Etude et modélisation de circuits résistants aux attaques non intrusives par injection de fautes

Y. Monnet

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : Microélectronique

préparée au laboratoire TIMA

dans le cadre de **l'Ecole Doctorale «Electronique, Electrotechnique, Automatique, Traitement du Signal »**

présentée et soutenue publiquement

par

Yannick Monnet

le 3 Avril 2007

ETUDE ET MODELISATION DE CIRCUITS RESISTANTS AUX ATTAQUES NON INTRUSIVES PAR INJECTION DE FAUTES

Directeur de thèse: Marc Renaudin
Co-directeur de thèse: Régis Leveugle

JURY

Mme. Florence Maraninchi	, Présidente
M. Stanislaw Piestrak	, Rapporteur
M. Jean-Jacques Quisquater	, Rapporteur
M. Marc Renaudin	, Directeur de thèse
M. Régis Leveugle	, Co-encadrant
M. Laurent Sourgen	, Examinateur

REMERCIEMENTS

Les travaux de cette thèse ont été réalisés au sein du laboratoire TIMA, sur le site Viallet de l'Institut National Polytechnique de Grenoble. Je remercie le directeur du laboratoire Bernard Courtois pour son accueil.

Je remercie Marc Renaudin, mon directeur de thèse, professeur à l'Institut National Polytechnique de Grenoble, pour m'avoir fait confiance depuis de nombreuses années. Que de chemin parcouru depuis 2000 et cette machine virtuelle Java !

Je remercie également Régis Leveugle, professeur à l'institut National Polytechnique de Grenoble, pour avoir co-encadré mes travaux de thèse. Ses conseils et son soutien m'ont été extrêmement précieux.

Je souhaite remercier les membres de mon jury de thèse:

M. Stanislaw Piestrak, professeur à l'université Paul Verlaine de Metz, pour avoir accepté de rapporter ma thèse et pour son aide apportée à la version finale de mon manuscrit.

M. Jean-Jacques Quisquater, professeur à l'université Catholique de Louvain, pour m'avoir fait l'honneur de participer à mon jury de thèse en tant que rapporteur.

Mme. Florence Maraninchi, professeur à l'institut National Polytechnique de Grenoble, pour m'avoir fait l'honneur de présider mon jury de thèse.

M. Laurent Sourgen, directeur du groupe Architectures & Technologies, division SmartCards de STMicroelectronics à Rousset, pour avoir accepté d'être examinateur de ma thèse.

Merci à toutes les personnes qui ont participé au projet DURACELL (mais qui a trouvé ce nom de projet ??) et qui ont contribué à valoriser ce travail.

Je remercie Laurent Fesquet et Gilles Sicard, pour tous les moments passés en leurs compagnies au sein du groupe CIS. Gilles, le vol du Picooz en N120 sur fond de supercopter restera inoubliable !

J'adresse mes remerciements aux personnes de TIMA et CMP qui m'ont épaulé pendant ces années. Merci en particulier à Isabelle, Chantal, Anne-Laure, Joelle, et Sophie M. pour leur gentillesse.

Une pensée pour tous les anciens du groupe CiS, en particulier Kamel, Manu (ahh ... la bonne époque des soirées octet !), Fabien et ses peluches, Dhanistha, Salim et tous les autres !

Et puis dans le désordre, merci à: Greg et David, capables d'inventer des paris toujours plus débiles mais drôles; Estelle pour sa gentillesse; Cédric et ses tongues; Yann qui m'a légué un bien lourd fardeau (t'inquiètes pas je continue à entretenir la légende); Aux établissements "Coste" et "La vieille" pour les sandwichs de midi; au Syfax pour les Kebabs à 23h les soirs de rédaction de publis. Merci à Nico pour sa tente; Vivian pour son aide en C++; Merci à Fraidy, mon cher vis-à-vis, pour qui j'ai beaucoup d'estime et qui m'a beaucoup aidé pendant ces années de thèse; A Eslam pour les discussions intéressantes que nous avons eu et pour ce succulent repas égyptien, un soir de fin de ramadan.

Un paragraphe spécialement pour Livier, que j'aime énormément, et avec qui je partage des souvenirs inoubliables aux quatre coins du monde. Sans elle je ne connaîtrais ni la troisième pharmacie la plus vieille d'Europe à Dubrovnik, ni la maison la plus ancienne de Reykjavik en Islande, ni le chakra associé à Jupiter de Cracovie, ni les fabuleux gâteaux hongrois de Dravazabolcs.

Merci à Bertrand pour tous les bons moments passés en sa compagnie: kangourous, jet ski, réseaux de Petri (comprendre: bières à Eischstatt), soirées diverses et variées, chaînes de Markov (comprendre: pauses café). Un grand merci à Sophie pour tous les moments

partagés ensembles, que ce soit dans les nombreuses soirées ou sur la plage du Grau-du-Roi. A Aurélien et Audrey, pour les nombreuses soirées (décidément …), et tous les bons moments passés ensembles ! Merci à toute l'équipe de préparation de mon pot de thèse (Claudia, Livier, Sophie, Audrey). Des vrais pros !

Pour terminer, un grand merci à ma famille, en particulier mon père et ma mère, qui m'ont toujours soutenu, Karine, Raph, et un gros bisou à Emma et Léo (mon ptit clone) qui sont à peine plus vieux que ces travaux de thèse...

TABLE DES MATIÈRES

LISTE DES FIGURES V

LISTE DES TABLEAUX VII

INTRODUCTION GENERALE: CONTEXTE ET MOTIVATIONS 1

CHAPITRE I
CRYPTANALYSE MATERIELLE : LES ATTAQUES PAR FAUTES 5

1.1 Introduction 5

1.2 Introduction sur la cryptographie 6

1.2.1 Le chiffrement/déchiffrement 6

1.2.2 Exemple de cryptage symétrique : le DES 7

1.2.3 Exemple de cryptage asymétrique : le RSA 9

1.3 Définition de la cryptanalyse 9

1.3.1 La cryptanalyse logicielle 11

1.3.2 La cryptanalyse théorique 11

1.3.3 La cryptanalyse matérielle 12

1.4 Les attaques par fautes 17

1.4.1 Attaque sur le RSA 17

1.4.2 Attaque sur le DES 19

1.4.3 Autres attaques 20

1.5 Conclusion 20

CHAPITRE II
L'INJECTION DE FAUTES ET LES MODELES DE FAUTES 23

2.1 Introduction 23

2.2 Les moyens d'injection de fautes 24

2.2.1 Les fautes naturelles 24

2.2.2 Les fautes intentionnelles 24

2.3 Les modèles de fautes au niveau logique 26
2.3.1 Introduction 26
2.3.2 Les fautes de délai (ou fautes de retard) 27
2.3.3 Les fautes transitoires (SET) 28
2.3.4 Les basculements mémoire (SEU) 28
2.4 Paramètres et extensions 28
2.5 Etat de l'art sur le durcissement (synchrone) 29
2.5.1 Introduction 29
2.5.2 Terminologie 30
2.5.3 Détection de fautes 31
2.5.4 Recouvrement 35
2.5.5 Tolérance aux fautes 35
2.6 Conclusion 37

CHAPITRE III
LES CIRCUITS ASYNCHRONES : CARACTERISTIQUES ET SECURITE 39

3.1 Présentation des circuits quasi insensibles aux délais 39
3.1.1 Introduction 39
3.1.2 Les différentes classes de circuits asynchrones 40
3.1.3 Les circuits quasi insensibles aux délais (QDI) 41
3.1.4 Synthèse des circuits asynchrones 46
3.2 Les circuits asynchrones et la sécurité 48
3.2.1 État de l'art sur la résistance des circuits asynchrones aux attaques par canaux cachés 48
3.2.2 État de l'art sur la résistance des circuits asynchrones aux attaques par fautes et leur durcissement 49
3.3 Utilisation des modèles de fautes dans le contexte des circuits asynchrones 53
3.3.1 Les fautes de délais 53
3.3.2 Les fautes transitoires 54
3.3.3 Les Soft Errors 55
3.3.4 Synthèse 55
3.4 Conclusion 56

CHAPITRE IV
ANALYSE DU COMPORTEMENT DES CIRCUITS ASYNCHRONES QDI EN PRESENCE DE FAUTES 59

4.1 Introduction 59

4.2 Analyse de sensibilité aux fautes transitoires 60

4.2.1 Définition du critère de sensibilité pour une porte de Muller 60

4.2.2 Définition du critère de sensibilité pour un circuit 61

4.2.3 Implantation de l'algorithme 61

4.2.4 Affinement du critère de sensibilité : Validation/Invalidation d'états 62

4.2.5 Implantation de l'outil 64

4.2.6 Exemple d'analyse de sensibilité 66

4.2.7 Conclusion 67

4.3 Analyse de sensibilité aux Soft Errors 68

4.3.1 Introduction 68

4.3.2 Le « token game » 69

4.3.3 Définition des jetons 70

4.3.4 Définition des règles d'évolution 72

4.3.5 Le modèle d'injection de fautes 74

4.3.6 Simulation symbolique du circuit 76

4.3.7 Exemple 78

4.3.8 Exploitation de la faille 80

4.3.9 Conclusion 81

4.4 Schéma global d'analyse du comportement 82

4.5 Conclusion 83

CHAPITRE V
PROPOSITIONS DE TECHNIQUES DE DURCISSEMENT DES CIRCUITS QDI CONTRE LES ATTAQUES PAR FAUTES 85

5.1 Introduction 85

5.2 Durcissement contre les fautes transitoires 86

5.2.1 Duplication des parties calculatoires 86

5.2.2 Technique de synchronisation des rails 87

5.2.3 Technique de synchronisation avec un circuit de contrôle 91

5.3 Durcissement contre les Soft Errors 92

5.3.1 Méthode de détection par blocage du circuit 92
5.3.2 Détection par alarmes 93
5.3.3 Améliorer la détection grâce aux synchronisations 93
5.4 Conclusion 94

CHAPITRE VI
EVALUATION EXPERIMENTALE DE LA SECURITE DES CIRCUITS QDI ET DES CONTRE-MESURES PROPOSEES 95
6.1 Introduction 95
6.2 Les circuits cryptographiques DES : circuit de référence et circuit durci 96
6.2.1 Architecture globale 96
6.2.2 Architecture du circuit de référence 98
6.2.3 Architecture du circuit durci 100
6.2.4 Caractéristiques des circuits 101
6.3 Mise en œuvre de la plateforme laser 103
6.3.1 Description du dispositif de test 103
6.3.2 Localisation temporelle 104
6.3.3 Localisation spatiale 105
6.3.4 Niveau d'énergie 105
6.3.5 Durée du tir 105
6.3.6 Campagnes d'injection de fautes 106
6.4 Résultats expérimentaux : évaluation des contre-mesures 106
6.4.1 Technique de synchronisation des rails 106
6.4.2 Alarmes 109
6.4.3 Blocage du circuit 110
6.5 Résultats expérimentaux : évaluation de sécurité des circuits QDI 111
6.5.1 Attaque sur les S-Boxes 111
6.5.2 Attaque sur le compteur 112
6.5.3 Compréhension des phénomènes physiques 119
6.6 Conclusion 120
CONLUSION ET PERSPECTIVES 121
BIBLIOGRAPHIE 125
BIBLIOGRAPHIE DE L'AUTEUR 133

LISTE DES FIGURES

Figure 1.1	Schéma de Feistel et la fonction *f* du DES	8
Figure 1.2	Attaques cryptanalytiques et les compétences requises	11
Figure 2.1	Architecture Duplication et Comparaison	29
Figure 2.2	Architecture de contrôle concurrent basé sur la redondance temporelle	32
Figure 2.3	Architecture TMR	36
Figure 2.4	Réalisation d'une machine à états avec code de Hamming	37
Figure 3.1	Différentes classes de circuits asynchrones	40
Figure 3.2	Principe du protocole quatre phases	42
Figure 3.3	Codage trois états	43
Figure 3.4	Porte de Muller à deux entrées	44
Figure 3.6	Structure des modules asynchrones	45
Figure 3.7	Réalisation de la fonction logique XOR en double rail	46
Figure 3.8	Flot de conception de l'environnement TAST	47
Figure 3.9	Codage double rail avec code d'alarme	51
Figure 3.10	Insertion et détection d'une alarme	52
Figure 3.11	Faute transitoire mémorisée dans la partie calculatoire	55
Figure 3.12	Injection de fautes dans un circuit asynchrone	56
Figure 4.1	Une porte de Muller à 2 entrées 1-sensible à 1 (a) et une porte à 5 entrées 3-sensible à 0 (b)	60
Figure 4.2	Evolutions possibles des états d'une porte de Muller pendant une exécution sans faute	64
Figure 4.3	Half Buffer	64
Figure 4.4	Flot de conception et intégration de l'outil d'analyse	62
Figure 4.5	Architecture du module analysé (a) avant et (b) après durcissement	66

Figure 4.6 Représentation de 3 étages d'un circuit 70
Figure 4.7 Règles d'évolution 73
Figure 4.8 Structure de rendez-vous 74
Figure 4.9 Structure de branchement 74
Figure 4.10 Etapes de la simulation symbolique 77
Figure 4.11 Machine à états à 3 étages 78
Figure 4.12 Etape 3 (Enumération) 79
Figure 4.13 Etape 4 (Injection de fautes) 79
Figure 4.14 Etape 5 (énumération des états fautés) 80
Figure 4.15 Résumé du comportement d'un circuit QDI en présence de fautes 83
Figure 5.1 Durcissement par duplication de la partie calculatoire 86
Figure 5.2 Porte de Muller durcie par la duplication de la partie calculatoire 87
Figure 5.3 Durcissement par la méthode de synchronisation des rails 88
Figure 5.4 Exemple d'implantation de la synchronisation des rails 89
Figure 5.5 Synchronisation d'un canal avec un circuit de contrôle redondant 92
Figure 6.1 Architecture générale du circuit DES 96
Figure 6.2 Interfaces sur 1 bit 98
Figure 6.3 Architecture du DES de référence 99
Figure 6.4 Architecture du DES durci 100
Figure 6.5 Circuit durci - (a) masques (b) plan de masse 102
Figure 6.6 Carte de test du circuit DES 104
Figure 6.7 Plateforme laser de Gemalto 104
Figure 6.8 Séquence de test 106
Figure 6.9 Compteur du DES 112
Figure 6.10 Comportement d'un démultiplexeur en fonctionnement normal ou en présence d'une faute 113
Figure 6.11 Réduction de rondes non détectée sur le DES durci 118

LISTE DES TABLEAUX

Tableau 4.1 Analyse de sensibilité pour une cellule du bloc non durci 67

Tableau 4.2 Analyse de sensibilité pour une cellule du bloc durci 67

Tableau 5.1 Analyse de sensibilité pour une porte de Muller du DES de référence 90

Tableau 5.2 Analyse de sensibilité pour une porte de Muller du DES durci 91

Tableau 6.1 Registres d'alarme du DES durci 98

Tableau 6.2 Caractéristiques des circuits DES 102

Tableau 6.3 Résultats de tolérance aux fautes 108

Tableau 6.4 Résultats de résistance aux fautes 109

Tableau 6.5 Blocages constatés dans le module de gestion des sous-clés et sensibilité des portes de Muller 111

INTRODUCTION GENERALE: CONTEXTE ET MOTIVATIONS

Dans un contexte d'ouverture, de mobilité et d'ubiquité croissante, la sécurité des systèmes de communication et d'information va reposer sur des architectures complexes associant logiciels et matériels. Afin de ne pas devenir un frein au développement des systèmes d'informations, les composants matériels utilisés doivent prendre en compte la problématique de la sécurité dès les premières phases de la conception.

Dans le domaine des systèmes sécurisés tels que la carte à puce ou le chiffrement, il existe une course poursuite permanente opposant les pirates et les fournisseurs d'équipements et de services. Le domaine de la cryptanalyse a été marquée ces dernières années par la découverte de nouvelles classes d'attaques, dont fait partie l'attaque par injection de fautes (Differential Fault Analysis : DFA).

Le principe est de perturber le fonctionnement d'un circuit en injectant des fautes dans sa partie combinatoire ou séquentielle. L'attaquant peut provoquer un dysfonctionnement temporaire du circuit lui permettant d'accéder à des zones mémoires protégées, ou de retrouver des informations secrètes telles que les clés cryptographiques.

Le travail de thèse s'inscrit en partie dans le cadre du projet RNRT Duracell dont les partenaires sont Gemalto, TIMA, Thales Communications et iRoC Technologies. Ce projet vise à développer des outils et des techniques, liés à une méthodologie de conception, destinés à rendre les circuits, ou certains blocs sensibles de ces circuits, tolérants aux attaques par injection de fautes. Dans le cadre de ce travail de thèse, on s'intéressera en particulier à étudier la modélisation et la conception de circuits sans horloge ou asynchrones résistants aux attaques par injection de fautes.

Les premières analyses montrent que les circuits asynchrones ont un comportement particulier en présence de fautes et sont par construction très robustes vis-à-vis de ce type d'attaques. Mais peu de travaux de recherche ont été effectués pour analyser de façon plus complète leurs forces et leurs faiblesses.

Le premier chapitre de ce manuscrit présente les différents types d'attaques existantes. On se focalise particulièrement sur les attaques non intrusives et sur les attaques par injection de fautes. Plusieurs attaques théoriques sont décrites après avoir introduit quelques notions de base en matière de cryptographie.

Le deuxième chapitre est consacré à l'injection de fautes et aux modèles de fautes. Il y a deux façons d'aborder l'injection de faute : l'injection de faute naturelle due aux rayonnements et l'injection de faute intentionnelle par un attaquant souhaitant provoquer un dysfonctionnement. Ces deux catégories de fautes, correspondant respectivement aux domaines du test en ligne et de la sécurité, ont un certain nombre de points communs qui justifient un rapprochement des idées en matière de modèles de fautes, de contre-mesures et d'analyse de comportement. Bien que ce travail soit orienté vers un objectif de sécurité des circuits, les fautes naturelles seront d'une certaine manière aussi considérées.

Ce chapitre définit également les différents modèles de fautes qui seront utilisés au cours de ce travail, en faisant des choix sur le niveau d'abstraction auquel nous souhaitons représenter une faute. Un bref état de l'art sur les techniques de protection employées dans le monde synchrone est enfin présenté.

Le chapitre III introduit la technologie asynchrone et en particulier les circuits quasi insensibles aux délais qui ont été étudiés au cours de ces travaux. Un accent particulier est mis sur leurs propriétés face aux problèmes liés à la sécurité et particulièrement aux injections de fautes. Un état de l'art des travaux effectués sur les circuits asynchrones à ce sujet est présenté.

Les propriétés particulières des circuits asynchrones nécessitent une analyse spécifique pour comprendre et maîtriser leur comportement en présence de fautes. Le chapitre IV propose dans ce sens deux analyses complémentaires, l'une basée sur une analyse de sensibilité dynamique et l'autre basée sur une analyse formelle.

Cette étude de sensibilité conduit à isoler des forces mais aussi des faiblesses qui doivent être éliminées. Le chapitre V propose des contre-mesures exploitant les propriétés des circuits asynchrones, et visant à durcir les circuits. Ces contre-mesures sont validées en simulation grâce à des outils d'analyse spécifiques.

Le chapitre VI propose de confronter la théorie et la pratique. Les contre-mesures sont implantées dans un circuit cryptographique DES asynchrone, et validées sur une plateforme de test avec une injection de fautes par laser. Ce test pratique nous permet de mesurer l'efficacité des méthodes employées, d'évaluer

la technologie asynchrone face aux injections de fautes, et enfin de mesurer le potentiel de ce type d'attaque.

Enfin, les différents points abordés dans ce travail sont résumés dans une conclusion, et plusieurs perspectives sont proposées.

CHAPITRE I

CRYPTANALYSE MATÉRIELLE : LES ATTAQUES PAR FAUTES

1.1 Introduction

Longtemps resté un domaine réservé aux services des gouvernements et des services d'espionnage (contre-espionnage) des militaires, le domaine de la cryptanalyse est devenu depuis les années 70 un domaine de recherche grand public porté par de nombreux laboratoires académiques.

En effet, le développement considérable des réseaux de communication a favorisé l'expansion de nouveaux moyens de paiements à distance avec notamment la carte à puce, le e-commerce, et le e-banking. Ces nouveaux types de communications nécessitent de plus en plus de moyens de transactions dits sûrs, pouvant résister à diverses attaques cryptanalytiques. De cette lutte effrénée entre d'une part les cryptographes qui mettent en place des systèmes ou algorithmes cryptographiques, et d'autre part les cryptanalystes qui développent des techniques pour briser les systèmes de sécurité, un net avantage est donné au second depuis le développement de la cryptanalyse matérielle. Ces nouvelles méthodes de cryptanalyse ont montré leur efficacité sur de nombreux produits commerciaux de type carte à puce.

Ce chapitre est consacré à l'analyse des attaques par injection de fautes. Dans un premier temps nous introduisons quelques notions de base de la cryptanalyse et des attaques matérielles. Parmi celles-ci, nous nous intéressons particulièrement à l'attaque par injection de fautes et aux différentes cryptanalyses existantes.

1.2 Introduction sur la cryptographie

1.2.1 Le chiffrement/déchiffrement

La cryptographie est une science consacrée à la protection des données en les rendant incompréhensibles sauf pour son destinataire. Elle a pour objectif la conception et l'analyse des mécanismes permettant d'assurer l'intégrité, l'authenticité, la confidentialité et le non désaveu des données et des communications. Pour cela elle utilise des processus cryptographiques.

Un processus cryptographique est une transformation d'un message appelé **texte en clair** en un message incompréhensible appelé **texte chiffré**. Ce processus est nommé **chiffrement** (**cryptage** ou **encryption**) et lorsque le processus inverse est réalisé, on parle de **déchiffrement** (**décryptage**).

$$\left.\begin{array}{ll} \textit{Chiffrement} & E(M) = C \\ \textit{Déchiffrement} & D(C) = M \end{array}\right\} \; D(E(M)) = M$$

E est la fonction de chiffrement,

D la fonction de déchiffrement,

C le texte chiffré et M le texte en clair.

Les fonctions de chiffrement ou de déchiffrement représentent des fonctions mathématiques encore appelées algorithmes cryptographiques (cryptosystèmes). La sécurité d'un cryptosystème dépend de deux paramètres : la sûreté de l'algorithme et la longueur de la clé utilisée. On entend par sûreté le fait qu'il n'existe pas de meilleur moyen de casser l'algorithme que d'utiliser une **attaque exhaustive** (essais de toutes les combinaisons de clés possibles).

La clé cryptographique est généralement mixée avec les données d'entrée des algorithmes cryptographiques par l'opérateur logique « ou exclusif » (XOR). 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 $3{,}6*10^{16}$ tentatives. En faisant l'hypothèse qu'un superordinateur peut essayer 1 million de clés par seconde, 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 : les algorithmes à clé secrète (ou **cryptage symétrique**) et les algorithmes à clé publique (ou **cryptage asymétrique**).

1.2.2 Exemple de cryptage symétrique : le DES

Dans un système de cryptage symétrique, la même clé est utilisée pour le chiffrement et le déchiffrement des messages. On parle alors de chiffrement à clé secrète. Le **DES** (**D**ata **E**ncryption **S**tandard) [FIPS46a] est l'un des algorithmes à clé secrète les plus utilisés. Il a été élaboré dans les années 70 par IBM et adopté en 1977 par le gouvernement Américain comme standard de protection d'informations non secrètes mais confidentielles.

L'algorithme DES transforme un bloc de 64 bits en un autre bloc de 64 bits. Il manipule des clés individuelles de 56 bits, représentées par 64 bits (avec un bit de chaque octet servant pour le contrôle de parité). L'algorithme est basé sur le principe des schémas de Feistel [Schn01]. D'une manière générale, on peut dire que DES fonctionne en trois étapes :

- Permutation initiale et fixe d'un bloc.
- Le résultat est soumis à 16 itérations (**rondes**) d'une transformation, ces itérations dépendent à chaque ronde d'une autre clé partielle de 48 bits. Cette clé de ronde intermédiaire est calculée à partir de la clé initiale de l'utilisateur (grâce à un réseau de tables de substitution et d'opérateurs XOR). Lors de chaque ronde, 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 (celui qui s'étend du bit 32 au bit 63) subira une transformation f (Figure 1.1).
- Le dernier résultat de la dernière ronde est transformé par la fonction inverse de la permutation initiale.

Le DES utilise huit tables de substitution, les S-Boxes, qui contribuent à la « confusion » en rendant l'information chiffrée inintelligible. Les S-Boxes permettent de casser la linéarité de la structure de chiffrement.

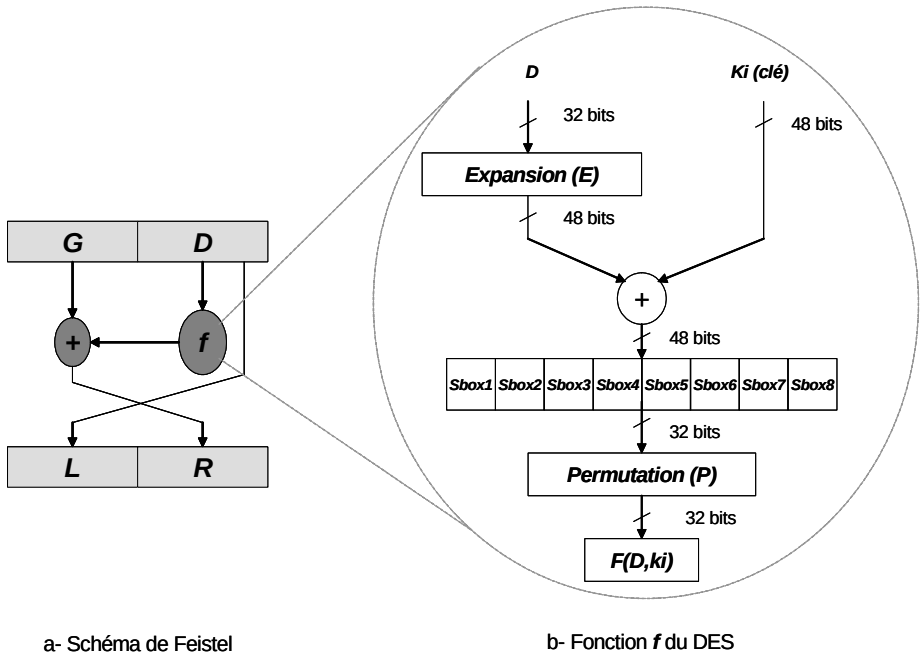


Figure 1.1 Schéma de Feistel et la fonction *f* du DES

Chaque S-Box prend une variable de 6 bits en entrée et produit une sortie de 4 bits. Les valeurs présentes dans les S-Boxes ont été choisies de manière à résister aux attaques, par divers moyens comme l'utilisation de fonctions courbes. Dans le cas de DES, il a été prouvé que les tables avaient été conçues de manière à résister à la cryptanalyse différentielle (technique qui ne sera publiée que bien des années plus tard) [Biha91a].

Afin d'améliorer la sûreté du DES (clé trop petite), une variante du DES en triple DES a été adoptée [FIPS46b]. L'algorithme reste inchangé sauf que le processus de chiffrement (déchiffrement) s'effectue avec deux clés de 64 bits selon le schéma suivant :

***Triple DES* Chiffrement** : $DES^{Clé1}_{Chiffrement} \rightarrow DES^{Clé2}_{Déchiffrement} \rightarrow DES^{Clé1}_{Chiffrement}$

***Triple DES* Déchiffrement** : $DES^{Clé1}_{Déchiffrement} \rightarrow DES^{Clé2}_{Chiffrement} \rightarrow DES^{Clé1}_{Déchiffrement}$

Depuis 2001, l'algorithme de DES est appelé à être remplacé par le nouveau standard de chiffrement avancé des données (**AES** pour **A**dvanced **E**ncryption **S**tandard [FIPS197]). Cependant, le DES reste aujourd'hui encore très utilisé dans les applications industrielles.

1.2.3 Exemple de cryptage asymétrique : le RSA

Dans le mode de cryptage asymétrique, deux clés sont utilisées, une clé publique et une clé privée. On parle alors de chiffrement à clé publique. Ce type de chiffrement a été inventé en 1976 par W. Diffie et M. Hellman [Schn01] pour résoudre le problème de gestion des clés posé par les chiffrements symétriques. Dans ce type de chiffrement, tout le monde peut utiliser votre clé publique pour vous envoyer des messages chiffrés que seul vous pouvez déchiffrer avec votre clé secrète. Il offre également la possibilité de créer des signatures numériques.

La sécurité du **RSA** (du nom des inventeurs R. **R**ivest, A. **S**chamir et L. **A**dleman [Riv78]) est basée sur la difficulté de factoriser des grands nombres entiers en leurs produits de deux nombres premiers.

Soient p et q deux nombres premiers et d un entier tel que d soit premier avec $(p-1)*(q-1)$. Les nombres (p, q, d) représentent la clé secrète. La clé publique est composée du couple (n, e) telle que :

$$n = p * q$$
$$e = \frac{1}{d} \bmod((p-1).(q-1)) \qquad (1.1)$$

Si **M** est le message à chiffrer, alors le processus de chiffrement est réalisé en exécutant le calcul suivant :

$$C = M^e \bmod n \qquad (1.2)$$

Pour déchiffrer le message chiffré C, il faut utiliser la clé secrète d telle que :

$$M = C^d \bmod n \qquad (1.3)$$

Plus le nombre n est grand, plus il est difficile de retrouver ces facteurs premiers p et q. A ce jour, le RSA est réalisé avec des valeurs de plus de 1024 bits.

1.3 Définition de la cryptanalyse

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 ou les clés utilisée(s). L'objectif de cette science est de pouvoir mettre en évidence les faiblesses d'un cryptosystème. On parle alors de tentatives de déchiffrement ou d'attaques cryptanalytiques.

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

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

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

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

- 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.

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

Ces attaques reposent sur le principe de Kerckhoff [Stin96] 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.

De nos jours, la conception des systèmes nécessite la collaboration des compétences dans différents domaines (informatique, mathématique, microélectronique etc.). La figure 1.2 représente les classes et sous classes de différentes méthodes cryptanalytiques en fonction des compétences qu'elles mettent en œuvre. Dans chacune de ces méthodes on peut utiliser chacun des cinq modèles d'attaques génériques définis ci-dessus.

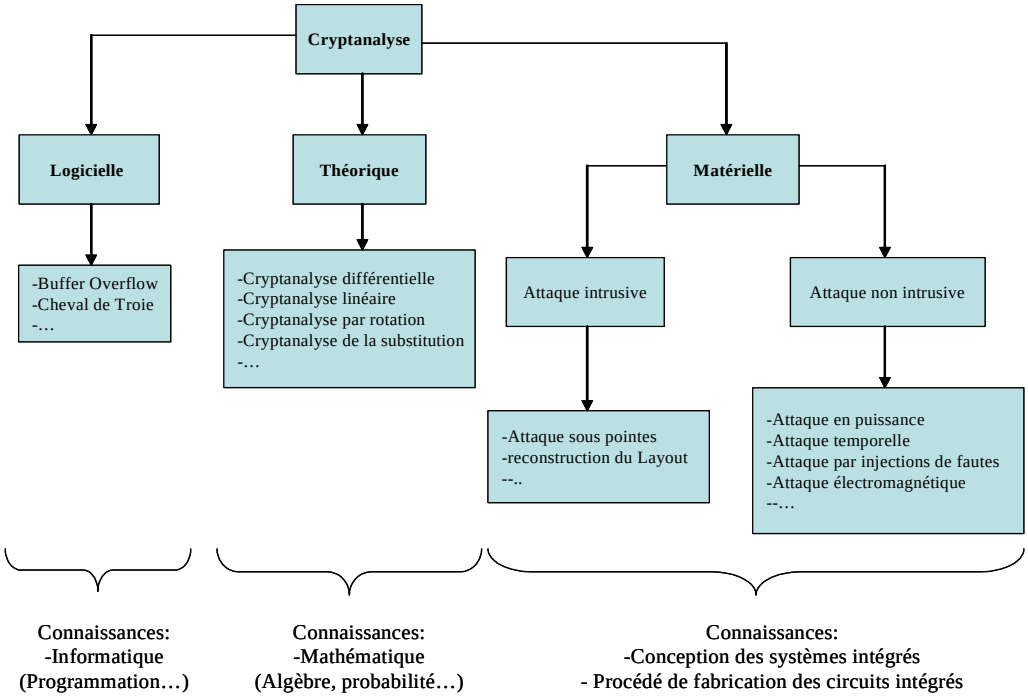


Figure 1.2 Les attaques cryptanalytiques et les compétences requises

1.3.1 La cryptanalyse 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 [Ross01] ou le débordement de pile [One96] 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.

1.3.2 La cryptanalyse théorique

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.

La **cryptanalyse linéaire** est une technique inventée par Matsui [Mats94]. Elle fut développée à l'origine pour casser l'algorithme du DES. Elle consiste à faire une approximation linéaire de l'algorithme de chiffrement en le simplifiant. Cette modélisation est faite en réalisant des attaques en textes clairs connus et clé constante. En augmentant le nombre de couples (texte clair, texte chiffré) disponibles, on améliore la précision de l'approximation et on peut en extraire la clé.

- Développée à la fin des années 80 par Eli Biham et Adi Shamir [Biha91a], la **cryptanalyse différentielle** consiste en l'étude sur la manière dont les différences entre les données en entrée affectent les différences de leurs sorties. Ce type d'attaque est très utilisé sur des algorithmes itératifs, type DES ou AES. Cependant, il fut montré que le DES était particulièrement résistant à cette attaque et que de petites modifications dans ses paramètres l'affaiblissaient. Ce constat a fait naître la rumeur que ses concepteurs (travaillant pour IBM) connaissaient déjà cette méthode dans les années 1970…

- En 1994, M. Hellman et K. Langford introduisent la **cryptanalyse différentielle linéaire** qui est une combinaison des deux approches [Lang94]. Avec cette attaque, ils purent casser avec 80% de réussite un DES de 8 rondes avec seulement 512 textes en clair et quelques secondes sur un PC. Ce résultat monte à 95% de succès avec 768 textes. Au final, il est possible d'obtenir 10 bits de la clé ou même plus selon le taux de réussite désiré et le nombre de messages à disposition (avec 768 textes, on peut obtenir 16 bits de la clé avec 85% de réussite).

Il est important de noter que ces attaques théoriques, et en particulier la cryptanalyse différentielle linéaire, peuvent être associées à d'autres types d'attaques comme les attaques par fautes. Comme nous le verrons par la suite, une attaque par faute peut engendrer une réduction du nombre de rondes d'un DES, permettant ensuite l'exploitation des résultats fautés par une cryptanalyse théorique.

1.3.3 La cryptanalyse matérielle

Cette branche de la cryptanalyse regroupe l'ensemble des techniques cryptanalytiques exploitant les vulnérabilités matérielles des circuits intégrés dédiés à des applications sécurisées. On distingue deux catégories d'attaques matérielles : les attaques intrusives et les attaques non intrusives.

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 reconstituer 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** 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, dans le sens inverse, les différentes étapes des procédés de fabrication des circuits intégrés [Ross96]. Ainsi les différentes couches de résines, des métaux, sont enlevées les unes après les autres par des solutions chimiques pour refaire le layout. Ce type d'attaque permet de lire par reconstitution les valeurs d'une mémoire [Samy02]. 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).

