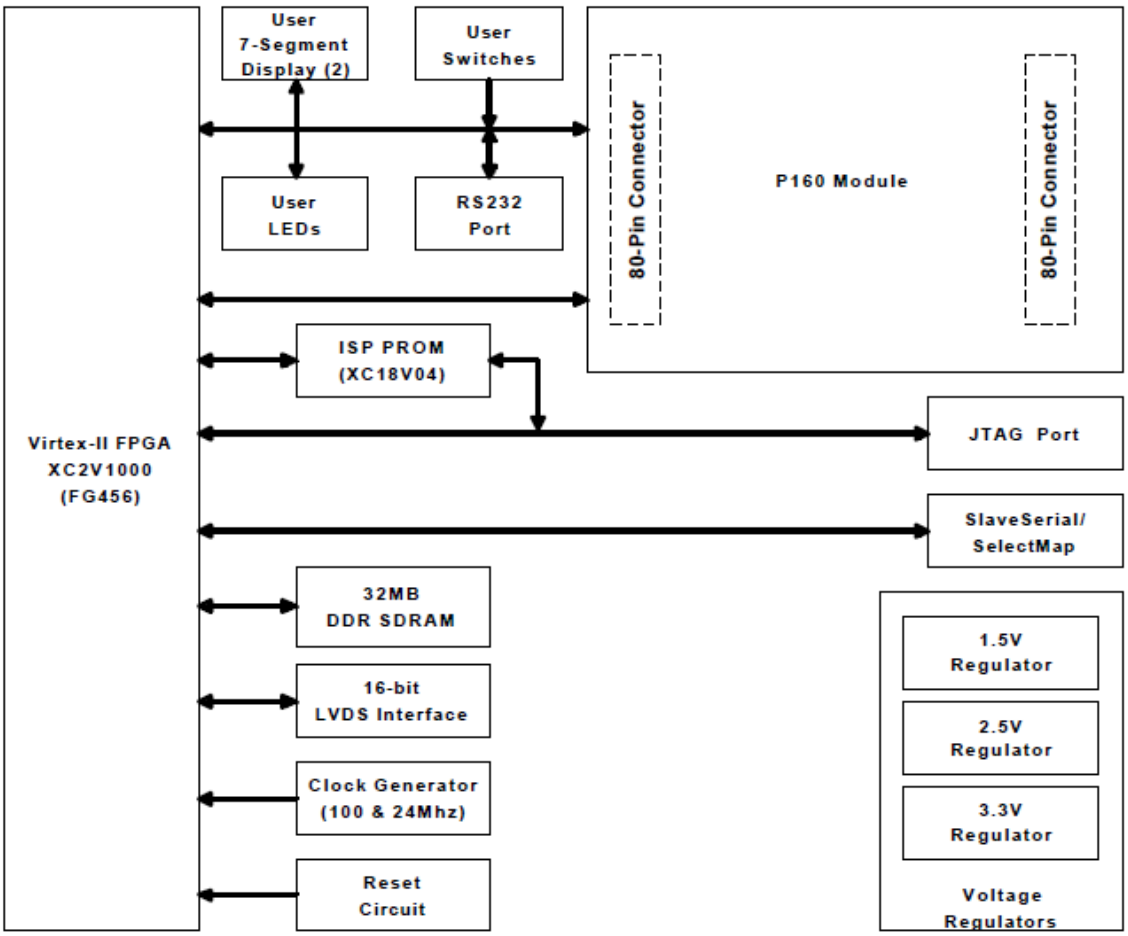


Figure (II.2) : schéma bloc de la carte de développement Virtex-II

II.2 Virtex-II Dispositif

La carte de développement Virtex-II utilise le dispositif Xilinx Virtex-II XC2V1000-4FG456C. La famille Virtex-II est une plate-forme FPGA développée pour la haute performance, utilisant des IP et des modules personnalisés.

La famille Virtex-II fournit des solutions complètes pour les télécommunications, sans fil, réseau, vidéo et DSP applications. La performance et la densité de la famille Virtex-II avec les E/S pris en charge tels que LVDS, PCI et DDR, permettent aux concepteurs de répondre aux exigences des applications de télécommunication et autres.

II.3 Mémoire DDR

La carte de développement Virtex-II fournit 32 Mo de mémoire DDR. Cette mémoire est mise en œuvre à l'aide du dispositif DDR Micron MT46V16M16TG-75 16Mx16. Le diagramme de haut niveau de l'interface DDR est illustré ci-dessous :

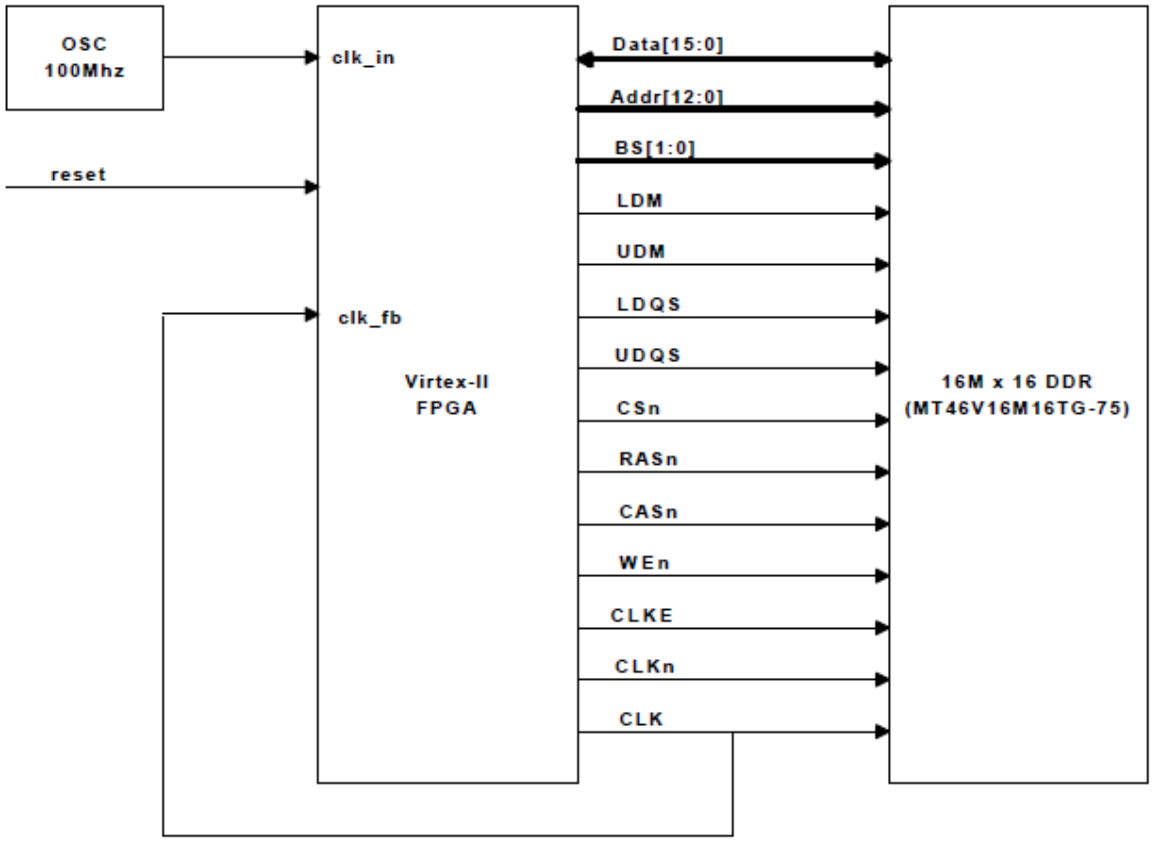


Figure (II.3) : Diagramme de haut niveau de l'interface DDR

II.4 Génération d'horloge

La carte de développement Virtex-II dispose de deux oscillateurs fonctionnant à 100 Mhz (CLK.CAN2) et 24 Mhz (CLK.CAN1). L'oscillateur fonctionnant à 100 Mhz est activé lorsque le cavalier JP24 est ouvert et désactivé lorsque JP24 est fermé. JP23 commande l'oscillateur 24 Mhz. Une troisième prise de l'horloge est fournie pour l'utilisateur.

