Conception sécurisée contre les attaques par fautes et par canaux cachés

V. Maingot

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'Institut Polytechnique de Grenoble

Spécialité : « Micro et Nano Electronique »

préparée au laboratoire TIMA

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

présentée et soutenue publiquement

par

MAINGOT Vincent

le 09 Juin 2009

TITRE

Conception sécurisée contre les attaques par fautes et par canaux cachés

DIRECTEUR DE THESE : Régis LEVEUGLE

JURY

M. Frédéric Pétrot	, Président
M. Viktor Fischer	, Rapporteur
M. Lionel Torres	, Rapporteur
M. Régis Leveugle	, Directeur de thèse
M. Frédéric Valette	, Examinateur

Que tourne le Vieux Moulin !

Remerciements

Cette thèse a été réalisée au sein du groupe ARIS (**A**rchitectures for **R**obust and complex **I**ntegrated **S**ystems) du Laboratoire TIMA (**T**echniques de l'**I**nformatique et de la **M**icroélectronique pour l'**A**rchitecture des systèmes intégrés). Je tiens à remercier Madame Dominique BORRIONE et Monsieur Bernard COURTOIS, directeurs du laboratoire pendant ma thèse, pour m'avoir accueilli et donné les moyens d'accomplir mes travaux de recherche.

Je remercie Monsieur Frédéric Pétrot, Professeur à Grenoble INP, pour m'avoir fait l'honneur d'être le président de mon jury de thèse.

Je remercie Messieurs Viktor Fischer, Professeur à l'Université de St Etienne, et Lionel Torres, Professeur à l'Université de Montpellier 2, pour l'intérêt qu'ils ont porté à mes travaux en tant que rapporteurs.

Je remercie Monsieur Frédéric Valette, expert en sécurité à la DGA/CELAR, pour avoir accepté d'être membre de mon jury de thèse.

Enfin, je tiens à remercier tout particulièrement Monsieur Régis Leveugle, Professeur à Grenoble INP, pour m'avoir proposé ce sujet de thèse, pour son encadrement, sa disponibilité, ses conseils et pour la confiance qu'il m'accordée d'abord en stage puis en thèse.

Je tiens aussi à remercier les administrateurs du CIME, Messieurs Robin Rolland, Bernard Bosc et Alexandre Chagoya pour l'aide qu'ils ont pu m'apporter durant tout ce temps. Je remercie tout particulièrement Alexandre pour sa disponibilité, son enthousiasme et son soutien.

Merci à tous les membres du TIMA que j'ai pu rencontrer et côtoyer en particulier les membres du groupe ARIS : Abdelaziz, Gaëtan, JB, Michele, Paolo et Pierre.

Merci à ma famille, en particulier mes parents, pour le soutien et les encouragements qu'ils m'ont apportés pendant toutes ces années.

Enfin, je remercie infiniment Jerem et Tex, mes amis du Vieux Moulin, pour tous ces bons moments passés ensemble.

Table des matières

Liste des Figures

Liste des Tableaux

Introduction

Dans notre vie quotidienne, le nombre d'activités où l'utilisateur demande une forte sécurité est en pleine croissance. Cette sécurité peut être décrite de diverses façons, dépendant de l'application visée. La principale contrainte de sécurité est la confidentialité de données sensibles : le but est de garantir que les données protégées ne soient lisibles que par un certain nombre d'individus (ou d'entités) bien définis ; pour toute autre personne, elles doivent demeurer inaccessibles ou inintelligibles. La solution apportée à ce problème doit garantir, d'une part, que des données ne puissent être lues et comprises que par leurs auteurs, ou d'autre part, que seul le destinataire légitime d'un message confidentiel puisse le lire. Ce besoin d'échange de données confidentielles entre deux personnes pose le problème de l'authenticité des données ou de leur provenance réelle. Ceci nécessite des mécanismes de signature de documents fiables. Cette signature a pour but d'authentifier à la fois le message et son auteur, de telle sorte qu'un vol de signature puisse être décelé. Les exemples les plus courants de ce type d'applications sont la banque, le contrôle d'accès et la télévision à la demande.

La cryptographie permet de garantir tous ces besoins de confidentialité et d'authentification de l'auteur. En effet, le chiffrement permet de rendre inintelligible un message à toute personne ne sachant pas comment le déchiffrer. De la même manière, il permet l'émission de signatures que personne ne peut imiter.