- L'identification des zones sensibles par la méthode de reconstruction du layout permet de réaliser des analyses en utilisant des **pointes de test**. L'utilisation des sondes ou pointes généralement consacrées aux tests de wafer (test sous pointes) [Moor00] permet la lecture des valeurs transitant sur des bus (données ou adresse) ou sur des entrées et sorties des cellules, bloc etc. A l'aide de ce type de matériel, on peut forcer des valeurs logiques sur certains nœuds de la puce afin de valider des instructions de test ou de comparaison donnant accès à des informations protégées. Dans certains composants, elle peut permettre de rétablir le fusible de programmation de la puce [Komm99, Gueu98].

De manière générale, les attaques intrusives nécessitent des moyens considérables et coûteux notamment avec l'utilisation d'un microscope optique (***FIB*** : ***F****ocused* ***I****on* ***B****eam*), 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.

1.3.3.2 Les attaques non intrusives

Contrairement aux attaques intrusives, les attaques non intrusives ne sont pas destructives. Selon les méthodes d'analyses misent en place, les attaques non intrusives peuvent être rangées en deux classes : les

attaques par injection de fautes, qui seront étudiées dans la section 1.4 et les attaques par analyse des signaux compromettants encore appelées attaques par canaux cachés.

Les attaques par canaux cachés sont basées sur la recherche et l'exploitation de toute corrélation entre les données manipulées par un circuit et ces signaux externes. Il peut s'agir par exemple des signaux d'alimentation ou des émissions électromagnétiques d'un circuit. Ils sont appelés signaux compromettants ou canaux cachés du fait qu'ils peuvent faire fuir des informations indésirables. En effet, la notion de canal est définie comme un endroit par lequel transite de l'information, et il est caché quand son utilisation est détournée du son fonctionnement initial. En fonction du signal compromettant utilisé, on les classe en attaque temporelle, en attaque électromagnétique et en attaque en puissance.

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***.

Les algorithmes possédant des fonctions de test, de branchement, de saut dépendant des données tant sur le plan logiciel (ligne d'instructions) que matériel (architecture), sont vulnérables aux analyses temporelles. D'autres modèles d'attaque sur le ***DES*** et sur l'***AES*** ont été proposés respectivement dans [Hevi99] et [Koeu99]. 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*** [FIPS197].

1.3.3.2.2 Les attaques électromagnétiques

Les attaques par analyses électromagnétiques sur les composants dédiés à des applications cryptographiques ont connu un développement considérable depuis la publication il y a quelques années par la ***NSA*** (***N**ational **S**ecurity **A**gency*) de rapports scientifiques classés secrets défense sur la cryptanalyse matérielle par effets électromagnétiques [NSA00].

La majorité des circuits numériques sont cadencés par un signal global d'horloge qui impose une synchronisation des traitements dans toutes les parties du circuit. A chaque cycle d'horloge (sur front

montant ou descendant), tous les signaux sont mémorisés dans des registres (ou mémoires) et de nouveaux calculs sont alors exécutés entraînant des appels en courant non négligeables. Dans ce type de circuit, le spectre électromagnétique est essentiellement dû à l'activité électrique des éléments logiques et des points mémoires constituant le circuit.

Les caractéristiques du champ électromagnétique (en terme de fréquence et amplitude) sont donc relatives à l'activité électrique et à la nature des données manipulées dans le circuit. Par conséquent, le principe de l'attaque électromagnétique est d'exploiter cette dépendance afin de déterminer toute corrélation entre les données traitées et les émissions électromagnétiques.

*** Mise en œuvre des attaques**

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 réalisées : 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 [Gand01].

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.

Des méthodes d'attaques simples ou différentielles par analyses électromagnétiques ont été introduites pour la première fois pas Quisquater sous les termes de ***SEMA*** (Simple Electromagnetic Attack) et de ***DEMA*** (Differential Electromagnetic Attack) [Quis00]. Ces attaques utilisent les mêmes algorithmes d'analyse que ceux utilisés par des attaques en puissances (***SPA*** : Single Power Analysis et ***DPA*** Differential Power Analysis), sauf que les analyses sont faites sur des courbes d'émissions électromagnétiques. Rao et Rohatgi ont montrés dans [Rao01] 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 implantations sécurisées contre la ***SPA*** ou la ***DPA*** ne le sont pas forcément contre la ***SEMA*** ou la ***DEMA*** et vice versa.

1.3.3.2.3 Les attaques en puissance ou attaques par analyse du courant

Contrairement aux attaques électromagnétiques qui exploitent l'activité électrique des circuits intégrés sous forme d'émissions électromagnétiques, les attaques en puissances 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 briser le secret. 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.

$$P_{charge} \neq P_{décharge} \Rightarrow i_{charge} \neq i_{décharge}$$

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 du 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.

1.4 Les attaques par fautes

Le principe des attaques par injections de fautes consiste à générer délibérément des fautes dans un composant en fonctionnement, l'exploitation des résultats erronés permettant de briser les sécurités du composant. Les fautes sont générées en faisant fonctionner le composant dans des conditions anormales. La réalisation des attaques par injections de fautes se déroule en deux phases : la génération de la faute et l'exploitation de la faute générée pour accéder aux données confidentielles du système.

La formalisation d'une attaque exploitant un modèle de fautes à été introduite pour la première fois en 1996 par Boneh, Demillo et Lipton avec l'attaque dite de «Bellcore» du nom de leur groupe de recherche [Bone97]. Cette attaque exploite à la fois un modèle de fautes et des faiblesses d'implantation des algorithmes cryptographiques comme c'est le cas de l'implantation du ***RSA*** en utilisant le théorème des restes chinois [Stin96]. En 1997, Biham et Shamir proposent une attaque sur le DES permettant de retrouver la clé avec moins de 200 textes clairs fautés [Biha97].

1.4.1 Attaque sur le RSA

Le ***RSA***, qui est l'algorithme à clé publique le plus utilisé pour des communications ou transactions sécurisées, utilise un modulo n de 1024 voir 2048 bits. Pour garantir la sûreté de l'information, il est nécessaire que n soit suffisamment grand pour que sa factorisation en p et q soit impossible à calculer. Une telle implantation est coûteuse en temps de calcul. Afin d'optimiser la vitesse d'exécution, des algorithmes adaptés aux calculs des produits modulaires sur des grands nombres sont utilisés avec notamment l'algorithme de Montgomery [Mont85, Kak96, Jeon97] et la réalisation par le théorème des restes chinois. Cette dernière réalisation est particulièrement vulnérable aux attaques par injection de fautes.

* **Rappel sur le théorème des restes chinois**:

Soient $m_1, m_2, \ldots m_n$ des entiers naturels supérieurs à 2, premiers entre eux deux à deux, et $a_1, a_2, \ldots a_n$ des entiers. Le système d'équation est :

$$\begin{aligned} x &= a_1 \bmod m_1 \\ x &= a_2 \bmod m_2 \\ &\ldots\ldots \\ x &= a_n \bmod m_n \end{aligned} \qquad (1.4)$$

Il admet une solution unique modulo $M = \prod_{i}^{n} m_i$ de la forme :

$$x = \sum_{i}^{n} a_i M_i y_i \bmod M$$

$$\text{avec } M_i = \frac{M}{m_i} \text{ et } y_i = M_i^{-1} \bmod m_i \text{ pour } 1 \le i \le n \tag{1.5}$$

L'application de ce théorème sur ***RSA*** permet de déduire la réalisation suivante : on calcule dans un premier temps les valeurs intermédiaires S_p et S_q telles que :

$$\begin{aligned} S_P &= m^{d_p} \bmod p \\ S_q &= m^{d_q} \bmod q \end{aligned} \text{ avec } \begin{cases} d_p = d \bmod (p-1) \\ d_q = d \bmod (q-1) \end{cases} \tag{1.6}$$

Le résultat de chiffrement S (ou déchiffrement) est obtenu par l'expression:

$$S = (a.S_p + bS_q) mod\ n$$

$$avec \quad a = \frac{n}{p}(q^{-1}\ mod\ p) \quad et \quad b = \frac{n}{q}(p^{-1}\ mod\ q) \tag{1.7}$$

L'apparition d'une faute sur le calcul de S_p ou sur celui de S_q permet de déterminer les nombres premiers p et q de n. Soit S_p^* la valeur obtenue avec l'erreur. Le résultat de chiffrement (déchiffrement) avec l'erreur est donné par :

$$S^* = (a.S_p^* + b.S_q) \bmod n \tag{1.8}$$

On déterminant le Plus Grand Commun Diviseur (***PGCD***) de n et S^*-S, on trouve la valeur de q :

$$PGCD(S^* - S; n) = q \tag{1.9}$$

En effet,

$$S^* - S = a.S_P^* + b.S_q - a.S_p - b.S_q = a.(S_P^* - S_q)$$
$$S^* - S = (q).(q^{-1} \bmod p).(S_p^* - S_p)$$

Comme les termes $(q^{-1} \bmod p)$ et $(S_P^* - S_p)$ sont inférieures à q et que $n=pq$, alors le PGCD de n et S^*-S est égal à q.

Il existe plusieurs variantes de cette attaque qui permettent de briser la majorité des cryptosystèmes à clé publique (***PKCs*** pour Public Key Cryptosystems) [Bao97, Mahe97, Biha97, Lens97]. La plus proche du modèle de Bellcore est celui de Lenstra [Lens97]. Dans ce modèle seule la connaissance du message à chiffrer ***M*** et du résultat ***S**** sont nécessaires pour trouver les facteurs de ***n***.

1.4.2 Attaque sur le DES

Des modèles d'attaques sur des protocoles cryptographiques symétriques (à clé secrète) ont été également développés par des cryptanalystes. Biham et Shamir ont développé l'attaque par analyse différentielle de fautes encore appelée ***DFA*** (Differential Fault Analysis) [Biha97] en partant du principe du modèle de Bellcore. L'attaque ***DFA*** consiste à analyser les différences entre deux résultats de chiffrement (ou déchiffrement) ***S*** (correct) et ***S**** (incorrecte par apparition d'une erreur) en utilisant le même message d'entrée et la même clé. En simulant des fautes dans une implantation du ***DES***, ils ont démontré qu'il est possible de retrouver toutes les clés de l'algorithme en utilisant 50 à 200 messages chiffrés.

* **Fonctionnement de l'attaque** :

On suppose sans perdre de généralité que seules la partie droite ($\mathbf{R_i}$) du calcul est affectée par une faute. La partie gauche ($\mathbf{L_i}$) n'est affectée par la faute qu'une fois l'échange effectué entre les deux blocs. Une faute peut être générée à n'importe quelle itération (round) i, et l'attaquant ne connaît pas le bit affecté. On suppose aussi que l'attaquant peut effectuer au moins deux encryptions avec le même texte clair (connu ou non) de façon à pouvoir comparer un résultat fauté avec un résultat correct.

Dans un premier temps, on identifie l'itération pendant laquelle la faute est survenue. Si la 16ème itération a été fauté, alors un seul bit diffère entre $\mathbf{R'_{16}}$ du résultat fauté et $\mathbf{R_{16}}$ du résultat correct. Ainsi la partie gauche du résultat fauté ne peut différer que par les bits de sorties des S-Boxes qui ont été activés

par un seul bit d'entrée différent. On peut ainsi deviner 6 bits de clé et éliminer les valeurs qui ne correspondent pas au résultat attendu. En moyenne pour la dernière itération, il y reste 4 valeurs possibles pour les 6 bits de clés de chaque S-Box activée.

On peut ainsi analyser des résultats fautés dans les $14^{ème}$, $15^{ème}$ et $16^{ème}$ rounds. On retrouve 48 bits de clé (la dernière sous-clé) en quelques dizaines de messages chiffrés, en injectant aléatoirement une faute dans n'importe quel round. Le nombre de fautés nécessaires est encore réduit si l'attaquant a la possibilité de choisir le moment d'injection (les derniers rounds) et la position de la faute.

L'attaque permet de trouver 48 bits des 56 bits de clé. Les derniers bits peuvent être retrouvés par une méthode exhaustive (256 possibilités) ou bien, puisque toutes les données du dernier round sont connues, par application de la même méthode sur l'analyse des rounds précédents. Cette méthode peut être utilisée pour l'attaque d'un triple DES (168 bits de clés) ou d'un DES utilisant des sous-clés indépendantes (768 bits de clés).

1.4.3 Autres attaques

Une seconde approche consiste à utiliser les propriétés des mémoires non volatiles. L'idée est de changer la valeur (les valeurs) d'un bit d'une mémoire (ROM ou registre). Par exemple, lors de la dernière itération du ***DES*** le cryptanalyste peut forcer à zéro certains bits du registre de gauche (***L***) et déterminer les valeurs d'entrées et de sorties de la fonction ***f*** du ***DES*** [Biha97].

Des modèles d'injection de fautes moins complexes dans leurs mises en œuvre ont été exploités par Ross et Kuhn dans [Ross96]. Ces types d'attaques sont particulièrement redoutables sur des instructions de saut, sur des compteurs, des boucles, des tests par comparaisons et des branchements sur conditions. Par exemple, la corruption du compteur d'itérations du DES permet une attaque encore plus rapide que celle présentée ci-dessus.

1.5 Conclusion

De nos jours, la conception des systèmes intégrés sécurisés nécessite non seulement l'utilisation d'algorithmes cryptographiques résistant à la cryptanalyse logicielle (cryptanalyse linéaire, différentielle etc.) mais également l'utilisation de méthodes d'implantation robustes contre les attaques matérielles. Le domaine de la conception d'un tel système requiert une étroite collaboration entre concepteurs de divers domaines, allant des mathématiques à l'informatique en passant actuellement par la microélectronique avec les attaques matérielles.

La problématique de la sécurité doit être considérée dans son ensemble, certaines attaques pouvant être combinées pour atteindre leurs objectifs. De plus, il est indispensable que l'implantation de contre-mesures visant à protéger un système contre un certain type d'attaque ne pénalise pas sa résistance face aux autres attaques. Par exemple, les contre-mesures envisagées pour se prémunir des attaques par fautes ne doivent pas rendre le système plus sensible aux attaques par canaux cachés.

Parmi les différentes attaques matérielles présentées dans ce chapitre, les attaques par fautes sont parmi les plus récentes mais aussi les plus efficaces. Parmi les articles scientifiques proposant ce type d'attaque, beaucoup sont théoriques et avancent un modèle de faute et un moyen d'injection de faute reposant sur de nombreuses hypothèses. Cependant, quelques travaux rapportent des mises en œuvre pratiques d'attaques par fautes, montrant que la menace est réelle.

Dans le contexte global de la sécurité des systèmes matériels, la logique asynchrone semble proposer des propriétés intéressantes pour concevoir des systèmes sécurisés. Ces travaux de recherche se focalisent uniquement sur les attaques par fautes. Nous allons d'une part évaluer la technologie asynchrone face à ce type d'attaque et d'autre part proposer des contre-mesures. Ces contre-mesures sont évaluées en utilisant des outils spécifiques dans le flot de conception des circuits. Puis l'ensemble de la méthode est validé par la pratique avec des campagnes d'injection de fautes par laser.

CHAPITRE II

L'INJECTION DE FAUTES ET LES MODÈLES DE FAUTES

2.1 Introduction

Le mauvais fonctionnement d'un circuit peut provenir d'une erreur de conception, d'un problème lors de la fabrication ou d'un problème survenant pendant l'exécution de l'application, soit à cause du vieillissement du circuit, soit à cause de son environnement (radiations, particules, etc. …), ou encore à cause d'un stress appliqué intentionnellement au circuit.

Les dysfonctionnements pouvant survenir dans un circuit sont représentés en fonction de leur origine, avec plus ou moins de précision, par de nombreux "modèles de fautes". Parmi les modèles existants, nous nous focalisons dans ce document sur les modèles de fautes représentatifs de moyens d'injections de fautes non intrusifs, c'est-à-dire sans effet destructeur sur le composant. On s'oriente donc vers des fautes **transitoires** ou **intermittentes** plutôt que vers des fautes **permanentes** qui sont plus représentatives des fautes liées par exemple aux défauts de fabrication.

On distingue d'autre part les fautes **naturelles** qui font l'objet d'études dans le domaine du test (particulièrement du test « en-ligne »), et les fautes **intentionnelles**, liées au domaine de la sécurité. Bien que ces deux domaines aient des espaces d'applications très différents, certains modèles de fautes classiquement employés dans le domaine du test peuvent être utilisés pour représenter certains types d'attaques.

Ces travaux s'orientent principalement vers le problème de la sécurité des circuits, et donc vers l'étude de l'effet des fautes intentionnelles. Cependant, les analyses de comportements et les méthodes de durcissement présentées dans ce manuscrit peuvent également s'appliquer à la protection des circuits contre les fautes naturelles. C'est pourquoi cet aspect est également considéré au cours du travail.

2.2 Les moyens d'injection de fautes

2.2.1 Les fautes naturelles

La première apparition des fautes dans un circuit suite au rayonnement cosmique a été remarquée en 1975 [Pick78] [Gor94]. A partir de là, plusieurs études ont été menées pour d'abord comprendre le mécanisme de la faute : comment elle intervient au niveau analogique [Dod94], pour enfin proposer des solutions de durcissement qui permettent d'éviter, de détecter, de réagir ou tout simplement de rendre la probabilité d'une faute très faible dans l'environnement d'utilisation du circuit.

Les applications spatiales et aéronautiques sont particulièrement sensibles à ce type de rayonnement car elles évoluent dans des environnements sensibles. Mais l'évolution des technologies dans le domaine "submicronique profond" (dimensions inférieures à 0,18 micron) s'accompagne par un fort accroissement de la sensibilité des circuits à différents parasites (rayonnements ou particules), même au niveau de la mer. Aujourd'hui, le problème est en passe de concerner les applications grand public.

2.2.2 Les fautes intentionnelles

Il existe de nombreux moyens d'injection de fautes dans un circuit, chacun ayant un coût et des propriétés spécifiques. En fonction du moyen d'injection de faute, on devra utiliser un modèle de faute avec des paramètres appropriés (multiplicité, durée des fautes, etc.). Les moyens proposés ici correspondent à des injections de fautes non intrusives.

2.2.2.1 Injection par variation de l'alimentation

Le principe de l'injection de faute repose sur la variation de la tension d'alimentation pendant un laps de temps assez grand pour que le circuit interprète mal une instruction ou une donnée. Même si il est difficile pour l'attaquant de maîtriser quelles parties du circuit seront affectées, il est possible d'effectuer des campagnes en faisant varier les paramètres et d'obtenir des résultats convaincants.

2.2.2.2 Injection par variation de l'horloge

L'objet est de faire varier le signal d'horloge externe en introduisant par exemple un cycle plus court que le cycle normal et ainsi mettre le circuit dans un état qui viole les paramètres de **setup** et de **hold** de ses composants. Le but est de conduire un ou plusieurs registres à mémoriser des bits erronés. Cette attaque n'est pas réalisable sur les circuits ne comportant pas d'horloge (circuits asynchrones).

2.2.2.3 Injection par variation de température

Le fondeur définit en général une fenêtre d'utilisation pour les caractéristiques telles que la température, l'horloge ou la tension d'alimentation au-delà de laquelle le fonctionnement du circuit devient imprévisible. L'objet de l'injection par variation de température est d'utiliser cette faille potentielle de fonctionnement.

2.2.2.4 Injection par flash lumière

La sensibilité des circuits électroniques à la lumière, qui se traduit au niveau analogique par l'apparition de photos-porteurs, est aussi néfaste au fonctionnement d'un circuit. Au niveau logique, cela peut se traduire par des « glitches » transitoires ou des basculements de points mémoire. Contrairement au laser, le flash lumière ne permet généralement pas une localisation précise de l'injection de faute et il est probable que cela provoque un disfonctionnement en de nombreux points que l'attaquant ne peut contrôler.

2.2.2.5 Injection par sources de particules

Les sources de particules, parmi lesquelles les accélérateurs de particules, représentent le premier simulateur de fautes générées dans l'espace dans les circuits électroniques. Cependant l'accès à un accélérateur présente plusieurs contraintes liées au coût et aux calendriers trop limités vu le nombre d'accélérateurs de particules existants. Tout cela a poussé à réfléchir à d'autres solutions qui permettent de simuler les mêmes effets à moindre coût. Ainsi l'utilisation du laser s'est imposée, cet outil étant d'une utilisation plus flexible. Des sources radioactives peuvent aussi être employées, comme par exemple des sources de particules alpha, mais elles ne permettent pas le même contrôle spatial et temporel qu'un laser.

2.2.2.6 Injection par laser

Le laser représente un moyen très répandu d'injection de fautes. En effet, il permet de simuler [Hab92] les effets d'injection par accélérateur de particules [Pou00] [Fou90], il permet aussi d'effectuer des cartographies de sensibilité des circuits électroniques aux fautes. En fonction des caractéristiques du

laser et de la technologie du circuit testé, il est possible de viser une zone très précise à attaquer. Cela permet à l'attaquant de mieux contrôler son action, et éventuellement de mieux comprendre le comportement du circuit dans le cadre de la validation de dispositifs de protection (contre-mesures). De plus, cela favorise la reproductibilité de l'attaque, en particulier par rapport au flash lumière.

Le laser est le moyen d'injection choisi au cours de ce travail pour valider les contre-mesures proposées et pour attaquer les circuits. Le chapitre VI revient en détail sur les caractéristiques du laser et de la plateforme expérimental (*setup*) utilisés.

2.3 Les modèles de fautes au niveau logique

2.3.1 Introduction

Il existe de nombreux modèles de fautes représentant les fautes physiques à divers niveaux d'abstraction : électrique, transistor, porte logique, comportemental, fonctionnel, etc. Pour différentes raisons, nous choisissons de représenter les fautes au niveau logique. Nous considérons que quelque soit les phénomènes physiques responsables de la faute, l'effet se traduit par une modification ou un comportement anormal d'un signal logique. Ce niveau d'abstraction présente plusieurs avantages :

- Le niveau logique offre une complexité raisonnable permettant d'analyser et de comprendre l'effet d'une faute à l'échelle du circuit. Il permet d'analyser le comportement d'un circuit au niveau portes et de simuler une injection de faute après synthèse du circuit sans que le temps de simulation ne soit prohibitif.

- Le niveau de modélisation logique est suffisamment fin pour être représentatif de l'effet des phénomènes physiques sur les circuits numériques, tout en étant moins complexe d'utilisation que les niveaux inférieurs (transistors par exemple).

- Le modèle est indépendant de la technologie et des bibliothèques utilisées. On peut ainsi agir au niveau de la structure du circuit pour proposer des contre-mesures relativement tôt dans le flot de conception du circuit.

- La technologie utilisée pour concevoir un circuit a une influence directe sur sa sensibilité aux fautes. En particulier, les technologies fines sont de plus en plus sensibles aux rayonnements naturels. Dans ce contexte, le choix technologique peut avoir une importance sur les propriétés de tolérance aux fautes du circuit. Mais dans un contexte d'attaque

intentionnelle, il est plus raisonnable de penser que quelque soit la technologie utilisée, l'attaquant sera capable d'apporter suffisamment d'énergie en faisant varier ses paramètres d'attaques pour qu'un effet se fasse sentir sur le circuit (au niveau logique, puis fonctionnel). Il est donc préférable de s'abstraire de la technologie et d'agir à un niveau plus élevé.

A l'origine, les modèles présentés dans cette partie cherchent à représenter l'impact naturel des rayonnements et des particules et sont classiquement employés dans le domaine du test. Mais ils peuvent aussi représenter certains types d'attaques. Par exemple, une attaque par laser, très focalisée sur un nœud interne du circuit, est assez similaire à un **SET** (Single Event Transient) ou un **SEU** (Single Event Upset). D'autres types d'attaques peuvent toutefois nécessiter d'adapter le modèle de fautes, en particulier au niveau de la multiplicité des fautes engendrées par une seule attaque. Ces attributs sont abordés dans la partie suivante.

2.3.2 Les fautes de délai (ou fautes de retard)

Certaines fautes ne changent pas le comportement logique mais vont agir uniquement sur la vitesse de calcul, c'est à dire le temps nécessaire pour obtenir une certaine transition sur un signal interne. Ces défauts ont conduit à proposer des modèles de fautes de retard ("delay faults"), qui se décomposent en **fautes de retard de portes** et en **fautes de retard de chemins.**

Les fautes de retard de portes correspondent à un retard d'obtention d'une transition en sortie (porte "lente à monter" ou "lente à descendre"). A l'extrême, une porte "lente à monter" (respectivement, "lente à descendre") peut être assimilée à une porte avec un collage à 0 (respectivement à 1) sur la sortie.

Les fautes de retard de chemins correspondent à un retard similaire, mais sur un chemin donné traversant une succession de portes. Une faute de retard de porte peut donc être vue comme une faute de retard sur un chemin traversant une seule porte. Dans les technologies sub-microniques profondes, les temps de propagation sur les interconnexions deviennent prépondérants. L'utilisation de modèles de fautes de retard de chemins devient donc indispensable.

Une perturbation de l'alimentation peut par exemple engendrer ce type de fautes. Il est par ailleurs probable dans ce cas, que des effets multiples soient engendrés (à moins d'une attaque très focalisée sur un nœud interne, par exemple par laser). Le phénomène inverse (accélération d'une transition) est aussi possible, mais a généralement moins d'impact fonctionnel.

2.3.3 Les fautes transitoires (SET)

Les **SET** (Single Event Transient) correspondent à l'inversion temporaire (« *toggle* ») d'un signal logique à cause d'un transfert de charges parasites dûes à l'impact d'une particule. Ce modèle correspond particulièrement bien aux phénomènes constatés dans le domaine aérospatial ou aux effets du laser.

La largeur d'impulsion obtenue dépend de la technologie, de la topologie des transistors, de la charge électrique collectée et de la capacité du nœud atteint; elle est donc difficile à connaître précisément, ce qui implique la prise en compte d'un intervalle de durées : classiquement de quelques dizaines à quelques centaines de picosecondes dans un contexte d'injection de fautes naturelles, parfois plus pour certaines attaques.

Un SET est injecté dans la logique combinatoire d'un circuit (dans le cas d'un circuit synchrone) et a la possibilité de se propager à travers elle jusqu'à un élément mémoire. En fonction de la topologie de la partie combinatoire, un SET peut très bien se propager vers plusieurs points mémoire. En fonction des paramètres choisis, on pourra aussi ajouter un attribut de multiplicité aux fautes transitoires.

2.3.4 Les basculements mémoire (SEU)

Un **SEU** (Single Event Upset) correspond au basculement d'un point mémoire, sans effet destructeur. Ce type de modèle peut s'étendre aux effets multiples et en particulier aux **MBU** (Multiple-Bit Upsets) correspondant à la commutation simultanée de plusieurs points mémoires sous l'effet d'un impact unique. Dans le domaine de la sécurité, de telles fautes peuvent être obtenues suite à certains types d'attaques non focalisées.

Lors de basculements mémoire, une ou plusieurs informations mémorisées sont directement modifiées. L'état du circuit est donc modifié.

2.4 Paramètres et extensions

Les modèles de fautes précédemment cités doivent être étendus pour être mieux adaptés à représenter une attaque. Les attributs suivants peuvent ainsi être envisagés :

- **Durée** : elle peut s'appliquer aux modèles SET et fautes de délai. Elle est exprimée en nombre de cycles (pour les circuits synchrones) ou en durée effective en fonction du niveau d'abstraction choisi. La

durée effective est dépendante de la nature de la faute. En général, les fautes naturelles sont exprimées en picosecondes. La durée des fautes intentionnelles dépend du moyen d'injection employé, en général de quelques picosecondes jusqu'à plusieurs nanosecondes.

- **Multiplicité :** dans le cadre des fautes naturelles, on considère généralement un modèle de faute unique. Mais dans un cadre plus général, il est utile de définir pour chacun des modèles de fautes un degré de multiplicité spatiale ou temporelle. La multiplicité temporelle correspond au nombre de répétitions ou à la fréquence de répétition d'une faute à un endroit particulier. La multiplicité spatiale correspond au nombre de points affectés simultanément par une injection de faute. En fonction du moyen d'injection de faute employé, la multiplicité peut être élevée, en particulier pour une injection par flash lumière ou par variation de l'horloge, très peu précises. La répartition géographique des fautes sur le circuit dépend également de l'injecteur : avec un laser par exemple, on pourra considérer que les fautes sont localisées dans la même zone géographique, la zone sur laquelle est focalisé le laser. En revanche, une attaque par variation de l'horloge aura un effet beaucoup plus global sur le circuit; la distribution géographique des fautes est alors très différente. On pourra considérer une distribution aléatoire ou une distribution déterministe non localisée sur une zone précise.

- **Instant d'injection** : le moment d'injection peut être défini selon plusieurs critères comme par exemple un temps absolu (pico, nanosecondes …), ou en terme de cycles d'horloge (en fonction du niveau d'abstraction), ou encore par rapport à un événement précis pendant le déroulement de l'expérience. Lors de la modélisation d'attaques sur un circuit asynchrone, il est préférable d'utiliser un événement particulier comme référence pour définir l'instant d'injection.

2.5 Etat de l'art sur le durcissement (synchrone)

2.5.1 Introduction

Cette section résume l'état de l'art des méthodes permettant de détecter ou de tolérer des fautes logiques dans un circuit numérique synchrone. L'objectif est essentiellement d'identifier les principales méthodes existantes et leurs caractéristiques. La plupart de ces méthodes ne peuvent pas être utilisées directement dans le cadre de ce travail, car elles sont spécifiques aux circuits synchrones, et leur utilisation pour durcir des circuits asynchrones contre les fautes serait soit inefficace, soit trop coûteuse et non adaptée. En revanche, elles peuvent être utilisées à titre de comparaison pour évaluer les techniques

spécifiques aux circuits asynchrones proposées dans ce manuscrit, en terme d'efficacité dans la protection contre les fautes et en terme de coût : consommation, surface, vitesse.

Seules les techniques appliquées lors de la conception logique seront prises en compte. En particulier, il ne sera pas fait mention des techniques d'évitement de fautes (blindages, durcissement technologique, …) ni des approches nécessitant des modifications au niveau du schéma électrique des cellules. L'objectif est de pouvoir utiliser des bibliothèques de cellules classiques, dans des technologies différentes. Les techniques de traitement au niveau logiciel ne seront pas non plus abordées.

Au vu des éléments pouvant être affectés par des fautes transitoires SEU/SET, les principaux types de blocs visés sont :

- Les mémoires : cache, RAM.
- Les registres, les bancs de registres, et les points mémoires isolés.
- Les machines à états finis ou FSM (*Finite State Machine*) : logique de séquencement, registre d'état, logique de sortie.
- Les Unités Arithmétique et Logique (UAL) : logique combinatoire.

2.5.2 Terminologie

Si la fonction modifiée par une faute est activée dans le circuit, le résultat obtenu va être faux. Ceci se traduit par une valeur erronée dans un (ou plusieurs) point(s) mémoire du circuit ; cette partie de l'état du circuit qui diffère de l'état attendu est appelée une **erreur**.

Une **défaillance** survient lorsqu'une erreur induit un écart inacceptable par l'utilisateur entre le service spécifié et le service délivré par le circuit. Cela implique en général que l'effet se propage à l'extérieur du circuit, mais toute erreur externe ne correspond pas forcément à une défaillance. Dans le domaine des cartes à puce, une défaillance peut en particulier correspondre à un accès non autorisé à une information confidentielle (par exemple, clé de cryptage).

La **détection de fautes** consiste à déceler et à signaler l'occurrence d'une faute (ou d'une erreur), susceptible d'altérer le service fourni par le circuit. Certaines techniques de durcissement permettent de rendre un **circuit résistant aux fautes**. Les fautes (ou erreurs) sont détectées par le circuit et celui-ci en informe son environnement.

La **tolérance aux fautes** doit assurer, en présence de fautes, la délivrance d'un service conforme à la spécification. Elle peut être obtenue par masquage des fautes ou par une détection suivie d'un recouvrement d'erreur.

On parlera généralement dans ce manuscrit de **circuit résistants/tolérants à n fautes**, ce qui signifie que le circuit a la capacité de détecter (respectivement tolérer) jusqu'à n fautes simultanées, en référence à un modèle de faute donné.

2.5.3 Détection de fautes

2.5.3.1 Redondance matérielle

- Duplication et comparaison (DWC : Duplication With Comparison) [Ang02]

C'est la technique de détection la plus simple comme le montre la figure 2.1. Le coût en surface est de +100%, plus la logique de comparaison. De même pour la consommation (+100%). Le coût en vitesse est faible car le signal de comparaison est obtenu uniquement avec de la logique et en parallèle du signal de sortie. Cette architecture peut s'appliquer à tous les blocs mais c'est l'une des moins optimisées.

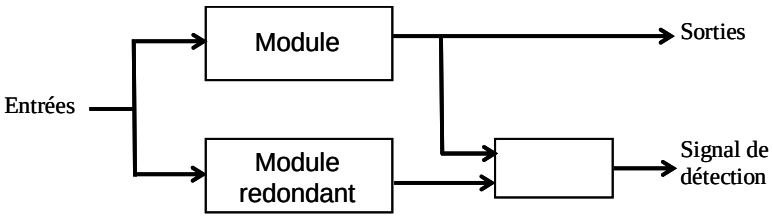


Figure 2.1 Architecture Duplication et Comparaison

- Duplication Avec Redondance Complémentaire (DWCR : Duplication With Complementary Redundancy) [Wen94]

Cette technique est similaire à la méthode DWC mais les signaux d'entrée et de sortie, les signaux de contrôle (distingués des entrées de données) et les données internes sont de polarités opposées dans les deux modules. Le surcoût en surface et en puissance consommée est de l'ordre de +100% et il faut ajouter la logique de comparaison. Le surcoût en vitesse est faible. Néanmoins cette méthode augmente la complexité de la conception par rapport à une duplication simple. La réalisation en double rail (DRC : Double Rail Checker), dans laquelle les deux sorties sont inversées si il n'y a pas de fautes, peut être vue comme un cas plus simple de redondance complémentaire.

Le détecteur double rail est le plus souvent utilisé comme bloc de comparaison (contrôleur). Un exemple d'utilisation se trouve dans [Pap01].

2.5.3.2 Redondance temporelle

- Redondance temporelle simple

L'opération est effectuée deux fois de suite et les résultats sont comparés. Cette méthode est simple, et bien adaptée aux fautes temporaires comme les SEU. Cette architecture entraîne un surcoût en vitesse très important car le temps de calcul est plus que doublé, le temps de cycle augmentant légèrement à cause de la comparaison et deux cycles étant nécessaires au lieu d'un. Le surcoût en surface est limité à la mémorisation du premier résultat et à la logique de comparaison, la puissance consommée augmente peu (mais l'énergie augmente notablement).

- Contrôle Concurrent Basé sur la Redondance Temporelle [Ang00]

Cette technique vise la détection de SET. Le circuit de détection se compose de deux bascules et d'un comparateur par sortie du circuit combinatoire (figure 2.2). Ce circuit permet de détecter une faute transitoire affectant le circuit combinatoire. S'il n'y a pas de faute, le résultat est directement disponible sur la bascule de sortie. Le retard δ est fonction du temps de set-up des bascules (Dsetup) et du temps maximum toléré pour le SET (Dtr) : δ = Dsetup + Dtr. En général Dtr vaut quelques picosecondes.

