Logique programmable asynchrone pour systèmes embarqués sécurisés

T. Beyrouthy

▶ **To cite this version:**

T. Beyrouthy. Logique programmable asynchrone pour systèmes embarqués sécurisés. Micro et nanotechnologies/Microélectronique. Institut National Polytechnique de Grenoble - INPG, 2009. Français. NNT : . tel-00481895

HAL Id: tel-00481895
https://theses.hal.science/tel-00481895

Submitted on 7 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
978-2-84813-141-2

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité : « MICRO ET NANO ELECTRONIQUE »

préparée au laboratoire TIMA dans le cadre de **l'Ecole Doctorale**

« Électronique, Électrotechnique, Automatique, Traitement du Signal »

présentée et soutenue publiquement par

Taha Beyrouthy

Le 2 Novembre 2009

LOGIQUE PROGRAMMABLE ASYNCHRONE POUR SYSTÈMES EMBARQUÉS SÉCURISÉS

Directeur de thèse : Laurent Fesquet

Co-directeur de thèse : Marc Renaudin

JURY

M. Habib Merhez	Président
M. Lionel Torres	Rapporteur
M. Victor Fischer	Rapporteur
M. Laurent Fesquet	Directeur de thèse
M. Marc Renaudin	Co-encadrant
M. Jean-Luc Danger	Examinateur

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
978-2-84813-141-2

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité : « MICRO ET NANO ELECTRONIQUE »

préparée au laboratoire TIMA dans le cadre de **l'Ecole Doctorale**

« Électronique, Électrotechnique, Automatique, Traitement du Signal »

présentée et soutenue publiquement par

Taha Beyrouthy

Le 2 Novembre 2009

LOGIQUE PROGRAMMABLE ASYNCHRONE POUR SYSTÈMES EMBARQUÉS SÉCURISÉS

Directeur de thèse : Laurent Fesquet

Co-directeur de thèse : Marc Renaudin

JURY

M. Habib Merhez	Président
M. Lionel Torres	Rapporteur
M. Victor Fischer	Rapporteur
M. Laurent Fesquet	Directeur de thèse
M. Marc Renaudin	Co-encadrant
M. Jean-Luc Danger	Examinateur

2 Novembre 2009

LOGIQUE PROGRAMMABLE ASYNCHRONE POUR SYSTÈMES EMBARQUÉS SECURISÉS

Cette thèse porte sur la conception et la validation d'un FPGA dédié à des applications sensibles nécessitant un haut niveau de sécurité et de confidentialité. Les FPGAs usuels présentent de nombreuses failles vis-à-vis de la sécurité : 1-Ils ne permettent pas une implémentation efficace de circuits logiques alternatifs, tels que les circuits asynchrones. 2-Le placement et le routage d'un circuit ne peuvent être complètement maîtrisés pour garantir une conception sécuritaire. 3-Ils ne sont pas protégés contre les attaques par canaux cachés tels que la DPA, l'EMA ou la DFA. Afin de lever ces obstacles technologiques, les travaux entrepris dans cette thèse ont permis de proposer une architecture autorisant la programmation de différents styles de circuits asynchrones, de disposer d'un système de programmation compatible avec les objectifs de sécurité et d'une conception garantissant un haut niveau de protection vis-à-vis des attaques citées ci-dessus. Enfin, une validation matérielle du prototype a permis d'appréhender la pertinence des modèles développés.

Taha Beyrouthy

Laboratoire TIMA

46 avenue Félix Viallet

38000 Grenoble Cedex

A mes parents

Remerciements

Mon tout premier remerciement est à Laurent Fesquet, mon directeur de thèse et responsable du groupe de recherche CIS, qui a su insuffler en moi le désir d'apprendre et le plaisir de réfléchir. Il a toujours donné sans compter pour que je puisse m'épanouir et exprimer le meilleur de moi-même dans mon travail. Laurent fait partie de ces personnes à qui je dois énormément et que j'apprécie sincèrement.

Je remercie aussi Marc Renaudin mon ex-directeur de thèse pour tout le temps qu'il m'a consacré et le soutien qu'il a su m'accorder pour bien démarrer ma thèse.

Je suis particulièrement reconnaissant envers Alin Razafindraibe pour son aide et son apport de connaissances techniques.

Un grand merci à Lionel Torres et Victor Fischer qui ont accepté d'être les rapporteurs de mon mémoire et de consacrer une partie précieuse de leur temps à sa relecture. Je suis aussi reconnaissant à Habib Merhez et Jean-Luc Danger d'avoir bien voulu faire partie de mon jury.

J'ai eu l'occasion de travailler à Telecom ParisTech, où j'ai passé de très bons moments et ce grâce à Sylvain Guilley, Sumantha Chaudhuri et à nouveau Jean-Luc Danger que je remercie fortement.

Je sais gré le personnel du CIME, Alexandre Chagoya, Robin Rolland et Bernard Bosc, d'avoir toujours mis à ma disposition les outils dont j'avais besoin pour mon travail. Je remercie aussi toute l'administration du Laboratoire TIMA, et particulièrement Sophie Martineau, Anne-Laure Fourneret-Itié, Corine Durand-Viel et Lucie Torella qui ont toujours été d'une grande disponibilité et dune patience exemplaire.

Je voudrais également remercier Livier Lizarraga, Cedric KochHofer, Gregory Lopin 'Greg' et mon collègue Docteur David Rios, qui sont devenus des amis avec les quels je passe des agréables moments

Je ne voudrais pas finir sans remercier l'ensemble de l'équipe CIS, Saeed, Eslam, Hatem, Oussama, Hawraa, Florant, Alexandre, khaled, Jérémie, Olivier, Mathieu et Franc

pour ses enrichissantes discussions et ses cours particuliers de français de son origine à nos jours. Je remercie également Hakim et à nouveau Docteur David Rios pour le plaisir que nous avons partagé dans la création de l'association d'accueil des doctorants de TIMA : A²DT.

Enfin qu'il me soit permis de rendre quelques hommages plus personnels.

A mes sœurs Nissrine et Sarah, à qui j'envoie mes plus tendres pensées et à qui je souhaite tout le succès et le bonheur du monde.

A Khaled k² le frère que je n'ai jamais eu et qui pourtant était toujours là pour moi comme un frère.

A Wiss, Houss, Basch, Abboudé et Samir simplement d'avoir été présents dans ma vie et pour qui j'éprouve une grande admiration.

Je souhaite exprimer ici ma profonde estime et grande considération envers ma mère et mon père pour tout ce que je leur dois déjà depuis 27 années et pour tout ce que je leur devrais pour les années du reste de ma vie. Ils m'ont aimé et guidé dans les premières étapes de ma vie. Ils ont cru en moi, sacrifié leur bonheur et consacré leur vie pour faire de moi l'homme que je suis.

Finalement, ''khitamouha misk'', je voue une immense et sincère gratitude à ma merveilleuse, magnifique et formidable femme Nada, ou devrais-je dire *Huny Buny*, qui m'a soutenu pendant toutes ces années, qui a été et sera derrière toutes mes réussites, qui croit en moi, me pousse à atteindre les sommets et auprès de qui je veux passer toute ma vie.

TABLE DES MATIÈRES

Chapitre 1. INTRODUCTION – CONTEXTE & MOTIVATION _____ 15

Chapitre 2. ETAT DE L'ART _____ 19

2.1 La cryptographie _____ 19

2.1.1 La cryptographie symétrique _____ 20

2.1.2 La cryptographie asymétrique _____ 23

2.1.3 Cryptanalyse & attaques _____ 24

2.2 La logique asynchrone pour la sécurité _____ 33

2.2.1 La logique asynchrone : notions de base _____ 34

2.2.2 Les différentes classes de circuits asynchrones _____ 35

2.2.3 Le protocole de communication _____ 40

2.2.4 Les circuits QDI asynchrones pour la sécurité _____ 43

2.3 Utilisations des FPGAs pour les applications cryptographiques _____ 45

2.3.1 Avantages des FPGAs pour la cryptographie _____ 46

2.3.2 Les défauts de sécurité des FPGAs _____ 47

2.4 Conclusion _____ 51

Chapitre 3. SPÉCIFICATION D'UN FPGA ASYNCRHONE POUR LA SÉCURITÉ _ 53

3.1 Aspects fonctionnels – buts et objectifs _____ 54

3.1.1 Implémentation du protocole 4 phases _____ 55

3.1.2 Implémentation du protocole 2 phases LEDR _____ 67

3.1.3 Implémentation du protocole 2 phases sur front _____ 76

3.2 Aspects sécuritaires du bloc programmable _____ 84

3.2.1 Le codage des données _____ 85

3.2.2 Profondeur logique – temps de propagation unique _____ 87

3.2.3 Logique asynchrone _____ 90

3.3 Aspects sécuritaires – réseau d'interconnexion _____ 91

3.3.1 Placement et routage sécurisé _____ 91

3.4 Conclusion _____ 91

Chapitre 4. CONCEPTION D'UN FPGA ASYNCRHONE POUR LA SÉCURITÉ __ 93

4.1 Architecture logique _____ 94

4.1.1 Technique de mémorisation _____ 94

4.1.2 Chaine de programmation asynchrone _____ 96

4.1.3 Bloc programmable de base – LUT6 _____ 98

4.1.4 Le « Logic-Element » : 'LE' _____ 102

4.1.5 Le « Programmable Logic Bloc » – PLB _____ 104

4.1.6 La *« Switsh-box »* & la *« connection box »* _____ 105

4.2 Architecture électrique _____ 106

4.2.1 Chaine de programmation asynchrone _____ 106

4.2.2 Bloc programmable de base – LUT6 _____ 108

4.3 Le LayouT Du FPGA _____ 110

4.3.1 Le FPGA Asynchrone : layout et die view _____ 113

4.4 Conclusion _____ 113

Chapitre 5. RÉSULTATS EXPÉRIMENTAUX – TESTS ET VALIDATION _____ 115

5.1 Cas d'une S-BOX asynchrone _____ 118

5.1.1 Implémentation en 4 phases, double rails _____ 119

5.1.2 Implémentation en 2 phases, doubler rails _____ 126

5.2 Test du FPGA fabriqué _____ 131

5.2.1 Nouvelle version _____ 134

5.3 Conclusion _____ 135

Chapitre 6. CONCLUSION ET PERSPECTIVES _____ 137

Bibliographie _____ 141

Bibliographie de l'auteur _____ 151

LISTE DES FIGURES

Figure 1: La fonction de tour du DES 22
Figure 2: La fonction de tour du DES 23
Figure 3: Les différentes méthodes de cryptanalyses et d'attaques 26
Figure 4: Communication poignée de main ou Handshake 34
Figure 5: Classes des circuits asynchrones 36
Figure 6: Equivalence entre les circuits SI et QDI 38
Figure 7: Structure de base des circuits micropipelines 39
Figure 8: Codage donnée groupées 41
Figure 9: Codage double rail : 3-états et 4-états 42
Figure 10 : Protocole 4 phases double rails - codage 3-états. 56
Figure 11 : Protocole 4 phases double rails 56
Figure 12 : Symbole et table de vérité du C-Elément 57
Figure 13 : Implémentation d'une porte de Muller dans une LUT à 3 entrées 58
Figure 14 : La taille minimale d'un bloc programmable PLB 60
Figure 15 : Architecture du PLB modifiée pour supporter les fonctions 7 ↦ 1. 62
Figure 16 : Codage triple rails (1 parmi-3), 4 phases 63
Figure 17 : Implémentation d'une fonction à 2 entrées triple rails en 4 phases 64
Figure 18 : Calcul du signal d'acquittement ack_{out} 65
Figure 19 : Architecture du PLB pour l'implémentation du protocole 4 phases 66
Figure 20 : Protocole 2 phases 67
Figure 21 : Protocole 2 phases LEDR. 68
Figure 22 : Implémentation d'une fonction à 2 entrées LEDR 71
Figure 23 : implémentation d'une fonction à 3 entrées non équivalentes en LEDR 72
Figure 24 : Le premier étage : Bloc de calcul de parité 73
Figure 25 : Le premier étage modifié : Bloc de calcul de parité 74
Figure 26 : Le deuxième étage : les sorties de la fonction 2 entrées LEDR 75
Figure 27 : Nouvelle architecture du PLB pour l'implémentation du 4 phases et du 2 phases LEDR 76
Figure 28 : Protocole de transmission de données à 2 phases sur front. 77
Figure 29 : Schéma général d'une porte à deux entrées n logique 2 phases sur front 77
Figure 30 : Bloc Decision-Wait 2x2 à base de portes de Muller 78
Figure 31: Nouvelle architecture du PLB 80
Figure 32 : Implémentation d'un D-W2x2 dans le PLB 81
Figure 33 : Décision-wait 2x1 83

Figure 34 : Implémentation du Décision-wait 2x1 ____ *83*
Figure 35 : Version multi-styles du PLB ____ *84*
Figure 36 : Dernière version du PLB non équilibrée ____ *88*
Figure 37 : Version de l'architecture du PLB équilibrée ____ *89*
Figure 38 : Version finale du PLB équilibré ____ *90*
Figure 39 : architecture à base de multiplexeurs d'une LUT à n-entrées ____ *94*
Figure 40 : architecture à base de multiplexeurs d'une LUT à 3-entrées ____ *95*
Figure 41 : architecture à base de multiplexe d'une LUT à n-entrées ____ *96*
Figure 42 : architecture de la mémoire de programmation ____ *97*
Figure 43 : Circuit 'bouchon', pour initialiser la FIFO asynchrone ____ *98*
Figure 44 : LUT à base de Multiplexeurs ____ *99*
Figure 45 : architecture logique d'une LUT équilibrée ____ *100*
Figure 46 : architecture logique du décodeur 6→64 ____ *101*
Figure 47 : Architecture du Logic Element 'LE' ____ *103*
Figure 48 : Architecture du Programmable Logic Bloc PLB ____ *104*
Figure 49 : A droite, le champ électromagnétique dans une paire torsadée et dans une paire parallèle - A gauche deux fils équitemporels générés par une Switch-box à paire torsadée. ____ *105*
Figure 50 : A droite: twist-on-turn. A gauche : Twist-always . ____ *106*
Figure 51 : Architecture électrique du Half buffer asynchrone ____ *107*
Figure 52 : Architecture électrique du circuit bouchon ____ *108*
Figure 53 : Architecture électrique d'une des 6 entrées du décodeur de la LUT6 ____ *109*
Figure 54 : Layout du Full buffer asynchrone ____ *110*
Figure 55 : Layout du PLB – niveau métal 2 ____ *111*
Figure 56 : Layout du PLB –niveau métal 3 ____ *111*
Figure 57 : Layout du PLB complet. ____ *112*
Figure 58 : Layout du FPGA asynchrone ____ *113*
Figure 59: PLB bloc programmable du FPGA ____ *117*
Figure 60: S-Box1 de l'algorithme DES ____ *118*
Figure 61: Circuit S-Box + XOR : 12 entrées double rails, et 4 sorties double rails ____ *120*
Figure 62: Calcul des valeurs des 8 équations des sorties intermédiaires de la S-Box ____ *122*
Figure 63: bloc de test du protocole 4 phases ____ *123*
Figure 64: bloc final de calcul des sorties de la S-Box ____ *124*
Figure 65: Profil de courant de la S-Box + XOR 4 phases ____ *125*
Figure 66: Sortie de la S-Box + XOR 4 phases ____ *126*
Figure 67: Calcul des sorties intermédiaires de la S-Box 2 phases ____ *128*
Figure 68: bloc de test : reconstruction du protocole de communication 2 phases LEDR ____ *129*
Figure 69: Profile de courant de la S-Box en 2 phases LEDR ____ *130*
Figure 70: Temps d'exécution de la S-Box en 2 phases LEDR ____ *130*
Figure 71: Prototype 3x3 du FPGA (CMP Run S65C8-1) ____ *131*
Figure 72: Carte de test Altera + FPGA asynchrone ____ *132*
Figure 73: Architecture de la LUT6 ____ *133*
Figure 74: Simulation fonctionnelle d'une LUT6 ____ *133*
Figure 75: Nouvelle version de la LUT6 ____ *134*
Figure 76: Simulation de la nouvelle version de la LUT6 ____ *135*

Chapitre 1.
INTRODUCTION – CONTEXTE & MOTIVATION

La conception des circuits micro-électroniques a été pendant longtemps synonyme de conception ASIC. En effet, les ASICs présentent différents avantages tels que : haute vitesse, faible consommation de traitement, etc.… L'utilisation de circuits programmables a débuté dans les années 80. Leur principal avantage était un temps de cycle de conception très court (comparativement à un ASIC). Cependant, leurs faibles performances et leurs capacités de traitement ne leur ont pas permis de se substituer aux ASICs. Leur utilisation a été longtemps cantonnée aux tests de petits circuits prototypes.

Ces dernières années, le gap entre FPGA[1]s et ASICs s'est considérablement réduit. Les FPGAs de nos jours, sont non seulement utilisés pour prototyper, mais sont aussi devenus des composants majeurs dans les systèmes embarqués. Ils sont utilisés dans tous types d'applications, dont certaines manipulent des données sensibles qui doivent être protégées à tout prix. En effet, la confidentialité des données transmises dans les réseaux ou contenues dans les terminaux de communications est devenue une exigence essentielle qui doit être prise en charge par les fournisseurs des systèmes de communications sécurisés. Ainsi les concepteurs des FPGAs ont été amenés à prendre en compte la sécurité de ces derniers, tant au niveau matériel que logiciel.

[1] Field Programmable Gate Array

Les différentes méthodes de cryptanalyse et d'attaque menacent aujourd'hui la sécurité de ces systèmes. Classées en attaques intrusives et attaques non intrusives, elles exploitent les faiblesses et les défauts de l'implémentation matérielle des circuits cryptographiques pour obtenir les informations confidentielles. Une classe particulière de ces attaques, les attaques dites « par canaux cachés », sont d'une efficacité redoutable. Elle représente de nouveaux défis auxquels les concepteurs se doivent d'apporter des solutions.

De nombreux travaux de recherches ont montré que les circuits asynchrones possèdent des propriétés intrinsèques (et notamment les circuits quasi insensibles aux délais) propres à mieux sécuriser ces circuits que leurs homologues synchrones face aux attaques par canaux cachés.

Dans cette optique, les travaux de recherches menés dans le cadre ce travail, visent à spécifier, à concevoir et à valider un FPGA embarqué asynchrone pour des applications sensibles nécessitant un haut niveau de sécurité. En effet, les FPGAs ordinaires présentent de nombreuses failles vis-à-vis de la sécurité :

- Ils ne sont pas prévus pour supporter des styles de circuits logiques alternatifs tels que les circuits asynchrones.

- Ils ne sont pas protégés contre certains types d'attaques par canaux cachés tels que la DPA (Differential Power Analysis) ou la DFA (Differential Fault Attack).

Afin de lever ces obstacles technologiques, les travaux entrepris dans le cadre de cette thèse ont permis de proposer une architecture alternative capable de supporter la programmation de différents styles de circuits asynchrones, ainsi qu'un système de programmation du FPGA prenant en compte ces spécificités sécuritaires, l'objectif étant de garantir un haut niveau de protection vis-à-vis des attaques citées ci-dessus.

Le deuxième chapitre commence par une présentation des différents types de cryptanalyses (logicielles et théoriques) et d'attaques. Il se focalise plus particulièrement sur les attaques par canaux cachés. Les notions de base de la logique asynchrone sont ensuite introduites. Le chapitre montre que l'utilisation d'un style particulier de logique asynchrone augmente la résistance des circuits aux attaques par canaux cachés. La troisième partie de ce chapitre introduit les avantages des circuits programmables notamment des FPGAs vis-à-vis de la sécurité. De même, il présente leurs faiblesses et leurs vulnérabilités à certaines attaques.

Dans le chapitre trois, les spécifications du FPGA asynchrone sont définies. Les critères « logiques » à respecter lors de la conception sont présentés et discutés afin de garantir flexibilité (implémentation de plusieurs protocoles de communication et codages de données) et résistance aux attaques par canaux cachés (attaques par analyse de courant, attaques temporelles, …).

Le chapitre quatre traite les aspects électriques et logiques du FPGA. Les architectures des principaux sous-blocs constituant le FPGA y sont présentées. Il montre également les contre-mesures électriques qui doivent être prise en compte pour assurer le niveau de sécurité souhaité. Ceci concerne surtout les blocs programmables du FPGA car la sécurisation du réseau d'interconnexion a été étudiée dans le cadre d'une autre thèse (S. Chaudhuri, 2009).

Le dernier chapitre est consacré à la validation de l'architecture proposée et au test du prototype fabriqué. Deux campagnes de validation au niveau électrique sont présentées. Elles montrent les résultats obtenus sur un sous-bloc d'un circuit DES (Data Encryption Standard) qui est d'abord évalué avec un protocole 4 phases associé à un codage 1 parmi 2 et ensuite évalué avec un protocole 2 phases associé à un codage LEDR, l'objectif étant de valider l'approche sécuritaire annoncée dans les chapitres précédents. Le prototype a été fabriqué en technologie CMOS 65 nm de ST Microelectronics. Le test du prototype a permis de découvrir une erreur de conception qui empêchait son fonctionnement normal. Une solution a été proposée pour corriger cette erreur et est également présentée dans ce chapitre. Elle fera potentiellement l'objet d'un futur prototype.

Enfin, il est à noter que la programmation du FPGA est assurée par un logiciel spécialement développé pour cette étude et dont l'algorithme est détaillé dans ce chapitre. Il est en effet nécessaire de le programmer en intégrant les contraintes de sécurité pour conserver les bénéfices des contre-mesures matérielles implémentées dans le FPGA.

Chapitre 2.
ETAT DE L'ART

2.1 LA CRYPTOGRAPHIE

La *cryptologie* est une science mathématique qui comporte deux branches : la cryptographie et la cryptanalyse. Traditionnellement, elle est l'étude des méthodes permettant de transmettre des données de manière confidentielle. Afin de protéger un message, il faut lui appliquer une transformation qui le rend incompréhensible ; c'est ce qui est appelé le *chiffrement* qui, à partir d'un texte en clair, donne un texte chiffré ou *cryptogramme*. Inversement, le déchiffrement est l'action qui permet de reconstruire le texte en clair à partir du texte chiffré. Dans la cryptographie moderne, les transformations en question sont des fonctions mathématiques, appelées algorithmes cryptographiques (cryptosystèmes), qui dépendent d'un paramètre appelé clé. La sécurité de d'un cryptosystème dépend de deux paramètres : la sûreté de l'algorithme et la longueur de la clé utilisée.

Si le but traditionnel de la cryptographie est d'élaborer des méthodes permettant de transmettre des données de manière confidentielle, la cryptographie moderne s'attaque en fait plus généralement aux problèmes de sécurité des communications. Le but est d'offrir un certain nombre de services de sécurité comme la confidentialité, l'intégrité, l'authentification des données transmises et l'authentification d'un tiers. Pour cela, un certain nombre de mécanismes basés sur des algorithmes cryptographiques sont utilisés.

Le concept de clé cryptographique a été créé pour augmenter la robustesse des algorithmes cryptographiques. Elle est généralement mixée avec les données d'entrées des algorithmes cryptographiques par l'opérateur logique « ou exclusif ». Plus la longueur de la clé est importante, plus il est difficile de la briser par une attaque exhaustive. Pour une clé de 56 bits, il faut en moyenne $3{,}6 \times 10^{16}$ tentatives. En faisant l'hypothèse qu'un superordinateur peut essayer 1 million de clés par secondes, il faudra environ 1000 ans pour trouver la bonne clé. De nos jours, les cryptosystèmes utilisent au minimum des clés de 128 bits.

La notion de clé cryptographique permet de définir deux schémas de base des cryptosystèmes ou algorithmes cryptographiques. Les algorithmes à clé secrète ou ***cryptographie symétrique*** et les algorithmes à clés publiques ou ***cryptographie asymétrique.***

Ce chapitre se consacre à la présentation des deux types de cryptographies cités ci-dessus ainsi qu'à l'introduction des attaques par canaux cachés. Dans cette optique, la première partie introduira quelques notions de base de la cryptographie sous ses deux formes « symétriques » et « asymétriques », avant de présenter dans la seconde partie un état de l'art bref sur les attaques matérielles existantes.

2.1.1 La cryptographie symetrique

La cryptographie est utilisée depuis l'antiquité pour cacher le sens du contenu d'une communication entre deux interlocuteurs. Depuis l'époque romaine, une technique simple, attribuée à César, consiste à appliquer à chaque symbole du message clair une rotation de valeur fixe à l'intérieur de l'alphabet utilisé, jusque vers la fin des années 1970, les algorithmes de chiffrement reposaient tous sur le même principe que l'émetteur et le destinataire d'un message confidentiel devaient chacun utiliser une même valeur secrète de clé k. Cette clé k, sert comme paramètre pour la fonction de cryptage $C = E_k(M)$, qui à partir d'un texte clair M, en génère une version inintelligible. Le texte chiffré C est ensuite envoyé au destinataire et déchiffré en appliquant la transformation inverse E^{-1}_k pour retrouver le message initial M. Ce principe est appelé cryptographie symétrique - ou aussi cryptographie à clé secrète - du fait que la clé utilisée par l'émetteur pour chiffrer est la même utilisée par le récepteur pour déchiffrer.

Pendant longtemps la sécurité des applications cryptographiques, reposait sur la confidentialité des algorithmes utilisés. Ce n'est qu'en 1883 qu'August Kerchoffs publie son essai « La Cryptographie Militaire » dans lequel il propose que la sécurité d'un système cryptographique ne doit pas reposer sur le caractère du système de codage, mais sur une petite quantité secrète d'information: la clé.

Plus tard, le premier standard public de chiffrement est apparu en 1976 le DES[2], et a participé à l'explosion de l'usage de la cryptographie dans le monde. Vingt cinq ans plus tard, ce standard est devenu obsolète à cause de taille de clé très faible par rapport à l'évolution des moyens de calcul, et est remplacé par le nouveau standard AES[3] en 2001.

EXEMPLE D'ALGORITHME DE CHIFFREMENT SYMETRIQUE : LE DES

Après avoir subie plusieurs modifications, l'algorithme Lucifer de IBM, est devenu officiellement le DES. Il a été ensuite approuvé en novembre 1976 et publié comme standard en 1977 ("Data Encryption Standard", 1977). En 2004 il a été recommandé de l'utiliser sous la forme de Triple-DES qui utilise une clé (K_1, K_2) de 112 bits au lieu d'une clé de 56 bits.

Le DES permet de chiffrer des blocs de 64 bits à l'aide d'une clé de 56 bits. La structure de cette fonction est essentiellement basée sur le réseau de FEISTEL (B. Schneier, 2001). D'une manière générale, le fonctionnement d'un DES, peut être traduit en trois étapes (cf. Figure 1) **:**

1. Permutation initiale et fixe d'un bloc par la fonction IP.

2. Le résultat subie ensuite 16 itérations appelées tour ou ronde. Ces itérations dépendent à chaque tour d'une autre clé partielle de 48 bits ; elle est calculée à partir de la clé initiale de l'utilisateur grâce à un réseau de tables de substitution et d'opérateurs XOR. Le tour numéro i transforme son entrée (L_{i-1}, R_{i-1}) en la sortie (L_i, R_i) tel que :

$$\boldsymbol{L_i = R_{i-1}}$$

$$\boldsymbol{R_i = L_{i-1} \oplus f_{k_i}(R_{i-1})}$$

Lors de chaque tour, le bloc de 64 bits est découpé en deux blocs de 32 bits, et ces blocs sont échangés l'un avec l'autre selon un schéma de Feistel. Le bloc de 32 bits ayant le poids le plus fort subira une transformation $\boldsymbol{f_{k_i}(R_{i-1})}$.

La fonction de transformation est détaillée à la Figure 2 Tout d'abord une permutation expansive PE commence par la duplication de 16 bits des 32 bits de $\boldsymbol{R_{i-1}}$ pour fournir 48 bits qui subissent la fonction XOR avec la clé de tour correspondante $\boldsymbol{K_i}$. La fonction XOR peut être interprétée comme étant 8 XOR à 12 entrées chacune. Les sorties de ces huit XOR sont utilisées dans des tables de substitution à 64 entrées appelées S-Box. Les S-Box sont la partie non-linéaire de l'algorithme qui contribue à la confusion[4] nécessaire en rendant

[2] Data Encryption Standard
[3] Advanced Encryption Standard
[4] *La diffusion et la confusion* ont été identifiées par Claude SHANNON **Invalid source specified.** comme des propriétés indispensables de toute fonction de chiffrement.

l'information inintelligible. Les valeurs présentes dans la S-Box ont été choisies d'une manière à résister aux attaques, par divers moyens comme l'utilisation de fonctions courbes. Il a été prouvé que les tables avaient été conçues de manière à résister à la cryptanalyse différentielle (E . Biham and A. Shamir, 1992).

Les sorties sur 4 bits des huit S-Box sont ensuite regroupées pour former 32 bits qui seront ensuite mélangés par la permutation P.

3. Le dernier résultat de la dernière ronde est transformé par la fonction inverse de la permutation initiale FP.

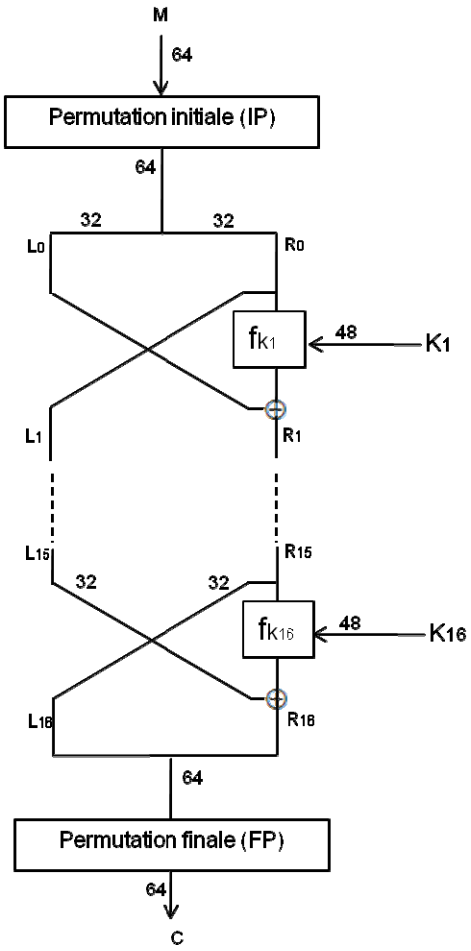


FIGURE 1: LA FONCTION DE TOUR DU DES

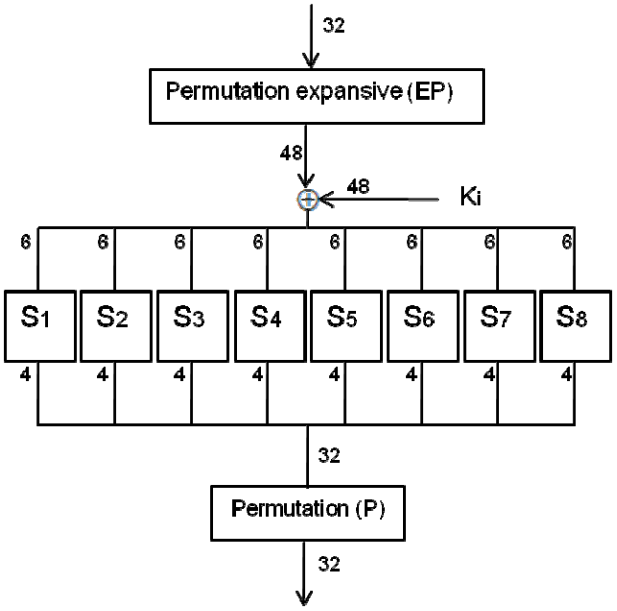


FIGURE 2: LA FONCTION DE TOUR DU DES

2.1.2 LA CRYPTOGRAPHIE ASYMETRIQUE

Pour que deux interlocuteurs, utilisant un algorithme de chiffrement symétrique, puissent communiquer de manière confidentielle, il leur est nécessaire de convenir au préalable d'une valeur secrète de clé. D'où l'inconvénient majeur de la cryptographie symétrique. En effet, cette phase d'échange de clé semble problématique puisqu'elle nécessite de pouvoir disposer d'un moyen sécurisé de transmission d'une telle information.

L'usage de la cryptographie a été révolutionné en 1976, par l'invention de la cryptographie asymétrique – encore appelée cryptographie à clé publique - par W. Diffie and M. Hellman (W. Diffie, M. Hellamn, novembre 1976). Cette cryptographie fait appelle à des permutations à sens unique à trappe. Une personne A, qui veut recevoir une correspondance confidentielle, va générer une clé K^B. Cette clé est en réalité composée de deux parties publique K_p^B et privée K_s^B. La partie publique K_p^B est transmise à une personne B pour qu'elle puisse chiffrer des données et les envoyer à la personne A. Tandis que la partie privée K_s^B est maintenue secrète par la personne A. Elle lui sert de clé de chiffrement pour inverser la fonction à sens unique et ainsi déchiffrer les messages qu'il a reçus de la part de la personne B.

Ce système de communication de clé résout le problème évoqué pour l'échange de clé dans le cas de la cryptographie symétrique. Il n'est plus nécessaire de disposer de canal sécurisé pour échanger la clé de chiffrement. En effet, la partie publique de la clé de la personne A n'est dédié à aucune personne. N'importe qui peut l'utiliser pour chiffrer des données et l'envoyer à la personne A. Cela peut créer un problème d'authenticité : une personne mal in-

tentionnée peut publier une clé publique au nom de la personne A, alors que c'est lui qui possède la contre-partie privée. Il sera ainsi capable de déchiffrer toute donnée confidentielle que la personne B aura cru envoyer à la personne A. Pour remédier à ce problème, les utilisateurs de ce protocole utilisent des certificats qui attestent de l'identité du possesseur de la clé publique. En 1984 A. Shamir (A. Shamir, 1985) a proposé le concept de chiffrement basé sur l'identité IBE[5]. Et c'est en 1991 qu'U. Maurer et Y. Yacobi ont publié un exemple concret d'IBE (U. Maurer, Y. Yacobi, 1996) (U. Maurer, Y. Yacobi) .

Généralement, les cryptosystèmes asymétriques sont utilisés pour échanger d'une manière sécurisée des données de petites tailles comme une clé de chiffrement symétrique. Tandis que les cryptosystèmes symétriques sont utilisés pour chiffrer/déchiffrer des données de grandes tailles. Cette répartition d'utilisation des deux familles est due à la rapidité des cryptosystèmes symétriques par rapport à leurs concurrents. Le premier cryptosystème RSA6 à clé publique a été proposé par R. Rivest, A. Shamir et L. Adleman en 1978 (R. Rivest, A. Shamir, L. Adleman, 1978). Sa sécurité repose sur la complexité de la factorisation d'entiers qui n'a toujours pas été résolue. Depuis, plusieurs cryptosystèmes à clé publique ont été publiés, certains reposant sur d'autres problèmes difficiles comme par exemple le problème du logarithme discret dans le groupe multiplicatif défini modulo un premier (T. ElGamal, 1985), (T. ElGamal, 1985) et (National Institute Of Standards And Technology, 2000), ou dans les groupes définis par des courbes elliptiques (Victor Miller, 1986) et (N. Koblitz, 1987).

2.1.3 Cryptanalyse & attaques

La cryptanalyse est le domaine de la cryptologie consacré à l'étude des méthodes ou techniques permettant de restituer un message chiffré en clair sans pour autant connaître la clé utilisée.

L'objectif de cette science est de pouvoir mettre en évidence les faiblesses d'un cryptosystème. Il s'agit de tentatives de déchiffrement ou d'attaques cryptanalytiques.

Il existe cinq types d'attaques génériques, avec par ordre croissant d'efficacité :

1. L'attaque avec un texte chiffré : le cryptanalyste dispose du texte chiffré de plusieurs messages obtenus avec le même algorithme et la même clé.

2. L'attaque avec un texte clair connu : le cryptanalyste dispose des textes chiffrés et des textes en clairs correspondants.

[5] Identity Based Encryption
[6] L'algorithme porte le nom de ses inventeurs : Rivest Shamir Adleman

3. L'attaque avec un texte en clair choisi : le cryptanalyste a accès aux textes chiffrés et aux textes en clair, et il peut choisir les textes en clair à chiffrer.

4. L'attaque avec texte adaptatif : le cryptanalyste peut choisir les textes en clair, et il peut également adapter ses choix en fonction des textes chiffrés obtenus.

5. L'attaque avec texte chiffré choisi : le cryptanalyste peut choisir différents textes chiffrés à déchiffrer.

Ces attaques reposent sur le principe de Kerckhoff (D. Stinson, 1996) stipulant que la connaissance complète et détaillée du cryptosystème n'est pas un secret pour le cryptanalyste. Ainsi, la sécurité et la robustesse de tout algorithme cryptographique ne repose pas sur la non connaissance de cet algorithme, mais essentiellement sur les propriétés mathématiques de l'algorithme et sur la longueur des clés utilisées.

Cependant, la conception des systèmes cryptographiques actuels nécessite la collaboration des compétences dans différents domaines (informatique, mathématique, microélectronique etc.). La Figure 3 représente les classes des différentes méthodes cryptanalytiques et des différentes attaques en fonction des compétences qu'elles mettent en œuvre. Chacun des cinq modèles d'attaques génériques définis ci-dessus peut être utilisé pour chacune de ces méthodes.

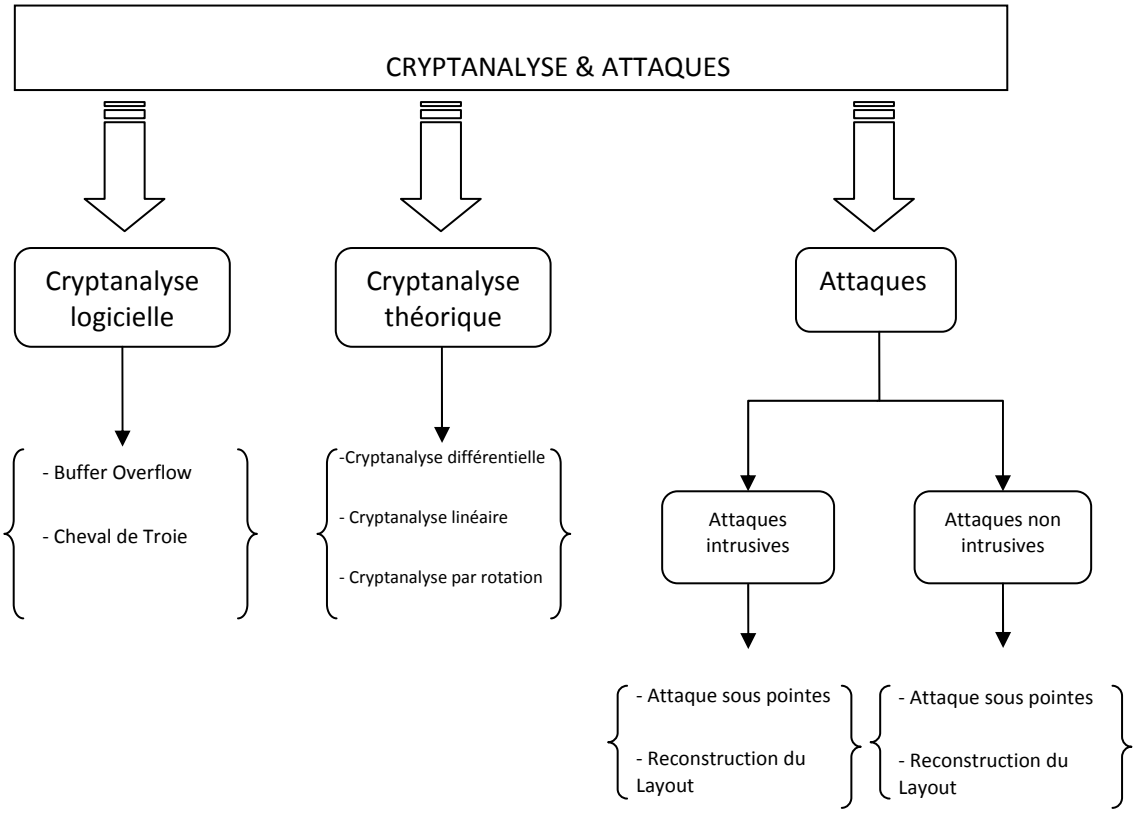


FIGURE 3: LES DIFFERENTES METHODES DE CRYPTANALYSES ET D'ATTAQUES

2.1.3.1 LA CRYPTANALYSE DITE LOGICIELLE

La cryptanalyse logicielle regroupe toutes les techniques qui permettent d'accéder aux informations confidentielles d'un cryptosystème en utilisant les failles des systèmes d'exploitation ou des applications du système. Des attaques bien connues comme le cheval de Troie (A. Ross, 2001.) ou le débordement de pile (A. One, 1996) ont montré toute leur efficacité aussi bien sur les systèmes d'exploitation des ordinateurs personnels que sur des systèmes d'exploitation moins complexes comme ceux des systèmes embarqués ou des cartes à puce.

2.1.3.2 LA CRYPTANALYSE DITE THEORIQUE

Les attaques dites théoriques sont basées sur des analyses utilisant des théories mathématiques pour briser des algorithmes cryptographiques. Elles exploitent les faiblesses de conception théorique des cryptosystèmes pour générer des attaques. Ces types d'attaques permettent aux cryptanalystes d'évaluer théoriquement la force ou la sûreté d'un algorithme cryptographique.

Plusieurs types de cryptanalyse théorique existent dans la littérature, parmi les plus connues : la cryptanalyse linéaire (M. Matsui, May 23-27, 1993) et la cryptanalyse différentielle(B. Schneier, 2001).

2.1.3.3 LES ATTAQUES

Cette branche regroupe l'ensemble des techniques cryptanalytiques exploitant les vulnérabilités matérielles des circuits intégrés dédiés à des applications sécurisées. Deux catégories d'attaques matérielles peuvent être distinguées : les attaques intrusives et les attaques non intrusives.

2.1.3.3.1 LES ATTAQUES INTRUSIVES

Les attaques intrusives sont directement réalisées sur le silicium du composant. La puce, mise à nue, est attaquée par des solvants chimiques pour retirer les différentes couches de passivations, d'isolations (etc.) utilisées dans les procédés de fabrication des circuits intégrés. Ces accès physiques sur la puce, permettent soit de découvrir le dessin des masques (*layout*) de la puce ou de réaliser des analyses sous pointes. Ces attaques sont destructives.

L'attaque par reconstruction du dessin des masques par exemple a pour objectif l'identification et l'analyse des zones sensibles d'une puce par reconstitution du *layout*. Le *layout* de la puce est reconstruit en réalisant, en sens inverse, les différentes étapes des procédés de fabrication des circuits intégrés (A. Ross, 2001.) Ainsi les différentes couches du circuit sont enlevées les unes après les autres par des solutions chimiques pour refaire le *layout*. Ce type d'attaque permet même de lire les valeurs d'une mémoire (D. Samyde, S. Skorobogatov, R. Anderson and J-J. Quisquater, December 2002). Elle permet également d'analyser les mécanismes de sécurité implantés dans une puce afin d'orienter et définir des schémas d'attaques (logicielles ou matérielles).

De manière générale, les attaques intrusives nécessitent des moyens considérables et coûteux notamment avec l'utilisation d'un microscope optique (FIB : Focused Ion Beam), du matériel sous pointes ou même d'une salle blanche. Comme ces attaques peuvent être destructives, elles nécessitent également souvent au cryptanalyste de disposer de plusieurs composants.

Les attaques intrusives permettent de collecter des informations sur les systèmes de protection matérielle embarqués dans les puces. Selon les fiabilités ou défaillances observées dans ces mécanismes de protection, les cryptanalystes mettent alors en place des méthodes d'attaque moins coûteuses, plus simples et surtout non destructives pour briser les composants possédant les mêmes systèmes de protection.

2.1.3.3.2 LES ATTAQUES NON INTRUSIVES

La robustesse des fonctions cryptographiques est un facteur très important pour assurer la sécurité des systèmes qui les utilisent. Elles doivent démontrer un niveau de sécurité théorique ne laissant place à aucune attaque réalisable en un temps, effort ou coût raisonnable qu'un adversaire menant ses analyses peut accepter.

Cependant, il est apparu récemment que les précautions prises quant à la fonction mathématiques utilisée et à son emploi, quoique toujours nécessaires, ne suffisent pas à garantir concrètement la sécurité d'un système. La mise en œuvre de la cryptographie requiert une implémentation physique qui produira nécessairement des grandeurs observables d'une autre nature que les entrées et les sorties de la fonction mathématique.

Ces grandeurs physiques sont aussi connues sous le nom de 'canaux cachés' ou '*canaux auxiliaires*'. Leur exploitation s'appelle '*attaques par canaux auxiliaires'* ou *Side Channel Attack* 'SCA'. La notion de canal est définie comme un endroit par lequel transite de l'information, et il est caché quand son utilisation est détournée de son fonctionnement initial. Ces attaques concernent essentiellement les crypto-systèmes embarqués – comme les cartes à puce - parce qu'elles nécessitent une interaction directe avec le dispositif cryptographique (ou au moins son observation).

La première analyse de canaux auxiliaires publiée, exploitait le temps d'exécution d'une commande (P. Kocher, 1996), mais d'autres grandeurs ont été analysées par la suite : comme la consommation de courant en 1998 (P. Kocher, J. Jaffe, B. Jun, 1998), ou le rayonnement électromagnétique en 2001 (K. Gandolfi, C. Mourtel, F. Olivier, May 2001). D'autres types de fuite d'information sont toujours envisageables : en 2007 à CHES'07 un participant a proposé d'exploiter la variation de température externe d'un composant électronique en fonction de son activité.

En fonction du signal compromettant utilisé, les attaques peuvent être classées sous différentes catégories : attaques temporelles, attaques électromagnétiques et attaques en puissance.

2.1.3.3.2.1 LES ATTAQUES TEMPORELLES

Les attaques temporelles exploitent les dépendances temporelles entre les données manipulées et le temps nécessaire au calcul des données. Le principe de l'attaque est de déterminer toute corrélation entre le temps d'exécution d'une fonction (en terme de cycles d'horloge) et les données traitées. Certains choix d'implantation d'algorithme cryptographique sont particulièrement vulnérables à ce type d'attaque comme c'est le cas de l'implantation par la méthode binaire du calcul de l'exponentielle modulaire utilisée dans le RSA.

Paul Kocher (P. Kocher, 1996) a donné en 1996 un premier exemple d'analyse de canal auxiliaire en proposant d'exploiter des variations de la durée de certaines opérations, comme

par exemple une multiplication modulaire, en fonction des valeurs de ses opérandes. Il montre qu'il est alors possible de retrouver les clés dans différents cryptosystèmes à clé publique basés sur l'exponentiation modulaire.

Plusieurs modèles d'attaque sur le DES et sur l'AES ont été proposés respectivement dans (A. Hevia and M. Kiwi, 1999) et (F. Koeune and J-J. Quisquater). Le premier définit une méthode d'attaque qui permet de déterminer le poids de Hamming des bits de la clé du DES et le second présente une attaque sur la fonction Mixcolumns de l'AES et plus particulièrement sur la sous-fonction Xtime (NIST, November 2001). D'autres attaques ont aussi été publiées dans (H. Handschuh and H.Heys, 2004) et (J. F. Dhem, F. Kouene, P-A. Leroux, P. Mestre, J- J. Quisquetr and J- L. Willems, 1998)

Les attaques par analyse du temps d'exécution n'ont pas donné lieu par la suite à beaucoup d'autres publications car il est apparu qu'il était souvent assez facile d'assurer qu'une implémentation logicielle sur carte à puce s'exécute en temps constant.

2.1.3.3.2.2 LES ATTAQUES ELECTROMAGNETIQUES

Depuis leurs publications il y a quelques années par la NSA[7] (NSA, 2000), les rapports scientifiques classés secrets défense sur la cryptanalyse matérielle par effets électromagnétiques ont contribué au développement considérable des attaques par analyses électromagnétiques.

Ces attaques ont été introduites dans (J-J. Quisquater and D. Samyde, 2000) ; elles sont basées sur l'étude du rayonnement électromagnétique au voisinage proche du circuit. Les caractéristiques du champ électromagnétique (en termes de fréquence et d'amplitude) sont relatives à l'activité électrique et à la nature des données manipulées dans le circuit. Par conséquent, le principe des attaques électromagnétiques est d'exploiter cette dépendance afin de déterminer toute corrélation entres les données traitées et les émissions électromagnétiques.

Les attaques électromagnétiques peuvent se décliner en attaques simples SEMA[8] et différentielles DEMA[9]. Elles utilisent les mêmes algorithmes d'analyse que ceux utilisés par des attaques en puissances (SPA[10] et DPA[11]), sauf que les analyses sont faites sur des courbes d'émissions électromagnétiques. Rao et Rohatgi ont montré dans (J. R. Rao and P. Rohatgi, 2001) que dans bien des cas, la SEMA s'avérait bien plus efficace que la SPA pour la détermination des valeurs traitées dans les instructions de test, de décalage ou de branchement sur condition. Par contre la DEMA reste aussi redoutable que la DPA. Toutefois, certaines implanta-

[7] National Security Agency
[8] Simple Electromagnetic Attack
[9] Differential Electromagnetic Attack
[10] Single Power Analysis
[11] Differential Power Analysis

tions sécurisées contre la SPA ou la DPA ne le sont pas forcément contre la SEMA ou la DEMA et vice versa.