Les algorithmes de chiffrement sont nombreux et variés ; cependant, ils possèdent tous le même point commun : ils utilisent une clef de chiffrement qui leur sert de paramètre à toute opération. Sans cette clef, un espion ne peut pas déchiffrer le message, et ce même avec une parfaite connaissance de l'algorithme. Il existe deux types d'algorithmes de chiffrement : ceux à clef privée, aussi nommés symétriques et ceux à clef publique, aussi appelés asymétriques.

Le chiffrement symétrique est basé sur le fait que l'émetteur et le récepteur utilisent la même clef secrète. Tant que cette clef n'est connue que par ces deux personnes, leurs échanges restent confidentiels.

Le chiffrement asymétrique est quant à lui basé sur l'utilisation de deux clefs par personne : une publique et une privée. La clef publique est connue de tous, potentiellement publiée sur un annuaire. La clef privée reste connue de son seul propriétaire. Dans ce cas, l'émetteur chiffre son

message avec la clef publique du récepteur, qui déchiffre avec sa clef privée. L'émetteur peut signer son message en utilisant sa clef privée assurant ainsi à toute personne le recevant qu'il en est bien l'auteur, car seule sa clef publique rend la signature intelligible.

La sécurité apportée par tous ces algorithmes est basée sur la complexité pour un attaquant de trouver la clef secrète d'un utilisateur (ou couple d'utilisateurs pour un système symétrique). Mathématiquement, ces protocoles de chiffrement sont conçus pour que l'opération consistant à obtenir une clef de manière indue ne soit pas réalisable en un temps raisonnable. Le choix de ce « raisonnable » influe sur le choix de l'algorithme et de la taille des clefs secrètes utilisées.

Les applications sécurisées que nous utilisons quotidiennement ne nous demandent pas de retenir ces clefs trop difficiles à mémoriser, elles sont donc stockées dans les dispositifs que nous utilisons pour accéder à ces services. Les plus courants sont les cartes à puces : elles embarquent un système complet capable de réaliser les transactions nécessaires pour nous authentifier et transmettre les données confidentielles nécessaires à l'application associée.

Le fait que ces dispositifs contiennent les clefs secrètes dans leurs circuits les rend susceptibles d'être attaquées par des pirates voulant s'octroyer l'accès à des services auxquels ils n'ont pas droit. Ainsi, l'augmentation de la valeur des services ou données protégés par ces techniques favorise le développement de nouvelles méthodes d'attaques de ces systèmes. Les progrès de la cryptanalyse ainsi que la forte croissance de la puissance de calcul des ordinateurs permettent de casser ces systèmes plus facilement. De plus, la découverte de faiblesses dans l'implantation des algorithmes de chiffrement rend les attaques des circuits plus rentables que l'attaque théorique des algorithmes utilisés. Ces attaques se regroupent en plusieurs catégories : chacune de ces catégories exploite une faiblesse de l'implantation utilisée par le circuit attaqué.

Pour limiter ces attaques, les concepteurs de ces dispositifs mettent au point diverses protections. Nous verrons plus tard quelles sont les faiblesses ciblées ainsi que les protections à la disposition des concepteurs. Dans la majeure partie des cas, le développement d'une protection vise à combler une faiblesse donnée de l'implantation. De ce fait, une protection ne comble qu'une des faiblesses de l'implantation protégée, laissant des brèches sur d'autres fronts. De plus, lors de l'évaluation d'une protection donnée, on trouve essentiellement dans la littérature des rapports sur son efficacité à combler la brèche visée, sans recul sur l'impact potentiel vis-à-vis d'autres types d'attaques. Nous nous concentrerons dans la suite de cette thèse sur l'analyse de l'impact d'un type de protection sur la sécurité globale d'un circuit protégé.

Le chapitre 1 de ce manuscrit est consacré à l'état de l'art sur les attaques à la disposition d'un pirate voulant s'octroyer un accès non autorisé à ces applications, puis sur les types de protections mis en place contre ces attaques.

Le chapitre 2 présente les résultats d'une étude analysant l'effet d'une attaque par fautes sur un circuit sécurisé grâce à des techniques d'injection au laser. Nous analyserons dans un premier temps le comportement du circuit sécurisé choisi en présence de ces attaques. Enfin, par l'utilisation de ces mêmes techniques sur un circuit programmable de type FPGA à base de SRAM, nous verrons les risques supplémentaires qu'induit l'utilisation de ce type de technologie.