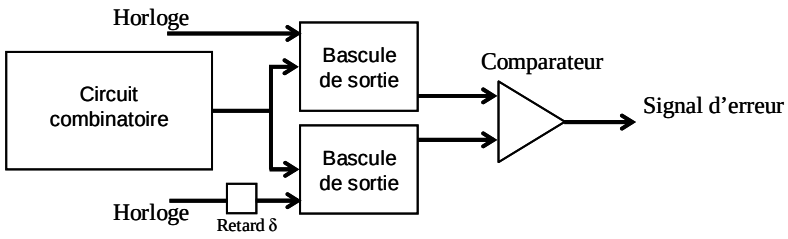


Figure 2.2 Architecture de contrôle concurrent basé sur la redondance temporelle

Le surcoût en surface est dû aux bascules supplémentaires, à la logique de comparaison et aux éléments rajoutés pour retarder le signal d'horloge. Les essais ont été effectués sur des additionneurs et des multiplieurs; pour les multiplieurs le surcoût en surface est compris entre +1,4% et +10%, pour les additionneurs le surcoût en surface est compris entre +22,6% et +70,2%.

La méthode de contrôle concurrent basé sur la redondance temporelle s'applique bien aux blocs se trouvant dans le chemin critique. Le surcoût en surface peut être assez faible (surtout pour les multiplieurs), les surcoûts en vitesse et en consommation sont faibles.

- Recalcul avec Duplication et Comparaison (REDWC : Recomputing with Duplication and Comparison) [Wen94]

REDWC est une méthode de détection de fautes par association de redondance temporelle et de duplication. Elle s'applique aux additionneurs et aux multiplieurs, le principe est de diviser les opérandes sur n bits en opérandes sur n/2 bits. Prenons l'exemple de l'addition :

- Les demi-mots de poids faibles sont additionnés simultanément par deux additionneurs n/2 bits, les résultats sont comparés et l'un deux est mémorisé.

- La retenue est réinjectée dans les additionneurs et on additionne les demi-mots de poids forts. Les résultats sont comparés et s'il n'y pas de faute, le résultat final est obtenu par concaténation du résultat obtenu pour les poids faibles et les poids forts.

Cette technique présente un bon compromis entre surcoût en surface (+75%), en vitesse (+31% pour le temps de cycle d'un multiplieur) et en puissance consommée.

2.5.3.3 Redondance d'information

La redondance d'information consiste à utiliser un code détecteur ou correcteur d'erreur. Quelques exemples de codes détecteurs sont cités ci-dessous.

- Codage par parité

La détection par génération de bit de parité est basée sur le calcul du nombre des bits à 1 d'un mot de données et permet la détection de toutes les erreurs de multiplicité impaire. La génération et le contrôle du bit de parité se fait à l'aide de portes XOR.

Ce mode de détection est simple et entraîne un coût temporel limité. Dans le cas des mémoires, le bit de parité est généré quand les données sont écrites dans la mémoire et il est contrôlé au moment de la lecture. Le code de parité est un code systématique.

Le codage par parité est très utilisé, voici quelques exemples d'application :

- Codage par parité à deux dimensions (lignes et colonnes) pour les mémoires et bancs de registres [Var00]. L'inconvénient est que deux fautes sur la même ligne entraînent la corruption de toute la mémoire.

- Pour la détection d'erreurs multiples et dans le cas de bancs de registres, on peut implanter la technique de la parité croisée (Cross-Parity), développée en détails dans [Pfl02]. Elle correspond au codage par parité des lignes, colonnes et diagonales qui constituent une mémoire et à la vérification par analyse des trois parités obtenues.

- Prédiction du code de parité notamment pour les applications à faible ou moyenne consommation [Pap01].

- Codage "Double Rail"

Le codage "Double Rail" consiste à dupliquer et complémenter chaque mot de données et permet de détecter les erreurs multiples. Le surcoût en surface est très important (+100%) car il y a autant de bits de contrôle que de bits de données.

- Codage m parmi n (m/n)

Le code m/n est un code non-séparable composé de mots de n bits comportant m bits à 1. Exemple le code 2/4 est composé des mots {1100,1010,1001,0101,0011,0110}. Ce type de codage peut-être utilisé dans les circuits de contrôle mais il entraîne une redondance très importante et des circuits de codage/décodage très compliqués.

- Codage résidu

Le code résidu (arithmétique) correspond à la concaténation de n bits de données et de m bits de contrôle représentant le résidu (modulo r) des données, ce code est noté (n,m). La base r du résidu doit être différente de 2^m, en effet si $r = 2^m$ les fautes simples affectant les bits de poids $\geq 2^m$ ne sont pas détectables. Pour la détection de fautes simples on prend r = 3, ce qui fait apparaître deux bits de redondance pour le codage de l'information.

Les surcoûts en surface et en consommation sont importants pour les circuits séquentiels et les circuits de mémorisation. Ce code est donc surtout préconisé pour les circuits arithmétiques traitant des mots de longueur importante.

2.5.3.4 Architectures pour séquenceurs (machines à états finis)

Des techniques de protection particulières peuvent être définies pour certains blocs, notamment les machines à états. Un exemple est la vérification du flot de contrôle par analyse de la signature (CFC : Control Flow Checking) qui donne lieu à deux réalisations possibles, avec ou sans ajustement, comme développées dans [Roc96]. La signature est calculée à partir du code de l'état futur et de la signature courante. Le surcoût en surface pour cette architecture dépend fortement de la structure de la machine à états considérée. Il peut être très faible, mais peut aussi atteindre des valeurs supérieures à +100%. Le surcoût en temps (chemin critique) est également très variable [Lev02].

2.5.4 Recouvrement

Les différentes techniques ci-dessus permettent la détection de fautes dans un circuit mais pour les applications critiques la détection n'est pas suffisante puisque le résultat en sortie reste faux. Pour effectuer un recouvrement, on peut simplement **répéter** la même fonction lorsqu'une erreur est détectée, sachant que la faute est supposée transitoire. Cette technique peut s'appliquer à tous les blocs d'un circuit. Selon le type de bloc, il est nécessaire de définir soigneusement les points de reprise possibles.

Dans le cas particulier des microprocesseurs, le recouvrement peut s'effectuer à plusieurs niveaux, selon que l'on considère l'exécution d'une macro-instruction (Branch ou Add par exemples), ou l'exécution des micro opérations qui sont réalisées par le processeur (lecture de la mémoire, chargement de registres, etc.). Ainsi on exécute lorsque cela est possible un "**micro-rollback**" après détection d'une faute lors de l'exécution d'une micro opération, au niveau d'un étage d'un processeur (pipeline ou non) [Gal02], [Tre89]. Mais il est parfois nécessaire d'effectuer un **"stage rollback"** ou un "**pipeline rollback**" (qui coûtent plusieurs cycles d'horloge), lorsque les données nécessaires à l'exécution de la micro opération ne sont plus disponibles. Le processeur doit alors ré-exécuter un certain nombre d'opérations effectuées auparavant [Gal02].

2.5.5 Tolérance aux fautes

2.5.5.1 Redondance matérielle

La méthode **TMR** (**Triple Modular Redundancy**) correspond à la triplication d'un module avec un circuit de vote majoritaire en sortie comme illustré figure 2.3. Lorsque plus d'un des modules n'est plus fonctionnel, la tolérance n'est plus garantie et la sortie peut être erronée. Le coût en surface et en consommation est élevé à cause du triplement du nombre de modules et du rajout de la logique de vote mais le surcoût en vitesse est faible car le voteur ne comprend que peu de logique combinatoire.

Cette méthode peut-être appliquée à tout bloc, globalement ou au niveau le plus bas, c'est-à-dire pour chaque bascule, ou bien encore, comme dans le cas des machines à états, pour une sous partie comme la logique de séquencement [Roc96].

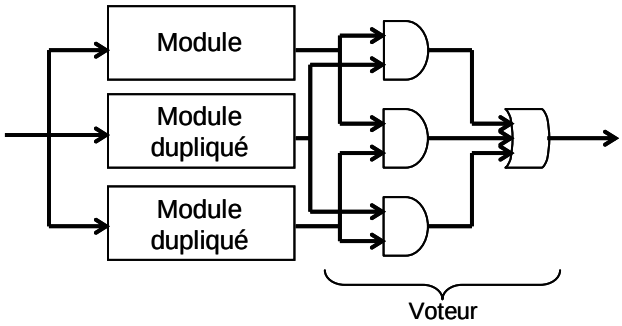


Figure 2.3 Architecture TMR

On peut étendre la méthode TMR à la méthode NMR qui utilise N-1 modules redondants. L'avantage est que la tolérance est toujours effective si un ou plusieurs modules (jusqu'à N/2 - 1) tombent en panne; l'inconvénient est un surcoût en surface encore plus important.

2.5.5.2 Redondance temporelle

La méthode est similaire à la technique de redondance temporelle REDWC (duplication et comparaison) présentée précédemment, mais appliquée à la tolérance donc avec triplication et vote [Wen94]. Le surcoût en surface est de l'ordre de +100% et le surcoût en temps (temps de cycle) est compris entre +76% (additionneur) et +131% (multiplieur).

2.5.5.3 Redondance d'information

Le **codage de Hamming** est l'un des codages les plus utilisés pour la correction d'erreurs dans les mémoires. Il rajoute des bits de contrôle à certaines positions dans un bloc de bits d'information. Il est tel que $2^m \geq n+m$, où n est le nombre total de bits (bits de données et bits de correction) et m est le nombre de bits de correction. Le résultat est un code séparable qui permet la correction d'erreur sur un bit (code SEC : Single Error Correction).

Pour calculer les bits de correction, on utilise une matrice de parité H. Ce codage de Hamming permet de corriger une erreur. Le décodage se fait en multipliant la matrice H par le mot code (écrit comme un vecteur colonne). Le résultat donne finalement la position du bit erroné.

On peut obtenir un code SEC/DED (Single Error Correction / Double Error Detection) en rajoutant un bit de parité, ce qui revient à rajouter une ligne de 1 dans la matrice H. L'implantation du codage de Hamming, comme présentée sur la figure 2.4 dans le cas particulier d'une machine à états (registre d'état), nécessite un bloc de codage, un bloc de décodage et de correction de l'erreur.

Les résultats obtenus dans le cadre d'un durcissement de micro-processeur 8051 [Lim00] montrent que le codage de Hamming est le plus efficace pour la protection de la mémoire interne (faible nombre de blocs logiques de codage/décodage) et le moins efficace pour l'UAL (grand nombre de blocs logiques de codage/décodage). Ses avantages sont un faible coût en surface (moins de +50%) et la correction d'erreur (pas seulement la détection).

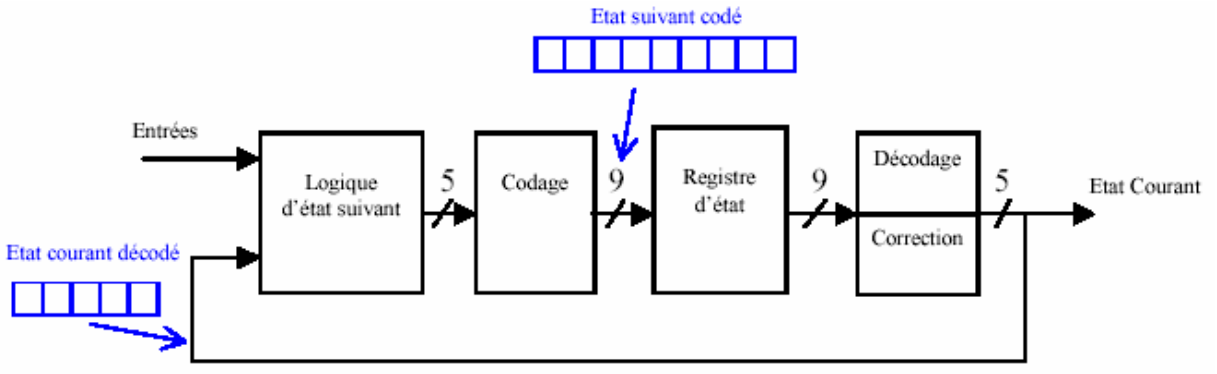


Figure 2.4 Exemple de réalisation d'une machine à états avec code de Hamming (9, 5)

2.6 Conclusion

L'injection de fautes naturelles et l'injection de fautes intentionnelles ont certains points communs bien que leur problématique soit différente. Les modèles de fautes utilisés pour modéliser une attaque sont souvent les mêmes que les modèles utilisés dans le test en ligne, mais avec des paramètres et des attributs plus spécifiques.

L'approche doit être différente sur certains aspects : pour les fautes naturelles, on s'appuiera sur un modèle de faute unique, et on parlera en termes de probabilité d'apparition, de couverture, et de fiabilité. Pour les fautes intentionnelles, on ne posera pas de limites fixes sur les possibilités de l'attaquant. En effet, celui-ci cherche à obtenir un résultat fauté exploitable pour son attaque et les moyens dont il dispose dépassent le cadre du modèle de faute unique et probabiliste du test en ligne.

La plupart des techniques de durcissement décrites dans cette section sont issues des travaux de recherche sur la protection des circuits synchrones contre les fautes naturelles. Les travaux de recherches sur la sécurité des circuits synchrones, plus récents, se basent aujourd'hui sur les mêmes techniques [Ber03] [Bre04].

En ce qui concerne le comportement des circuits asynchrones en présence de fautes (naturelles ou intentionnelles), peu de travaux ont été effectués. Le chapitre suivant présente les caractéristiques des circuits asynchrones et dresse un état de l'art des techniques de durcissement spécifiques à ces circuits. Les modèles de fautes qui seront présentés sont dérivés de ceux décrits dans ce chapitre.

CHAPITRE III

LES CIRCUITS ASYNCHRONES : CARACTERISTIQUES ET SÉCURITÉ

3.1 Présentation des circuits quasi insensibles aux délais

3.1.1 Introduction

Ce chapitre est consacré à la présentation de la technologie asynchrone, et à l'évaluation de ses propriétés pour la conception de 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 octroie une faible émission électromagnétique et de faibles pics de courant. L'utilisation souvent naturelle d'un codage des données et d'un protocole de communication peut permettre une consommation faible, répartie, et indépendante des données. L'étude a pour objectif d'évaluer les propriétés intrinsèques des circuits asynchrones à résister aux attaques par fautes, et de définir des techniques permettant d'améliorer cette résistance.

Nous introduisons dans un premier temps la technologie asynchrone et plus particulièrement les circuits quasi insensibles aux délais (QDI, Quasi Delay Insensitive) qui, parmi les différentes classes de circuits asynchrones existantes, semblent proposer les meilleures propriétés pour la conception de circuits sécurisés. Nous évaluons les propriétés de ces circuits face aux différentes attaques matérielles, puis nous dressons un état de l'art des études focalisées sur les attaques par injection de fautes. Enfin, parmi les différents modèles de fautes présentés dans le chapitre précédent, certains sont adaptés ou redéfinis pour être mieux intégrés à l'analyse de sensibilité des circuits asynchrones.

3.1.2 Les différentes classes de circuits asynchrones

Lorsque l'on parle de circuits synchrones, on fait référence à des circuits contrôlés et séquencés par un signal périodique et global, l'horloge. Les circuits asynchrones représentent un type de circuits dont le contrôle et le séquencement sont assurés par d'autres méthodes.

Il existe différentes classes de circuits asynchrones en fonction des hypothèses faites sur les délais et des relations temporelles faites entre les événements (données ou contrôle). La figure 3.1 présente la terminologie habituellement utilisée pour nommer les circuits asynchrones.

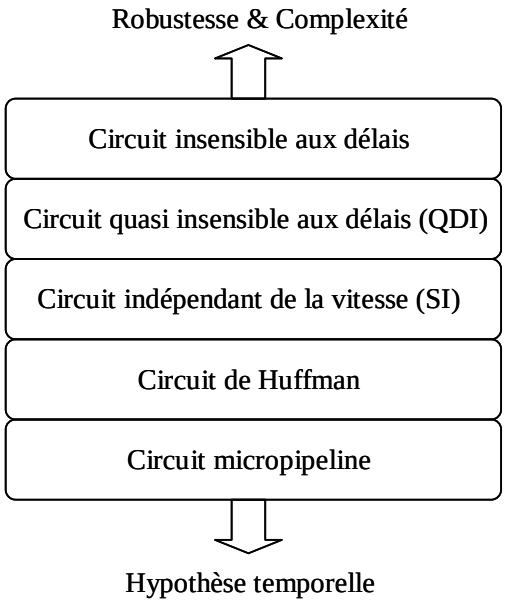


Figure 3.1 Différentes classes de circuits asynchrones

Plus le fonctionnement du circuit respecte la notion d'asynchronisme, plus le circuit est robuste (en terme d'hypothèses sur les délais) et complexe.

Les **circuits de Huffman** utilisent un modèle de délais identique à celui des circuits synchrones. Ils supposent que tous les éléments et les connexions ont un délai borné. La technique de **micropipeline** a été introduite par Sutherland [Suth89]. Les circuits de cette classe sont composés de parties contrôles insensibles aux délais qui commandent des chemins de données conçus en utilisant un modèle de délai borné similaire aux circuits synchrones.

La classe de **circuits indépendants de la vitesse** (Speed Independant - SI) considère que tous les fils ont un délai équivalent, ce qui représente une hypothèse forte par rapport aux **circuits quasi insensibles aux délais** pour lesquels cette hypothèse est nécessaire uniquement sur certains fils. Les circuits quasi insensibles aux délais sont décrits plus en détail dans le paragraphe suivant. Enfin, les **circuits insensibles aux délais** nécessitent l'utilisation de cellules complexes très coûteuses en raison des contraintes extrêmement fortes. Dans la pratique, on revient souvent à des hypothèses équivalentes à celles des circuits quasi insensibles aux délais.

3.1.3 Les circuits quasi insensibles aux délais (QDI)

Le fonctionnement des circuits quasi insensibles aux délais (QDI) [Ren00] est correct quelques soient les temps de propagation dans les fils et les éléments logiques qui composent le circuit. Le modèle de délai dans ce type de circuit est donc non borné. Seules quelques fourches dites isochrones font l'objet d'une hypothèse temporelle. Une fourche est isochrone si les temps de propagation dans les branches de la fourche sont égaux. En pratique, l'hypothèse temporelle de fourche isochrone est assez faible et elle est facilement remplie par une conception soignée, en particulier au niveau du routage et des seuils de commutation.

Un circuit QDI répondra toujours correctement à une sollicitation externe indépendamment du temps nécessaire au calcul. Ceci impose donc au récepteur d'un signal de toujours informer l'émetteur que l'information a été reçue. L'émetteur doit attendre l'autorisation du récepteur pour émettre une nouvelle donnée. Le protocole utilisé pour accomplir cette communication est décrit dans le paragraphe suivant. De plus, cela nécessite un codage particulier des données.

3.1.3.1 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"). Le fonctionnement du protocole 4 phases est décrit dans la figure 3.2. Il faut noter que dans les deux cas tout changement d'un signal par l'émetteur est acquitté par le changement d'un signal du récepteur et vice-versa. C'est ce qui permet d'assurer l'insensibilité aux délais. Il existe bien sûr de nombreuses variantes de ces protocoles.

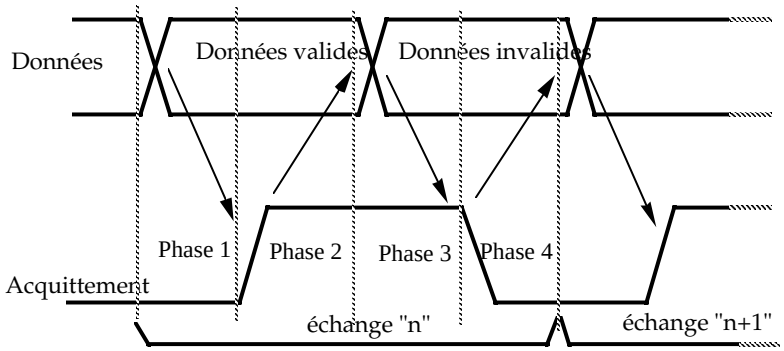


Figure 3.2 Principe du protocole quatre phases

Nous nous intéressons dans ce travail uniquement au protocole 4 phases, qui est le plus couramment utilisé en raison de sa simplicité d'implantation. Son fonctionnement est le suivant :

- **Phase 1** : le récepteur détecte la présence de nouvelles données, effectue le traitement et génère le signal d'acquittement.

- **Phase 2** : l'émetteur détecte le signal d'acquittement et émet des données invalides (retour à zéro).

- **Phase 3** : le récepteur détecte le passage des données dans l'état invalide et place le signal d'acquittement dans l'état initial (retour à zéro).

- **Phase 4** : l'émetteur détecte le retour à zéro de l'acquittement. Il est alors prêt à émettre de nouvelles données.

3.1.3.2 Codage des données

Afin de détecter la présence d'une donnée et de générer un signal qui indique la fin d'un traitement, il est indispensable d'adopter un codage particulier pour les données. Il est en effet impossible d'utiliser un seul fil par bit de donnée. Un fil par bit ne permet pas de détecter que la nouvelle donnée prend un état identique à la précédente. Les protocoles utilisent communément un codage bifilaire ou double rail 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.

Le codage 3 états est le mieux adapté au protocole de communication 4 phases. Un fil prend la valeur 1 pour coder une donnée à 1 et l'autre fil prend la valeur 1 pour coder la donnée 0. L'état "11" est interdit alors que l'état "00" représente l'invalidité d'une donnée (utile pour le protocole 4 phases). Ainsi, passer d'une valeur valide à une autre implique de toujours passer par l'état invalide (figure 3.3). Ce codage garantit que le passage d'un état à un autre se fait toujours par changement de l'état d'un seul bit sur les deux, passage détectable sans aléa par une réalisation matérielle insensible aux délais. Le signal de fin de calcul d'un opérateur peut facilement être généré en détectant qu'un des bits de sortie est passé à 1.

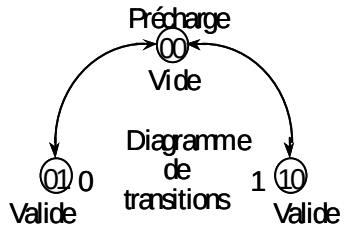


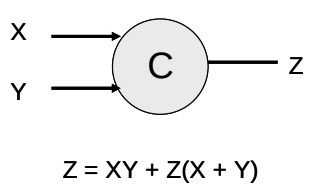
Figure 3.3 Codage trois états

Le codage double rail est le plus couramment utilisé dans la conception de ces circuits. Cependant on peut tout aussi bien utiliser n'importe quel autre codage de type *1 parmi n* tel qu'il a été défini au chapitre précédent, ou un codage de type *p parmi n* avec 1<p<n, ou dans le cas général les codes non ordonnés ou insensibles aux délais.

3.1.3.3 La porte de Muller

La porte de Muller est une fonction couramment utilisée dans la conception de circuits asynchrones et en particulier des circuits QDI. Elle fera l'objet d'une analyse poussée dans la suite de ce travail. Une porte de Muller ou C-élément est une fonction qui copie le niveau logique des entrées sur la sortie lorsque ceux-ci sont égaux. Bien qu'il soit possible de réaliser cette fonction avec des cellules de bibliothèques synchrones standard, il est préférable d'utiliser des cellules dédiées pour obtenir de meilleures performances en terme de vitesse, consommation, taille [Fol05] et sécurité [Mau03].

La figure 3.4 présente la table de vérité, le symbole et l'équation d'une porte de Muller à deux entrées. La porte réalise un rendez-vous sur les transitions. Une transition est propagée en sortie si et seulement si une transition a eu lieu sur chacune des entrées. On peut définir cette fonction pour un nombre d'entrées quelconque.



X	Y	Z
0	0	0
0	1	Z^{-1}
1	0	Z^{-1}
1	1	1

Figure 3.4 Porte de Muller à deux entrées

3.1.3.4 Structure des circuits QDI

Un circuit asynchrone est composé de modules de complexité variable qui communiquent entre eux par l'intermédiaire du protocole de communication de type requête-acquittement (figure 3.5). Cette localité du contrôle permet une très grande modularité/réutilisation. Il est en effet très facile de construire une fonction ou un système complexe en connectant des modules préexistants par simple assemblage. On s'affranchit donc de toutes les contraintes d'assemblage liées à l'horloge et des problèmes de placement routage habituellement rencontrés lors de la conception de circuits synchrones.

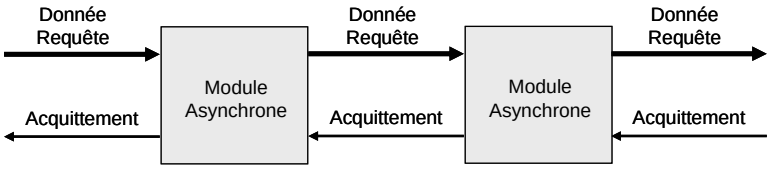


Figure 3.5 Communication entre modules asynchrones

Un bloc synchrone est habituellement composé d'une partie combinatoire qui traite les données et d'une partie mémoire (les registres) qui mémorisent les données à chaque cycle. Par analogie, il est possible de définir dans un module asynchrone (appelé un étage) une **partie calculatoire** et une **partie mémoire**, telles que présentées sur la figure 3.6. Ces définitions ne relèvent pas de critères habituellement utilisés dans la littérature asynchrone. Comme nous le verrons par la suite, elles sont nécessaires dans le cadre de ce travail pour distinguer les modèles de fautes et leurs effets sur le circuit.

La **partie calculatoire** correspond à l'implantation des fonctions du bloc, similaire à la partie combinatoire d'un circuit synchrone. Nous distinguons les termes « calculatoire » et « combinatoire » car il est fréquent que la partie calculatoire d'un bloc asynchrone contienne des éléments mémoire (les portes de Muller) en plus des portes combinatoires utilisées pour l'implantation d'une fonction. Cette partie n'est donc pas « combinatoire » à proprement parler.

Les éléments mémoire implantés dans cette partie sont utilisés pour réaliser une fonction (un rendez-vous entre deux signaux) et non pour mémoriser le résultat d'un bloc calculatoire dans le circuit. Leur propriété mémorisante n'est nécessaire que pour garantir les propriétés QDI du bloc. Ils ne participent donc pas au calcul de **l'état global** du circuit, tel que nous le définissons un peu plus loin dans cette section.

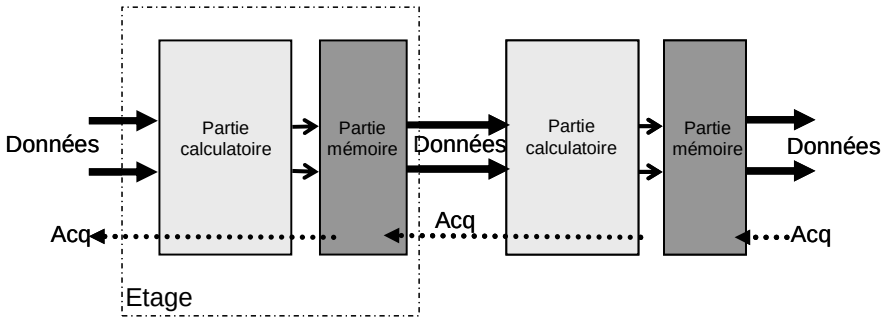


Figure 3.6 Structure des modules asynchrones

La **partie mémoire** correspond à l'implantation des éléments qui mémorisent les données en sortie de la partie calculatoire. Sa fonction est donc similaire aux registres des circuits synchrones, mais elle est en plus chargée d'assurer l'implantation du protocole de communication entre ce bloc et le bloc suivant. **L'état global** d'un circuit asynchrone est défini comme étant l'état de l'ensemble des portes de Muller implantées dans les parties mémoires du circuit.

La figure 3.7 présente un exemple de module implantant un XOR (opérateur OU exclusif) entre deux données double rail A(A0, A1) et B(B0, B1). Le résultat est alors mémorisé en sortie dans un **Half-Buffer** WCHB (Weak Charge Half Buffer) [Ren00]. Un **Half-Buffer** est une structure permettant de mémoriser les données soit dans leurs états valides soit dans leurs états invalides et d'assurer la communication entre les blocs. On note que pour que les quatre phases du protocole de communication soient assurées, il est nécessaire d'implanter un **Full-Buffer** (ou deux Half-Buffers), permettant de mémoriser les états valides et invalides des données.

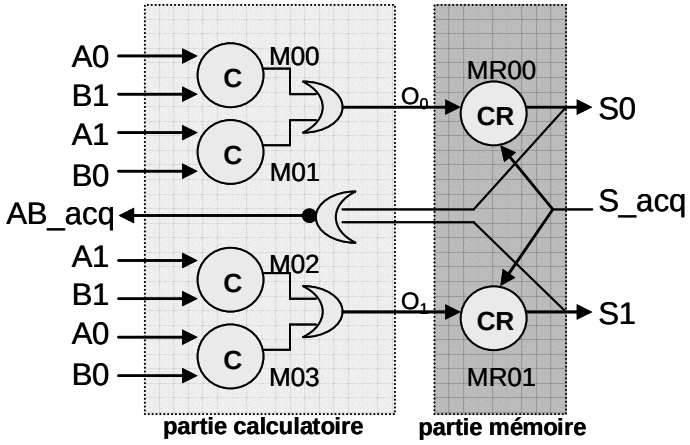


Figure 3.7 Réalisation de la fonction logique XOR en double rail

L'état global de ce circuit est défini comme étant l'état de la sortie des deux portes de Muller présentes dans la partie mémoire (MR00 et MR01). Ces portes de Muller, notées CR sur la figure, nécessitent une entrée *Reset* (R) permettant de définir l'état du circuit à l'initialisation. Les portes de Muller M00, M01, M02 et M03 présentes dans la partie calculatoire sont utilisées pour réaliser la fonction XOR. Elles ne nécessitent pas d'initialisation.

3.1.4 Synthèse des circuits asynchrones

La conception des circuits asynchrones reste encore le domaine de spécialistes utilisant des outils semi-automatiques. Sa diffusion dans le secteur industriel ne sera possible que si des outils avec un niveau d'automatisation égal aux outils disponibles actuellement pour les circuits synchrones sont développés et que des ingénieurs soient formés à leur utilisation. Pour cela le groupe CIS du laboratoire TIMA développe l'outil TAST qui permet de décrire et de synthétiser des circuits asynchrones.

TAST (Tima Asynchronous Synthesis Tool) est un environnement de conception composé principalement d'un compilateur/synthétiseur capable de cibler différents types de circuits asynchrones décrits dans un langage de haut niveau proche de CSP (Communicating Sequential Process). Ce langage, appelé CHP (Communicating Hardware Processes), inclut la définition des protocoles de communication, du codage des données, des choix déterministes et indéterministes, la gestion de la hiérarchie et la génération de traces. Une forme synthétisable du langage, appelée DTL (Data Transfer Level) a été définie pour permettre la génération automatique des circuits QDI et micropipeline.

La figure 3.8 présente le flot de conception supporté par l'environnement TAST.

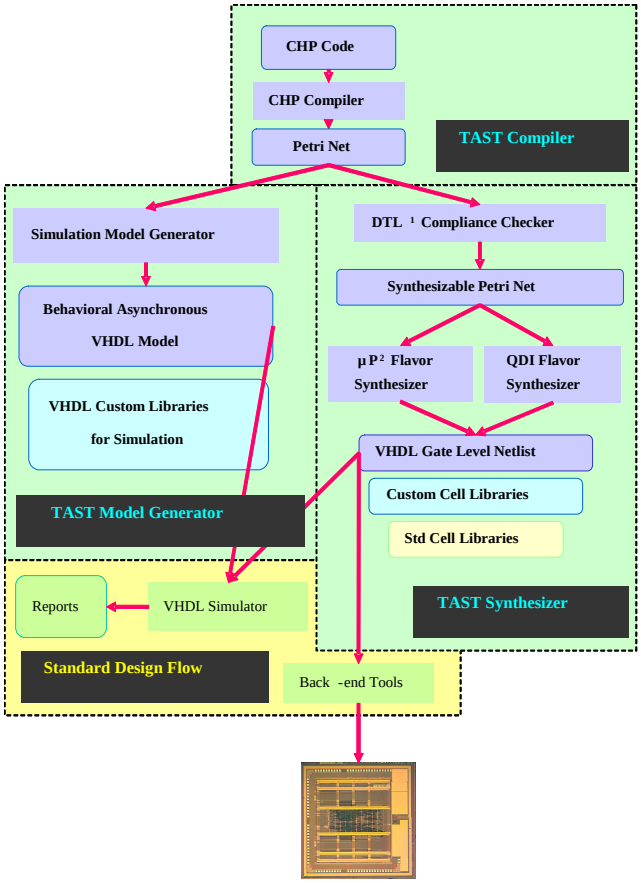


Figure 3.8 Flot de conception de l'environnement TAST

L'environnement TAST a pour but de permettre l'automatisation de la synthèse des circuits, mais aussi l'automatisation de certaines optimisations, en fonction de nombreux paramètres :

- Réduction de la surface [Mau03] [Fol05].
- Conception de systèmes à faible consommation [Slim04] [Essa02].
- Réduction des émissions électromagnétiques [Pany04].
- Conception de systèmes dédiés à la sécurité [Boue04a] [Ren04].

Les méthodes d'analyse de sensibilité aux fautes ainsi que les méthodes de durcissement contre les fautes présentées dans ce manuscrit sont destinées, à court terme, à être intégrées à l'environnement de synthèse pour permettre leur automatisation.

3.2 Les circuits asynchrones et la sécurité

3.2.1 État de l'art sur la résistance des circuits asynchrones aux attaques par canaux cachés

Tous les circuits asynchrones ne sont pas égaux faces aux attaques par canaux cachés. En particulier, les circuits de type micropipeline présentent les mêmes caractéristiques que les circuits synchrones face aux attaques en puissance de type DPA. Dans le cas des circuits synchrones, l'analyse des courbes de courant est complexifiée par l'effet de la consommation des arbres d'horloge qui introduisent des signaux parasites. Dans le cas des circuits micropipeline, l'absence d'horloge globale rend l'analyse des courbes encore plus facile car elle reflète directement l'activité des blocs du composant.

L'analyse de la résistance des circuits QDI face aux attaques en puissance a été développée dans le cadre d'une thèse menée dans le groupe CIS [Boue05a]. Il y est montré que les circuits QDI offrent une meilleure résistance naturelle à ces attaques, en raison de plusieurs facteurs :

- Le codage des données de type 1 parmi n permet de garder un poids de Hamming constant quelles que soient les données manipulées. On supprime donc les corrélations entre les données et le courant consommé liées à ce facteur.

- 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.

- 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.

- Le contrôle local permet de mieux répartir l'activité du circuit dans le temps et de rendre plus difficile la synchronisation sur un événement particulier.

Cependant, il reste possible de réaliser une attaque en puissance sur les circuits QDI [Boue04b]. En effet, si il est possible d'équilibrer parfaitement les circuits et de rendre la consommation indépendante des données au niveau logique, la phase de placement routage apporte de nombreux déséquilibres entre les chemins, sources de fuites pour réaliser les attaques. Ce problème a été formalisé dans [Boue05a] et [Boue05b] et des solutions sont proposées pour améliorer la résistance des circuits [Boue05c] [Boue06]. Ces techniques, basées sur des *jitters* temporels ou sur un échange de chemin de calcul (*path swapping),*

permettent d'exploiter les propriétés des circuits QDI et de considérablement augmenter leur résistance avec un coût en surface, vitesse et consommation très faible.