Contrairement aux attaques en puissance, ces attaques peuvent avoir une résolution de l'ordre de quelques portes en utilisant des sondes adaptées. Un exemple de mise en place d'une telle attaque peut être trouvé dans (K. Gandolfi, C. Mourtel, F. Olivier, May 2001).

La mise en œuvre des attaques électromagnétiques commence par des mesures du champ électromagnétique (CEM) du circuit. Pour cela, les modèles développés dans le domaine de la CEM sont généralement utilisés [Pany04]. Deux types de mesures peuvent être réalisés : des mesures d'émissions conduites ou des mesures d'émissions rayonnées.

Les mesures d'émissions conduites sont effectuées sur les signaux d'alimentation des composants. En terme de cryptanalyse, ces analyses sont quasi identiques aux analyses réalisées sur les signaux d'alimentation. Ici, le signal de courant est remplacé par son spectre électromagnétique.

Les mesures rayonnées permettent d'obtenir plus de précision et sont plus efficaces pour des analyses de corrélations. Elles permettent d'établir une cartographie de l'activité électromagnétique des circuits aidant ainsi à identifier les zones les plus sensibles du circuit sur lesquelles seront focalisées les attaques. Pour cela, des capteurs ou sondes électromagnétiques munis d'inductance sont utilisés et selon leurs caractéristiques (taille de la sonde, bande passante, sensibilité), ils peuvent mesurer l'activité électromagnétique d'une dizaine de portes de la puce (K. Gandolfi, C. Mourtel, F. Olivier, May 2001).

Cette précision dans les analyses rend ce type d'attaque particulièrement redoutable sur des instructions de branchement sur conditions, de saut, de test aux accès mémoires ou sur des bus de données (d'adresse) en déterminant leurs poids de Hamming.

2.1.3.3.2.3 LES ATTAQUES EN PUISSANCE OU ATTAQUES PAR ANALYSE DE COURANT

Les attaques en puissances, comme leur nom l'indique, exploitent l'activité électrique des composants en mesurant leur profil de courant. Elles sont focalisées sur la recherche de corrélation entre les données manipulées et les caractéristiques des signaux d'alimentation (Vdd ou Gnd) du circuit (variations en amplitudes, pics de consommation, décalage temporel, puissance etc.). Une fois des corrélations observées, des méthodes d'analyse sont mises en œuvre afin d'exploiter ces dépendances pour trouver les informations secrètes. Les bases des attaques en puissance reposent sur l'hypothèse suivante : En technologie CMOS la puissance consommée par un élément logique (porte) est différente selon qu'elle charge ou décharge sa capacité de sortie.

$$P_{charge} \neq P_{décharche} \rightarrow I_{charge} \neq I_{décharche}$$

Ces différences qui sont directement liées à la nature des données manipulées sont exploitées pour établir des corrélations entre les courbes de courant et les données. Plusieurs techniques et méthodes ont été développées en vue d'exploiter cette dépendance et ont été appliquées avec succès sur la majeure partie des cryptosystèmes. Ces méthodes permettent de retrouver directement les clés cryptographiques. Les attaques par analyse de courant sont parmi les attaques matérielles les plus efficaces, les plus faciles à mettre en œuvre et les moins coûteuses. Elles sont à la portée de tout laboratoire possédant un oscilloscope, une chaîne d'acquisition du courant et un ordinateur de bureau.

Plusieurs modèles d'analyse de courant existent dans la littérature, dont : l'analyse simple de courant SPA[12] , l'analyse différentielle de courant DPA[13], l'analyse de courant par corrélation CPA[14], etc.

Les analyses simples de courant SPA : Elles consistent à mesurer et à observer la consommation de courant d'un circuit cryptographique. Cette consommation est proportionnelle à la tension observée aux bornes d'une résistance connectée en série à l'alimentation du circuit. Elle résulte pour l'essentiel de la somme des contributions des différents sous-blocs du circuit, reflétant son activité interne qui dépend des données manipulées.

Les attaques simples de courant, exploitent ces différences de consommation visibles sur une ou quelques traces de courant, chacune pouvant éventuellement être produite par moyenne de plusieurs exécutions avec les mêmes données d'entrées afin d'atténuer le bruit de mesure. C'est ensuite la comparaison de ces courbes qui va permettre de trouver un secret caché dans le circuit.

Une autre attaque, dérivée de la SPA et appelée attaques par dictionnaire TA[15], permet aussi d'exploiter la dépendance de la consommation en fonction des données manipulées (S. Chari, J. Rao, P. Rohatgi, 2003), (C. Rechberger and E. Oswald, 2004), (C. Archambeau, E. Peeters, F-X. Standaert and J-J Quisquater, 2006) et (B. Gierlichs, K. Lemrke-Rust and C. Paar, 2006).

Les analyses statistiques du courant DPA, CPA... : Ce type d'analyses, nécessite un grand nombre de traces de courant, obtenues en faisant varier le message en entrée de l'algorithme. Un traitement statistique de ces traces permettra ensuite de réaliser un test d'hypothèse sur une partie portion de la clé, appelé sous-clé dans le cas des algorithmes de chiffrement par bloc. Le test d'hypothèse exploite la consommation de courant produite par la manipulation d'une donnée intermédiaire de l'algorithme qui ne dépend que du message d'entrée (ou du chiffré de sortie) et de la valeur de la sous-clé.

[12] Simple Power Analysis
[13] Differential Power Analysis
[14] Correlation Power Analysis
[15] Template Attack

Paul Kocher, Joshua Jaffe et Benjamin Jun ont introduit dès 1998, la première méthode d'*analyse différentielle de courant* DPA (P. Kocher, J. Jaffe, B. Jun, 1999) et (P. Kocher, J. Jaffe, B. Jun, 1998). Cette méthode sélectionne un bit – appelé *bit de sélection* ou *bit cible* – intermédiaire de l'algorithme qui ne dépend que du message d'entrée et d'une sous-clé. Des traces de courant sont ensuite faites pour chaque supposition sur la valeur de la sous-clé, et sont classées sous deux catégories : la première contient les traces pour lesquelles la valeur du bit de sélection vaut 0, et l'autre contient les traces pour lesquelles il vaut 1. Une fois que le classement est fait, la courbe de courant moyenne de chacune des deux catégories est calculée, et sont soustraites l'une à l'autre. Il en résulte une courbe DPA associé à chacune des suppositions sur la sous-clé.

Si la valeur supposée de la sous-clé est correcte, alors le bit de sélection reflète exactement sa valeur au moment de l'exécution et la courbe de DPA montre une différence de courant significative à chaque instant où ce bit a été manipulé. Cela se traduit par un pic à chacun de ces instants. En revanche, si la supposition est incorrecte, les courbes de chaque paquet sont indépendantes de ce qui s'est réellement passé dans le circuit pour ce bit ; et les courbes DPA qui en résultent sont présumées être plates à un bruit près.

L'avantage de la DPA est qu'elle ne nécessite pas de faire une hypothèse forte sur le modèle de consommation en fonction des données. Il suffit tout simplement que la moyenne de la consommation de courant lorsqu'un mot de donnée est manipulé soit différente selon que l'un des bits de ce mot vaille 0 ou 1. Cependant, l'inconvénient de la DPA est d'être victime des dites : *pics fantômes.* Ces pics – dont la l'amplitude peut rivaliser avec les vrais pics – sont présents sur les courbes DPA qui correspondent à des suppositions incorrectes de la sous-clé. Cela pose un vrai problème d'identification de la vraie valeur de la sous-clé. Une analyse détaillée de ce sujet a été fait pat Cécile Canovas et Jessy Clédière dans (C. Canovas and J. Clédière, 2005).

Une alternative à la DPA, a été proposée par Eric Brier, Francis Olivier et Christophe Clavier dans (E. Brier, C. Clavier and F.Olivier, 2004) et (E. Brier, C. Clavier and F.Olivier, 2003). Cette méthode appelée «*analyse de courant par corrélation CPA*, vise à éviter les hypothèses trop simplificatrices et à prendre en considération toute l'information dont dispose l'attaquant qui se révèle pertinente pour la fonction de consommation à l'instant de formation du pic.

2.2 LA LOGIQUE ASYNCHRONE POUR LA SECURITE

Les circuits asynchrones ont des caractéristiques qui diffèrent significativement des celles des circuits synchrones. Ces caractéristiques ont permis – depuis l'exploitation de la logique asynchrone – de concevoir des circuits possédant des caractéristiques de performance très intéressantes en termes de consommation, émission électromagnétique, vitesse etc.

L'étude des circuits asynchrones a commencé au début des années cinquante. En 1956, Muller et Bartky de l'université d'Illinois ont travaillé sur la théorie des circuits asynchrones. Huffman était le premier à concevoir des machines à états asynchrones en 1968. Par la suite, Muller a proposé d'associer un signal de validité aux données en introduisant un protocole de communication quatre-phases. En 1966, le *« Macromodule Project »* à l'université de Washington (W.A. Clark, 1967) a montré qu'il est possible de concevoir des machines spécialisées complexes par simple composition de blocs fonctionnels asynchrones. Quelques années plus tard, Seitz a introduit un formalisme proche des réseaux de Pétri pour concevoir des circuits asynchrones, ce qui a aboutit à la conception du premier calculateur data-flow DDM1 (A. L. Davis, 1978). Enfin, Sutherland a largement contribué à l'intérêt croissant porté par les académiques et les industriels à la conception de circuits asynchrones (Ivan E. Sutherland, 1989). Depuis, les travaux de recherche n'ont cessé de s'intensifier (S. Hauck, 1995).

Aujourd'hui, malgré une prépondérance des circuits synchrones et des outils de conception associés qui ne cessent de progresser, de plus en plus d'acteurs s'intéressent au style asynchrone en raison des difficultés croissantes rencontrées pour concevoir les circuits synchrones. Cela se traduit par la création de plusieurs entreprises qui fabriquent des circuits asynchrones dont les plus connues sont : Theseus Logic, TIEMPO (TIEMPO), ACHRONIX (Achronix), Handshake Solution (handshake solutions), Silistix (Silistix), et Fulcrum (Fulcrum)....

La suite de ce chapitre est consacrée à la présentation des concepts fondamentaux qui régissent le fonctionnement des circuits asynchrones, et à leur évaluation pour la conception des systèmes sécurisés. Les propriétés spécifiques des circuits asynchrones les prédisposent à mieux résister aux attaques que leurs équivalents synchrones. L'absence de signal d'horloge global leur octroi une faible émission électromagnétique et de faibles pics de courant. L'utilisation d'un codage des données et d'un protocole de communication de type poignée de main (ou *Handshake*) permet une consommation faible, répartie et indépendante des données. L'étude a pour objectif de présenter les concepts de base du fonctionnement des circuits asynchrones et d'évaluer leurs propriétés intrinsèques qui leur permettent de mieux résister aux attaques en puissances.

La première partie, introduit les notions de bases de la logique asynchrone en se focalisant sur une classe de circuits asynchrones dit « Quasi-Insensible aux Délais » qui au sein des différentes classes existantes, semble proposer les meilleures propriétés pour la conception des circuits sécurisés. Ensuite, la deuxième partie donne un rapide aperçu sur les avantages de l'utilisation de la logique asynchrone vis-à-vis de la sécurité des circuits numériques.

2.2.1 LA LOGIQUE ASYNCHRONE : NOTIONS DE BASE

Le mot asynchrone signifie qu'il n'existe pas de relation temporelle *a priori* entre les signaux, cependant le comportement des circuits asynchrones est parfaitement déterminé. Ces signaux implémentent contrôle et données.

La conception de circuits synchrones repose fondamentalement sur la présence d'un signal unique qui gère les communications dans tout le système de manière globale : l'horloge. Ce signal détermine quand les unités du chemin de données doivent fonctionner. La fréquence de cette horloge est calculée en fonction du délai le plus long ou chemin critique du circuit. De ce fait, un concepteur ne sait pas quand une opération est terminée, mais par contre, il est sûr qu'à la fin du cycle, elle est réellement terminée. Il y a donc une synchronisation implicite de toutes les unités à la fin de chaque cycle.

Par opposition, les circuits asynchrones, n'utilisent pas de signal d'horloge. La gestion des communications se fait de manière locale par une synchronisation entre blocs fonctionnels. Un des concepts les plus importants dans les circuits asynchrones est le contrôle distribué. Cela signifie que chaque unité se synchronise seulement avec les blocs dont la synchronisation a une signification fonctionnelle. De plus, elle se fait uniquement quand elle doit avoir lieu, indépendamment du reste du système. C'est pour cela que chaque bloc d'un circuit asynchrone possède des signaux explicites de synchronisation qui exécutent des protocoles de type requête/acquittement ou « poignée de main » avec les blocs voisins (cf. Figure 4)

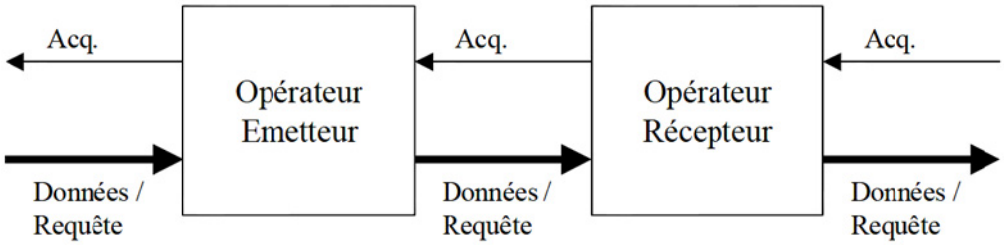


FIGURE 4: COMMUNICATION POIGNEE DE MAIN OU HANDSHAKE

Une traduction possible du dialogue qui se fait entre les deux unités serait celle-ci :

1. Le récepteur : envoie un signal à l'émetteur pour lui dire qu'il est prêt à recevoir des nouvelles données.
2. L'émetteur : reçoit le signal du récepteur et lui envoie les données.
3. Le récepteur : effectue le calcul, et envoie un signal à l'émetteur pour lui dire qu'il a terminé le calcul

4. L'émetteur : enlève les données utilisées à la réception de ce dernier signal.

Par ce mécanisme, le bon fonctionnement du système devient indépendant du temps de calcul de chaque opération dans les blocs. Plus précisément, le circuit ne dépendra pas de la distribution des retards dans les portes et les connexions. Cependant, la synchronisation dépend de certaines hypothèses simplificatrices d'un point de vue fonctionnel. Ces hypothèses traduisent le mode de fonctionnement des circuits asynchrones.

Les modes de fonctionnement gouvernent l'interaction entre un bloc asynchrone et son environnement. Certaines hypothèses sur ces modes peuvent simplifier la conception du circuit. De plus ces modes de fonctionnement vont permettre de classifier les circuits asynchrones.

Du point de vue du concepteur, la principale difficulté est de concevoir un système sans aléa. En effet, dans un circuit synchrone, la discrétisation du temps implémentée par un échantillonnage par le signal d'horloge, permet d'ignorer tous les phénomènes transitoires qui peuvent survenir dans les parties logiques combinatoires « *glitchs* ». En asynchrone, par contre, toute transition d'un signal peut être interprétée comme un événement porteur d'information. La principale difficulté réside donc dans la conception de parties combinatoires et séquentielles sans aléa ou « *hazard free* » (J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev., 2002).

2.2.2 LES DIFFERENTES CLASSES DE CIRCUITS ASYNCHRONES

Il existe différentes classes de circuits asynchrones en fonction des hypothèses sur les délais et des relations temporelles entre les événements (données ou contrôle). La Figure 5 présente la terminologie habituellement utilisée pour nommer les circuits asynchrones. Plus le fonctionnement du circuit respecte la notion d'asynchronisme, plus le circuit est robuste (en terme d'hypothèse sur les délais) et complexe.

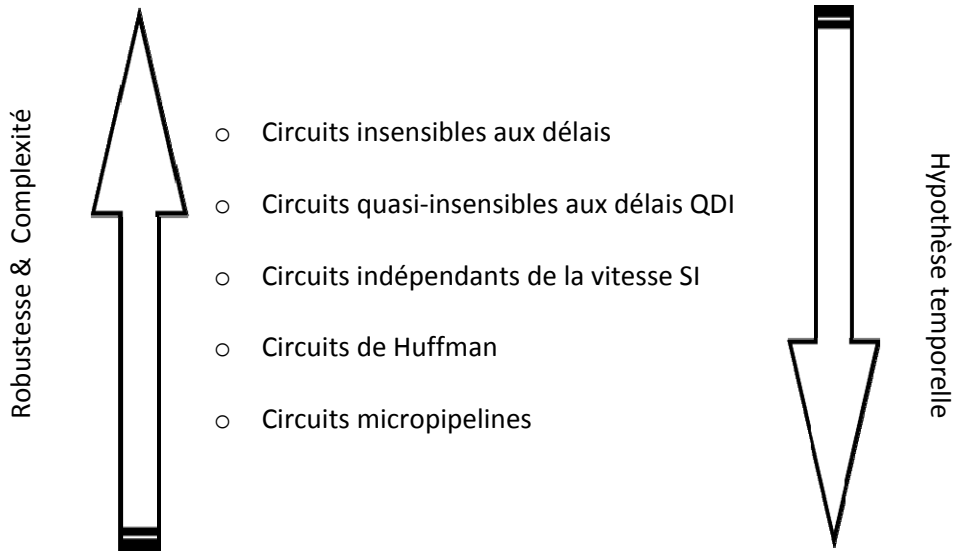


FIGURE 5: CLASSES DES CIRCUITS ASYNCHRONES

2.2.2.1 LES CIRCUITS INSENSIBLES AUX DELAIS : DI

Dans cette classe de circuits aucune hypothèse temporelle n'est introduite. Ceci dit qu'ils sont fonctionnellement corrects indépendamment des délais introduits par les fils ou les éléments logiques. Ils sont en fait basés sur un modèle de délai pour les fils et les éléments qui est « non borné».

Basés sur des travaux de Clark Wasley, et re-formalisés dans (J.T. Udding, 1986), les circuits DI sont supposés répondre toujours correctement à une sollicitation externe pourvu qu'il ait assez de temps pour calculer. Ceci impose donc au récepteur d'un signal de toujours informer l'expéditeur que l'information a été reçue. Les circuits récepteurs doivent donc être capables de détecter la réception d'une entrée et/ou la fin de son traitement. Les circuits émetteurs doivent attendre un compte rendu avant d'émettre une nouvelle donnée. Les contraintes de réalisation pratique, qu'impose l'utilisation de ce modèle, sont très fortes. De plus, la plupart des circuits conçus aujourd'hui utilisent des portes logiques à une seule sortie. L'adoption du modèle de délai non borné ne permet pas leur utilisation. En effet, les portes logiques standard ont leurs sorties qui peuvent changer si une seule de leurs entrées change. Toutes les composantes de ce type de circuits doivent s'assurer du changement des entrées avant de produire une sortie. Si la sortie change alors qu'une seule des entrées change, seul le changement de l'entrée active est acquitté, sans pouvoir tester l'activité des autres entrées. La seule porte à une sortie qui respecte cette règle est la porte de Muller. Malheureusement, les fonctions réalisables avec seulement des portes de Muller sont très limitées.

La seule solution est d'avoir recours à un modèle de circuits de type « portes complexes » pour les composants élémentaires. Dans ce cas la construction de ces circuits se fait à partir de composants standard plus complexes que de simples portes logiques et qui peuvent posséder plusieurs entrées et plusieurs sorties (S. Hauck, 1995), (J. Ebergen, July 1991).

2.2.2.2 LES CIRCUITS QUASI INSENSIBLES AUX DELAIS : QDI

Cette classe de circuits adopte le même modèle de délai non borné pour les connexions mais la notion de fourche isochrone (« *isochronic fork* ») est ajoutée. Une « fourche » est un fil qui connecte un expéditeur unique à deux récepteurs. Elle est qualifiée d'isochrone lorsque les délais entre l'expéditeur et les récepteurs sont identiques.

Cette hypothèse à des conséquences importantes sur le modèle et les réalisations possibles. Elle résout notamment le problème de l'utilisation de portes logiques à une seule sortie. Car si les fourches sont isochrones, il est permis de ne tester qu'une branche d'une fourche en supposant que le signal s'est propagé de la même façon dans l'autre branche. En conséquence, on peut autoriser l'acquittement d'une des branches de la fourche isochrone.

A.J.Martin a montré (A.J. Martin, 1993) que l'hypothèse temporelle de fourche isochrone est la plus faible à ajouter aux circuits insensibles aux délais pour les rendre réalisables avec des portes à plusieurs entrées et une seule sortie. Ainsi les circuits quasi insensibles aux délais se caractérisent par l'adoption d'un modèle de délais pour les connexions qui est de type non borné. En plus, l'hypothèse de fourche isochrone, et un modèle de type porte simple à délai non borné pour les composants élémentaires du circuit. Les circuits quasi insensibles aux délais sont donc réalisables avec des cellules standard telles qu'on les utilise pour la conception de circuits synchrones.

En pratique, la contrainte de fourche isochrone est assez faible, et est facilement satisfaite par une conception soignée, en particulier au niveau du routage et des seuils de commutation. Il suffit, finalement, que la dispersion des temps de propagation jusqu'aux extrémités de la fourche soit inférieure aux délais des opérateurs qui lui sont connectés (au minimum une porte et un fil dont une sortie interagit avec la fourche (A.J. Martin, 1990), (K. Berkel, 1993) et (K. Berkel, 1993).

2.2.2.3 LES CIRCUITS INDEPENDANTS DE LA VITESSE : SI

Les circuits indépendants de la vitesse font l'hypothèse que les délais dans les fils sont négligeables, tout en conservant un modèle non borné, pour les délais dans les portes. Ce modèle, qui n'est pas réaliste dans les technologies actuelles, est en pratique équivalent au modèle QDI à l'exception des fourches qui sont toutes considérées comme isochrones. Même si pendant longtemps la communauté a tenté de cerner les différences entre ces deux modèles, il y a aujourd'hui un consensus pour considérer les modèles QDI et SI équivalents. Hauck montre dans (S. Hauck, 1995) comment une fourche isochrone peut être représentée par un circuit indépendant de la vitesse (cf. Figure 6).

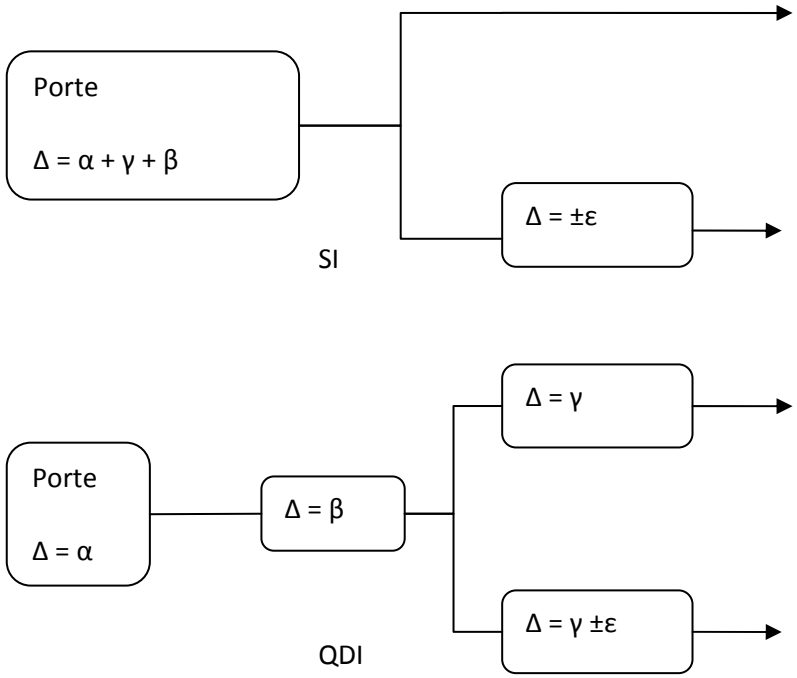


FIGURE 6: EQUIVALENCE ENTRE LES CIRCUITS SI ET QDI

Cependant, il semble plus pertinent d'utiliser le modèle QDI car celui-ci identifie les fourches isochrones de celles qui ne le sont pas. Ceci offre le moyen de vérifier les connexions du circuit qui ne sont pas insensibles aux délais au cours des différentes étapes de conception, et seulement celles-ci.

2.2.2.4 LES CIRCUITS MICRO PIPELINE

La technique micro pipeline a été introduite par Ivan Sutherland (Ivan E. Sutherland, 1989). Les circuits de cette classe sont composés de parties de contrôle insensibles aux délais qui commandent des chemins de données conçus en utilisant le modèle de délais bornés. La

structure de base de cette classe de circuits est le contrôle d'une file (FIFO) (cf. Figure 7). Elle se compose d'éléments identiques connectés tête-bêche. Les opérateurs notés « C » sont des portes de Muller dont la sortie est une copie des niveaux d'entrée lorsqu'ils sont identiques, et une copie du niveau de sortie précédente lorsque les entrées sont différentes.

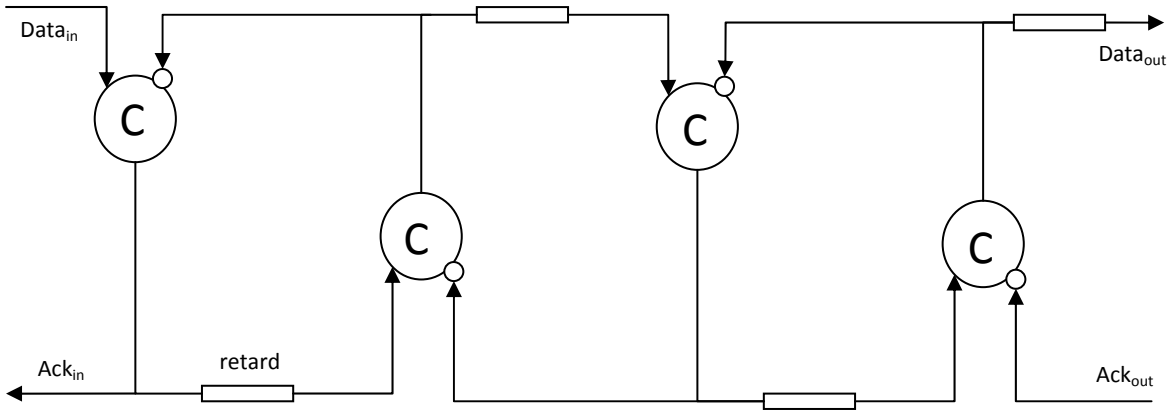


FIGURE 7: STRUCTURE DE BASE DES CIRCUITS MICROPIPELINES

Une transition positive sur Data$_{in}$ provoque une transition positive sur Ack$_{in}$ qui se propage également à l'étage suivant. Le deuxième étage produit une transition positive qui d'une part, se propage à l'étage suivant mais qui d'autre part, revient au premier étage l'autorisant à traiter une transition négative cette fois. Les transitions de signaux se propagent donc dans la structure tant qu'elles ne rencontrent pas une cellule occupée. C'est fonctionnement de type FIFO est bien insensible aux délais.

Cette structure est une structure de contrôle simple. Elle peut être utilisée pour contrôler un chemin de données possédant des opérateurs de mémorisation et des opérateurs de traitements combinatoires.

La première motivation pour le développement de cette classe de circuits était de permettre un pipeline élastique. En effet, le nombre de données présentes dans le circuit peut être variable, les données progressant dans le circuit aussi loin que possible en fonction du nombre d'étages disponibles ou vides. Cependant ce type de circuits révèle un certain nombre d'inconvénients. Tout d'abord, il faut remarquer que les problèmes d'aléas ont été écartés en ajoutant des retards sur les signaux de contrôle. Cela permet en fait de se ramener à un fonctionnement en temps discret dans lequel la mémorisation des données est autorisée seulement lorsqu'elles sont stables (à la sortie des portes combinatoires).

De nombreuses méthodes de conception s'attachent à concevoir des circuits micropipelines avec des modèles de délais plus ou moins différents. L'idée générale de ces approches est de séparer dans la spécification le contrôle des données et de les implanter séparément dans des styles différents. En particulier, les contrôleurs des registres de pipeline peuvent être obtenus avec un grand nombre de méthodes et avec des hypothèses de délais plus ou moins fortes.

2.2.2.5 LES CIRCUITS DE HUFFMAN

Les circuits de cette classe utilisent un modèle de délai identique aux circuits synchrones. Ils supposent que les délais dans tous les éléments du circuit et dans les connexions sont bornés ou même de valeurs connues. Les hypothèses temporelles sont donc du même ordre que pour la conception de circuits synchrones. Leur conception repose sur l'analyse des délais dans tous les chemins et boucles de façon à dimensionner les signaux de contrôles locaux qui s'apparentent d'avantage ici à des horloges locales. Ces circuits sont d'autant plus difficiles à concevoir et à caractériser qu'une faute de conception ou de délai les rend totalement non fonctionnels.

Dans la suite de ce manuscrit, les circuits QDI sont les seuls étudiés et évalués.

2.2.3 LE PROTOCOLE DE COMMUNICATION

Pour gérer les échanges d'informations, deux principaux protocoles de communication sont utilisés dans les circuits asynchrones : le protocole 2 phases (ou NRZ pour Non Retour à Zéro ou encore "*Half-handshake*"), et le protocole 4 phases (ou RZ pour Retour à Zéro ou encore "*Full-handshake*"). Les fonctionnements des deux protocoles sont décrits dans le chapitre Chapitre 3.

2.2.3.1 LE CODAGE DES DONNEES

Afin de détecter la présence d'une donnée et sa disponibilité, il est nécessaire d'adopter un codage particulier pour les données. Il est en effet impossible d'utiliser un fil unique par bit de donnée : cela ne permet pas de détecter que la nouvelle donnée prend un état identique que la valeur précédente.

Deux types de solutions sont possibles, soit la création d'un signal de requête associé aux données (codage de données groupées), soit l'utilisation d'un codage insensible aux délais double-rail par bit de donnée (codage double rail) ou multi-rail (J.B. RIGAUD, 2002).

2.2.3.1.1 CODAGE DE DONNEES GROUPEES

Cette solution consiste simplement à ajouter un fil au bus de données afin de spécifier si les données y sont valides (cf. Figure 8). Les bus de données utilisent le traditionnel schéma de la logique synchrone : un fil par bit de données, ce qui s'appelle parfois mono-rail « *single rail* » dans la littérature. Le fil spécifiant la validité des données, dit signal de requête, est typiquement implanté avec le retard adéquat. Ce retard est conçu égal ou supérieur au temps de calcul dans le pire cas.

Ce type de codage permet une bonne réalisation des circuits asynchrones a cause de son efficacité en terme de surface (en terme de nombre de fils et donc en nombre de portes pilotant ces fils). Cependant, les inconvénients de ce codage dans certains circuits sont qu'en utilisant le retard assorti, le fonctionnement est fixé au pire cas : le fonctionnement ne dépend donc pas de la propagation réelle des données à l'intérieur d'un étage, et les circuits obtenus ne sont plus insensibles aux délais.

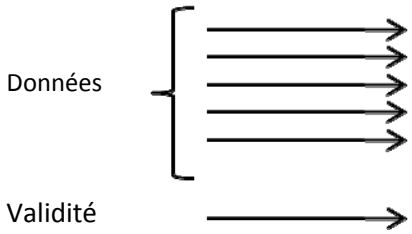


FIGURE 8: CODAGE DONNEE GROUPEES

2.2.3.1.2 CODAGE DOUBLE RAILS

Cette solution est, contrairement au codage « données groupées », insensible aux délais. Elle consiste à intégrer l'information de la validité dans les données. Cette approche possède la propriété suivante : les données sont détectées à l'arrivée sans que cela ne repose sur aucune hypothèse temporelle. Ceci implique donc robustesse, portabilité et facilité lors de la conception.

Ce codage double rails utilise deux fils pour chaque bit de donnée. Cela double donc le nombre de fils par rapport à un codage binaire standard. Avec deux fils par bit de donnée, quatre états sont utilisables pour exprimer deux valeurs logiques ("0", "1"). Deux codages sont communément adoptés : l'un utilisant trois états seulement, l'autre utilisant les quatre états.

1. Codage 4-états : dans ce codage, chaque bit de donnée est représenté par deux fils. Parmi les quatre états possibles pour ces 2 fils, la moitié est réservée à la valeur 0, l'autre à la valeur 1 (cf. Figure 9). L'émission d'une

nouvelle donnée se traduit par le changement d'un seul fil : celui de droite pour exprimer la même valeur que précédemment, celui de gauche pour exprimer la valeur opposée. La validité des données est donc assurée par le changement du couple de fils.

2. Codage 3-états : dans ce codage également, chaque bit est représenté par 2 fils. En revanche, les valeurs représentées ne sont pas dupliquées : une seule combinaison représente chaque valeur, tandis que la troisième indique l'état invalide et le quatrième est inutilisée (cf. Figure 9). Ainsi, le passage d'une valeur valide à l'autre se traduit nécessairement par le passage par l'état invalide.

Parmi tous ces codages, le codage 3 états, qui peut par ailleurs sembler le moins naturel, est aujourd'hui de loin le plus utilisé pour des raisons d'implantations et de sécurité.

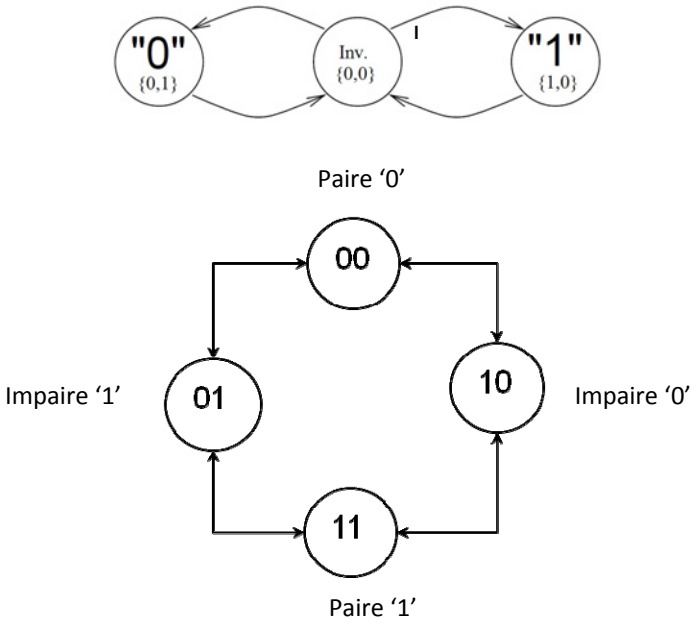


FIGURE 9: CODAGE DOUBLE RAIL : 3-ETATS ET 4-ETATS

2.2.4 LES CIRCUITS QDI ASYNCHRONES POUR LA SECURITE

Les circuits Quasi-insensibles aux délais semblent avoir la meilleure résistance contre les attaques par canaux cachés que les autres classes de circuits asynchrones, en particulier, les circuits de type micropipeline. Ces derniers présentent des caractéristiques semblables à celles des circuits synchrones face aux attaques en puissance de type DPA[16].

L'analyse des courbes de courant des circuits synchrones, est fortement brouillée par les effets de la consommation des arbres d'horloge qui introduisent des signaux parasites. L'absence d'horloge globale dans les circuits micropipeline rend l'analyse des courbes encore plus facile car elle reflète directement l'activité des blocs du composant.

En effet, les circuits QDI offrent une meilleure résistance naturelle aux attaques par canaux cachées (F. Bouesse, 2005), et ce grâce à leurs propriétés intrinsèques qui augmentent considérablement leur résistance à ce genre d'attaques :

1. Le codage des données de type 1 parmi n : Ce codage permet de garder un poids de Hamming constant quelles que soient les données manipulées. Réduisant ainsi les dépendances entre les données manipulées et le courant consommé. Il augmente aussi la résistance des circuits QDI aux attaques par injection de fautes en utilisant l'état invalide pour générer des alarmes en cas de détection de fautes.

2. Le protocole de communication 4-phase : Contrairement aux circuits synchrones dans lesquels la consommation du courant dépend de la valeur précédente, l'utilisation du protocole de communication 4 phases assure une remise à zéro de tous les nœuds logiques avant le prochain calcul. Il permet non seulement de mieux répartir la consommation du courant dans le temps, mais également de réduire considérablement les pics de courant ainsi que l'émission électromagnétique tout en lissant la forme d'onde du courant. Dans la littérature, les travaux qui traitent les caractéristiques des circuits asynchrones face aux attaques par analyse des émissions électromagnétiques sont très peu nombreux. Il est ainsi difficile aujourd'hui de conclure sur les capacités de résistance des circuits asynchrones face à ce type d'attaque.

3. Chemins de données équilibrés : Les circuits asynchrones offrent la possibilité de contrôler avec précision le nombre de transitions logiques dans chaque bloc de calcul. Cela permet d'équilibrer les chemins de données afin de garder une consommation indépendante des données sur chaque chemin. Par exemple, dans un circuit QDI, le concepteur peut

[16] Differential Power Analysis

assurer la commutation d'un nombre constant de portes, quelques soient les données.

4. Le contrôle local : Il permet de mieux répartir l'activité du circuit dans le temps et de rendre plus difficile la synchronisation sur un événement particulier. Cela rend quasiment impossible toute attaque temporelle en augmentant les difficultés de synchronisation.

Cependant, il reste possible de réaliser une attaque en puissance sur les circuits QDI (F. Bouesse, M. Renaudin, B. Robisson, E. Beigne, P. Y. Liardet, S. Prevosto, and J.Sonzogni, 2004). En effet, la phase de placement routage apporte de nombreux déséquilibres sur les chemins de données. Ces déséquilibres diminuent significativement les avantages apportés par la logique asynchrone, et créent une source de fuite pour réaliser les attaques. Ce problème a été formalisé dans (F. Bouesse, 2005) et (F. Bouesse, M. Renaudin, S. Dumont and F. Germain, March 7-11, 2005), et des solutions sont proposées pour améliorer la résistance des circuits (F. Bouesse and M. Renaudin, June 8 -11th 2005.) et (F. Bouesse, G. Sicard and M. Renaudin, Oct 10th-13th, 2006).

2.3 UTILISATIONS DES FPGAS POUR LES APPLICATIONS CRYPTOGRAPHIQUES

L'utilisation des FPGA[17]s pour la cryptographie est un domaine qui devient de plus en plus attirant pour divers raisons. Malgré son intérêt, les FPGAs actuels souffrent de problèmes de sécurité majeurs en tant que systèmes complets.

En effet, les critères de sécurité qui régissent les différentes applications cryptographiques, sont uniques aux contextes dans lesquels ils sont utilisés. Outre les aspects généraux : algorithmiques, vitesse et coût de fabrication qui se trouvent dans la plupart des domaines, existent des critères spécifiques à la cryptographie : la sécurité physique des composants, la flexibilité des composants, leurs consommation du courant et d'autres sources de fuite dues aux canaux cachés.

Les avantages des implémentations dites logiciels ou *« software implementations »* sont multiples y compris, la facilité d'usage, la portabilité, les coûts faibles de développement, le coût faible de l'unité et la flexibilité et la possibilité de la mise à jour. Cependant, les applications offrent des performances moyennes en termes de vitesse, de consommation relativement élevée par rapport aux ASIC, et surtout une sécurité limitée du système (B. Schneier, 1996).

Les implémentations matérielles ASIC, quant à eux, offrent un coût très faible par unité, de très bonnes performances, et de très basses consommations. De plus, les circuits ASICs semblent donner des résultats plus adaptés aux problèmes de sécurité, parce qu'ils ne sont pas modifiables ou reconfigurables ou facilement lisibles par un attaquant (R. Doud, April 1999). Les inconvénients des ASICs ne sont non plus négligeables ; ils se manifestent par un coût de développement élevé et un manque de flexibilité au niveau de changement de paramètres ou d'adaptation des algorithmes utilisés.

Dans ce compromis, les circuits reconfigurables, comme les FPGAs semblent pouvoir combiner les avantages des implémentations logicielles et matérielles. En même temps la sécurité de FPGAs reste toujours un inconvénient majeur quand il s'agit des applications cryptographiques. Depuis les années 90, plusieurs travaux de recherche ont été menés pour étudier le comportement des FPGAs vis-à-vis des algorithmes de cryptographie, en terme de haute performance (O. Kommerling and M.G. Kuhn, May 1999.)(C . K. Koc, D. Naccache, and C. Paar, editors, May 13-16, 2001)(C . K. Koc and C. Paar, editors, August 17-18, 2000)(B. S. Kaliski, Jr., C. K. Koc, and C. Paar, editors, August 13-15, 2002). Cependant très peu de travaux ont été faits sur les problèmes de sécurité spécifiques aux FPGAs et les contremesures à mettre en œuvre pour se protéger des attaques existantes. Il convient de noter que la principale menace pour une application cryptographique n'est plus la cryptanalyse dite logicielle, mais plutôt

[17] Field Programmable Gate Array

l'exploitation d'une faiblesse physique du circuit qui l'implémente. Il s'agit d'analyser les sources de fuite d'information dites canaux cachés.

2.3.1 AVANTAGES DES FPGAS POUR LA CRYPTOGRAPHIE

Les FPGAs offrent des avantages majeurs pour les applications cryptographiques. Cette partie présente une liste des principaux avantages qui ont fait l'objet de plusieurs travaux de recherches dans la littérature (A. Elbirt, W. Yip, B. Chetwynd, and C. Paar, August 2001) :

L'Agilité : C'est la capacité à changer l'algorithme de cryptographie. Il existe de nombreux protocoles de sécurité de nos jours. Ils diffèrent en fonction des plateformes, et du niveau de sécurité exigé. La majorité de ces protocoles sont indépendants des algorithmes de cryptage, et en peuvent utiliser plusieurs. Cette indépendance est intéressante pour plusieurs raisons car :

1. Elle facilite la suppression d'un algorithme cassé, et plus utilisé.
2. Elle permet de choisir un algorithme adapté en fonction de l'application et des préférences de l'utilisateur.
3. Elle permet d'ajouter des nouveaux algorithmes au système.

Cette agilité est coûteuse pour être implémentée sur un ASIC mais, en revanche elle ne l'est pas sur un FPGA. En effet il est possible de le reprogrammer.

Le téléchargement d'algorithme : Les systèmes cryptographiques sont mis à jour continuellement pour plusieurs raisons, comme par exemple la compatibilité du système avec de nouvelles applications. D'un point de vue cryptographie, le téléchargement d'un nouvel algorithme peut être nécessaire parce que l'algorithme de cryptage utilisé est cassé (comme le DES[18] par exemple) ou que le standard actuel est expiré, ou bien qu'un nouveau standard a fait son apparition (DES→ AES[19]) ou pour toute autre raison. Alors que cette mise à niveau du système est facile à mettre en œuvre sur les FPGAs, elle est infaisable pour une implémentation ASIC où l'algorithme de cryptage a été défini une fois pour toute.

L'efficacité de l'architecture : c'est la possibilité de personnaliser les paramètres d'un système cryptographique en reprogrammant le FPGA. En effet, dans certains cas, l'architecture d'un système peut être plus efficace lorsqu'elle est conçue spécifiquement. En effet, plus l'implémentation d'un système est spécifique meilleures seront ses performances. Le FPGA offre ce genre de possibilité. Il est ainsi possible de faire évoluer et d'optimiser totalement ou partiellement l'architecture d'une application.

[18] Data Encryption Standard
[19] Advanced Encryption Standard

L'efficacité des ressources : Le FPGA, grâce à sa capacité de se reconfigurer, permet une meilleure gestion des ressources que l'ASIC, quand il s'agit des applications cryptographiques. En effet, la majorité des protocoles de sécurité utilisés sont hybrides, ex. SSL (A. O. Freier, P. Karlton, and P. C. Kocher, November 1996), TLS (C. Allen, T. Dierks, January 1999). Cela implique qu'un algorithme à clé publique est utilisé pour transmettre la clé de la session. Ensuite, un autre algorithme à clé privé est nécessaire pour le cryptage des données. Puisque les algorithmes ne sont pas utilisés simultanément, le même dispositif peut être utilisé pour les deux fonctions, à condition de le reconfigurer pendant l'exécution.

Performance en vitesse : Les performances des ASIC sont supérieures à celles des FPGAs actuels. Dans le domaine de la cryptographie, le circuit Amphion CS5240TK atteint une vitesse de cryptage de 25,6 Gbit/s à 200 MHz (Amphion) ce qui est le double de celle atteinte par le FPGA Virtex XCV-1000BG560-6 : 12Gbit/s (K. Gaj, P. Chodowiec, April 8-12, 2001). Cependant, la performance des FPGAs dépasse celle des processeurs à usage général. Ces derniers ne sont pas optimisés pour des exécutions rapides. C'est particulièrement vrai dans certains cas d'algorithmes de chiffrement : ex. un bloc AES a enregistré une vitesse de 112,3Mbit/s pour un DSP TI TMS320C6201 (T. Wollinger, M. Wang, J. Guajardo, and C. Paar, April 13–14 2000), une vitesse de 718,4Mbit/s pour un Pentium III (H. Lipmaa), contre une vitesse de 12 Gbits/s pour un Virtex XCV-1000BG560-6 (K. Gaj, P. Chodowiec, April 8-12, 2001).

Coût : Pour déterminer le coût de fabrication d'un circuit numérique, il faut prendre en considération deux facteurs : le coût de développement et le coût de l'unité. Le FPGA gagne sur le premier facteur. En effet le coût de développement d'une implémentation sur FPGA d'un circuit donné est inférieur à celui d'un ASIC, grâce à sa reconfigurabilité. Un concepteur est capable de reconfigurer un FPGA et de tester autant qu'il veut de nouvelles architectures sans coût additionnel. Cela aussi induit en général un temps plus court pour la mise sur le marché, chose qui est très importante dans nos jours. Quant au deuxième facteur, les ASIC coûtent moins que les FPGAs lorsqu'il s'agit de gros volumes de production. Cependant, la plupart des temps, le coût de l'unité est généralement moins significatif que le coût de développement.

2.3.2 LES DEFAUTS DE SECURITE DES FPGAS

Les algorithmes cryptographiques utilisés dans la plupart des applications commerciales sont connus. La connaissance de la clé utilisée, que se soit une clé publique ou privée, est le danger qui menace ces applications. En conséquence, un attaquant cherche à trouver cette clé qui va lui permettre de décrypter des données transmises, ou des données déjà utilisées. Une autre menace peut être le « clonage », d'un algorithme cryptographique avec sa clé. Dans certains cas, cela peut être suffisant pour faire fonctionner l'algorithme cloné, dans un mode de décryptage. Dans le cas où les algorithmes de cryptage sont propriétés de l'entreprise, un autre défi menace leur sécurité : le « *reverse-engineer* ». C'est le cas par exemple des applications militaires, ou des applications du genre télé-paiement, etc.

Les attaques par canaux cachés appliquées sur les FPGAs, quant à elles, commencent à intéresser les chercheurs et les fraudeurs à cause de leurs efficacités, de leurs faibles coûts et de la facilité pour les mettre en place.

2.3.2.1 VULNERABILITE DES FPGAS AUX ATTAQUES PAR ANALYSE DE COURANT

2.3.2.1.1 FPGAS VS ATTAQUES PAR ANALYSE DE COURANT

La consommation de courant dans un circuit se fait sous deux formes : statiques et dynamiques.

La consommation dynamique est due au changement d'état des portes CMOS, lorsque les capacités de charge « *load capacitance* » se chargent ou se déchargent selon la transition logique 0→1 ou 1→0 respectivement. Cela inclue aussi le bref temps où les p et n –mos sont tous les deux passants lors d'une commutation. Ces phénomènes peuvent être modélisés par l'équation suivante :

$$P = C_l . V_{dl}^2 . f . A$$

Où 'C_l' est la capacité de charge de la porte concernée. Elle comprend la capacité parasite, la capacité du fil et la capacité de l'entrée qui se chargent et se déchargent à chaque transition ; 'V_{dl}' est la tension d'alimentation de la porte ; 'f' est la fréquence et 'A' est la probabilité d'une transition 0→1 ou 1→0. Standaert, dans son papier (F-X. Standaert, S. B. Ors and B. Preneel, 2004) confirme ce modèle sur un FPGA par une simple expérience.

Pour obtenir les traces de courant, le courant est mesuré à l'aide une résistance placée entre le circuit et la source d'alimentation externe. Bucci propose dans (M. Bucci, L. Giancane, R. Luzzi, G. Scotti and A. Tri, 2006) une autre méthode qui améliore la qualité des traces. Shang et al. montrent dans (L. Shang, A. S. Kaviani and K. Bathala, 2002), que 60% de la consommation dynamique du courant d'un FPGA Virtex-II 150 nm est due au réseau d'interconnexion, 16% au logique programmable, et 14% à l'horloge.