Le tableau suivant fournit une brève description de ces signaux d'horloge :

Signal Name	Virtex-II Pin #	Direction	Description
CLK.CAN2	B11	Input	On-board 100 MHz Oscillator
CLK.CAN1	A11	Input	On-board 24 MHz Oscillator
CLK.CAN3	F12	Input	User clock socket (2.5V supply)

Tableau (II.1) : description des signaux d'horloge

II.5 Circuit de reset

La carte de développement Virtex-II utilise un dispositif de surveillance de tension TI TPS3125 pour surveiller le FPGA Virtex-II (1,5 V). Ce circuit fournit un signal de réinitialisation (FPGA_RESETn) au Virtex-II lorsque la tension de base de 1,5 V descend en dessous de ses spécifications minimales (1.425V). La remise à zéro envoie au FPGA une faible impulsion de 100ms fixes. En plus de surveiller la tension de base, ce circuit peut être utilisé pour générer une impulsion de remise à zéro en activant le Master Reset du signal (MRN) avec le dispositif TPS3125 via le commutateur de la carte bouton-poussoir (SW3).

La figure (II.4) suivante montre le circuit de réinitialisation sur la carte de développement FPGA Virtex-II.

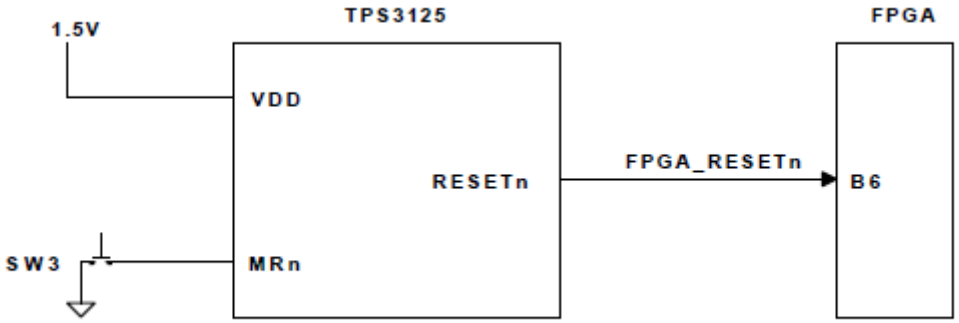


Figure (II.4) : circuit reset

II.6 Utilisation des afficheurs 7 segments

La carte de développement Virtex-II utilise deux afficheur 7 segments LED, à cathode commune qui peuvent être utilisés pendant l'essai et la phase de mise au point d'un dessin. L'utilisateur peut activer un segment donné par l'envoi d'un signal haut.

La figure (II.5) suivante montre le mode d'affichage à 7 segments interfacé avec le FPGA Virtex-II:

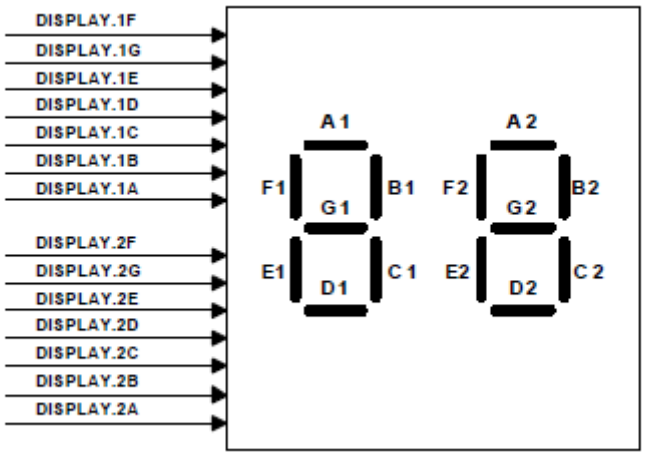


Figure (II.5) : afficheur 7 segments

Le tableau (II.2) suivant présente l'affectation des broches de l'afficheur 7 segments :

Signal Name	Virtex-II Pin #	Description
DISPLAY.1A	D9	7-Segment LED Display1, Segment A
DISPLAY.1B	C9	7-Segment LED Display1, Segment B
DISPLAY.1C	F11	7-Segment LED Display1, Segment C
DISPLAY.1D	F9	7-Segment LED Display1, Segment D
DISPLAY.1E	F10	7-Segment LED Display1, Segment E
DISPLAY.1F	D10	7-Segment LED Display1, Segment F
DISPLAY.1G	C10	7-Segment LED Display1, Segment G
DISPLAY.2A	B9	7-Segment LED Display2, Segment A
DISPLAY.2B	A8	7-Segment LED Display2, Segment B
DISPLAY.2C	B8	7-Segment LED Display2, Segment C
DISPLAY.2D	E7	7-Segment LED Display2, Segment D
DISPLAY.2E	E8	7-Segment LED Display2, Segment E
DISPLAY.2F	E10	7-Segment LED Display2, Segment F
DISPLAY.2G	E9	7-Segment LED Display2, Segment G

Tableau (II.2) : affectation des broches de l'afficheur 7 segments.

II.7 Utilisation de la LED

La carte de développement Virtex-II fournit une seule LED pour l'utilisateur qui est active pour un signal haut. Elle est reliée au Pin A9 du FPGA.

II.8 Utilisation des interrupteurs à bouton poussoir (SW5 et SW6)

La carte de développement Virtex-II dispose de deux entrées de commutation avec le FPGA Virtex-II qui sont des boutons poussoirs. Chaque commutateur à bouton poussoir peut être utilisé pour générer un signal actif au niveau bas.

Le tableau (II.3) suivant présente l'affectation des broches pour les boutons poussoirs :

Signal Name	Virtex-II Pin #	Description
FPGA.PUSH1	D7	User Push Button Switch Input 1 (SW5)
FPGA.PUSH2	A6	User Push Button Switch Input 2 (SW6)

Tableau (II.3) : affectation des broches des boutons poussoirs

II.9 User Switch DIP (SW2)

La carte de développement Virtex-II fournit 8 entrées de commutation pour l'utilisateur. Ces commutateurs sont statiques, mis à un niveau bas ou niveau haut logique.

La figure (II.6) suivante montre l'interface du commutateur DIP à l'FPGA Virtex-II.

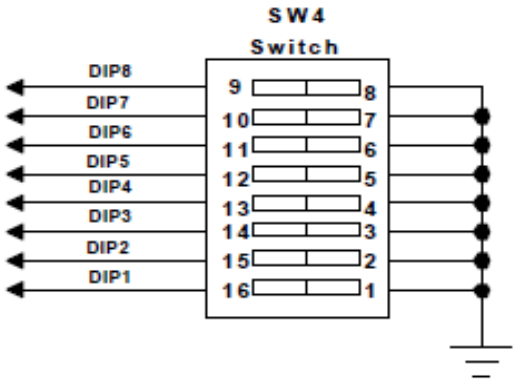


Figure (II.6) : interface commutateur DIP à l'FPGA

Le tableau (II.4) suivant présente l'affectation des broches du commutateur DIP :

Signal Name	Virtex-II Pin #	Description
DIP8	C6	User Switch Input 8
DIP7	D6	User Switch Input 7
DIP6	A5	User Switch Input 6
DIP5	B5	User Switch Input 5
DIP4	C5	User Switch Input 4
DIP3	C4	User Switch Input 3
DIP2	A4	User Switch Input 2
DIP1	B4	User Switch Input 1

Tableau (II.4) : affectation des broches du commutateur DIP

II.10 Port RS232

La carte de développement Virtex-II fournit un port RS232 qui peut être géré par le FPGA Virtex-II. Le sous-ensemble des signaux RS232 est utilisé sur la carte de développement Virtex-II pour mettre en œuvre cette simple interface (signaux TD et RD).