Il existe très peu de travaux mettant en évidence les caractéristiques des circuits asynchrones face aux attaques par analyse des émissions électromagnétiques. Il n'est pas possible aujourd'hui de conclure sur les capacités de résistance des circuits asynchrones face à ce type d'attaque.

3.2.2 État de l'art sur la résistance des circuits asynchrones aux attaques par fautes et leur durcissement

Nous proposons un résumé des travaux menés sur les circuits asynchrones vis-à-vis de l'injection de fautes. Certains travaux analysent le comportement et la résistance intrinsèque des circuits asynchrones, et d'autres proposent des techniques de durcissement pour améliorer leur résistance ou leur tolérance. Nous proposons une synthèse structurée suivant le type de circuits asynchrones considéré. Cette synthèse montre que très peu de travaux ont été menés sur l'injection de fautes dans les circuits asynchrones et que presque tous les articles omettent de précisément définir le type de faute considéré.

3.2.2.1 Généralités : comportement des circuits asynchrones en fonction du moyen d'injection de fautes

Les circuits asynchrones insensibles aux délais sont par construction très robustes vis-à-vis des fluctuations de l'alimentation. Ainsi, la génération d'aléas sur les lignes d'alimentation crée très difficilement des fautes dans le circuit. Cependant si ces aléas sont créés de façon intentionnelle (par un attaquant) avec une grande amplitude, il sera possible de provoquer des fautes de type SEU ou MBU.

Les circuits asynchrones ne peuvent être attaqués par l'horloge, comme c'est le cas pour les circuits synchrones qui sont sensibles à des « glitch » appliqués à l'horloge. Ils peuvent cependant faire l'objet d'attaques par perturbation des entrées primaires. En effet, les circuits asynchrones sont sensibles à des transitions sur les signaux d'entrée. Cette sensibilité dépend du style de logique asynchrone utilisée pour leur conception.

3.2.2.2 Circuits de Huffman

Les circuits de Huffman en mode fondamental sont des circuits asynchrones dont les entrées ne peuvent changer que lorsque le circuit est dans un état stable [Ren00].

L'article [Saw74] propose une méthode d'assignation d'états pour des machines séquentielles asynchrones de type Huffman. Les types de fautes considérés initialement sont les collages à 0 et à 1.

Cependant, le système de détection de fautes présenté étant continu, on peut élargir le type de fautes considérées à une classe plus large, comme celle des fautes temporaires (fautes transitoires).

Cette approche n'est pas basée sur la redondance matérielle mais sur une méthode systématique d'assignation des états permettant la détection de toutes les fautes simples, aussi bien les fautes engendrant une erreur sur la sortie que sur les variables internes de la machine à états.

Il est montré que le nombre de portes logiques nécessaires pour implanter ce système de détection d'erreur dans une machine à états est inférieur ou égal au double du nombre de portes utilisées avec un codage minimal des états, ce qui peut être un avantage par rapport à des méthodes basées sur la duplication de matériel. De plus, les fautes sont détectées en temps continu. Cela permet de considérer une classe de fautes plus larges que les collages.

3.2.2.3 Circuits Micropipeline

L'article [Ver02] présente une méthode de protection des circuits asynchrones micropipeline basée sur la duplication de matériel. Les méthodes de détection d'erreurs basées sur la duplication sont couramment utilisées dans les circuits synchrones, comme cela a été présenté dans le chapitre précédent. Les blocs sont physiquement dupliqués et un comparateur est placé à leurs sorties pour détecter une erreur sur un des blocs.

Il est montré que l'adaptation de cette technique aux circuits asynchrones de type Micropipeline pose des difficultés, notamment au niveau de la synchronisation des entrées du comparateur. Les auteurs proposent un type de comparateur permettant la détection d'erreurs dans des cas limités : ceux pour lesquels la différence de temps de calcul entre le bloc original et le bloc dupliqué reste faible.

De façon générale, les circuits asynchrones de type micropipeline ont des caractéristiques semblables à celles des circuits synchrones vis-à-vis de l'injection de faute.

3.2.2.4 Circuits quasi insensibles aux délais

La robustesse des circuits QDI a fait l'objet de plusieurs travaux de recherche. Les canaux de communication confèrent à ces circuits une propriété très intéressante vis-à-vis de la sécurité : la capacité à bloquer le système. En effet, une faute peut conduire à perturber le protocole de communication d'un canal et provoquer un blocage. Contrairement aux circuits synchrones, le séquencement des circuits asynchrones peut être exploité pour se protéger contre l'injection de fautes [Pies95].

L'idée proposée dans [Moor02] est de concevoir des circuits asynchrones en utilisant un protocole quatre phases et un codage trois états pour les données. Le codage double rail pour un bit de donnée est enrichi du code « alarme » tel qu'illustré sur la Figure 3.9. La détection d'un code erroné par une alarme n'est pas une contre mesure spécifique aux circuits asynchrones. En effet, elle pourrait très bien être implantée dans un circuit synchrone utilisant un codage multi-rail. Cependant, cette contre-mesure apparaît comme « naturelle » dans le cas d'un circuit QDI.

D0	D1	Valeur
0	0	Invalide
1	0	Zéro
0	1	Un
1	1	Alarme

Figure 3.9 Codage double rail avec code d'alarme

D'après [Moor02], l'injection d'une faute unique dans le circuit conduit à observer l'un des trois comportements suivants :

- Une donnée peut disparaître, ce qui va forcément causer un blocage du circuit.
- Une donnée peut apparaître, ce qui va forcément causer un blocage du circuit.
- Le code d'alarme peut apparaître ce qui ne va pas provoquer de blocage, mais conduire à la propagation des valeurs, zéro, un ou alarme.

L'idée proposée dans [Moor02] est de supposer que dans les deux premiers cas, le blocage est une situation sûre et donc acceptable. Par contre dans le troisième cas, il est proposé d'agir sur la conception pour garantir la propagation de l'alarme en vue de sa détection. Il faut noter que la source de la faute peut être externe ou interne. Il faut considérer que dans la méthode présentée ici, réside un moyen de stopper le fonctionnement du circuit en réponse à la détection d'une intrusion quelle qu'elle soit : capteurs de lumière, température, tension… Ainsi, la figure 3.10 illustre l'insertion d'une alarme dans le circuit et sa détection.

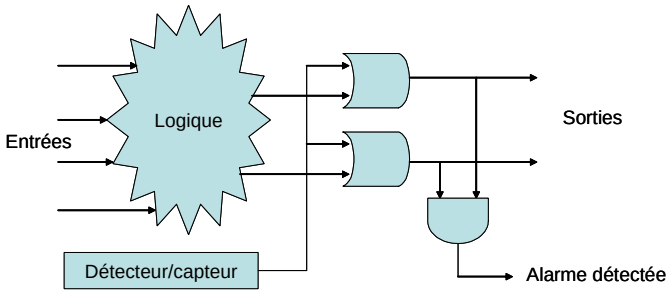


Figure 3.10 Insertion et détection d'une alarme

Il reste à montrer que toute fonction logique combinatoire ou séquentielle peut être réalisée sous la forme d'un réseau de portes logiques tel que l'apparition d'un code d'alarme dans le réseau conduit dans tous les cas à produire un code d'alarme en sortie. Ainsi, si l'injection d'une faute conduit à générer un code d'alarme dans le circuit, il sera possible de détecter la faute en plaçant un détecteur d'alarme sur les sorties. Dans [Moor02], seul le cas des fonctions combinatoires est abordé. Comme nous le montrerons dans le chapitre 4, l'étude de cas proposée lors de l'injection d'une faute unique est correcte mais non exhaustive.

Dans [Won05], une méthode basée sur la duplication est proposée pour rendre un circuit QDI tolérant aux SEU. Ce travail propose une approche formelle en se basant sur une description du circuit dans le langage HSE (Hanshaking Expansion). Le principe est de dupliquer chaque nœud et de provoquer un rendez vous pour chaque nœud dupliqué. La tolérance aux SEU y est prouvée formellement.

Cette méthode implique une augmentation considérable de la surface du circuit. Ainsi, l'application de la méthode à un buffer simple rail conduit à un circuit 3 fois plus gros et 2 fois plus lent que le circuit initial. Cette solution est donc peu intéressante vis-à-vis des solutions proposées sur les circuits synchrones. De plus, le modèle de faute employé est imprécis car aussi bien le circuit que le modèle de fautes sont décrits dans un langage de haut niveau, c'est-à-dire potentiellement peu semblable de l'implantation réelle après synthèse.

Plusieurs techniques de détection de fautes sont proposées dans [Lafr04] à différents niveaux d'abstraction. Plusieurs classes de fautes y sont considérées. En particulier, une analyse de l'effet des fautes de délais sur les fourches isochrones est détaillée et une solution est proposée. La plupart des solutions apportées dans ce travail se font à un niveau inférieur au niveau logique (transitor, layout ...) pour des cellules spécifiques à la conception de circuits QDI. Ainsi, les buffers de type PCHB (Pre Charge

Half Buffer) sont décrits en tant que cellules complexes et considérés au niveau transistor. Cette approche rend donc le travail très dépendant de l'implantation des cellules.

[Wad05] propose une attaque théorique sur les circuits asynchrones utilisant un codage double rail. L'idée est de montrer qu'il est possible d'attaquer un système double rail en injectant une seule faute. On suppose (1) une très grande précision sur le moyen d'injection de faute : la faute est injectée à un instant choisi et sur une porte logique déterminée de la partie combinatoire, et (2) une parfaite connaissance du layout, ce qui constitue des hypothèses extrêmement fortes. Si on applique la méthode à la fonction XOR présentée sur la figure 3.7, on injecte une faute (une inversion logique d'un signal) sur la sortie O_0 au moment où le bloc est actif. En fonction de la réponse du circuit à cette perturbation, on est capable de déduire quelle était la donnée manipulée. En effet, si le fil O_0 était actif, alors la faute aura pour effet de rendre le fil inactif et de provoquer un blocage du circuit. Si c'est le fil O_1 qui était actif, alors les deux fils seront actifs et une erreur sera potentiellement observable par le déclenchement d'une alarme ou par un résultat erroné.

Plusieurs autres analyses sont proposées dans [Wald05], en prenant en compte différents modèles de fautes (transitoires, bit flip ...). Bien que le travail théorique soit intéressant, les hypothèses qu'il suppose sur la connaissance du circuit et sur la précision de l'injection de fautes (précision de l'instant, du lieu, et de l'effet de la faute) le rendent aujourd'hui peu réalisable en pratique.

3.3 Utilisation des modèles de fautes dans le contexte des circuits asynchrones

Nous utilisons dans ce travail des modèles de fautes définis à partir des modèles couramment utilisés dans la littérature et décrits dans le chapitre précédent. De la même façon qu'il a été nécessaire d'adapter les termes utilisés dans la conception des circuits synchrones à la définition de la structure des circuits asynchrones, il est aussi nécessaire d'adapter les termes utilisés pour les modèles de fautes. Nous indiquons aussi dans quelle mesure ces modèles seront utilisés au cours de ce travail.

3.3.1 Les fautes de délais

Les circuits QDI sont par construction très tolérants aux fautes de délais, que ce soit des fautes de retards de portes ou de chemins. La seule possibilité de dysfonctionnement serait une faute de délai sur une branche d'une fourche isochrone. Si la faute de délai affecte toutes les branches de la fourche, le

fonctionnement reste correct. En revanche si toutes les branches ne sont pas concernées, la propriété isochrone de la fourche est perdue, ce qui peut avoir comme conséquence un dysfonctionnement.

Nous avons choisi de ne pas analyser explicitement l'effet des fautes de délai dans ce travail car les circuits asynchrones auxquels on s'intéresse dans ce travail sont naturellement très tolérants à ce type de fautes [TIM01]. On se focalise donc sur l'effet des fautes transitoires et des fautes sur les éléments mémoire, plus dangereux en terme de sécurité. De plus, des solutions aux fautes de délai dans les circuits QDI sont présentées dans [Lafr04] et sont apportées à des niveaux d'abstraction inférieurs à celui défini dans le cadre de ce travail.

3.3.2 Les fautes transitoires

Dans un circuit synchrone, les fautes transitoires SET sont injectées dans la partie combinatoire et correspondent à l'inversion temporaire d'un signal logique. Il est possible d'utiliser ce même modèle pour définir l'injection d'une faute transitoire dans la partie calculatoire d'un circuit asynchrone.

Il convient toutefois de justifier cette équivalence, car comme il a été précisé dans la section 3.1.3.4, la partie calculatoire d'un circuit asynchrone est fréquemment implantée en utilisant des éléments mémorisant. Une faute transitoire injectée sur la sortie d'un de ces éléments ou propagée puis mémorisée par cet élément ne correspond donc plus à une faute transitoire, puisqu'elle a été mémorisée. Cependant cela n'affecte pas l'état global du circuit tel qu'il a été défini précédemment.

Il est possible de montrer que l'on peut se ramener à un cas similaire à celui d'une faute transitoire. La figure 3.11 (a) montre l'effet d'une faute transitoire injectée sur un nœud combinatoire. Le signal est inversé pendant une durée *t1* puis revient a son état initial. La figure 3.11 (b) montre l'effet de la même faute mais sur un élément mémorisant. Le signal ne revient pas à son état initial car la faute a été mémorisée. Cependant lors de la phase de remise a zéro du protocole (phase 2), tous les nœuds internes sont remis à zéro (propriété des circuits QDI), ce qui forcera obligatoirement le point mémoire à revenir à son état initial.

On considère donc qu'une faute transitoire capturée par un élément mémorisant est équivalente à une faute transitoire de durée *t2*. L'effet de l'élément mémorisant est donc de « prolonger » l'effet de la faute transitoire, jusqu'à ce qu'un rendez vous sur les entrées de la porte ne provoque un retour à son état initial.

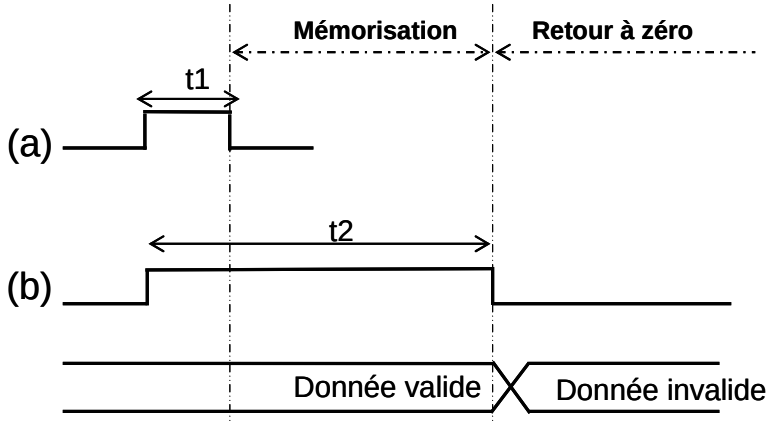


Figure 3.11 Faute transitoire mémorisée dans la partie calculatoire

En conclusion, nous considérons toute perturbation injectée dans la partie calculatoire d'un circuit asynchrone comme une faute transitoire de type SET (ou étendu aux fautes multiples), par analogie avec les fautes injectées dans la partie combinatoire d'un circuit synchrone.

3.3.3 Les Soft Errors

Le modèle SEU défini précédemment correspond au basculement d'un point mémoire dans un circuit synchrone. Par équivalence, nous définissons une *Soft Error* (ou *bit flip*) comme un basculement d'un point mémoire dans un circuit asynchrone, ce qui correspond à l'inversion de la valeur de sortie d'une porte de Muller implantée dans une partie mémoire du circuit. Une Soft Error modifie l'état global du circuit.

Une Soft Error peut être obtenue de deux façons :

- Une faute injectée directement dans la partie mémorisante de la porte de Muller.
- Une faute transitoire est injectée dans la partie calculatoire, elle est propagée jusqu'à l'entrée de la partie mémoire, et elle est mémorisée par cet élément.

3.3.4 Synthèse

La figure 3.12 résume une injection de fautes dans un circuit asynchrone. Une Soft Error est obtenue (a) par propagation et capture d'une faute transitoire, ou (b) par injection d'une faute directement dans un élément mémoire.

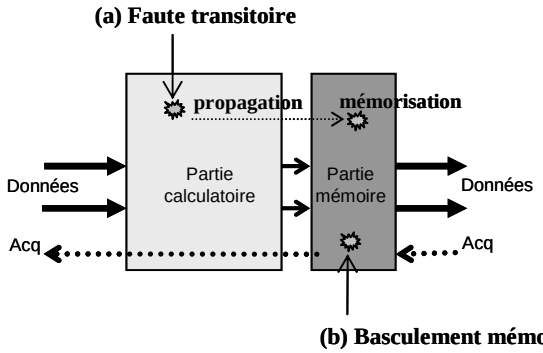


Figure 3.12 Injection de fautes dans un circuit asynchrone

Les conditions de mémorisation d'une faute transitoire font l'objet d'une étude approfondie dans le prochain chapitre. Si la faute est mémorisée, alors nous étudions l'effet de la Soft Error sur le circuit au niveau comportemental. Cette analyse est également développée dans le prochain chapitre.

3.4 Conclusion

Il a été montré que la technologie asynchrone pouvait constituer un apport intéressant pour la conception de systèmes sécurisés contre les attaques DPA. Cette technologie est « pressentie » intéressante pour concevoir des systèmes résistants aux attaques par fautes grâce a ses propriétés inhérentes (codage multi-rail, blocages possibles, tolérance aux fautes de délais). Cependant, peu de travaux ont été réalisés pour confirmer ou infirmer cette idée. Tous les travaux effectués sont théoriques et beaucoup reposent sur des modèles de fautes ou des circuits dont les spécifications sont peu précises.

Nous proposerons dans le chapitre 4 une étude formelle et une analyse comportementale plus détaillée que celle proposée dans [Moor02]. Dans le chapitre 5, nous reprenons le principe de l'alarme avec une implantation différente. Le principe et l'implantation sont ensuite tous deux validés dans la pratique par la conception et le test d'un crypto-processeur DES.

Nous proposerons dans le chapitre 5 une autre faille que celle proposée dans [Wad05], réalisable, permettant à un attaquant d'effectuer une attaque sur un système utilisant un codage double rail et protégé par une alarme, en injectant une faute unique.

Les modèles de fautes classiquement utilisés pour les circuits synchrones doivent être précisément adaptés lorsqu'ils sont employés dans le contexte d'un circuit asynchrone. Les choix effectués et les définitions données pour les fautes transitoires et les Soft Errors ont pour but d'organiser l'analyse de façon

cohérente et de garder une certaine similitude avec les études menées sur les circuits synchrones, comme le résume la figure 3.12. Le prochain chapitre développe en détails l'analyse de sensibilité aux fautes des circuits QDI.

CHAPITRE IV

ANALYSE DU COMPORTEMENT DES CIRCUITS ASYNCHRONES QDI EN PRESENCE DE FAUTES

4.1 Introduction

Ce chapitre est composé de deux parties principales : l'analyse de sensibilité des circuits asynchrones QDI aux fautes transitoires et l'analyse de sensibilité aux Soft Errors. Puis un schéma présente la liste des différents comportements possibles des circuits QDI en présence de fautes. C'est à partir de la liste de ces comportements que des méthodes de durcissement seront proposées dans le chapitre 5.

L'analyse de sensibilité des circuits asynchrones aux fautes transitoires nécessite de déterminer les critères selon lesquels les fautes peuvent être capturées (et donc présenter un danger potentiel pour le circuit) par les éléments de la partie mémoire. La sensibilité des circuits asynchrones dépend donc de la capacité des portes de Muller à capturer une faute transitoire (correspondant à une mémorisation en Soft Error), ou à la filtrer. Ce critère est défini dans la section 4.2 et nous présentons un outil basé sur des analyses en simulation permettant de mesurer la sensibilité aux fautes et de déterminer les points les plus sensibles du circuit.

L'analyse de sensibilité aux Soft Errors est basée sur une analyse formelle du comportement du circuit en présence d'erreurs dans les éléments mémoire. Nous développons un modèle de circuit et nous

établissons une liste exhaustive des comportements en utilisant une simulation symbolique du modèle de circuit en présence d'erreurs.

4.2 Analyse de sensibilité aux fautes transitoires

4.2.1 Définition du critère de sensibilité pour une porte de Muller

On rappelle que la porte de Muller implante une fonction qui copie le niveau logique des entrées sur la sortie lorsque celles-ci ont le même niveau. Si les entrées diffèrent, la sortie reste inchangée.

Définition : une porte de Muller à N entrées est dite M-sensible à 0 (respectivement à 1) lorsque exactement M de ses entrées ainsi que sa sortie valent 0 (respectivement 1).

En effet, si ces M entrées changent de valeur logique, le rendez-vous peut s'opérer entre toutes les entrées et la sortie basculent vers 1 (respectivement 0).

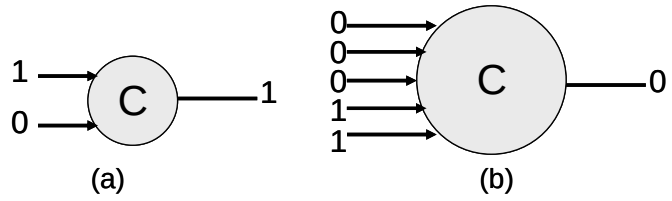


Figure 4.1 Une porte de Muller à 2 entrées 1-sensible à 1 (a) et une porte à 5 entrées 3-sensible à 0 (b)

Sur l'exemple (a) de la figure 4.1, la porte est dans un état 1-sensible à 1. Si la première entrée change de valeur alors la sortie change de valeur, elle bascule vers 0. On peut noter que la porte n'est pas sensible à une faute sur la deuxième entrée, car si cette entrée bascule vers 1, la sortie reste inchangée. Une faute transitoire sur la deuxième entrée est donc filtrée.

L'exemple (b) montre une porte de Muller à 5 entrées dans un état 3-sensible à 0. Une faute unique propagée sur n'importe laquelle des entrées sera filtrée. Il en est de même pour deux fautes sur deux entrées. En revanche, 3 fautes simultanées qui seraient propagées sur les 3 premières entrées généreraient une faute en sortie.

On définit les états d'une porte de Muller à N entrées selon 4 cas :

1 - M-sensible à 0 (avec M variant de 1 à N-1)
2 - M-sensible à 1 (avec M variant de 1 à N-1)
3 – Set
4 - Reset

Les cas 1 et 2 sont les cas dans lesquels la porte est sensible. Chaque cas comporte N-1 états, de 1-sensible à (N-1)-sensible. Les cas 3 et 4 sont les cas où la porte se trouve dans l'état de *Set* et *Reset* (toutes les entrées à 1 ou toutes les entrées à 0). Pour une porte de Muller à N entrées il y a donc 2*(N-1) + 2 = 2N états à caractériser.

Il est clair que les portes de Muller à 2 entrées sont susceptibles d'être particulièrement sensibles aux fautes uniques. Si une porte se trouve dans un état défini par le cas 1 ou 2, la porte est sensible à une faute unique. Seuls les états Set et Reset peuvent filtrer toutes les fautes uniques. On peut remarquer que si la porte se trouve dans l'état Set (respectivement Reset) et que 2 fautes simultanées sont injectées sur les entrées, alors la porte bascule vers l'état Reset (Respectivement Set). Cependant, puisque les fautes sont transitoires, la porte revient finalement dans son état initial. Les états Set et Reset sont des états « passants ». L'effet sur le circuit peut donc être interprété comme étant l'injection d'une faute transitoire unique sur une entrée du bloc suivant.

4.2.2 Définition du critère de sensibilité pour un circuit

L'état du circuit est défini comme étant l'état de l'ensemble de ses portes de Muller. En analysant la séquence des états atteints par chacune des portes de Muller, on est capable d'établir dynamiquement lors de la simulation une cartographie des zones sensibles du circuit.

Les points mémoires qui se trouvent fréquemment dans un état 1-sensible (à 1 ou à 0) sont les points les plus sensibles des circuits, car une faute unique de type SET se propageant sur l'une de leurs entrées sera potentiellement mémorisée par le circuit. Cette faute pourra conduire à une défaillance du circuit.

4.2.3 Implantation de l'algorithme

On souhaite lors de la simulation, pour chaque porte de Muller présente dans le circuit, calculer les paramètres nécessaires à une évaluation de sa sensibilité. Pour chaque porte de Muller, la fonction doit

calculer pour chaque état le temps moyen T_{moy} pendant lequel la porte se trouve dans cet état, en appliquant l'algorithme suivant :

Définitions :

T_{accu} [e] : le temps total pendant lequel la porte est restée dans l'état e.
Occ [e] : le nombre d'occurrences de l'état e.
T_{moy} [e] : le temps moyen pour l'état e sur l'ensemble des occurrences.

Initialisation :

T_{moy} [tous les états] : = 0
T_{accu} [tous les états] : = 0
Occ [tous les états] : = 0
Timer : = 0

A chaque transition d'un état vers un autre, faire :

T_{accu} [etat_prec] : = T_{accu} [etat_prec] + Timer
Timer := 0
Occ [etat_prec] : = Occ [etat_prec] + 1
T_{moy} [etat_prec] : = T_{accu} [etat_prec] / Occ etat_prec]

Le passage d'un état à un autre correspond à une transition sur une ou plusieurs entrées. On peut noter que si une seule entrée bascule, on change d'état. On pourra, à partir de ces valeurs, calculer un indice de sensibilité pour chaque porte.

4.2.4 Affinement du critère de sensibilité : Validation/Invalidation d'états

La figure 4.2 présente l'évolution de la séquence des états d'une porte de Muller, pendant une exécution normale (sans injection de fautes). Nous prenons l'état Reset comme état initial. Pendant l'exécution, la porte atteint une séquence d'états sensibles aux fautes, c'est-à-dire les cas 1 et 2 définis précédemment, car certains signaux d'entrée deviennent actifs. Il y a alors deux scénarios possibles : soit tous les signaux d'entrée convergent vers 1 et l'état Set est atteint (Figure 4.2a), soit les signaux convergent vers 0 et la porte revient dans l'état Reset (Figure 4.2b) sans que sa sortie ait basculé.

Dans le cas (a), la porte n'a finalement pas été sélectionnée pour mémoriser la donnée courante. Une autre porte a mémorisé la donnée. La porte revient donc à son état initial lors de la phase de remise à zéro du protocole de communication. Dans le cas (b), la porte a mémorisé la donnée.

- **Validation d'un état sensible** :

Lorsque la porte traverse une séquence d'états sensibles et qu'elle revient dans son état initial (a), les états sensibles qui ont été atteints doivent être validés comme étant effectivement sensibles. En effet, si N fautes étaient injectées pendant que la porte se trouve dans un des états N-sensibles traversés, alors une Soft Error serait mémorisée et la porte atteindrait l'état Set, ce qui est différent du comportement attendu.

- **Invalidation d'un état sensible** :

Lorsque la porte traverse une séquence d'états sensibles et atteint l'état Set (b), les états sensibles qui ont été atteints doivent être invalidés car si N fautes faisaient basculer la porte pendant l'un de ces états N-sensibles, la porte basculerait prématurément dans l'état Set, qu'elle devait atteindre. Cela peut être interprété comme une faute de délai (déclenchement prématuré) sur la sortie de la porte de Muller. Puisqu'il ne s'agit pas d'une branche dans une fourche isochrone, cette faute de délai n'a pas de conséquence sur le fonctionnement du circuit. C'est pourquoi on retire (invalide) cette séquence d'états de la liste des états sensibles atteints par la porte.

Il faut cependant noter que l'invalidation d'un état sensible n'est autorisée que dans des structures en boucles (structures itératives) où le nombre de données circulant dans le circuit est connu et maîtrisé. S'agissant des structures linéaires, où le nombre de données présentes est inconnu, la faute ne peut être interprétée comme une mémorisation prématurée car il est possible qu'elle corresponde à la génération d'une donnée. Cet aspect est abordé dans la section 4.3.

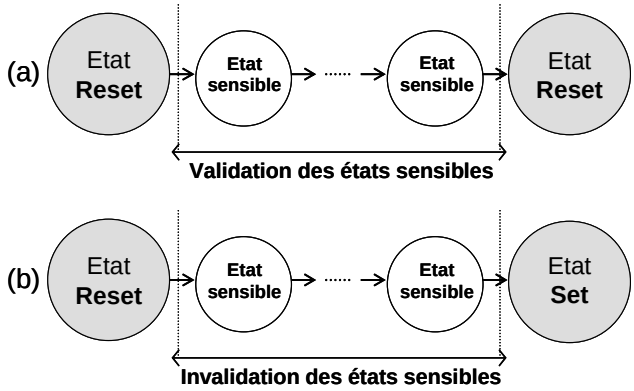


Figure 4.2 Evolutions possibles des états d'une porte de Muller pendant une exécution sans faute

La figure 4.3 présente une structure de Half-Buffer. Quand cet étage est prêt à mémoriser une donnée, l'acquittement 'Acq' vaut 1. Les portes M00 et M01 sont dans un état sensible car elles peuvent mémoriser une donnée sur (I_0, I_1) ou une faute propagée sur ce canal. Supposons que la donnée soit (1,0). La porte M00 entre dans l'état Set (Figure 4.2b), tandis que M01 reste dans l'état sensible. Lorsque la donnée est acquittée par l'étage suivant, 'Acq' revient à 0 et M00 revient à l'état Reset initial (Figure 4.2a). L'état sensible atteint par M01 est donc validé car la mémorisation d'une faute par cette porte aurait conduit le circuit dans un état global incorrect.

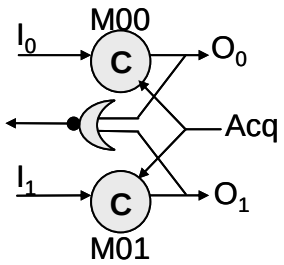


Figure 4.3 Half Buffer

4.2.5 Implantation de l'outil

La figure 3.4 présente un résumé du flot de conception des circuits asynchrones QDI au sein du groupe CIS ainsi que la façon dont l'outil d'analyse de sensibilité aux fautes peut s'intégrer. Après synthèse, le circuit au format netlist verilog peut être simulé avec les outils standards du commerce.

L'outil d'analyse de sensibilité est un module lancé par le simulateur. Pendant la simulation, l'outil surveille l'activité des portes de Muller de la partie mémoire (qui définissent l'état global du circuit) et

enregistre les états successifs atteints par ces portes pendant la simulation, en fonction de leur activité. Pour parvenir à un résultat plus précis, il est possible d'analyser le circuit après l'étape de placement routage, grâce aux informations de délais fournis par un fichier SDF (Standard Delay Format).

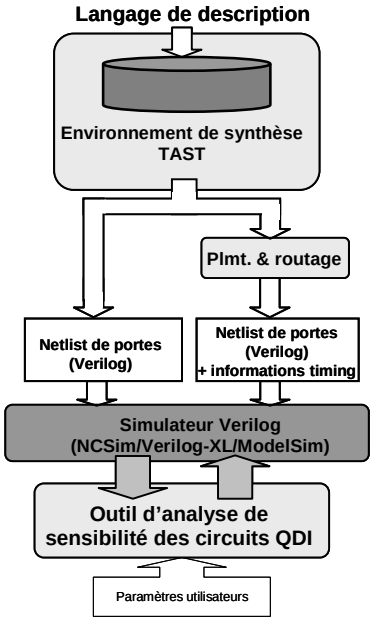


Figure 4.4 Flot de conception et intégration de l'outil d'analyse

L'algorithme a été implanté sous forme d'une bibliothèque de fonctions écrite en langage C. Le simulateur utilise cette bibliothèque grâce à l'interface PLI (Progamming Language Interface) qui permet une interaction directe entre la netlist Verilog et le simulateur. Le PLI permet d'enrichir le code Verilog en ajoutant des pseudo-instructions qui pilotent les fonctions d'analyse de sensibilité. La bibliothèque a été réalisée en quelques centaines de lignes de code C et elle fonctionne sur la plupart des simulateurs Verilog du commerce. Les routines PLI sont très peu coûteuses en temps de simulation. Ainsi, on peut analyser des circuits complexes sans que le temps de simulation ne soit pénalisé.

L'utilisateur a la possibilité de définir plusieurs paramètres pour mieux cibler son analyse :

- Choix des types de portes à analyser, en fonction de leur nom.

- Choix du module à analyser : il est possible d'analyser tout le circuit ou seulement un ensemble de sous modules particuliers, considérés par exemple comme particulièrement critiques pour une application.

- Possibilité de définir le temps de simulation et l'instant du début de l'analyse pour cibler une zone temporelle précise.

A la fin de la simulation, l'outil fournit un rapport exprimant la sensibilité globale du circuit simulé et un rapport d'activité détaillé pour chaque porte. On connaît ainsi les points du circuit les plus sensibles aux fautes, c'est-à-dire les points qui sont le plus susceptibles de capturer une ou plusieurs faute(s) transitoire(s) propagée(s) jusqu'à l'entrée des éléments de mémorisation.

Cette approche permet d'obtenir une évaluation quantitative de la sensibilité aux fautes transitoires des circuits QDI. Ainsi, il est possible de comparer soit plusieurs architectures soit une même architecture avant et après l'implantation d'une technique de durcissement.

4.2.6 Exemple d'analyse de sensibilité

Nous présentons à titre d'exemple l'analyse d'un bloc de l'architecture d'un crypto-processeur DES asynchrone, qui sera détaillée dans le chapitre 6. La figure 4.5 présente un bloc de logique constitué de la logique implantant les S-Box et de la fonction XOR. Nous souhaitons comparer la sensibilité d'une architecture avant (figure 4.5a) et après (figure 4.5b) durcissement. La technique de durcissement employée est une simple duplication de la partie calculatoire du bloc. Nous reviendrons sur cette méthode dans le chapitre 5 dédié au durcissement.

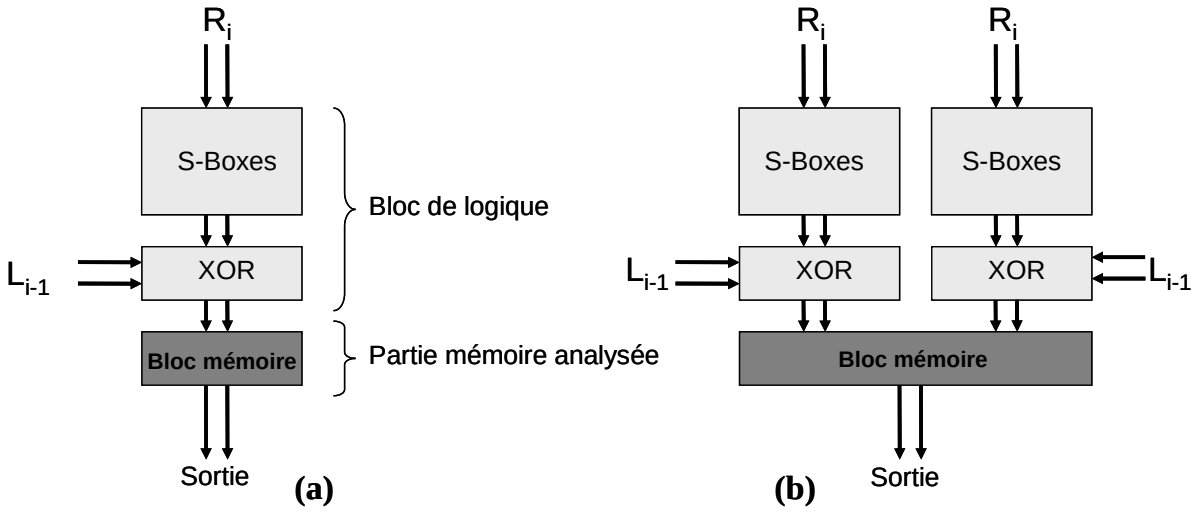


Figure 4.5 Architecture du module analysé (a) avant et (b) après durcissement

Un jeu de tests est appliqué pour réaliser les analyses de sensibilité sur un temps de simulation de 53.3 ns avant placement/routage. 64 cellules du bloc de mémoire sont analysées. Les tableaux 4.1 et 4.2 présentent les résultats obtenus pour une cellule. Sur le tableau 4.1, la cellule du bloc non durci est sensible à une faute unique pendant 14.4 ns, ce qui signifie que pendant 27% du temps de simulation, une faute transitoire serait mémorisée en Soft Error par la cellule si elle se propageait jusqu'à son entrée A. Les portes de Muller implantées dans ce bloc de mémoire sont des portes de Muller à 2 entrées.