L'analyse de courant d'un circuit intégré permet de révéler des informations sur les données manipulées, comme par exemple : la clé de chiffrement dans le cas d'un cryptosystème. La plupart des recherches dans ce domaine ont été appliquées sur des ASICs notamment sur des microprocesseurs (ex. carte à puce) pour lesquels la construction d'un modèle et l'acquisition des traces de courant sont faciles à obtenir. Dans (S. Mangard, E. Oswald, T. Popp, 2007), S. Mangard présente les techniques de l'analyse de courant pour les cartes à puce. En 2003, Ors et al. dans (S. B. Ors, E. Oswald, B. Prenee, 2003) et Standaert et al. dans (F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde and J.-J. Quisquater, September 2003) étaient les premiers à tester la possibilité d'appliquer la méthode d'analyse de courant aux FPGAs. Ors et al. ont présenté dans leur papier une plateforme pour réaliser une attaque SPA

sur un FPGA Xilinx Virtex 220 nm. En revanche Standaert et al. ont démontré qu'une SPA n'est vraiment pratique - sur le même FPGA - que lorsqu'il s'agit d'applications cryptographiques parallèles où plusieurs opérations concurrentes tournent sur le même dispositif.

En général, la DPA est préférée, et Standaert et al. ont publié une attaque réussie basée sur des techniques de corrélation, contre l'implémentation d'un DES et d'un AES (F-X. Standaert, S. B. Ors and B. Preneel, 2004) et (F.-X. Standaert, S. B. Ors, J.-J. Quisquater and B. Preneel, August 2004). Le résultat obtenu a été ensuite amélioré en 2006 par Standaert et al. dans (F.-X. Standaert, F. Mace, E. Peeters and J-J. Quisquater, 2006), dans laquelle ils ont analysé la signature de courant d'une structure pipeline sur un FPGA Spartan-II 180 nm. Ils ont aussi prouvé qu'une seule contre-mesure peut rendre l'attaque difficile, mais qu'elle ne suffit pas pour résister totalement à ce type d'attaques.

Pour rendre une attaque par analyse de courant difficile à appliquer, un cryptosystème doit avoir la même signature de courant pour touts ses opérations, chose qui est difficile à réaliser. Standaert et al. présentent dans (F.-X. Standaert, E. Peeters, G. Rouvroy and J.J. Quisquater, Febuary 2006) une multitude de contre-mesures et notamment : *timing randomization, noise addition, masking, etc.…*. Tiri et Verbauwhede ont proposé dans (K. Tiri, I. Verbauwhede, 2004) une contre-mesure spécifique pour les FPGAs appelée WDDL[20], basée sur des portes spécifiques WDDL. Leur méthode devrait être capable d'augmenter la résistance aux DPAs, en rendant la consommation du courant indépendante des transitions logiques. En 2008 S. Guilley et al. ont proposé dans (S. Guilley, S. Chaudhuri, L. Sauvage, P. Hoogvorst, R. Pacalet and G.M. Bertoni) une contre mesure appelée WDDL$^+$ pour améliorer la résistance des circuits aux attaques par analyse de courant. Ils ont ensuite publié en 2009 la première attaque EMA réussie sur un crypto processeur DES implémenté sur un FPGA et utilisant cette contre mesure (S. Guilley, L. Sauvage, J-L. Danger, T. Graba and Y. Mathieu, july 2008). Ils ont montré que la logique WDDL$^+$ est adaptée pour lutter contre les attaques par analyses du courant. En revanche elle ne l'est pas contre les attaques électromagnétiques.

2.3.2.1.2 FPGAs VS ATTAQUES ELECTROMAGNETIQUES

Les attaques électromagnétiques sont basées sur l'analyse du champ magnétique produit par un cryptosystème pendant l'exécution d'une opération. Ce champ est dû aux mouvements des charges dans le circuit. Il peut être mesuré à l'aide d'une antenne spécifique placé tout prêt du circuit. Kuhn raconte dans (M.G. Kuhn, December 2003) l'histoire de l'évolution de ce type d'attaques, tout en présentant aussi les expériences réalisées ainsi que les résultats obtenus.

[20] Wave Dynamic Differential Logic

Ces attaques EMAs[21] ont intéressé la communauté de recherche depuis le début des années 90. Toutes les applications à l'époque concernaient les ASICs. La différentiation entre analyse électromagnétique simple et différentielle a été introduite pour la première fois à la conférence Eurocrypt en 2000 par Quisquater et Samyde. Ils ont ensuite publié leurs techniques et les résultats obtenus dans (J.-J. Quisquater and D. Samyde, 2001). Dans la même période, Gandolfi et al. (K. Gandolfi, C. Mourtel, F. Olivier, May 2001) ont publié leurs travaux réalisés sur trois circuits cryptographiques. Leurs résultats ont démontré qu'une EMA bien mise en place, peut être plus efficace et produit des signaux avec un rapport signal sur bruit meilleur que celui des DPAs et SPAs.

Carlier et al. ont été les premiers à avoir publié l'analyse d'une EMA sur un AES implémenté sur un FPGA Cyclone 130 nm de Altera (V. Carlier, H. Chabanne, E. Dottax and H. Pelletier, 2004). Ils ont introduit une nouvelle méthode d'analyse électromagnétique appelée *Square EMA*. Dans cette attaque, ils ont fixé tous les octets d'entrée sauf un seul. Ils ont ensuite observé sa propagation partout dans la fonction « *round* » de l'algorithme. Les auteurs ont réussi à obtenir des bits de quelques clés utilisées, en plaçant une antenne près du FPGA et utilisant la technique DEMA; ils étaient aussi capables de distinguer certains signaux du bruit produit par des processus parallèles.

De Mulder et al. (E. De Mulder, P. Buysschaert, S. B. Ors, P. Delmotte, B. Preneel, G. Vandenbosch and I. Verbauwhede, November 2005) ont publié une attaque réussie contre une implémentation d'une application cryptographique sur un FPGA Virtex-800 220 nm de Xilinx. Plus tard ils ont donné dans (E. De Mulder, S. B. Ors, B. Preneel and I. Verbauwhede, July 2006) plus de détails concernant l'attaque qu'ils ont réalisée.

Finalement, l'émission électromagnétique est un canal qui peut fourni beaucoup d'information à un attaquant. Argrawal et al. (D. Agrawal, J. R. Rao and P. Rohatgi, September 2003) ont démontré que la combinaison d'une attaque électromagnétique avec une attaque par analyse de courant s'avère être la meilleur méthode pour améliorer les résultats obtenus.

2.3.2.1.3 FPGAs VS ATTAQUES TEMPORELLES

Chaque opération qui se produit dans un circuit, nécessite un certain temps pour être achevée. Cette signature temporelle est considérée comme source de fuite d'information surtout lorsque les données manipulées sont secrètes comme une clé d'un algorithme de chiffrement. En analysant les différents temps de traitement obtenus, un attaquant peut déterminer la nature des données manipulées. Kocher(P. Kocher, 1996), qui a introduit ce type d'attaques, et Dhem (J. F. Dhem, F. Kouene, P-A. Leroux, P. Mestre, J- J. Quisquetr and J- L. Willems, 1998) ont prouvé l'efficacité d'une telle attaque contre des applications cryptographiques.

[21] Electromagnetic Analysis

Cependant, cette analyse, n'a pas été de la même efficacité contre les implémentations sur FPGAs. La raison est que, contrairement aux microcontrôleurs, les processus tourne en concurrence. Très peu de travaux ont été publiés dans ce domaine. Généralement, pour empêcher une attaque temporelle, il faut s'assurer que toutes les opérations utilisant des données sensibles prennent le même temps de traitement (ou plus particulièrement, le même nombre de cycle d'horloge dans le cas des circuits synchrones). D'autres contre-mesures sont aussi proposées comme par exemple l'ajout d'un tirage aléatoire de l'instant des opérations ou « *timing randomization* « .

2.4 CONCLUSION

Depuis les années 90, les systèmes embarqués sont devenus très présents dans notre vie quotidienne ; beaucoup d'applications commencent à traiter des données où leur confidentialité et leur intégrité deviennent des éléments très importants. La conception de ces systèmes nécessite l'utilisation à la fois d'algorithmes cryptographiques résistant à la cryptanalyse logicielle et l'utilisation de support matériel d'implantation robustes contre les attaques matérielles. Ces derniers prouvent jours après jour leur efficacité à déchiffrer les données secrètes. Certaines d'entre elles ont même été combinées pour plus d'efficacité.

Plusieurs travaux de recherches ont montré que la logique asynchrone peut constituer une défense efficace pour la conception de systèmes sécurisés contre les attaques par canaux cachés.

D'autre part, les FPGAs ont atteint un haut niveau de performance, qui les a permis de remplacer les ASIC dans certaines applications. En outre, ils offrent des propriétés vis-à-vis de la sécurité, qui les rendent robustes contre certains types d'attaques (S. Drimer, 2008), (T. Wollinger and C. Paar, 2004) et (T. Wollinger, J.Guajardo and C. Paar, 2004).

Dans la littérature, plusieurs FPGAs asynchrones existent : (A.J. Martin, September,1997), (Achronix), (A.J.Martin and M. Nystrom, May 1997.), (J.D. Garside et al, April 2000) etc. D'un point de vue flexibilité, ils sont tous dédiés soit pour un seul style de logique asynchrone (PGA-STC (K. Makeswaran and V. Akella, March 1998), PAPA (J. Teifel and R. Manohar, February 2004)) soit pour une application spécifique (MONTAGE (S. Hauck, G. Boriello and C. Ebeling, August 1992), GALSA (B. Gao, December 1996), STACC (R. Payne, 1997)). PGA-STC implémente le protocole 2 phases, GALSA est dédié aux architectures massivement parallèles, PAPA et Speedster (le FPGA le plus rapide au monde) ont été optimisés pour des débits très élevés et reposent sur une architecture de type pipeline à grains fins.

Du point de vue de la sécurité, tous ces FPGAs sont vulnérables aux attaques par canaux cachés, notamment les SPAs et les DPAs. Très peu de travaux ont traité de la sécurité des FGAs au niveau matériel.

Les chapitres suivants présenteront l'architecture d'un FPGA asynchrone, nativement robuste contres les attaques par canaux cachés. Ils décrivent ses caractéristiques et présentent son prototype.

Chapitre 3. SPÉCIFICATION D'UN FPGA ASYNCRHONE POUR LA SÉCURITÉ

Après avoir fait le tour des principaux axes (FPGA, sécurité, logique asynchrone) sur lesquels se positionne ce manuscrit, ce chapitre introduit le travail réalisé dans le cadre de cette thèse. Il présentera les objectifs qui ont été fixés et qui sont répartis en deux types :

- Objectifs fonctionnels : ils décrivent le niveau de flexibilité que doit atteindre le FPGA, ainsi que les fonctions de bases et les protocoles de communication qu'il doit mettre en œuvre.

- Objectifs pour la sécurité : ceux-ci concernent les contremesures qui doivent être prises en compte pour améliorer la résistance du FPGA à certains types d'attaques notamment les attaques par canaux cachés. Sur ce point, il faut prendre en compte aussi bien l'aspect électrique que l'aspect logique du FPGA.

Ce chapitre est organisé en trois parties :

La première partie présentera les spécifications fonctionnelles du FPGA, et l'objectif qu'il faut atteindre en termes de flexibilité et de capacité à implémenter les différentes fonctions asynchrones, les différents codages de données et protocoles de communication.

La deuxième partie décrit les contraintes à respecter pour équiper le FPGA asynchrone de contremesures électriques et logiques, capables de le protéger contre différentes types d'attaques (analyse de courant, analyse électromagnétique, analyse en temps…).

Finalement la troisième partie montre l'impact du tech-mapping ainsi que celui du routage sur la robustesse du FPGA aux cryptanalyses.

3.1 Aspects fonctionnels – buts et objectifs

Le premier objectif de ce travail est de concevoir un FPGA asynchrone multi-style. Multi-style au sens qu'il est capable d'implémenter plusieurs types de protocoles de communication (protocoles 4 phases et 2 phases) ainsi que différents types de codages de données (codage de données 1 parmi n, double rails…). En effet le FPGA est destiné – en sus des traditionnelles fonctions sécurisées – à étudier et évaluer les différents types de contre-mesures, ainsi que l'impact des différents styles de logiques asynchrones sur la sécurité et les performances du circuit. Le FPGA permet aussi de comparer ces différents styles en termes de surface d'implémentation, de vitesse, de robustesse et de résistance contre les attaques par canaux cachés.

Les circuits QDI[22] asynchrones utilisent deux types de protocoles de communications, le protocole 4 phases et le protocole 2 phases. Ce dernier peut être divisé en deux sous types : 2 phases LEDR[23] et 2 phases sur fronts. Dans ce qui suit, une présentation de chaque protocole montre des implémentations de fonctions utilisant des codages en 1 parmi n. Le but étant de construire une liste de spécification à respecter lors de la conception de la logique programmable et notamment du PLB[24] : combien d'entrées doit-il avoir ? Combien de sorties ? Comment le rendre capable d'implémenter plusieurs protocoles ? Comment générer les signaux d'acquittement ? Et surtout comment l'optimiser et le sécuriser ?

[22] Quasi Delay Insensitive
[23] Level Encoded Dual Rail
[24] Programmable Logic Bloc

3.1.1 IMPLEMENTATION DU PROTOCOLE 4 PHASES

3.1.1.1 PRINCIPE DU PROTOCOLE « HANDSHAKE » : 4 PHASES 1 PARMI N.

Le protocole *Handshake* est un protocole bidirectionnel établi entre les différents blocs d'un circuit. Il assure une synchronisation locale de la communication entre sous-blocs. Le protocole *Handshake* possède deux variantes : le protocole 4 phases données groupées et le protocole 4 phases 1 parmi n. En 4 phases données groupées, les données sont séparées des signaux de requête et d'acquittement. Dans le cas du codage en 1 parmi n, le signal de requête est codé dans le bus des données. Dans la suite du chapitre, le protocole 4 phases choisi est le protocole 4 phases avec codage 1 parmi n. Le terme 1 parmi n défini le mode de codage des données. Il signifie que pour coder un mot logique de cardinal n, il faut utiliser « n » signaux (analogiques), où un seul prend un niveau logique haut à la fois. Par exemple, dans le cas du codage 1 parmi 2 - aussi appelé double rail - la représentation d'un bit logique nécessite 2 signaux. Cela se traduit physiquement par l'utilisation de deux fils : w_0 et w_1. Si le bit est un '0' logique, w_0 est mis à l'état haut tandis que w_1 est mis à l'état bas (ce qui donne w_0 =1 et w_1 =0). Si le bit est un '1' logique, w_1 est mis à 'état haut tandis que w_0 est mis à l'état bas (ce qui donne w_0 =0 et w_1 =1).

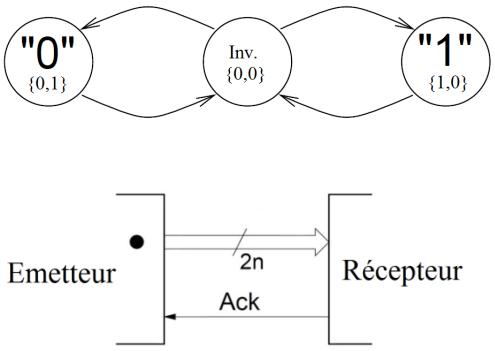
Dans un but de simplifier et d'améliorer la lisibilité et la clarté de ce document, le codage en 1 parmi 2 est adopté dans ce qui suit.

Le codage en 1 parmi 2 utilise deux rails pour coder non seulement les données mais aussi le signal de requête (cf. Figure 10). En effet pour un bit d'information « d », le protocole utilise un signal de requête codé sur les deux rails « d_t » et « d_f ». Le premier sert à coder le bit '1' ou « *true* » et le deuxième sert à coder le bit '0' ou « *false* ». Le protocole 4 phases utilisé exploite un tel codage. Ce protocole est très robuste car la communication entre deux blocs ne dépend pas des délais dans les interconnexions. Il s'agit du protocole quasi-insensible au délai.

Soit $\{x_f, x_t\}$ la paire de fils utilisée pour coder un bit :

- $\{x_f, x_t\} = \{1, 0\}$ et $\{x_f, x_t\} = \{0, 1\}$ représentent respectivement une donnée valide à l'état logique 0 et 1.
- $\{x_f, x_t\} = \{0, 0\}$ traduit l'absence de données, il s'agit de «données invalides».
- $\{x_f, x_t\} = \{1, 1\}$ est un cas interdit, donc non utilisé.

La transition directe d'un état valide à un état valide est aussi interdite. Il faut toujours passer par l'état invalide comme le montre la Figure 10. Ce codage est appelé codage 3-états.



Donnée	d_t	d_f
Invalide - Ω	0	0
Valide '0'	0	1
Valide '1'	1	0
Non utilisée	1	1

FIGURE 10 : PROTOCOLE 4 PHASES DOUBLE RAILS - CODAGE 3-ETATS.

Cela amène à une vue plus abstraite du fonctionnement de ce protocole, Figure 11:

1. L'émetteur envoie une donnée valide : {0,1} ou {1,0}.
2. Le récepteur reçoit la donnée, effectue son calcule, et active le signal d'acquittement.
3. L'émetteur, après réception du signal d'acquittement, bascule le canal de données à l'état invalide : {0,0}.
4. Le récepteur, reçoit la donnée invalide, et désactive son signal d'acquittement. A ce moment l'émetteur peut recommencer le cycle, en envoyant une nouvelle donnée valide.

Dans ce scénario, le bloc actif qui initie l'échange de données est l'émetteur. Il est connu sous le nom de « Push Channel ». Dans le cas contraire, où l'échange est initié par le récepteur, est appelé « Pull Channel ».

Ce type de protocole 4 phases, est connu sous le nom de WCHB[25] (J.B. RIGAUD, 2002) dans la communauté asynchrone et sous le nom de DPL[26] (T. Popp and S. Mangard, 2005) dans la communauté de sécurité. C'est un protocole très connu pour sa facilité d'implémentation.

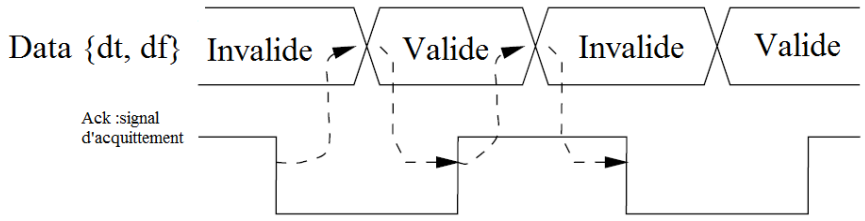


FIGURE 11 : PROTOCOLE 4 PHASES DOUBLE RAILS

[25] Weak Condition Half Buffer
[26] Dual-rail Precharge Logic

La même approche est associée au protocole 4 phases avec un codage en 1 parmi n : En effet, un bus de données de N bits est formé par une simple concaténation de N fils. Ces fils utilisent le codage décrit ci-dessus. Un récepteur est toujours en mesure de détecter les moments où toutes les données sont valides - auxquels il répond en activant son signal d'acquittement -, et ceux où les données sont invalides - auxquels il répond en désactivant son signal d'acquittement - (T. Verhoeff, 1988).

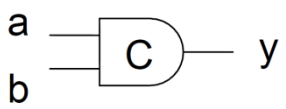
3.1.1.2 La porte de Muller – element de memorisation

Dans un circuit synchrone l'échantillonnage est assuré par un signal d'horloge qui arrive à des moments bien précis, où les signaux sont stables et valides. Entre les évènements d'horloge, les signaux peuvent avoir un comportement hasardeux, et subir plusieurs transitions avant que les circuits combinatoires ne se stabilisent. Ce genre de comportement est acceptable car il n'affecte pas le circuit d'un point de vue fonctionnel.

La situation est différente dans le cas des circuits asynchrones. En effet les signaux doivent être valables à tout moment. Chaque transition est significative dans ce genre de circuits, et par conséquent les aléas sont interdits. Considérons le cas d'une porte ET à deux entrées ; une transition ascendante de 0 à 1 indique bien que les deux entrées soient égales à 1. Par contre une transition descendante de la sortie – de 1 à 0 – ne contient pas suffisamment d'information sur l'état actuel de chacune des deux entrées. La seule information qu'elle fournie est qu'au moins une entrées est égale à 0.

Pour éviter cette situation, les concepteurs des circuits asynchrones, utilisent la porte la mieux adaptée à ce genre de fonctionnement : La porte de Muller, aussi connue sous le nom de « *C-Element* ».

La porte de Muller est une fonction qui copie le niveau logique des entrées sur la sortie, lorsque ceux-ci sont égaux. Vis-à-vis des transitions, la porte de Muller implémente un rendez-vous. Une transition est propagée en sortie si et seulement si une transition a eu lieu sur chacune des entrées. Cette fonction peut être définie pour un nombre d'entrées quelconque. La Figure 12 présente le symbole et la table de vérité d'une porte de Muller à 2 entrées.



a	a	y
0	0	0
0	1	y^{-1}
1	0	y^{-1}
1	1	1

Figure 12 : Symbole et table de verite du C-Element

L'implémentation d'une porte Muller dans un FPGA peut se faire de plusieurs façons. D'une manière générale, une porte Muller à « n-1 » entrées s'implémente dans une LUT (Look Up Tables) à « n » entrées (L. Fesquet, B. FOLCO, M. STEINER, M. RENAUDIN, 2006). La $n^{ème}$ entrées servira pour reboucler la sortie de la LUT sur elle-même comme le montre la Figure 13 .

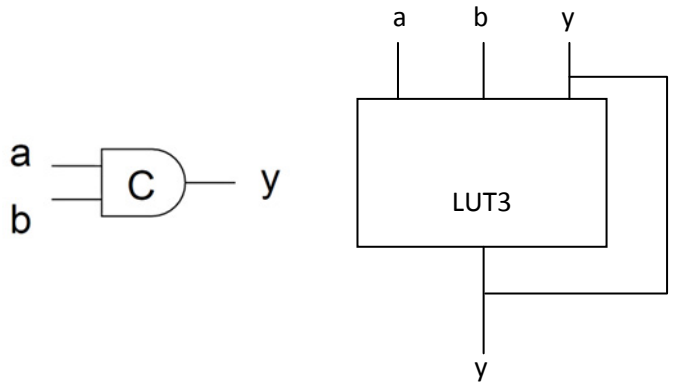


FIGURE 13 : IMPLEMENTATION D'UNE PORTE DE MULLER DANS UNE LUT A 3 ENTREES

3.1.1.3 IMPLEMENTATION D'UNE FONCTION A 2-ENTREES DOUBLE RAILS (1 PARMI 2)

3.1.1.3.1 LE SIGNAL DE "REQUETE"

Le protocole Handshake ou requête-acquittement est, comme son nom l'indique, un protocole qui consiste en une communication bidirectionnelle entre deux blocs. Ces derniers échangeront entre eux : un bus de données, un signal de requête et un signal d'acquittement.

En effet, un bloc est prêt à effectuer un calcul, quand ses entrées reçoivent une requête, et que tous les blocs connectés à ses sorties ont acquitté les précédentes valeurs des ses sorties. Dans le cas d'un codage en 1 parmi n, le signal de requête est codé dans le bus de données. De ce fait, deux blocs échangent entre eux un bus de données/requête et un signal d'acquittement seulement.

3.1.1.3.2 LE SIGNAL D'"ACQUITTEMENT"

Le signal d'acquittement, est codé sur un seul fil. Il est calculé par un XOR (ou XNOR) de toutes les sorties du bloc. En 4 phases, son rôle est de signaler l'arrivée d'une nouvelle donnée et d'informer si cette dernière est valide ou non. Le signal d'acquittement est envoyé ensuite au circuit qui alimente les entrées du bloc.

3.1.1.3.3 L'IMPLEMENTATION DE LA FONCTION A DEUX ENTREES.

L'implémentation de la fonction doit respecter les critères d'implémentation du protocole 4 phases double rails :

- Un signal S doit être codé sur 2 fils : s_0 et s_1.
- Un signal d'acquittement ack_{in} synchronisera la communication du circuit avec le bloc en aval.
- Un signal d'acquittement ack_{out} synchronisera la communication du circuit avec le bloc en amont.

Soit la fonction $f(x, y) : \mathbb{F}_2 \times \mathbb{F}_2 \rightarrow \mathbb{F}_2$ une fonction booléenne à deux variables. Sa sortie est un signal 1 parmi 2 représenté sur deux fils O_0 et O_1. Les fonctions $f^0(x, y)$ et $f^1(x, y)$ calculent respectivement les sorties O_0 et O_1. L'Équation 1 présente la forme des ces deux sorties, ainsi que celle de la sortie ack_{out} :

$$O_1 = \begin{cases} f^1(x,y) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (ack_{in} = 1) \\ O_1^{-1} & sinon \end{cases}$$

$$O_0 = \begin{cases} f^0(x,y) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (ack_{in} = 1) \\ O_0^{-1} & sinon \end{cases}$$

$$\mathrm{ack_{out}} = \mathrm{O_1} \oplus \mathrm{O_0}$$

ÉQUATION 1: SORTIES DE LA FONCTION A 2 ENTREES DOUBLE RAILS

Il est clair que chaque fonction possède :

- 6 entrées dont 5 primaires x_0, x_1, y_0, y_1, ack_{in} et un rebouclage O_0^{-1} ou O_1^{-1},
- et une sortie : O_0 ou O_1.

Ainsi l'implémentation de la sortie O requiert 2 LUTs[27] à 6 entrées et une sortie chacune : LUT6. Les deux LUT6 calculent respectivement les sorties O_0 et O_1.

L'effet mémoire imposé par l'Équation 1, est implémenté par le rebouclage de chacune des sorties O_0 et O_1 aux entrées des LUT6 correspondantes. Ceci dit la taille minimale d'une LUT6 nécessite 64 bits – ou points de programmation – pour implémenter toute fonction de

[27] Look Up Table

type 6-bit ↦ 1-bit. Comme il s'agit de deux sorties (cas de la fonction double-rail), la taille minimale du PLB est de 2 LUT6. A cela s'ajoute la porte XNOR qui calcule l'acquittement de sortie ack_{out}.

Les deux fonctions partagent les mêmes entrées : x_0, x_1, y_0, y_1, ack_{in} à l'exception des rebouclages. De ce fait les entrées des deux LUT6 peuvent être communes, sauf celles correspondant au rebouclage (cf. Figure 14). Cette mise en commun des entrées, permet dans le futur d'utiliser une seule « *connection-box* » dans le réseau d'interconnexion du FPGA divisant ainsi par deux la taille totale des « *connection-boxes* » du réseau d'interconnexion. La Figure 14 montre la structure minimale d'un bloc programmable PLB nécessaire pour implémenter une porte à deux entrées double rail. Les entrées étant les suivantes : 2 signaux X et Y codés en double rails (x_0, x_1) et (y_0, y_1) respectivement, 1 signal d'acquittement, et 2 rebouclages permettant de réaliser la mémorisation en cas d'entrées invalides : (x_0, x_1) = (0, 0) ou (y_0, y_1) = (0, 0). Le XNOR à la sortie permettra de calculer l'acquittement sortant.

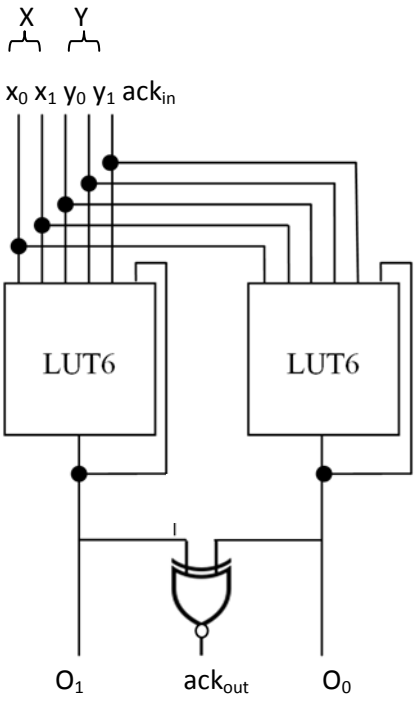


FIGURE 14 : LA TAILLE MINIMALE D'UN BLOC PROGRAMMABLE PLB

3.1.1.4 IMPLEMENTATION D'UNE FONCTION A 3 ENTREES DOUBLE RAILS (1 PARMI 2)

L'Équation 1 peut être facilement modifiée pour inclure une troisième entées z :

$$O_1 = \begin{cases} f^1(x,y,z) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (z \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (z = \Omega)\ et\ (ack_{in} = 1) \\ O_1^{-1} & sinon \end{cases}$$

$$O_0 = \begin{cases} f^0(x,y,z) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (z = \Omega)\ et\ (ack_{in} = 1) \\ O_0^{-1} & sinon \end{cases}$$

$$ack_{out} = \overline{O_1 \oplus O_0}$$

ÉQUATION 2: SORTIES DE LA FONCTION A 3 ENTREES DOUBLE RAILS

Comme le montre l'Équation 2, chaque sortie est une fonction de 8 variables : 3 correspondent aux entrées double-rail {x, y, z} , une à l'acquittement entrant ack_{in} et la dernière variable correspond au rebouclage ou mémorisation. Il est donc impossible de l'implémenter sur l'architecture du PLB de la Figure 14. Cette dernière étant adaptée pour l'implémentation des fonctions 6 ↦ 1 où 5 entrées sont primaires connectées au réseau d'interconnexion et une provenant du rebouclage interne. Toutefois, concaténer plusieurs LUT6 pour implémenter la fonction 8 ↦ 1 induit un coût supplémentaire au niveau du réseau d'interconnexion.

Ainsi, la première solution était d'augmenter la taille des LUT6 pour faire des LUT8. Ces dernières possèderont alors 8 entrées et donc 256 points de programmation chacune. L'implémentation de la fonction à 3 entrées coutera 512 points de programmation en utilisant ce type de LUT8. Cette solution n'a pas été retenue à cause de son coût important en points de programmation et donc en surface.

Une autre solution - celle adoptée - consiste à séparer le signal d'acquittement ack_{in} du reste des entrées et à modifier l'architecture interne du PLB, pour supporter les fonctions 7 ↦ 1 (Figure 15). L'acquittement sera ensuite pris en compte dans l'étage suivant.

La Figure 15 présente l'architecture du PLB nécessaire pour implémenter des fonctions 7 ↦ 1. Il s'agit de 4 LUT6, regroupées deux à deux. Chaque groupe possède deux sorties connectées à un multiplexeur, et calcule les sortie O_0 et O_1 respectivement. Comme le montre la Figure 15, chaque groupe de 2 LUT6 possède 6 entrées qui sont dans ce cas les 6 variables correspondants aux signaux x, y et z en double-rail. La septième variable de la fonction 7 ↦ 1 est celle de la mémorisation. Elle est calculée par le multiplexeur qui reboucle sur lui-même.

L'acquittement des sorties ack_{out} est calculé par une porte XNOR alimentée par les deux sorties du PLB. Cette implémentation nécessite 4x64 bits de programmation, donc deux fois moins que la première.

Une Remarque très importante : il y a une grande différence entre cette architecture de PLB et celle de la section précédente. La première différence est que dans cette architecture les sorties utilisées sont les sorties des multiplexeurs, et non pas les sorties directes des LUT6 comme c'était le cas dans l'architecture précédente. La deuxième différence, est que la porte XNOR n'est pas la même dans les deux architectures.

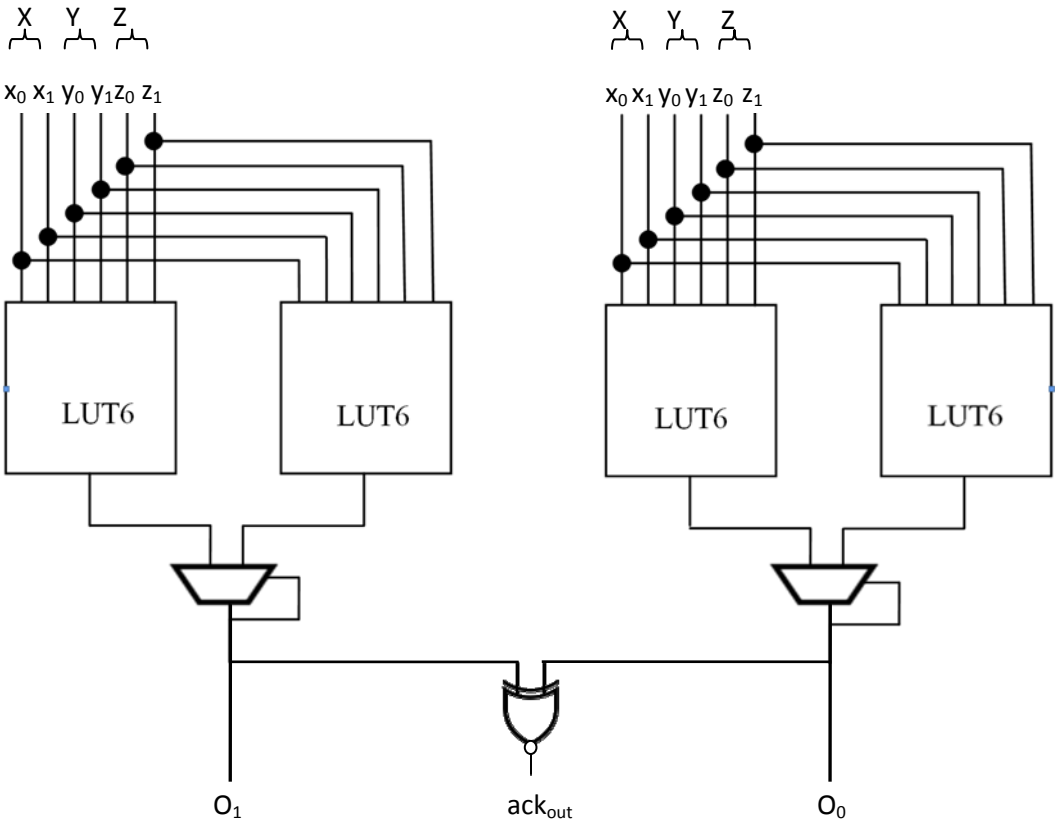


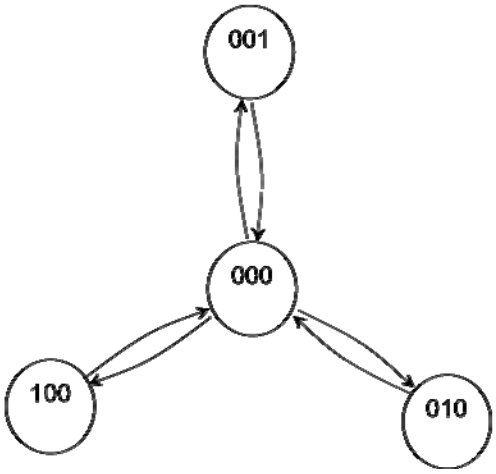
FIGURE 15 : ARCHITECTURE DU PLB MODIFIEE POUR SUPPORTER LES FONCTIONS 7 ↦ 1.

3.1.1.5 IMPLEMENTATION D'UNE FONCTION A 2 ENTREES TRIPLE RAILS (1 PARMI 3)

Dans la suite de cette partie, les données sont présentées en triple rails ou aussi appelé 1 parmi 3. Trois fils sont donc utilisés pour coder une donnée. Soit $\{x_2, x_1, x_0\}$ la représentation d'une donnée X en triple rails, l'utilisation du protocole 4 phases permet de distinguer les états suivants :

- $\{x_2, x_1, x_0\}$= {1, 0, 0} ou {0, 1, 0} ou {0, 0, 1} représentent les cas des données valides.
- $\{x_2, x_1, x_0\}$ = {0, 0, 0} représente l'absence de données.
- $\{x_2, x_1, x_0\}$ = {1, 1, 1} ou {0, 1, 1} ou {1, 0, 1} ou {1, 1, 0} sont des cas interdits, donc non utilisés.

La transition d'un état valide à un état valide n'est pas autorisée. Il faut nécessairement passer par l'état invalide comme le montre la Figure 16.



Donnée	X_2	X_1	X_0
Invalide - Ω	0	0	0
Etat Valide	0	0	1
Etat Valide	0	1	0
Etat Valide	1	0	0
Non utilisée	1	1	∀
Non utilisée	1	∀	1
Non utilisée	∀	1	1

FIGURE 16 : CODAGE TRIPLE RAILS (1 PARMI-3), 4 PHASES

Soit la l'équation suivante, elle présente les sorties O_2, O_1, O_0 comme suit :

$$O_2 = \begin{cases} f^2(x,y) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (ack_{in} = 1) \\ O_2^{-1} & sinon \end{cases}$$

$$O_1 = \begin{cases} f^1(x,y) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (ack_{in} = 1) \\ O_1^{-1} & sinon \end{cases}$$

$$O_0 = \begin{cases} f^0(x,y) & si\ (x \neq \Omega)\ et\ (y \neq \Omega)\ et\ (ack_{in} = 0) \\ 0 & si\ (x = \Omega)\ et\ (y = \Omega)\ et\ (ack_{in} = 1) \\ O_0^{-1} & sinon \end{cases}$$

$$ack_{out} = \overline{O_2 \oplus O_1 \oplus O_0}$$

D'après ce qui précède, chaque sortie est une fonction qui dépend de 8 variables dont 7 sont primaires : x_2, x_1, x_0, y_2, y_1, y_0, ack_{in} et un rebouclage. Comme dans le cas précédent, chaque sortie peut être implémentée sur deux LUT6, en utilisant la même architecture du PLB de Figure 15. Ainsi six LUT6 (autrement dit 1 PLB et un demi-PLB) sont nécessaires pour l'implémentation des trois sorties (Figure 17). Il est important de rappeler que le signal d'acquittement ack_{in} a été séparé des autres signaux d'entrées, comme dans le cas de fonctions à 3 entrées double-rails. Il sera pris en charge par le bloc en aval.

Le demi-PLB non utilisé servira à calculer l'acquittement des trois sorties.

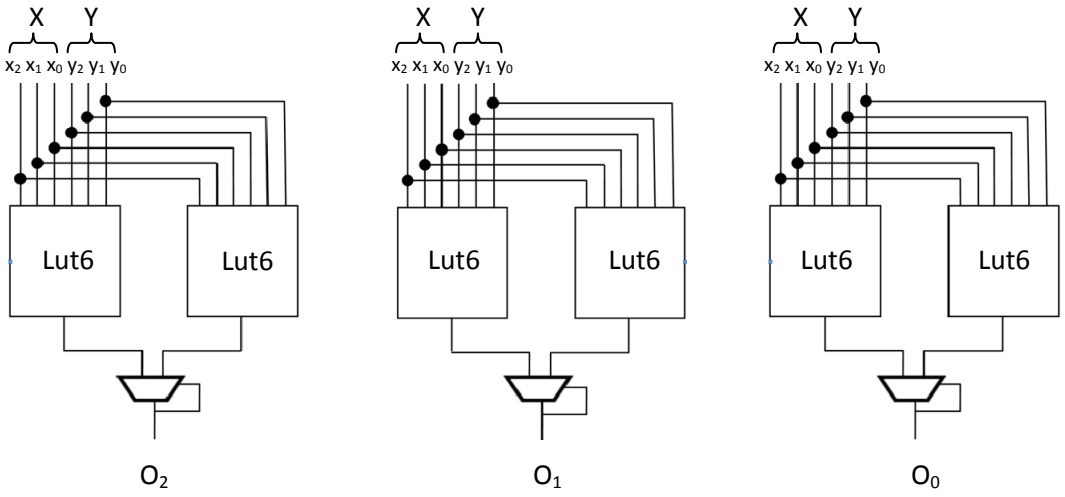


FIGURE 17 : IMPLEMENTATION D'UNE FONCTION A 2 ENTREES TRIPLE RAILS EN 4 PHASES

Quant au calcul du signal d'acquittement ack_{out}, il est fait en deux étapes :

- $ack = \overline{O_2 \oplus O_1}$, calculé par la porte XNOR qui connecte les sorties O_2 *et* O_1 .
- $ack_{out} = \overline{ack \oplus O_0}$. Pour calculer cette équation, une porte XNOR à 2-entrées suffira. Il est à noter que pour des raisons de flexibilité (implémentations des portes logiques en 2 phases), la porte XNOR est remplacée par une LUT2 (cf. Figure 19). Celle-ci peut être programmée pour faire une XNOR ou toute autre porte à 2-entrées.

Noter que les entrées de cette LUT2 sont les sorties des deux XNOR du PLB. Il faut alors bien programmer la quatrième LUT6 non utilisée pour que la XNOR concernée fasse une fonction identité (cf. Figure 18).

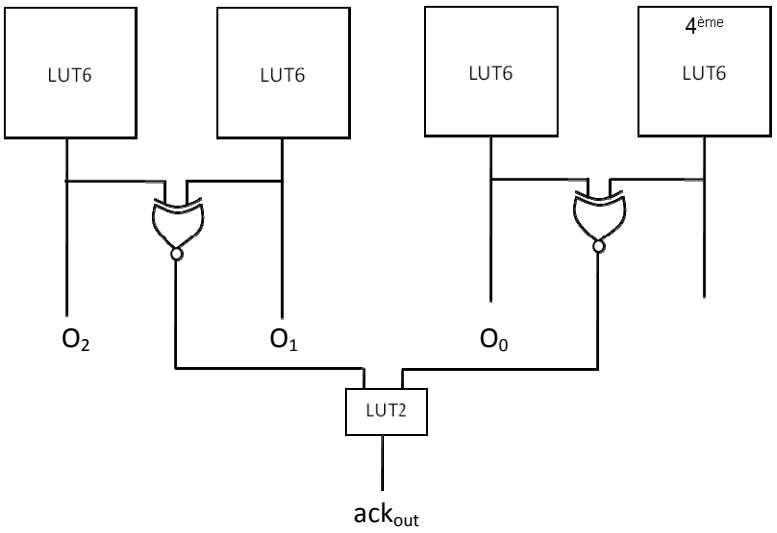


FIGURE 18 : CALCUL DU SIGNAL D'ACQUITTEMENT ACK_{OUT}

3.1.1.6 IMPLEMENTATION D'UNE FONCTION A N ENTREES, N RAILS (CODAGE EN 1 PARMI N)

La généralisation de travail en 1 parmi n est très compliquée. En effet, l'implémentation nécessite soit une augmentation de la tailles des LUTs soit une division de la fonction en sous fonctions plus petites. Dans les deux cas, il apparaît que l'implémentation occupera un grand nombre de PLBs, sans qu'il y ait un vrai avantage des codages en 1 parmi n par rapport à leurs concurrents « 1 parmi 2 » et « 1 parmi 3 ».

3.1.1.7 CONCLUSION SUR L'IMPLEMENTATION DU PROTOCOLE 4 PHASES

L'implémentation des fonctions à 2 et 3 entrées en 4 phases nécessite l'utilisation d'une LUT de taille minimale de 64 bits. C'est une LUT6 à 6 entrées et une seule sortie. Le PLB doit être composé de 4 LUT6 minimum pour implémenter les fonctions à 3 entrées. Des multiplexeurs et des portes XNOR sont aussi nécessaires pour réaliser l'effet mémoire et le signal ack_{out} respectivement.

L'architecture de la Figure 19, est la synthèse des deux architectures présentées dans les sections 3.1.2.4 et 3.1.2.3. C'est un PLB qui possède 12 entrées et 10 sorties. Il permet d'implémenter toute fonction ayant les propriétés suivantes :

- Les entrées dépendent de 8 signaux ou moins : au maximum 7 sont primaires venant du réseau d'interconnexion dont un est le signal d'acquittement ack_{in}, et un signal de rebouclage pour réaliser l'effet mémoire.

- Les sorties ne dépassent pas 4 sorties. Il ne faut pas compter ici le signal d'acquittement ack_{out}

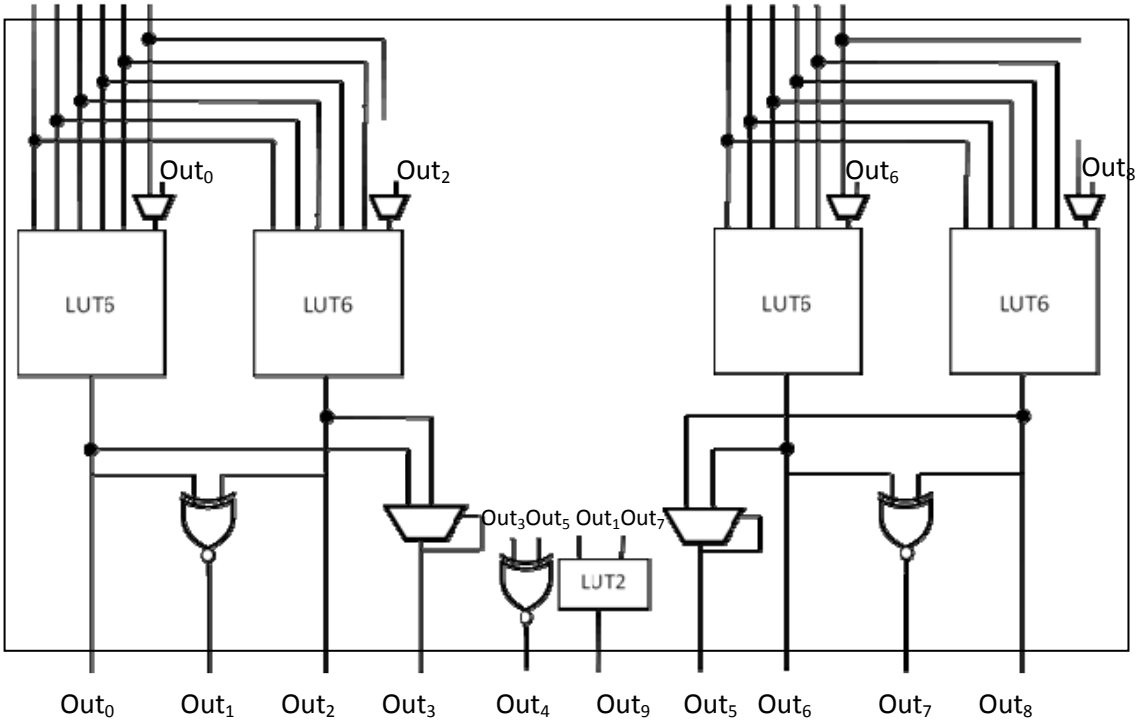


FIGURE 19 : ARCHITECTURE DU PLB POUR L'IMPLEMENTATION DU PROTOCOLE 4 PHASES

Les multiplexeurs qui se trouvent aux entrées des LUT6, permettent de basculer entre deux cas : un rebouclage et une entrée extérieure. Le rebouclage sert à réaliser l'effet mémoire dont l'implémentation des fonctions à 2 entrées double rails a besoin. L'autre cas est utilisé pour implémenter les fonctions à 3 entrées ou plus. L'ajout de ce multiplexeur va bien sûr déséquilibrer les entrées du point de vue temps de propagation. Ce problème sera résolu plus loin dans ce manuscrit.

La taille minimale de l'architecture d'un PLB permet d'implémenter des fonctions de petites et moyennes tailles tout en minimisant le coût des interconnexions et des points de programmation. Pour implémenter des fonctions plus grandes que celles présentées, il suffit de les diviser en sous-fonctions plus petites implantables sur l'architecture adoptée.

3.1.2 IMPLEMENTATION DU PROTOCOLE 2 PHASES LEDR[28]

3.1.2.1 PRINCIPE DU PROTOCOLE 2 PHASES LEDR

3.1.2.1.1 LE PROTOCOLE 2 PHASES

Comme son nom l'indique, le protocole 2 phases possède une séquence d'échange de données plus réduite que celle du 4 phases. Un cycle de communication se fait en deux étapes au lieu de quatre. Le protocole 2 phases, est aussi appelé NRZ pour « *non-return-to-zero* » ou « *level signaling* ». Les requêtes et acquittements sont encodés en tant que transitions sur les fils et il n'y a pas de différence entre les transitions $0 \mapsto 1$ et $1 \mapsto 0$. Chacune représente un nouveau ''évènement'' sur le signal concerné. Théoriquement ce comportement doit amener à de circuits plus rapides que ceux du protocole 4 phases. Cependant ce n'est pas toujours le cas, car l'implémentation des circuits dont la communication est basée sur les évènements est compliquée. Il est donc toujours difficile de décider lequel des deux protocoles est le meilleur.

Généralement en deux phases double rails deux fils sont utilisés $\{d_t, d_f\}$ pour présenter un bit de donnée. Comme il a été expliqué précédemment, l'information est codée comme une transition. Un événement sur le fil d_f signifie que le nouveau bit est un '0' logique, tandis qu'un événement sur le fil d_t signale l'arrivée d'un '1' logique. Sur un bus de N-bit, un nouveau mot est reçu lorsqu'un événement arrive sur un et un seul fil. L'état « absence de données » n'existe pas (cf. Figure 20). Une donnée valide est acquittée et suivie par une autre donnée valide qui a son tour sera acquittée. La Figure 20 présente le comportement d'un bus de 2 bits en 2 phases double rails.