II.10.1 Interface RS232

La carte de développement Virtex-II dispose d'un port DB-9 pour connecter un simple port RS232. La carte utilise le pilote TI MAX3221 RS232 pour piloter les signaux RD et TD. L'utilisateur fournit le Code RS232 UART, qui réside dans l'FPGA Virtex-II.

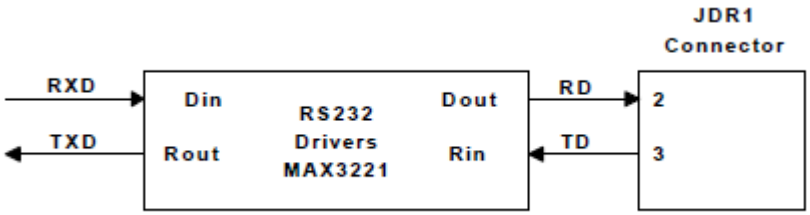


Figure (II.7) : interface RS232.

II.10.2 Description du Signal RS232

Le tableau (II.5) suivant montre les signaux RS232 et leurs affectations aux broches du FPGA Virtex-II.

Signal Name	Virtex-II Pin #	Description
RXD	A7	Received Data, RD to DB9
TXD	B7	Transmit Data, TD from DB9

Tableau (II.5) : affectation des broches du RS232

II.11 Port JTAG

La carte de développement Virtex-II fournit un connecteur JTAG qui peut être utilisé pour programmer la PROM de la carte et configurer le FPGA Virtex-II. Deux options de connexion sont fournies, J2 est utilisé pour la connexion standard JTAG, et JP29 est utilisé pour la connexion Xilinx parallèle IV avec câble JTAG.

II.11.1 Connecteur standard JTAG

La figure (II.8) suivante montre l'affectation des broches du connecteur JTAG J2 sur La carte de développement Virtex-II.

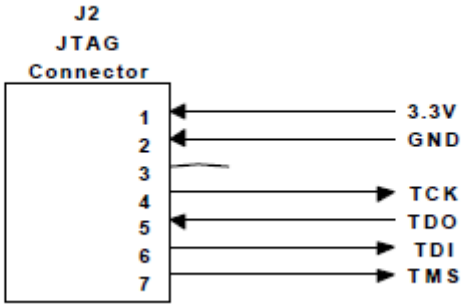


Figure (II.8) : Connexion J2 JTAG

II.11.2 Câble parallèle IV Port

La figure (II.9) suivante montre les connexions du connecteur du câble parallèle IV. Le Câble IV parallèle peut également être utilisé pour configurer le FPGA via mode Slave de configuration de série.

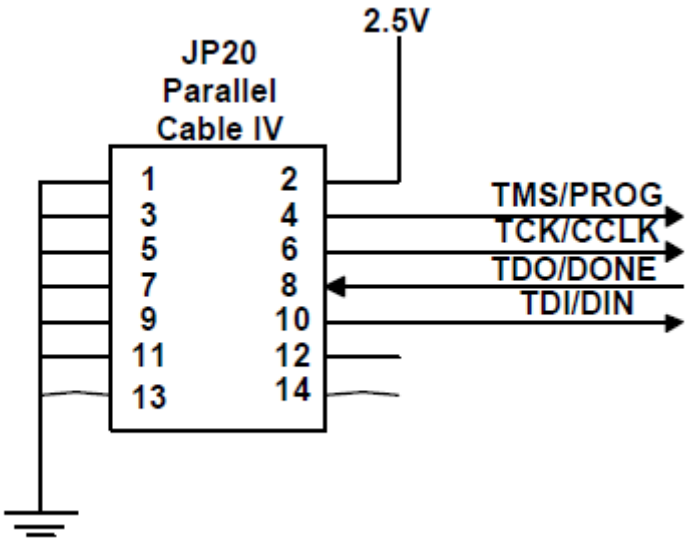


Figure (II.9) : Port parallèle IV JP29

II.11.3 Chaîne JTAG

La figure (II.10) suivante montre la chaîne JTAG sur la carte de développement Virtex-II. Le Jumper JP22 offre la possibilité de retirer la PROM de la chaîne JTAG pour la connexion directe à l'FPGA.

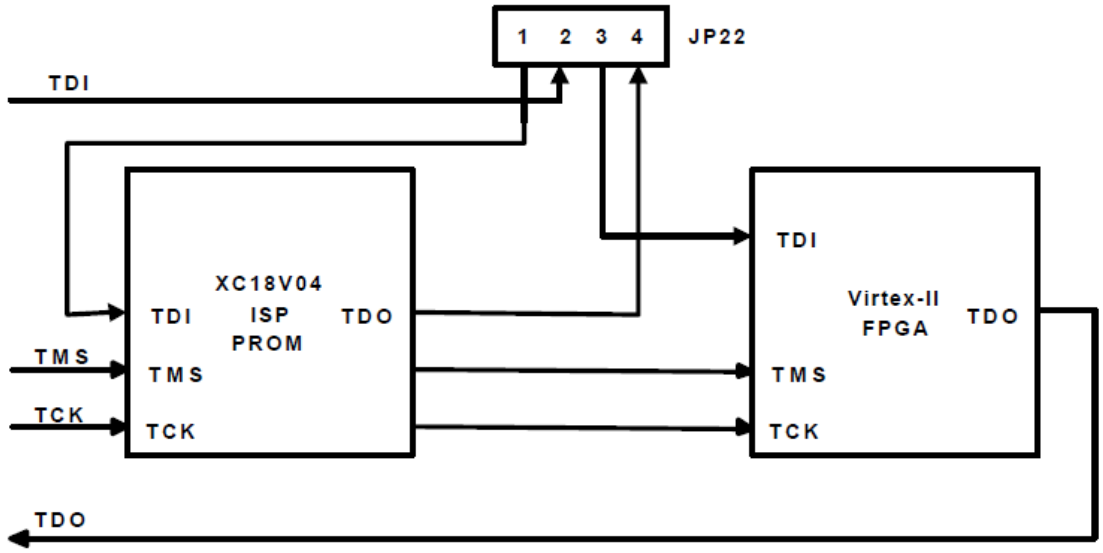


Figure (II.10) : chaîne JTAG de la carte de développement Virtex-II

II.11.4 Réglages des cavaliers dans la chaîne JTAG

Le tableau (II.6) suivant indique les cavaliers à mettre pour avoir la chaine JTAG sur la carte de développement Virtex-II:

Jumper	Setting	Description
JP28	1-2 Closed	Disable PROM
	2-3 Closed	Enable PROM (normal setting)
JP22	1-2, 3-4	PROM in chain (normal setting)
	2-3	Remove PROM from chain (FPGA only)

Figure (II.6) : configuration de la chaîne JTAG

III. Utilisation de la carte

La carte de développement Virtex-II prend en charge plusieurs méthodes de configuration du FPGA Virtex-II. Le port JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer directement le FPGA Virtex-II, ou de programmer le XC18V04 FAI PROM. Une fois la PROM programmée, elle peut être utilisée pour configurer le FPGA Virtex-II. Le port série SelectMap/esclave sur cette carte de développement peut également être utilisé pour configurer le FPGA Virtex-II.