Après durcissement, la cellule n'est plus sensible à une faute unique (tableau 4.2). La méthode de duplication basique permet de rendre le bloc tolérant aux fautes transitoires uniques. Deux fautes transitoires simultanées sur les entrées A et B de la cellule sont nécessaires pour être mémorisées en Soft Error. Pour permettre le rendez vous entre les signaux du bloc de la logique et du bloc dupliqué, la partie mémoire doit être implantée avec des portes de Muller à 4 entrées.

Tableau 4.1 Analyse de sensibilité pour une cellule du bloc non durci

Temps de simulation :	53 300 ps
Temps moyen dans un état 1-sensible :	14 400 ps
Ports sensibles :	Entrée A

Tableau 4.2 Analyse de sensibilité pour une cellule du bloc durci

Temps de simulation :	53 300 ps
Temps moyen dans un état 1-sensible :	0 ps
Temps moyen dans un état 2-sensible :	14 550 ps
Ports sensibles :	Entrées A,B

D'autres exemples d'analyses plus complètes effectuées après l'étape de placement/routage sont présentés dans le chapitre 5. Ils permettent de valider d'autres techniques de durcissement, plus efficaces en terme de surface et de consommation.

4.2.7 Conclusion

Le critère défini nous permet de mieux comprendre la façon dont une faute transitoire est mémorisée ou filtrée dans un circuit asynchrone. L'implantation de l'algorithme dans un environnement de simulation permet de connaître dynamiquement quels sont les blocs les plus sensibles du circuit afin de les protéger, ou de comparer l'efficacité de deux techniques de durcissement. Il est clair que la quantification de la sensibilité dépend d'une part des données manipulées, et d'autres part des délais du circuit. C'est pourquoi il est préférable d'effectuer plusieurs simulations (en faisant varier les données) avec des circuits routés pour avoir une meilleure précision dans l'analyse.

Cette approche ne nous permet pas de comparer la sensibilité d'un circuit asynchrone avec celle d'un circuit synchrone. Une approche similaire pour les circuits synchrones a précédemment été réalisée avec l'outil ROBAN d'IroC Technologies [Ale02]. Les critères de calcul sont dans la pratique très différents de ceux définis pour les circuits asynchrones. La comparaison quantitative entre synchrone et asynchrone n'a pour l'instant pas été étudiée.

4.3 Analyse de sensibilité aux Soft Errors

4.3.1 Introduction

Lorsqu'une Soft Error est mémorisée, l'état global du circuit est changé. Il faut alors analyser le circuit au niveau comportemental : comment, à partir de cet état erroné, le circuit va-t-il évoluer ?

Bien qu'une approche en simulation soit possible, avec un modèle d'injection de fautes dans le circuit au niveau portes logiques, cette solution serait non exhaustive et peu efficace car elle ne s'appuie pas sur les spécificités d'un circuit asynchrone, qui a un fonctionnement similaire à celui des systèmes de flot de données. C'est pourquoi nous envisageons une approche formelle avec l'utilisation d'un modèle de circuit. Il est nécessaire pour comprendre le comportement du circuit de construire un modèle permettant :

- De reproduire et simuler le comportement du circuit, avec une complexité acceptable.
- D'injecter un modèle de Soft Error dans ce circuit.
- De reproduire avec fidélité l'évolution du circuit en présence de cette erreur.

Plusieurs modèles formels existants peuvent être envisagés, tels que les réseaux de Petri [Dia01], le formalisme SMV [Bur90], ou les STGs [Chu85]. Cependant, il apparaît qu'aucun d'entre eux n'est entièrement satisfaisant sur les trois critères énoncés.

La représentation des circuits asynchrones en système de jetons communicant entre chaque module, appelé « *token game* » a été développée dans [Spar01] puis dans [Ren03]. Dans le cadre de ce travail, ce modèle constitue une base à partir de laquelle nous construisons un modèle plus fin pour représenter le circuit et l'injection de fautes à un niveau d'abstraction satisfaisant. A partir de ce nouveau modèle, nous proposons une étude comportementale basée sur la simulation symbolique qui nous permet d'établir des preuves de propriétés ou au contraire des exemples de failles dans le comportement du circuit.

4.3.2 Le « token game »

Dans un système basé sur le flot de jetons (ou « *tokens* »), un jeton représente une information stockée dans un élément mémoire. Lorsqu'on utilise un protocole de communication à 4 phases, l'activité du circuit est représentée par un flot alternant des **jetons valides** notés **V**, des **jetons invalides** (la remise à zéro) notés **I**, et des **bulles** (mémoire vide ou libre) notés **B**. Une **donnée** est composée d'un jeton V et d'un jeton I.

Dans le modèle existant [Ren03], les règles de flots de données sont les suivantes :

- Règle 1 : une mémoire peut mémoriser un jeton (V ou I) envoyé par son prédécesseur si et seulement si elle contient une bulle.

- Règle 2 : une mémoire se vide (devient B) si et seulement si son successeur a reçu et mémorisé le jeton qu'elle contenait.

La figure 4.6 présente un circuit composé de 3 étages « pipelinés ». Dans cette représentation, chaque rectangle représente la partie mémoire d'un étage (un Half Buffer). Les parties calculatoires sont complètement abstraites. Le deuxième étage contient une donnée valide (jeton V). Cette donnée a la possibilité d'avancer au troisième étage car celui-ci est vide (règle 1). Le premier étage contient un jeton I qui complète la donnée et qui assure la remise à zéro imposée par le protocole de communication. Lorsque la donnée aura été mémorisée par le troisième étage, le deuxième étage pourra alors se vider (règle 2) et le jeton I pourra avancer sur cet étage (règle 1). Les deux jetons de ce circuit auront alors avancé d'un étage dans le pipeline.

Figure 4.6 Représentation de 3 étages d'un circuit

Il est possible de modéliser des architectures complexes mettant en jeu des boucles avec initialisation (machines à états), des rendez-vous et des fourches, avec génération et consommation de jetons. La relation entre le nombre d'étages qui composent une boucle et le nombre de données qui peuvent y circuler a été établie dans [Wil91] et [Wil94].

Soit N le nombre d'étages dans une boucle et K le nombre de données qui circulent dans cette boucle. On rappelle qu'une donnée est constituée d'un jeton V et d'un jeton I. Il a été établi que N doit être supérieur ou égal à 2K+1 pour assurer la propriété de vivacité de la boucle. Si N < 2K+1, alors la boucle se bloque.

Pour les besoins particuliers de ce travail, il est nécessaire d'enrichir ce modèle :

- De nouveaux jetons sont nécessaires pour modéliser une Soft Error (données erronnées)
- D'autres règles doivent être définies pour modéliser l'évolution du circuit lorsqu'il atteint un état anormal.
- Le modèle de fautes doit être cohérent avec le modèle de Soft Error défini au niveau porte (basculement d'un bit de mémoire). Il faut donc redéfinir le modèle du circuit à un niveau d'abstraction satisfaisant.

4.3.3 Définition des jetons

Nous définissons l'alphabet suivant :

Jetons valides = {**V1**, **V2**, **F**}
Jeton invalide = {**I**}
Bulles = {**BV1**, **BV2**, **BI**}

4.3.3.1 Jetons valides

Un jeton valide représente la valeur d'une donnée qui circule dans le circuit. La valeur de cette donnée est abstraite. Cependant il est nécessaire de faire une distinction entre plusieurs types de valeurs.

- **V1** est un jeton dont la valeur est correcte, non corrompue par une faute.
- **V2** est un jeton dont la valeur est erronée; toutefois cette valeur appartient à l'ensemble des valeurs correctes, c'est-à-dire dont le code est parfaitement valide.
- **F** (*Forbidden*) est un jeton dont la valeur est erronée et son code n'appartient plus à l'ensemble des codes valides du circuit.

Par exemple, lorsque le circuit manipule des données double rail, les jetons V1 et V2 appartiennent à l'ensemble {"01, "10"} et F est le code interdit "11". Cette définition peut être étendue aux codes *1 parmi n*. Dans ce cas, V1 et V2 appartiennent à l'ensemble des codes *1 parmi n* valides, et F est le sous-ensemble des valeurs *m parmi n* avec $1 < m \leq n$.

4.3.3.2 Jeton invalide

Le jeton invalide est la deuxième partie de la donnée, nécessaire pour assurer la remise à zéro. Il n'y a qu'un seul jeton invalide, noté **I** (par exemple "00" en double rail).

4.3.3.3 Bulles

Une bulle indique que la mémoire est inutilisée et qu'elle est libre de recevoir un jeton. Quand une mémoire est vide, elle contient la copie du jeton qu'elle mémorisait précédemment. Ce jeton a été reçu et acquitté par l'étage suivant, c'est pourquoi la mémoire est considérée comme vide. On distingue donc **BV1**, la copie d'un jeton V1, **BV2** la copie d'un jeton V2, et **BI** la copie d'un jeton I. On ne considère pas de jeton "BF" car c'est inutile pour notre analyse : quand un jeton F est produit, on considère que sa détection est possible en implantant une alarme et qu'il est inutile de poursuivre l'analyse de l'évolution du circuit. Nous expliquons ce choix dans le paragraphe "Génération d'un jeton F" de la Section 4.3.6.

Une bulle contient donc la même valeur que le jeton qu'elle représentait à l'origine. En double rail, les valeurs de BV1 et BV2 appartiennent à l'ensemble {"01, "10"} et BI est "00". Si on considère le circuit au niveau logique, cela signifie que les portes de Muller qui mémorisent les jetons V et les bulles BV mémorisent la même valeur en sortie. Cependant il existe une différence au niveau des signaux d'entrée : dans le cas d'un jeton V, le signal d'acquittement branché sur la porte a pour valeur '1' car le jeton n'a pas été reçu et acquitté par l'étage suivant; dans le cas d'une bulle BV, le signal d'acquittement aura la valeur '0' car le jeton a été acquitté et la mémoire courante est effectivement une bulle.

Il est intéressant de noter que la distinction opérée entre les différents types de jetons/bulles se retrouve aussi au niveau logique, ce qui tend à montrer que le niveau d'abstraction choisi est satisfaisant.

4.3.4 Définition des règles d'évolution

Pour effectuer une simulation symbolique du circuit avec l'alphabet de jetons défini, nous devons définir de nouvelles règles d'évolution du circuit. Les règles doivent répondre à la spécification du comportement du circuit en fonctionnement normal mais aussi prévoir l'évolution du circuit lorsque son état a été corrompu par une faute. Les règles sont définies dans le format suivant :

$$\{S1, S2\} \rightarrow \{S1', S2'\}$$

Avec S1, S2, S1', S2' appartenant à l'alphabet spécifié dans la section précédente.

{S1, S2} représente un couple d'étages adjacents (l'étage courant et son successeur) et {S1', S2'} représente le même couple après exécution de la règle. On note qu'il est inutile de considérer un étage courant par rapport à son prédécesseur car cette relation est incluse dans la définition des règles ayant pour état courant l'état de ce prédécesseur. Nous définissons l'ensemble de règles suivant lorsque l'**étage courant** (premier élément du couple) a pour valeur **V1** :

(1) {V1, BI} → {V1, V1}
(2) {V1, V1} → {BV1, V1}
(3) {V1, V2} → {BV1, V2}, {V1, F}
(4) {V1, BV1} → {BV1, BV1}
(5) {V1, BV2} → {BV1, BV2}

- La règle (1) a la même spécification que la première règle définie dans la section 4.3.2. Le jeton V1 avance à l'étage suivant car cet étage est libre (BI).

- La règle (2) spécifie la suite de l'évolution après l'application de la première règle: une fois la donnée reçue par l'étage suivant, un acquittement est envoyé à l'étage courant qui devient donc BV1.

- Les règles (3) (4) et (5) ne sont jamais utilisées lorsque aucune erreur n'a été injectée dans le circuit. En revanche elles sont utiles pour simuler le comportement du circuit en cas d'injection

d'une faute. La règle (3) spécifie deux évolutions possibles du couple {V1, V2}. Dans le premier cas, l'étage courant est acquitté par l'étage suivant et V1 devient donc BV1. Dans le second cas, le jeton de l'étage courant avance et "fusionne" avec le jeton de l'étage suivant. Par définition de V1 et V2, nous savons que l'opération génère un jeton F. Physiquement, le choix de l'évolution se fait en fonction des temps de propagation et des contraintes du circuit. Puisque dans ce modèle nous abstrayons complètement la notion de délai, nous devons considérer les deux possibilités lors de la simulation symbolique.

On peut noter qu'aucune règle n'est définie pour la paire {V1, I} car il n'y a aucune possibilité d'évolution pour cette paire.

Les 5 règles présentées ci-dessus sont spécifiées lorsque l'état courant a pour valeur V1. D'autres règles (24 au total) sont définies pour spécifier de façon **exhaustive** le comportement du circuit en fonctionnement normal et en présence de fautes. La figure 4.7 présente l'ensemble des règles.

{V1, BI} → {V1, V1}	{V2, BI} → {V2, V2}
{V1, V1} → {BV1, V1}	{V2, V2} → {BV2, V2}
{V1, V2} → {BV1, V2}, {V1, F}	{V2, V1} → {BV2, V1}, {V2, F}
{V1, BV1} → {BV1, BV1}	{V2, BV2} → {BV2, BV2}
{V1, BV2} → {BV1, BV2}	{V2, BV1} → {BV2, BV1}
{BV1, V2} → {BV1, F}	{BV2, V1} → {BV2, F}
{BV1, I} → {V1, I}	{BV2, I} → {V2, I}
{BV1, BI} → {BV1, V1}, {V1, BI}	{BV2, BI} → {BV2, V2}, {V2, BI}
{BI, V1} → {I, VI}	{I, BV1} → {I, VI}
{BI, V2} → {I, V2}	{I, BV2} → {I, V2}
{BI, BV1} → {BI, I}, {I, BV1}	{I, I} → {BI, I}, {I, BV1}
{BI, BV2} → {BI, I}, {I, BV2}	{I, BI} → {BI, I}, {I, BV2}

Figure 4.7 Règles d'évolution

Dans des architectures de circuit plus complexes, des jetons peuvent être consommés ou produits dans des structures de rendez-vous (*joins*) ou de branches (*forks*). La figure 4.8 représente un élément de rendez-vous réalisant une opération logique "op" quelconque (par exemple un ET logique entre deux valeurs d'entrées). Les règles d'évolution sont appliquées à cet élément mais il faut prendre en considération la synchronisation des jetons. Il faut donc combiner les règles entre elles. Dans l'exemple figure 4.8a, les jetons V1 et V2 peuvent être consommés et être mémorisés dans l'étage suivant. Ce système évolue donc vers l'état présenté sur la figure 4.8b.

L'évolution de l'état (a) vers l'état (b) est possible car d'une part les deux jetons sont présents en entrée (le rendez-vous est donc possible) et d'autre part l'étage suivant est disponible. Si l'une de ces deux conditions n'était pas remplie, aucune évolution ne serait possible.

On remarque également que l'opération "V1 op V2" produit un jeton "V2" quelque soit la signification de l'opérateur "op". En effet, nous considérons que de façon générale si une des deux opérandes est corrompue alors le résultat de l'opération sera corrompu, même si dans certains cas la faute pourrait être absorbée pour produire un résultat juste.

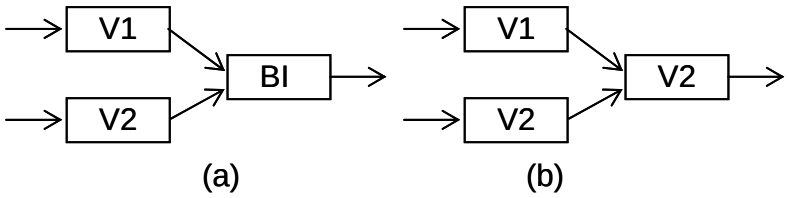


Figure 4.8 Structure de rendez-vous

La figure 4.9 illustre l'évolution d'un jeton V1 dans une structure de branche. Puisque les deux étages suivants sont libres, le jeton peut se propager dans les deux branches (b). Le jeton ne sera acquitté que lorsque les deux branches auront mémorisé et acquitté V1.

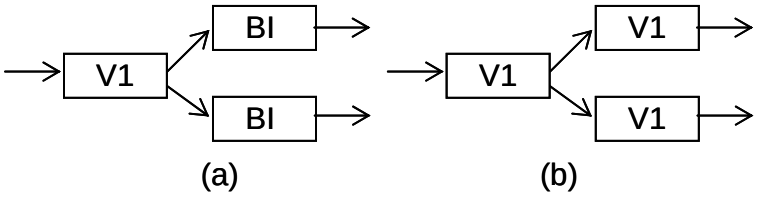


Figure 4.9 Structure de branchement

Ces deux structures de rendez-vous sont nécessaires et suffisantes pour modéliser des circuits asynchrones complexes.

4.3.5 Le modèle d'injection de fautes

Cette section spécifie un ensemble de règles qui modélisent une injection de fautes. Pour chacun des éléments appartenant à l'alphabet, nous définissons la manière dont il est corrompu par une Soft Error (unique). Nous nous basons sur le modèle de Soft Error au niveau logique, c'est-à-dire le basculement d'un

point mémoire, et nous l'interprétons pour produire les règles suivantes sur les jetons. Chaque règle a pour format:

$$S1 \rightarrow S1'$$

Avec S1, S1' appartenant à l'alphabet. S1' est l'état corrompu après injection d'une faute sur S1.

Cinq règles sont produites:

(1) V1 → BI
(2) BI → V2
(3) BV1 → I
(4) I → BV2
(5) V1 → F

- La règle (1) est la **consommation d'un jeton**. Le jeton V1 disparaît et la mémoire devient une bulle. Par exemple, la valeur "01" est corrompue en "00". Cette opération est possible en basculant un seul bit de mémoire.

- La règle (2) est la **génération d'un jeton**. Nous considérons que le jeton généré est incorrect (V2). Nous écartons le cas particulier de la génération d'un jeton V1 (basculement prématuré) car le circuit se comporte alors de la même façon que sans injection de faute.

- Les règles (3) et (4) sont respectivement **consommation d'une bulle** et **génération d'une bulle**. Leur fonctionnement est similaire aux deux premières règles.

- La règle (5) correspond à la **corruption d'une donnée**. Dans ce cas une donnée valide est corrompue en donnée interdite (par exemple "01" → "11").

Nous ne considérons ici que l'injection d'une faute unique, c'est-à-dire un seul basculement d'une porte de Muller. La modélisation de fautes multiples est cependant possible, en rajoutant des règles tels que V2 → BI, V2 → F, ou encore V1 → V2.

4.3.6 Simulation symbolique du circuit

Un algorithme permettant d'effectuer une simulation symbolique du circuit à partir des règles établies a été implanté en C. La figure 4.10 résume son fonctionnement et le positionne par rapport au flot de conception des circuits asynchrones.

Etape 1: Le modèle de circuit est construit à partir de l'un des formats intermédiaires internes à l'outil de synthèse. Il est possible d'extraire depuis ce format intermédiaire l'architecture du circuit en terme d'étages et son initialisation. A l'heure actuelle, cette étape est manuelle.

Etape 2: Une opération d'expansion (*unfolding*) du circuit est nécessaire. Elle a pour but de rendre le circuit indépendant des données. Si les valeurs des données manipulées peuvent être abstraites facilement, il n'en va pas de même pour les valeurs de signaux de contrôle qui pilotent des structures comme les multiplexeurs et les démultiplexeurs. Pour rendre le modèle indépendant de ces variables, le modèle du circuit doit subir une expansion. Cette étape est à l'heure actuelle également manuelle.

Etape 3: Enumération des états du circuit. Une simulation symbolique est effectuée pour déterminer le graphe des états atteignables dans le fonctionnement normal du circuit, en appliquant les règles d'évolution du circuit. L'ensemble des états atteignables est stocké sous forme de liste (L1).

Etape 4: Injection de faute. Pour chaque état de L1 et pour chaque étage du circuit, toutes les règles d'injection de fautes sont appliquées. On obtient alors une liste L2 d'états corrompus.

Etape 5: Enumération des états. A partir de chaque état corrompu de L2, une simulation symbolique est effectuée pour analyser l'évolution du circuit. Ainsi, on connaît le comportement du circuit quelle que soit le point et le moment d'injection de la soft error.

Etape 6: Analyse des résultats.

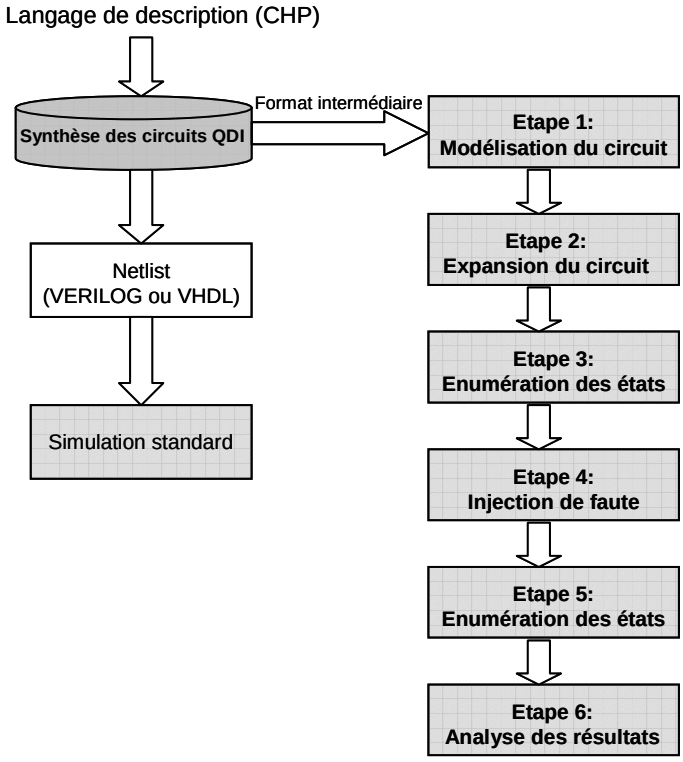


Figure 4.10 Etapes de la simulation symbolique

Il y a quatre types d'évolution du circuit à partir d'un état corrompu :

- **Blocage du circuit** : la simulation atteint un état pour lequel aucune règle ne peut plus être appliquée.

- **Génération d'un jeton F** : lorsqu'un jeton F est généré, nous choisissons de stopper la simulation car le circuit atteint un état corrompu qui est supposé facilement détectable, par exemple par des alarmes. Sur un circuit non protégé par des alarmes, l'évolution d'un tel jeton n'est pas possible à déterminer avec notre modèle. La propagation ou la perte d'un jeton F dépend (1) des propriétés des éléments de la partie calculatoire (qui sont abstraits dans notre modèle) à filtrer ou à propager un jeton F, et (2) des délais de propagation relatifs des rails (qui sont indéterminés dans notre modèle). Il n'y a donc pas d'intérêt à continuer la simulation. On peut cependant noter qu'en pratique il est facile d'implanter des structures calculatoires qui propagent toujours un code faux. Cela permet de détecter un jeton F par le biais d'une alarme à un autre point du circuit.

Si il existe une différence importante entre les délais de propagation des rails du canal sur lequel se trouve le jeton F, alors il est possible que celui-ci soit perdu. En effet, cela peut permettre à l'étage suivant de ne capturer qu'une partie du jeton F et donc de le perdre (F → V2). C'est un aspect théorique non abordé ici car il s'agit d'un cas extrême qui, en pratique, est évité très facilement.

- **La faute est filtrée:** Le circuit finit par atteindre un des états de L1. Cela signifie que la faute n'a pas eu d'effet sur le fonctionnement du circuit.

- **La faute n'est pas détectée**: La simulation couvre un ensemble d'états disjoint de L1. La faute ne peut pas être détectée car aucun blocage n'est constaté et aucun jeton n'a été corrompu en jeton F.

La prochaine section s'appuie sur un exemple pour illustrer les étapes 3, 4 et 5.

4.3.7 Exemple

La figure 4.11a présente une machine à états représentée par un anneau à 3 étages qui comporte un jeton initial V1. Les entrées et sorties éventuelles de la machine à états ne sont pas représentées. La figure 4.11b représente l'**état initial** de ce circuit.

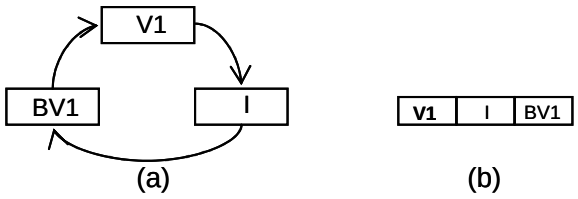


Figure 4.11 Machine à états à 3 étages

La figure 4.12 présente la liste des états atteignables après l'énumération effectuée à l'étape 3. La liste L1 est donc composée de 12 états. On peut observer la façon dont le jeton V1 se propage dans l'anneau et revient à sa position initiale. On constate également que chaque état a un et exactement un successeur, ce qui signifie qu'une seule règle est applicable à chaque fois. Sur un circuit plus complexe, par exemple un anneau contenant plusieurs V1 pouvant évoluer de façon relativement indépendante, on constaterait des évolutions concurrentes et la liste des états atteignables serait un arbre d'états.

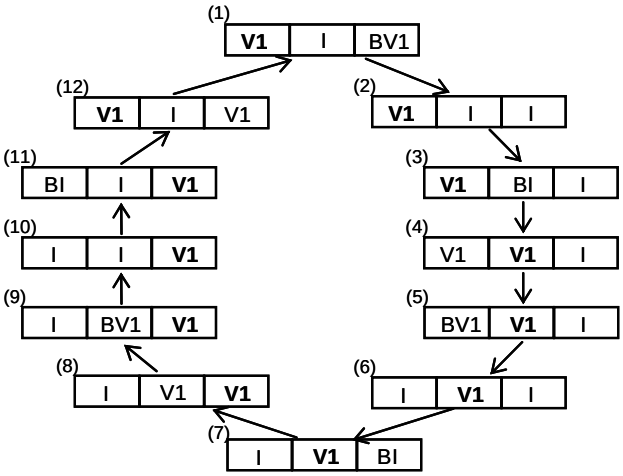


Figure 4.12 Etape 3 (Enumération)

Pour chacun des 12 états de L1 et pour chaque étage, on effectue une injection de faute. La figure 4.13 montre l'injection de fautes effectuée pour l'état **(3) de L1**, ce qui mène aux états corrompus (f1), (f2), et (f3). Il en va de même pour les 11 autres états de L1.

On obtient donc une liste **L2 = {f1, f2, f3}** ∪ **E**, avec **E** l'ensemble des états corrompus obtenus à partir des 11 autres états atteignables.

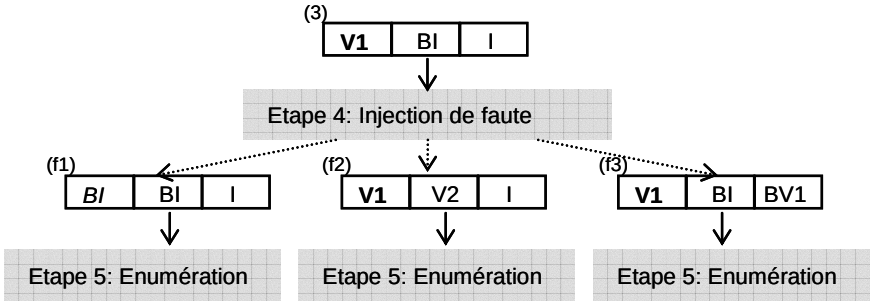


Figure 4.13 Etape 4 (Injection de fautes)

Puis, à partir de chaque état corrompu, on effectue une énumération pour connaître l'évolution du circuit. Dans cet exemple, la liste finale comporte 124 états. L1 représente un sous-ensemble de cette liste.

L'état (f1) conduit à un blocage du circuit. Aucune règle n'est applicable à partir de cet état. En effet on constate que la faute a consommé le jeton, l'anneau est donc vide. La figure 4.14 montre le comportement du circuit à partir de l'état erroné (f2).

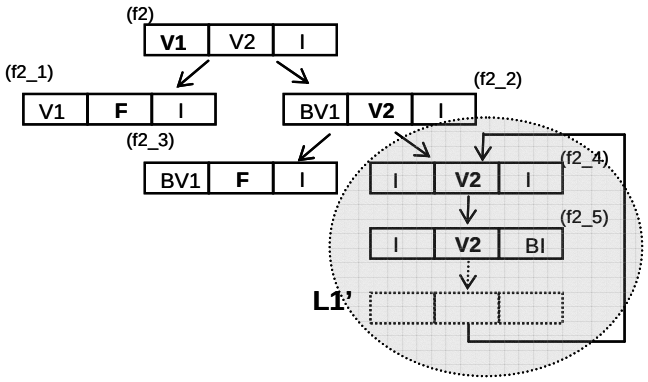


Figure 4.14 Etape 5 (énumération des états fautés)

A partir de l'état (f2), deux états sont atteignables. Si l'état (f2_1) est atteint, la simulation s'arrête car un jeton F a été généré. Si les états (f2_2) puis (f2_4) sont atteints, le circuit entre dans une séquence d'états **L1'** qui est équivalente à L1, mais différente. En effet, l'état (f2_4) est équivalent à l'état (6) de L1. Cependant le jeton V1 a été substitué par un jeton V2. Cette situation illustre le cas où une faute modifie la donnée courante sans qu'elle puisse être détectée.

L'étape 6 analyse les résultats et reporte pour chaque cas le comportement du circuit : blocage, détection par code faux, faute filtrée, faute non détectée. Les seuls cas dangereux sont les cas où la faute n'est pas détectée. L'outil fournit un rapport détaillé sur tous les cas dangereux constatés et rapporte la séquence d'événements permettant d'arriver à cette situation à partir de l'injection d'une Soft Error dans un lieu donné à un moment donné.

Dans l'exemple proposé, 3 cas dangereux ont été reportés. Tous sont équivalents à la situation présentée sur la figure 4.14.

4.3.8 Exploitation de la faille

La faille relevée dans l'exemple précédent est commune à tout type de circuit QDI utilisant ce type de protocole de communication. Son existence réelle dépend des différents délais du circuit. En fonction du délai de propagation des signaux ou des contraintes de rendez-vous avec un circuit extérieur, le circuit

évoluera de façon différente. Prenons l'exemple de l'évolution de l'état (f2_2). Si le premier étage du circuit est "rapide" alors la donnée aura le temps de se propager et de générer un jeton F. Si au contraire le premier étage est "lent" alors le jeton I présent dans le troisième étage a la possibilité d'être mémorisé à la place de BV1.

Ce scénario a été vérifié dans un environnement de simulation. Le module simulé est le compteur d'un circuit DES asynchrone. Ce module est une machine à états qui peut être modélisée par l'anneau à 3 étages présenté dans la section précédente. Une Soft Error a été injectée manuellement pendant la simulation en forçant un signal logique de sortie d'une porte de Muller afin de vérifier qu'il était possible de corrompre le compteur sans que la faute ne soit détectée.

Le chapitre 6 revient en détail sur l'exploitation d'une telle faille à des fins d'attaque cryptographique. Cette attaque a été réalisée en pratique sur les circuits DES avec une injection de faute par laser. Le chapitre 6 propose aussi une solution pour s'en prémunir.

4.3.9 Conclusion

Le modèle formel développé pour analyser le comportement des circuits asynchrones en présence de Soft Error nous permet de mettre en évidence des faiblesses dans les circuits. Ce type d'analyse prouve aussi grâce à sa complétude qu'il n'existe pas d'autres failles que celles constatées.

L'outil réalisé permet d'injecter une faute unique. Il est possible en poursuivant sa réalisation d'analyser le comportement des circuits en présence de fautes multiples. Plusieurs autres extensions peuvent être envisagées :

- Prise en compte d'autres protocoles de communication, pour comparer leur résistance face à l'injection de Soft Error. Les règles d'évolution ainsi que l'alphabet des jetons est entièrement dépendant du protocole de communication choisi.

- Insertion de délais de propagation pour déterminer quelle évolution du circuit est à privilégier. On peut ainsi prouver formellement qu'un circuit est tolérant ou résistant à l'injection d'une Soft Error sous certaines conditions de délais.

La principale limitation de cette approche est comme pour beaucoup de méthodes formelles ou semi formelles la rapide explosion de la complexité, en particulier si on souhaite injecter des fautes multiples. Cependant, il faut remarquer plusieurs aspects importants :

- L'analyse d'un circuit très complexe n'est pas obligatoire pour mettre en évidence ses faiblesses. Elles sont mises en évidence dès l'analyse de petits circuits.

- Un circuit peut être analysé incrémentalement module par module en simulant la réponse des entrées et sorties du module.

- La largeur de bus peut facilement être abstraite. Ainsi on va modéliser un Half Buffer de largeur quelconque en un seul étage, ce qui abstrait une grande partie de la complexité d'un circuit.

4.4 Schéma global d'analyse du comportement

La figure 4.15 propose un résumé des comportements d'un circuit QDI en présence de fautes (uniques ou multiples).

Cas 1 : L'impulsion transitoire est atténuée jusqu'à disparaître naturellement. Cette propriété n'est pas spécifique aux circuits asynchrones et elle dépend de la technologie et du dimensionnement des portes.

Cas 2 : La faute transitoire est filtrée par la logique (masquage). Elle n'atteint pas l'entrée de la partie mémoire.

Cas 3 : La faute transitoire est propagée jusqu'à l'entrée d'un élément mémoire, mais elle est filtrée par cet élément car il n'était pas dans un état sensible à cette faute.

Cas 4 : La faute transitoire est propagée jusqu'à l'entrée d'un élément mémoire, elle est mémorisée mais elle correspond à une mémorisation prématurée (invalidation de la sensibilité).

Cas 5 : La faute transitoire est propagée jusqu'à l'entrée d'un élément mémoire, elle est mémorisée mais il est possible de la détecter soit parce qu'elle a provoqué un blocage du circuit, soit parce qu'elle a généré un code faux.

Cas 6 : La faute est mémorisée en Soft Error, et elle n'est pas détectée. C'est le cas dangereux identifié dans la section précédente.