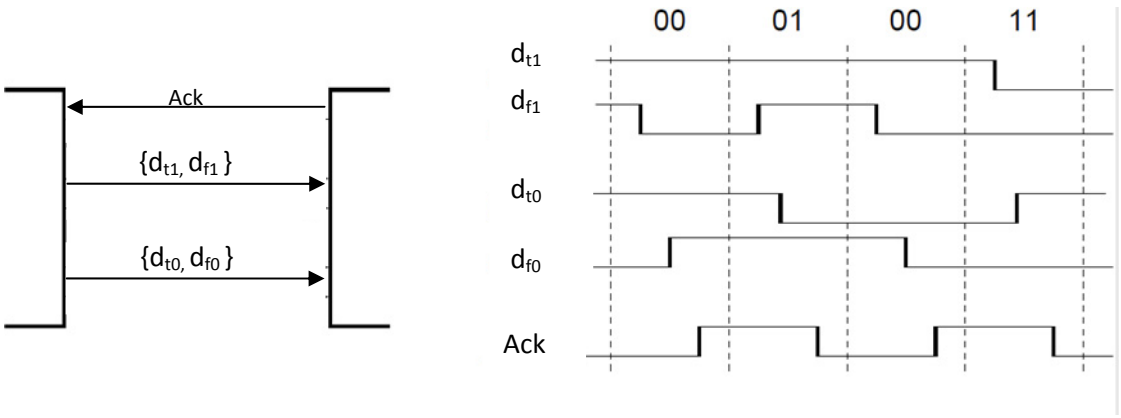


FIGURE 20 : PROTOCOLE 2 PHASES

[28] Level Encoded Dual Rail

3.1.2.1.2 Le protocole 2 phases LEDR

Le protocole 2 phases LEDR est une sous-famille du protocole 2 phases (D. H. Linder, J. C. Harden, 1996). Dans ce protocole, chaque variable est représentée par deux fils :

- Le premier dénommé V_d pour « donnée », indique par ses transitions montantes l'arrivée d'un '1' logique et par ses transitions descendantes celle d'un '0' logique.
- Le deuxième dénommé V_r pour « répétition », indique par ses transitions, montantes ou descendantes, l'arrivée d'une nouvelle valeur identique à la précédente.

La Figure 21 montre un exemple de trafic obéissant à ce protocole. Contrairement aux protocoles 4 phases et 2 phases sur front (cf. section 3.1.3), le protocole LEDR ne peut être utilisé qu'avec une représentation en 1 parmi 2.

En effet, l'arrivée d'une donnée différente de la donnée précédente se traduit par un changement du bus de données codé sur n bits. Ce changement se traduit par le basculement simultané de tous ces signaux. Or un changement simultané de plusieurs signaux serait potentiellement générateur de *glitches* inacceptables dans un circuit asynchrone

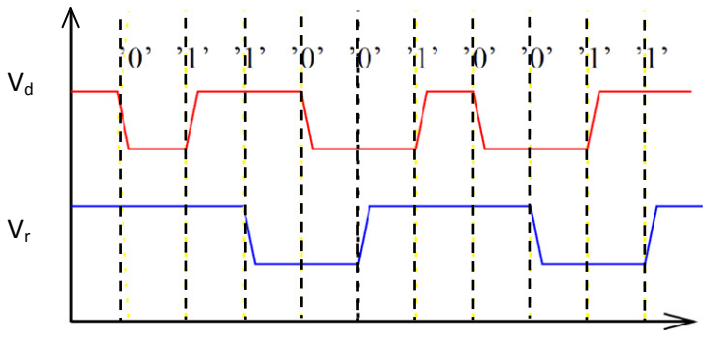


FIGURE 21 : PROTOCOLE 2 PHASES LEDR.

3.1.2.2 MODELE DE CIRCUIT – ELEMENTS DE BASE

3.1.2.2.1 Détection de l'arrivée des données.

Lors de l'initialisation globale du système, tous les fils des données X_d et X_r sont mises à '0', tandis que toutes les sorties de synchronisation sortantes (ack_{out}) sont mises à '1'. L'arrivée d'une donnée nouvelle sur une ligneV se traduit par le basculement $0 \mapsto 1$ et $1 \mapsto 0$ d'une et

une seule des lignes V_d ou V_r. La détection unifiée de ces deux évènements peut être réalisée en surveillant l'expression $V_d \oplus V_r$.

3.1.2.2.2 Décision de calcul.

Du fait de l'inversion de la valeur de $X_d \oplus X_r$ à chaque arrivée d'une nouvelle valeur sur la ligne logique X et de l'initialisation de toutes les lignes de données à (0, 0), une porte d'entrées A_0,. . . ,A_p a effectué son calcul et, de ce fait, a consommé les valeurs présentes à ses entrées lorsque :

$$A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = O_d \oplus O_r \qquad \textbf{Équation 3}$$

De même, la porte est prête à effectuer son calcul lorsque :

$$A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p \neq O_d \oplus O_r \qquad \textbf{Équation 4}$$

D'autre part, comme la valeur logique de la paire (A_d, A_r) est égale à la valeur de la ligne A_d, les équations des sorties de la porte sont :

$$O_d = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 0\ et\ O_d \oplus O_r = 1 \\ f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 1\ et\ O_d \oplus O_r = 0 \\ O_d^{-1} & sinon \end{cases}$$

$$O_d = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 0\ et\ O_d \oplus O_r = 1 \\ \overline{f(A_d^0, A_d^1, \dots, A_d^p)} & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 1\ et\ O_d \oplus O_r = 0 \\ O_d^{-1} & sinon \end{cases}$$

En effet, si, avant le calcul, les sorties de la porte étaient telles que $O_d = O_r$, alors à l'issue du calcul, on doit avoir, $O_d \neq O_r$ pour signaler l'arrivée d'une nouvelle donnée aux portes située en aval. L'expression : $O_d = f(A_d^0, A_d^1, \dots, A_d^p)$ et $O_r = \overline{f(A_d^0, A_d^1, \dots, A_d^p)}$ garantit que cette condition sera remplie.

3.1.2.2.3 Synchronisation du calcul

La détection de l'arrivée des données n'est pas nécessairement une condition suffisante pour déclencher le calcul de $f(A_d^0, A_d^1, \dots, A_d^p)$. En effet, les portes dont une entrée est connectée à la sortie de la porte prête à calculer peuvent ne pas avoir encore utilisé l'ancienne valeur, qui va être détruite par le nouveau calcul.

Les équations 3 et 4 permettent à une porte de signaler aux portes en amont qu'elle a consommé les valeurs présentes à ses entrées en leur transmettant la valeur

$$ack_{out} = \overline{O_d \oplus O_r}\ .$$

Si une porte P a sa sortie connectée à n portes S^1, S^2,…, S^n, le rendez-vous des $ack_{out}^1, \dots, ack_{out}^n$ transmis par les portes en aval indique à P si toutes les portes suivantes sont prêtes à accepter une nouvelle valeur.

La porte P sera donc prête à calculer lorsque les conditions suivantes seront remplies :

1. $A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p$
2. $A_d^0 \oplus A_r^0 \neq O_d \oplus O_r$ et
3. $O_d \oplus O_r = ack_{in}$

Enfin, du fait de la propagation des parités, la condition 2 est une conséquence de la condition 3. Les équations des sorties des portes avec synchronisation deviennent donc :

$$O_d = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 0\ et\ ack_{in} = 1 \\ f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 1\ et\ ack_{in} = 0 \\ O_d^{-1} & sinon \end{cases}$$

$$O_r = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 0\ et\ ack_{in} = 1 \\ \overline{f(A_d^0, A_d^1, \dots, A_d^p)} & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = 1\ et\ ack_{in} = 0 \\ O_r^{-1} & sinon \end{cases}$$

Afin de garantir une compatibilité de la structure avec les portes en 4 phases, une logique complémentaire a été introduite pour les signaux de synchronisation. Ainsi le signal d'acquittement sortant est : $ack_{out} = \overline{O_d \oplus O_r}$

Les équations de la sorties deviennent :

$$O_d = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = \overline{ack_{in}} \\ O_d^{-1} & sinon \end{cases}$$

$$O_r = \begin{cases} f(A_d^0, A_d^1, \dots, A_d^p) & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = \overline{ack_{in}} = 0 \\ \overline{f(A_d^0, A_d^1, \dots, A_d^p)} & si\ A_d^0 \oplus A_r^0 = A_d^1 \oplus A_r^1 = A_d^2 \oplus A_r^2 = \cdots = A_d^p \oplus A_r^p = \overline{ack_{in}} = 1 \\ O_r^{-1} & sinon \end{cases}$$

$$ack_{out} = \overline{O_d \oplus O_r}$$

ÉQUATION 5: SORTIES D'UNE FONCTION A P ENTREES LEDR

3.1.2.3 IMPLEMENTATION D'UNE FONCTION A 2 ENTREES

Soit une fonction F à 2 entrées A et B ; d'après l'Équation 5 ses sorties O_d et O_r s'écrivent :

$$O_d = \begin{cases} f(A_d, B_d) & si\ A_d \oplus A_r = B_d \oplus B_r = \overline{ack_{in}} \\ O_d^{-1} & sinon \end{cases}$$

$$O_r = \begin{cases} f(A_d, B_d) & si\ A_d \oplus A_r = B_d \oplus B_r = \overline{ack_{in}} = 0 \\ \overline{f(A_d, B_d)} & si\ A_d \oplus A_r = B_d \oplus B_r = \overline{ack_{in}} = 1 \\ O_r^{-1} & sinon \end{cases}$$

$$ack_{out} = \overline{O_d \oplus O_r}$$

ÉQUATION 6: SORTIES DE LA FONCTION A 2 ENTREES LEDR

Ces équations montrent que :

- O_d est une fonction des 6 variables : A_d , A_r , B_d , B_r , $\overline{ack_{in}}$ et O_d^{-1} et que
- O_r est une fonction des 6 variables : A_d , A_r , B_d , B_r , $\overline{ack_{in}}$ et O_r^{-1}

Chacune de ces expressions comporte six termes, elle est donc implantable dans la même architecture définie dans la section 3.1.1.7 (cf. Figure 19). Son implémentation nécessite 2 LUT6 comme le montre la Figure 22

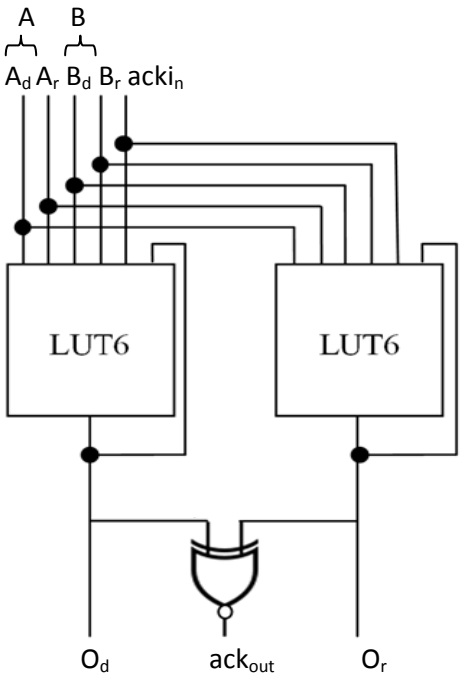


FIGURE 22 : IMPLEMENTATION D'UNE FONCTION A 2 ENTREES LEDR

3.1.2.4 IMPLEMENTATION D'UNE FONCTION A 3-ENTREES

Les équations de sorties d'une fonction à 3 entrées s'écrivent :

$$O_d = \begin{cases} f(A_d, B_d, C_d) & si\ A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = \overline{ack_{in}} \\ O_d^{-1} & sinon \end{cases}$$

$$O_r = \begin{cases} f(A_d, B_d, C_d) & si\ A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = \overline{ack_{in}} = 0 \\ \overline{f(A_d, B_d, C_d)} & si\ A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = \overline{ack_{in}} = 1 \\ O_r^{-1} & sinon \end{cases}$$

$$ack_{out} = \overline{O_d \oplus O_r}$$

ÉQUATION 7: SORTIES DE LA FONCTION A 3 ENTREES LEDR

La Figure 23 donne un schéma possible d'implémentation d'une fonction à 3 entrées LEDR, en se basant toujours sur la même architecture du PLB définie dans la section 3.1.1.7 (cf. Figure 19), avec de petites modifications aux niveaux des entrées. Cependant, les trois entrées ne sont pas équivalentes, pour ce qui est du temps de propagation, tout au moins.

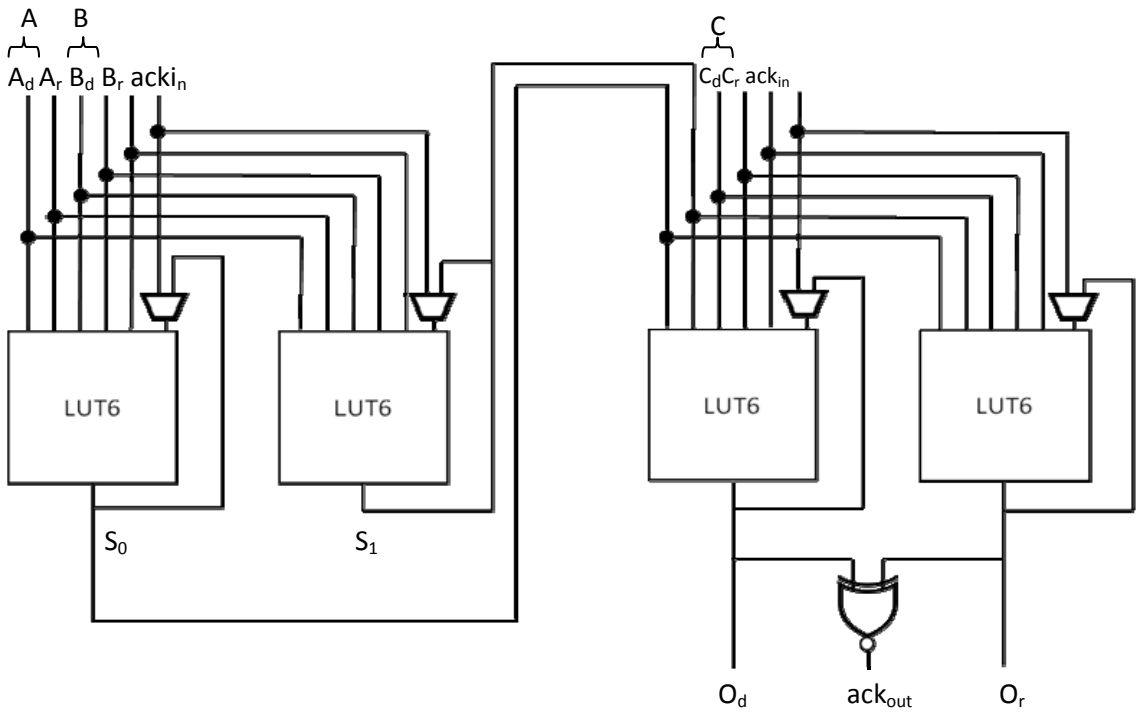


FIGURE 23 : IMPLEMENTATION D'UNE FONCTION A 3 ENTREES NON EQUIVALENTES EN LEDR

3.1.2.4.1 IMPLEMENTATION AVEC EQUIVALENCE DES ENTREES

Pour que les trois entrées soient indiscernables de l'extérieur du point de vu du temps de propagation, la porte doit être implémentée en deux étages donc 2 PLB. C Le premier couple des LUT6 détecte les parités des trois entrées de la porte (voir Figure 24).

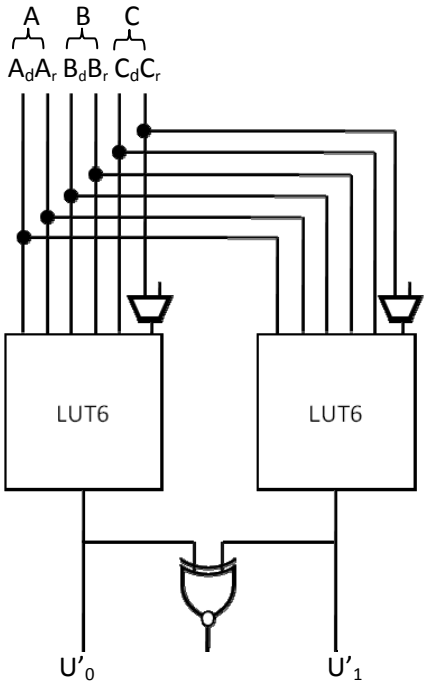


FIGURE 24 : LE PREMIER ETAGE : BLOC DE CALCUL DE PARITE

$$U'_0 = \begin{cases} 1 & si \quad A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = 0 \\ 0 & sinon \end{cases}$$

Équation 8

$$U'_1 = \begin{cases} 1 & si \quad A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = 1 \\ 0 & sinon \end{cases}$$

Il est à noter que les fils $A_r, B_r \; et \; C_r$ n'interviennent plus dans le second étage. En effet, le changement d'une des entrées $A_d, B_d \; et \; C_d$ peut arriver au second étage de calcul avant que le résultat du premier étage ne vienne bloquer le calcul. Il y a là un risque potentiel de disfonctionnement !

Pour remédier à cela il faut introduire dans le premier étage les variables de synchronisation $\mathrm{ack_{in}}$ et $\overline{\mathrm{O_d} \oplus \mathrm{O_r}}$ (voir Figure 25), et les équations des sorties du premier étages deviennent :

$$U_0 = U_0' \wedge \quad (O_d \oplus O_r = 0\,) \wedge \overline{ack_{in}} = 0$$

Équation 9

$$U_1 = U_1' \wedge \quad (O_d \oplus O_r = 1\,) \wedge \overline{ack_{in}} = 1$$

De cette façon, dès que le calcul d'une nouvelle valeur est effectué, la parité des sorties change et le dysfonctionnement évoqué précédemment devient impossible.

Remarque : Il existe ici une apparente contradiction entre la nécessité d'introduire les sorties de la porte dans la condition de départ du calcul (cf. Équation 9) et le fait que la condition sur ack_{in} est formellement suffisante pour autoriser le calcul (cf. Équation 5). En fait, il faut tenir compte du retard introduit dans la propagation des signaux par la traversée d'une LUT6 et interdire tout changement des signaux de sortie dès que le calcul est terminé alors que le signal ack_{in} changera beaucoup plus tard, c. à d. lorsque toutes les portes en aval auront utilisé la valeur qui vient d'être calculée. Le problème ne se pose pas dans le cas de la porte à deux entrées du fait de l'absence des signaux intermédiaires U_0 et U_1.

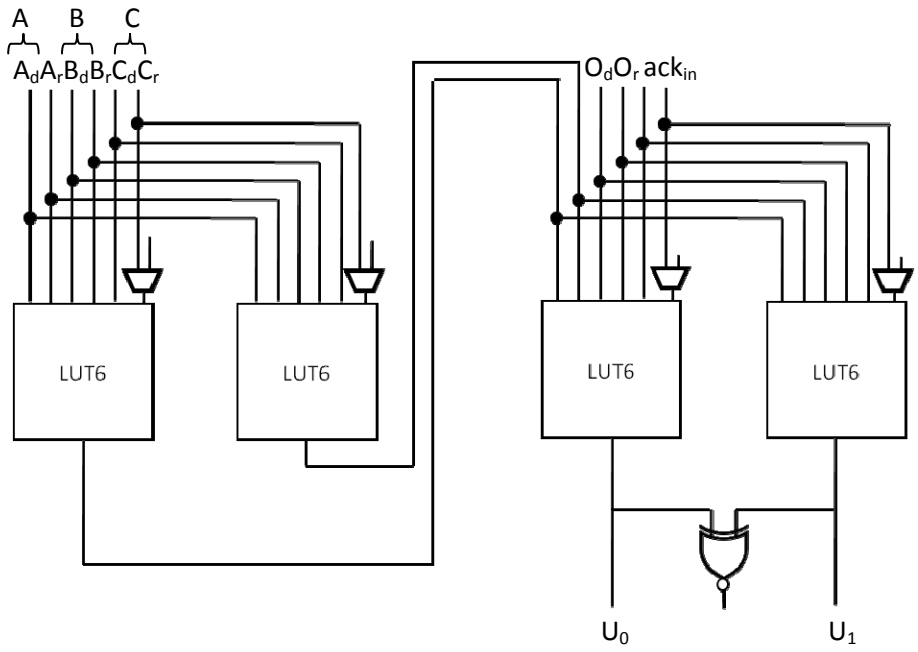


FIGURE 25 : LE PREMIER ETAGE MODIFIE : BLOC DE CALCUL DE PARITE

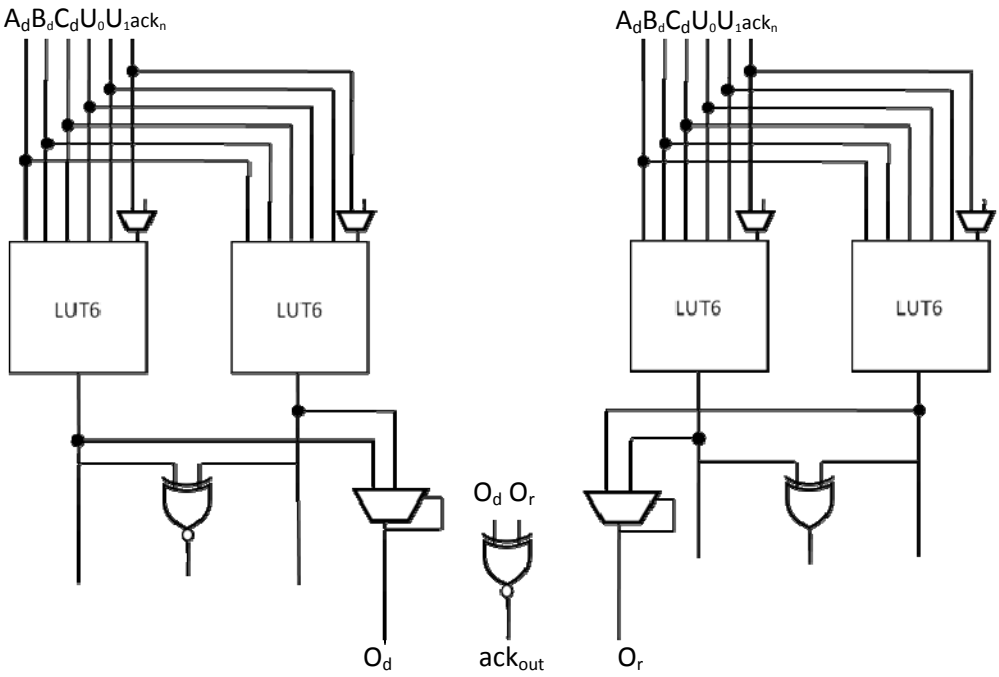


FIGURE 26 : LE DEUXIEME ETAGE : LES SORTIES DE LA FONCTION 2 ENTREES LEDR

L'étude dans ce manuscrit s'arrête sur les fonctions à trois entrées, à cause de la complexité du problème.

3.1.2.5 CONCLUSION SUR L'IMPLEMENTATION DU PROTOCOLE 2 PHASES LEDR

Les implémentations des fonctions à 2 et 3 entrées en 2 phases LEDR ont besoin des mêmes ressources en nombre de PLB que celles en 4 phases. Par contre, un circuit doit être théoriquement plus rapide et moins consommant en 2 phases qu'en 4 phases à cause de l'absence du retour à zéro en 2 phases. Ces caractéristiques pourront être vérifiées par simulation et après la fabrication du FPGA.

Avec une architecture identique à celle du PLB utilisé en 4 phases, les fonctions en 2 phases LEDR ont été implémentées. Une modification a été introduite aux niveaux des entrées pour permettre de connecter les sorties d'un couple de LUT6 aux entrées d'un autre couple LUT6 du même PLB. Cela est réalisé à l'aide de multiplexeurs comme le montre la Figure 27 de la nouvelle version du PLB.

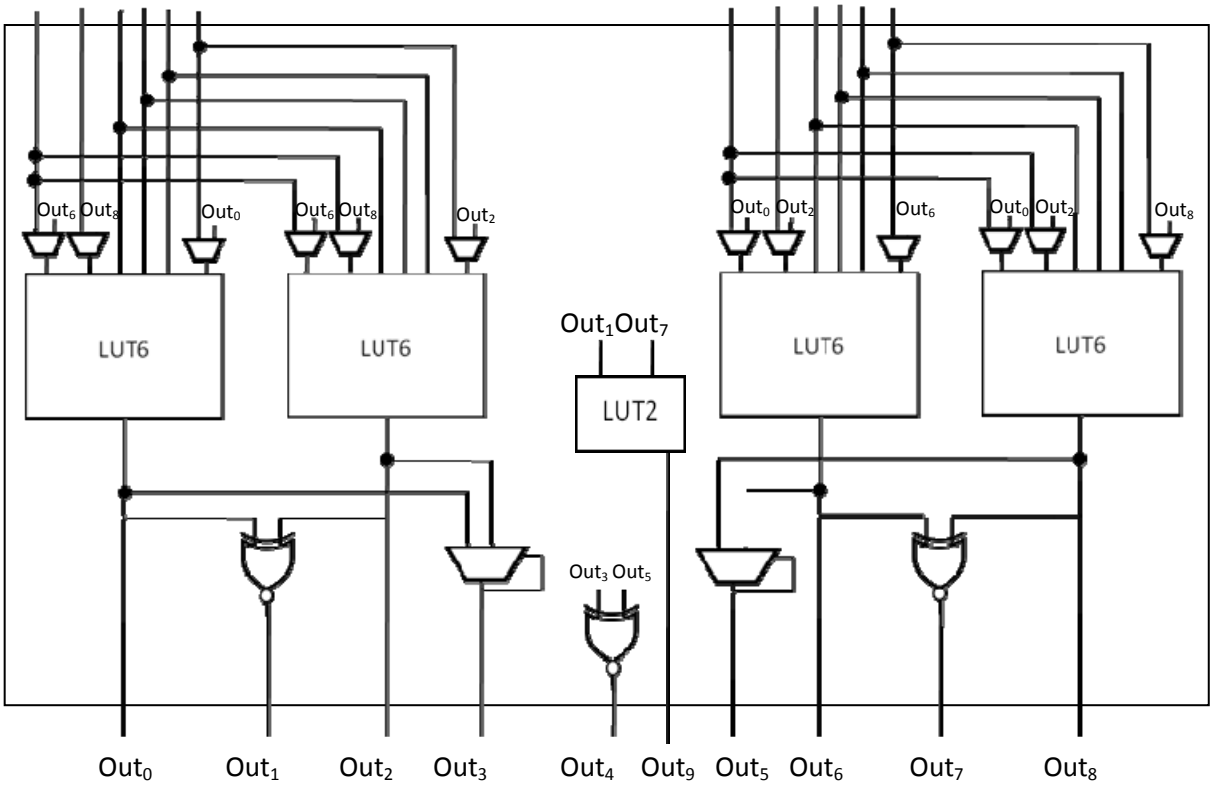


FIGURE 27 : NOUVELLE ARCHITECTURE DU PLB POUR L'IMPLEMENTATION DU 4 PHASES ET DU 2 PHASES LEDR

L'ajout des multiplexeurs sur les entrées des LUT6 induit - comme il est expliqué précédemment - des délais sur ces entrées. Ce déséquilibre peut d'être une source de fuite d'information qui impacte la sécurité du bloc. Il est résolu plus tard dans ce manuscrit.

3.1.3 IMPLEMENTATION DU PROTOCOLE 2 PHASES SUR FRONT

3.1.3.1 PRINCIPE DU PROTOCOLE 2 PHASES SUR FRONT

Dans ce protocole chaque variable est représentée par 'n' signaux électriques. Chacun de ces signaux peut se trouver dans deux états électriques : V_{ss} et V_{dd} . L'arrivée d'une valeur i, $i \in [0, n-1]$, est signalée par un changement d'état du fil 'i' de V_{ss} à V_{dd} ou inversement. Seule l'occurrence d'une transition est significative mais, en aucun cas, son type montant ou descendant. Cette caractéristique engendre naturellement une résistance intrinsèque aux attaques par mesure de la consommation électrique.

Les variables binaires sont représentées par deux fils, les variables ternaires par 3 fils… Cependant la complexité des portes en terme de nombre de fils par variables est quadratique. Ainsi, l'utilisation de ce protocole est en pratique limitée aux signaux binaires. Cette complexi-

té est aussi quadratique avec le nombre des entrées. Une fois encore, l'utilisation de ce protocole se restreint aux fonctions à 2 entrées. Dans ce qui suit, les signaux sont des signaux binaires. Un signal X est représenté par deux fils (x_0, x_1).

La Figure 28 donne un exemple de trafic sur une ligne gérée suivant ce protocole.

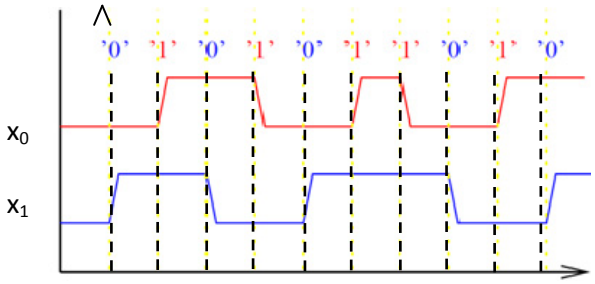


FIGURE 28 : PROTOCOLE DE TRANSMISSION DE DONNEES A 2 PHASES SUR FRONT.

3.1.3.2 MODELE DE CIRCUIT – IMPLEMENTATION D'UNE FONCTION A 2 ENTREES 1 PARMI 2

Le schéma général d'une porte à deux entrées en logique 2 phases sur front en 1 parmi 2, est représenté sur la Figure 29.

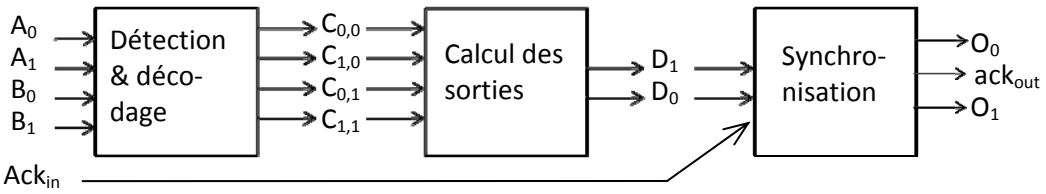


FIGURE 29 : SCHEMA GENERAL D'UNE PORTE A DEUX ENTREES N LOGIQUE 2 PHASES SUR FRONT

Son activité se compose de trois étapes bien distinctes :

- *Détection et décodage* : cette étape se charge de la reconnaissance de l'arrivée de nouvelles valeurs sur les entrées ainsi que du décodage de ces entrées. Cette étape est réalisée par un bloc appelé *Decision-wait* 2x2 « D-W2x2 ».
- *Calcul des sorties* : cette étape calcul les sorties de la porte à partir du résultat du décodage des entrées. C'est la partie combinatoire de la porte.
- *La synchronisation* : Elle assure la synchronisation du flot de données de la porte avec son entourage. Elle est assurée par un D-W2x1.

3.1.3.2.1 DECISION-WAIT 2X2

Ce circuit se charge de la détection de l'arrivée d'un front sur :

- A_1 ou A_0 d'un part et
- B_1 ou B_0 d'autre part.

Lorsqu'un front est arrivé sur le fil A_i et le fil B_j, un front est généré sur le fil $C_{i,j}$.

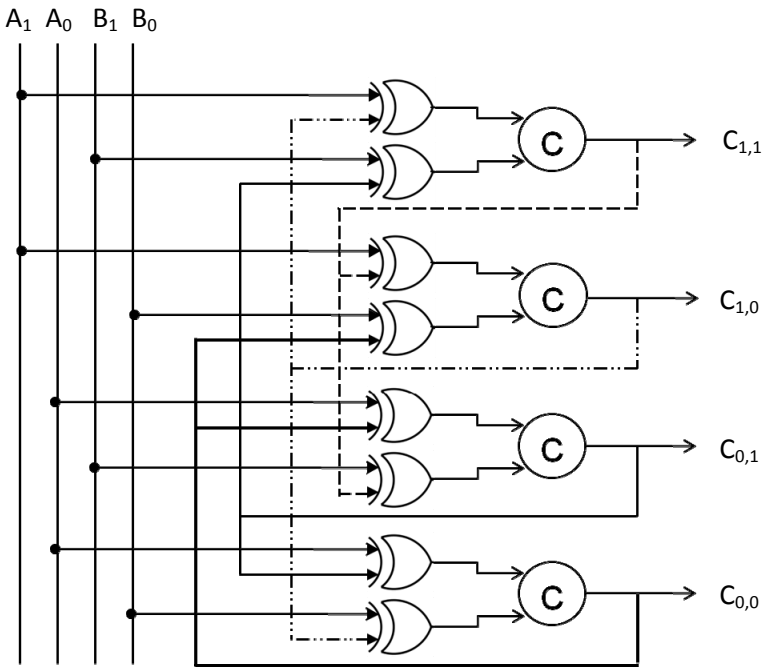


FIGURE 30 : BLOC DECISION-WAIT 2X2 A BASE DE PORTES DE MULLER

La Figure 30 présente un circuit qui réalise la fonction *Decision-Wait* à base de portes de Muller. Lorsque le circuit est prêt à recevoir des signaux sur ses entrées, les entrées de chacun des C-éléments sont égales, ainsi que la valeur de la sortie.

Cette condition est vérifiée juste après l'initialisation qui force chaque entrée à zéro, ainsi que la sortie de chaque C-élément.

Lorsqu'un front est détecté sur le fil A_i, une des entrées des C-éléments $C_{i,0}$ et $C_{i,1}$ s'inverse mais, du fait qu'aucun signal n'est arrivé sur les fils B_1 et B_0, aucune sortie ne change. Lorsqu'un front est ensuite détecté sur le fil B_j, une des entrées de chacun des C-éléments $C_{1,j}$ et $C_{0,j}$ bascule. A cet instant, le C-élément $C_{i,j}$ a vu ses deux entrées basculer.

Le changement d'état de $C_{i,j}$ génère un front sur le fil de sortie correspondant et ce signal vient également annuler les changements des entrées sur les C-élément $C_{i,\bar{j}}$ *et* $C_{\bar{i},j}$. A ce stade, l'hypothèse de départ - où tous les C-éléments ont leurs entrées égales à leur sortie - est de nouveau vérifiée.

La structure de ce circuit est parfaitement régulière, avec quatre blocs identiques, composés de deux portes XOR et d'un C-élément. L'équation de chacune des sorties de D-W2x2 est décrite par l'équation ci-dessous.

$$C_{i,j} = \mathrm{C}\left(A_i \oplus C_{i,\bar{j}}, B_j \oplus C_{\bar{i},j}, C_{i,j}^{-1}\right) \qquad avec\ (i,j) \in \{(0,0),(0,1),(1,0),(1,1)\}$$

ÉQUATION 10 : ÉQUATIONS DES SORTIES DU D-W2X2

L'Équation 10 montre que chaque $C_{i,j}$ est une fonction de cinq variables : 2 externes (A_i *et* B_j) et 3 rebouclages ($C_{\bar{i},j}$, $C_{i,\bar{j}}$ *et* $C_{i,j}$). Une telle équation est implantable sans problème dans une LUT6. Le PLB de la section 3.1.2.5, devra être capable, avec quelques améliorations, d'implémenter un D-W2x2 complet. Ces modifications concernent les trois rebouclages $C_{\bar{i},j}$, $C_{i,\bar{j}}$ *et* $C_{i,j}$. En effet, le fait que ces rebouclages se fassent à l'intérieur d'un PLB est un grand avantage car ils ne passent pas par le réseau d'interconnexion du FPGA :

- Il consomme moins de ressources d'interconnexion du FPGA et donc moins de courant
- Les temps de propagation de chacun des rebouclages vont être différents, si les rebouclages passent à travers le réseau d'interconnexion du FPGA ; ce qui pourra être plus tard un défaut de sécurité
- L'implémentation du D-W2x2, nécessitera toujours 2 PLBs, même si le rebouclage est fait à l'extérieur.

Pour toutes ces raisons, des multiplexeurs ont été rajoutés aux entrées des LUT6 du PLB. Ainsi la nouvelle architecture du PLB devient celle de la Figure 31. Les équations de sorties des 4 LUT6 du PLB s'écrivent comme suit – noter que les multiplexeurs sont désignés par le symbole [/]:

$$Out_0 = LUT\ (\ [Out_0 / I_0\ ], [\ Out_2 / I_1\ ], [\ Out_6 / I_2\ ], [\ Out_8 / I_3\ ], I_4\ , I_5\ )$$

$$Out_2 = LUT\ (\ [Out_2 / I_0\ ], [\ Out_0 / I_1\ ], [\ Out_6 / I_2\ ], [\ Out_8 / I_3\ ], I_4\ , I_5\ )$$

$$Out_6 = LUT\ (\ [Out_6 / I'_0\ ], [\ Out_8 / I'_1\ ], [\ Out_0 / I'_2\ ], [\ Out_2 / I'_3\ ], I'_4\ , I'_5\ )$$

$$Out_8 = LUT\ (\ [Out_8 / I'_0\ ], [\ Out_6 / I'_1\ ], [\ Out_0 / I'_2\ ], [\ Out_2 / I'_3\ ], I'_4\ , I'_5\ )$$

ÉQUATION 11 : ÉQUATIONS DE SORTIE DES 4 LUT6 DU PLB

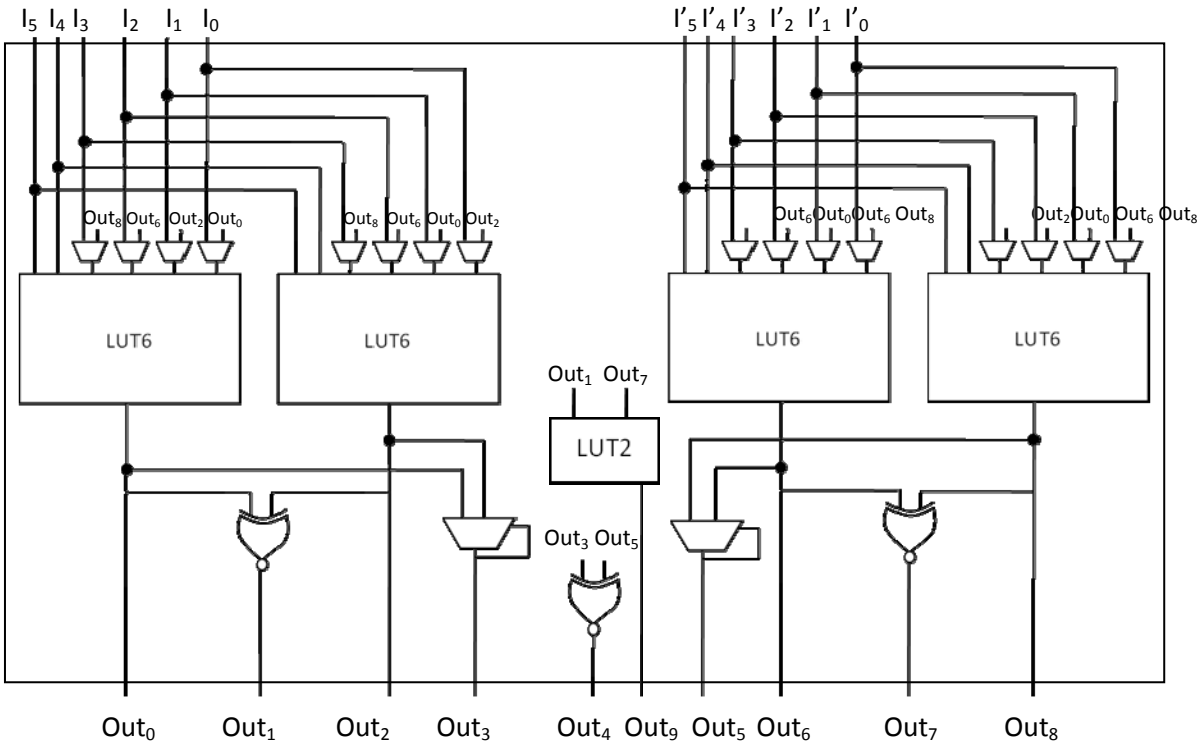


FIGURE 31: NOUVELLE ARCHITECTURE DU PLB

Pour implémenter le D-W2x2 sur le nouveau PLB, les entrées seront attribuées de la façon suivante :

$$I_0 \rightarrow NC\ ;\ I_1 \rightarrow NC\ ;\ I_2 \rightarrow B_1\ ;\ I_3 \rightarrow B_0\ ;\ I_4 \rightarrow A_0\ ;\ I_5 \rightarrow A_1$$

$$I'_0 \rightarrow NC\ ;\ I'_1 \rightarrow NC\ ;\ I'_2 \rightarrow B_1\ ;\ I'_3 \rightarrow B_0\ ;\ I'_4 \rightarrow A_0\ ;\ I'_5 \rightarrow A_1$$

Où *NC* signifie non connectée. Les équations ci-dessous montrent l'interconnexion des rebouclages dont a besoin l'implémentation du D-W2x2 :

$$Out_0 = C_{0,0} = LUT\ (C_{0,0},\ C_{0,1},\ C_{1,0},\ B_0,\ A_0,\ A_1)$$

$$Out_2 = C_{0,1} = LUT\ (C_{0,0},\ C_{0,1},\ B_1,\ C_{1,1},\ A_0,\ A_1)$$

$$Out_6 = C_{1,0} = LUT\ (C_{0,0},\ B_0,\ C_{1,0},\ C_{1,1},\ A_0,\ A_1)$$

$$Out_8 = C_{1,1} = LUT\ (B_1,\ C_{0,1},\ C_{1,0},\ C_{1,1},\ A_0,\ A_1)$$

ÉQUATION 12 : ÉQUATIONS DES SORTIES DU D-W2X2

Noter que d'après l'Équation 10, la variable A_0 n'entre pas dans le calcul de $C_{0,0}$ et $C_{0,1}$. La raison pour laquelle elle l'est dans les fonctions de l'Équation 12 est que B_0 et B_1 sont con-

nectés aux 4 LUT6 (Figure 32). Il est de même pour le cas de A_1 dans les équations de $C_{1,1}$ et $C_{1,0}$. De cette façon les 4 entrées A_0, A_1, B_0 et B_1 conservent leur symétrie de charge, chose qui est très importante pour la sécurité du bloc.

La Figure 32 montre un exemple d'implémentation du D-W2x2 dans le PLB déjà défini (la partie grisée signifie qu'elle n'est pas utilisée).

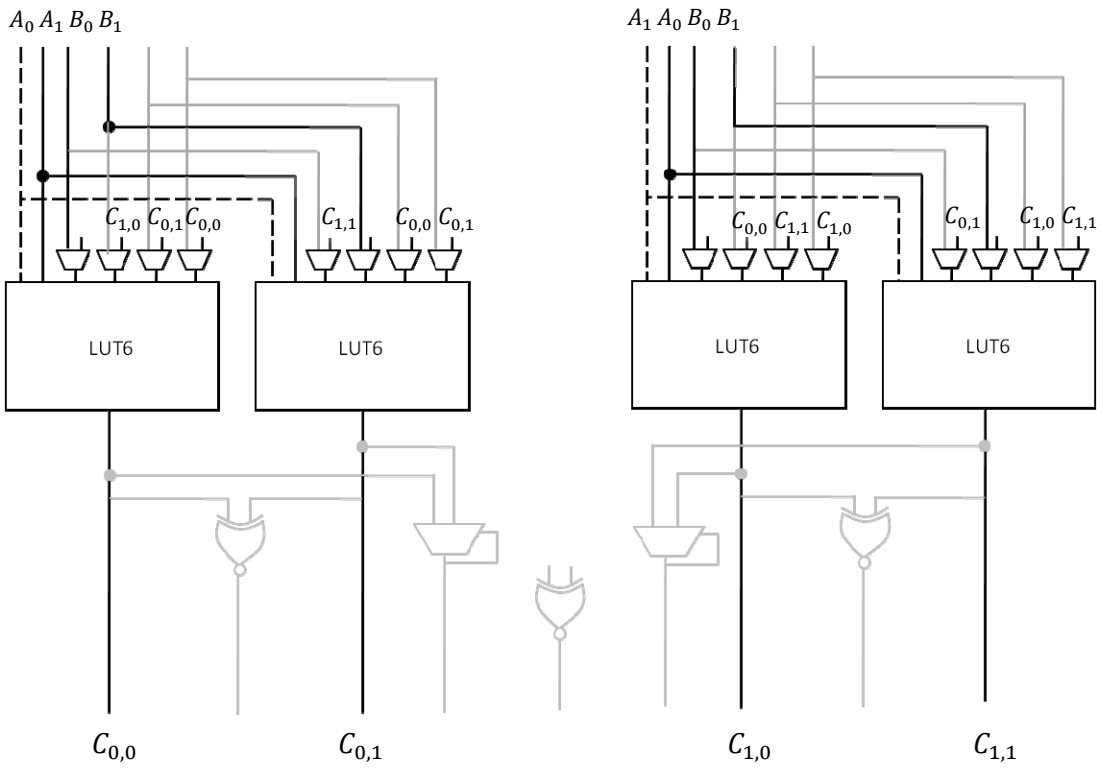


FIGURE 32 : IMPLEMENTATION D'UN D-W2X2 DANS LE PLB

3.1.3.2.2 ETAGES DE CALCUL ET DE SYNCHRONISATION

Le D-W2x2 fournit à sa sortie des données codées : les $C_{i,j}$ qui subissent des transitions à chaque fois que des nouvelles valeurs de i et j arrivent. Le Tableau 1 donne les expressions de calcul des signaux D_0 et D_1 en fonction des $C_{i,j}$. Ces résultats sont calculés en effectuant des « Ou exclusifs » des $C_{i,j}$ correspondants.

f(a,b)	D_0	D_1
0	$C_{1,1} \oplus C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$	0
1	0	$C_{1,1} \oplus C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$
OR	$C_{1,1} \oplus C_{1,0} \oplus C_{0,1}$	$C_{0,0}$
AND	$C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$	$C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$
XOR	$C_{1,1} \oplus C_{0,0}$	$C_{1,0} \oplus C_{0,1}$
NOR	$C_{0,0}$	$C_{1,1} \oplus C_{1,0} \oplus C_{0,1}$
NAND	$C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$	$C_{1,0} \oplus C_{0,1} \oplus C_{0,0}$
XNOR	$C_{1,0} \oplus C_{0,1}$	$C_{1,1} \oplus C_{0,0}$
a	$C_{1,1} \oplus C_{1,0}$	$C_{0,1} \oplus C_{0,0}$
b	$C_{1,1} \oplus C_{1,0}$	$C_{0,1} \oplus C_{0,0}$
$\bar{a}$	$C_{0,1} \oplus C_{0,0}$	$C_{1,1} \oplus C_{1,0}$
$\bar{b}$	$C_{0,1} \oplus C_{0,0}$	$C_{1,1} \oplus C_{1,0}$
$a\ OR\ \bar{b}$	$C_{0,1}$	$C_{1,1} \oplus C_{1,0} \oplus C_{0,0}$
$\bar{a}\ OR\ b$	$C_{1,0}$	$C_{1,1} \oplus C_{0,1} \oplus C_{0,0}$
$a\ AND\ \bar{b}$	$C_{1,1} \oplus C_{0,1} \oplus C_{0,0}$	$C_{1,0}$
$\bar{a}\ AND\ b$	$C_{1,1} \oplus C_{1,0} \oplus C_{0,0}$	$C_{0,1}$

TABLEAU 1 : EXPRESSION DES SORTIES EN FONCTIONS DE LA FONCTION F(A,B) CALCULEE.

Les équations du tableau sont des équations combinatoires à 4 variables. Elles peuvent être implantées sur un couple de LUT6 dans un PLB.

Le dernier étage de synchronisation de la Figure 29, est réalisé par le D-W2x1 de la Figure 33. Ses sorties O_1 et O_0 qui ne sont autre que les sorties de la porte à 2-entrées, sont décrites par les équations suivantes :

$$O_0 = \begin{cases} I_0 & si \quad \overline{ack_{in} \oplus O_1} = I_0 \\ O_0^{-1} & sinon \end{cases}$$

$$O_1 = \begin{cases} I_1 & si \quad \overline{ack_{in} \oplus O_0} = I_1 \\ O_1^{-1} & sinon \end{cases}$$

$$\overline{ack_{out}} = \overline{O_0 \oplus O_1}$$

ÉQUATION 13 : ÉQUATIONS DE SORTIE DU BLOC DE SYNCHRONISATION D-W2X1

Lors de l'initialisation $O_1 = I_1$, $O_0 = I_0$ et donc ($ack_{in} \oplus O_1$) $\neq I_0$ et ($ack_{in} \oplus O_0$) $\neq I_1$. Lorsqu'un événement arrive sur I_i , celle-ci change de valeur et devient égale à ($ack_{in} \oplus O_j$) ; avec $\bar{\imath} = j$.

Ce changement transmet la valeur de I_i à la sortie O_i correspondante à travers la porte de Muller. Lorsque la sortie O_i bascule, la valeur du ($ack_{in} \oplus O_i$) change aussi et devient égale à O_j.

A ce stade, les sorties de la porte se sont stabilisées. Même si les entrées changent de valeur, les sorties garderont leur anciennes valeurs jusqu'à l'arrivée d'un nouvel acquittement ack_{in} signalant que la porte en aval est prête à recevoir de nouvelles données.