La figure (II.11) ci-après montre le branchement pour tous les modes de configuration FPGA Virtex-II qui sont pris en charge sur La carte de développement Virtex-II.

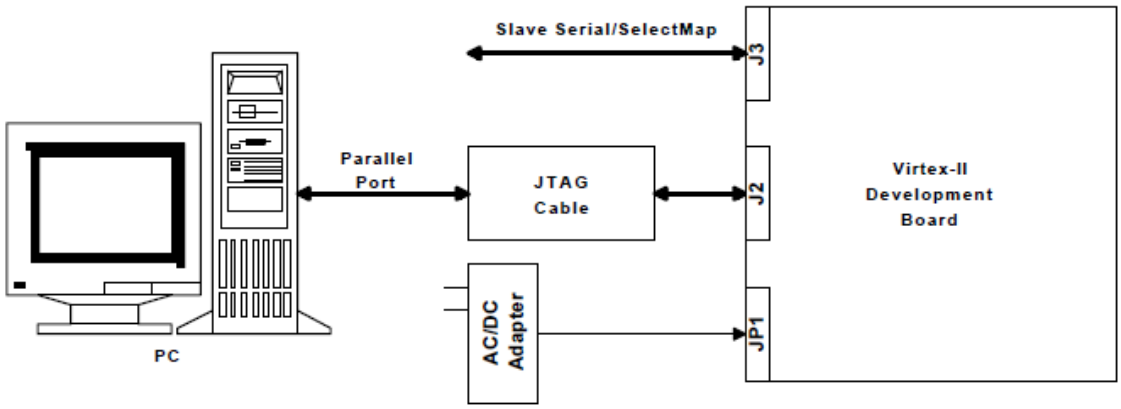


Figure (II.11) : branchement des modes de configuration

III.1 Interface JTAG

Le connecteur J2 JTAG sur la carte de développement FPGA Virtex-II peut être utilisé pour configurer le Virtex-II ou de programmer le XC18V04 FAI PROM. Le câble JTAG est connecté à la carte de développement Virtex-II via J2 à une extrémité et au port parallèle d'un PC à l'autre extrémité.

III.1.1 Configuration du FPGA Virtex-II

Lorsque le port JTAG est utilisé pour configurer le FPGA Virtex-II, les mesures suivantes doivent être prises:

- En utilisant le tableau (II.7) régler le mode de configuration du FPGA Virtex-II en mode JTAG.
- Utiliser l'utilitaire de programmation JTAG Xilinx (IMPACT) pour charger le fichier binaire de conception dans le FPGA Virtex-II, associer la PROM FAI soit à un fichier dummy.mcs ou à un fichier .bsd afin de permettre au logiciel de programmation JTAG de transmettre les données à la PROM.

III.1.2 Programmation de la XC18V04 FAI PROM

Lorsque le port JTAG est utilisé pour programmer le FAI PROM, effectuer les étapes suivantes :

- En utilisant le tableau (II.7) régler le mode de configuration du FPGA Virtex-II en Mode Master Serial ou Master SelectMap.
- Utiliser l'utilitaire de programmation JTAG Xilinx (IMPACT) pour charger le fichier de conception sur la PROM, associer le FPGA soit à un fichier fictif .bits ou à un fichier .bsd. permettant au logiciel de programmation JTAG de transmettre les données à l'FPGA.

Mode	PC Pull-up	J1			
		1-2 (M0)	3-4 (M1)	5-6 (M2)	7-8 (M3)
Master Serial	No	Closed	Closed	Closed	Closed
Master Serial	Yes	Closed	Closed	Closed	Open
Slave Serial	No	Open	Open	Open	Closed
Slave Serial	Yes	Open	Open	Open	Open
Master SelectMap	No	Closed	Open	Open	Closed
Master SelectMap	Yes	Closed	Open	Open	Open
Slave SelectMap	No	Open	Open	Closed	Closed
Slave SelectMap	Yes	Open	Open	Closed	Open
JTAG	No	Open	Closed	Open	Closed
JTAG	Yes	Open	Closed	Open	Open

Tableau (II.7) : configuration du mode sélectionné

Conclusion

Ce chapitre est dédié à la présentation de la carte Virtex-II, expliquant ses constituants que nous exploiterons dans le chapitre suivant qui sera consacré à la présentation de notre protocole et à son implémentation.

Chapitre 4 :

Solution et contribution

Introduction

Dans le chapitre I nous avons décrit la multiplication modulaire de Montgomery, le protocole de cryptage RSA et l'algorithme Squart and Multiply. En se basant sur ces derniers on va d'abord présenter notre protocole RSA, auquel nous avons abouti en se basant sur la multiplication modulaire de Montgomery et l'algorithme Squart and Multiply dans la perspective de réduire la taille afin de l'implémenter sur un FPGA. Par la suite on va programmer notre algorithme RSA en utilisant le langage VHDL et faire quelques simulations et on passera à l'environnement de développement ISE afin de l'implémenter et le tester sur notre carte de développement Virtex-II.

II. Présentation de notre protocole RSA

Notre protocole consiste à chiffrer un message M avec le protocole RSA qui est un protocole simple, fiable ayant fait ses preuves dans la cryptographie moderne ; surtout qu'il est facile à programmer sur un ordinateur. Mais ce qui pose problème c'est la taille mémoire nécessaire à son implémentation. Afin d'essayer de remédier à ça, on va utiliser la multiplication modulaire de Montgomery et l'algorithme Squart and Multiply qu'on a présentés dans le chapitre I.

II.1 Description de notre protocole RSA

On veut crypter un message m, en utilisant le chiffrement RSA. On va avoir à la sortie un message crypté c, tel que $c = m^e \bmod n$. Ce type de chiffrement peut nécessiter un grand espace mémoire et de calcul. Pour essayer de réduire cet espace, on va opter pour la multiplication modulaire de Montgomery.

Le protocole que nous proposons est basé sur l'algorithme square and multiply et la multiplication de Montgomery, qui va calculer $C = M^{\wedge e} \bmod n$ et qui va donner un résultat dans le domaine de Montgomery pour aboutir à une puissance qu'on appellera la puissance de Montgomery qui va nous donner le cryptage RSA en réduisant l'espace utilisé.

$\wedge$: puissance de Montgomery.

II.2 Algorithme de notre protocole RSA

On commence par calculer les paramètres de l'algorithme de Montgomery comme on l'a expliqué dans le chapitre I, dont on aura besoin aussi dans notre algorithme de puissance.

- On transforme m vers le domaine de Montgomery :

$$M = m*r \bmod n. \ (m = mont\ (m)).$$

- On écrit e sous forme binaire :

$$E = \text{écriture binaire de e.}$$

- On initialise les paramètres d'entrée et de sortie :

$$C\ (0) = r\text{-}n.$$

$$z\ (0) = M.$$

- On fait une boucle allant de 0 jusqu'à (n-1) et on calcule :

$$z\ (i+1) = z\ (i) \text{ x } z\ (i) \bmod n.$$

- On fait un test sur chaque bit de e et si e (i) = 1 on calcule :

$$C\ (i+1) = C\ (i) \text{ x } z\ (i) \bmod n.$$

- Sinon on calcule :

$$C\ (i+1) = c\ (i).$$

La sortie de notre module est un résultat C qui est dans le domaine de Montgomery, donc on doit le remettre dans le domaine naturel comme suit :

x : multiplication de Montgomery.

$$c = C * r^{-1} \bmod n.$$

III. Implémentation

Dans ce qui suit nous allons présenter l'implémentation de notre protocole matériel décrit avec le langage de description du matériel VHDL en utilisant Le navigateur de projet ISE.