Le chapitre 3 se concentre sur l'analyse, en phase de conception, de la sensibilité d'un circuit face aux attaques par canaux cachés. Dans un premier temps, nous décrirons les flots d'évaluation de robustesse face à ces attaques. Puis, nous étudierons plusieurs versions protégées d'un registre pour évaluer l'impact des protections contre les fautes sur cette sensibilité et ainsi définir un nouveau critère d'évaluation des différentes protections choisies.

Enfin, le chapitre 4 présente les résultats obtenus lors de l'analyse de deux primitives cryptographiques : une boite de substitution (S-Box) AES et un co-processeur RSA.

Chapitre 1 : Applications Sécurisées : Menaces et Contre-mesures

Dans ce premier chapitre, nous allons voir quelles sont les vulnérabilités des dispositifs de chiffrement utilisés pour des applications sécurisées, ainsi que les moyens à la disposition de leurs concepteurs pour rendre toute attaque au niveau matériel plus difficile voire impossible.

1.1. Attaques contre une implantation matérielle d'une application sécurisée

Dans la littérature, on présente une classification des attaques contre les circuits pour applications sécurisées basée sur deux axes : le premier est le caractère intrusif de l'attaque et le second l'activité de l'attaquant.

En effet, ces attaques peuvent se différencier d'une part sur leur caractère intrusif par rapport au circuit. On distingue ici majoritairement deux grands groupes : les attaques intrusives et les attaques non intrusives. Les attaques intrusives nécessitent durant leur réalisation de sortir le circuit de son emballage (boîtier, carte à puce …) et souvent de lui faire de lourds dommages. Ces attaques nécessitent souvent un équipement de pointe très coûteux généralement indisponible à un pirate moyen. Les attaques non intrusives, quant à elles, n'endommagent pas le circuit. Elles sont donc une réelle menace, car leur caractère non intrusif les rend indétectables par le propriétaire du dispositif. De plus, elles nécessitent souvent beaucoup moins d'équipement. [Skor 02] introduit la notion d'attaques semi intrusives, pouvant nécessiter l'ablation de l'emballage sans toucher à la structure du circuit en lui-même.

D'autre part, les attaques peuvent être classifiées selon l'activité de l'attaquant durant sa réalisation. Ici aussi, on note une répartition en deux sous-ensembles : les attaques actives et les attaques passives. Lors d'une attaque passive, le pirate ne fait que mesurer certaines grandeurs physiques émanant du circuit pendant son utilisation. Les attaques actives requièrent diverses actions de la part du pirate venant agir sur le comportement du circuit ou sur son environnement.

Selon [Komm 99], les attaques contre les circuits intégrés se répartissent en quatre groupes: les attaques dites de sondage (actives ou passives et intrusives), les attaques logicielles (actives et non intrusives), les attaques d'écoute (passives et non intrusives) et les attaques par génération de fautes (actives et non intrusives ou semi-intrusives). Les attaques de sondage sont basées sur l'accès direct à des zones du circuit pour le perturber, l'écouter ou réaliser de l'ingénierie inverse. La pose de sondes sur certaines pistes de métal permet la lecture directe de valeurs, le forçage de potentiels erronés ou l'étude de la structure finale du circuit. Les attaques logicielles se basent sur l'étude approfondie des protocoles et algorithmes utilisés pour trouver les faiblesses de leurs implantations. Elles visent les moyens de communication du circuit avec l'extérieur, le logiciel embarqué et non le circuit en lui-même ; de ce fait, elles ne seront pas analysées plus en détail. Les attaques d'écoute, aussi appelées attaques par canaux cachés, sont basées sur la corrélation entre les données traitées par un circuit et certaines grandeurs physiques. Enfin, les attaques par génération de fautes sont basées sur la production d'erreurs logiques au sein du circuit pour soit corrompre les données soit détourner le chemin d'exécution.

Toutes ces attaques ne sont pas exclusives ; pour arriver à ses fins, un pirate est souvent amené à utiliser plusieurs types d'entre elles successivement ou simultanément. Un pirate préfèrera souvent commencer par une attaque intrusive pour avoir une vision plus précise de l'implantation utilisée dans le circuit qu'il veut attaquer. Ensuite, avec les données ainsi acquises, il peut définir plus précisément son protocole d'attaque en utilisant des techniques non intrusives.