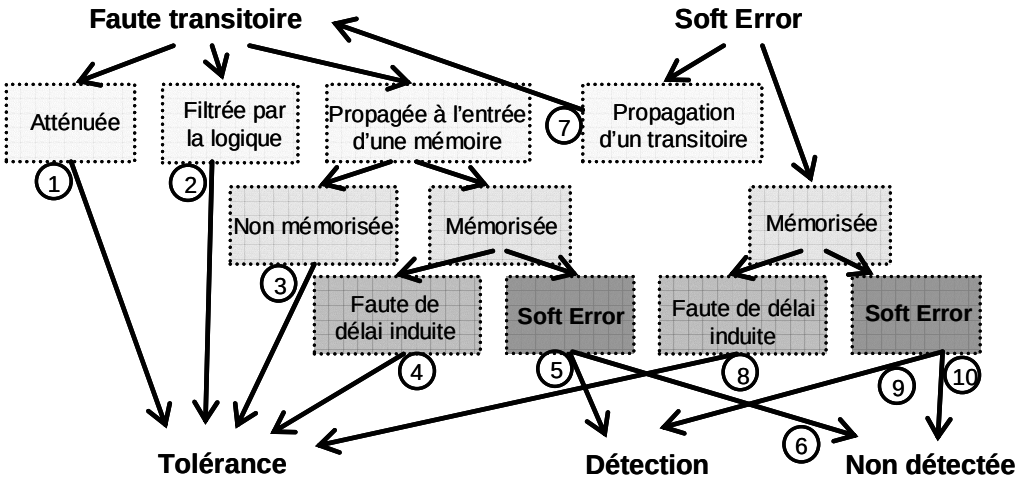


Figure 4.15 Résumé du comportement d'un circuit QDI en présence de fautes

Cas 7 : une Soft Error est mémorisée mais la porte dans laquelle elle a été injectée est dans un état Set ou Reset qui est « passant ». Il s'agit donc d'une faute transitoire propagée sur l'étage suivant.

Cas 8 : c'est un cas similaire au cas 4. Une Soft Error est mémorisée mais cela n'affecte pas le fonctionnement du circuit.

Cas 9 : une Soft Error est injectée mais elle est détectée car elle engendre un blocage du circuit ou un code faux. Cas similaire au cas 5.

Cas 10 : comme pour le cas 6, la Soft Error est mémorisée et elle n'est pas détectée.

Dans les cas 1, 2, 3, 4 et 8, le circuit est **tolérant** à l'injection de faute. Dans les cas 5 et 9, le circuit est **résistant**. Enfin, les cas 6 et 10 sont des **dangers** potentiels car un attaquant peut réussir à mettre le circuit en défaut en provoquant ces situations.

4.5 Conclusion

La première partie de ce chapitre analyse la sensibilité des circuits asynchrones aux fautes transitoires avec une approche analytique, alors que la deuxième partie applique une approche formelle pour étudier le comportement du circuit vis-à-vis des Soft Errors. Ces deux approches nous ont permis d'établir une classification des comportements des circuits asynchrones QDI en présence de fautes. A

partir de cette classification, nous proposons dans le chapitre suivant des méthodes de durcissement afin de minimiser la probabilité d'occurrence des cas dangereux et de favoriser les cas de détection ou de tolérance.

CHAPITRE V

PROPOSITIONS DE TECHNIQUES DE DURCISSEMENT DES CIRCUITS QDI CONTRE LES ATTAQUES PAR FAUTES

5.1 Introduction

L'analyse du comportement des circuits QDI en présence de faute montre que de nombreux cas sont favorables à la détection ou à la tolérance aux fautes. Cependant, il est nécessaire de protéger les circuits contre les faiblesses mises en évidence pour garantir un niveau de sécurité supérieur.

Comme nous l'avons vu dans le chapitre 2, Il existe de nombreuses contre-mesures adaptées aux circuits synchrones. L'enjeu de ces techniques est d'augmenter le niveau de sécurité du circuit tout en limitant le coût. En fonction des applications, on limitera le coût en surface, en vitesse ou en consommation. Pour évaluer une contre-mesure il faut donc prendre en considération son efficacité en terme de sécurité mais aussi son coût.

Un circuit asynchrone est par construction environ 2 fois plus gros qu'un circuit synchrone équivalent. Mais cette pénalité s'accompagne de propriétés intéressantes qu'il faut exploiter pour durcir les circuits contre les attaques (fautes, DPA , EMA …). Nous proposons dans ce chapitre des techniques de durcissement qui exploitent au maximum les propriétés intrinsèques des circuits QDI en pénalisant le moins possible les circuits en terme de vitesse, de consommation et de surface. Certaines de ces techniques sont implantées dans un circuit DES présenté dans le chapitre 6, afin d'évaluer en pratique leur efficacité sur un prototype.

5.2 Durcissement contre les fautes transitoires

5.2.1 Duplication des parties calculatoires

La figure 5.1 illustre la protection d'une partie calculatoire par duplication. Seule la partie calculatoire est dupliquée. Les deux résultats des blocs sont ensuite synchronisés par un rendez-vous dans la partie mémoire.

Cette méthode de durcissement est inspirée de la méthode de duplication proposée dans [Won05] pour les circuits asynchrones, et plus généralement de la duplication des blocs dans les circuits synchrones. Elle est cependant plus efficace qu'en synchrone. La duplication dans un circuit synchrone permet d'être résistant aux fautes uniques et aux fautes multiples injectées dans (1) l'un des deux blocs ou (2) dans les deux blocs mais avec des effets différents sur les sorties. Les fautes peuvent être détectées en comparant les résultats de sortie des deux blocs, mais il n'est pas possible de voter pour corriger l'erreur. Dans un circuit asynchrone, la duplication des parties calculatoires permet de filtrer une faute unique ou des fautes multiples injectées dans le même bloc (ou dans les deux blocs mais avec des effets différents). Le circuit est donc tolérant à ces fautes.

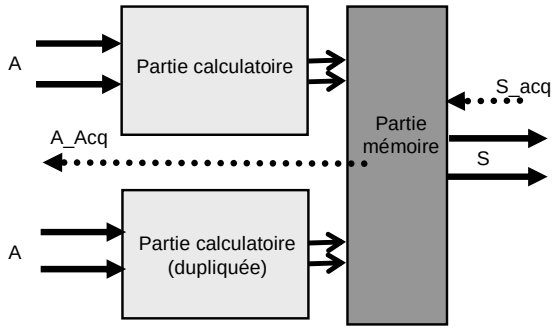


Figure 5.1 Durcissement par duplication de la partie calculatoire

La partie mémoire n'est pas dupliquée mais elle est durcie par l'ajout de nouveaux signaux de synchronisation. La figure 5.2 illustre une porte de Muller durcie. D1 est l'un des rails de sortie de la partie calculatoire, D2 est le même rail de sortie de la partie dupliquée. La sortie S ne peut être activée que si D1 et D2 sont actifs. Si une faute affecte l'un des deux signaux alors le rendez-vous ne peut pas se faire. Puisque les fautes injectées dans la partie calculatoire sont transitoires, le rendez-vous pourra s'opérer lorsque le système sera redevenu stable.

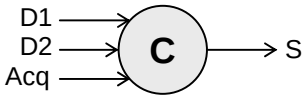


Figure 5.2 Porte de Muller durcie par la duplication de la partie calculatoire

Lorsque le circuit est inactif, aucune donnée n'est traitée par les parties calculatoires. Donc tous les rails d'entrée sont à zéro et l'acquittement de la sortie S est à 1. Dans un bloc non durci, les portes de Muller de la partie mémoire sont dans un état sensible : si une faute est injectée dans la partie calculatoire et propagée à l'entrée de la partie mémoire, celle-ci la mémorise. Dans le bloc durci, les portes de Muller sont dans un état sensible à 2 fautes (sur D1 et D2) mais pas à une faute unique. En effet, si une faute est injectée sur D1 elle ne peut pas être mémorisée car le bloc calculatoire de la partie dupliquée est inactif (D2 est à zéro). La faute est donc filtrée.

Cependant cette méthode est très coûteuse en terme de surface et de consommation. La partie calculatoire est dupliquée et le durcissement des mémoires augmente sensiblement leur surface. La technique présentée dans la prochaine section permet de mieux utiliser la redondance naturelle du circuit et de mieux exploiter les propriétés de synchronisation entre les éléments.

5.2.2 Technique de synchronisation des rails

5.2.2.1 Présentation générale

La technique de synchronisation des rails permet d'utiliser la redondance fonctionnelle du circuit pour améliorer la tolérance et la résistance du circuit. La figure 5.3a représente un canal C dont les digits C(0) et C(1) ont été synchronisés. Il s'agit d'une synchronisation d'événements (rendez-vous) et aucune hypothèse de synchronisation temporelle n'est faite.

La synchronisation garanti qu'une donnée présente dans C(0) ne peut être mémorisée que si une donnée est également présente dans C(1), et inversement. La synchronisation a lieu aussi pour les phases de remise à zéro. Comme pour la duplication, la synchronisation se fait au niveau des portes de Muller. La figure 5.3b présente une porte de Muller durcie par la synchronisation des rails. D1 est un rail de sortie de C(0), D2 est un signal qui indique la présence d'une donnée sur C(1). Acq(0) et Acq(1) sont les deux signaux d'acquittement des canaux.

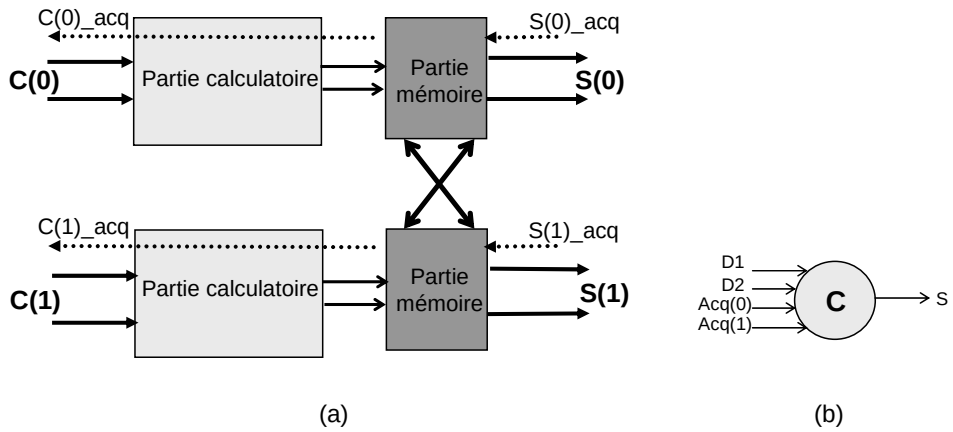


Figure 5.3 Durcissement par la méthode de synchronisation des rails

La figure 5.4 présente une implantation possible de ce Half Buffer durci. Une porte OR est utilisée pour générer le signal D2 qui s'assure de la présence d'une donnée sur l'autre canal. Sur la figure, seul le canal C(0) est représenté. Il est synchronisé avec C(1) par l'intermédiaire de D2, Acq(0) et Acq(1). On peut noter que si D2 indique la présence d'une donnée, il n'indique pas la valeur de cette donnée. En conséquence, il est possible qu'une faute se synchronise avec une donnée. Soit le scénario suivant :

Supposons que C(0) et C(1) soient inactifs. Injectons une faute dans C(0) qui crée artificiellement une donnée. La donnée reste bloquée sans être mémorisée car C(1) est inactif. Puis supposons que le canal C reçoive une donnée dans C(0) et C(1), mais que C(0) soit en retard par rapport à C(1). La faute se synchronise alors avec la donnée qui peuvent toutes deux être mémorisées et acquittées. Deux suites sont alors possibles :

- Soit la donnée de C(0) se propage et se combine avec la faute (génération d'un jeton F).

- Soit la faute est acquittée avant que la vraie donnée n'arrive sur C(0). Dans ce cas il y a un risque que la faute ait généré un jeton supplémentaire dans le circuit.

Ce problème est très similaire à la faille mise en évidence dans le chapitre précédent. Pour s'assurer que le cas ne se produit jamais, il faut prendre des hypothèses temporelles sur le délai entre les données C(0) et C(1). Dans la pratique, ce délai est toujours négligeable car les données d'un même canal sont traitées simultanément. Seul le routage peut engendrer une différence. Pour que le cas dangereux se produise il faut un délai d'un ordre de grandeur bien supérieur, équivalent à l'exécution des 4 phases du

protocole. Il s'agit donc d'un cas extrême. On peut noter que cette vérification de contrainte de délais peut être effectuée en simulation après placement/routage.

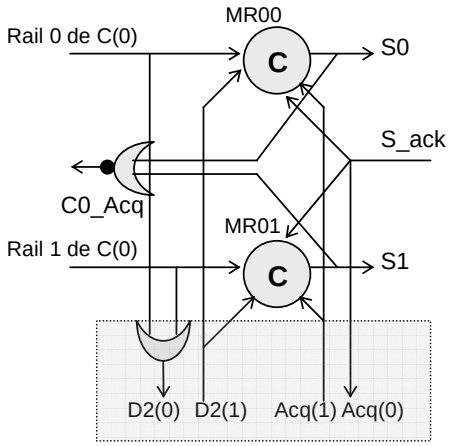


Figure 5.4 Exemple d'implantation de la synchronisation des rails (Half Buffer durci)

Cette technique améliore la tolérance aux fautes en filtrant les fautes transitoires lorsque le circuit est inactif, ou lorsque aucune donnée n'est momentanément présente dans le canal. Dans le cas contraire, le rendez-vous favorise la combinaison de la faute avec une donnée et donc force la génération d'un jeton F facilement détectable par une alarme. La résistance est donc elle aussi améliorée.

Pour durcir un canal de n bits, il suffit de synchroniser les digits par paires. On peut aussi synchroniser des blocs qui ne sont pas fonctionnellement identiques, mais il faut s'assurer qu'une donnée présente sur l'un des canaux implique la présence d'une donnée sur le deuxième, sinon il y a un risque de blocage du circuit.

Les coûts en vitesse et en consommation sont très faibles, voire négligeables à l'échelle d'un circuit. Le coût en surface est faible car aucune partie n'est dupliquée. Seuls les registres sont durcis et leur surface augmente par l'ajout d'entrées supplémentaires sur les portes de Muller et de portes OR. Une estimation du coût en surface est donnée sur un exemple concret dans le chapitre suivant. Cette technique a été implantée pour protéger plusieurs blocs calculatoires du circuit DES.

5.2.2.2 Exemple d'application et évaluation de la méthode

Nous présentons ici un exemple de mesure de sensibilité aux fautes pour évaluer l'efficacité de la méthode de synchronisation des rails selon la méthode d'analyse de sensibilité aux fautes transitoires

présentée dans la section 4.2. Nous comparons la sensibilité d'un bloc du DES, les S-Boxes, avant et après implantation de la technique de durcissement. L'analyse est effectuée sur des netlists du DES après placement routage et avec les informations de délais, ce qui permet d'analyser le circuit avec une très bonne précision.

Le tableau 5.1 présente les résultats obtenus sur le DES de référence (non durci). La simulation d'un cryptage simple dure 153 ns. La première colonne indique l'état atteint par la porte (p-sensible à m est présenté sous la forme "p à m") en précisant si cette sensibilité a été validée ou invalidée conformément a ce qui a été défini dans le chapitre précédent. La seconde colonne indique le temps total pendant lequel la porte est restée dans cet état et la troisième colonne indique le nombre d'occurrences. Enfin, les ports concernés par la sensibilité sont indiqués dans la dernière colonne.

Tableau 5.1 Analyse de sensibilité pour une porte de Muller du DES de référence

DES Référence (Muller2)			
Etat	**Temps total (ps)**	**Occ**	**ports**
Valide ‘1 à 0’	32008	9	A
Invalide ‘1 à 0’	6272	1	A
Invalide ‘1 à 0’	25120	8	B
Invalide ‘1 à 1’	20490	9	B
Reset	51330	16	
Set	17780	9	

Le DES effectue 16 rondes pour terminer son calcul. On observe dans le tableau que la porte atteint 16 fois l'état Reset (une fois à chaque ronde) et 9 fois l'état Set. Cela signifie que sur les 16 rondes du DES, la porte a été sélectionnée 9 fois pour mémoriser une donnée. Les données des 16 rondes restantes ont été mémorisées par la porte duale du code double rail. Ce chiffre est bien entendu dépendant des données du jeu de test et différent pour chaque porte.

On constate que cette porte est sensible à une faute unique sur son entrée A pendant 21% du temps de simulation (32008 ps). Le tableau 5.2 présente les résultats obtenus pour la même porte après durcissement des S-boxes. La porte est sensible pendant 12% du temps de simulation (23330 ps dans le tableau 5.2). La tolérance est améliorée par un facteur 1,7.

Ce gain est assez faible car cette partie du circuit est très active. Plus le circuit est actif, plus le gain en tolérance est faible mais plus la résistance est améliorée. Plus le circuit est inactif, plus le gain en tolérance est important car une faute injectée dans un canal n'aura pas de donnée avec laquelle se synchroniser et sera donc filtrée. Nous avons vérifié cette propriété en analysant la sensibilité d'un autre bloc du circuit beaucoup moins actif. Le bloc IP_1 n'est utilisé qu'à la fin du calcul lorsque les 16 rondes sont terminées et reste inactif le reste du temps. Une fois protégé, ce bloc est tolérant aux fautes uniques 99% du temps de simulation. Les 1% du temps restant correspondent à la phase active du bloc pendant laquelle une faute sera détectée par une alarme.

Tableau 5.2 Analyse de sensibilité pour une porte de Muller du DES durci

DES Durci (Muller4)			
Etat	**Temps total (ps)**	**Occ**	**ports**
Valide ‘1 à 0’	23330	7	D1
Valide ‘2 à 0’	20850	9	D2,D1
Valide ‘2 à 0’	3440	7	Ack2,D1
Valide ‘3 à 0’	2440	6	Ack1,D2,D1
Valide ‘3 à 0’	30	1	Ack2,D2,D1
Valide ‘3 à 0’	7550	7	Ack2,Ack1,D1
Invalide ‘1 à 0’	7690	9	D2
Invalide ‘1 à 1’	8370	9	D2
Invalide ‘2 à 0’	6030	9	D2,D1
Invalide ‘2 à 1’	50	1	Ack1, D2
Invalide ‘2 à 1’	3160	8	D2, D1
Invalide ‘3 à 0’	2950	9	Ack1,D2,D1
Invalide ‘3 à 1’	160	1	Ack2,D2,D1
Invalide ‘3 à 1’	4190	8	Ack1,D2,D1
Reset	53820	16	
Set	28340	9	

On constate que le reste de la sensibilité est reportée sur des fautes de multiplicité 2 et 3 (valide '2 à 0' et valide '3 à 0').

5.2.3 Technique de synchronisation avec un circuit de contrôle

Il n'est pas toujours possible d'appliquer la technique de synchronisation des rails, car certains canaux sont indépendants et ne peuvent pas trouver de partie avec laquelle se synchroniser. Ce peut être le cas des signaux de contrôle qui sont implantés sur 1 digit. Dans ce cas, il faut créer un circuit de contrôle

artificiel avec lequel le canal à protéger pourra se synchroniser. La figure 5.5 illustre une implantation de ce circuit de contrôle.

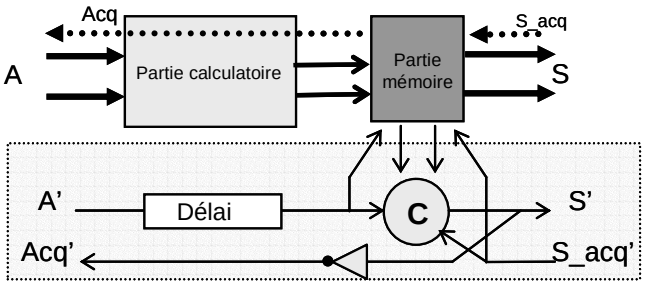


Figure 5.5 Synchronisation d'un canal avec un circuit de contrôle redondant

Le canal et le circuit de contrôle se synchronisent de la même façon que précédemment. Puisque le circuit de contrôle ne possède pas de partie calculatoire, il fonctionne beaucoup plus rapidement que le canal. Pour que la technique soit efficace, il est préférable d'ajouter un élément de délai qui « simule » la partie calculatoire. Ainsi, la fenêtre de sensibilité se réduit et on évite le cas critique précédemment décrit. Pour évaluer la valeur de ce délai, on pourra calculer le délai moyen de la partie calculatoire du canal.

Cette technique a la même efficacité que la synchronisation des rails. Elle est peu coûteuse en surface et la pénalité en vitesse est négligeable. Le coût en consommation est difficile à déterminer : si on souhaite par exemple protéger une machine à états, le coût en consommation par rapport à cette machine sera sans doute non négligeable; cependant le coût de la consommation d'une machine à états au sein d'un système est souvent négligeable.

5.3 Durcissement contre les Soft Errors

5.3.1 Méthode de détection par blocage du circuit

L'une des propriétés des circuits asynchrones QDI est de pouvoir se bloquer. Il y a plusieurs façons de bloquer un circuit :

- Disparition d'un jeton : la consommation d'un jeton due à l'injection d'une faute peut conduire un circuit à se bloquer. C'est le cas pour les circuits basés sur des structures itératives où le nombre de jetons est maîtrisé : la disparition d'un jeton entraîne le blocage d'un anneau puis du

circuit car un rendez-vous ne peut plus avoir lieu. Ce n'est en revanche pas obligatoirement le cas pour des structures linéaires où le nombre de jetons n'est pas connu.

- Génération d'un jeton : la génération d'un jeton peut conduire à un blocage du circuit dans des structures d'anneaux si le nombre d'étages est insuffisant pour contenir ce jeton supplémentaire, c'est-à-dire si la règle $N \geq 2K+1$ n'est plus respectée.

Le blocage d'un circuit est considéré comme une situation favorable car elle est détectable par un environnement extérieur. De plus, aucun résultat fauté n'est fourni à l'attaquant. Il est cependant souhaitable que l'environnement applique une stratégie de sécurité par exemple en fournissant à l'attaquant un résultat aléatoire. En effet, bloquer un circuit pour n'obtenir aucun résultat est un moyen de contrôler le circuit de façon déterministe. C'est un « pouvoir » qu'il est préférable de ne pas donner à l'attaquant.

5.3.2 Détection par alarmes

La détection de codes faux par l'intermédiaire d'alarmes apparaît naturel et indispensable pour protéger les circuits. Une alarme est une contre-mesure en soit, mais aussi un complément pour d'autres contre-mesures comme la synchronisation des rails.

Une alarme a pour rôle de détecter un code faux sur un canal double rail ou plus généralement sur un canal *1 parmi n*. Une fois qu'une alarme est levée, elle informe l'environnement qui doit alors appliquer une stratégie de sécurité : reset du circuit, sorties aléatoires. L'alarme doit rester active même si la faute est temporaire, c'est-à-dire si le code faux disparaît.

Les cellules d'alarmes utilisées dans le circuit DES (chapitre 6) sont implantées par des portes de Muller dont une des entrées est pilotée par le signal Reset. Une fois active, la sortie ne peut être désactivée que par un reset général du circuit.

5.3.3 Améliorer la détection grâce aux synchronisations

Les deux techniques de synchronisation pour durcir les circuits contre les fautes transitoires présentées dans ce chapitre ont pour effet d'améliorer la tolérance du circuit mais aussi de faciliter la détection des Soft Errors en évitant les cas extrêmes tel que celui présenté dans ce chapitre.

5.4 Conclusion

Les techniques présentées permettent de durcir les blocs et de les rendre résistants à l'injection de fautes uniques. La résistance ne peut pas être formellement prouvée sans prendre d'hypothèse sur les délais de propagation des données. En effet, les cas critiques présentés dans ce chapitre ainsi que la faille mise en évidence dans le chapitre précédent ne sont pas entièrement couverts par ces contre-mesures sans prendre d'hypothèses sur les délais.

Il faut noter que les contre-mesures présentées prennent peu en considération les modèles de fautes multiples, en particulier les Softs Errors multiples. Cet aspect sera abordé dans les perspectives de ce travail.

Le chapitre suivant présente en pratique les résultats obtenus par ces contre-mesures en comparant la résistance à l'injection de fautes par laser d'un circuit DES durci avec un DES de référence. L'apport en sécurité de ces contre-mesures y est démontré. Nous montrerons aussi que le cas critique obtenu à partir de la méthode formelle est un cas réaliste et qu'il est possible de l'exploiter si le module visé n'est pas protégé par une technique de synchronisation.

CHAPITRE VI

EVALUATION EXPERIMENTALE DE LA SECURITE DES CIRCUITS QDI ET DES CONTRE-MESURES PROPOSEES

6.1 Introduction

Les contre-mesures présentées dans le chapitre précédent ont été validées par simulation au travers des outils d'analyse. Peu de travaux de recherche vont plus loin en proposant de confronter la théorie et la pratique. La collaboration avec l'équipe sécurité des cartes à puces de Gemalto dans le cadre du projet Duracell nous a permis grâce à leur plateforme de test de mener des campagnes d'injections de fautes sur des prototypes de DES asynchrones et ainsi d'évaluer les circuits par la pratique .

Nous présentons dans un premier temps les deux circuits DES qui ont été testés. Le premier est un circuit de référence dont l'architecture globale était existante [Boue05a]. Ce circuit ne comporte aucune optimisation en surface, consommation ou vitesse, et ne comporte aucun élément dédié à améliorer la sécurité. Le second circuit est un circuit durci dans lequel plusieurs contre-mesures ont été implantées : synchronisation des rails, alarmes, et stratégie de blocage par contrôle du nombre d'étages. Ces contre-mesures ont été implantées sur des points spécifiques du circuit qui sont les points critiques en terme de sécurité. Les deux circuits comportent la même interface externe pour faciliter les tests sur la plateforme.

Les objectifs de ces évaluations sont multiples :

- Valider les contre-mesures et évaluer leur efficacité.

- Evaluer la résistance intrinsèque des circuits asynchrones aux injections de fautes. A notre connaissance, aucune campagne d'injection de fautes n'avait encore été effectuée en pratique sur ce type de circuit.

- Attaquer les circuits dans le but d'effectuer une cryptanalyse. Il faut pour cela mettre en défaut les circuits en obtenant des résultats fautés, et exploiter ces résultats mathématiquement pour trouver la clé.

- Observer et comprendre l'effet d'un tir laser sur le circuit. A partir d'un résultat fauté, on cherche à remonter le plus d'informations possible afin de connaître le nombre de fautes injectées, leur type (modèle de faute), et le lieu d'impact, et enfin la suite d'événements qui a conduit à ce résultat de façon la plus précise possible.

6.2 Les circuits cryptographiques DES : circuit de référence et circuit durci

6.2.1 Architecture globale

L'architecture globale des circuits a été définie lors de travaux précédents [Boue05a], ayant donné lieu à la fabrication de prototypes DES. Les deux circuits DES (référence et durci) implantés dans ce travail comportent la même interface globale. La seule différence se situe au niveau de la gestion du signal d'alarme, absent sur le DES de référence. Les circuits sont constitués de trois grands blocs : le banc de registres, les interfaces, et le bloc asynchrone de chiffrement/déchiffrement (figure 6.1)

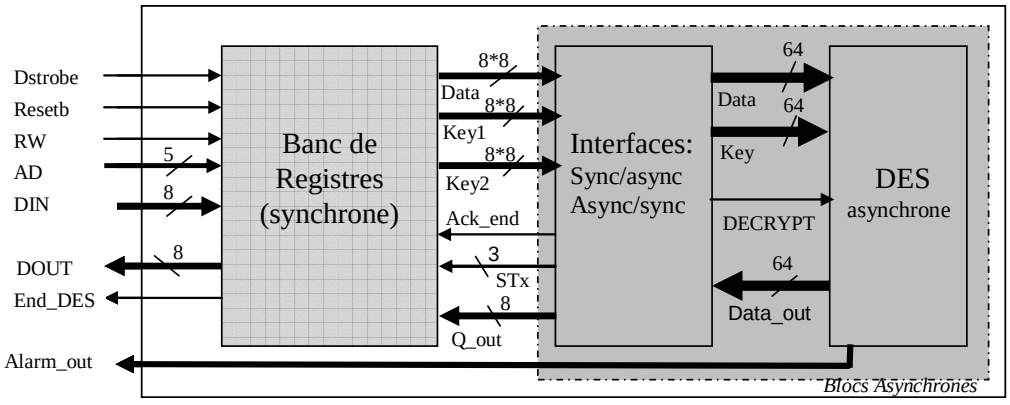


Figure 6.1 Architecture générale du circuit DES

Le banc de registres comprend les registres de mémorisation des données, des clés et du mode. Ces registres sont synchronisés en lecture/écriture sur front montant d'un signal d'horloge.

Le bloc d'interfaces permet de gérer les interfaces synchrones/asynchrones et asynchrones/synchrones et d'assurer le déroulement des opérations de simple et triple DES.

Le bloc DES asynchrone est le cœur DES dont les architectures durcie et non durcie seront détaillées dans cette section.

6.2.1.1 Les registres

Le banc de registres est composé d'un registre 5 bits (registre de mode) et de 32 registres 8 bits comprenant :

- 8 registres de 8 bits pour les données,
- 16 registres de 8 bits pour les deux clés nécessaires au calcul d'un triple DES,
- 8 registres de 8 bits pour les sorties.

Le registre de mode permet de configurer le mode opératoire du crypto-processeur (mode simple/triple DES, mode chiffrement/déchiffrement) et de déclencher un calcul. Il contient également une bascule dont la valeur permet d'indiquer la fin du processus de calcul. La valeur de cette bascule correspond au signal 'End_DES' en sortie du composant.

Le DES durci comprend deux registres de 8 bits supplémentaires pour les alarmes. Chaque bit des registres d'alarme correspond à une alarme locale à un module interne du cœur DES. Le tableau 6.1 présente la signification de chaque bit des registres. Seuls les deux premiers bits du registre Alarm2 sont utilisés. Le DES durci comporte aussi un signal d'alarme global en sortie qui indique qu'au moins une des alarmes locales à été déclenchée.

Tableau 6.1 Registres d'alarme du DES durci

Alarm1	Module concerné
bit0	XOR48
bit1	XOR32
bit2	IP_1
bit3	MUX_3L
bit4	MUX_3R
bit5	MUX_K
bit6	Shiftreg
bit7	DMux_k

Alarm2	Module concerné
bit0	Compteur (crypt/decrypt)
bit1	Compteur16
bit2	0
bit3	0
bit4	0
bit5	0
bit6	0
bit7	0

6.2.1.2 Les interfaces

Les registres sont synchrones afin de faciliter les communications et le contrôle avec un environnement externe (FPGA, Microcontrôleur). Il faut donc implanter des interfaces pour assurer la communication entre ces registres synchrones et le cœur de DES asynchrone. Elles sont présentées sur la figure 6.2.

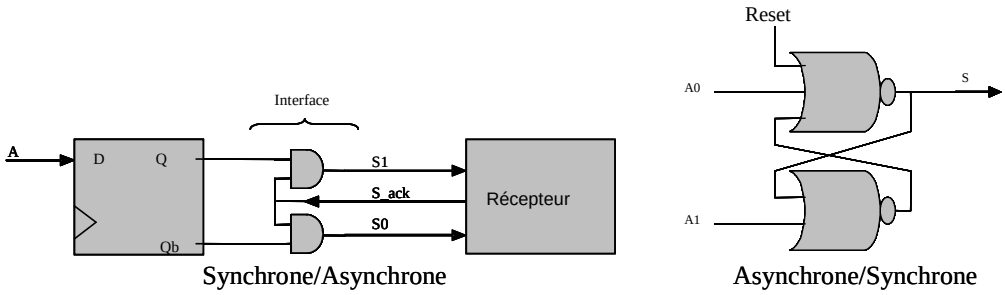


Figure 6.2 Interfaces sur 1 bit

L'interface synchrone /asynchrone est essentiellement constituée de portes ET. On utilise les sorties Q et Qb pour générer du double rail. Un signal d'acquittement venant du cœur du DES est utilisé pour assurer le déroulement du protocole 4 phases. Les interfaces asynchrone/synchrone sont composées de registres RS avec un signal d'initialisation.

6.2.2 Architecture du circuit de référence

La figure 6.3 présente l'architecture du DES de référence. C'est une structure itérative basée sur 3 boucles de calcul : la boucle de gestion des sous-clés (a), la boucle de chiffrement/déchiffrement (b), et un compteur qui assure le contrôle (c). Ces 3 blocs prennent respectivement en entrée la donnée à

chiffer/déchiffrer, la clé et le signal de mode déterminant le type de calcul à effectuer (chiffrement ou déchiffrement).

Les blocs représentés en gris sur la figure 6.3 sont des blocs contenant une structure de Half-Buffer qui implante le protocole de communication et éventuellement une partie calculatoire. Les blocs représentés en blanc sont purement calculatoires.

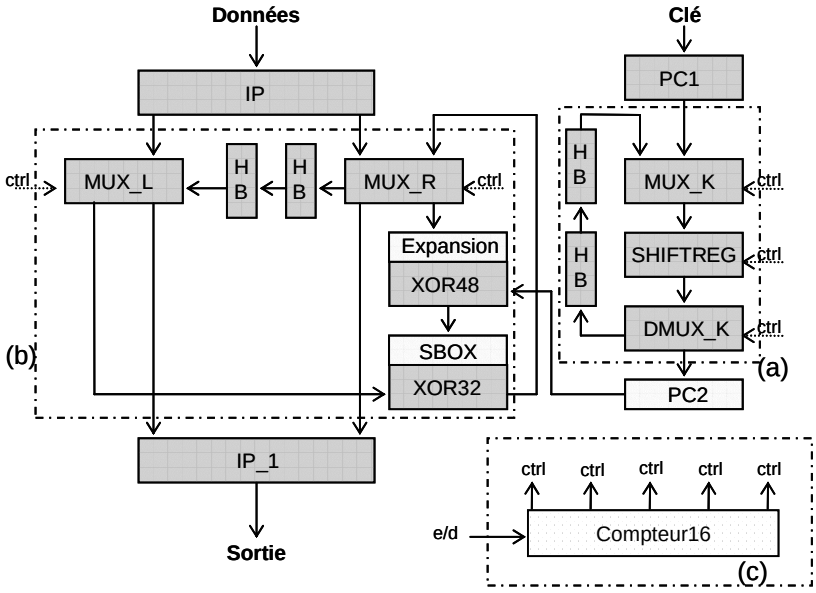


Figure 6.3 Architecture du DES de référence

6.2.2.1 Le bloc de gestion des clés

Ce module est un anneau composé de 5 étages. Il est chargé de calculer à chaque ronde une sous-clé de 48 bits à partir de la clé initiale de 56 bits. Chaque sous-clé est calculée en fonction de la précédente par un décalage à gauche ou à droite, en fonction du mode de calcul. La valeur du décalage est déterminée par un signal de contrôle généré par le compteur.

6.2.2.2 Le bloc de chiffrement/déchiffrement

Ce module est chargé de combiner les données avec les sous-clés à chaque ronde. Il contient des blocs calculatoires comme les opérateurs XOR et les S-Boxes. Deux multiplexeurs (MUX_L et MUX_R) commandés par des signaux de contrôle générés par le compteur dirigent les données dans la boucle. A

chaque ronde, les données sont rebouclées pour effectuer la prochaine ronde, ou consommées et dirigées vers la sortie du cœur si il s'agit de la dernière ronde.

6.2.2.3 Le compteur

Le compteur est une machine à états qui génère un ensemble de signaux de contrôle qui pilotent les autres modules. Le compteur de ronde est implanté en utilisant un **code 1 parmi 17** (un fil pour chaque ronde plus le fil de sortie). Nous revenons plus en détail sur son architecture dans la section 6.5.2 qui décrit une procédure d'attaque sur ce module.