Nous commençons ainsi par une brève description du langage VHDL, ainsi que le navigateur de projet ISE. Par la suite, nous présenterons l'algorithme de notre protocole RSA sous VHDL, et testeront sa fonctionnalité par des simulations, puis sur la carte de développement Virtex-II.

III.1. Langage de description VHDL

VHDL (VHSIC Hardware Description Language) est un langage de description pour les circuits numériques. Il a été développé dans le cadre du projet VHSIC (Very high speed integrated circuits) commandité par le département de la défense américaine DoD au début des années quatre-vingts. C'est un standard IEEE depuis 1987 largement utilisé en Europe. Il autorise plusieurs méthodologies de conception (comportemental, flot de données, structurel).

Il est indépendant de la technologie utilisée FPGA, CPLD, ASIC, etc., et permet d'aller d'un niveau d'abstraction très élevé par une description algorithmique jusqu'à un niveau proche du matériel, où l'on décrit le système par un ensemble de portes logiques et d'interconnexions. Entre les deux se trouve le niveau RTL (Register Transfer Level) qui permet de définir le système par une architecture de type machine de Moore ou Mealy. Cette abstraction permet d'ailleurs de simuler sur ordinateur avant de programmer le moindre circuit.

III.2. Environnement de développement ISE

Le navigateur de projet ISE sera utilisé comme outil de conception. Cet outil de Xilinx permet de créer des projets comportant plusieurs types de fichiers (HDL, schématique, UCF, EDIF, etc.), de compiler, d'effectuer des designs rule check (DRC), de créer des contraintes d'implémentation dont des contraintes de timing sur les horloges, de déterminer l'emplacement des broches, de créer des bancs d'essais de simulations (testbench) et de gérer efficacement les projets d'envergure.

Le navigateur de projet ISE offre un environnement de conception centralisé extrêmement efficace qui regroupe tous les outils nécessaires à la conception, la simulation

et à l'implémentation d'un projet d'envergure ainsi qu'à la configuration de la carte, Virtex-II en ce qui nous concerne.

III.3. simulation de notre protocole

Avant de passer à la simulation de notre protocole, on doit d'abord définir les paramètres du RSA comme on l'a vu dans le chapitre I :

- n = p*q = 13*17 = 221.
- Ø = (p-1)*(q-1) = 192.
- e premier avec Ø. e = 11.
- b = (inverse modulaire e) modulo Ø. b = 35.

Après avoir défini les paramètres du RSA, on définit les paramètres de notre protocole :

- R = puissance supérieure de n = 256.
- R^{-1} = (inverse modulaire R) modulo n = 120.
- Et définir v tel que : $R*R^{-1} - v*n = 1$ et v = 139.
- On choisit un m qu'on veut crypter : m = 203.

III.3.1. chiffrement avec notre protocole

L'algorithme présenté en annexe permettra le chiffrement de m avec notre protocole RSA. La figure qui suit illustre notre simulation.

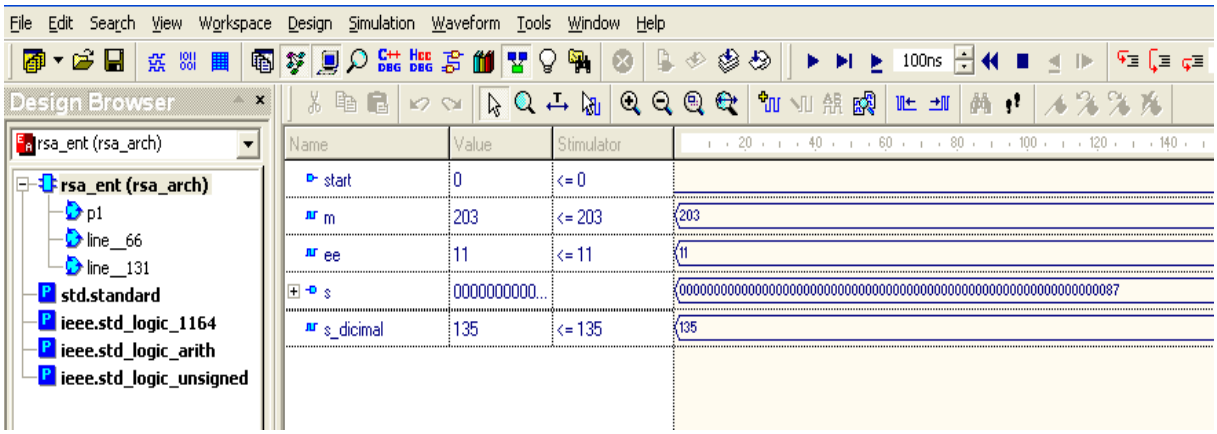


Figure VI.1 : simulation du chiffrement avec notre protocole RSA

- Start = 0, lancement du calcul.
- S est la sortie en hexadécimal.
- S_décimal est la sortie en décimal.

III.3.2 déchiffrement avec notre protocole

Pour le déchiffrement, on utilisera le même algorithme qu'on a présenté en annexe ; il suffit juste de changer la clé publique **« e »** par la clé privée **« b ».** La figure suivante illustre notre simulation.

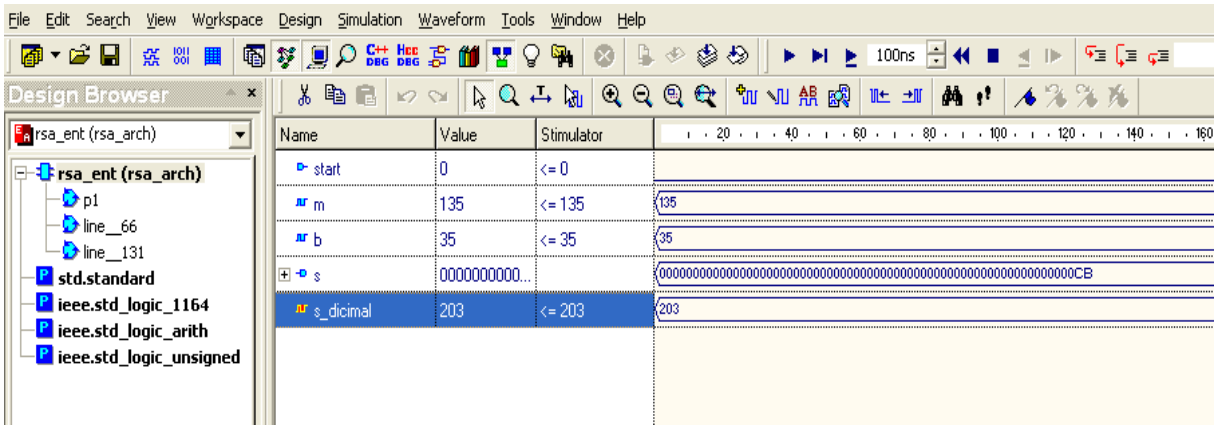


Figure VI.2 : simulation du déchiffrement avec notre protocole RSA.