1.1.1. Attaques par sondage

Les attaques intrusives requièrent un long temps de mise en place ainsi qu'un équipement à la pointe. Ce type d'attaques n'est souvent réalisable que dans des laboratoires spécialisés et bien équipés. Durant l'attaque, le boîtier du circuit est toujours détruit.

Les premières étapes de l'attaque consistent à préparer le circuit pour faciliter l'accès à certaines de ses pistes métalliques. [Komm 99] présente en détail le mode opératoire pour réaliser la mise dans un boîtier de test du circuit d'une carte à puce. La première tache est d'extraire le circuit de la carte en elle-même et de la colle qui l'y maintenait et le protégeait. Une fois le circuit mis à nu, il faut enlever les couches de passivation protectrices du circuit pour permettre un accès aux niveaux de métallisation les plus élevés. Enfin, il ne reste plus qu'à fixer le circuit dans un boîtier de test. La création des points d'accès aux pistes métalliques peut nécessiter de l'équipement assez coûteux tel qu'un banc laser ou à faisceau d'ions. Après cette préparation du circuit, il faut l'installer dans un banc de test pour le piloter comme il faut et enregistrer les valeurs voulues.

Il est aussi possible à ce niveau de réaliser l'ingénierie inverse du circuit en étudiant la structure de chaque couche du circuit, ceci aura pour effet sa destruction totale.

1.1.2. Attaques par génération de fautes

Les attaques par génération de fautes, aussi appelées attaques en fautes, ont pour principe la création de valeurs logiques erronées au sein du circuit permettant soit de corrompre les données soit de modifier le flot d'exécution. Ces attaques se réalisent en deux grandes étapes : l'induction de fautes puis l'analyse de fautes. L'étape d'induction de fautes consiste à obtenir un certain nombre de traces d'exécution en présence ou non de perturbations extérieures. L'analyse des effets de ces perturbations sur le comportement du circuit et la déduction d'informations sur la clef à partir des traces obtenues durant la première étape constituent l'analyse de fautes.

Il existe deux types de paramètres sur lesquels un pirate peut jouer pour générer des fautes dans le circuit : une modification des conditions environnementales ou la modification des signaux de contrôle.

Certaines conditions environnementales d'un circuit peuvent avoir des effets sur son comportement ; les principaux paramètres sur lesquels il est possible de jouer sont la température, les tensions d'alimentation et le voisinage électromagnétique.

La modification de la température opérationnelle d'un circuit, au-delà des bornes spécifiées par le constructeur, modifie son comportement. Les blocs de mémoire sont les éléments les plus sensibles à ce type de modifications. En effet, ces changements peuvent altérer les temps de lecture et d'écriture dans la mémoire et donc être la source d'incohérences dans les données lors de l'exécution.

En modifiant les tensions sur les alimentations lors de la commutation des portes, il est possible de fausser leurs calculs. De plus, des variation plus fines des tensions d'alimentation peuvent tout comme les variations de température altérer les propriétés temporelles des portes logiques et ainsi induire des retards dans l'établissement de valeurs mémorisées par des registres.

La modification du voisinage électromagnétique du circuit peut prendre diverses formes, mais le but reste toujours le même : induire des courants parasites dans certaines zones du circuit pour modifier des valeurs logiques internes qui se propageront suffisamment. Les techniques les plus courantes sont liées à l'utilisation de faisceaux lumineux [Skor 02], étant donné l'impact des photons sur un circuit. L'utilisation de flash lumineux peut suffire à générer des fautes dans un circuit. Cependant, il est souvent préférable d'injecter une erreur localisée ; dans ce cas, l'utilisation de faisceaux lumineux focalisés est nécessaire tels que les faisceaux laser [Habi 92]. Dans certains cas, la lumière est remplacée par un faisceau de rayons X. De manière similaire, des particules ionisantes

peuvent être utilisées, mais sont plus difficiles à mettre en place qu'un laser. Dans certains cas, certaines de ses attaques peuvent nécessiter l'ablation d'une partie du boîtier du circuit pour maximiser les effets du faisceau électromagnétique ou ionique. C'est dans ce cas que l'on parle d'attaques semi intrusives.

Les signaux de contrôle du circuit tels que les horloges et les entrées peuvent aussi être utilisés pour injecter des fautes. Les modifications des signaux d'horloge sont diverses : modification de la fréquence, modification de la symétrie du créneau et ajout d'aléas. Le but est de créer à l'instant voulu un front d'horloge prématuré provoquant l'échantillonnage d'une valeur transitoire erronée. Dans la pratique, ceci est utilisé pour éviter une lecture en mémoire, sauter une ou plusieurs transitions d'états. La symétrie du signal d'horloge est un paramètre qui voit son importance grandir dans les circuits utilisant les deux fronts de l'horloge.