D'après l'Équation 13, l'implémentation de ce bloc peut être faite sur 2 LUT6 d'un PLB comme le montre la Figure 34

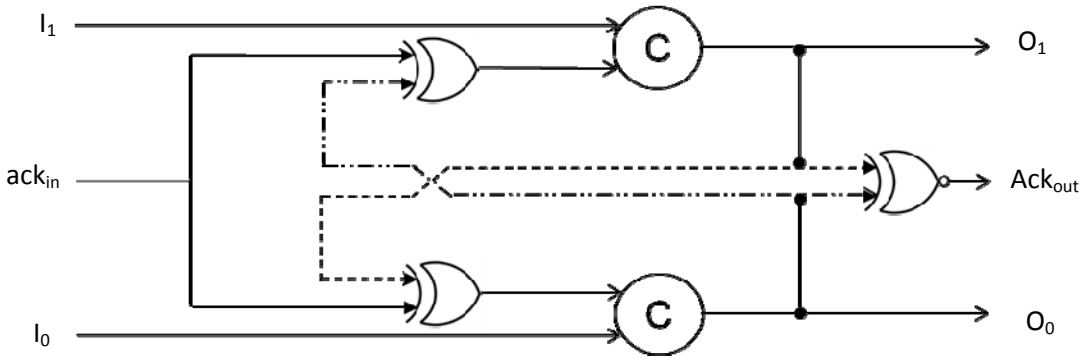


FIGURE 33 : DECISION-WAIT 2X1

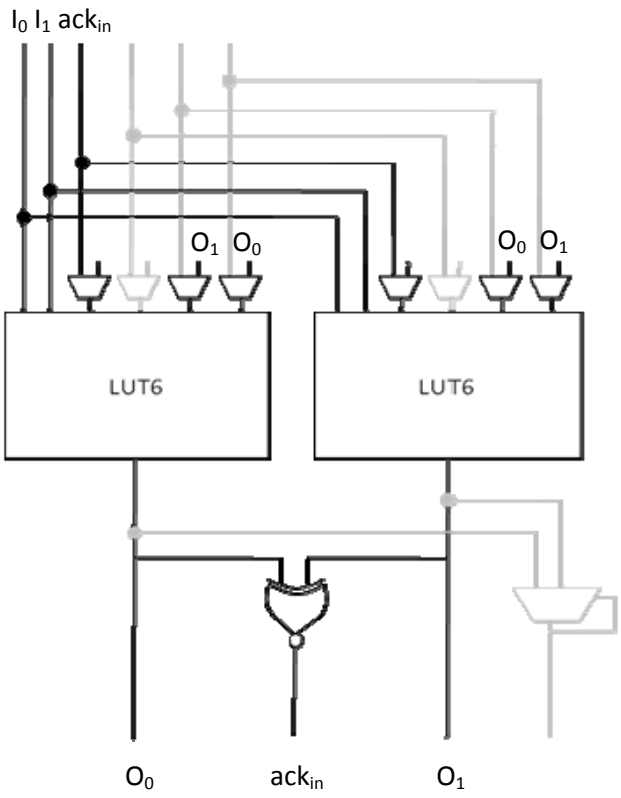


FIGURE 34 : IMPLEMENTATION DU DECISION-WAIT 2X1

3.1.3.3 CONCLUSION SUR L'IMPLEMENTATION DU PROTOCOLE 2 PHASES SUR FRONT

Le protocole 2 phases sur front est difficile à implémenter. Il consomme beaucoup de ressources du FPGA. L'implémentation d'une porte à deux entrées requiert 2 PLBs complets.

Cependant ce protocole présente des avantages vis-à-vis de la sécurité parce qu'il utilise le système de transitons pour coder ses données. Un '1' logique est présenté soit par une transition montante soit par une transition descendante.

La version multi-styles du PLB est donc la suivante :

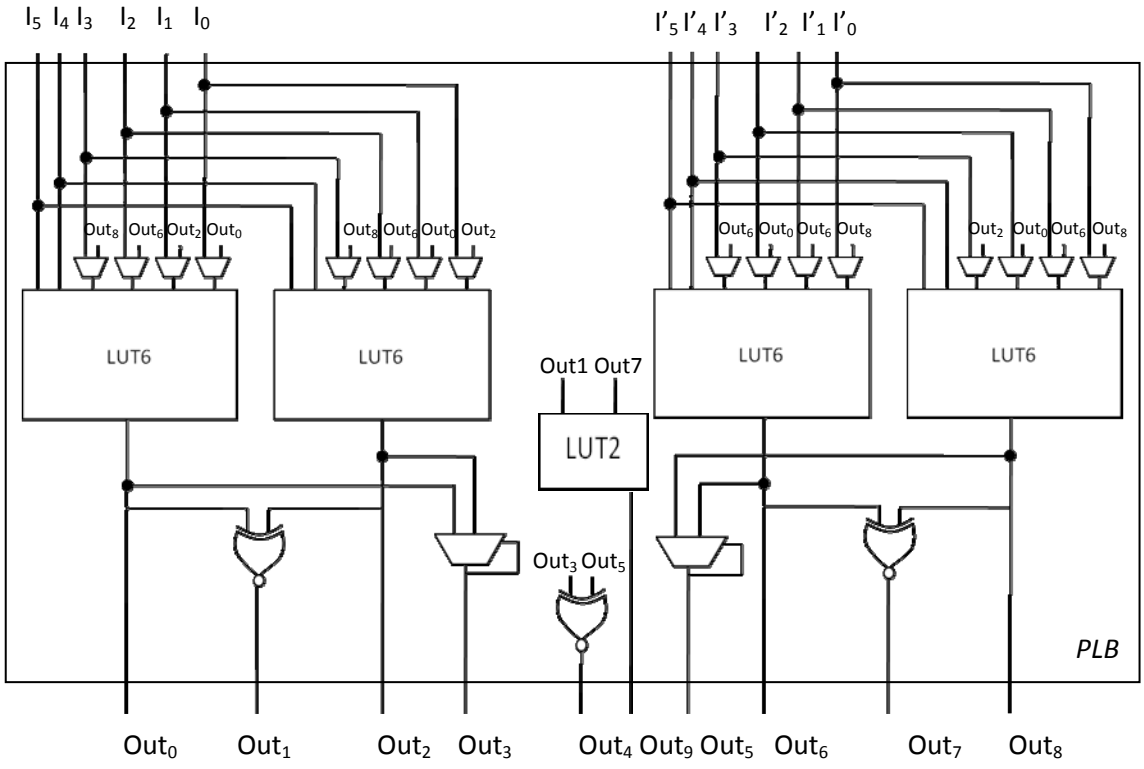


FIGURE 35 : VERSION MULTI-STYLES DU PLB

3.2 ASPECTS SECURITAIRES DU BLOC PROGRAMMABLE

De nombreuses contre-mesures ont été développées pour protéger les circuits contres les attaques dites matérielles. Les enjeux économiques ont rendus ce domaine tellement sensible que les systèmes de sécurité implémentés dans les circuits sont devenus des informations confidentielles. En effet, les industriels communiquent très peu sur les moyens de protection développés et implémentés dans leurs produits. Toutefois, les concepts proposés dans les laboratoires académiques publics sont proches de ceux développés dans l'industrie.

Cette partie traite du deuxième objectif : « la sécurité du FPGA asynchrone». L'idée est de concevoir un FPGA asynchrone capable de tester les différentes contre-mesures pour les attaques en puissance, les attaques électromagnétiques et les attaques temporelles d'une part et, d'autre part, d'évaluer sa résistance contre d'autres types d'attaques telles que les attaques par injection de faute.

Les contre-mesures considérées peuvent être classées en deux sous familles : les contre-mesures matérielles et les contre-mesures logiques. La première famille concerne l'architecture électrique des blocs programmables, et du réseau d'interconnexion (*Switch-box* et *Connection-box*) du FPGA. Les contre-mesures logiques concernent les techniques de codage des données, la profondeur logique du circuit, ainsi que le placement et le routage sur le FPGA.

3.2.1 LE CODAGE DES DONNEES

En technologie CMOS, la dissipation de courant est dominée par les pertes par consommation ou aussi appelé : « *switching activity* ». Ainsi, la consommation de courant d'un circuit est fortement dépendante de l'activité produite par le changement des données sur les entrées du circuit. Dans le cas du codage simple rail, où un bit est codé sur un seul fil, ce phénomène est considéré comme le poids de Hamming des changements des états.

Les attaques par analyse de courant DPA (T. Messerges, E. Dabbish, R. Sloan, 1999) (P. Kocher, J. Jaffe, B. Jun, 1999) ont évitées en cachant ou en réduisant les fuites d'informations dues à l'activité. Ces informations peuvent être supprimées ou au moins réduites en essayant de rendre la consommation du courant du circuit indépendante des données. Au niveau logique, il s'agit de trouver une technique pour annuler la corrélation entre les données et la consommation du courant (S.chari, C.S. Jutla, J.R. Rao and P. Rohatgi, 1999). Bien que ces techniques soient susceptibles de fournir un degré élevé de sécurité, elles impliquent souvent des coûts très élevés en surface (S. Moore, R. Andersona, R. Mullinsa, G. Taylora and J. Fournier, 2003).

Dans le paragraphe présent, la contre-mesure présentée s'appuie sur l'équilibrage de la consommation en courant au niveau de la porte. En effet, la logique asynchrone possède des propriétés intrinsèques particulièrement bien adaptées pour faire face aux attaques non intrusives et plus spécialement aux attaques en courant. Parmi ces propriétés, le codage de données en 1parmi n (D. Suzuki and M. Saeki, 2006) est une bonne solution. De manière plus générale le codage en m parmi n pourrait être utilisé, mais il s'avère que le codage en 1 parmi n est le plus utilisé car il est le plus approprié à la logique asynchrone.

Plusieurs alternatives existent également au niveau logiciel, pour équilibrer la consommation en courant, cependant implémenter une solution au niveau matériel reste obligatoire. Différents types de logiques en technologie CMOS ont été proposés dans (K. Tiri and I.

Verbauwhede, 2004). Toutefois cette approche de très bas niveau requiert de modifier les bibliothèques de portes standards. Cette approche est ainsi très coûteuse pour les utilisateurs de cellules standards ou de FPGAs. En conséquence, l'utilisation d'un codage de données équilibré comme le 1 parmi n, combinée avec de la logique asynchrone a été proposée (S. Moore, R. Andersona, R. Mullinsa, G. Taylora and J. Fournier, 2003), (S. Moore, R. Andersona, P. Cunningham, M. Robert and G. Taylor, 2002) et (Z. Yu and S. Furber, 2003). Simon Moore montre comment l'utilisation d'un codage des données de type 1 parmi n permet :

- De réduire les dépendances entre les données manipulées et le courant consommé.
- De résister aux attaques par injection de fautes en utilisant l'état invalide (1 1) du codage de données pour générer des alarmes en cas de détection de fautes.
- De disperser les dépendances temporelles.

Ce codage garantit un poids de Hamming constant quelques soient les données manipulées. Le poids de Hamming étant le facteur logique le plus important sur l'origine des fuites d'information à travers le courant consommé. Un bit logique '1' doit posséder le même poids de Hamming qu'un bit logique '0', s'il est codé en double rail ('0 1' et '1 0' respectivement).

Ce type de codage associé au codage trois états présentés dans la section 3.1.1.1 permet de maîtriser le nombre de transitions logiques dans une structure indépendamment des données. Cela induit une consommation équilibrée du courant, indépendante de l'activité liée aux commutations. En partant d'un état invalide (00) vers un état valide (01) ou (10), une seule transition sur l'un des rails est effectuée.

Malheureusement, le codage en 1 parmi n n'est pas suffisant pour garantir une indépendance totale de la consommation des données utilisées (D. Sokolov, J. Murphy , A. Bystrov and A.Yakovlev, 2004). Pour les circuits QDI 4 phases, l'insensibilité aux délais est assurée par l'utilisation d'un troisième *code-word* '0 0' (J. Sparso, 2006), qui sépare les données valides (cf. section 3.1.1). Ce comportement, souvent appelé RTZ[29], a deux conséquences majeures à l'égard de la DPA (K. J. Kulikowski, M. Su, A. Smirnov, A. Taubin, Mark G. Karpovsky and D. MacDonald, 2005) :

- Chaque porte va basculer d'un état invalide '0 0' à un état de données valides '0 1' ou '1 0', ce qui donne une idée à un attaqueur sur l'état de la porte à chaque transition.
- La complexité du système en terme de nombre de transitions est réduite de n^2 à n, où n est le nombre de vecteurs d'entrées possibles.

[29] Return to zero

Pour résoudre ce problème, il est crucial d'avoir des portes bien équilibrées, qui ne favorisent pas ce genre de fuites. Dans le cadre de ce travail, les portes sont les PLB, les *Connection-boxes* et les *Switch-boxes.* Leurs designs doivent garantir des charges d'entrées égales et une consommation électrique unique pour chacune. Généralement ce travail est facile à faire aux niveaux portes, mais il doit être appliqué lors du placement et du routage. En effet, bien que chaque bloc puisse avoir une signature électrique indépendante des données, l'activité globale peut ne pas l'être. Un travail très important doit être achevé sur ces niveaux pour assurer un bon niveau de résistance contre les attaques par canaux cachés et spécialement les attaques par analyse de courant. Dans la suite de ce chapitre tous ces problèmes sont examinés en détails afin de trouver des solutions convenables.

Il est à noter qu'une autre contre mesure a été introduite par le protocole 2 phases sur front : le codage des données sous formes de transitions. Un bit logique '1' peut être désigné par une transition montante comme par une transition descendante. Cela augmente la résistance du circuit à la DPA parce qu'un attaquant doit trouver la différence entre la consommation moyenne des deux transitions sur le fil 1 d'une part et sur le fil 2 d'une autre part.

3.2.2 PROFONDEUR LOGIQUE – TEMPS DE PROPAGATION UNIQUE

Pour éviter toute dépendance temporelle entre les données manipulées et le temps d'exécution, les concepteurs doivent, à tous les niveaux de conception, éviter d'utiliser des fonctions, des blocs et des branchements qui dépendent directement des données, ou de la clé dans le cas des algorithmes de cryptage. L'idée est soit d'uniformiser le temps de calcul de chaque fonction ou de rendre l'exécution des fonctions totalement aléatoire en rajoutant des instructions aléatoires afin de complexifier l'analyse du temps d'exécution (J. F. Dhem, 1998) et (P. Kocher, 1996).

Le temps de propagation des données dans un FPGA dépend de trois principaux facteurs : le temps de propagation dans un PLB, le temps de propagation dans les *Switch-boxes* et *Connection-boxes* et finalement le délai induit par les fils du réseau d'interconnexion. Ce travail vise à uniformiser le temps de calcul des PLBs, quelque soit les données manipulées, en équilibrant les chemins de propagations des entrées du PLB vers les sorties. Ces chemins devront aussi être uniques pour tous types de fonctions implémentées. Les autres facteurs liés au réseau d'interconnexions, ont été le sujet de thèse de S. Chaudhuri (S. Chaudhuri, 2009) effectuée dans le département ComElec à Telecom ParisTech.

La dernière version du Bloc programmable présentée dans la section 3.1.3 est la suivante :

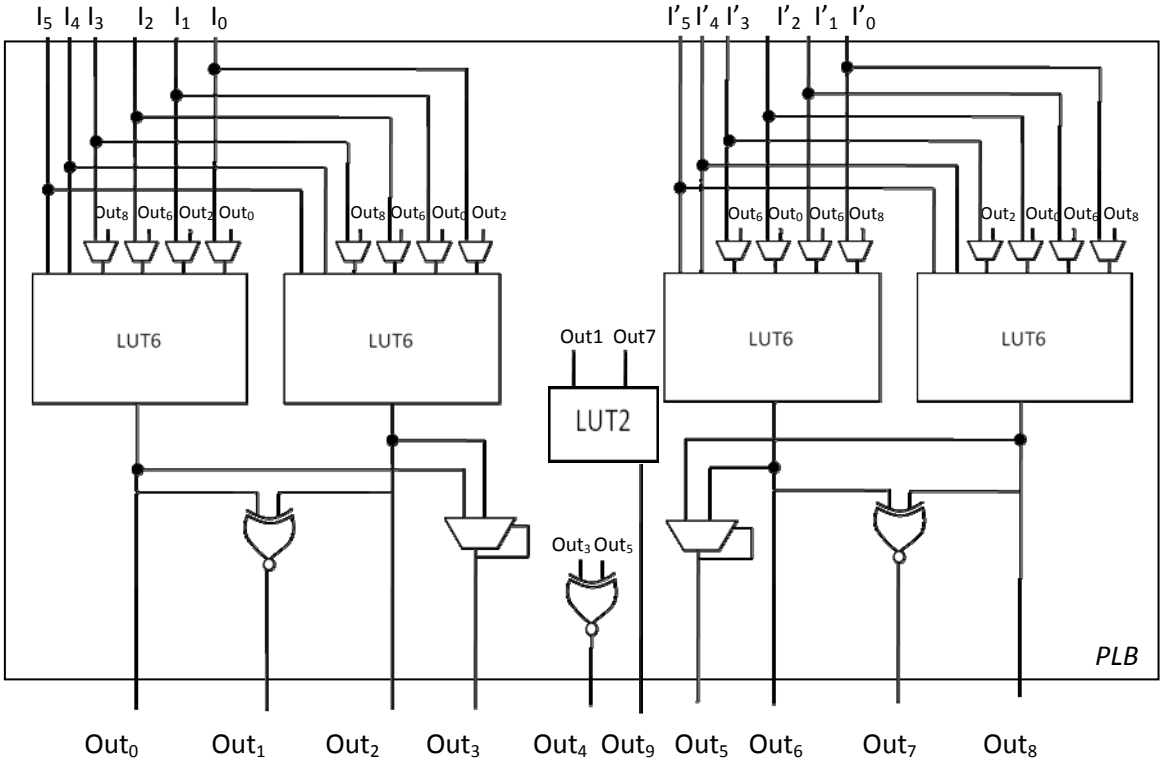


FIGURE 36 : DERNIERE VERSION DU PLB NON EQUILIBREE

Comme le montre la Figure 36, une dissymétrie existe entre les deux premières entrées et les 4 dernières de chaque LUT6. Elle est considérée comme un défaut de sécurité, à cause du retard causé par les multiplexeurs sur les 4 entrées. Ce défaut est une source de fuite d'informations, qui peut être exploitée par un attaquant pour réaliser des attaques par analyses de temps.

L'ajout de deux multiplexeurs sur les deux premières entrées, peut éventuellement corriger ce problème. Le premier multiplexeur sera connecté à une entrée extérieure d'une part et à la sortie de la LUT6 concernée (cf. Figure 37). Outre l'équilibrage du délai, ce multiplexeur ajoute plus de flexibilité au niveau de la connexion du PLB au réseau d'interconnexion du FPGA. Le deuxième multiplexeur sera aussi connecté à une entrée extérieure d'une part, et à la sortie Out_5 dans le cas du couple des LUT6 de droite (cf. Figure 37). Celui-ci permettra de concaténer les LUT6 pour implémenter des fonctions à plus de 3 entrées.

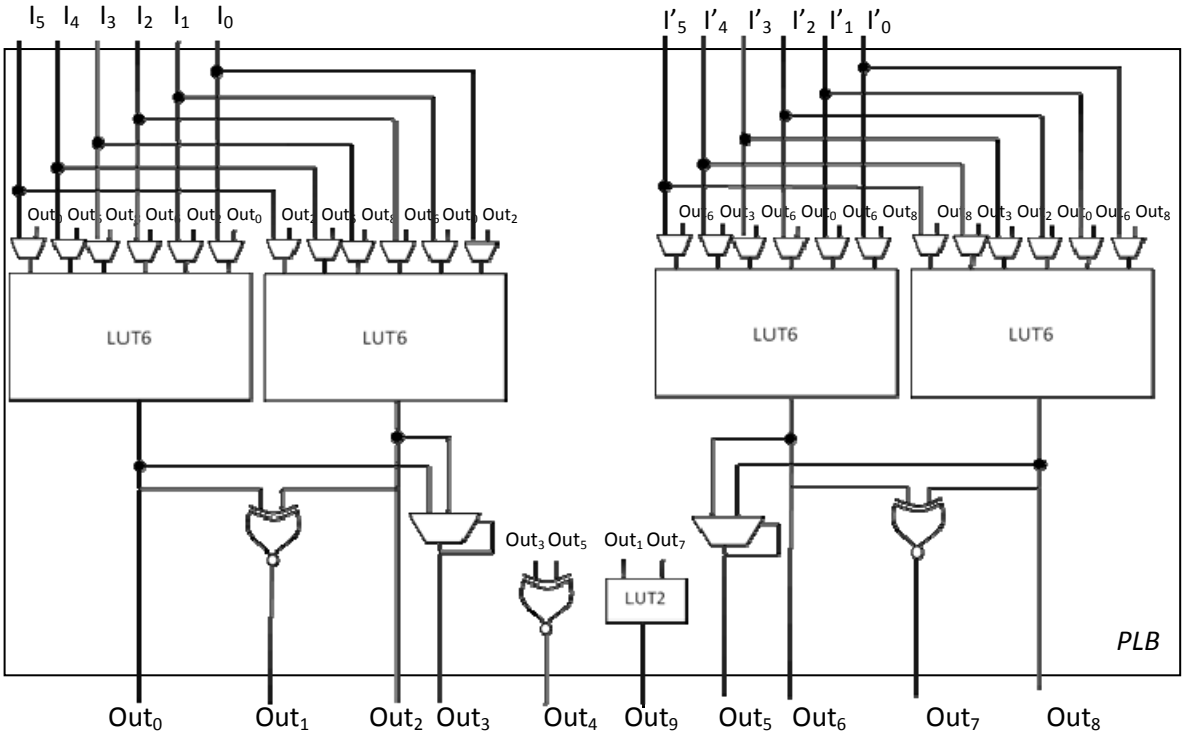


FIGURE 37 : VERSION DE L'ARCHITECTURE DU PLB EQUILIBREE

Cette nouvelle architecture, est équilibrée au niveau logique. Les charges de toutes les entrées sont identiques ainsi que le temps de propagation de chaque entrées vers les sorties du PLB. Pour conserver cet avantage, il faut bien penser à équilibrer l'architecture électrique et les fils à l'intérieure du PLB, et bien dimensionner les transistors de ce dernier (ces aspects seront étudiés dans le chapitre suivant).

Une amélioration de cette architecture peut être faite au niveau du nombre de sorties du PLB. En fait d'après ce qui a été présenté en 4 phases et 2 phases (section 3.1.1, 3.1.2 et 3.1.3), les sorties Out_1 et Out_3 ne sont jamais utilisées en même temps. Il est de même pour les sorties Out_5/Out_7 et les sorties Out_4/Out_9. Avec cette hypothèse, le nombre de sorties du PLB peut être réduit de 10 à 7 en ajoutant 3 multiplexeurs comme le montre la Figure 38. Noter que les sorties 'Out_1, Out_3, Out_4, Out_9, Out_5, Out_7' sont maintenant dans cette figure nommées 'O_1, O_3, O_4, O_9, O_5, O_7'. Le terme « Out_i » est conservé pour les sorties réelles du PLB. Cette réduction ajoutera 3 points de programmation mais simplifiera la complexité du réseau d'interconnexion du FPGA.

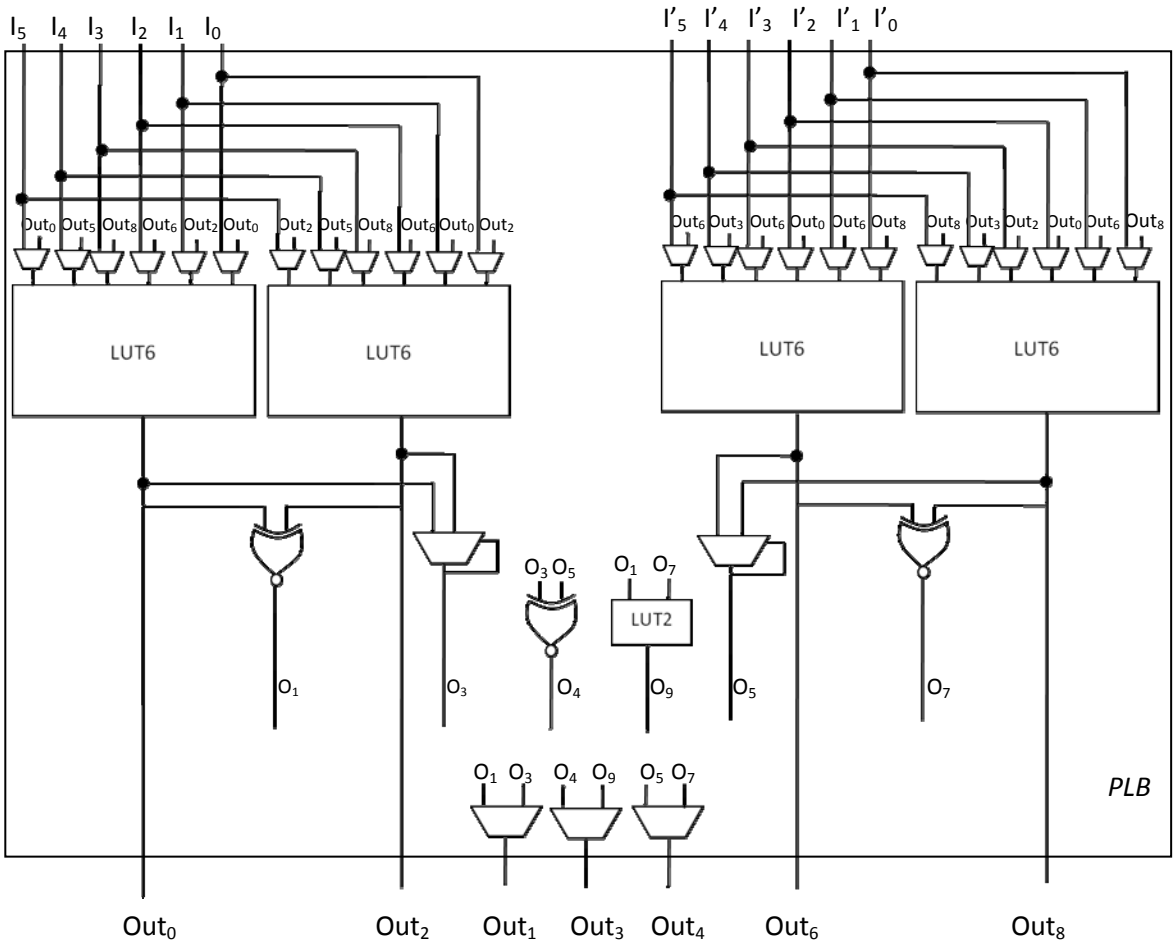


FIGURE 38 : VERSION FINALE DU PLB EQUILIBRE

3.2.3 LOGIQUE ASYNCHRONE

Le fait d'utiliser la logique asynchrone, et spécialement la famille des circuits QDI, a des effets positifs en termes de sécurité et de résistance du bloc programmable.

En effet, l'absence de signal d'horloge signifie que les attaques par injection de fautes basées sur le signal d'horloge sont évitées. En outre, les attaques DPA et SPA, en l'absence du signal d'horloge vont devenir plus difficiles à réaliser. Son absence rend plus difficile voire très difficile la synchronisation de la DPA et la SPA.

Enfin l'absence de l'horloge et la communication des blocs entre eux permettent de répartir la consommation au cours du temps au lieu d'avoir un pic de consommation comme dans le cas du synchrone. De ce fait, les circuits asynchrones ont une consommation plus lisse que celle des circuits synchrones. Cela diminue aussi l'émission électromagnétique du circuit.

Cela devrait rendre les attaques par analyse de courant et du champ électromagnétique plus difficiles à effectuer.

Un autre bénéfice s'ajoute au compte de la logique asynchrone : la tolérance aux variations des paramètres de l'environnement. Les circuits QDI s'adapte bien aux variations de tension, température, etc. Cela signifie qu'ils tolèrent plusieurs formes d'injection de fautes (*power glitches, thermal gradients*, etc.) (Y. Monnet, M. Renaudin and R. Leveugle, 2006).

3.3 ASPECTS SECURITAIRES – RESEAU D'INTERCONNEXION

3.3.1 PLACEMENT ET ROUTAGE SECURISE

Bien que le placement et routage joue un rôle très important au niveau de la sécurité du FPGA, ils n'ont pas été traités dans le cadre de cette thèse. Une partie a été faite par S. Chaudhuri dans le cadre de sa thèse (S. Chaudhuri, 2009). Cette partie concerne le réseau d'interconnexion : *Switch-boxes + Connection-boxes* ainsi que le routage. Au niveau Logiciel, l'auteur a utilisé le logiciel de placement et routage VPR (V. Betz and J. Rose, 1997). Celui-ci prend une *netlist* VHDL en entrée et génère le *bitstream* pour placer et router sur le FPGA asynchrone. Concernant l'aspect matériel, l'auteur a montré dans son manuscrit qu'il a pu équilibrer électriquement les *Connection-boxes* et les *Switch-boxes* pour garder une consommation équilibrée en courant. Il a aussi présenté une solution pour que le routage n'induise pas de variations de délai sur le chemin des données, notamment entre les rails d'un mot logique.

Cependant un algorithme de *tech-mapping* a été développé dans le cadre de cette thèse, permettant d'implémenter d'une manière sécurisée, les fonctions sur le FPGA. Il sera présenté en détails dans ce manuscrit dans le chapitre **Chapitre 5**. L'algorithme permet de générer une *netlist* en VHDL qui respecte bien l'équilibrage logique ainsi qu'électrique du circuit. Cette *netlist* peut ensuite être utilisée par l'outil VPR pour achever le placement & routage sur le FPGA fabriqué. Il reste à noter que le logiciel VPR ne respecte pas les contraintes de placement sécurisé.

3.4 CONCLUSION

Ce chapitre a présenté les protocoles de communications 4 phases et 2 phases. Il a aussi présenté les spécifications du bloc programmable du FPGA asynchrone. Le réseau d'interconnexion a fait l'objet d'une autre thèse, ses caractéristiques ont été présentées dans (S. Chaudhuri, 2009). Parmi les éléments de spécification, la problématique de la sécurité est présentée et étudiée. Il s'agit notamment d'analyser les critères et les contre-mesures qui permettront d'une part de durcir l'architecture du FPGA et de la rendre résistante aux at-

taques par canaux cachés et, d'autre part, de lui assurer un certain niveau de flexibilité pour implémenter plusieurs types de protocoles de communication (4 phases et 2 phases) et plusieurs codages de données : 1 parmi n, double rail LEDR.

Les contre-mesures ont été classées en deux types :

- Contre-mesures matérielles qui concernent l'équilibrage électrique du bloc programmable du FPGA.
- Contre-mesures logiques qui concernent les techniques de codage de données, la profondeur logique.

L'architecture logique proposée pour le PLB possède 12 entrée et 7 sorties. Elle est optimisée pour une architecture multi-style. Elle contient 4 LUT6 dont les entrées et les sorties sont connectées d'une manière spécifique pour assurer la flexibilité de l'ensemble tout en étant sécurisé. Elle contient aussi de la logique combinatoire qui implémente une partie des protocoles de communication. L'architecture proposée est équilibrée logiquement et électriquement afin de garantir une indépendance des données vis-à-vis de la consommation et du temps de calcul.

Chapitre 4.
CONCEPTION D'UN FPGA ASYNCRHONE POUR LA SÉCURITÉ

Le chapitre précédent a détaillé les caractéristiques du bloc programmable asynchrone en termes de taille, nombre d'entrées et de sorties, nombre de points de programmation etc. Il a aussi identifié les différentes contre-mesures logiques et matérielles nécessaires pour avoir un FPGA asynchrone sécurisé.

Dans cette optique, ce chapitre sera divisé en trois parties principales :

- La première présente l'architecture logique détaillée du bloc programmable PLB.
- La deuxième partie définie l'architecture électrique du PLB. Une brève explication sur le dimensionnement des transistors sera aussi présentée.
- Finalement la troisième partie montre le *Layout full custom* de chaque élément du PLB.

Les différentes parties mettront en évidence l'effort qui a été fait aussi bien au niveau logique qu'au niveau électrique pour sécuriser le FPGA asynchrone.

4.1 ARCHITECTURE LOGIQUE

4.1.1 TECHNIQUE DE MEMORISATION

Le C-élément est une cellule très importante dans la conception des circuits asynchrones. Cette porte est une fonction qui copie les entrées sur la sortie lorsqu'elles sont égales, et maintient la valeur de la sortie si les entrées sont différentes. Cela bien évidemment nécessite une technique spéciale d'implémentation dans le FPGA d'un point mémoire, qui assure ce comportement.

Cette technique dépend fortement de l'architecture du bloc primitif de programmation (dans le cas de ce travail ce bloc primitif est la LUT6). Dans le cas d'une architecture à base de multiplexeurs (cf. Figure 39), les entrées de la LUT sont connectées aux entrées de sélection des multiplexeurs. Les points de programmation de la LUT sont les points de mémoire de la SRAM. Selon les entrées manipulées, le point de mémoire correspondant est propagé à la sortie (P. Chow, S. Ong Seo, J. ROse, K. Chung, G. Paéz-Monzon and I. Rahardja, 1999). Pour concevoir une LUT à « n » entrées, « $n + \frac{n}{2} + \frac{n}{4} + \cdots + 1 = 2^n - 1$ » multiplexeurs sont nécessaires (L. Fesquet, M. Renaudin, 2005). La notation LUTn signifie que la LUT possède n entrées.

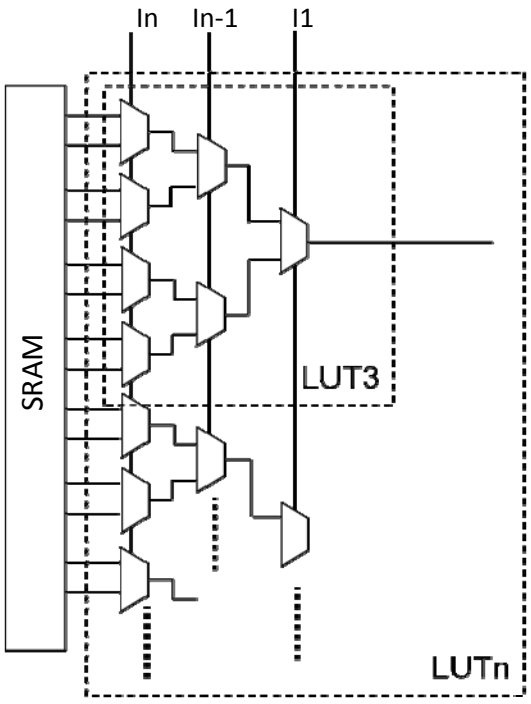


FIGURE 39 : ARCHITECTURE A BASE DE MULTIPLEXEURS D'UNE LUT A N-ENTREES

Afin d'être compatible avec la logique asynchrone et en particulier les portes Muller, une LUT doit être capable d'implémenter un point mémoire. La Figure 40 montre un exemple d'architecture supportant une mémorisation à sa sortie. En effet, un rebouclage « interne » au niveau de la sortie est ajouté. Ce rebouclage interne est commandé par un multiplexeur Mux2. Selon la fonction à calculer, le point de mémoire est requis ou non. Ainsi il est possible soit d'utilisé la dernière entrée « R » de la LUT3, soit d'activer la mémorisation. La mémorisation est commandée par un bit de sélection « M » : si « M » est égal à '0' la mémorisation est activée, sinon (« M » = '1') le mode de fonctionnement de la LUT repasse en mode normal.

Cette structure possède plusieurs avantages :

- Elle a un faible coût : elle requiert un seul multiplexeur, donc un seul point de programmation. Dans le cas d'une LUT6, il s'agit d'ajouter un point de programmation sur les 64 existants.
- La mémorisation est programmable, la LUT peut fonctionner avec ou sans la mémorisation.

Par ailleurs le coût induit en vitesse est très bas (L. Fesquet, M. Renaudin, 2005).

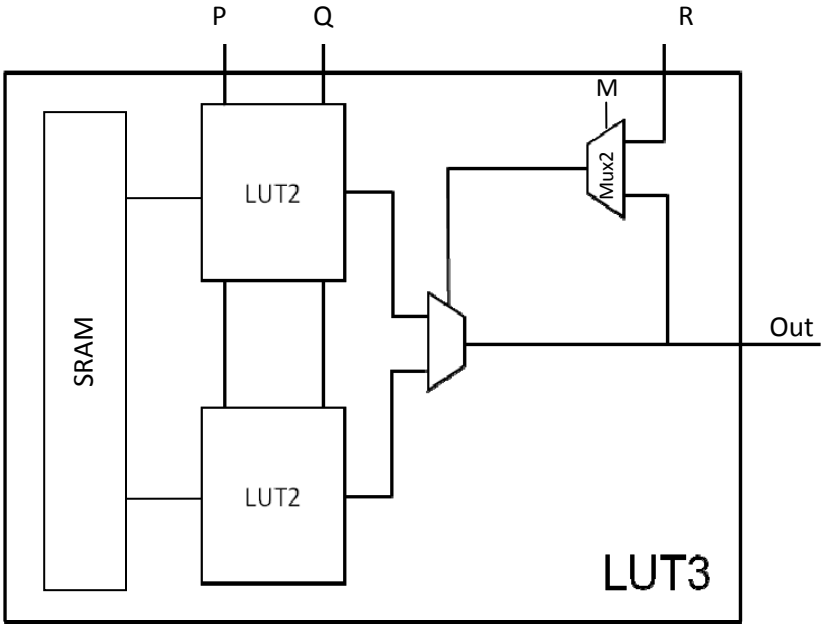


FIGURE 40 : ARCHITECTURE A BASE DE MULTIPLEXEURS D'UNE LUT A 3-ENTREES

La LUT3 de la Figure 40 permet d'implémenter une porte Muller à 2 entrées. Pour le faire le point de programmation « M » est activé (M=1), donc R n'est plus utilisée et les entrées de la porte Muller sont : P et Q. Quant à la mémorisation, elle est assurée par le multiplexeur Mux2 à la sortie.

D'une manière générale une LUTn permet d'implémenter une fonction à n-1 entrées + 1 mémorisation.

4.1.2 CHAINE DE PROGRAMMATION ASYNCHRONE

La structure à base de multiplexeurs souffre d'un inconvénient majeur qui est un temps de propagations non équilibré. Ce problème devient inacceptable, lorsqu'il s'agit de sécurisé l'architecture. En effet, le temps de propagation d'un point de programmation de la SRAM vers la sortie n'est pas constant. Il dépend de l'entrée activée. Dans le cas de la LUT3 (cf. Figure 40) un point de programmation prend plus de temps pour atteindre la sortie après une permutation de l'entrée « P », alors qu'il est plus rapide si Q a changé de valeur. Cela signifie un temps de calcul dépendant des données qui compromet la sécurité de la LUT3 et la rend une cible facile pour les attaques temporelles. L'architecture adoptée – dans la suite du chapitre – est une architecture basée sur le même principe, mais qui est équilibrée (cf. Figure 41).

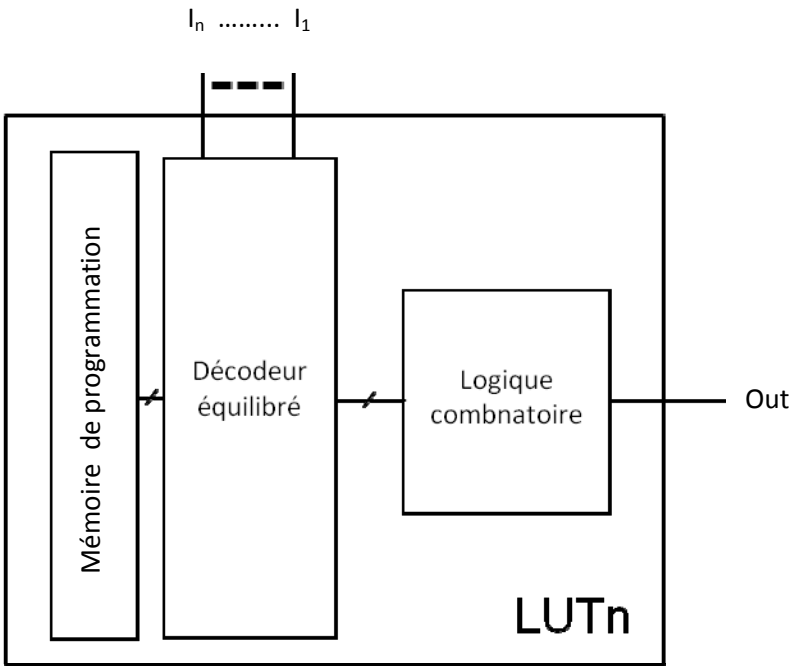


FIGURE 41 : ARCHITECTURE A BASE DE MULTIPLEXE D'UNE LUT A N-ENTREES

4.1.2.1 FIFO ASYNCHRONE

La mémoire de programmation est une FIFO asynchrone. Cette FIFO est constituée d'une suite de Full-Buffer asynchrone à base de portes de Muller et de XNOR qui communiquent entre elles en utilisant le protocole 4 phases (cf. Figure 42). L'utilisation de la logique asynchrone et du protocole 4 phases n'est pas ici lié au fait que le FPGA cible de la logique asynchrone. Une FIFO synchrone normale aurait suffit. Par contre, l'utilisation d'une FIFO asynchrone augmente la résistance du FPGA car les propriétés intrinsèques de l'asynchrone bénéficie à la sécurité notamment en supportant mieux les variations de l'environnement (température, tension, etc.). Chaque LUT6 contient une chaine de 64 *Full-Buffers* en séries.

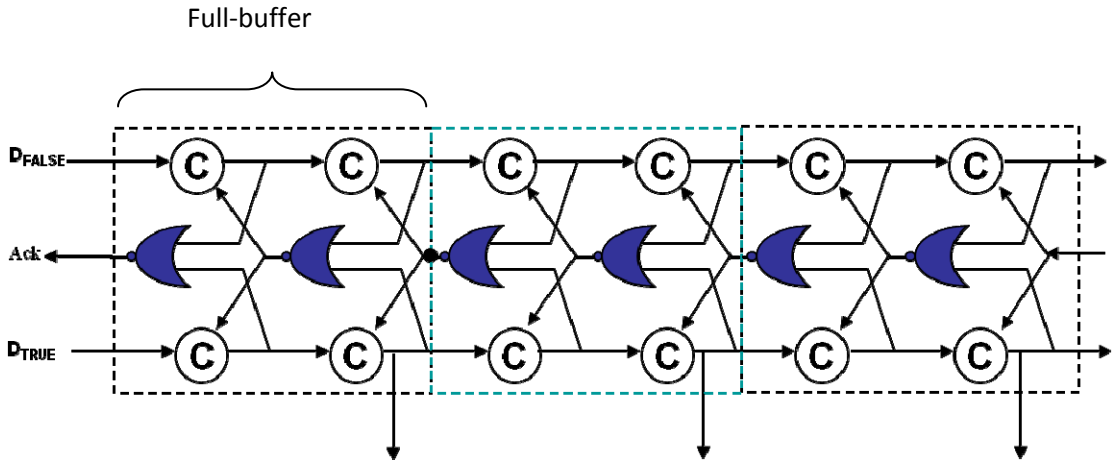


FIGURE 42 : ARCHITECTURE DE LA MEMOIRE DE PROGRAMMATION

4.1.2.2 RESET: CIRCUIT BOUCHON

Le circuit « bouchon » de la Figure 43 permet de mettre tous les bits de la chaine de programmation à (0,0), et d'éviter une mauvaise initialisation (cas interdit : (1,1)) (cf. section 3.1.1) après le démarrage. Elle permet d'effacer une configuration existante pour la remplacer par une autre. Pour initialiser, le signal « Init » et les entrées de la Fifo D_{false} et D_{true} sont mis à zéro pendant un temps suffisant, pour que les couples (0,0) aux entrées se propagent tout au long de la chaine. Cela correspond à un *RESET* de la FIFO. Si un état interdit apparait à la mise sous tension, il sera effacé pendant cette phase et remplacé par (0,0).

Pendant la phase de programmation de la chaîne, le signal « Init » et mis à 1. Cela permet de propager à partir des entrées les bonnes valeurs de configuration du FPGA.

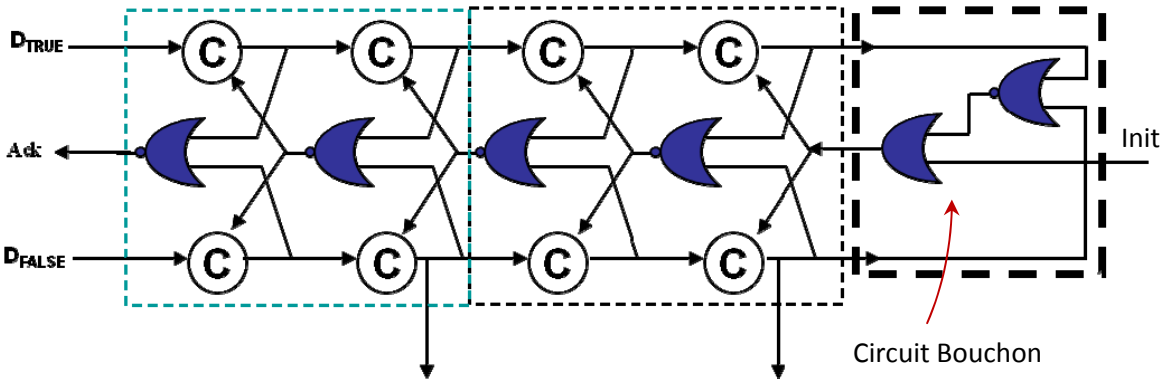


FIGURE 43 : CIRCUIT 'BOUCHON', POUR INITIALISER LA FIFO ASYNCHRONE

4.1.3 BLOC PROGRAMMABLE DE BASE – LUT6

Une LUT est le cœur programmable du FPGA. Elle peut être considérée comme une table de vérité. Elle est composée d'une partie mémoire qui contient les bits de configuration et d'une partie décodeur (cf. Figure 41). Une LUT à base de multiplexeurs est présentée précédemment (cf. Figure 44). D'un point de vue sécurité, cette structure présente plusieurs inconvénients :

- Capacités d'entrées non-équilibrées : La capacité vue par l'entrée A est plus grande que celle de C. Electriquement parlant, cela signifie que la consommation du courant de la LUT dépend de la permutation des entrées.

- Variation de temps : la profondeur logique traversée par les données est différente selon l'activité des entrées. Cette profondeur non constante entraine un temps de calcul dépendant des entrées manipulées. En d'autre terme, les courbes de courant sont différentes selon l'activité des entrées ce qui n'est pas désirable pour la sécurité du circuit.

Dans le cadre de ce travail, une nouvelle architecture est proposée. Le but est d'obtenir une architecture qui préserve les bénéfices des contre-mesures logiques déjà introduit dans le chapitre Chapitre 3. Cette architecture est électriquement équilibrée, et possède des capacités d'entrées égales pour toutes les entrées. Une profondeur logique unique est imposée entre les bits de configuration et la sortie de la LUT. De plus, tous les chemins de données dans le décodeur sont équilibrés pour éviter les variations de temps de calcul. Cela est fait d'une part en utilisant une matrice de *switches* entre les bits de configuration et la sortie de la LUT et, d'autre part, en bien dimensionnant les transistors du décodeur comme cela est présenté dans la suite.

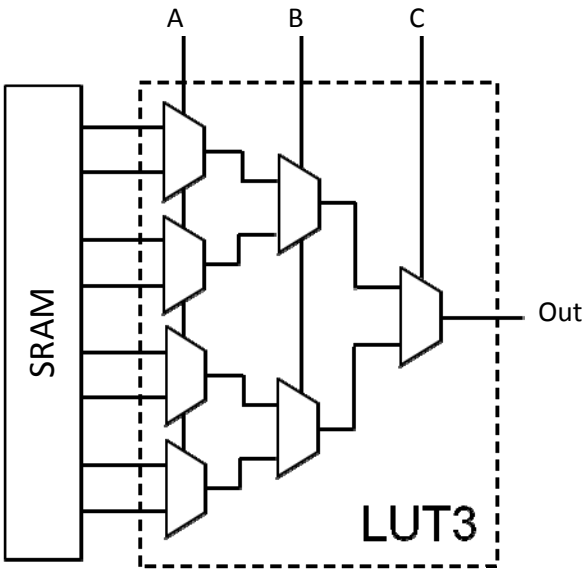


FIGURE 44 : LUT A BASE DE MULTIPLEXEURS

4.1.3.1 Problèmes liés aux aléas

Afin d'éviter le fonctionnement défectueux, un circuit asynchrone doit être sans aléa (J. Sparso, 2006). Pour respecter cette contrainte dans le FPGA asynchrone, une bonne coordination entre les bits de configuration et les combinaisons des entrées du décodeur est nécessaire.

La Figure 45 présente l'architecture finale adoptée de la LUT6. Les sorties du décodeur dec_0, dec_1, …, dec_i, dec_{63} commandent le réseau d'interrupteurs. Chaque interrupteur est connecté à un *Full-Buffer* et donc à un point de configuration (cf. Figure 42). Il permet de propager le bit de configuration auquel il est connecté vers la sortie de la LUT6.

Les interrupteurs ne sont rien d'autres que des *transmission-gates* qui relient les bits de configurations à l'unique sortie de la LUT6. Ces interrupteurs sont mutuellement exclusifs, au sens où un et un seul à la fois peut être fermé lorsque tous les autres sont ouverts. Ainsi un seul bit de configuration se propage vers la sortie.