IV. Implémentation de notre protocole sur la carte Virtex-II

L'implémentation sur la carte se fait à l'aide de l'environnement de développement ISE. Dans ce dernier on va introduire le code de notre protocole sous VHDL, comme le montre la figure suivante :

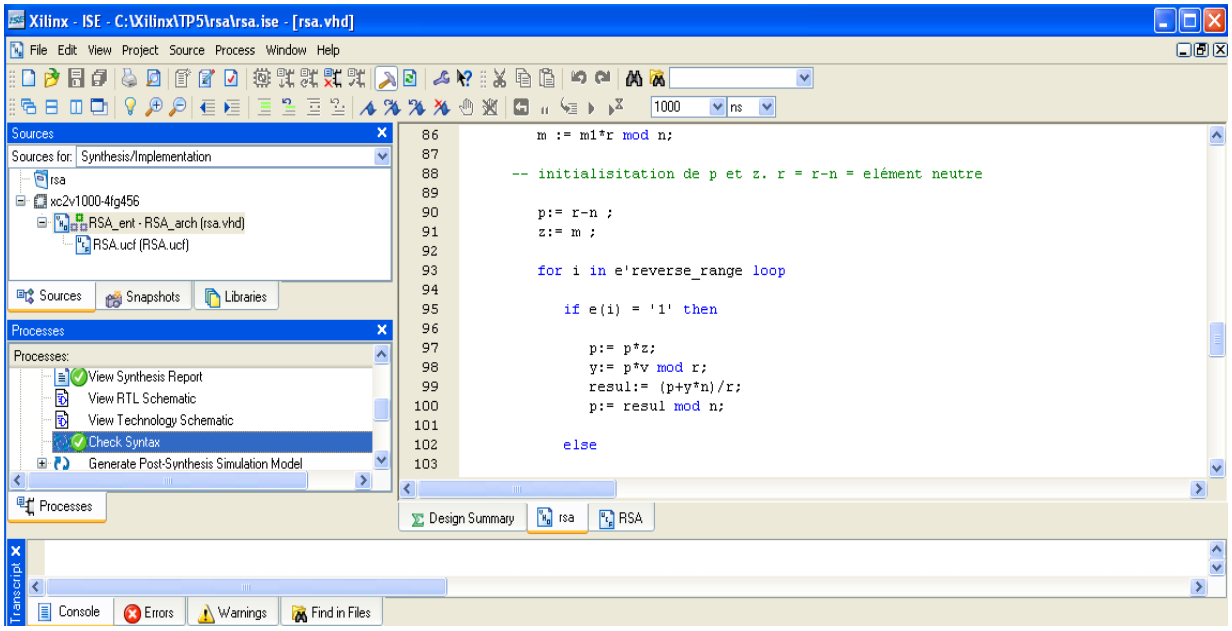


Figure VI.3 : notre protocole sous ISE

Une fois la saisie du code de notre protocole terminée, on doit vérifier s'il est correct. Pour cela aller dans process puis check syntax. (Voir figure suivante.)

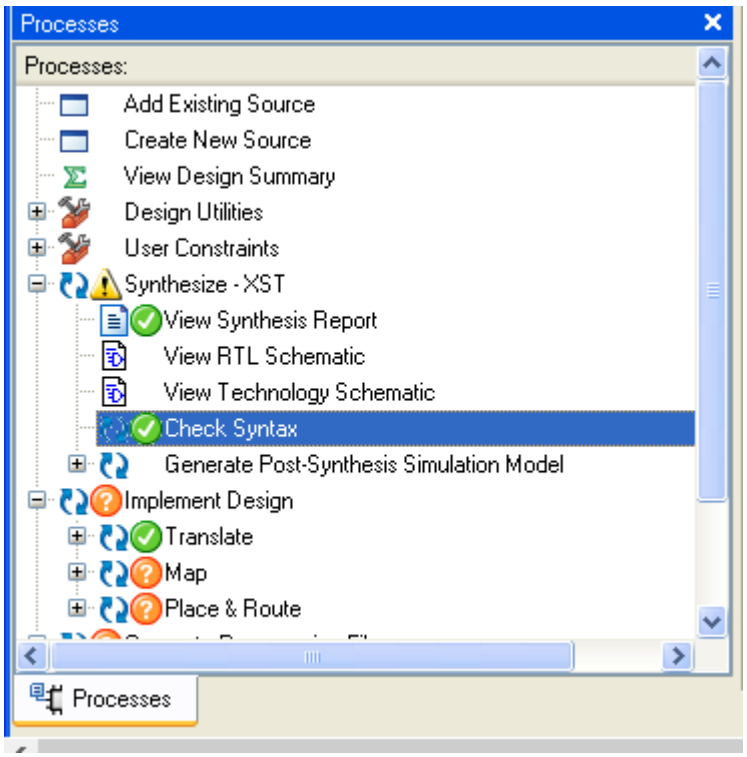


Figure VI.4 : vérification de la syntaxe du protocole

Arrivé à ce niveau, notre protocole introduit dans l'environnement ISE est correct. L'étape suivante consiste à affecter les entrées et les sorties aux pins du FPGA de la carte

Virtex-II. Pour cela on ouvre une nouvelle source nommée « implémentation contrainte file (.ucf) » et introduire l'affectation de chaque pin. Les figures suivantes montrent comment créer une source .ucf et l'affectation des pins :