Le nombre de couples de traces fautées et non fautées nécessaire pour trouver la clef dépend de l'algorithme de chiffrement attaqué et de son implantation. Certaines attaques ne nécessitent que très peu de traces d'exécutions fautées avec un contrôle très faible sur la position spatiotemporelle des fautes induites ; d'autres, quant à elles, nécessitent un grand nombre de traces ou un contrôle beaucoup plus précis des fautes induites. L'analyse de ces traces erronées requiert une bonne connaissance de l'algorithme de chiffrement attaqué ainsi que de son implantation.

L'attaque de Bellcore sur une implantation matérielle de l'algorithme de chiffrement RSA, présentée dans [Bine 97] est un très bon exemple d'attaque nécessitant un très faible contrôle sur les fautes induites et ayant une analyse de faute très simple. Supposons une opération de signature (ou déchiffrement) RSA utilisant l'optimisation en performance qu'apporte le théorème des reste chinois (CRT). Cette optimisation consiste à remplacer le calcul d'élévation à la puissance sur la taille de N par deux élévations à la puissance sur la taille de ses facteurs premiers, de taille deux fois plus petite. La Figure 1 décrit le calcul de la signature RSA.

$$N = P * Q, \text{ où } P \text{ et } Q \text{ sont premiers}$$

$$Signature : S = M^{d} \bmod N$$

$$SignatureCRT : \begin{cases} Scrt = A * (M^{d \bmod (P-1)} \bmod P) + B * (M^{d \bmod (Q-1)} \bmod Q) \\ \text{où } A = 1 \bmod P \text{ et } A = 0 \bmod Q \\ \text{et } B = 0 \bmod P \text{ et } B = 1 \bmod Q \end{cases}$$

Figure 1 : Calcul RSA en utilisant le reste chinois

Lors de l'implantation matérielle, le concepteur a le choix sur le nombre d'élévateurs à la puissance à utiliser : un pour des économies sur la surface ou deux pour gagner sur les performances

temporelles. Supposons désormais qu'une ou plusieurs fautes corrompent le calcul d'une des deux sous-exponentielles, dont nous nommons les résultats Sp et Sq. La Figure 2 montre comment à partir d'une signature correcte et d'une signature fausse, il est possible de factoriser N et donc de trouver la clef secrète du circuit. Un simple calcul de plus grand diviseur commun permet de réaliser l'attaque. Sur l'exemple, Sp est attaqué et ainsi le calcul de PGCD donne Q, de la même manière, si Sq est fauté, on trouvera P.

$$Scrt = A * Sp + B * Sq$$
$$Scrt' = A * Sp' + B * Sq$$
$$Scrt - Scrt' = A * (Sp - Sp') \; avec \; A = 0 \bmod Q$$
$$PGCD(Scrt - Scrt', N) = Q$$

Figure 2 : Exploitation du CRT pour factoriser une clef RSA

Cet exemple montre que certaines attaques ne nécessitent qu'un très faible contrôle des fautes injectées. Ici, il suffit de toucher une des deux opérations. Si elles sont réalisées sur deux blocs distincts, une injection au cœur du bloc ne devrait pas influencer le second. Si elles sont réalisées sur le même bloc séquentiellement, une injection ni trop tôt ni trop tard devrait suffire ; il faut préciser qu'un calcul d'élévation à la puissance nécessite un grand nombre de cycles d'horloges.

Un exemple d'attaque sur l'algorithme de chiffrement DES (Data Encryption Standard) peut être trouvé dans [Biha 93]. Cette attaque requiert une mise en place plus complexe que l'attaque de Bellcore. L'algorithme DES est basé sur le chiffrement de blocs de 64 bits contenant un bit de parité par octet (ie. 56 bits de données réelles), par l'itération de 16 rondes. Chaque ronde génère à partir du bloc de données et d'une sous-clef de 48 bits (calculée à partir de la clef globale) le bloc de données suivant. Une ronde est composée d'une permutation, d'un ajout de la clef et d'une substitution. Le bloc de données est divisé en deux mots de 32 bits (Li et Ri, où i représente l'indice de ronde). La Figure 3 présente la structure de la dernière ronde.