6.2.3 Architecture du circuit durci

La figure 6.4 présente l'architecture du circuit DES durci. La figure indique les emplacements où ont été implantées les contre mesures. L'architecture générale est très similaire à celle du circuit de référence.

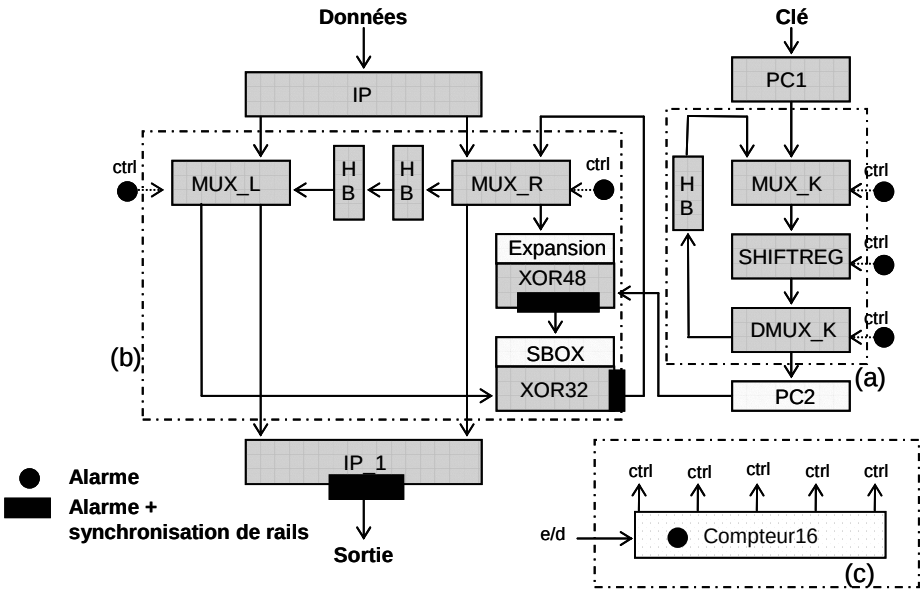


Figure 6.4 Architecture du DES durci

- Les blocs calculatoires tels que les opérateurs XOR et les S-Boxes ont été protégés par la technique de synchronisation des rails. Chaque digit a été synchronisé avec un autre digit. Des alarmes ont été implantées à la sortie des Half-Buffers pour compléter la protection.

- Le bloc IP_1 a également été protégé car il est très sensible aux fautes tout au long du calcul, comme cela a été indiqué dans la section 5.2.2.2.

- Des alarmes ont été implantées sur chaque signal de contrôle généré par le compteur. Si un de ces signaux est corrompu, les multiplexeurs peuvent se comporter de façon dangereuse, comme nous le verrons dans la section 6.5.2.

- Le compteur est également protégé par une alarme. Cette alarme permet de détecter tout code faux sur le codage **1 parmi 17**, c'est-à-dire tout code de la forme **p parmi 17**, avec $1<p\leq17$.

- Un Half-Buffer a été supprimé dans le bloc de calcul des sous-clés. Ce module était initialement composé de 5 étages et il contient 1 donnée (la sous-clé). Selon la règle évoquée dans la section 5.3.1, un anneau composé de 5 étages peut contenir jusqu'à 2 données. Dans une structure à 5 étages comportant une seule donnée, il y a donc un risque qu'une faute génère une donnée sans que cela ne provoque de blocage. C'est pourquoi nous supprimons un étage pour évaluer l'impact de la nouvelle structure à 4 étages sur le nombre de blocages constatés lors d'une campagne d'injection de faute.

6.2.4 Caractéristiques des circuits

Le placement routage des deux circuits DES a été contraint pour faciliter les attaques laser et l'interprétation des résultats. Ainsi, les interfaces synchrones ont été distribuées sur la périphérie du circuit car on ne cherche pas à les caractériser. En revanche, chaque module asynchrone a été placé et délimité avec l'aide des ingénieurs du groupe CIS, et en particulier de Sophie Dumont. Le placement contraint facilite la caractérisation d'une contre-mesure car il est possible de cibler un bloc particulier sans risquer d'effets de bords ou de phénomènes parasites trop importants. La Figure 6.5 présente le plan de masse du circuit durci en exemple.

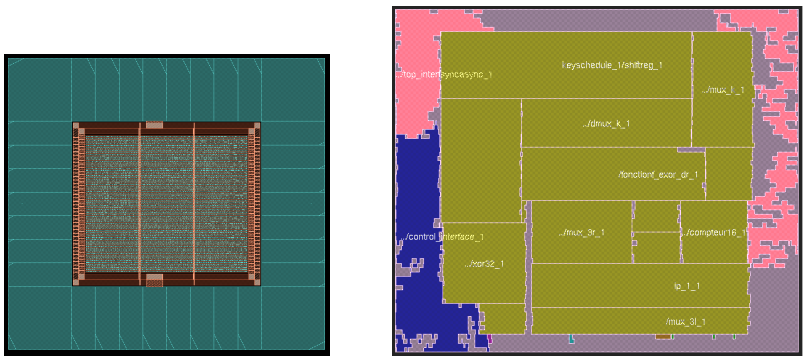


Figure 6.5 Circuit durci - (a) masques (b) plan de masse

Les circuits ont été fabriqués en technologie **CMOS 0,13 µm de STMicroelectronics** via le CMP. Certaines cellules ont été implantées en utilisant la bibliothèque asynchrone TAL compatible avec la bibliothèque standard de STMicroelectronics. Cette bibliothèque propose certaines fonctions spécifiques aux circuits asynchrones, telles que les portes de Muller [Raz04].

Le tableau 6.2 résume les caractéristiques des circuits. Le circuit de référence est environ 5 fois plus gros qu'un circuit synchrone (non durci) équivalent. Le facteur peut être largement réduit en utilisant des architectures moins bufferisées et des cellules complexes dédiées [Raz04] et en ne contraignant pas le placement/routage. Les dernières études montrent qu'il est possible de réduire le coût en surface et obtenir un facteur 2 par rapport au synchrone.

Le circuit de référence exécute le calcul d'un simple DES en 165 ns, ce qui est environ 10 fois plus rapide qu'une architecture synchrone standard.

Tableau 6.2 Caractéristiques des circuits DES

Circuit	Surface/ portes équivalentes	Temps de calcul. Simple DES (1,2 V)	Temps de calcul. Triple DES (1,2 V)
Référence	0,156 mm² / 20,3 Kgates	165 ns	485 ns
Durci	0,168 mm² / 22,3 Kgates	201 ns	592 ns

Le coût en surface pour sécuriser le circuit est de 7,7%, ce qui est très faible. L'implantation de la technique de synchronisation de rails et l'implantation des alarmes augmentent localement la surface, tandis que la suppression d'un étage dans le bloc de gestion des sous-clés permet de réduire le coût.

La vitesse est réduite de 18% sur le circuit durci. Il y a deux facteurs qui expliquent ce coût : la suppression d'un étage de pipeline dans le bloc de gestion des clés réduit la vitesse du bloc; la technique de synchronisation des rails accroît de plus légèrement le nombre de transitions nécessaires pour mémoriser les données à cause de la porte OR et des portes de Muller à 4 entrées. On peut aussi noter que les portes de Muller à 4 entrées n'étaient pas implantées en utilisant des cellules spécifiques. Ces portes sont donc particulièrement coûteuses en surface et en vitesse dans cette version du DES.

6.3 Mise en œuvre de la plateforme laser

6.3.1 Description du dispositif de test

La plateforme de Gemalto est constituée d'un PC, un oscilloscope, une table XY pour cibler un emplacement précis sur le circuit, et un laser. Le PC commande la table XY, l'armement du laser, et récupère les acquisitions de l'oscilloscope.

Afin de procéder aux tests sur la puce, nous disposons d’une carte XilinX - conception réalisée sur PC (Windows 2000) sous environnement ISE version 7.1- sur laquelle est montée la puce (figure 6.6). Nous disposons d’une connexion série pour communiquer avec la plateforme de test ainsi que d’un analyseur logique afin d’observer les signaux générés lors de la communication entre le PC, le FPGA et le circuit DES. Tout le développement de la carte de test ainsi que le développement des outils (logiciels et matériels) de tests ont été réalisés par nos partenaires au sein du laboratoire de Gemalto.

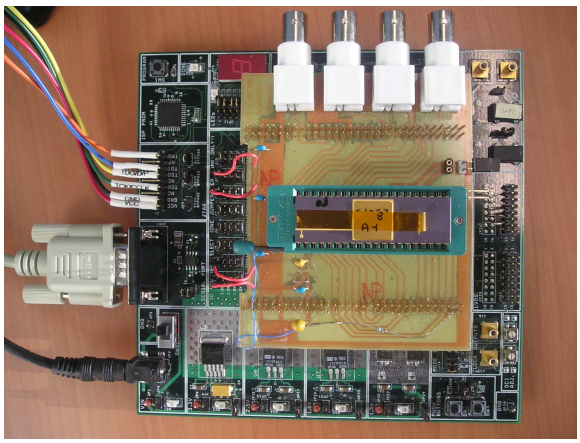


Figure 6.6 Carte de test du circuit DES

Le laser en lui même est un laser YAG vert destiné initialement à la découpe des pistes métalliques dans les circuits intégrés. La longueur d'onde est de 532 nm et l'énergie est réglable de 0 à 100%. On peut également régler l'ouverture pour cibler une zone plus ou moins importante du circuit. La figure 6.7 illustre la plateforme laser.

Figure 6.7 Plateforme laser de Gemalto

6.3.2 Localisation temporelle

La première étape pour réaliser une campagne d'injection de fautes est de déterminer le bon moment pour injecter une faute dans le circuit afin de le mettre en défaut. L'objectif est de réaliser l'injection de fautes pendant le calcul du DES. La plateforme FPGA qui commande le déclenchement du laser fonctionne avec une fréquence de 100 Mz, ce qui nous permet d'avoir une précision temporelle pour les

tirs de 10 ns. Puisque le DES exécute un chiffrement simple en 200 ns (version durcie), nous avons environ 20 positions temporelles sur lesquelles il est possible de procéder à une injection de faute. Pour chaque coordonnée spatiale sur le circuit, nous avons donc la possibilité d'effectuer 20 tirs différents.

Avec cette granularité, nous avons la possibilité d'effectuer au moins une injection de faute dans chaque ronde du DES.

6.3.3 Localisation spatiale

La seconde étape est de déterminer les positions spatiales sur lesquelles effectuer l'injection de fautes. Deux approches sont possibles: l'approche "boite noire" et l'approche "boite blanche".

L'approche en boite noire consiste à considérer que l'attaquant n'a aucune connaissance sur les masques (layout) du circuit et qu'il doit donc scanner toute la surface du circuit pour déterminer, en étudiant les résultats bruts, quelles sont les parties du circuit sensibles ou intéressantes avant de se concentrer sur elles. L'inconvénient de cette approche est qu'elle demande de nombreuses campagnes d'injection et beaucoup de temps pour interpréter les résultats.

L'approche en boite blanche consiste à se concentrer sur les blocs sensibles en connaissant leur localisation spatiale (coordonnées XY). Cette approche est la plus intéressante pour nous afin de caractériser au mieux les contre-mesures. Après avoir essayé plusieurs configurations, la taille choisie pour la fenêtre du faisceau est de 220 μm^2. Pour chaque campagne d'injection de fautes on procèdera donc à un balayage spatial du bloc testé avec cette ouverture.

6.3.4 Niveau d'énergie

Un test effectué avec une trop grande énergie pour une taille de fenêtre donnée du faisceau laser peut irrémédiablement détériorer la puce. Il faut donc adapter l'énergie du laser en fonction de la surface de la fenêtre et de la technologie visée. L'objectif est de produire un effet sur la puce, c'est-à-dire apporter suffisamment d'énergie pour générer des fautes, mais sans la détériorer. Sur le circuit DES, une densité d'énergie de 0,8 $pJ/\mu m^2$ a été utilisée pour générer les fautes.

6.3.5 Durée du tir

La durée de la pulsation laser est de 6 ns (durée fixe).

6.3.6 Campagnes d'injection de fautes

La figure 6.8 présente la séquence de test qui a été réalisée pour chaque campagne d'injection de fautes. La campagne consiste à quadriller temporellement (T) et spatialement (X, Y) le bloc à tester. Pour chaque position (X, Y, T) la séquence suivante est effectuée :

- Reset du circuit afin d'éviter des éventuelles fautes latentes du tir précédent.
- Chargement des clés et des données. On choisira des valeurs aléatoires ou fixées en fonction de l'objectif de la campagne.
- Exécution du DES (chiffrement simple) et tir du laser à la position (X, Y, T)
- Si une erreur est constatée, on enregistre les informations dans un fichier de log.
- Il est possible de répéter N fois le même tir à la position (X, Y, T) de façon à mesurer la reproductibilité d'une faute, c'est-à-dire à déterminer si le même tir produit toujours le même effet sur le circuit.
- On procède au tir suivant.

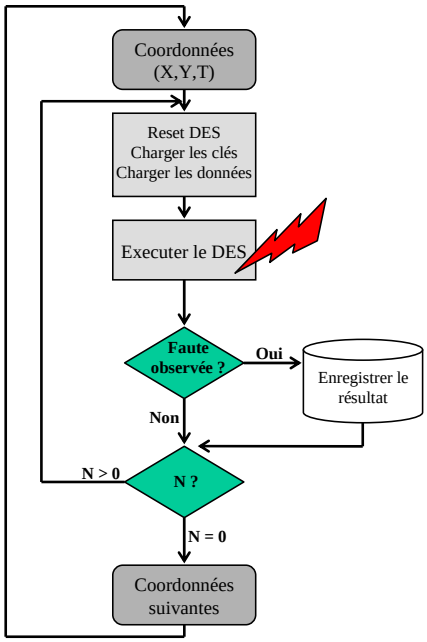


Figure 6.8 Séquence de test

Les résultats sont classés de la façon suivante :

- **Résultat correct** : le calcul du DES est correct, le signal de fin de calcul est observé et aucune alarme n'est déclenchée. L'injection de faute n'a pas eu d'effet sur le fonctionnement du circuit.

- **Erreur détectée** : Le calcul du DES est incorrect et il est possible de le détecter. Il y a deux façon de détecter une erreur de calcul : une alarme s'est déclenchée (DES durci) ou un blocage est observé. Dans le cas du blocage, le signal de fin de calcul n'est jamais observé.

- **Erreur non détectée** : le calcul du DES est incorrect et il n'y a aucun moyen de le détecter. Cela signifie qu'aucune alarme ne s'est déclenchée et que le signal de fin de calcul est observé normalement, mais que le résultat obtenu est différent de celui attendu.

Les résultats obtenus sont enregistrés avec le format suivant :

NbCmd	Fauté	Attendu	Position	Délai	Clé	Donnée	Alarme	End
0434	0000000000000000	D1AB817CB041A016	277.5 347.3	17	1122334455667788	9A11BAC21490B2B8	0000	04

NbCmd indique le numéro du tir dans la campagne courante. Le **fauté** est le résultat du calcul obtenu, différent du résultat de calcul qui était **attendu**. La **position** indique les coordonnées spatiales du tir et le **délai** indique la tranche temporelle dans laquelle le tir a été effectué. On enregistre aussi la clé et la donnée utilisées pour le chiffrement. Les registres d'alarmes **[Alarm2:Alarm1]** sont lus juste après le résultat du calcul pour déterminer quelles alarmes ont été déclenchées. Enfin la valeur du signal de fin de calcul 'End_DES' est indiquée.

La valeur '04' du signal 'End_DES' signifie qu'il y a eu un blocage du circuit car le signal de fin de calcul n'a pas été généré. La valeur '00' signifie que le calcul est terminé. Ce signal est généré lorsque tous les bits du Half-Buffer de sortie ont reçu une valeur valide sur le double rail. Si le circuit se bloque pendant le calcul, le Buffer ne reçoit pas les données et le signal de fin de calcul n'est pas généré.

Dans l'exemple ci-dessus, le tir a provoqué un blocage complet du circuit. Aucun résultat n'a été mémorisé par les registres de sortie, qui conservent donc leur valeur initiale (zéro).

6.4 Résultats expérimentaux : évaluation des contre-mesures

6.4.1 Technique de synchronisation des rails

Pour valider l'efficacité de cette technique, nous avons choisi d'effectuer une campagne d'injection de fautes sur le module des S-Boxes qui est protégé en sortie par cette méthode. Le module durci (S-Boxes + registres durcis) est 8,8% plus gros que le module de référence (S-Boxes + registres standards). Le coût est donc très faible.

La campagne consiste à scanner temporellement et spatialement le bloc calculatoire, ce qui représente environ 5600 tirs sur chaque circuit. L'objectif est de comparer le nombre de résultats fautés enregistrés (tous résultats fautés confondus) et de conclure de façon statistique sur la tolérance des blocs. Les blocs analysés sont de taille et de densité comparables, ce qui rend la comparaison pertinente.

6.4.1.1 Résultats: gain en tolérance

Le tableau 6.3 présente le nombre de résultats fautés obtenus, tout type confondu (détectés et non détectés). Sur le DES de référence, 955 fautés ont été obtenus (17% des tirs). On enregistre seulement 377 fautés sur le DES durci (7% des tirs), soit un gain de 2,5 de tolérance aux fautes.

Tableau 6.3 Résultats de tolérance aux fautes

Bloc (S-Boxes)	Nombre de fautés sur 5600 tirs	Gain
DES référence	**955**	
DES durci	**377**	**2,5**

On peut remarquer que le nombre d'erreurs observées par rapport au nombre de tirs est relativement faible. 83% des tirs n'ont pas d'effet observable sur le circuit de référence, et 93% sur le circuit durci.

En conclusion, les résultats expérimentaux nous ont permis de vérifier les résultats théoriques. La technique de synchronisation des rails permet d'améliorer sensiblement la tolérance aux fautes par un facteur 2,5.

6.4.1.2 Résultats : gain en résistance

Le tableau 6.4 présente une classification des résultats fautés obtenus, en distinguant les erreurs détectées et les erreurs non détectées. Sur le DES de référence, 67% des erreurs sont détectées. Elles correspondent aux blocages du circuit. 313 cas ne sont pas détectés : le circuit fournit un résultat faux et le calcul se termine normalement.

Sur le DES durci, toutes les erreurs sont détectées, soit par blocage (82 cas), soit par l'intermédiaire des alarmes (307 cas), soit par les deux. Lorsque les fautes ne peuvent pas être filtrées, elles génèrent un code faux qui est détecté.

Tableau 6.4 Résultats de résistance aux fautes

Bloc (S-Boxes)	Erreurs détectées	Erreurs non détectées
DES Référence	642 (67%)	313 (33%)
DES Durci	377 (100%)	0 (0%)

6.4.2 Alarmes

La caractérisation des alarmes implantées dans le DES asynchrone durci s'est déroulée implicitement au travers des campagnes de tirs effectuées. Nous avons pu :

- Valider leur implantation et leur fonctionnalité grâce à des fautés ayant déclenché des alarmes.

- Localiser leur origine et leur propagation grâce aux registres d'alarmes.

On remarque que dans la plupart des cas l'alarme se propage à plusieurs blocs et un code faux ne conduit pas systématiquement à un blocage du circuit. En effet, les structures de calcul implantées dans le DES favorisent la propagation de codes faux en sortie (le code faux étant un élément absorbant pour les structures de multiplexeurs, démultiplexeurs et opérateurs XOR présents dans le DES).

Lorsque le tir est effectué dans les **S-Boxes**, on constate que l'alarme se propage uniquement dans les blocs de calcul des données : XOR32, XOR48 et le bloc de sortie IP_1, soit un registre d'alarme valant 0007.

En revanche lorsque le **compteur16** est attaqué les signaux d'alarme sont plus nombreux et plus hétérogènes. En effet, le compteur génère tous les signaux de contrôle des autres blocs du DES. Lorsque le bloc de logique générant ces signaux est attaqué, les signaux de contrôle peuvent être corrompus et les alarmes sont propagées jusqu'aux blocs concernés. Ainsi, les valeurs 02DF et 02FF des registres d'alarme sont fréquentes lorsqu'une faute est injectée dans le compteur. La valeur 02FF signifie que 9 alarmes sur 10 ont été déclenchées.

On peut noter que nous n'avons constaté aucun cas de déclenchement d'alarme avec un résultat juste.

6.4.3 Blocage du circuit

Le blocage est un comportement spécifique aux circuits asynchrones. Il a pu être vérifié au cours des campagnes de tirs effectuées. Il se caractérise par l'absence de signal de fin de calcul. On distingue deux types de blocages :

- Blocage complet, pour lequel aucun bit de sortie n'est observé. C'est le cas de l'exemple présenté dans la section 6.3.5.

- Blocage partiel : certains bits (éventuellement faux) sont propagés jusqu'aux registres de sortie alors que d'autres sont bloqués. Dans ce cas la situation de blocage est détectée car le signal de fin de calcul n'est émis que lorsque tous les bits du résultat sont disponibles. Dans l'exemple suivant, certains bits des registres de sortie sont restés à zéro :

NbCmd	Fauté	Attendu	Position	Délai	Clé	Donnée	Alarme	End
0419	2CD0C32A74D00341	C48FA5D5B3FD63A8	277.5	347.3	10	1122334455667788	862709C65488BF54	0000	04

Une campagne de test a été effectuée sur le module de gestion des clés pour mesurer l'influence du nombre d'étages dans un anneau sur le comportement du circuit en présence de fautes. On rappelle que le module du DES de référence est composé de 5 étages, et celui du DES durci est composé de 4 étages. Une seule donnée circule dans cet anneau.

Le bloc attaqué est l'un des Half Buffers (HB) 56 bits du module de génération des sous-clés, que l'on scanne temporellement et spatialement (1600 tirs). Les résultats obtenus sont indiqués dans le tableau 6.5.

Tableau 6.5 Blocages constatés dans le module de gestion des sous-clés et sensibilité des portes de Muller

Bloc	% de blocage	sensibilité mesurée
HB DES référence	45%	48 ns/porte
HB DES durci	20%	80 ns/porte

On constate qu'il y a deux fois moins de blocages dans le DES durci que dans le DES de référence. Ce résultat signifie que réduire le nombre d'étages d'un anneau ne suffit pas pour provoquer plus de blocages en cas d'injection de fautes. Le phénomène inverse est même constaté : la réduction du nombre de Buffers dans le module de génération des sous-clés du DES durci a déséquilibré l'architecture et les temps de sensibilité aux fautes des portes de Muller ont été modifiés. Les fenêtres de sensibilité ont été agrandies, rendant le DES plus sensible à la mémorisation d'une faute.

Nous avons vérifié cette hypothèse en mesurant cette sensibilité en simulation rétro-annotée avec l'outil d'analyse de sensibilité. Sur le DES durci, chaque porte du Buffer est sensible en moyenne pendant 80ns sur la durée du calcul, alors que cette sensibilité n'est que de 48ns sur le DES non durci (tableau 6.5).

La conclusion de cette campagne est que la gestion du nombre d'étages dans une boucle n'est pas suffisante pour favoriser une stratégie de blocage de cette boucle. En revanche c'est la gestion du temps de sensibilité des portes qui va favoriser un comportement particulier, c'est à dire le choix parmi plusieurs comportements possibles dans l'arbre d'états atteignables du modèle formel développé dans le chapitre V.

6.5 Résultats expérimentaux : évaluation de sécurité des circuits QDI

Le second objectif des campagnes d'injection de fautes est de mettre les circuits en défaut afin d'obtenir des résultats fautés exploitables, pour effectuer une cryptanalyse. Deux modules sont ciblés : les S-Boxes et le compteur du DES.

6.5.1 Attaque sur les S-Boxes

Les campagnes d'injection de fautes sur les S-Boxes ont permis à la fois de caractériser les contre-mesures implantées et aussi de fournir des résultats fautés à analyser pour effectuer une cryptanalyse. Ces analyses ont été effectuées par nos partenaires en utilisant l'outil FADADES.

FADADES est un outil développé par Gemalto qui permet étant donnés un message et une clef donnée, de retrouver toutes les explications possibles de fautes dans l'exécution de l'algorithme DES sous l'hypothèse d'un octet modifié. On peut alors sélectionner certains fautés exploitables pour remonter à la clé [Biha91b].

Les analyses effectuées sur les fautés non détectés du DES de référence montrent qu'aucun de ces fautés n'est suffisamment exploitable pour remonter à la clé. Il semble donc très difficile de réussir une attaque de ce type, même sur le DES de référence. Le DES durci ne fournit aucun fauté non détecté, il n'est pas possible de réaliser l'attaque sur les S-Boxes à partir des résultats expérimentaux obtenus.

6.5.2 Attaque sur le compteur

L'attaque consiste à corrompre le compteur du circuit DES pour réduire le nombre de rondes effectuées. On obtient ainsi le résultat du calcul d'un DES non complet, ayant effectué un nombre de rondes inférieur à 16. Si le nombre de rondes est insuffisant il devient possible d'effectuer une cryptanalyse différentielle linéaire [Hell94].

Cette attaque avait été effectuée auparavant en pratique sur une implantation non protégée du compteur d'un circuit synchrone [Chou05]. A notre connaissance, c'est la première fois qu'une attaque est réalisée sur une machine à états multi rail asynchrone et protégée.

6.5.2.1 Description du compteur

Le compteur est une machine à états à trois étages chargée de compter le nombre de rondes (Figure 6.9). Pour chaque ronde, il génère un ensemble de signaux de contrôle qui pilotent le DES, comme par exemple le calcul des sous-clés et les multiplexeurs/démultiplexeurs du module de chiffrement. On utilise un codage *1 parmi 17* pour compter les rondes. Chaque ronde est codée par un rail, et le dernier rail est utilisé pour sortir le résultat.

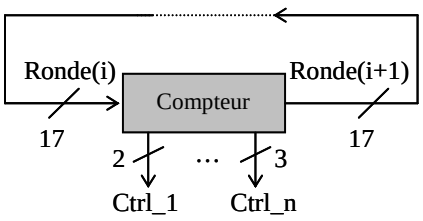
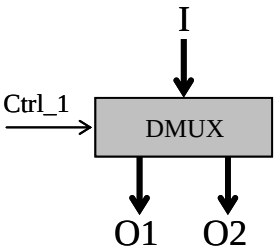


Figure 6.9 Compteur du DES

La figure 6.10 montre un exemple d'utilisation d'un signal de contrôle sur un démultiplexeur. Lorsque Ctrl_1 vaut "01", la sortie O1 est sélectionnée. Lorsque Ctrl_1 vaut "10", la sortie O2 est sélectionnée. Si Ctrl_1 est fauté alors les deux sorties sont sélectionnées. Supposons que Ctrl_1 commande l'envoie des données sur les registres de sortie à la fin du calcul. O1 est utilisée pour reboucler les données au cours du calcul, et O2 est utilisée pour sortir les données à la fin de la dernière ronde. La conséquence d'une faute sur ce signal de contrôle peut être critique car le code faux permet de sortir des résultats et de générer un signal de fin de calcul prématurément. Les valeurs de sortie sont des valeurs des calculs intermédiaires qu'il est dangereux d'exposer comme nous le montrons dans la prochaine section.



Ctrl_1	Sortie Activée
“01”	O1
“10”	O2
“11 “(code faux)	O1 et O2

Figure 6.10 Comportement d'un démultiplexeur en fonctionnement normal ou en présence d'une faute

6.5.2.2 Description de l'attaque

La campagne d'injection de fautes effectuée sur le compteur regroupe plus de 5000 tirs sur chacun des circuits. On rappelle que le compteur du DES de référence est non protégé, et le compteur du DES durci est protégé par une alarme sur le code multi rail *1 parmi 17*, et des alarmes sont implantées sur tous les signaux de contrôle générés.

Sur chacun des circuits, environ 40% des tirs aboutissent à une erreur observable. Certains résultats fautés ont été identifiés comme étant le résultat d'une modification de la séquence et du nombre de rondes effectuées. La prochaine sous-section présente l'analyse effectuée, puis nous décrivons les attaques réalisées sur le circuit de référence et le circuit durci.

6.5.2.3 Modification du nombre de rondes

Plusieurs résultats fautés correspondent au calcul d'un DES dont la séquence des rondes a été modifiée. La séquence correcte des rondes est notée de la façon suivante:

$$T_c = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17)$$

T_c représente le calcul des 16 rondes plus l'opération de sortie du résultat final, notée "17". L'expression [i → j] représente une faute qui a été injectée dans la **i**$^{\text{ème}}$ ronde et qui a corrompu le compteur vers la valeur **j**. La **i**$^{\text{ème}}$ ronde n'est pas exécutée et le calcul se poursuit en exécutant la **j**$^{\text{ème}}$ ronde. Par exemple, l'expression [7 → 10] représente la séquence suivante:

$$T_1 = (1,2,3,4,5,6,10,11,12,13,14,15,16,17)$$

La ronde 7 n'est pas exécutée. Le calcul saute de la ronde 6 à la ronde 10. On peut noter que des sauts de différentes natures ont été observés sur les deux circuits : des sauts en avant (i > j) et des sauts en arrière (j > i) pour lesquels le circuit a effectué plus de 16 rondes.

Comme spécifié par l'algorithme DES, une sous-clé est calculée à chaque ronde. Puisque le compteur contrôle le module de chiffrement mais aussi le module de gestion des clés, la corruption du compteur entraîne la corruption du calcul des sous-clés :

$$S_c = (1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1)$$
$$S_1 = (1,1,2,2,2,2,2,2,2,2,2,2,2,1)$$

S_c représente la séquence de décalages effectuée pour calculer chaque sous-clé lors d'un chiffrement. S_1 est la séquence de décalage corrompue par la faute [7 → 10]. On note que plusieurs séquences T différentes peuvent correspondre à la même séquence S.

A partir d'une séquence S_i, on dérive la séquence des décalages accumulés σ_i depuis le début du calcul :

$$\sigma_c = (1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,29,31,32)$$
$$\sigma_1 = (1,2,4,6,8,10,12,14,16,18,20,22,23)$$

La section suivante montre comment exploiter ces séquences pour remonter à la clé.

6.5.2.4 Exploitation

Dans l'analyse qui suit, nous considérons que l'on a réalisé une campagne d'injection de fautes dans laquelle le texte clair (entrée du DES) n'a pas varié, et qu'il est connu. On obtient pour ce texte clair un ensemble de séquences $\{\sigma_1 \ldots \sigma_I\}$ correspondant à I séquences erronées obtenues dans la campagne d'injection de fautes. Soit Σ l'ensemble des séquences de sous-clés. Pour chacune de ces séquences, l'attaquant dispose du texte clair, du chiffré correct et du chiffré fauté.

$$\Sigma = \{\sigma_1,\ldots,\sigma_I\} \cup \{\sigma_C\} \cup \{\sigma_P\}$$

σ_C représente la séquence des sous-clés correcte, et σ_P représente l'ensemble de la séquence des sous-clés correspondant au texte clair (ensemble vide).

Pour exploiter Σ, nous proposons d'analyser les couples $(\sigma, \sigma') \in \Sigma$ qui sont proches au sens où ils ont un préfixe commun qui représente la majeure partie de chaque séquence. Pour ces couples, nous définissons π le préfixe commun, $\{\alpha_1, \ldots, \alpha_t\}$ le suffixe de σ et $\{\alpha'_1, \ldots, \alpha'_{t'}\}$ le suffixe de σ'. Sans perte de généralité, on suppose que $t' \leq t$.

Par exemple, supposons que les séquences σ_2 et σ_3 correspondent respectivement au saut de rondes [9 → 12] et [9 → 13]. Nous avons donc :

$$\sigma_2 = (1,2,4,6,8,10,12,14,16,18,20,22,23) = \pi \mid \alpha_1 \mid \alpha_2$$
$$\sigma_3 = (1,2,4,6,8,10,12,14,16,18,20,21) = \pi \mid \alpha'_1$$

Avec:

$$\pi = (1,2,4,6,8,10,12,14,16,18,20)$$
$$\alpha_1 = 22,\ \alpha_2 = 23\ (t = 2)$$
$$\alpha'_1 = 21\ (t'=1)$$

Comme nous allons le montrer, la cryptanalyse est très simple sur un exemple comme celui-ci (lorsque t et t' sont tous deux petits). On remonte alors à la clé très facilement.

Soit L_i (respectivement L'_i) et R_i (respectivement R'_i) les parties gauches et droites du calcul intermédiaire au début de la ronde i, avec pour séquence de sous-clé σ (respectivement σ'). Soit k le numéro de la ronde qui suit la dernière ronde du préfixe commun. Nous avons alors :

$L_k = L'_k$ et $R_k = R'_k$
L_{k+t} , R_{k+t} , $L'_{k+t'}$ et $R'_{k+t'}$ sont connus

- **Cas t=2 et t'=1**

Dans ce cas, nous avons les informations suivantes :

- $L_{k+1} \oplus R_{k+2}$ connu. R_{k+2} est connu car t=2 et L_{k+1} est connu car $L_{k+1} = R_k$, et $R_k = R'_k$, et $R'_k = L'_{k+1}$ qui est connu également car t'=1.

- R_{k+1} est connu car $R_{k+1} = L_{k+2}$ qui est lui-même connu car t=2.

Nous connaissons également les entrées et les sorties de la fonction de transformation de la ronde k+2 de la séquence σ. Connaissant les entrées et les sorties des 8 S-Boxes, il est possible d'en déduire 32 bits d'information sur la sous-clé K_{k+2}. Par conséquent, la clé K peut facilement être retrouvée par recherche exhaustive sur les 24 bits restants (soit un espace de valeur de 2^{24}).

- **Autres cas :**

La clé peut être facilement retrouvée pour les valeurs de (t, t') suivantes: (1,0), (2,0), (1,1), (2,1) et (2,2). Ces cas correspondent à de nombreux couples de résultats fautés obtenus dans nos campagnes d'injection. Pour les cas t>2, la complexité de la cryptanalyse augmente avec t.

On peut remarquer que lorsque t'=0, l'attaquant connaît les entrées et les sorties d'un DES de t rondes. Pour un t relativement petit (t=3 ou t=4), et en faisant varier les textes clairs il est possible d'appliquer la technique de cryptanalyse linéaire classique.

6.5.2.5 Résultats sur le DES de référence

Sur le DES de référence, environ 50 séquences fautées ont été observées. Cela donne un Σ largement suffisant pour effectuer une cryptanalyse. 8 de ces séquences sont dans le cas t=1 et t'=0.

De plus, certaines séquences correspondent à un "2-round" DES, c'est-à-dire un DES réduit à l'exécution de ses deux premières rondes, et plus généralement des "n-round" DES avec n<10. On peut appliquer dans ce cas une cryptanalyse différentielle linéaire classique [Hell94].

Dans la plupart des cas, l'injection de faute a probablement généré des codes faux sur le compteur 1 parmi 17 qui se propagent pour générer des signaux de contrôle faux, ce qui provoque des comportements incorrects sur les démultiplexeurs.