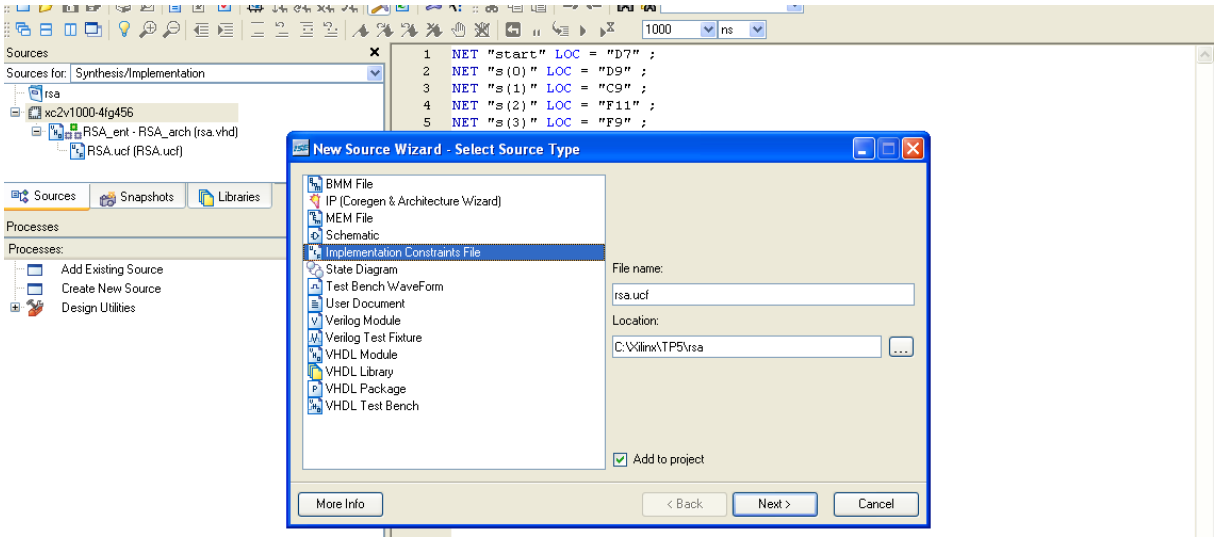


Figure VI.5 : créer une source .ucf

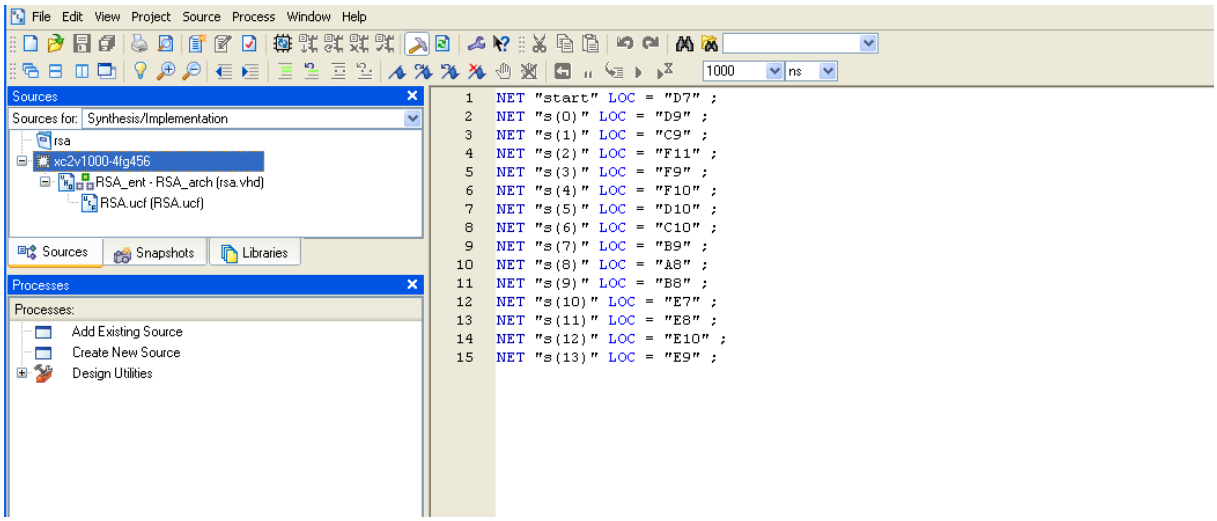
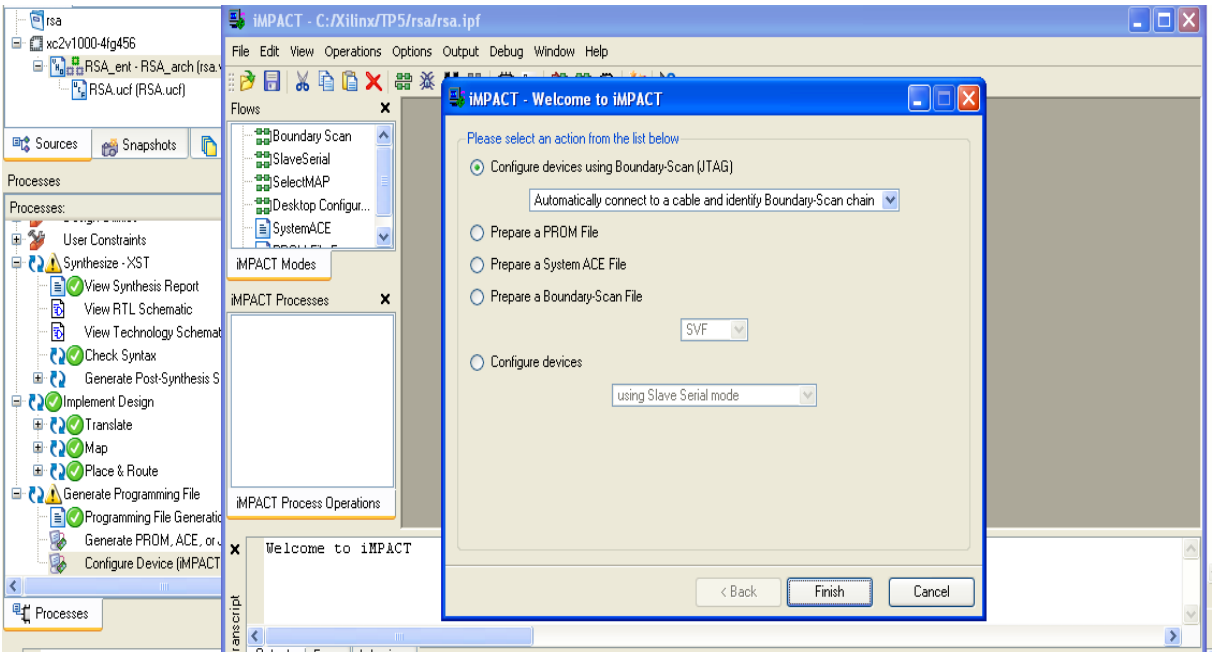


Figure VI.6 : affectation des pins au FPGA

Une fois ces étapes terminées avec succès, on va générer notre programme. Pour cela aller dans « processes » puis « generation programming file » pour obtenir le programme qu'on implémentera sur la carte.

Une fois la carte connectée à notre machine et le programme obtenu, on le chargera avec l'outil IMPACT se trouvant dans « processes » puis « configure device (impact) ».comme le montre la figure suivante :



FigureVI.7 : lancement d'IMPACT.

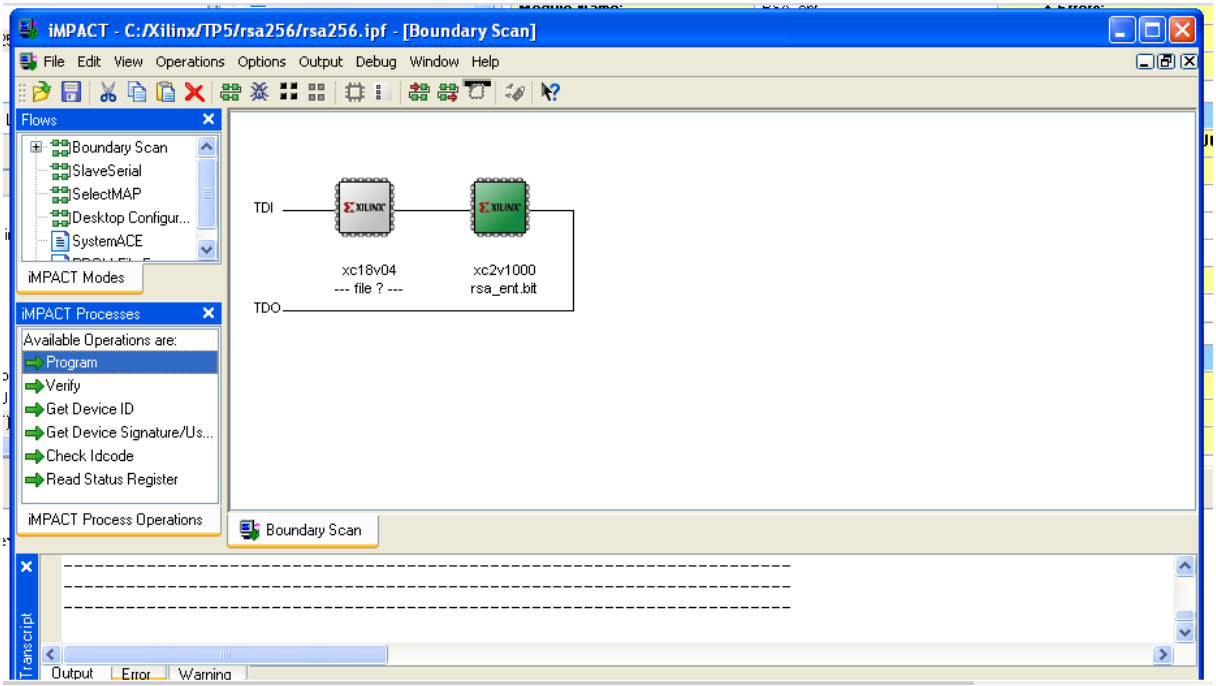


Figure VI.8 : chargement du programme

Arrivé à cette étape, on retrouve le résultat qui s'affiche sur les afficheurs 7 segments de notre carte comme le montre la figure suivante :