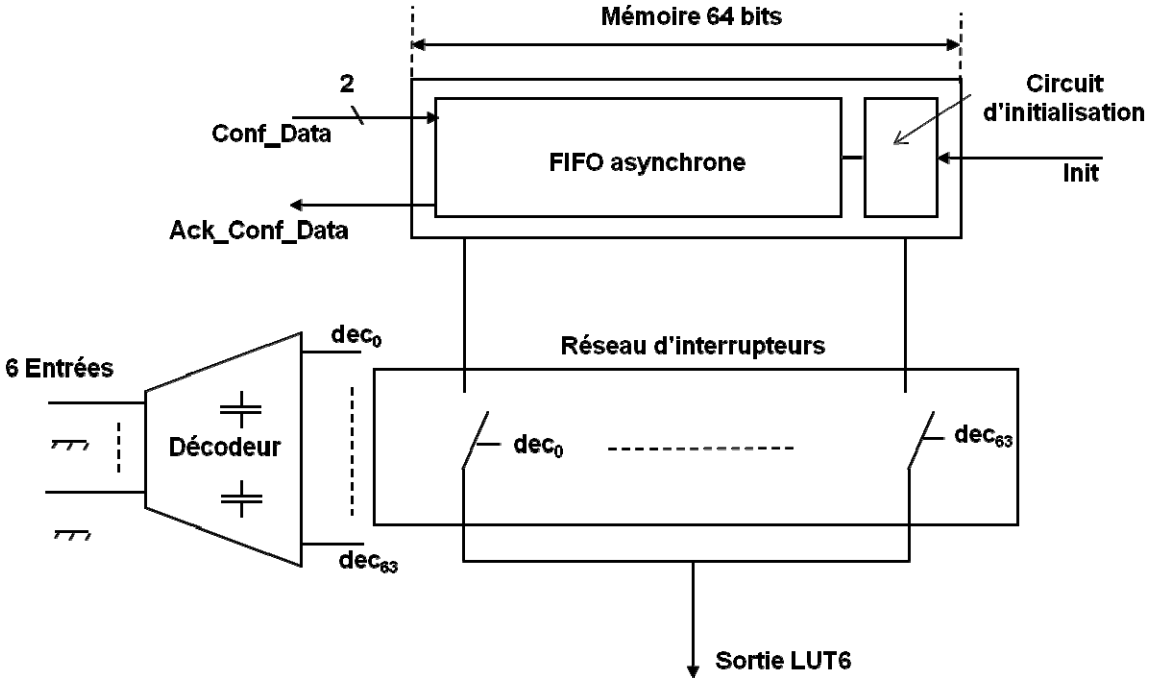


FIGURE 45 : ARCHITECTURE LOGIQUE D'UNE LUT EQUILIBREE

La Figure 46 présente l'architecture logique du décodeur $6 \rightarrow 64$. Ce circuit se charge de décoder les 6 entrées de la LUT6 pour donner accès au bit de configuration souhaité. Tout d'abord les 6 entrées sont doublées. Dans cette étape, le décodeur génère à partir de chaque entrée, deux signaux. Le premier correspond à l'entrée elle-même, il a les mêmes valeurs que celle-ci. Le deuxième est son complémentaire (cf. Figure 46). La deuxième étape est gérée par un bloc combinatoire qui permet d'activer un interrupteur à la fois. En effet, selon la combinaison des bits des 6 entrées, un interrupteur correspondant est fermé. Ce dernier va permettre le transfert du bit de configuration vers la sortie de la LUT6.

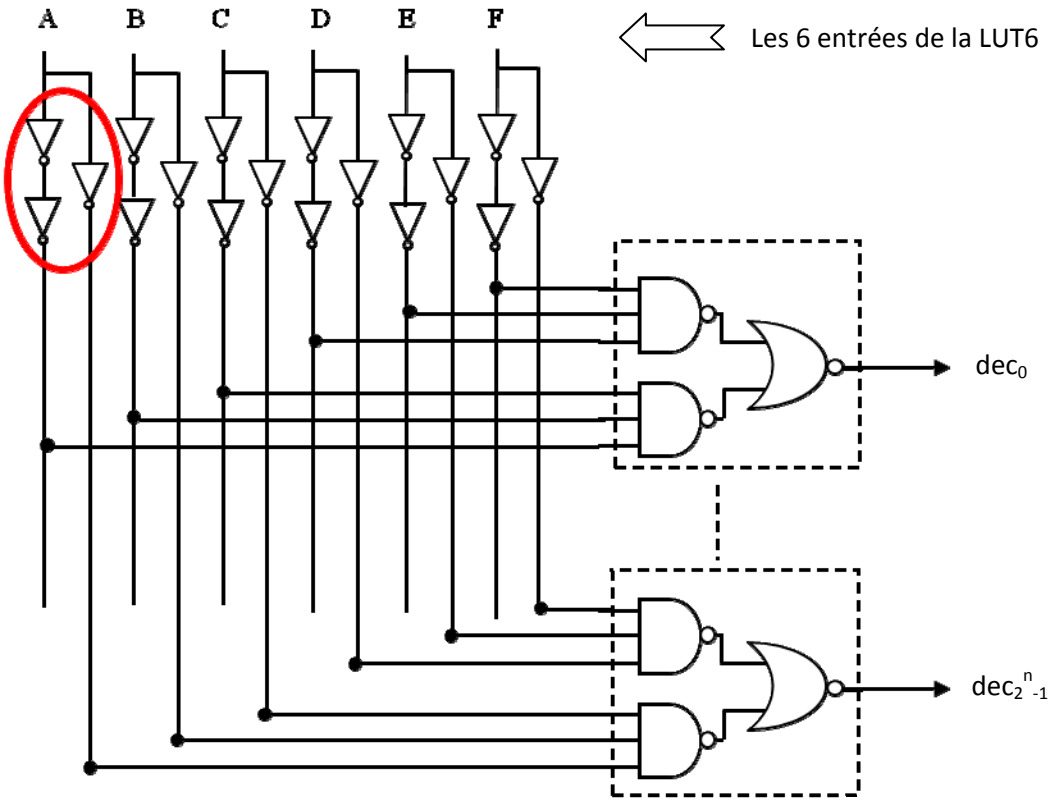


FIGURE 46 : ARCHITECTURE LOGIQUE DU DECODEUR 6→64

Dans le décodeur, à l'issue de l'entré A par exemple, se trouve 2 fils. Sur le premier existe, comme le montre la Figure 46, un inverseur. Il permet de générer le complémentaire de A. Sur le deuxième fil, existe deux inverseurs en séries. La raison pour laquelle, ces deux inverseurs sont ajoutés est principalement pour équilibrer les délais sur les deux fils. Ce point est très important pour garantir une bonne résistance du FPGA aux attaques temporelles. En outre, ceci n'est pas le seul avantage que gagne le FPGA ; en effet le fait de propager chaque entrées avec son complémentaire, crée à l'intérieure de la LUT6, un codage 1 parmi 2 des données. Cela est d'une très grande importance pour plusieurs raisons :

- Il ajoute plus de symétrie électrique à la consommation de la LUT.
- Il tolère l'utilisation du simple rail aux entrées de la LUT6. Ce qui est le cas pour certains protocoles de communication.
- Il permet d'évaluer la résistance du codage simple rail implémenté sur ces LUT6.

Pour conclure, la LUT6, constitue le bloc programmable de base du FPGA asynchrone. Elle possède 6 entrées et une sortie. Elle est construite à partir d'une mémoire asynchrone de

type FIFO et d'un décodeur. Ces deux blocs sont équilibrés logiquement et électriquement comme cela a été présenté précédemment.

4.1.4 Le « Logic-Element » : 'LE'

Le *Logic Element* 'LE ' peut être considéré comme le générateur de fonctions du FPGA. Avec 6 entrées et 4 sorties, il est constitué de deux LUT6 connectées par un multiplexeur (cf. Figure 47) et une porte XNOR. A l'intérieure du 'LE', les entrées des deux LUT6 sont connectées à des multiplexeurs comme il a été recommandé dans le chapitre précédent. Chacun de ces 12 multiplexeurs est nommé $M_kI_lOut_m$. Dans cette nomenclature, chaque terme signifie :

- « M_k » correspond au numéro du multiplexeur dans le PLB, avec $k \in \{0,1,2,3,\dots,11\}$,
- « I_l » est le nom de la première entrée du multiplexeur, elle connectée directement au réseau d'interconnexion du FPGA, avec $l \in \{0,1,2,3,4,5\}$,
- « Out_m » est la deuxième entrée du multiplexeur ; et aussi une sortie d'une des 4 LUT6 du PLB (cf. Figure 48).

Chacun de ces multiplexeurs permet de choisir entre deux options :

- La première est de connecter la LUT6 à une entrée primaire externe, venant du réseau d'interconnexion du FPGA.
- La deuxième est le rebouclage d'un signal interne du PLB à l'entrée de la LUT6. Cela réalise l'effet mémorisation dont les circuits asynchrones ont besoin.

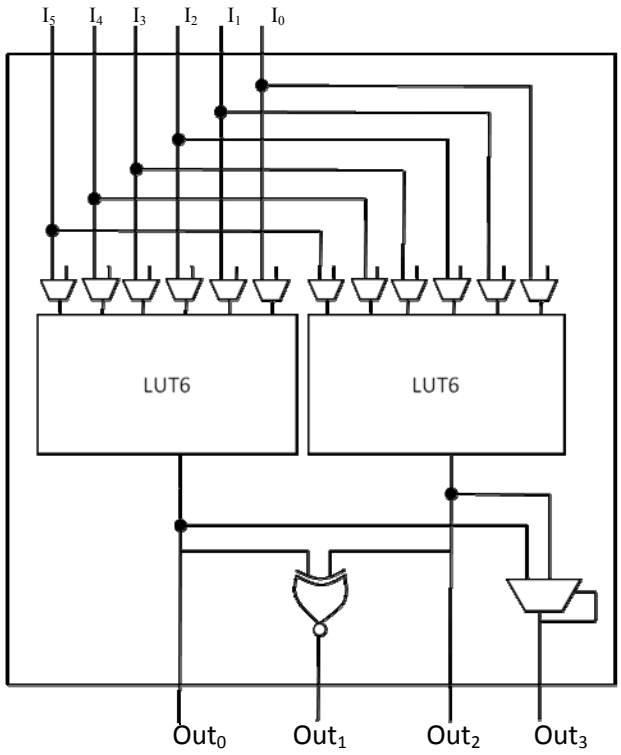


FIGURE 47 : ARCHITECTURE DU *LOGIC ELEMENT* 'LE'

Les rebouclages Out_5, Out_6 et Out_8 proviennent du deuxième *Logic Element du même PLB*.

Tous ces rebouclages au niveau des entrées donnent - comme il a été montré dans le chapitre précédent – plus de flexibilité en lui permettant d'implémenter des fonctions complexes. Pour assurer ce comportement, chaque multiplexeur est contrôlé par un bit de sélection programmable S_k : si S_k est configuré à '0', le rebouclage s'active. Dans le cas contraire une entrée primaire est utilisée, et donc c'est un signal externe qui se connecte à l'entrée de la LUT6 concernée.

Un 'LE' est ainsi capable d'implémenter une fonction à 6 entrées simple rail, ce qui est l'équivalent d'une fonction à 2 entrées en double rails (en prenant en considération le signal d'acquittement et la mémorisation de la sortie). Cette implémentation nécessite l'utilisation des sorties Out_0 et Out_6. Dans ce cas, la sortie Out_1 sert à calculer le signal d'acquittement.

Un 'LE' est aussi capable d'implémenter des fonctions à 3 entrées double rails. Dans ce cas, c'est la sortie Out_1 du premier 'LE' du PLB (cf. Figure 48) qui supporte le premier rail de la fonction. La même sortie correspondante du deuxième 'LE' supporte alors l'implémentation du deuxième rail de la fonction à 3 entrées double rails.

Les équations des sorties s'écrivent :

Out0 = LUT ([Out0 / I_0],[Out2 / I_1],[Out6 / I_2],[Out8 / I_3],],[Out5 / I_4],],[Out0 / I_5])

Out2 = LUT ([Out2 / I_0],[Out0 / I_1],[Out6 / I_2],[Out8 / I_3],],[Out5 / I_4],],[Out2 / I_5])

Out3 = f ([Out0 / Out2], Out3)

$$\overline{Out_1} = \overline{Out_0 \oplus Out_2}$$

Aussi, est-il important de mentionner que les multiplexeurs aux entrées, permettent - outre la mémorisation - de cascader des fonctions à plus de 6 entrées.

4.1.5 Le « Programmable Logic Bloc » – PLB

L'architecture de la Figure 48 présente le PLB comme il a était défini précédemment. C'est une architecture symétrique. Les symétries capacitives des entrées sont bien respectées, ainsi que l'équilibre des profondeurs logiques.

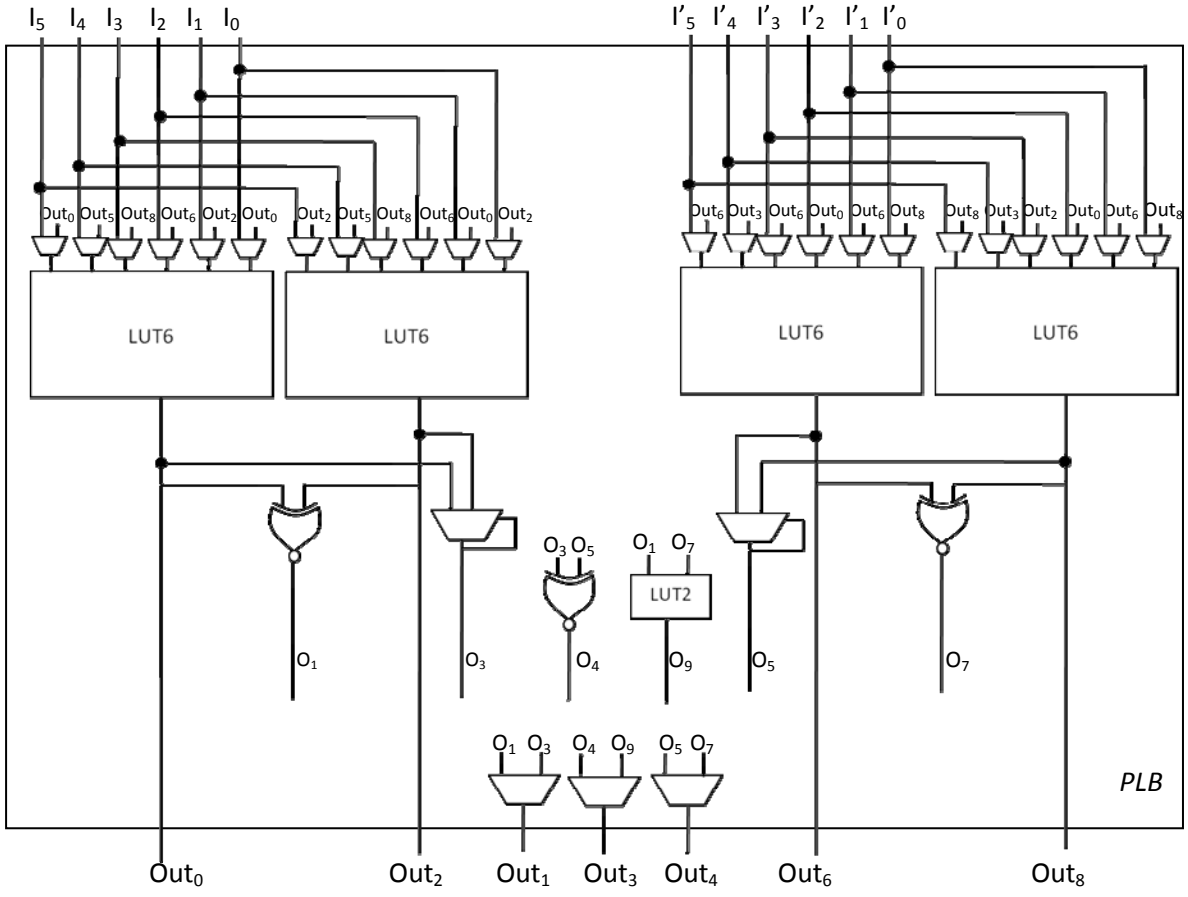


Figure 48 : Architecture du Programmable Logic Bloc PLB

4.1.6 LA « *SWITSH-BOX* » & LA « *CONNECTION-BOX* »

Ces deux blocs ont été développés au sein du Laboratoire COMELEC de Telecom Paris-Tech dans le cadre de la thèse de Sumantha Chaudhuri. Dans son manuscrit, l'auteur utilise la technique de paire torsadée (S. Chaudhuri, 2009). La paire torsadée est une contre-mesure contre la fuite d'information via l'émission électromagnétique. La Figure 49 montre l'avantage de l'utilisation d'une paire de fils torsadés sur l'utilisation d'un routage parallèle traditionnel.

En effet, la paire torsadée peut être considérée comme une succession de rayonnements élémentaires qui s'opposent (cf. Figure 49). Ces émissions s'opposent ainsi, et s'annulent. En outre, l'autre avantage de la paire torsadée est la réduction de l'effet *cross-talk*.

Dans la Figure 49, les deux signes ⊙ et ⊕ , indiquent deux sens de champs opposés.

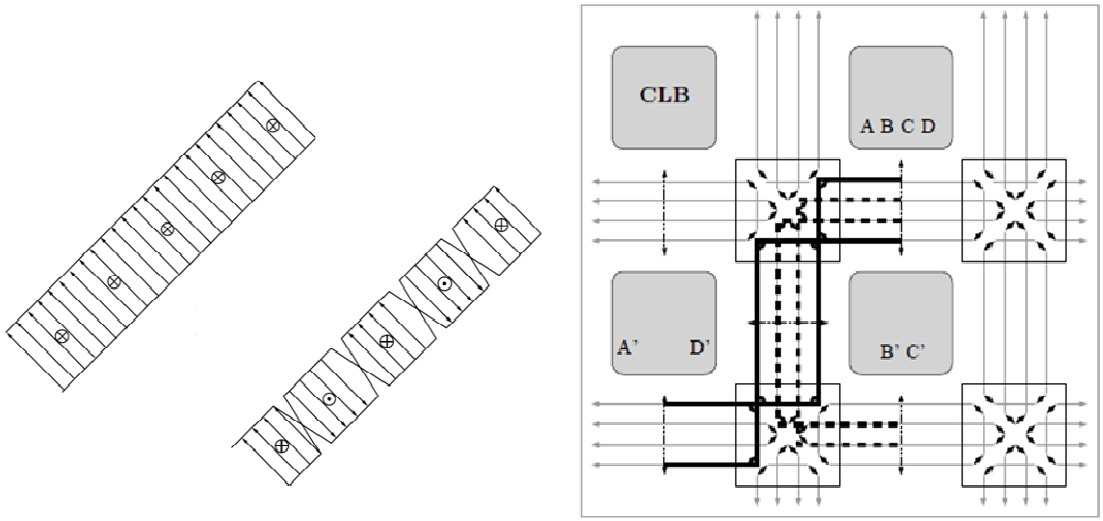


FIGURE 49 : A DROITE, LE CHAMP ELECTROMAGNETIQUE DANS UNE PAIRE TORSADEE ET DANS UNE PAIRE PARALLELE - A GAUCHE DEUX FILS EQUITEMPORELS GENERES PAR UNE SWITCH-BOX A PAIRE TORSADEE.

L'auteur a présenté deux types de *Switch-Box*, basés sur le principe de paires de torsadées :

- La « *Twist on turn switch box* » : l'idée est que chaque paire d'un signal n-rail sort de la *Switch-box* torsadée, si elle change de direction.
- La « *Twist always switch box* » : Dans ce cas une paire de fils est toujours twistée à la sortie de la *Switch-box* (cf. Figure 50).

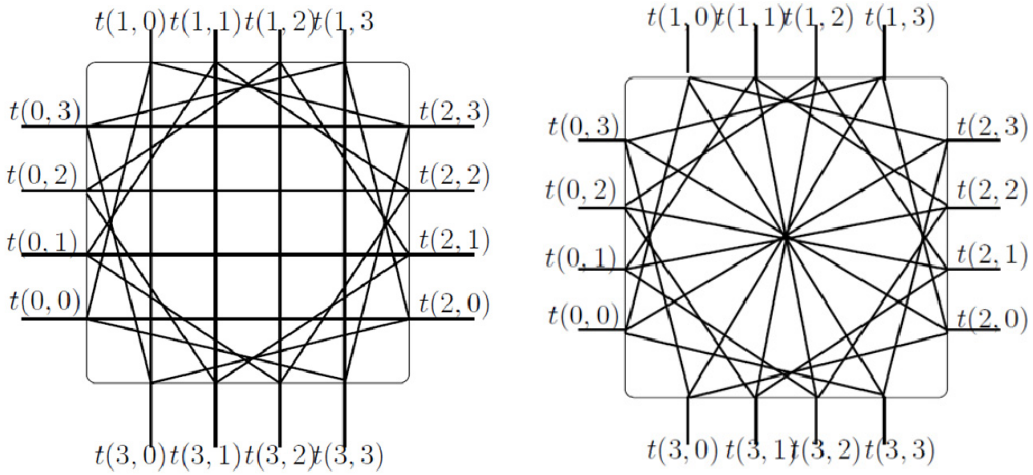


FIGURE 50 : A DROITE: *TWIST-ON-TURN.* A GAUCHE : *TWIST-ALWAYS* .

Pour plus d'information voir (S. Chaudhuri, 2009).

4.2 ARCHITECTURE ELECTRIQUE

Cette partie présente l'architecture électrique au niveau transistor des différents blocs constituant le FPGA asynchrone. Tous les transistors ont été dimensionnés manuellement, en utilisant la méthode « logical effort » (I. Sutherland, B. Sproull and D. Harris, 1999), pour respecter toutes les contraintes électriques détaillées précédemment dans le manuscrit et pour équilibrer la consommation et le temps de calcul du FPGA. A savoir que la conception du PLB est *Full custom*, tandis que la partie interconnexion est basée sur des cellules standard de ST avec un routage manuel pour équilibrer le réseau d'interconnexion. La technologie utilisée est la CMOS 65 nm de *ST Microelectronics.*

4.2.1 CHAINE DE PROGRAMMATION ASYNCHRONE

Une LUT6, contient 64 points de programmation. Donc il lui faut 64 *Full Buffer*, avec un nombre total de transistors = 2304 dont 10 pour le circuit bouchon (cf. Figure 51 & Figure 52). Le dimensionnement de ces transistors prend en compte principalement les performances du circuit. Les résultats du test sont présentés dans le chapitre suivant. La Figure 51 présente un *Half-Buffer*. Pour faire un *Full Buffer* asynchrone, il suffit de mettre deux *Half-Buffer* en série. Le premier sera chargé de stocker un bit de programmation en double rail. Le rôle du deuxième est plutôt d'assurer le bon fonctionnement du protocole 4 phases dans la chaine complète, en stockant des bits (0,0).

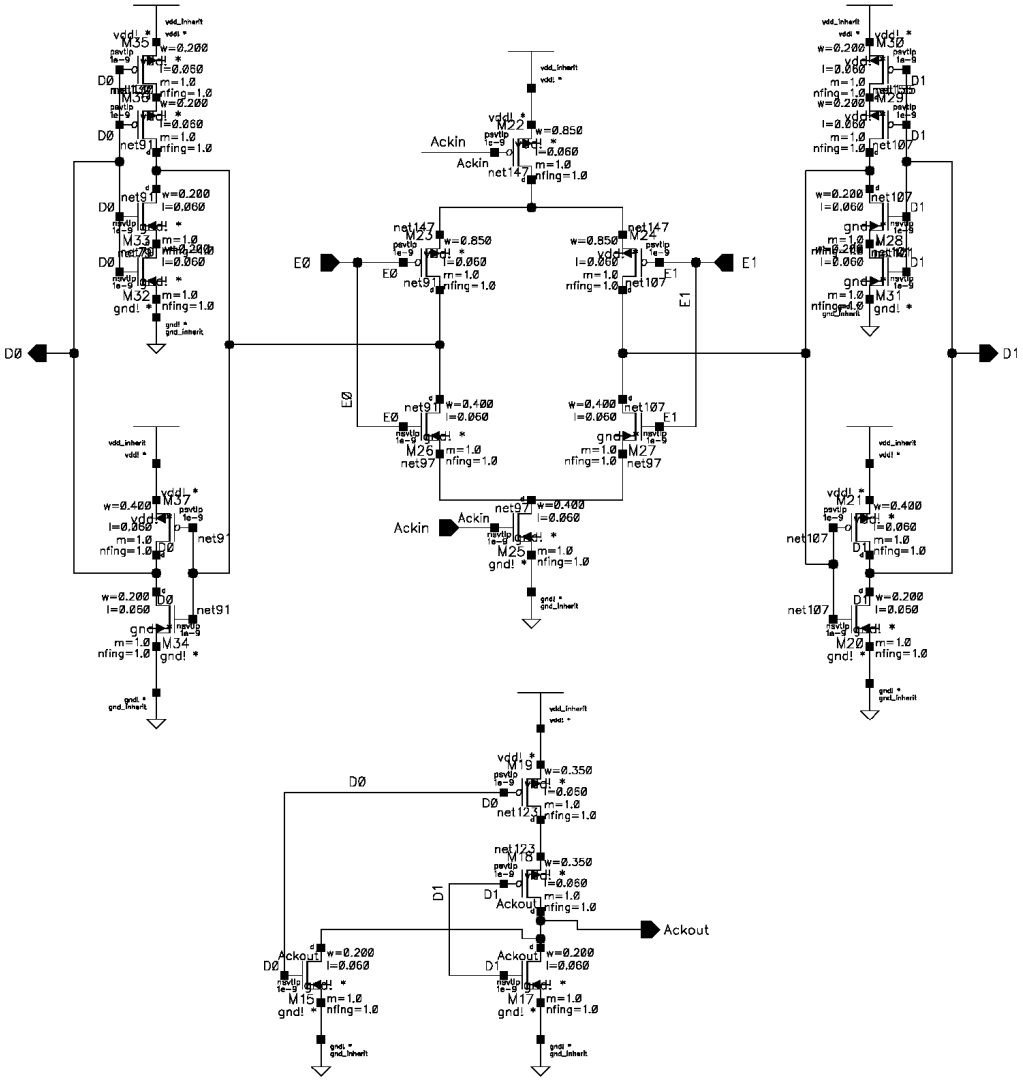


FIGURE 51 : ARCHITECTURE ELECTRIQUE DU *HALF BUFFER* ASYNCHRONE

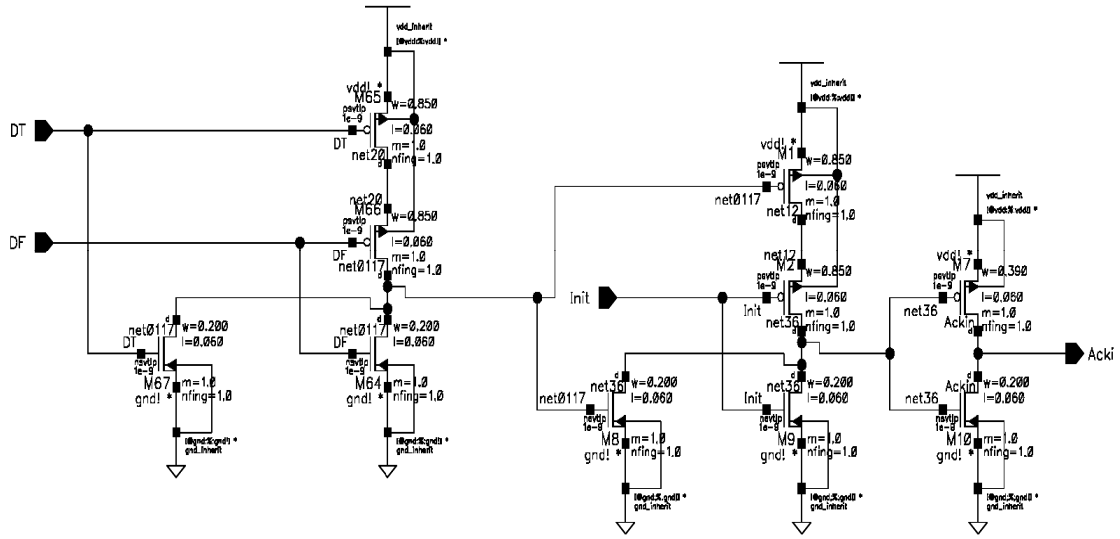


FIGURE 52 : ARCHITECTURE ELECTRIQUE DU CIRCUIT BOUCHON

4.2.2 BLOC PROGRAMMABLE DE BASE – LUT6

La Figure 53 présente la première entrée du décodeur de la LUT6. Ce décodeur possède 6 entrées identiques. Les dimensions des 3 inverseurs sur les deux signaux générés à partir de chaque entrée garantie deux choses : la première est un délai identique sur les deux fils, et la deuxième est la symétrie capacitive de toutes les entrées de la LUT6.

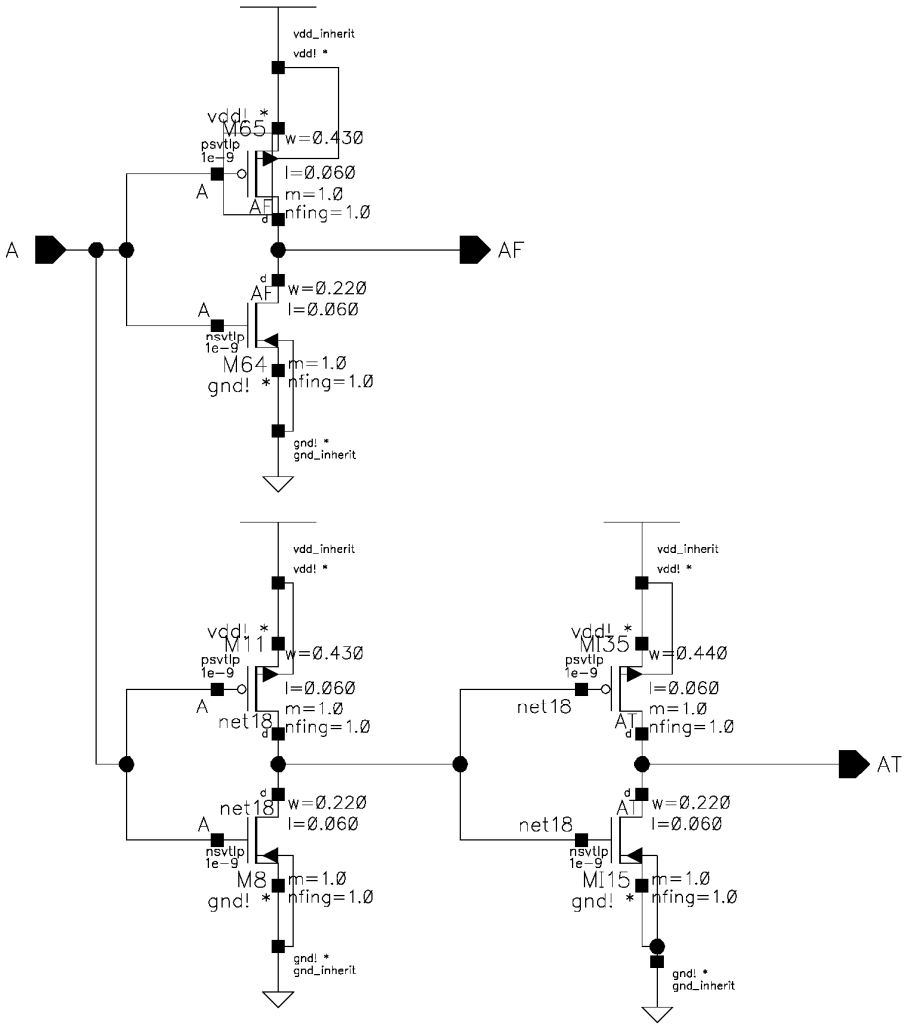


FIGURE 53 : ARCHITECTURE ELECTRIQUE D'UNE DES 6 ENTREES DU DECODEUR DE LA LUT6

4.3 Le Layout Du FPGA

Cette partie présente le LAYOUT de principaux sous-blocs du PLB asynchrone. Le LAYOUT de ces sous-blocs est également *Full Custom*. Les lignes de métal à l'intérieur de chaque partie sont équilibrées de façon à avoir une même longueur de chemin pour toutes les données.

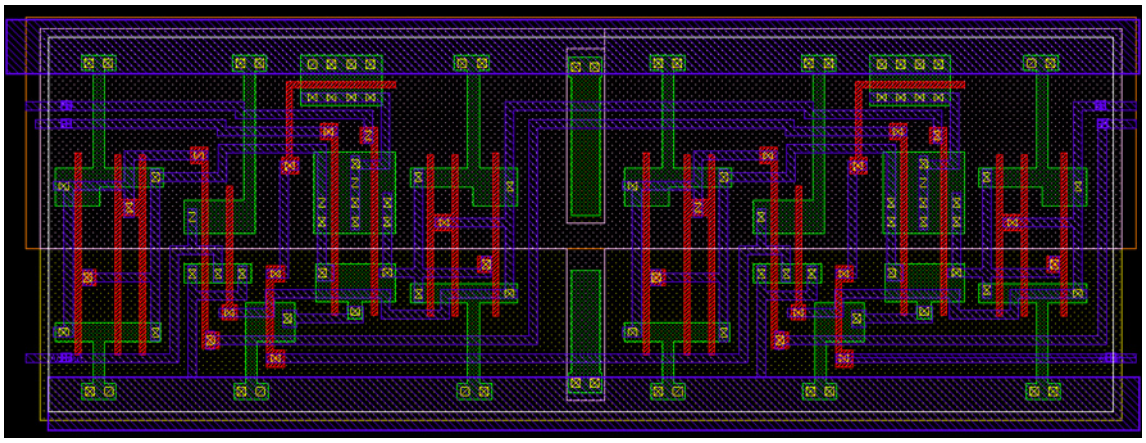


FIGURE 54 : *LAYOUT* DU *FULL BUFFER* ASYNCHRONE

Les photographies d'écran suivantes correspondent toutes à un bloc programmable PLB. Chacune d'entre elles correspond à un niveau de métallisation. Le but est de mettre en évidence les efforts qui ont été faits pour équilibrer les longueurs des interconnexions (même si ce n'est pas complètement évident sur un seul niveau de métal) et de montrer la symétrie des sous-circuits constituant le PLB.

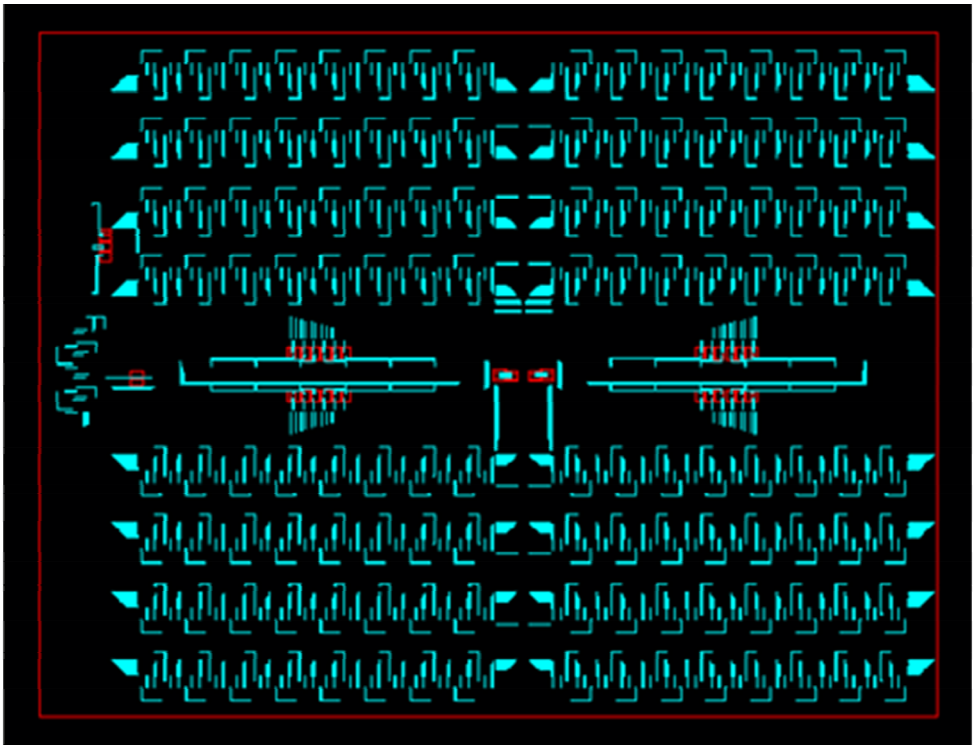


FIGURE 55 : *LAYOUT* DU PLB – NIVEAU METAL 2

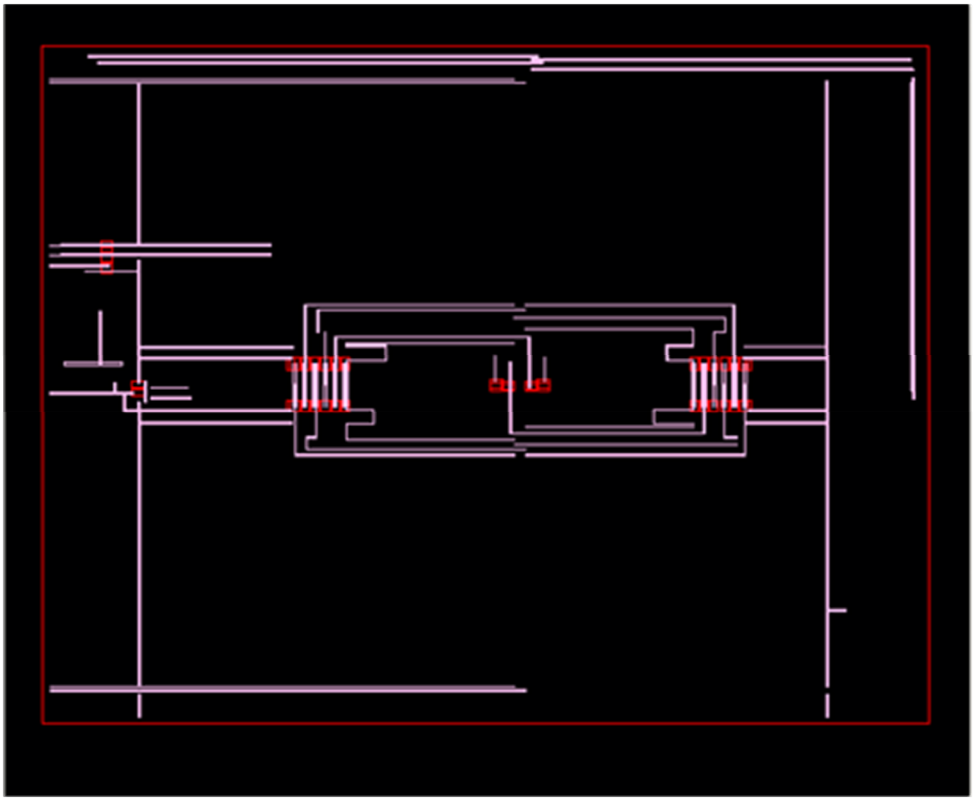


FIGURE 56 : *LAYOUT* DU PLB –NIVEAU METAL 3

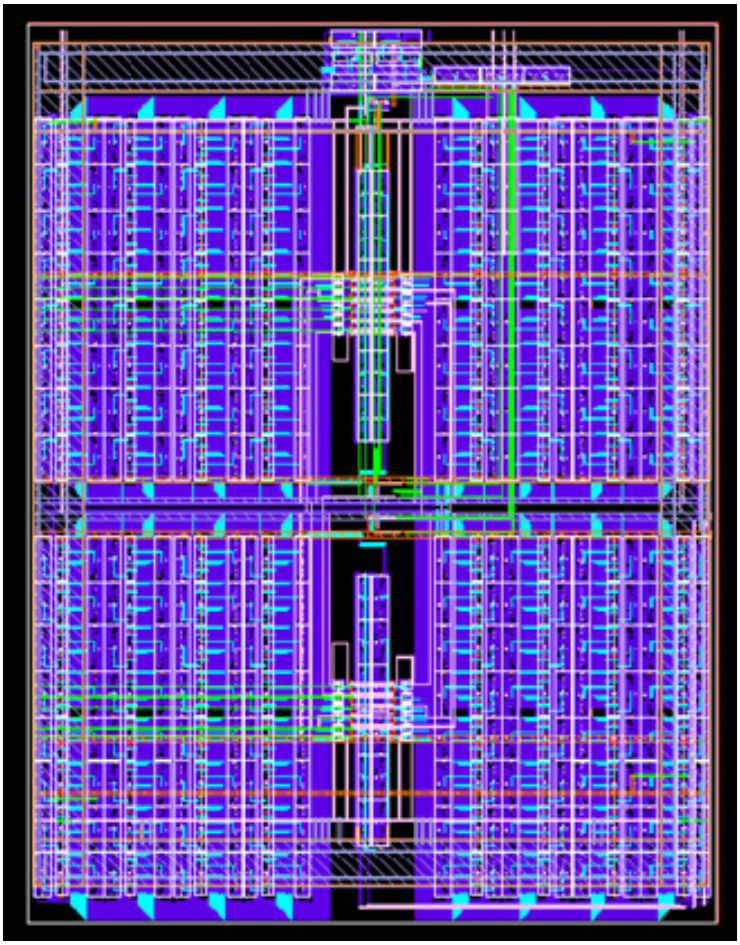


FIGURE 57 : *LAYOUT* DU PLB COMPLET.

4.3.1 LE FPGA ASYNCHRONE : LAYOUT ET DIE VIEW

Le prototype fabriqué est de taille 3x3. Il comporte donc 9 PLBs. Il a été fabriqué en utilisant la technologie CMOS 65 nm de ST Microelectronics. Le FPGA possède 36 Entrées/Sorties reparties sur les quatre côtés. Il occupe une surface de 1111.6 μm x 947.6 μm et contient environ 200,000 transistors.

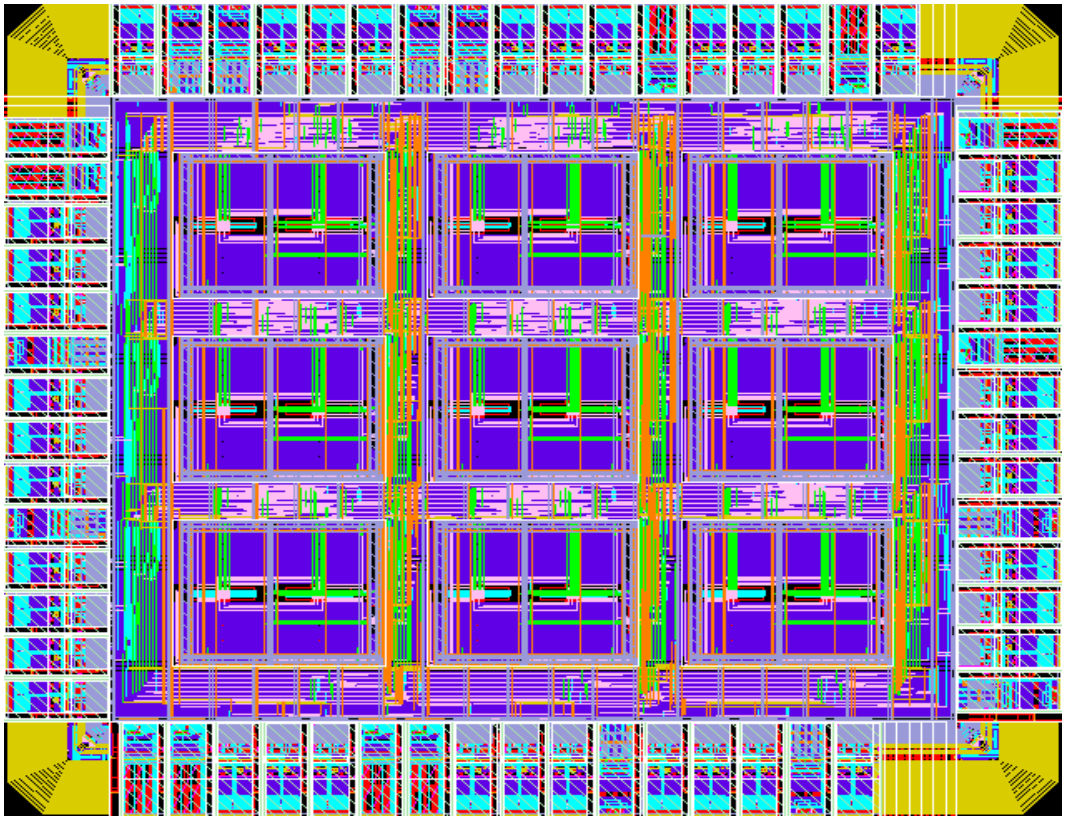


FIGURE 58 : *LAYOUT* DU FPGA ASYNCHRONE

4.4 CONCLUSION

Dans cette partie, l'architecture électrique du PLB a été présentée. C'est une architecture qui a été implanté en *full custom* (heureusement qu'un FPGA est une répétition de motifs !) afin de garantir un équilibrage optimal. Le circuit de programmation du FPGA est constitué d'une FIFO asynchrone commandée par un décodeur bien spécifique. L'utilisation de la logique asynchrone pour programmer le FPGA n'est pas lié à la nature asynchrone du FPGA, mais a été faite pour des raisons de robustesse, notamment parce qu'elle procure au circuit une meilleure résistance aux variations de température, de tension, etc.

Le prototype a été fabriqué en technologie CMOS 65 nm de ST Microelectronics. Les transistors ont été dimensionnés avec la méthode dite *Logical effort* pour garantir un temps de propagation constant. Enfin les sous blocs ainsi que les fils d'interconnexion ont tous été équilibrés manuellement dans ce même but

Chapitre 5. RÉSULTATS EXPÉRIMENTAUX – TESTS ET VALIDATION

Après avoir présenté dans les chapitres précédents les spécifications du FPGA asynchrone sécurisé, ainsi que son architecture électrique correspondante, ce chapitre traitera du test et de la validation de cette architecture sur deux plans : fonctionnelle et sécuritaire. Le premier étant pour vérifier si le FPGA permet d'implémenter et d'exécuter avec succès une fonction donnée. Le deuxième sert à valider l'aspect sécuritaire du FPGA. Les tests préliminaires sont faits en utilisant les simulateurs d'ELDO et CADENCE. Ensuite un test de validation est réalisé sur le prototype après fabrication.

Dans la suite du chapitre le taux de remplissage est un critère qui a été défini pour évaluer le taux d'utilisation des ressources programmable dans le FPGA.

$$taux\ de\ remplissage = \frac{nombre\ d'entrées\ utilisées\ d'un\ bloc}{nombre\ totale\ d'entrées}$$

La Figure 59 présente l'architecture finale du bloc programmable du FPGA. Il a été démontré précédemment que cette architecture est capable d'implémenter une fonction qui possède maximum 7 variables en entrées dont une sert à la mémorisation.

Soit le cas d'un additionneur complet 2 bits double rails QDI. Ses équations de sorties s'écrivent :

$$FA_Out_0 = f_1(A_0, A_1, B_0, B_1, C_0, C_1, FA_Out_0^{-1})$$

$$FA_Out_1 = f_2(A_0, A_1, B_0, B_1, C_0, C_1, FA_Out_1^{-1})$$

$$FA_C_0 = f_3(A_0, A_1, B_0, B_1, C_0, C_1, FA_C_0^{-1})$$

$$FA_C_1 = f_4(A_0, A_1, B_0, B_1, C_0, C_1, FA_C_1^{-1})$$

ÉQUATIONS 14

Chaque équation s'implémente sur deux LUT6 jumelles (c.à.d. d'un même LE). Ainsi l'additionneur nécessite 2 PLBs pour être implémenté. Le taux de remplissage est égal à 100% pour les 2 PLBs.

Lorsque le nombre d'entrées d'une fonction dépasse '7', une décomposition de cette dernière en sous fonctions devient inévitable pour assurer son implantation sur les PLBs du FPGA. Chaque sous fonction ne doit bien évidemment pas dépasser le nombre maximum d'entrées : 7.