6.5.2.6 Résultats sur le DES durci

Le DES durci se comporte de la même manière que le DES de référence, mais les codes faux sont détectés par les alarmes. Les séquences obtenues ne sont donc pas à proprement parler exploitables, puisque l'attaquant n'y aurait pas accès si le circuit était contrôlé par un environnement. En revanche, certains fautés correspondent à des sauts de rondes et ne déclenchent pas d'alarme. L'injection de faute a généré un code *1 parmi 17* valide. Parmi ces cas dangereux, nous distinguons deux types :

- **Les erreurs qui sont reproductibles** : La figure 6.11 présente une séquence [16 → 17] comparée avec la séquence normale. On constate que le signal de fin de calcul a été généré prématurément à cause du saut de ronde. Le calcul fauté correspond à un DES de 15 rondes. Ce résultat se retrouve sur la courbe de consommation de courant du DES. Le temps de calcul est réduit d'environ 12 ns, ce qui correspond bien au temps de calcul d'une ronde. Aucune alarme n'a été déclenchée. Cette erreur est exploitable (t=0, t'=1) et elle est reproductible : nous avons effectué 10 tirs avec les mêmes coordonnées temporelles et spatiales et obtenu 10 fois le même fauté. Il est également intéressant de noter que la zone temporelle dans laquelle a été injectée la faute correspond bien à la ronde qui a été mise en défaut.

- **Les erreurs non reproductibles, ou difficilement reproductibles** : plusieurs séquences correspondant à des calculs de "n-round" ont été observées. Cependant la plupart d'entre eux semblent difficilement reproductibles.

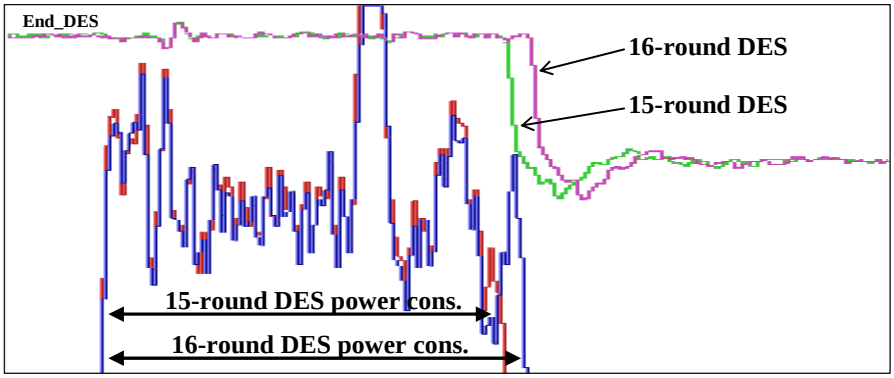


Figure 6.11 Réduction de rondes non détectée sur le DES durci

6.5.2.7 Contre mesures

Deux hypothèses principales sont avancées pour expliquer l'attaque réussie sur le DES durci :

- Une injection de fautes multiple a permuté deux fils du code 1 parmi 17. La première faute désactive un fil et la deuxième faute active un autre fil. Puisque l'attaque au laser est relativement localisée, il faut que les deux cellules concernées soient assez proches.

- La faille théorique mise en évidence par le modèle formel a été exploitée : une substitution s'est produite entre la donnée de la ronde courante et la faute qui a activé un autre rail.

Il est impossible d'affirmer que l'une ou l'autre des hypothèses (ou aucune) reflète l'injection de faute réelle. Il est en effet très difficile de le vérifier. Plusieurs éléments semblent cependant converger vers la deuxième solution :

- La faille théorique a été reproduite dans un environnement de simulation, avec des paramètres qui semblent réalistes. La mise en pratique de cette faille est parfaitement envisageable.

- Après étude du layout sur plusieurs cas de sauts de rondes, il semble difficile de produire une double faute sur le code multi rail car les portes concernées ne sont pas situées dans la même fenêtre du laser.

L'application de la technique de synchronisation du compteur avec un circuit de contrôle redondant serait efficace pour protéger le compteur de cette faille, si la deuxième hypothèse est la bonne. Cette

contre-mesure n'a pas été implantée dans le circuit DES pour des raisons de contraintes de temps sur l'envoi du circuit en fabrication.

6.5.3 Compréhension des phénomènes physiques

Le dernier objectif est de comprendre dans la mesure du possible l'effet d'un tir laser sur le circuit. Comme nous l'avons vu dans le paragraphe précédent, il est très difficile de corréler les données physiques telles que les coordonnées géographiques et temporelles du tir laser avec le phénomène observé, pour plusieurs raisons :

- Imprécisions sur la zone géographique de tir : les coordonnées du laser sont peu précises car l'origine choisie pour les coordonnées laser n'est qu'approximativement centrée sur l'origine (0,0) du layout.

- Imprécision temporelle : on connaît la tranche temporelle dans laquelle a eu lieu l'injection de faute avec une précision de 10 ns, ce qui est énorme compte tenu de la vitesse du circuit.

- Imprécision sur le modèle de faute : nous ne disposons que de très peu d'information sur la multiplicité, la nature et la durée des fautes injectées par le laser.

Si il n'est pas possible d'obtenir des informations précises sur les phénomènes au niveau logique, nous avons tout de même pu vérifier la cohérence des résultats et des hypothèses :

- Les valeurs des registres d'alarmes sont cohérentes avec la propagation théorique du signal d'alarme en fonction du bloc dans lequel on effectue l'injection de faute.

- Les sauts de rondes ont été vérifiés à plusieurs niveaux : cohérence temporelle, spatiale (attaque sur le compteur), cohérence avec la courbe de consommation et avec le signal de fin de calcul.

6.6 Conclusion

Les résultats expérimentaux ont validé plusieurs techniques de durcissement. Les résultats obtenus sont cohérents avec les résultats théoriques, en particulier pour la technique de synchronisation des rails qui améliore sensiblement à la fois la tolérance, et la résistance lorsqu'elle est couplée avec l'implantation d'alarmes.

Les techniques présentées améliorent la sécurité des circuits et sont peu coûteuses en surface, consommation, et vitesse car elles exploitent les propriétés des circuits asynchrones et leur redondance naturelle.

Le circuit de référence a montré un bon comportement puisqu'il n'a pas été possible de mettre en œuvre une attaque sur les S-Boxes. Le circuit durci a quant à lui montré une résistance complète.

Les alarmes apportent un réel gain de sécurité, et leur implantation semble naturelle dans ce type de circuit. Nous avons cependant constaté qu'elles avaient certaines limitations, car il a été possible d'attaquer le circuit durci. Les campagnes d'injections de faute par laser montrent la puissance et le potentiel de ce type d'attaque. Il a été possible de mettre en défaut une structure multi rail protégée, et nous avons montré que même ce type de protection était insuffisant. Cependant, il faut considérer que ces attaques ont été effectuées dans des conditions idéales : boite blanche, un placement contraint et une connaissance parfaite du circuit. Une attaque sur un produit réel serait sans doute beaucoup plus difficile à réaliser, mais ce travail montre la faisabilité.

CONCLUSION ET PERSPECTIVES

Les attaques par injections de fautes représentent une menace récente, mais elles disposent d'un potentiel important. Il devient aujourd'hui indispensable de protéger les systèmes contre ce type d'attaques, tout comme les attaques par canaux cachés.

Nous avons montré dans ces travaux de recherche que la technologie asynchrone possédait un potentiel important et des propriétés intéressantes pour concevoir des systèmes sécurisés. Après avoir analysé le comportement de ces circuits en présence de fautes, nous avons proposé des contre-mesures dont l'implantation est efficace et peu coûteuse.

L'alternative asynchrone est d'autant plus intéressante qu'elle cumule des propriétés attractives contre différents types d'attaques: fautes, DPA tout en proposant des contre-mesures peu coûteuses en terme de surface, de vitesse et de consommation.

Le travail de thèse a consisté dans un premier temps à établir des critères de sensibilité aux fautes des circuits QDI. Ces critères ont permis de comprendre et déterminer le comportement de ces circuits en présence de fautes selon une approche dynamique pour évaluer la sensibilité aux fautes transitoires, et une approche formelle pour étudier le comportement des circuits en présence de fautes mémorisées. La classification des comportements nous a permis d'établir les propriétés des circuits QDI en présence de fautes et des points faibles ont été mis en évidence.

Les contre-mesures ont été proposées pour minimiser voire supprimer les occurrences des cas dangereux. Plusieurs techniques ont été présentées pour améliorer la tolérance des circuits (synchronisation des rails, circuit de contrôle) et pour faciliter la détection des fautes mémorisées (alarmes) avec un faible coût en surface, en consommation et en vitesse. Ces techniques ont été validées à la fois en simulation par l'outil d'analyse de sensibilité aux fautes transitoires et en pratique par le test de prototypes DES avec injection de fautes par laser.

Les résultats expérimentaux ont permis d'évaluer le comportement intrinsèque de la logique asynchrone QDI face à une injection de faute par laser, et de démontrer l'apport en sécurité des techniques de durcissement proposées. Cependant nous avons aussi constaté qu'elles avaient certaines limites et qu'il était possible, dans de bonnes conditions expérimentales, d'attaquer le circuit même durci. Ceci montre la puissance et le potentiel de ce type d'attaques.

La faille ayant permis d'attaquer le compteur du DES a été analysée et mise en évidence par le modèle formel proposé. Nous sommes aussi en mesure d'appliquer une contre-mesure afin de réduire la sensibilité du circuit et de rendre la faille inexploitable.

Le travail a été orienté principalement vers des objectifs de sécurité des circuits, aussi bien dans l'analyse que dans l'application des contre-mesures et dans le moyen d'injection de fautes utilisé pour valider la technologie et le durcissement. Cependant, les méthodes considérées dans ce travail pourraient être exploitées dans le domaine du test en ligne même si certains aspects sont différents. En particulier, la stratégie appliquée après une détection ou un blocage du circuit est à déterminer en fonction du contexte considéré. Parmi les perspectives de ce travail, le test des circuits asynchrones dans un environnement radiatif représente une évaluation intéressante et complémentaire de la technologie asynchrone et des solutions proposées au cours de ce travail.

L’analyse de sensibilité aux fautes transitoires pourrait faire l’objet d’un développement plus important. En particulier, une quantification de la sensibilité aux fautes plus générale que celle proposée ici pourrait permettre de comparer quantitativement un circuit synchrone et asynchrone. Ceci nécessite un développement plus important de l’outil et une certaine standardisation des critères d’analyse pour que la comparaison soit objective et juste.

L’outil d’analyse formelle fera l’objet de plusieurs extensions. L’une des plus importantes est l’ajout de critères temporels dans le modèle formel. La connaissance des délais de propagation des éléments du circuit nous permettra de mettre en évidence les cas dangereux en raison des fenêtres temporelles ou au contraire de prouver la résistance d’un circuit. La deuxième extension concerne l’ajout des règles permettant l’injection de fautes multiples. Les techniques de durcissement proposées dans ce travail concernent principalement les fautes uniques, et seulement une partie des fautes multiples. En particulier, aucune contre-mesure ne permet de détecter l’injection de deux fautes simultanées inversant la

valeur logique d'un canal double rail. L'extension de l'outil d'analyse aux fautes multiples pourrait permettre de mieux comprendre le comportement du circuit soumis à ces conditions, et ainsi de proposer et valider des contre-mesures adaptées.

Enfin, les analyses et les contre-mesures proposées pourront faire l'objet d'une intégration au sein du flot de synthèse des circuits QDI, avec un certain degré d'automatisme. Cela permettra ainsi la synthèse de circuits sécurisés.

BIBLIOGRAHIE

[Ale02] D. Alexandrescu, L. Anghel, M. Nicolaidis, “Simulating single event transients in DVSM ICs for ground level radiation”, 3rd IEEE Latin American Test Workshop (LATW'02), Montevideo, Uruguay, February 10-13, 2002.

[Ang00] L. Anghel, M. Nicolaidis, “Cost Reduction and Evaluation of a Temporary Faults Detecting Technique”, Design, Automation and Test in Europe Conference (DATE), Conference IEEE CS. pp 591-597. March 2000. Paris, France.

[Ang02] L. Anghel, “Test et Conception en vue du Test des Circuits Intégrés”. Cours ENSERG. Septembre 2002.

[Bao97] F. Bao, R. H. Deng, Y. Han, A. B. Jeng, A. D. Narasimhalu, T. H. Ngair, "Breaking Public Key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults", 5th International Workshop on Security Protocols, LNCS, vol. 1361, pp. 115-124, Paris, France, 1997.

[Ber03] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri, “Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard”, IEEE Transactions on Computers, Vol. 52, No. 4, pp. 493-505, April, 2003

[Biha91a] E. Biham, A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES", Advances in Cryptology CRYPTO '91 Proceedings, Springer-Verlag, LNCS, vol. 740, pp. 487-496, 1992.

[Biha91b] E. Biham, A. Shamir, “Differential Cryptanalysis of DES-like Cryptosystems”, Journal of Cryptology, Vol. 4 No. 1, pp.3-72, 1991.

[Biha97] E. Biham, A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems", 17th Annual International Cryptology Conference on Advances in Cryptology CRYPTO'97, vol. 1294, pp. 513-525, California, USA, 1997.

[Bone97] D. Boneh, R. A. DeMillo, R. J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", EUROCRYPT'97, LNCS, vol. 1233, pp. 37-51, Berlin, 1997.

[Boue04a] F. Bouesse, M. Renaudin, F. Germain, "Asynchronous AES Crypto-processor Including Secured and Optimized Blocks", Journal of Integrated Circuits and Systems (JICS), vol. 1, pp. 5-13, 2004.

[Boue04b] F. Bouesse, M. Renaudin, B. Robisson, E. Beigne, P. Y. Liardet, S. Prevosto, and J.

Sonzogni, "DPA on Quasi Delay Insensitive Asynchronous Circuits: Concrete Results," presented at XIX Conference on Design of Circuits and Integrated Systems (DCIS2004),Bordeaux, France, November, 24-26, 2004.

[Boue05a] F. Bouesse, "Contribution à la conception de circuits intégrés sécurisés: l'alternative asynchrone", thèse de doctorat de Institut National Polytechnique de Grenoble (INPG), 2005.

[Boue05b] F. Bouesse, M. Renaudin, S. Dumont, F. Germain, “DPA on Quasi Delay Insensitive Asynchronous Circuits: Formalization and Improvement”, Design Automation and Test in Europe Conference and Exhibition (DATE 2005), Munich Germany, March 7-11, 2005, p.424

[Boue05c] F. Bouesse, M. Renaudin, "Improving DPA resistance of Quasi-delay Insensitive Asynchronous Circuits", 3rd International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2005, Saint-Etienne, France, June 8 - 11th 2005.

[Boue06] F. Bouesse, G. Sicard, M. Renaudin, "Path Swapping Method to Improve DPA resistance of Quasi Delay Insensitive Asynchronous circuits", Workshop on Cryptographic Hardware and Embedded Systems (CHES), LNCS, vol. 4249, pp. 384-398, Yokohama, Japan, Oct 10th-13th, 2006.

[Bre04] L. Breveglieri, I. Koren, P. Maistri, "Detecting Faults in Four Symmetric Key Block Ciphers", 15th IEEE International Conference on Application-Specific Systems, Architectures, and Processors, (ASAP'04), pp.258-268, Sept 2004.

[Bur90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, “Sequential circuit verification using symbolic model checking”, 27th ACM/IEEE Design Automation Conference, pp.46-51, 1990.

[Chou05] H. Choukri, M. Tunstall, “Round Reduction Using Faults”, 2nd Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 05), pp. 13-24, Edinburgh, Scotland, September 2, 2005.

[Chu85] T.-A. Chu, C. K. C. Leung, T. S. Wanuga, "A design methodology for concurrent VLSI systems", International Conference on Computer Design (ICCD). 1985, pp. 407-410, IEEE Computer Society Press.

[Dia01] M. Diaz, "Les réseaux de Petri, Hermes", Traité IC2, Paris, 2001.

[Dod94] P. E. Dodd, F. W. Sexton, P. S. Winokur, “Three-Dimensional simulation of Charge Collection and Multiple-Bit Upset in Si Devices”, IEEE Transactions on Nuclear Science, vol. 41, no. 6, pp. 2005-2017, December 1994

[Essa02] M. Essalhiene, L. Fesquet, M. Renaudin, "Dynamic Voltage Scheduling for Real Time Asynchronous Systems", 12th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS02),Sevilla, Spain, 11-13, September, 2002.

[FIPS197] NIST, "Advanced Encryption Standard (AES)," National Institute of Standard and Technology, FIPS PUBS 197, November 2001

[FIPS46a] NIST, "Data Encryption Standard (DES)", National Institute of Standard and Technology FIPS PUB46-2, December 1993

[FIPS46b] NIST, "Data Encryption Standard (DES and Triple-DES)," National Institute of Standard and Technology FIPS PUB46-3, October 1999

[Fol05] B. Folco, V. Bregier, L. Fesquet, M. Renaudin, “Technology Mapping for Area Optimised Quasi Delay Insensitive Circuits”, International Conference on Very Large Scale Integration (VLSI-SOC), Perth, Australia, 2005, pp. 146-151.

[Fou90] P. Fouillat, ”Contribution à l’étude de l’interaction entre un faisceau laser et un milieu semi-conducteur. Applications à l’étude du Latchup et à l’analyse d’états logiques dans les circuits intégrés en technologie CMOS”, thèse de doctorat de l’université Bordeaux I, 1990.

[Hell94] M. Hellman, S. Langford, “Differential-Linear Cryptanalysis”, Advances in Cryptology - CRYPTO ’94 (Lecture Notes in Computer Science no. 839), Springer-Verlag, pp. 26-39, 1994.

[Gal02] C. Galke, M. Pflanz, H. T. Vierhaus, “On-line Detection and Compensation of Transient Errors in Processor Pipeline-Structures”, Eighth IEEE International On-Line Testing Workshop (IOLTW'02). July 08/10 2002. Isle of Bendor, France.

[Gand01] K. Gandolfi, C. Mourtel, F. Olivier, "Electromagnetic Analysis: Concrete Results" Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS, vol. 2162, pp. 251-261, Paris, France, 2001.

[Gor94] T.J. O’Gorman, "The effect of cosmic rays on soft error rate of a DRAM at ground level”, IEEE Transactions on Electron Devices, vol. 41, no.4, pp. 553-557, 1994.

[Gueu98] P. Gueulle, "*Cartes à Puce: Initiation et Applications"*, vol. 2, ETSF ed. Paris: Dunod, pp. 16-18, 1998.

[Hab92] D. H Habing, “The Use of Lasers to Simulate Radiation-Induced Transients in Semiconductor Devices and Circuits”, IEEE Transactions On Nuclear Science, vol. 39, n°6, p. 1647-1653, 1992.

[Hevi99] A. Hevia, M. Kiwi, "Strength of Two Data Encryption Standard Implementation Under Timing Attacks", *ACM Transaction on Information and System Security (TISSEC)*, vol. 2, pp. 416-437, 1999.

[Jeon97] Y. J. Jeong, W. P. Burleson, "VLSI array Algorithms and Architectures for RSA modular mutiplication", *IEEE Transactions on Very Large Scale of Integration (VLSI) Systems*, vol. 5(2), pp. 211-217, 1997.

[Kak96] C. K. Koç, T. Acar, B. S. Kaliski, "Analyzing and Comparing Montgomery Multiplication

Algorithms", *IEEE Micro*, vol. 16(6), pp. 26-33, 1996.

[Koeu99] F. Koeune, J. J. Quisquater, "Timing Attack against Rijndael", UCL Crypto Group, Louvain la Neuve June 10 1999. http://web.engr/oregonstate.edu/~aciicmez/osutass/data/Koeune99.dpf.

[Kömm99] O. Kömmerling, M. G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors", USENIX Workshop on Smartcard Technoloy (Smartcard 99), pp. 9-19, Chicago, Illinois, USA, May,10-11, 1999.

[Lafr04] C. LaFrieda, R.Manohar, “Fault Detection and Isolation Techniques for Quasi Delay-Insensitive Circuits“, International Conference on Dependable Systems and Networks (DSN'04), Florence Italy, June 28 - July 01, 2004, p.41-50.

[Lang94] S. K. Langford, M. E. Hellm, "Cryptanalysis of DES", presented at RSA Data Security Conference, pp. 353-365, Berlin, Jan 12-14, 1994.

[Lens97] A. K. Lenstra, "Memo on RSA signature generation in the presence of faults”, Sept 28, 1996, arjen.lenstra@citicorp.com.

[Lev02] R. Leveugle, ”Automatic Modifications of High Level VHDL Descriptions for Fault Detection or Tolerance”, Design, Automation and Test in Europe Conference (DATE). pp.837-841. March 4-8 2002.

[Lim00] F. Lima, E. Costa, L. Carro, M. Lubaszewski, R. Reis, S. Rezgui, R. Velazco, “Designing and Testing a Radiation hardened 8051-like Micro-Controller”, 3rd Military and Aerospace Applications of Programmable Devices and Technologies International Conference. v.1. 2000. Maryland.

[Mahe97] D. P. Maher, "Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective", First International Conference on Financial Cryptography, LNCS, vol. 1318, pp. 109-122, Anguilla, British West Indies, 1997.

[Mats94] M. Matsui, "Linear Cryptanalysis Method for DES Cipher", Eurocrypt'93. LNCS, vol. 765, pp. 386-397, May 23-27, 1993.

[Mau03] P. Maurine, J. B. Rigaud, F. Bouesse, G. Sicard, M. Renaudin, "Static Implementation of QDI Asynchronous Primitives", 13th International Workshop on Power and Timing Modeling, Optimization and Simulations (PATMOS2003),LNCS, vol. 2799, pp. 181-191, Torino, Italy, 10-12 September, 2003.

[Mont85] P. Montgomery, "Modular Multiplication without Trial Division", *Mathematics of Computation*, vol. 44 N°170, pp. 519-521, 1985.

[Moor00] S. Moore, R. Anderson, and M. Kuhn, "Improving Smartcard Security Using Self-timed Circuit Technology", presented at 4th Asynchronous Circuit Design Workshop (ACiD 2000),Grenoble, France, January 31, February 1st, 2000.

[Moor02] S. Moore, R. Anderson, P. Cunningham, R. Mullins, G. Taylor, “Improving Smart Card

security using Self-timed Circuits", Eighth International Symposium on Asynchronous Circuits and Systems (Async'02), Manchester, U.K, 8-11 April, 2002.

[NSA00] NSA, "NACSIM 5000 Tempest Fundamentals", Fort George G. Meade, Maryland 20755, December 31, 2000.

[One96] A. One, "Smashing the stack for fun and profit", Phrack Magazine, vol. 7, no. 49, p. File 14, 1996.

[Pany04] D. Panyasak, "Réduction de l'Emission Electromagnétique des Circuits Intégrés: L'Alternative Asynchrone," Thèse de l'*Institut National Polytechnique de Grenoble*. Grenoble, France, 2004.

[Pap01] K.S. Papadomanolakis, A.P. Kakarountas, V. Kokkinos, N. Sklavos, C.E. Goutis. "A Comparative Study on Fault Secure Signed Multiplication Designs", 11th International Conference on VLSI, The Global System On Chip Design & CAD Conference (IFIP VLSI SOC '01), pp.183-188, Dec 3-5 2001. Montpellier, France.

[Pick78] J.C. Pickel, J.T. Blandford, "Cosmic ray induced errors in MOS memory circuits", IEEE Transactions on Nuclear Science., vol NS-25, no. 6, pp. 1166-1171, Dec. 1978.

[Pies95] S. J. Pisetrak, T. Nanya, "Toward totally self-checking delay-insensitive systems", Dig. Pap. 25th Int. Symp. On Fault-Tolerant Computing, 1995, pp.228-237.

[Pfl02] M. Pflanz, K. Walther, C. Galke, H.T. Vierhaus, "On-Line Detection and Correction in Storage Elements with Cross-Parity Check", Eighth IEEE International On-Line Testing Workshop (IOLTW'02).pp 69-73. July 08-10, 2002, Isle of Bendor, France.

[Pou00] V. Pouget, "Simulation expérimentale par impulsions laser ultra-courtes des effets des radiations ionisantes sur les circuits intégrés.", Thèse de doctorat de l'université de Bordeaux I, 2000.

[Quis00] J. J. Quisquater, D. Samyde, "A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions, the SEMA and DEMA methods", EUROCRYPT'2000,Bruges, Belgium, May 14-18, 2000.

[Rao01] J. R. Rao, P. Rohatgi, "EMpowering Side-Channel Attacks," IACR ePRINT 2001/037, May 11,2001.

[Raz04] A. Razafindraibe, P. Maurine, M. Robert, F. Bouesse, B. Folco, M. Renaudin, "Secured structures for secured asynchronous QDI circuits", XIX Conference on Design of Circuits and Integrated Systems, Bordeaux, France, November 24-26, 2004.

[Ren00] M. Renaudin, "Asynchronous Circuits and Systems : a promising design alternative", in "Microelectronics for Telecommunications : managing high complexity and mobility" (MIGAS 2000), special issue of the Microelectronics-Engineering Journal, Guest Editors : P. Senn, M. Renaudin, J. Boussey, Vol. 54, N°1-2, Dec. 2000, pp. 133-149.

[Ren03] M. Renaudin, J. Fragoso, "Asynchronous Circuits Design: An Architectural Approach",

chapter in “V Escola de Microeletrônica da SBC-Sul”, Edited by José Gunzel & Ricardo Reis, Rio Grande, Sept. 17-20, 2003.

[Riv78] R. Rivest, A. Shamir, L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM, Vol. 21 (2), pp.120–126, 1978. Previously released as an MIT "Technical Memo" in April 1977.

[Roc96] R. Rochet “Synthèse Automatique de contrôleurs avec contraintes de Sûreté de Fonctionnement", Thèse (CSI-INPG), 1996, Grenoble.

[Ross01] A. Ross, "Security Engineering:A Guide to Building Dependable Distributed Systems", vol. 1, Wiley Computer publishing ed. United States of America, 2001.

[Ross96] A. Ross, M. Kuhn, "Tamper Resistance a cautionary Note", second USENIX Workshop on Electronic Commerce Proceedings, pp. 1-11, November, 1996.

[Samy02] D. Samyde, S. Skorobogatov, R. Anderson, J. J. Quisquater, "On a New Way to read Data from Memory", First International IEEE Security in Storage Workshop,Greenbelt Maryland, December 11-11, 2002.

[Saw74] D. H. Sawin, G. K. Maki, “Asynchronous Sequential Machines Designed for Fault Detection", IEEE Transactions on Computers, Vol C-23, March, 1974, pp. 239-249.

[Schn01] B. Schneier, “*Cryptographie Appliquée"*, 2 ed: Vuibert Informatique, 2001.

[Slim04] K. Slimani, "Une Méthodologie de Conception de Circuits Asynchrones à Faible Consommation d'Energie: Application au Microprocesseur MIPS," Thèse de l’*Institut National Polytechnique de Grenoble,* Grenoble, France, 2004.

[Spar01] J. Sparso, S. Furber, “Principles of Asynchronous Circuit Design: A systems Perspective”, Kluwer Academic Publishers, 2001.

[Stin96] D. Stinson, "*Cryptographie: Théorie et Pratique"*, Paris: International Thomson Publishing France, 1996, pp.106.

[Suth89] I. Sutherland, "Micropipelines", Turing award lecture. Communications of the ACM, 32 (6):720--738, June 1989.

[Tre89] M. Tremblay, Y. Tamir, “Fault-Tolerance for High-Performance Multi-Module VLSI Systems Using Micro Rollback”, The Mit Press March 1989, pp. 297-316.

[TIM01] "Etude de testabilité des circuits asynchrones QDI", rapport interne, TIMA.

[Var00] F. Vargas, A. Amory, ”Recent Improvements on the Specification of Transient Fault-Tolerant VHDL Descriptions: A Case-Study for Area Overhead Analysis”, 13th Symposium on Integrated Circuit and Systems Design (SBCCI’00). pp 249-254. September 18/24 2000. Manaus, Brazil.

[Ver02] T. Verdel, Y.Makris, “Duplication-Based Concurrent Error Detection in Asynchronous

Circuits: Shortcomings and Remedies", 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 345-353, 2002.

[Wad05] J.D. Waddle, D. A. Wagner, "Fault Attacks on Dual-Rail Encoded Systems", 21st Annual Computer Security Applications Conference (ACSAC 2005), pp.483-494, December 5-9, 2005.

[Wen94] X. Wendling, "Synthèse de Circuits PC/PO à Haute Sûreté de fonctionnement" Rapport de Projet. Laboratoire CSI-INPG. Mars 1994.

[Wil91] T.E. Williams, "Self-timed rings and their application to division", Ph.D dissertation, Stanford university, May 1991.

[Wil94] T.E. Williams, "Performance of iterative computation in self timed rings", Journal of VLSI signal processing, N°7, pp. 17-31, Feb.1994.

[Won05] W. Jang, A. J. Martin, "SEU-Tolerant QDI Circuits", 11th IEEE International Symposium on Asynchronous Circuits and Systems, NY, USA, March 13-16, 2005, pp. 156-165.

BIBLIOGRAHIE DE L'AUTEUR

- **Revues**

[Mon06b] Y. Monnet, M. Renaudin, R. Leveugle, "Designing Resistant Circuits against Malicious Faults Injection Using Asynchronous Logic," IEEE Transactions on Computers, vol. 55, no. 9, Sept., 2006, pp. 1104-1115.

- **Conférences internationales avec comité de lecture**

[Mon04b] Y. Monnet, M. Renaudin, R. Leveugle, "Asynchronous Circuits Sensitivity to Fault Injection", 10th IEEE International On-Line Symposium, Madeira Island, Portugal, July 12-14, 2004, pp. 121-126.

[Mon05c] Y. Monnet, M. Renaudin, R. Leveugle, S. Dumont, F. Bouesse, "An Asynchronous DES Crypto-Processor Secured against Fault Attacks", International Conference on Very Large Scale Integration (VLSI-SOC), 2005, pp. 21-26.

[Mon05d] Y. Monnet, M. Renaudin, R. Leveugle, "Hardening Techniques against Transient Faults for Asynchronous Circuits", 11th IEEE International On-Line Testing Symposium (IOLTS), Saint Raphael, French Riviera, France, July 6th-8th, 2005, pp. 129-134.

[Mon05e] Y. Monnet, M. Renaudin, R. Leveugle, "Asynchronous circuits transient faults sensitivity evaluation", 42th Design Automation Conference (DAC), Anaheim, USA, June 13-17, 2005, pp. 863-868.

[Mon06a] Y. Monnet, M. Renaudin, R. Leveugle, N. Feyt, P. Moitrel, F. M'Buwa Nzenguet, "Practical Evaluation of Fault Countermeasures on an Asynchronous DES Crypto Processor", 12th IEEE International On-Line Testing Symposium (IOLTS), Lake of Como, Italy, July 10th-12th, 2006, pp. 125-130.

[Mon06c] Y. Monnet, M. Renaudin, R. Leveugle, C. Clavier, P. Moitrel, "Case Study of a Fault Attack on Asynchronous DES Crypto-Processors", 3rd Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 06), Springer-Verlag, LNCS, vol. 4236, Yokohama, Japan, Oct., 2006, pp.88-97.

[Mon07] Y. Monnet, M. Renaudin, R. Leveugle, "Formal Analysis of Quasi Delay Insensitive Circuits Behavior in the Presence of SEUs", 13th IEEE International On-Line Testing Symposium (IOLTS), Hersonissos-Heraklion, Crete, Greece, July 8-11, 2007.

- **Conférences internationales sans comité de lecture (invité)**

[Ren06] M. Renaudin, Y. Monnet, “Asynchronous Design : Fault Robustness and Security Characteristics”, 12th IEEE International On-Line Testing Symposium (IOLTS), Lake of Como, Italy, July 10th-12th, 2006, pp. 92-95.

[Ren04] M. Renaudin, F. Bouesse, and Y. Monnet, "Improving DPA and DFA resistance of circuits using asynchronous logic," presented at e-Smart 2004 and e-govermment & Smartcart International Meeting 5th Edition, Sophia Antipolis, French Riviera, France, September, 22-24, 2004.

[Ren05a] M. Renaudin, F. Bouesse, Y. Monnet, "Secure asynchronous circuits for Smart-Card applications: Design and Methodologies", MEDEA+ DAC Conference, Les Mesnuls, France, 24-26 May 2005.

[Fes05] L. Fesquet, F. Bouesse, Y. Monnet, M. Renaudin, "Implementing Asynchronous Circuits on LUT Based FPGAs", 3rd International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2005, Saint-Etienne, France, June 8 - 11th, 2005

[Ren05b] M. Renaudin, F. Bouesse, Y. Monnet, L. Fesquet, "Secure asynchronous circuits design and prototyping", 3rd International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2005, Saint-Etienne, France, June 8 - 11th 2005.

- **Conférences nationales**

[Mon03] Y. Monnet, M. Renaudin, R. Leveugle, "Etude et modélisation de circuits résistants aux attaques par injection de fautes", Journée des doctorants de l'ED EEATS 2003, 11 Décembre 2003, Grenoble, France.

[Mon04a] Y. Monnet, M. Renaudin, R. Leveugle “Etude et modélisation de circuits résistants aux attaques par injection de fautes”, 7èmes Journées Nationales du Réseau Doctoral de Microélectronique 2004 (JNRDM’04), Marseille, France.

[Mon05a] Y. Monnet, M. Renaudin, R. Leveugle, "Analyse du comportement des circuits asynchrones en présence de fautes", Journées Nationales du Réseau des Doctorants en Microélectronique (JNRDM’05), Paris, France, 9-11 mai, 2005.

[Mon05b] Y. Monnet, F. Bouesse, M. Renaudin, "Designing resistant asynchronous circuits against malicious fault injection", 3rd International Workshop on Cryptographic Architectures Embedded in Reconfigurable Devices - CryptArchi 2005, Saint-Etienne, France, June 8-11th 2005.

RESUME

Le domaine de la cryptanalyse a été marqué ces dernières années par la découverte de nouvelles classes d'attaques, dont font partie les attaques par injection de fautes. Le travail de thèse vise à développer des outils et des techniques destinés à rendre les circuits robustes face aux attaques par injection de fautes (Differential Fault Analysis : DFA). On s'intéresse en particulier à étudier la modélisation et la conception de circuits asynchrones résistants à ces attaques. Le travail porte dans un premier temps sur l'analyse de la sensibilité aux fautes de ces circuits, puis sur le développement de contre-mesures visant à améliorer leur résistance et leur tolérance. Les résultats sont évalués en pratique sur des circuits cryptographiques asynchrones par une méthode d'injection de fautes par laser. Ces résultats valident les analyses théoriques et les contre-mesures proposées, et confirment l'intérêt des circuits asynchrones pour la conception de systèmes sécurisés.

MOTS CLES

Circuits asynchrones, modélisation, attaques non intrusives, injection de fautes.

ABSTRACT

New hardware cryptanalysis methods such as fault-based attacks have shown their efficiency to break cryptosystems. This work is focused on the development of new techniques and tools that enable the design of robust circuits against fault injection attacks (Differential Fault Analysis: DFA). The study and the design of resistant asynchronous circuits against these attacks are particularly addressed. We first specify a faults sensitivity evaluation of asynchronous circuits. Then, hardening techniques are proposed in order to improve circuits resistance and tolerance. Practical results are evaluated on asynchronous cryptographic circuits using a laser beam fault injection system. These results validate both the theoretical analysis and the hardening techniques, and confirm that asynchronous technology is an efficient solution to design secure systems.

KEY WORDS

Asynchronous circuits, circuit modelling, non invasive attacks, fault injection.

ISBN: 978-2-84813-099-6