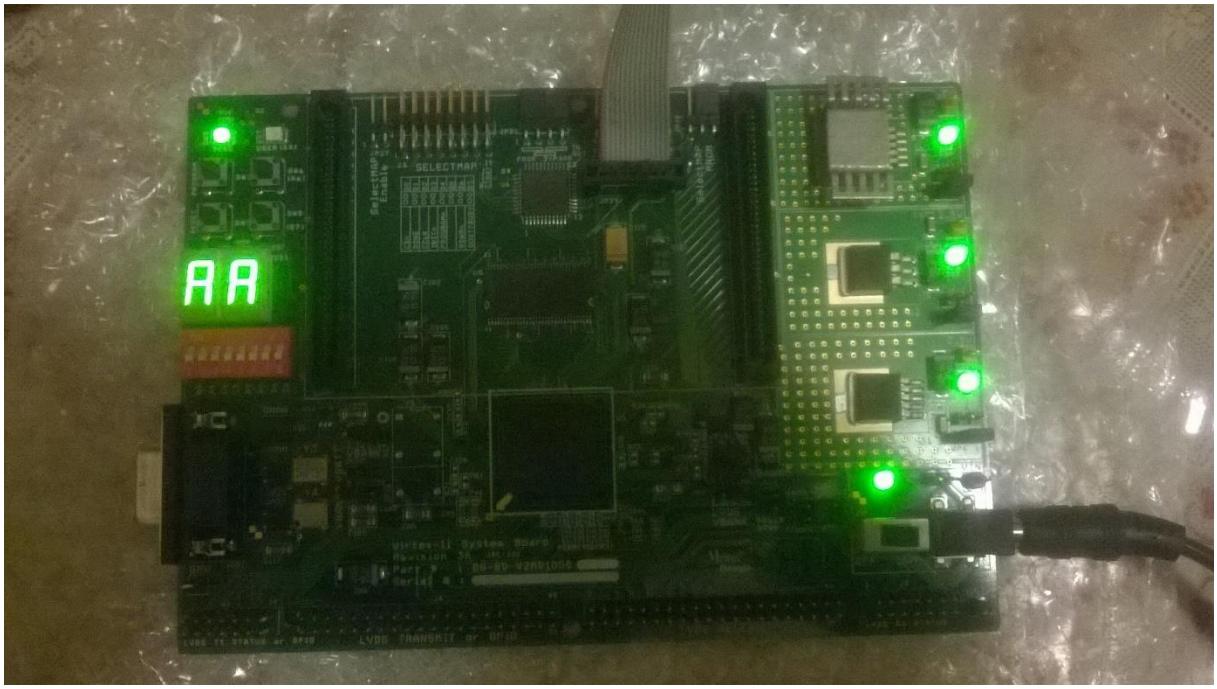


Figure VI.9 : affichage du résultat sur la carte Virtex-II

Vu qu'on ne dispose que de deux afficheurs à 7 segments et qu'on ne peut pas afficher le résultat en décimal, on a opté pour un test qui affiche un « A A » si c'est juste ou bien « 0 0 » si c'est faux.

Conclusion

Dans ce chapitre on a présenté notre protocole RSA basé sur la réduction modulaire de Montgomery et l'algorithme Squart and Multiply. Ce travail s'inscrit dans la contribution à la réduction de la taille du RSA pour pouvoir l'utiliser sur des systèmes embarqués.

Conclusion et perspectives

Les systèmes embarqués représentent un enjeu majeur dans les nouvelles technologies vu qu'ils équipent en grande partie les équipements et matériels utilisés au quotidien et ils ne cessent d'être le sujet de travaux de recherche afin de les développer.

Le RSA est l'un des protocoles les plus fiables consacrés à la sécurité des données grâce à des clés de 1024 bits.

Notre travail a consisté à trouver le moyen d'implémenter le RSA sur un système embarqué de type FPGA équipant la carte de développement Virtex-II sur laquelle a été chargé notre protocole RSA utilisant des clés de 256 bits.

Actuellement il existe des FPGA tels que le Virtex-5 ou le Virtex-7 sur lesquels peut être implémenté notre protocole en utilisant des clés de 1024 bits.

En conclusion, afin que notre système constitue un produit fini, on sera amené à lui associer un protocole de communication via SPI, Bluetooth ou I2C.

Annexes

Algorithme de notre protocole RSA

```
----------------------------------------------------------------------------------------------
--
-- Title       : RSA_ent
-- Design      : multiplier
-- Author      : mémoire
-- Company     : UMMTO
--
----------------------------------------------------------------------------------------------
--
-- File        : RSA.vhd
-- Generated   : Thu Jan 16 23:12:20 2014
-- From        : interface description file
-- By          : Itf2Vhdl ver. 1.20
--
----------------------------------------------------------------------------------------------
--
-- Description :
--
----------------------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity RSA_ent is
        port(
```

```
        start : in STD_LOGIC;
        s : out STD_LOGIC_vector(255 downto 0)
          );
end RSA_ent;


architecture RSA_arch of RSA_ent is


signal  res_aff : std_logic_vector(255 downto 0);
signal  e : std_logic_vector(255 downto 0);
signal calc : std_logic;


begin

        p1:process (start)

        constant e1: integer := 11;

        begin
                if start = '0' then

                -- convertir e on un std_logic_vector

                        e <= conv_std_logic_vector (e1,256);

                        calc <='1';
    else
                        calc <= '0';
```

```
end if;


end process      P1      ;

process(calc)

        variable p,resul,y,m,z,res : integer;

        -- declaration des paramétres de notre protocole

        constant m1: integer := 203;
        constant n: integer := 221 ;
        constant r: integer := 256;
        constant j: integer := 120 ;
        constant v: integer := 139 ;


begin



        if calc = '1' then

        -- transformation de m vers le domaine de Montgomery

                m := m1*r mod n;

        -- initialisitation de p et z. r = r-n = elément neutre
```

```
p:= r-n ;
z:= m ;

for i in e'reverse_range loop

    if e(i) = '1' then

        p:= p*z;
        y:= p*v mod r;
        resul:= (p+y*n)/r;
        p:= resul mod n;

    else

        p:= p;

    end if;

    z:= z*z;
    y:= z*v mod r;
    resul := (z+y*n)/r;
    z := resul mod n;

end loop;

-- transformation de notre resultat vers le domaine naturel

res := p*j mod n;

-- conversion du resultat en un std_logic_vector afin de
```

```
                -- pouvoire l'afficher sur un afficheur 7 segment

                res_aff <= conv_std_logic_vector (res,256);

                end if;

        end process;

        -- en fait un convertiseur 7 segment

s <=  res_aff;

end RSA_arch;
```

Bibliographie

- ✓ Thomas Plantard « Arithmétique modulaire pour la cryptographie » thèse Doctorale université de Montpelier II Sciences et Techniques du Languedoc.
- ✓ Bruce Scheneier, «Cryptographie appliquée », deuxième édition, Algorithmes, Protocoles et codes sources en C.
- ✓ Sylvain GUILLEY « Implémentation d'un multiplieur de Montgomery sécurisé et cascadable » http://biblio.telecom-paristech.fr/cgi-bin/download.cgi?id=5713
- ✓ Network Associates Inc. « Introduction à la cryptographie »
- ✓ Michel HUBEN http://michel.hubin.pagespersoorange.fr/physique/microp/chap_mp5.htm.
- ✓ D. AIT AZZI, R.TIKOUBANI, « multiplieur en ligne pour le calcul en longe précision sur circuit FPGA » thèse ingéniorat, UMMTO, 2002.
- ✓ Z. AIT OUALI, « application des FPGA a la commande d'un moteur asynchrone » thèse magister, UMMTO, 2011.
- ✓ D. HADJ SAID, « implémentation d'une application de tracking sur FPGA » thèse master, UMMTO, 2013.
- ✓ B. ALOUINI, « implementation of IP cores dedicated to cryptography », thèse magister, UMBB, 2012.
- ✓ Virtex-II™ V2MB1000 Development Board User's Guide.

Site web :

- ✓ http://fr.wikipedia.org/wiki/Portail:Cryptologie
- ✓ http://www.acrypta.com/index.php/telechargements
- ✓ http://arabia.ni.com/
- ✓ http://tima.imag.fr/tima/fr/mediatheque/publications.html