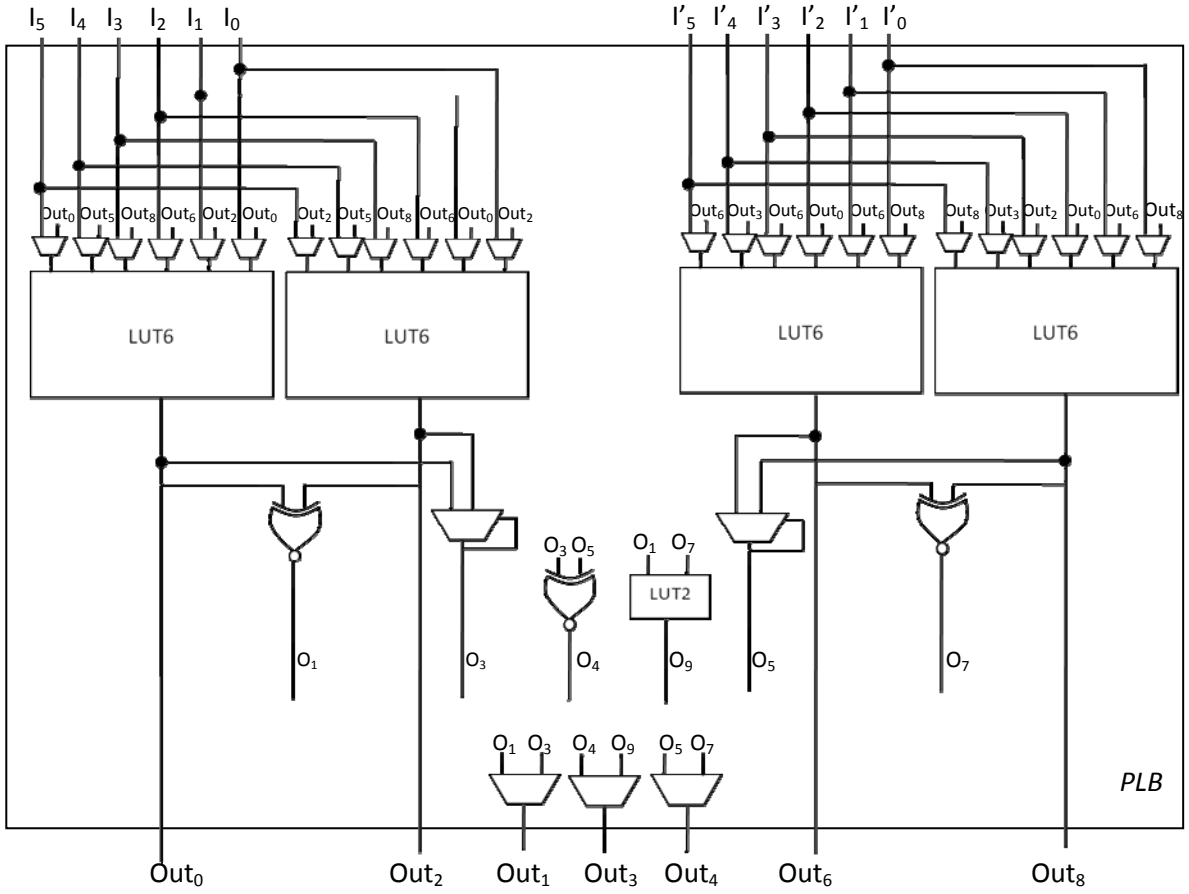


FIGURE 59: PLB BLOC PROGRAMMABLE DU FPGA

C'est le cas des fonctions factorisables, où la sortie peut être calculée en plusieurs étapes. Pour rendre les choses plus claires, soit l'équation F suivante :

$$F = f(A0, A1, B0, B1, C0, C1, D0, D1, E0, E1, F\text{-}1) \quad \text{ÉQUATION 15}$$

Pour que celle-ci réponde aux critères ci-dessus, il faut être capable de la diviser en 2 sous fonctions différentes, et les concaténer pour réaliser la fonction finale :

$$F' = f(A0, A1, B0, B1, C0, C1, F'\text{-}1) \quad \text{ÉQUATION 16}$$

$$F = f(F', D0, D1, E0, E1, F\text{-}1) \quad \text{ÉQUATION 17}$$

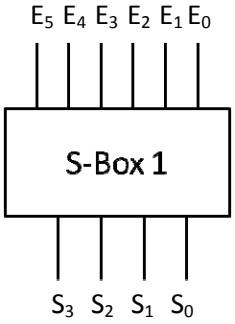
La première fonction s'implémente ainsi sur deux LUT6 jumelles, alors que la deuxième nécessite une seule LUT6. La fonction F finale requiert donc 1 PLB pour être implémentée avec un taux de remplissage de 100%, sans avoir besoin du réseau d'interconnexion du FPGA. Une fonction plus grande que F nécessitera bien entendu plus de ressources programmables, voire même l'utilisation du réseau d'interconnexion.

Certaines fonctions ne répondent pas à ces critères. Elles n'acceptent pas d'être décomposées en sous fonctions, comme c'est le cas par exemple de la fonction de substitution de l'algorithme DES : la S-Box[30]. Ce cas sera détaillé dans la suite.

5.1 CAS D'UNE S-BOX ASYNCHRONE

La S-box est une fonction qui contribue à la « confusion » en rendant l'information originale inintelligible. Elle permet de casser la linéarité de la structure de chiffrement. Elle est utilisée dans différents algorithmes de chiffrement comme le DES et l'AES. Elle prend en général une variable de m bits en entrée et produit une sortie de n bits, les entrées et les sorties n'ont pas forcément la même taille.

Une S-Box est en réalité une matrice de n colonnes et m lignes. Sa taille varie d'un algorithme de chiffrement à l'autre. Les valeurs de cette matrice sont choisies de manière à éviter certains types d'attaques. Dans un DES, cette table possède 4 lignes et 16 colonnes. Elle est représentée par une fonction à 6 entrées et 4 sorties (cf. Figure 60). Il existe par exemple 8 S-Box dans un algorithme DES.



Ligne/ Colonne	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

FIGURE 60: S-BOX1 DE L'ALGORITHME DES

La sortie de la S-Box d'un DES est codée sur 4 bits. Elle est calculée à partir des 6 entrées de la façon suivante : les 6 entrées se divisent en deux parties. Le premier et le dernier bit d'entrée sert à déterminer le numéro de la ligne correspondante dans la matrice. Les quatre bits du milieu contribuent à la détermination de la colonne. Ainsi la valeur présente dans cette case sera transmise à la sortie. Cette valeur est codé sur 4 bits, et peut donc aller de 0 jusqu'à 15. Les 4 équations de la sortie de la S-Box s'écrivent :

[30] Substitution box

$$S_0 = f_0(A, B, C, D, E, F)$$

$$S_1 = f_1(A, B, C, D, E, F)$$

$$S_2 = f_2(A, B, C, D, E, F)$$

$$S_3 = f_3(A, B, C, D, E, F)$$

ÉQUATIONS 18

Chaque S-Box est connectée par ses entrées à une fonction Ou Exclusive. Cette dernière traite les données codées sur 48 bits, avec une sous-clé de 48 bits aussi. Le sous-circuit S-Box + XOR est considéré, pour cette raison, un point sensible dans l'algorithme. Il est ainsi susceptible de fournir des informations concernant la clé de chiffrement à travers les canaux auxiliaires. Ce circuit est utilisé dans le cadre de ce travail afin de valider l'approche sécuritaire du FPGA.

5.1.1 IMPLEMENTATION EN 4 PHASES, DOUBLE RAILS

La Figure 61 montre le circuit S-Box + XOR testé en protocole 4 phases. Les équations de sorties s'écrivent en double rail comme suit :

$$S_{00} = f_{00}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{00}^{-1})$$

$$S_{01} = f_{01}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{01}^{-1})$$

$$S_{10} = f_{10}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{10}^{-1})$$

$$S_{11} = f_{11}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, \mathrm{F}_0, F_1, S_{11}^{-1})$$

$$S_{20} = f_{20}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{20}^{-1})$$

$$S_{21} = f_{21}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{21}^{-1})$$

$$S_{30} = f_{30}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{30}^{-1})$$

$$S_{31} = f_{31}(A_0, A_1, B_0, B_1, C_0, C_1, D_0, D_1, E_0, E_1, F_0, F_1, S_{31}^{-1})$$

ÉQUATIONS 19

Le codage de données 3-états est utilisé dans cet exemple.

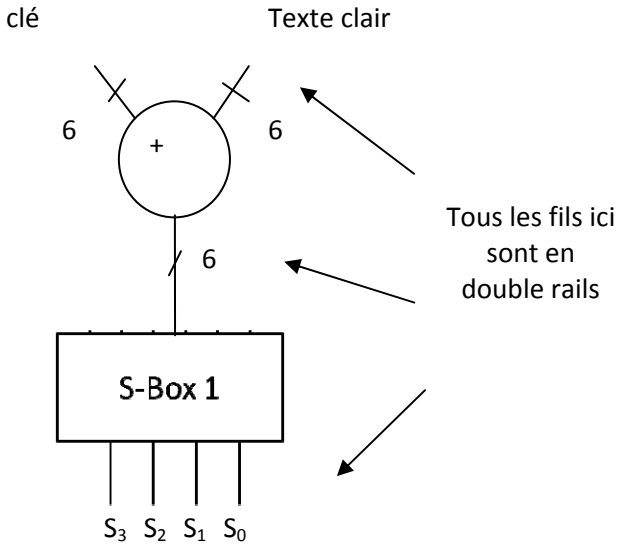


FIGURE 61: CIRCUIT S-BOX + XOR : 12 ENTREES DOUBLE RAILS, ET 4 SORTIES DOUBLE RAILS

La nature de ces fonctions impose la présence de toutes les variables en même temps sur les entrées d'une LUT, afin de calculer la valeur de la sortie. Elles ne peuvent pas être divisées en sous fonctions plus petites. Son implémentation sur l'architecture des blocs programmables du FPGA est ainsi très coûteuse autant au niveau de ressources programmables qu'au niveau sécurité. En effet, chaque fonction nécessite la concaténation de plusieurs PLBs, pour pouvoir construire une grande FIFO pour contenir la matrice de la S-Box. Cette concaténation va aussi déséquilibrer le temps de propagation des variables d'entrées dans le circuit, et donc le rendre vulnérable aux attaques.

Cette problématique est causée par le nombre limité d'entrées primaires de la LUT6 du PLB (6 entrées par LUT6). En effet, compte tenue de l'aspect complémentaire des deux rails, chacun d'eux tient en lui-même la valeur du bit logique qu'il représente : soit par exemple les deux rails (A_0, A_1) = (0,1) qui représentent le bit logique A = 1. Le rail A_1 possède la même valeur que 'A', tandis que la valeur de A_0 est l'opposée. Cette propriété, peut être prise en considération pour facilité l'implémentation de tels types de fonctions sur le FPGA.

Dans ce contexte un algorithme a été utilisé pour permettre non seulement l'implémentation de la S-Box sur le FPGA mais aussi sa sécurité. Il est basé sur une méthode de projection technologique qui conserve le bénéfice de la sécurisation du matérielle et du routage du FPGA. Il permet aussi de générer le *bitstream* de la fonction à implémenter.

L'utilisation de cet algorithme concerne surtout les fonctions qui sont du même type que la S-Box (fonction matricielle non linéaire), et qui ont un nombre de variables supérieur à 7. Il profite du fait qu'en double rails, un des deux rails possède toujours la même valeur lo-

gique du bit représenté. Cela permet de faire le calcul de la valeur de la sortie en n'utilisant qu'un seul rail à la fois. C'est ainsi qu'il diminue de moitié le nombre de variables sur les entrées de la LUT6 et rend possible l'implémentation d'une fonction de grande taille sur les LUT6 sans être obligé de la diviser en sous fonctions et de perdre ainsi l'avantage sécuritaire du FPGA. Les différentes étapes de cet algorithme sont présentées ci-dessous, elles sont ensuite détaillées par un exemple concret d'implémentation.

Les étapes principales de l'algorithme :

- Vérifier le nombre d'entrées de la fonction
- Identifier les différents rails et les séparer en deux groupes : pairs et impairs
- Utiliser chaque groupe à part pour calculer la valeur de la sortie
- Reconstruire le protocole de communication
- Calculer les valeurs finales des sorties & équilibrer la profondeur logique des entrées si nécessaire

L'application de ces étapes pour implémenter la S-Box se fait de la façon suivante :

I. Vérification du nombre d'entrées de la fonction

La S-box possède 4 sorties double rails, donc 8 fonctions. Chacune répond au modèle suivant :

$$F = f(E_{10}, E_{11}, E_{20}, E_{21}, E_{30}, E_{31}, E_{40}, E_{41}, E_{50}, E_{51}, E_{60}, E_{61}, F^{-1})$$

ÉQUATION 20

Elle possède 13 variables dont une est une mémorisation. Chaque fonction de sortie de la S-Box répond donc au premier critère et peut être implémentée sur le FPGA.

II. Identifier les différents rails et les séparer en deux groupes : pairs et impairs

Les variables de l'***Équation 20*** sont codés en double rails, elles peuvent être séparées en deux groupes :

- rails pairs → $\xi_p = \{E_{10}, E_{20}, E_{30}, E_{40}, E_{50}, E_{60}\}$, et
- rails impairs → $\xi_i = \{E_{11}, E_{21}, E_{31}, E_{41}, E_{51}, E_{61}\}$.

Les rails impairs ont les mêmes valeurs que leurs bits logiques correspondants. En revanche les valeurs des rails pairs sont inversées.

III. Utiliser chaque groupe à part pour calculer la valeur de la sortie

Chacun des deux groupes possède maintenant 6 variables. Il est capable de calculer la valeur d'une des 8 sorties de la S-Box. Les rails pairs et impairs sont tous deux utilisés, pour conserver la symétrie électrique du circuit. La Figure 62 illustre d'une manière simplifiée cette implémentation.

Les sorties intermédiaires f'_{ij} possèdent les mêmes valeurs logiques des sorties S_{ij} sans tenir compte du protocole de communication, qui est ici le protocole 4 phases. Cela veut dire que les LUT6 calculent les valeurs f'_{ij} , sans prendre en considération s'il s'agit de données invalides (c.-à-d. d'une mémorisation de l'état précédent de la sortie par exemple) ou d'une remise à zéro, etc.

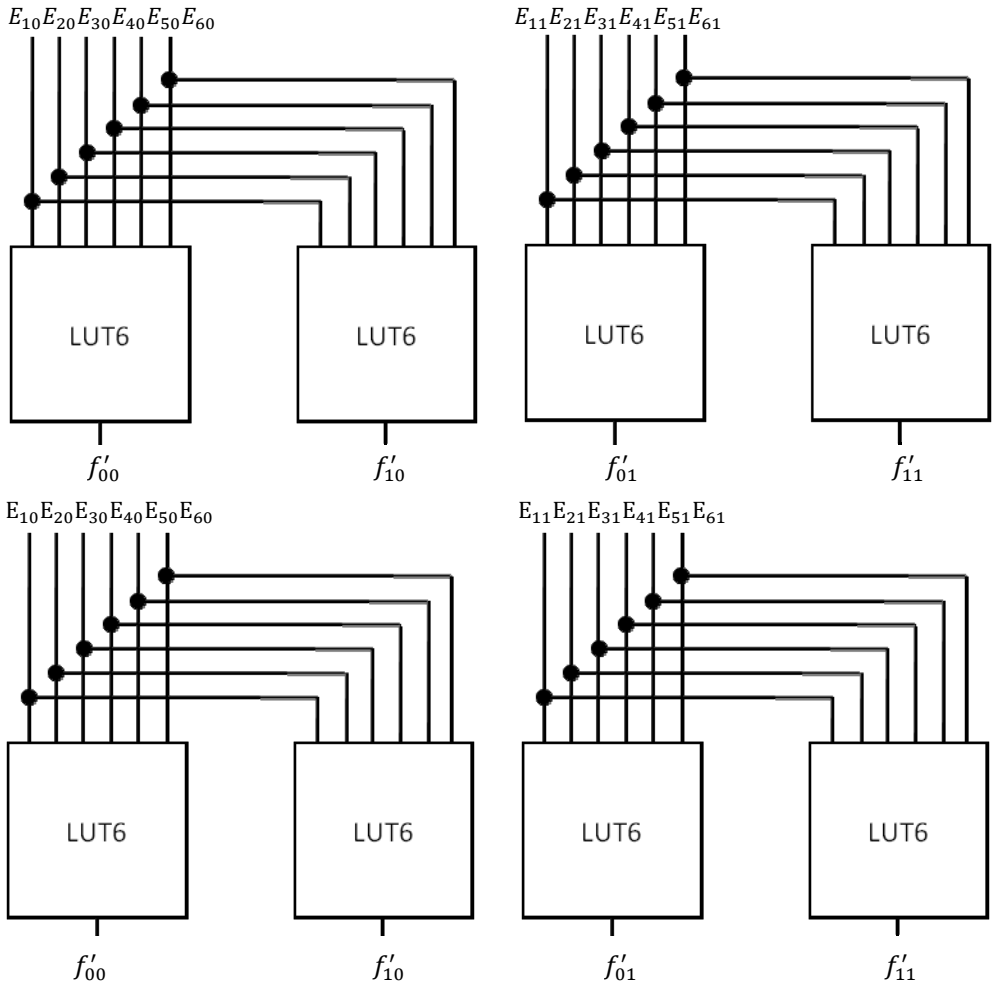


FIGURE 62: CALCUL DES VALEURS DES 8 EQUATIONS DES SORTIES INTERMEDIAIRES DE LA S-BOX

IV. Reconstruire le protocole de communication

Le protocole 4 phases contrôle la communication entre les différents blocs du circuit, par l'identification des états des données traitées à chaque instant. Cette information est normalement fournie par le couple de deux rails de chaque bit logique. La séparation de ses derniers dans l'étape précédente a cassé cette information. Cette étape a pour but de reconstruire cette information et d'identifier les cas suivants :

- Remise à zéro de toutes les entrées : $\xi_p = 0 \;\; et \;\; \xi_i = 0$
- Données valides : $\xi_p = 0 \;\; ou \;\; \xi_i = 0$
- Une ou plusieurs pairs invalides $\begin{cases} \xi_p \neq 0 \;\; et \;\; \xi_i \neq 0 \\ et \\ \xi_p \neq 1 \;\; et \;\; \xi_i \neq 1 \end{cases}$

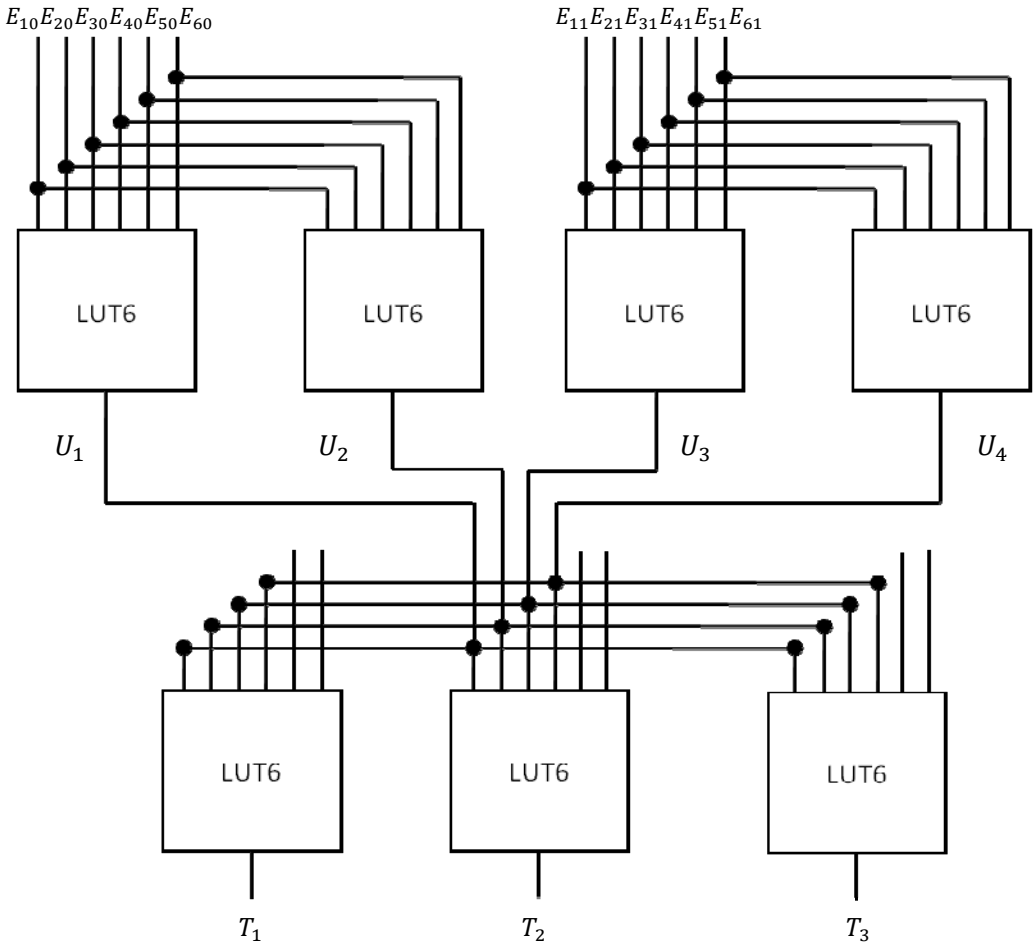


FIGURE 63: BLOC DE TEST DU PROTOCOLE 4 PHASES

La Figure 63 illustre dune manière simplifiée l'implémentation de cette vérification. Ce bloc permet à la fois de vérifier l'état des données traitées et de décider de la bonne valeur des sorties de la S-Box. Ainsi, il génère un signal pour commander la remise à zéro des sorties de la S-Box lorsque les entrées sont toutes à zéro, la mémorisation de l'état précédent des sorties si une des entrées double rails est invalide, et la transmission des valeurs f'_{ij} aux sorties de la S-Box lorsqu'il s'agit de données valides sur ses entrées. Il est important de noter que les trois sorties T_1 , T_2, T_3 de ce bloc utilisent le codage en 1 parmi 3, ce qui signifie un poids de Hamming constant ; même les quatre signaux intermédiaires de ce bloc U_1 , U_2, U_3, U_4 sont codés selon le codage 1 parmi 4.

V. Calculer les valeurs finales des sorties et équilibrer la profondeur logique des entrées si nécessaire

Finalement cette étape rassemble le protocole de communication et les sorties intermédiaires f'_{ij} dans un bloc pour fournir les sorties finales S_{ij} de la S-Box. Pour le réaliser une LUT6 par sortie est nécessaire (cf. Figure 64).

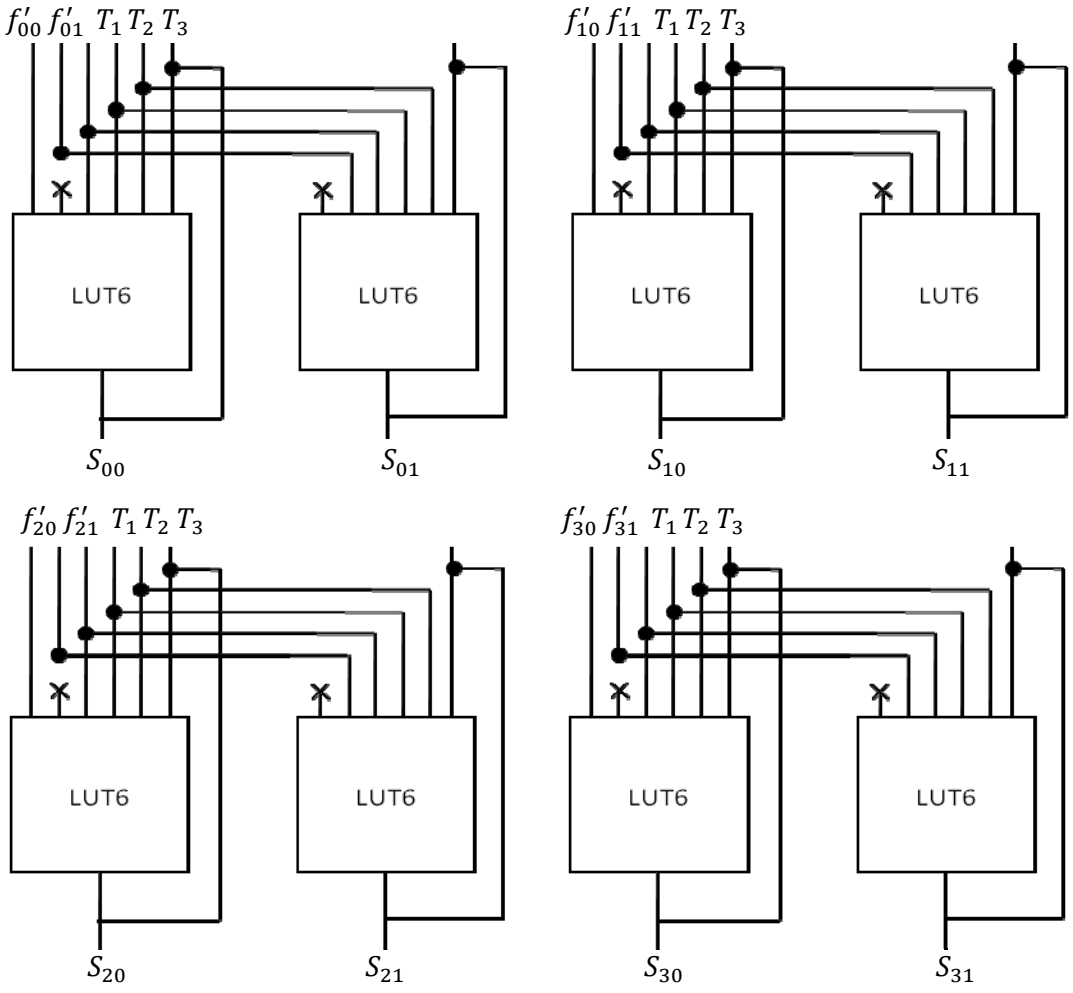


FIGURE 64: BLOC FINAL DE CALCUL DES SORTIES DE LA S-BOX

Une dernière remarque très importante à mentionner, est que les signaux intermédiaires f'_{ij} traversent chacun une LUT6 pour atteindre le dernier bloc, alors que les signaux de contrôle T_k traversent deux LUT6s. Pour conserver la symétrie de temps de propagation et éviter les *glitches* (non acceptables en logique asynchrone), une LUT6 est ajoutée par signal intermédiaire. Cette implémentation nécessite donc 31 LUT6s, ce qui est équivalent à 8 PLBs.

Pour valider l'aspect sécuritaire de cette implémentation (circuit : XOR + S-Box) une campagne de simulation électrique a été menée, en utilisant le logiciel Eldo de Synopsys. La technologie utilisée est la 65nm de ST. Dans cette campagne, plusieurs vecteurs de test ont été utilisés. La Figure 65 montre le profile de courant de la S-Box pour les différents vecteurs utilisés. Comme le montre cette figure, toutes les courbes sont presque superposées, ce qui prouve que la consommation de ce circuit est presque constante. Le circuit a donc une consommation quasi indépendante des données.

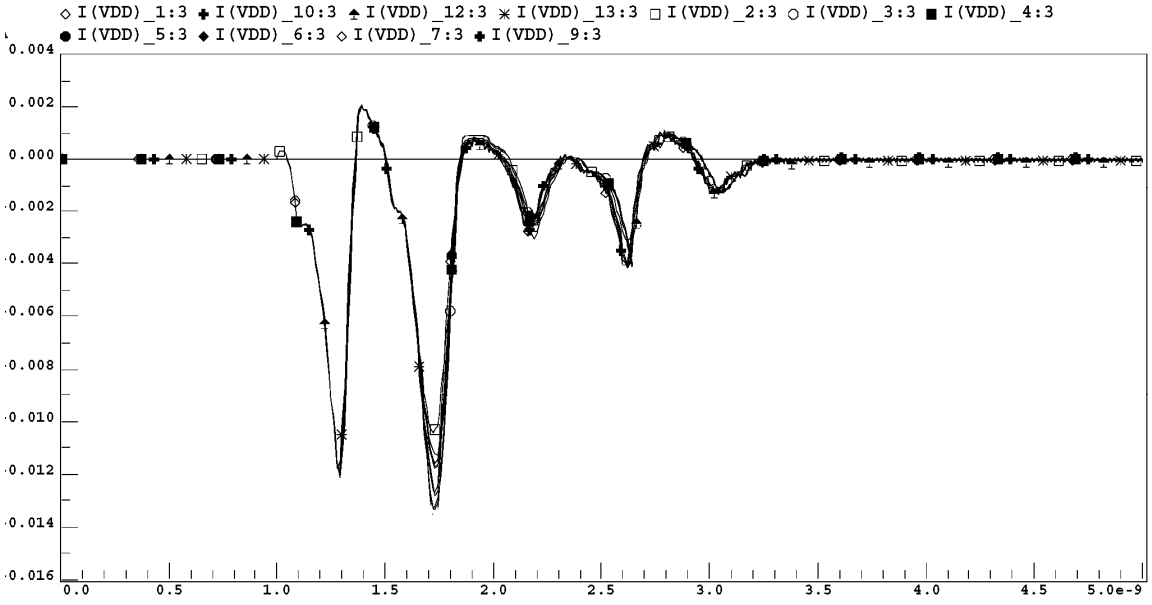


FIGURE 65: PROFIL DE COURANT DE LA S-BOX + XOR 4 PHASES

La Figure 66 montre que le temps d'exécution est indépendant des données manipulées. L'indépendance des données manipulées vis-à-vis de la consommation et du temps augmente d'une manière très significative la résistance du circuit aux attaques par analyse de courant et par analyse temporelle. Cela rend les attaques plus difficiles voire même couteuses à les réaliser. Sachant que faire un circuit 100% résistant à ce genre d'attaques semble impossible (il y aura toujours des canaux auxiliaires source d'informations), le but d'un concepteur se résume à rendre ces attaques très coûteuses en termes de temps, d'effort et de moyen technologiques nécessaire à leur réalisation

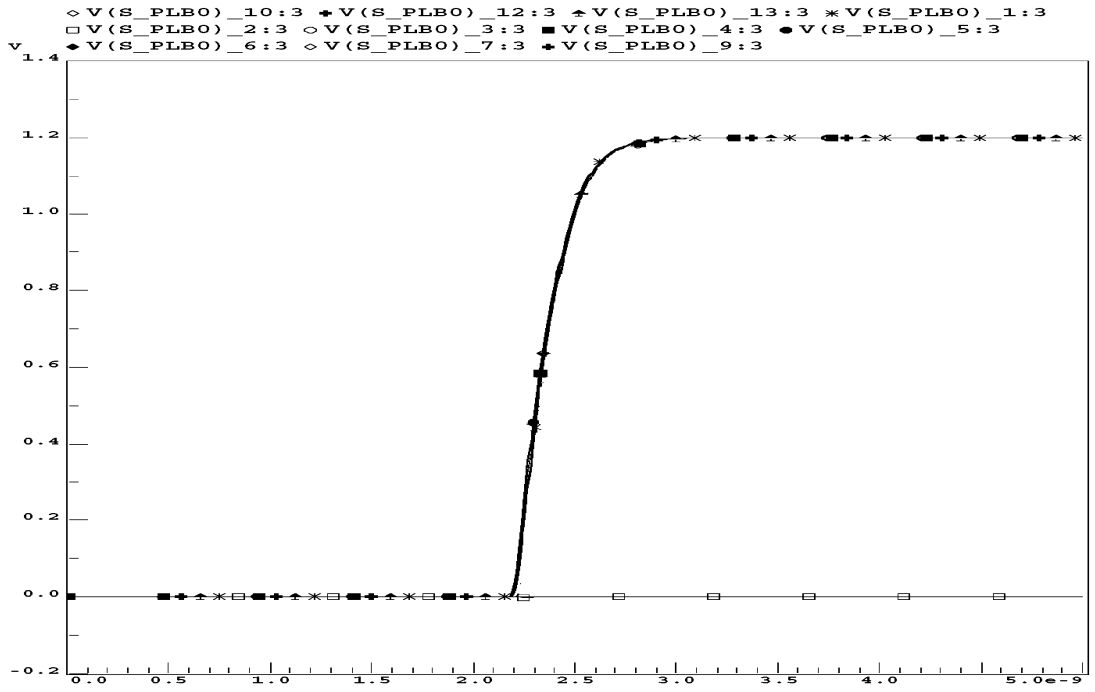


FIGURE 66: SORTIE DE LA S-BOX + XOR 4 PHASES

5.1.2 IMPLEMENTATION EN 2 PHASES, DOUBLER RAILS

Le même circuit XOR + S-box est implémenté à nouveau dans cet exemple, cette fois ci en 2 phases LEDR. Les équations ci-dessous décrivent les fonctions de sorties de la S-Box :

$$S_{0d} = f_{0d}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{0d}^{-1})$$

$$S_{0r} = f_{0r}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{0r}^{-1})$$

$$S_{1d} = f_{1d}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{1d}^{-1}$$

$$S_{1r} = f_{1r}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{1r}^{-1})$$

$$S_{2d} = f_{2d}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{2d}^{-1})$$

$$S_{2r} = f_{2r}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{2r}^{-1})$$

$$S_{3d} = f_{3d}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{3d}^{-1})$$

$$S_{3r} = f_{3r}(A_d, A_r, B_d, B_r, C_d, C_r, D_d, D_r, E_d, E_r, F_d, F_r, S_{3r}^{-1})$$

ÉQUATION 21

Le problème du nombre d'entrées de ces fonctions est le même que dans le cas précédent. Cependant, même si les données ici sont codées en double rails, elles ne sont pas codées en 1 parmi 2. Un bit logique A en 2-phase LEDR est codé sur deux fils (A_d, A_r) : le premier ap-

pelé fil de données $\boldsymbol{A_d}$ possède la même valeur que A, alors que le deuxième appelé fil de répétition sert à indiquer s'il s'agit d'un nouveau bit, ou d'une répétition du même bit précédent. L'algorithme décrit précédemment s'applique aussi sur ces fonctions avec les différences suivantes :

- Dans la deuxième étape les rails sont séparés en deux groupes :

  Rails de données : $\xi_d = \{A_d, B_d, C_d, D_d, E_d, F_d\}, et$

  Rails de répétition : $\xi_r = \{A_r, B_r, C_r, D, E_r, F_r\}.$

Seul le premier groupe est capable de calculer les valeurs des sorties de la S-Box. Le rôle du deuxième groupe se limite à contrôler le protocole de communication 2 phases.

- Dans la troisième étape, le calcul des sorties de données intermédiaires f_{id} de la S-Box se fait à partir de ξ_d. Le groupe ξ_r, étant non utilisé dans ce calcul, passe par une fonction identité comme le montre la Figure 67. Cela est fait dans le but d'équilibrer la profondeur logique de toutes les entrées de la S-Box.

- La reconstruction du protocole de communication 2 phases LEDR, dans la troisième étape nécessite à nouveau l'utilisation des couples double rails (E_d, E_r). L'objectif de cette étape est de signaler l'arrivée des nouvelles données (cf. Figure 68) (cf. section 3.1.2.1).

En fait ce bloc permet de distinguer entre les trois états suivants :

- $A_d \oplus A_r = B_d \oplus B_r = C_d \oplus C_r = D_d \oplus D_r = E_d \oplus E_r = F_d \oplus F_r$
- $A_d \oplus A_r \neq B_d \oplus B_r \neq C_d \oplus C_r \neq D_d \oplus D_r \neq E_d \oplus E_r \neq F_d \oplus F_r$
- sinon

Il est à noter que ce bloc utilise le codage en 1 parmi 3, pour garder un poids de Hamming constant.

Enfin, un bloc finale permet d'assembler les sorties intermédiaires et le signal de sortie du bloc de test pour calculer les sorties finales de la S-Box.

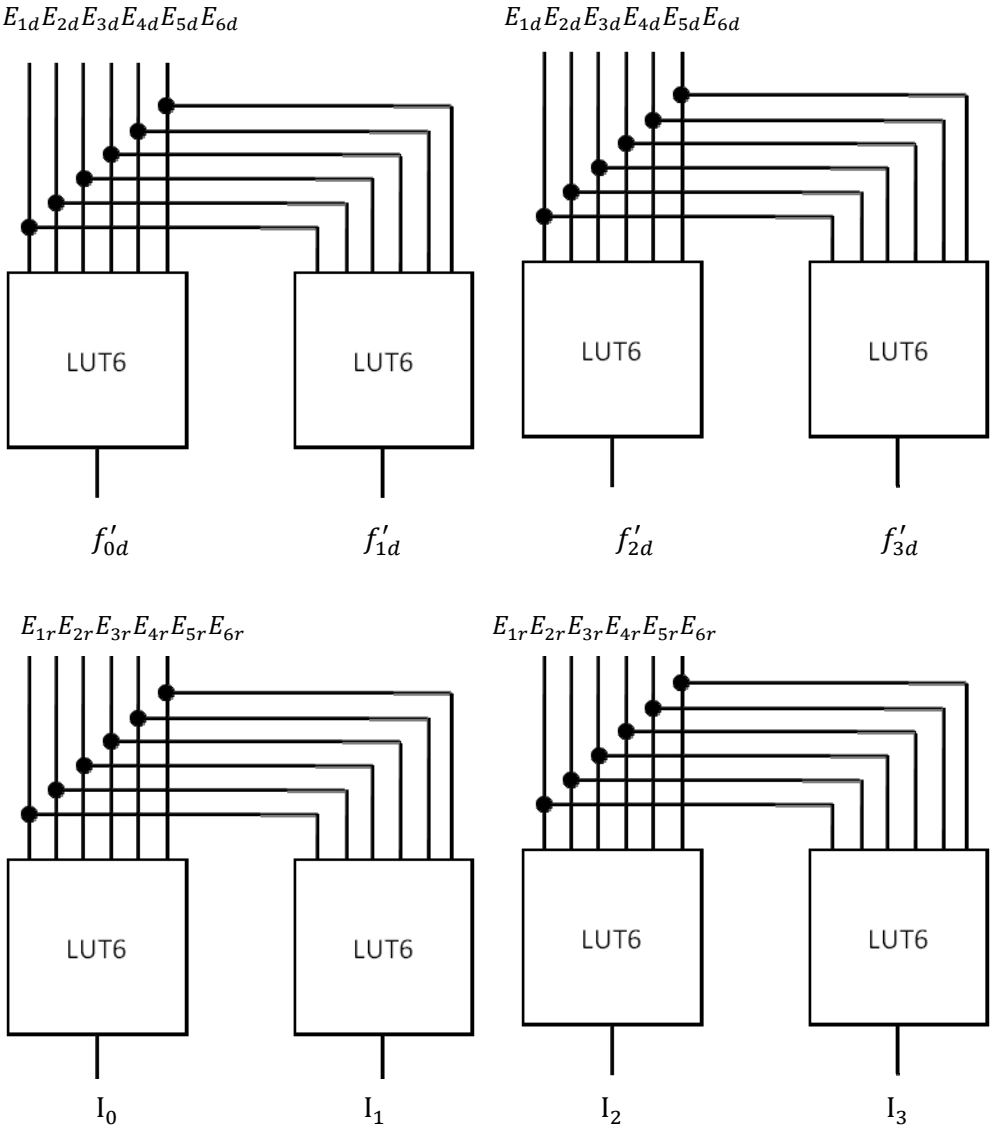


FIGURE 67: CALCUL DES SORTIES INTERMEDIAIRES DE LA S-BOX 2 PHASES

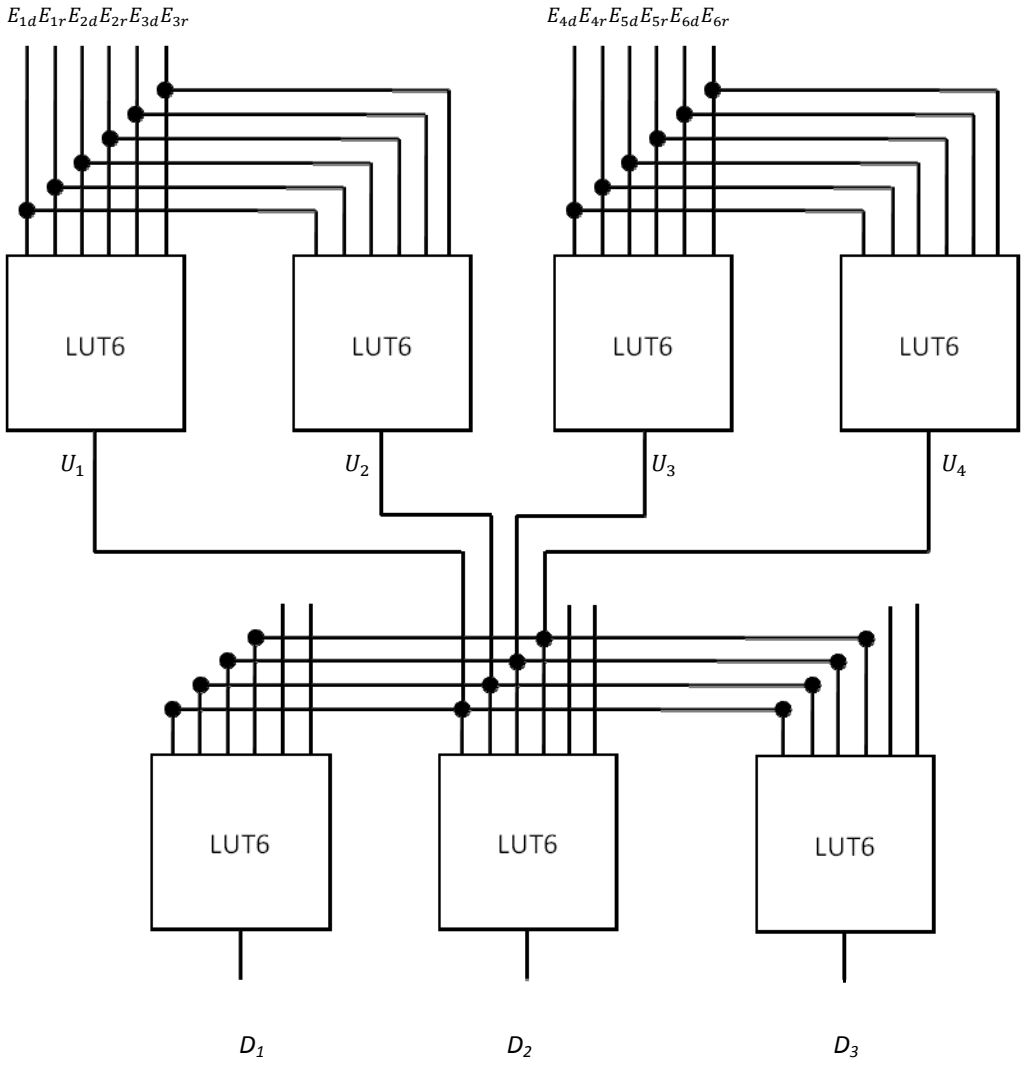


FIGURE 68: BLOC DE TEST : RECONSTRUCTION DU PROTOCOLE DE COMMUNICATION 2 PHASES LEDR

L'implémentation de la S-Box en 2 phases requiert donc 31 LUT6, dont 8 pour équilibrer le chemin de données. Une campagne de simulation électrique a été menée pour valider l'aspect sécuritaire du circuit. La technologie utilisée est la 65 nm de ST. Dans cette campagne plusieurs vecteurs de test sont été utilisés. La Figure 69 présente le profil de courant de la S-Box pour les différents vecteurs utilisés. Comme le montre cette figure, toutes les courbes sont superposées, ce qui prouve que la consommation de ce circuit est presque constante. Le circuit a donc une consommation indépendante des données manipulées. La Figure 70 montre que le temps d'exécution est aussi indépendant des données manipulées. L'indépendance des données vis-à-vis de la consommation et du temps, augmente d'une manière très significative la résistance du circuit aux attaques par analyse de courant et celle temporelles, et ce, en les rendant très difficiles et même coûteuses à les réaliser

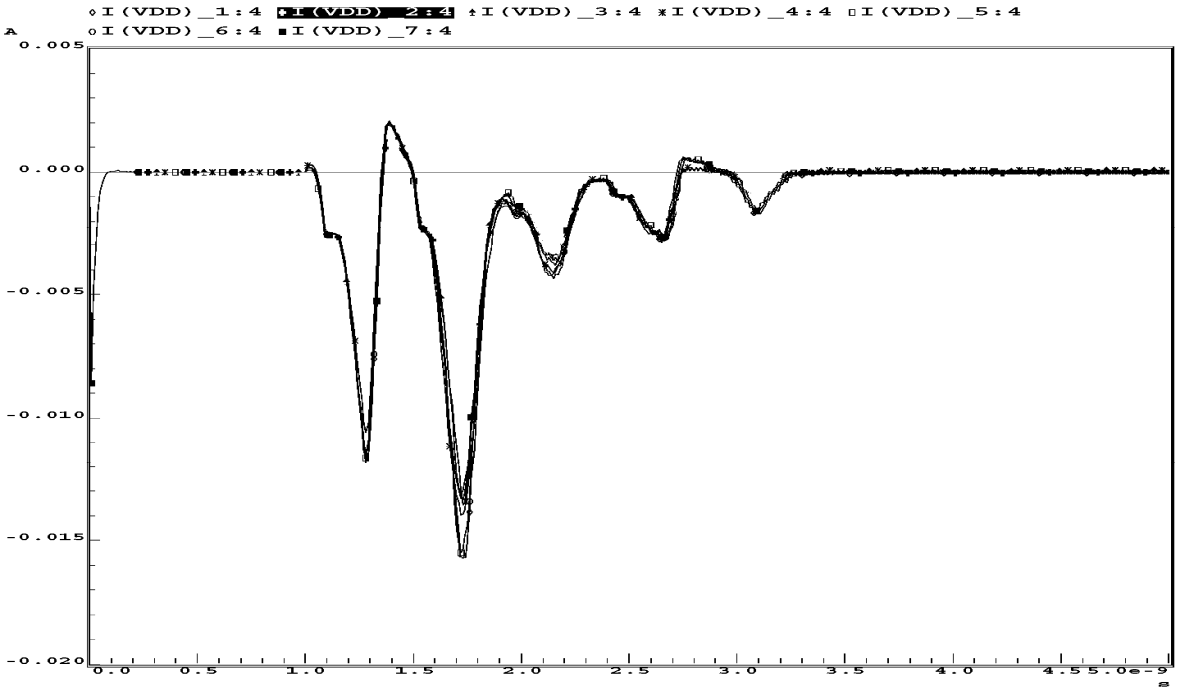


FIGURE 69: PROFILE DE COURANT DE LA S-BOX EN 2 PHASES LEDR

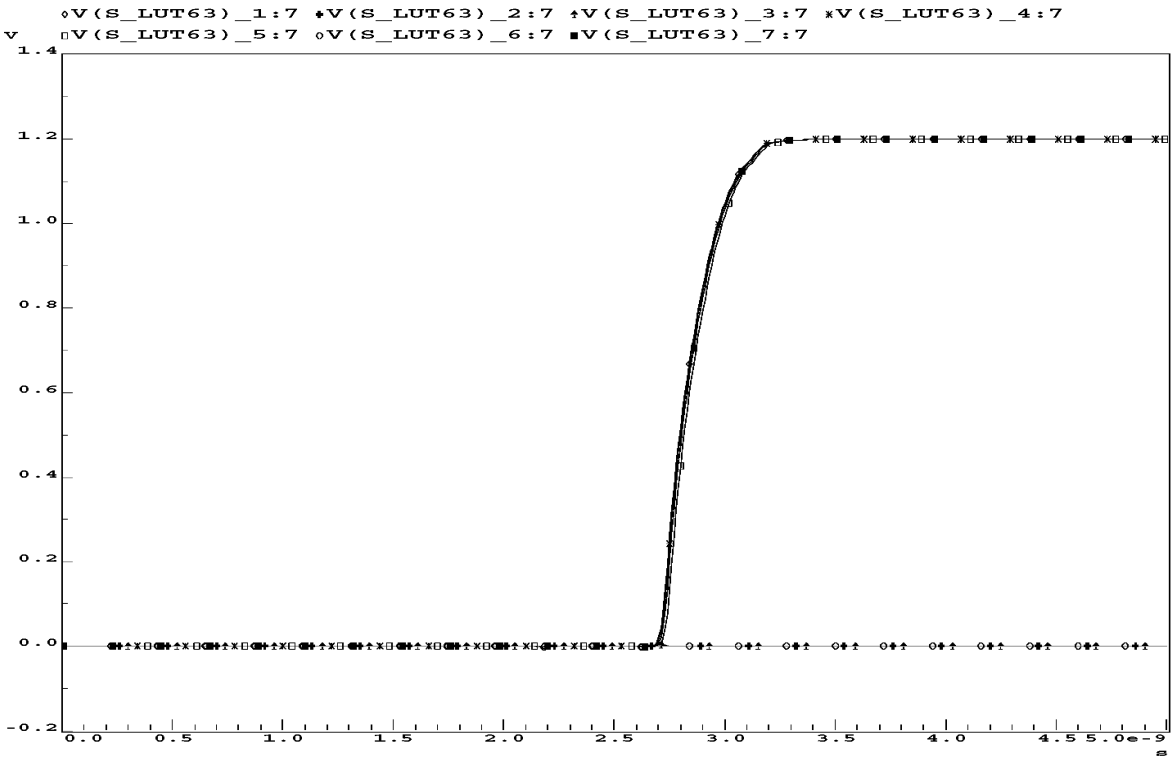


FIGURE 70: TEMPS D'EXECUTION DE LA S-BOX EN 2 PHASES LEDR

5.2 TEST DU FPGA FABRIQUE

Le prototype fabriqué possède 9 PLBs : 3x3 (cf. Figure 71). Ce circuit a été fabriqué en technologie 65nm de ST Microelectronics CMOS065. Il possède 9 I/Os de chaque côté. Le *Layout* de FPGA a été généré comme il est décrit dans (S. Chaudhuri, 2009) :

- Tout d'abord les *switches* sont placés d'une façon symétrique.
- Les fils sont routés manuellement pour obtenir le niveau de sécurité demandé.
- Le bloc programmable et les points de mémoire sont ensuite placés et routés automatiquement

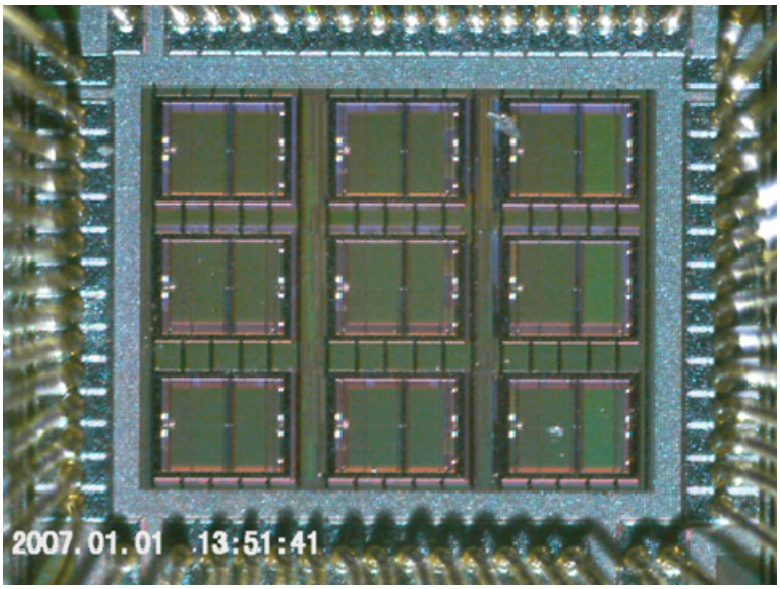


FIGURE 71: PROTOTYPE 3X3 DU FPGA (CMP RUN S65C8-1)

Le prototype occupe une surface de 1111.6 x 947.6 µm² et contient 200,000 transistors.

Pour tester le FPGA, une carte Altera DE2 a été utilisée. Elle fournit les signaux d'entrées du FPGA, reçoit et sauvegarde ses sorties. La carte est elle-même contrôlée par un PC utilisant un *Shell Python* développé spécialement pour cela.

Le FPGA est connecté à la carte Altera au travers d'une carte spécialement fabriquée à Telecom ParisTech. Il possède 9x4 I/Os en total ; seulement 12 plots sont bidirectionnels sur la carte, tandis que les autres sont soit des plots d'entrées, soit des plots de sorties. Le but était alors de limiter les *buffers* sur la carte, tout en conservant la possibilité de l'implémentation des protocoles 2 phases et 4 phases.

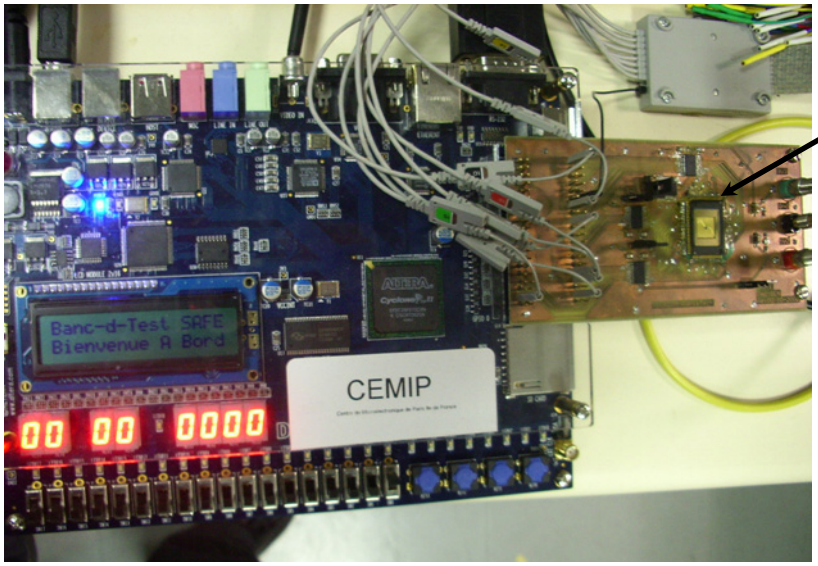


FIGURE 72: CARTE DE TEST ALTERA + FPGA ASYNCHRONE

La génération du *bitstream* du routage est faite grâce au logiciel VPR, alors que le *bitstream* des PLB est généré grâce à un programme développé au sein de ce travail, et basé sur l'algorithme présenté précédemment dans ce chapitre,

Le test fonctionnel du FPGA a été mené de la manière suivant :

Tout d'abord la chaîne de configuration du FPGA est initialisée, en forçant l'entrée INIT de FPGA à 0 pendant un temps de 2 sec.

Le *bitstream* généré par VPR est chargé dans la RAM de la carte Altera DE2, en utilisant un PC. Ensuite, il est transformé en codage 1 parmi 2 et est transféré au FPGA via les entrées *configuration_0* et *configuration_1.*

La carte DE2 surveille les acquittements de la chaîne de configuration et vérifie que le FPGA est configuré avec succès. Elle affiche un rapport en temps réel du déroulement de la configuration. Le nombre d'acquittement compté doit être 4692 pour que la configuration soit considérée comme réussite.

La configuration se fait avec une grande vitesse 50 Mbits /sec. Cependant un bug a été remarqué, et a empêché la finalisation du test, ainsi qu'une éventuelle attaque pour tester la sécurité du FPGA. Ce bug est dû à un court-circuit au niveau de la sortie de la LUT6. Ce court-circuit, pendant un temps très court, modifie le contenue de la mémoire de la LUT6.

En effet, une LUT6 est faite à partir d'une chaîne de 64 *Full-buffers*, et d'un décodeur (cf. Figure 73). Chaque *Full-Buffer* est connecté à une *transmission gate*. Les sorties des 64

transmission-gates sont toutes connectées à la sortie de la LUT6. Le rôle du décodeur est de commander les *transmission-gates* de telle manière qu'une seule à la fois soit passante. Cependant, à chaque variation des entrées de la LUT6, deux *transmission-gates* reçoivent une commande de la part du décodeur : la première devient passante, pendant que la deuxième s'ouvre. A ce moment un court-circuit se crée, changeant ainsi les valeurs des deux *Full-Buffers* concernés.

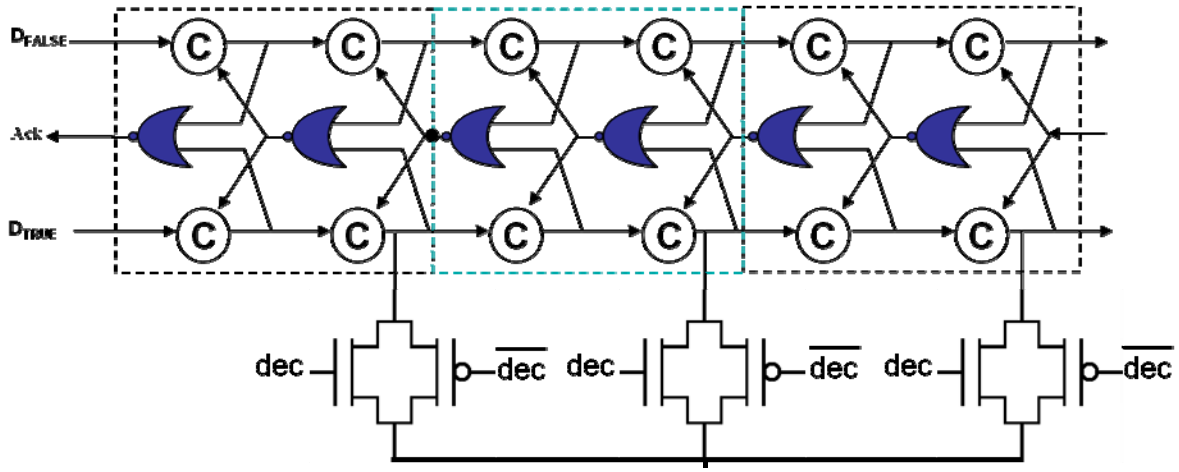
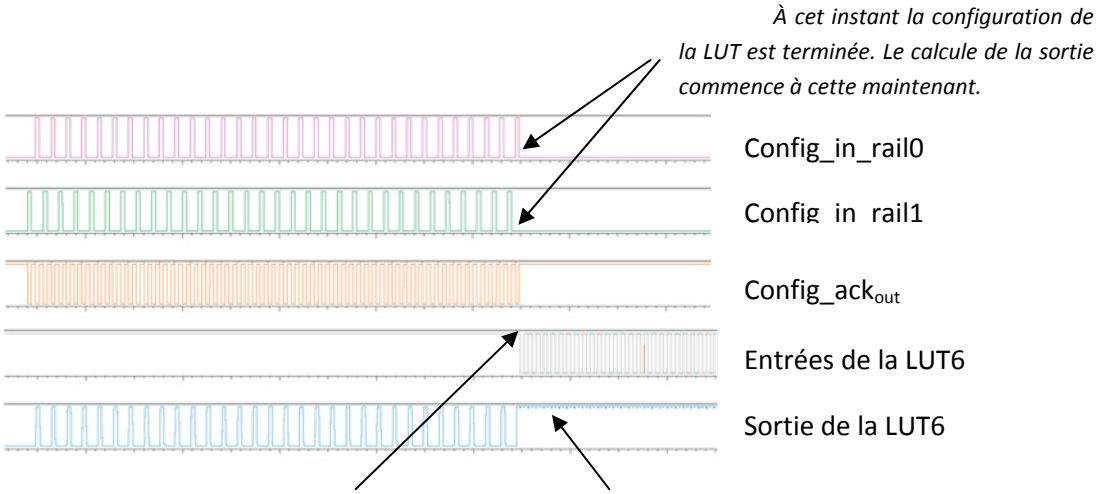


FIGURE 73: ARCHITECTURE DE LA LUT6



Lorsque les entrées commencent à varier, un court-circuit se présente, au niveau de la sortie de la LUT6. Ce court bascule les valeurs des bits de Full-Buffer vers un état haut. Et la sortie de la LUT6 prend une valeur constante = 1 quelque soit les entrées.

FIGURE 74: SIMULATION FONCTIONNELLE D'UNE LUT6

5.2.1 NOUVELLE VERSION

Plusieurs architectures ont été étudiées pour corriger cette erreur. La Figure 75 présente la solution retenue (pour le moment). Elle consiste à ajouter un inverseur entre la sortie du *Full-Buffer* et la l'entrée de la *transmission-gate.* La sortie du *Full-Buffer* correspond désormais à une sortie inversée par rapport à l'architecture initiale. Cette architecture a été testée fonctionnellement (cf. Figure 76) avec succès. Les courbes de simulations électriques présentent les mêmes propriétés que celles de la première version avec une petite dégradation. Ce travail d'investigation est toujours en cours. D'autre architectures ont également été étudiées et dont le profil de courant n'été pas uniforme. Des améliorations ont déjà été proposées pour les corriger afin de retenir la meilleure solution.

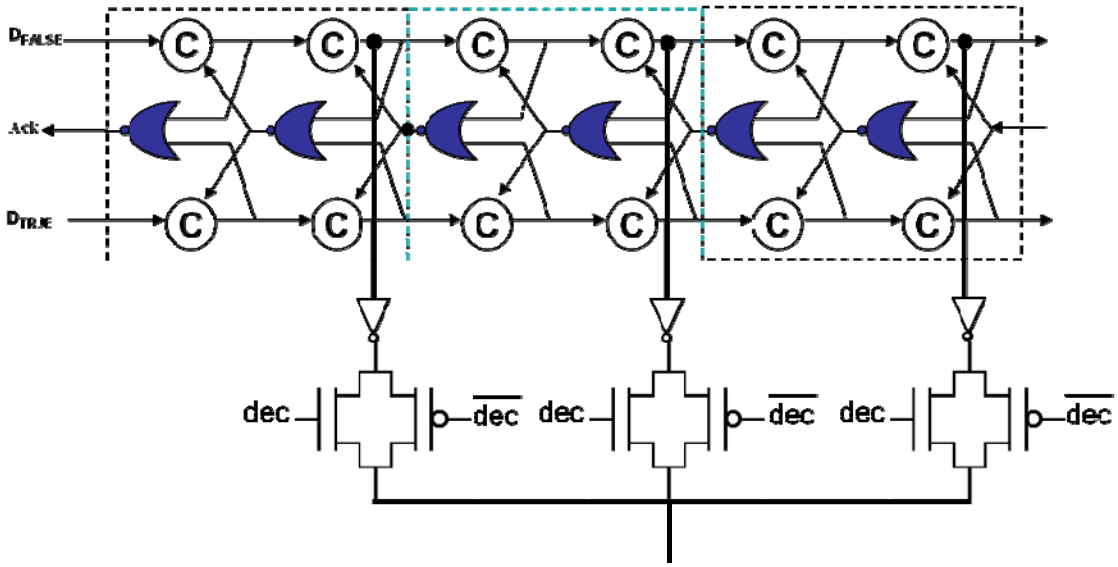
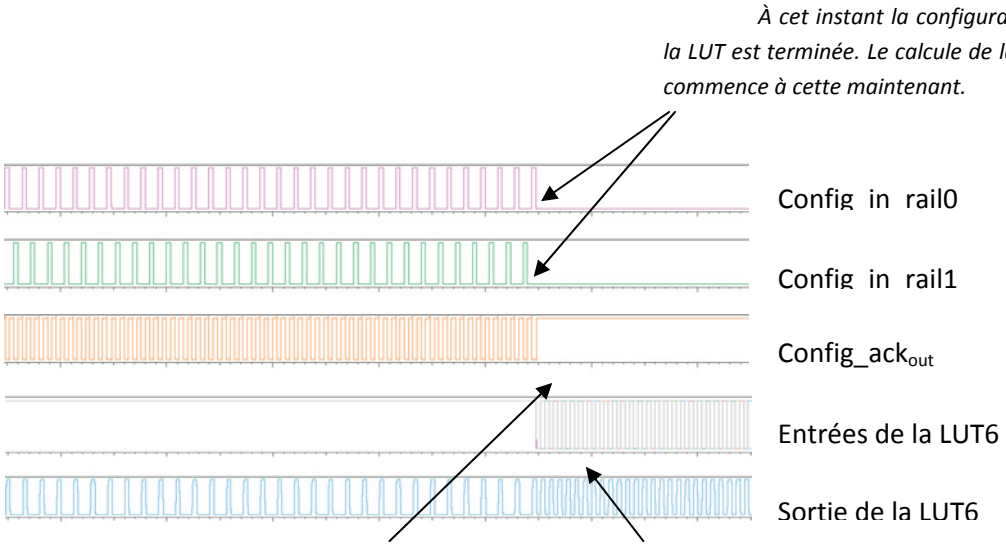


FIGURE 75: NOUVELLE VERSION DE LA LUT6



Lorsque les entrées commencent à varier, La sortie varie aussi. La LUT6 ici implémente une fonction d'identité. La sortie de la LUT6 varie bien comme il est prévu. En effet l'ajout de l'inverseur à l'entrée de la transmission gate, rend cette dernière unidirectionnelle, ainsi, le court-circuit qui se crée à chaque variation des entrées n'a plus d'impacte sur les bits de programmation des Full-Buffer. Ces derniers gardent toujours leurs valeurs.

FIGURE 76: SIMULATION DE LA NOUVELLE VERSION DE LA LUT6

5.3 CONCLUSION

Ce chapitre décrit comment implémenter une fonction de 7 entrées ou moins sur le FPGA. Les fonctions plus complexes nécessitent une décomposition en sous-fonctions. Cependant, il est à noter que nos techniques de simplification par factorisation ne s'appliquent pas à certaines fonctions à plus de 7 entrées car leur factorisation n'est pas possible. C'est par exemple le cas des S-Box des algorithmes de cryptage.

Pour implémenter un circuit sur le FPGA, un programme de tech-mapping a été développé. Il est basé sur l'algorithme qui est décrit dans ce chapitre.

Deux campagnes électriques sont présentées dans ce chapitre. Elles ont validé l'approche sécuritaire définie au début du manuscrit. Le circuit étudié dans ce but est un circuit sensible aux attaques par canaux cachés; il s'agit d'un sous-bloc d'un circuit de chiffrement DES et est constitué d'une S-BOX et d'un XOR. Il possède 12 entrées doubles rails et 4 sorties doubles rails. Les résultats des campagnes ont prouvé que la consommation électrique et le temps de calcul du bloc étaient indépendants des données manipulées et cela pour les protocoles 2 phases et 4 phases.

Le nombre de LUT6 utilisés était le même pour les deux protocoles à la différence que le protocole 4 phases et deux fois moins rapide que le protocole 2 phases à cause du retour à zéro qu'il implémente.

Le prototype du FPGA a été fabriqué en technologie CMOS 65 nm de ST Microelectronics. Il comporte 36 entrées/sorties et occupe une surface de : 1 mm² et contient 200,000 transistors. Il est soudé sur une carte spécifique fabriquée à Telecom ParisTech.

Une carte Altera a été utilisée pour communiquer avec le FPGA. Le *bitstream* du routage a été généré par le logiciel VPR, tandis que le *bitstream* de programmation des PLBs a été généré par un programme basé sur l'algorithme présenté au chapitre précédent. Cela a permis de découvrir une erreur de conception au niveau de la mémoire des PLBs. Cette erreur nous a empêchée d'aller jusqu'au bout de la validation de notre prototype.

Plusieurs solutions pour corriger ont été étudiées dont une a été présentée dans le chapitre. La validation électrique de cette architecture est en cours. La fabrication d'un nouveau prototype est prévue.

Chapitre 6.
CONCLUSION ET PERSPECTIVES

De nos jours, aucun FPGA asynchrone ne permet l'implémentation de plusieurs styles de logique asynchrone, ni possède une architecture robuste aux attaques par canaux cachées. Les travaux menés dans le cadre de cette thèse portent sur ces deux points : la conception d'un FPGA asynchrone multi-styles (qui implémente plusieurs types de protocoles de communication et plusieurs codage de données) et dont l'architecture est nativement résistante aux attaques par canaux cachés, notamment les analyses en courant et en temps.

Dans un premier temps, les critères ainsi que les spécifications logiques et électriques ont été choisis soigneusement pour répondre aux exigences de sécurité et de flexibilité de l'implémentation. Des contre-mesures ont été intégrées à l'architecture du bloc programmable pour le rendre nativement robuste contre certaines attaques par canaux cachés, notamment l'analyse du courant et du temps de calcul.

L'architecture du FPGA est du type *Island*. Elle n'est rien d'autre qu'une répétition d'un motif « PLB + *Switch-Box + Connection-Box* » (cf. Figure 58). Les blocs programmables et les points de mémoires ont été conçus en *full custom* en technologie CMOS 65 nm de ST Microelectronics.

Sur le plan électrique, cette architecture est bien équilibrée permettant ainsi l'implémentation de fonctions sécurisées. En effet, les contre-mesures proposées ont permis de minimiser significativement la dépendance des données vis-à-vis de la consommation et du temps de calcul. Les blocs programmables présentent ainsi - pour une fonction programmée correctement - un profil de courant indépendant des données manipulées. De plus, l'utilisation de la logique asynchrone renforce cet aspect. Il a été montré que cette logique possède un fort potentiel et des propriétés intéressantes pour concevoir des circuits sécurisés.

Sur le plan logique, l'architecture a été conçue non seulement pour assurer une flexibilité d'implémentation de différents styles de logiques asynchrones, mais aussi pour garantir un temps de calcul indépendant des données manipulées. Ainsi une fonction implémentée sur le FPGA réagit toujours de la même façon quelque soient les données. Cela permet, en outre de tester différents styles de logique asynchrone en tant que contre-mesures, d'évaluer leur impact sur la sécurité du circuit implémenté.

L'aspect sécuritaire de l'architecture a été validé par des circuits (S-Box + XOR) en 2 phases et en 4 phases. Chacun montre un profil de courant uniforme, et un temps de calcul indépendant des données manipulées. Un prototype a été fabriqué en technologie CMOS 65 nm de ST Microelectronics. Il a été testé à l'aide d'une carte Altera qui a assurée la communication entre un PC et le FPGA asynchrone. La programmation de ce FPGA a été assurée par un programme spécifique pour ce type d'application, développé au cours de cette thèse. Le routage du circuit a été, quant à lui, obtenu par le logiciel VPR.

Les tests de ce prototype ont dévoilé une erreur de conception au niveau de la mémoire (FIFO) de programmation. Une nouvelle version a été proposée et est en cours de validation avant d'être envoyée en fabrication. Elle permettra de tester plus complètement le FPGA et de réaliser de réelles attaques afin de valider l'approche sécuritaire.

Parmi les perspectives de ce travail, la finalisation de la nouvelle version et la validation électrique par une simulation d'une attaque DPA. Cette simulation représente une évaluation intéressante et complémentaire des travaux de test menés lors de cette thèse. Elle permettra aussi d'avoir une idée plus précise sur les performances de la solution proposée.

L'optimisation de la nouvelle architecture proposée pourrait faire l'objet d'un travail de développement plus approfondi. En particulier, une diminution de la surface du FPGA permettrait l'implémentation d'un cryptosystème plus complet. Ceci nécessitera un développement plus complet de l'outil de programmation et une standardisation des règles d'implémentations des différents types de fonctions. Il nécessite aussi le développement d'une nouvelle carte de test et d'une interface de communication plus complète.

L'utilisation de plusieurs types de logiques asynchrones à pénaliser l'architecture du FPGA sur plusieurs niveaux. Une tâche très importante pourrait être la définition du meilleur style de logique asynchrone et du meilleur codage de donnée qui permettront d'atteindre les

objectifs sécuritaires au moindre coût (avec une surface minimale). L'exploration des différentes topologies de FPGAs pourrait mener aussi à une architecture optimisée non seulement en terme de surface mais aussi en termes de performance et de complexité de programmation.

Les techniques de durcissement proposées dans ce travail concernent principalement les attaques par analyse en courant et en temps. Des campagnes de validation ont été menées pour évaluer les deux premiers types d'attaques. D'autres tests pourraient être réalisés pour valider la robustesse du FPGA contre des attaques de type électromagnétique. Finalement, de nouvelles contremesures pourraient être étudiées et intégrées dans l'architecture, en particulier les contremesures contre les attaques par injection de fautes. Cela permettra une comparaison juste et objective des résultats obtenus avec d'autres implémentations ASIC développer au sein de l'équipe CIS.

Bibliographie

- A.J. Martin. (September,1997). "The design of an asynchronous MIPS R3000 Microprocessor". *ARVLSI'97*, (pp. 164-181).
- A.J. Martin. (1990). "The limitations to delay-insensitivity in asynchronous circuits". *Proceedings of the Sixth MIT Conference on Advanced* , p. 263-278.
- K. Berkel. (1993). "Beware the isochronic fork, Integration". *The VLSI journal* , 103-128.
- M. Matsui. (May 23-27, 1993). "Linear Cryptanalysis Method for DES Cipher". *Eurocrypt'93*, *765*, pp. 386-397.
- National Institute Of Standards And Technology. (2000). *"Digital Signature Standard DSS"*. (F. I. Standard, Éd.)
- W.A. Clark. (1967). "Macromodule Computer Systems". *Spring Joint Computer Conference AFIPS.*
- (1977). *"Data Encryption Standard"*. National Bureau of Standards . Federal Information Processing Standard.
- A. Elbirt, W. Yip, B. Chetwynd, and C. Paar. (August 2001). "An FPGA-based performance evaluation of the AES block cipher candidate algorithm finalists.". *IEEE Transactions on VLSI Design* , 545–557.
- A. Hevia and M. Kiwi. (1999). "Strength of Two Data Encryption Standard Implementation Under Timing Attacks". *ACM Transaction on Information and System Security (TISSEC)*, *2*, pp. 416-437.
- A. L. Davis. (1978). "The architecture and system method of DDM1 : A recursively straucters data driven machine". *International Symposium Conference on Computer Architecture*, (pp. 210-215).
- A. O. Freier, P. Karlton, and P. C. Kocher. (November 1996). "The SSL Protocol Version 3.0". *Transport Layer SecurityWorking Group INTERNET-DRAFT.*
- A. One. (1996). "Smashing the stack for fun and profit". *Phrack Magazine , 7.*

- A. Ross. (2001.). "Security Engineering:A Guide to Building Dependable Distributed Systems". *Wiley Computer publishing ed. United States of America* .
- A. Shamir. (1985). "Idntity-based cryptosystems and signature schemes". *Lecture Notes in Computer Science - CRYPTO'84* , 47-53.
- A.J. Martin. (1993). "Synthesis of Asynchronous VLSI Circuits". *Internal Report* .
- A.J.Martin and M. Nystrom. (May 1997.). "The Lutonium: A sub-nanjoule asynchronous 8051 microcontroller". *ASYNC'03*, (pp. 14-23). Vancouver, Canada.
- *Achronix*. (s.d.). Récupéré sur http://www.achronix.com/
- *Amphion*. (s.d.). Récupéré sur High Performance AES Encryption Cores: http://www.chipcenter.com
- B. Gao. (December 1996). *"A globally asynchronous locally synchronous configurable array architecture for algorithlm embeddings"*. University of Edinburg.
- B. Gierlichs, K. Lemrke-Rust and C. Paar. (2006). "Template vs stochastic methods". *Cryptographic Hardware and Embedded Systems- CHES'06. 4249*, pp. 15-29. Springer-Verlag.
- B. S. Kaliski, Jr., C. K. Koc, and C. Paar, editors. (August 13-15, 2002). *Workshop on Cryptographic Hardware and Embedded Systems — CHES'02. LNCS 2523.* Berlin, Germany: Springer-Verlag.
- B. Schneier. (1996). *"Applied Cryptography"* (éd. 2nd). New York, USA: John Wiley & Sons Inc.
- B. Schneier. (2001). *"Cryptographie appliquée"*. Vuibert Informatique.
- C . K. Koc and C. Paar, editors. (August 17-18, 2000). *Workshop on Cryptographic Hardware and Embedded Systems — CHES'00. LNCS 1965.* Berlin, Germany: Springer-Verlag.
- C . K. Koc, D. Naccache, and C. Paar, editors. (May 13-16, 2001). *Workshop on Cryptographic Hardware and Embedded Systems — CHES'01. LNCS 2162.* Berlin, Germany: Springer-Verlag.
- C. Allen, T. Dierks. (January 1999). "The TLS Protocol Version 1.0". *Corporation for National Research Initiatives, Internet Engineering Task Force, Network Working Group.* Virginia, USA.
- C. Archambeau, E. Peeters, F-X. Standaert and J-J Quisquater. (2006). "Template attacks in principal subspaces". *Cryptographic Hardware and Embedded Systems - CHES'06. 4249*, pp. 1-14. Springer-Verlag.
- C. Canovas and J. Clédière. (2005). *"What do S-Boxes say in differential side channel attacks?"*. Cryptology ePrint Archive.

- C. Rechberger and E. Oswald. (2004). "Practical template attacks". *Workshop on Information Security Applications - WISA'04. 3325*, pp. 440-456. Sringer-Verlag.
- D. Agrawal, J. R. Rao and P. Rohatgi. (September 2003). "Multi-channel attacks". *Cryptographic Hardware and Embedded Systems Workshop CHES'03*, *2779*, pp. 2-16.
- D. H. Linder, J. C. Harden. (1996). "Phased Logic: Supporting the Synchronous Design Paradigm with Delay-Insensitive Circuitry". *IEEE Transactions on Computers* , 1031-1044.
- D. Samyde, S. Skorobogatov, R. Anderson and J-J. Quisquater. (December 2002). "On a New Way to read Data from Memory". *First International IEEE Security in Storage Workshop,*. Greenbelt Maryland.
- D. Sokolov, J. Murphy , A. Bystrov and A.Yakovlev. (2004). "Improving the Security of Dual-Rail Circuits". *CHES'04. 3156/2004*, pp. 255-317. Springer Berlin / Heidelberg.
- D. Stinson. (1996). "Cryptographie: Théorie et Pratique". *International Thomson Publishing France* , 106.
- D. Suzuki and M. Saeki. (2006). "Security evaluation of DPA contermesures using dual-rail pre-charge logic style". *CHES*, *4249*, pp. 255-269.
- E . Biham and A. Shamir. (1992). "Differential Cryptanalyse of the full 16-Round DES". *Advances in Crytography CRYPTO'91. 740*, pp. 487-496. Springer-Verlag.
- E. Brier, C. Clavier and F.Olivier. (2004). "Correlation power analysis with a leakage model". Dans Springer-Verlag (Éd.), *Cryptographic Hardware and Embedded Systems - CHES'04*, (pp. 135-152).
- E. Brier, C. Clavier and F.Olivier. (2003). *"Optimal statistical power analysis"*. Cryptlogy ePrint archive.
- E. De Mulder, P. Buysschaert, S. B. Ors, P. Delmotte, B. Preneel, G. Vandenbosch and I. Verbauwhede. (November 2005). "Electromagnetic analysis attack on an FPGA implementation of an elliptic curve cryptosystem". *EUROCON: Proceedings of the International Conference on "Computer as a tool"*, (pp. 1879-1882).
- E. De Mulder, S. B. Ors, B. Preneel and I. Verbauwhede. (July 2006). "Diffirential electromagnetic attack on an FPGA implementation of elliptic curve cryptosystems". *World Automation Congress.*
- F. Bouesse. (2005). *"Contribution à la conception de circuits intégrés sécurisés: l'alternative (thèse de doctorat)"*. Institut National Polytechnique de Grenoble.
- F. Bouesse and M. Renaudin. (June 8 -11th 2005.). "Improving DPA resistance of Quasi-delay Insensitive Asynchronous Circuits". *3rd International Workshop on*

Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi'05. Saint-Etienne, France.

- F. Bouesse, G. Sicard and M. Renaudin. (Oct 10th-13th, 2006). "Path Swapping Method to Improve DPA resistance of Quasi Delay Insensitive Asynchronous circuits". *Workshop on Cryptographic Hardware and Embedded Systems (CHES'06)*, *4249*, pp. 384-398. Yokohama, Japan.
- F. Bouesse, M. Renaudin, B. Robisson, E. Beigne, P. Y. Liardet, S. Prevosto, and J.Sonzogni. (2004). "DPA on Quasi Delay Insensitive Asynchronous Circuits: Concrete Results". *XIX Conference on Design of Circuits and Integrated Systems (DCIS2004)*, (pp. , 24-26). Bordeaux, France.
- F. Bouesse, M. Renaudin, S. Dumont and F. Germain. (March 7-11, 2005). "DPA on Quasi Delay Insensitive Asynchronous Circuits: Formalization and Improvement". *Design Automation and Test in Europe Conference and Exhibition (DATE'05)*, (p. 424). Munich, Germany.
- F. Koeune and J-J. Quisquater. (s.d.). *"Timing Attack against Rijndael"*. Récupéré sur cr.yp.to/bib/1999/koeune.ps
- F.-X. Standaert, E. Peeters, G. Rouvroy and J.J. Quisquater. (Febuary 2006). "An overview of analysis attacks against field programmable gate arrays". *Proceedings of the IEEE*, (pp. 383-394).
- F.-X. Standaert, F. Mace, E. Peeters and J-J. Quisquater. (2006). "Updates on the security of FPGAs against power analysis attacks". *Reconfigurable Computing:Architectures and Apllications*, *3985*, pp. 335-346.
- F.-X. Standaert, L. van Oldeneel tot Oldenzeel, D. Samyde and J.-J. Quisquater. (September 2003). "Differential power analysis of FPGAs : How practical is the attack?". *Fielad Programmablmle Logic and Applications*, (pp. 701-709).
- F.-X. Standaert, S. B. Ors, J.-J. Quisquater and B. Preneel. (August 2004). "Power analysis attacks against FPGA implementations of the DES". *Field Programmable Logic and Applications*, (pp. 84-94).
- *Fulcrum*. (s.d.). Récupéré sur http://www.fulcrummicro.com.
- F-X. Standaert, S. B. Ors and B. Preneel. (2004). "Power analysis of an FPGA implementation of Rijndael: Is pipelining a DPA countermeasure?". *Cryptographic Hardwareand Embedded Systems CHES'04*, *3156*, pp. 30-44.
- H. Handschuh and H.Heys. (2004). "A timing attack on RC5". Dans S. E. Meijer (Éd.), *Cryptographic Hardware and Embedded Systems CHES'04. 3156*, pp. 254-267. Srpinger - Verlag.

- H. Lipmaa. (s.d.). *"Fast Software Implementations of AES"*. Récupéré sur http://www.tcs.hut.fi/ helger/aes/rijndael.html
- *handshake solutions*. (s.d.). Récupéré sur http://www.handshakesolutions.com/
- I. Sutherland, B. Sproull and D. Harris. (1999). *"Logical Effort"*. San Francisco, California: Morgan Kaufmann.
- Ivan E. Sutherland. (1989). "Micropiplines". *Communication of the ACM , 32* (6).
- J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. (2002). "Logic Synthesis of Asynchronous Controllers and Interfaces". *Springer-Verlag* .
- J. Ebergen. (July 1991). "A formal approach to designing delay-insensitive circuits". *Distributed Computing , 5*, 107-119.
- J. F. Dhem, F. K. (1998). A practical implementation of the timing attack. *The third conference on Smard card research and applications CARDIS'98*, *1820*, pp. 167-182. Louvain La Neuve, Belgium.
- J. F. Dhem, F. Kouene, P-A. Leroux, P. Mestre, J- J. Quisquetr and J- L. Willems. (1998). "A practical implementation of the timing attack". *The third conference on Smard card research and applications CARDIS'98*, *1820*, pp. 167-182. Louvain La Neuve, Belgium.
- J. R. Rao and P. Rohatgi. (2001). "EMpowering Side-Channel Attacks". IACR ePRINT 2001/037.
- J. Sparso. (2006). *"Principles of Asynchronous Circuit design - A systems perspective"*. Boston/Dordrecht/London: Kluwer Academic Publishers.
- J. Teifel and R. Manohar. (February 2004). "Highly Pipelined Asynchronous FPGAs". *2th ACM International Symposium on Field-Programmable Gate Arrays*. Monterey, CA.
- J.B. RIGAUD. (2002). *"Spécification de bibliothèques pour la synthèse de circuits asynchrones"*. Grenoble: Institut National Polytechnique de Grenoble.
- J.D. Garside et al. (April 2000). "AMULET3i – an Asynchronous Syste-on-Chip". *ASYNC'00*, (pp. 162-175).
- J.-J. Quisquater and D. Samyde. (2001). "ElectroMagnetic Analysis (EMA): Measures and countermeasures for smart cards". *In E-SMART: Proceedings of the International Conference on Research in Smart Cards*, (pp. 200-210).
- J.T. Udding. (1986). "A formal model for defninf and classifying delay-insensitive circuits". *Distributed computing* , 197-204.

- J-J. Quisquater and D. Samyde. (2000). "A nex tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods". *EUROCRYP'00.*
- K. Berkel. (1993). *"Handshake Circuits - An Asynchronous Architecture for VLSI".* Cambridge University Press.
- K. Gaj, P. Chodowiec. (April 8-12, 2001). "Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays". *In D. Naccache, editor, Topics in Cryptology- CT-RSA 2001 , LNCS 2020*, 84 – 99.
- K. Gandolfi, C. Mourtel, F. Olivier. (May 2001). "Electromagnetic analysis: Concrete results". *In Cryptographic Hardware and Embedded Systems Workshop CHES'01, 2162*, pp. 251-261.
- K. J. Kulikowski, M. Su, A. Smirnov, A. Taubin, Mark G. Karpovsky and D. MacDonald. (2005). "Delay Insensitive Encoding and Power Analysis: A Balancing Act". *11th IEEE International Symposium on Asynchronous Circuits and Systems* (pp. 116-125). Asynchronous Circuits and Systems, International Symposium.
- K. Makeswaran and V. Akella. (March 1998). "PGA-STC : Programmable Gate Array for Implementating Self-Timed Circuits". *84*.
- K. Tiri and I. Verbauwhede. (2004). "A Dynamic and Differential CMOS Logic Style to Resist Power and Timing Attacks on Security IC's.". *Cryptoprint.* Cryptology ePrint Archive.
- K. Tiri, I. Verbauwhede. (2004). "Synthesis of secure FPGA implementations". *International Workshop on Logic and Synthesis*, (pp. 224-231).
- L. Fesquet, B. FOLCO, M. STEINER, M. RENAUDIN. (2006). "State-holding in Look-Up Tables : application to asyncronous logic". *14th International Conference on Very Large Scale Integration VLSI-SOC*, (pp. 12-17). NICE.
- L. Fesquet, M. Renaudin. (2005). "Programmable logic architecture for prototyping clockless circuits". *FPL'05*, (pp. 293-298.). Tampere, Finland.
- L. Shang, A. S. Kaviani and K. Bathala. (2002). "Dynamic power consumption in Virtex-II FPGA family". *Field Programmable Gate Arrays Symposium FPGA'02*, (pp. 157-164).
- M. Bucci, L. Giancane, R. Luzzi, G. Scotti and A. Tri. (2006). "Enhancing power analysis attacks against cryptographic devices". *In Circuits and Systems Symposium.*

- M.G. Kuhn. (December 2003). *"Compromising emanations: eavesdropping risks of computer displays"*. Technical Report 577, University of Cambridge, Computer Laboratory.
- N. Koblitz. (1987). "Elliptic curve cryptosystems". *Mathematics of Computation* , 203-209.
- NIST. (November 2001). *"Advanced Encryption Standard (AES)"*. National Institute of Standard and Technology NIST.
- NSA. (2000). "NACSIM 5000 Tempest Fundamentals". Dans F. G. Mead (Éd.). Maryland.
- O. Kommerling and M.G. Kuhn. (May 1999.). "Design Principles for Tamper-Resistant Smartcard Processors". *USENIX Workshop on Smartcard Technology (Smartcard '99)*, (pp. 9–20,).
- P. Chow, S. Ong Seo, J. ROse, K. Chung, G. Paéz-Monzon and I. Rahardja. (1999). "The design of an SRAM based Field-Programmable Gate Array, Part II : circuit design and layout". *IEEE Transactions on Very Large Sacal Integration (VLSI) Systems* , 321-330.
- P. Kocher. (1996). "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". *Advances iin Cryptography CRYPTO'96. 1109*, pp. 104-113. Springer-Verlag.
- P. Kocher, J. Jaffe, B. Jun. (1999). "Differential power analysis". *Advances in Cryptography - CRYPTO'99. 1666*, pp. 388-397. Springer-Verlag.
- P. Kocher, J. Jaffe, B. Jun. (1998). *"Introduction to diferntial power analysis and related attacks"*. Récupéré sur http://www.cryptography.com/resources/whitepapers/DPATechInfo.pdf
- R. Doud. (April 1999). "Hardware Crypto solutions boost VPN". *Electronic Engineering Times* , 57-64.
- R. Payne. (1997). *"Self Timed Field Programmable Gate Array Architectures"*. University of Edinbugh.
- R. Rivest, A. Shamir, L. Adleman. (1978). "A method for obtaining digital signatures and ublic-key cryptosystemes". *Communications of the ACM* , 120-216.
- S. B. Ors, E. Oswald, B. Prenee. (2003). "Power-analysis attacks on an FPGA first experimental results". *Cryptographic Hardware and Embedded Systems Workshop CHES'03*, *2779*, pp. 35-50.
- S. Chari, J. Rao, P. Rohatgi. (2003). "Template Attacks". *Cryptographic Hardware and Embedded Systems CHES'02. 2523*, pp. 13-28. Springer-Verlag.

- S. Chaudhuri. (2009). *"Asynchronous FPGA architectures for cryptographic applictions"*. Paris, France: Telecom ParisTech.
- S. Drimer. (2008). *"Volatile FPGA design security -- a survey v 0.96"*. Computer Laboratory, University of Cambridge.
- S. Guilley, L. Sauvage, J-L. Danger, T. Graba and Y. Mathieu. (july 2008). "Evaluation of Power-Constant Dual-Rail Logic as a Protection of Cryptographic Applications in FPGAs". *In SSIRI*, (pp. 16–23). Yokohama, Japan.
- S. Guilley, S. Chaudhuri, L. Sauvage, P. Hoogvorst, R. Pacalet and G.M. Bertoni. (s.d.). "Security Evaluation of WDDL and SecLib Countermeasures against Power Attacks". *IEEE Transactions on Computers* , 1482–1497.
- S. Hauck. (1995). "Asynchronous design methodology : An overview". *IEEE , 83* (1), 69-93.
- S. Hauck, G. Boriello and C. Ebeling. (August 1992). "Montage: An FPGA for Synchronous and Asynchronous Circuits". *2nd International Workshop on Field-Programmable Logic and Applications.* Vienna.
- S. Mangard, E. Oswald, T. Popp. (2007). *"Power analysis attacks: Revealing the secrets of smart cards"*. Secaucus, NJ, USA.
- S. Moore, R. Andersona, P. Cunningham, M. Robert and G. Taylor. (2002). "Improving smart card security using self-timed circuits". *8th asynchronous symposium on asynchronous circuits ASYNC'02.* Manchester U.K.
- S. Moore, R. Andersona, R. Mullinsa, G. Taylora and J. Fournier. (2003). "Balanced self-checking asynchronous logic for smart card applications". *Microprocessors and Microsystems 27 , 27*, 421-430.
- S.chari, C.S. Jutla, J.R. Rao and P. Rohatgi. (1999). "Towards sound approaches to counteract power-analysis attacks". *Lecture notes in computer science , 1666*, 398-412.
- *Silistix*. (s.d.). Récupéré sur http://www.silistix.com/
- T. ElGamal. (1985). "A public Key cryptosystem and a signature scheme based on discrete logaritrhms". *IEEE Transactions on Information Theory* , 469-472.
- T. Messerges, E. Dabbish, R. Sloan. (1999). "Investigations of power analysis attacks on smartcards". *USENIX workshop on smartcard technology.*
- T. Popp and S. Mangard. (2005). "Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. *CHES*, *3659 of LNCS*, pp. 172-186.
- T. Verhoeff. (1988). "Delay insensitive codes - an overview". *3*, pp. 1-8. Distributed Computing.

- T. Wollinger and C. Paar. (2004). *"How secure Are FPGAs in cryptographic applications? (long version)"* .
- T. Wollinger, J.Guajardo and C. Paar. (2004). "Security on FPGAs: State-of-the-art implementations and attacks". *ACM Transactions on Embedded Computing Systems (TECS) , 3* (3), 534-574.
- T. Wollinger, M. Wang, J. Guajardo, and C. Paar. (April 13–14 2000). "How well are high-end DSPs suited for the AES algorithms?". *The Third Advanced Encryption Standard Candidate Conference* (pp. 94–105). New York, USA: National Institute of Standards and Technology.
- *TIEMPO*. (s.d.). Récupéré sur www.tiempo-ic.com
- U. Maurer, Y. Yacobi. "A Remark on a Non-interactive Public-Key Distribution System". *EUROCRYPT 1992*, (pp. 458-460).
- U. Maurer, Y. Yacobi. (1996). "Non-interactive Public-Key Distribution System. *Des. Codes Cryptography , 9*, 305-316.
- V. Betz and J. Rose. (1997). "VPR : A new packing, Placement and Routing Tool for FPGA researcher". *Int'l Workshop on FPL*, (pp. 213-222).
- V. Carlier, H. Chabanne, E. Dottax and H. Pelletier. (2004). *"Electromagnetic side channels of an FPGA implementation of AES"*. Cryptology ePrint Archive , (145).
- Victor Miller. (1986). "Use of an elliptic curves in cryptography". *Lecture Notes in Computer Science - CRYPTO'85 , 218*, 417-426.
- W. Diffie, M. Hellamn. (novembre 1976). "New directions in cryptography". *IEEE Transactions on Informatin Theory* , 644-654.
- Y. Monnet, M. Renaudin and R. Leveugle. (2006). "Designing resistant circuits against Malcious faults injection using asynchronous logic". *Transactions on computers , 55*, 1104-1115.
- Z. Yu and S. Furber. (2003). "An investigantion into the security of self-timed cicruits". *ASYNC*, (pp. 206-215).

Bibliographie de l'auteur

Journaux

1. Chaudhuri S., Guilley S., Hoogvorst Ph., Danger J.-L., Beyrouthy T., Razafindraibe A., Fesquet L., Renaudin M. : "An Asynchronous FPGA Architecture for Cryptographic Applications" ACM Transactions on Reconfigurable Technology and Systems, submitted 5 June 2008, accepted.
2. Beyrouthy T., Fesquet L.: "A secured technology mapping algorithm for an asynchronous secured FPGA". In submission in IEEE Circuit and Systems TCAS.
3. Beyrouthy T., FesquetL. : "BANYAN NETWORK-BASED FPGA : An alternative to implement secure applications", In Submission in Computer architecture Letters

Conférences internationales avec actes

4. Beyrouthy T., Razafindraibe A., Fesquet L., Renaudin M., Hoogvorst P., Guilley S., Chaudhuri S., Danger J.-L.: "A novel asynchronous e-FPGA architecture for security applications" International Conference onField-Programmable Technology 2007 (ICFPT'07) Field-Programmable Technology (ICFPT'07), Kokurakita, Kitakyushu, Japan, December 12th - 14th, 2007
5. Beyrouthy T., Fesquet L., Razafindraibe A., Chaudhuri S., Guilley S., Hoogvorst P., Danger J.-L., Renaudin M. : "A Secure Programmable Architecture with a Dedicated Tech-mapping Algorithm: Application to a Crypto-Processor". 23rd International Conference on Design of Circuits and Integrated Systems (DCIS'08), Grenoble, France, November 12-14, 2008
6. Hoogvorst P., Beyrouthy T., Guilley S., Razafindraibe A., Fesquet L. : "A Reconfigurable Cell for a Multi-Style Asynchronous FPGA". Reconfigurable Communication-centric SoC (RecoSoC'07), Montpellier, 18-20 juin, 2007

7. Guilley S., Chaudhuri S. , Sauvage L., Danger J.-L. , Beyrouthy T., Fesquet L. : " Updates on the Potential of Clock-Less Logics to Strengthen Cryptographic Circuits against Side-Channel Attacks" ICECS'09

Conférences internationales sans acte

8. Beyrouthy T., Fesquet L. : "DPA robust S-BOX implementation on a secure asynchronous FPGA" Workshop on Cryptographic Architectures embedded in reconfigurable devices Cryptarchi'09, Prague, Tchèque June 24-27.
9. Beyrouthy T., Fesquet L. : "A secure asynchronous configurable cell: an embedded programmable logic for smartcards" Workshop on Cryptographic Architectures embedded in reconfigurable devices CryptArchi'08, Tregastel, France, June 1-4, 2008
10. Beyrouthy T., Fesquet L. : "Asynchronous FPGA for secure applications", PhD Forum, DATE 2008 Nice-France

Documentation web

11. Hoogvorst P., Guilley S., Chaudhuri S., Danger J.-L., Beyrouthy T., Fesquet L. : "A Reconfigurable Programmable Logic Block for a Multi-Style Asynchronous FPGA resistant to Side-Channel Attacks", CoRR 2008, vol a0809.3942

RÉSUMÉ

Cette thèse porte sur la conception et la validation d'un FPGA dédié à des applications sensibles nécessitant un haut niveau de sécurité et de confidentialité. Les FPGAs usuels présentent de nombreuses failles vis-à-vis de la sécurité :

1- Ils ne permettent pas une implémentation efficace des circuits logiques alternatifs, tels que les circuits asynchrones.

2- Le placement et le routage d'un circuit ne peuvent être complètement maîtrisés pour garantir une conception sécuritaire.

3- Ils ne sont pas protégés contre les attaques par canaux cachés tels que la DPA, l'EMA ou la DFA.

Afin de lever ces obstacles technologiques, les travaux entrepris dans cette thèse ont permis de proposer une architecture autorisant la programmation de différents styles de circuits asynchrones, de disposer d'un système de programmation compatible avec les objectifs de sécurité et d'une conception garantissant un haut niveau de protection vis-à-vis des attaques citées ci-dessus. Enfin, une validation matérielle du prototype a permis d'appréhender la pertinence des modèles développés.

MOTS CLÉS

Logique asynchrone, Circuits programmables asynchrones 'a-FPGA', Circuits Quasi Insensibles aux Délais 'QDI', Attaques par canaux cachés 'SCA', Data Encryption Standard 'DES'.

ABSTRACT

This thesis focuses on the design and the validation of an embedded FPGA dedicated to critical applications which require a high level of security and confidentiality. Nowadays FPGAs exhibit many weaknesses toward security:

1- They are not intended to efficiently support alternative styles of circuits such as asynchronous circuits.

2- The place and route flow is not completely manageable by the user in order to target our security goal.

3-They are not protected against side channel attacks such as DPA, EMA or DFA.

In order to overcome these technological problems, the work presented in this thesis proposes an architecture that supports the programming of different styles of asynchronous circuits. In addition, it presents a secure programming system and a design that ensures a high-level of security against the attacks mentioned above. Finally, the circuit prototype has been evaluated in order to validate the relevance of the proposed solutions.

KEYWORDS

Asynchronous logic, Asynchronous programmable circuits 'a-FPGA', Quasi-Delay-Insensitive circuits 'QDI', Side Channel Attacks 'SCA', Date Encryption Standard 'DES'.

ISBN : 978-2-84813-141-2