$$\begin{cases} L_{16} = R_{15} \\ R_{16} = L_{15} \oplus S(R_{15} \oplus K_{16}) \end{cases}$$

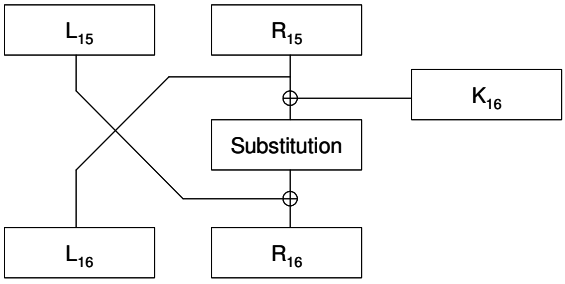

Figure 3 : Principe de la dernière ronde DES

L'attaque consiste à corrompre le calcul de L_{15} permettant ainsi grâce au résultat du même chiffrement non fauté d'obtenir une équation à une inconnue (K_{16}) ($R_{16}'\oplus R_{16} = S(L_{16} \oplus K_{16}) \oplus S(L_{16}'\oplus K_{16})$). Il ne reste plus qu'à mener une recherche exhaustive des clefs vérifiant cette équation. Ceci représente environ 2^{10} tests au lieu des 2^{56} d'une attaque par force brute. En pratique, répéter l'opération avec différents paramètres (message à chiffrer, localisation ou instant d'injection) permet de réduire encore le nombre de possibilités à explorer. Cet exemple montre une attaque par fautes nécessitant plus de traces fautées, un contrôle assez précis des fautes injectées et une analyse de faute plus complexe.

Pour d'autres exemples d'attaques, on peut se référer à [Blom 02] pour la mise en place d'attaques sur AES, IDEA, Blowfish ou SAFER, à [Bert 03] sur RC5 et [Otto 05] pour des attaques sur des algorithmes basés sur les courbes elliptiques.

1.1.3. Attaques par canaux cachés

Les attaques par canaux cachés sont basées sur l'observation de grandeurs physiques dépendantes du calcul exécuté au sein du circuit. Ces grandeurs sont appelées canaux cachés car bien qu'elles soient mesurables sur le circuit, elles ne constituent pas son résultat et ne devraient pas être porteuses d'informations sur l'état interne du circuit. Leur observation peut néanmoins donner de nombreuses informations sur les clefs de chiffrement utilisées. Les grandeurs les plus utilisées sont le temps d'exécution, la consommation électrique ainsi que le rayonnement électromagnétique. A partir de cette liste de canaux cachés, on obtient naturellement la liste correspondante d'attaques : les attaques temporelles (Timing Attack), les attaques en puissance simples ou différentielles (Simple/Differential Power Attack) et les attaques électromagnétiques (ElectroMagnetic Attack).

Les attaques temporelles sont basées sur l'analyse de données obtenues lors de la mesure précise du temps de calcul du circuit attaqué (ou le nombre de cycles nécessaires à cette opération dans le cas d'un circuit synchrone) pour un ensemble de valeurs. La mesure du temps d'exécution est faite par le dispositif pilotant le circuit attaqué, qui lui fournit entre autres son signal d'horloge, permettant ainsi de mesurer la durée entre le début du calcul et l'obtention du résultat. Une analyse de ces données aboutit à des informations sur la clef de chiffrement. Cette attaque n'est valide que sur des implantations en nombre de cycles non constant, étant donné que si aucune variation n'est observable, aucune information sur la clef n'est déductible. Les différences de temps d'exécution sont souvent dues à certaines instructions ou opérations qui ne sont réalisées que dans certaines branches de l'algorithme. Ces branches dépendent souvent des données traitées et sont donc la source des fuites sur

ce canal caché. Cette attaque a été pour la première fois présentée par [Koch 96] et réalisée par [Dhem 98]. D'autres réalisations de ce type d'attaques sont disponibles dans [Schi 00] et [Cath 03].

Les attaques par analyse de la consommation sont basées sur la variation de consommation électrique du circuit en fonction des données traitées. Ceci est lié aux différences de consommation d'une porte logique suivant ses transitions. Ainsi, il est possible à partir du profil de consommation du circuit pour un ensemble d'entrées différentes d'obtenir les valeurs de clef. On peut distinguer plusieurs types d'attaques en puissance : l'attaque en puissance simple, l'attaque en puissance différentielle et l'attaque en puissance différentielle d'ordre supérieur. L'idée est venue de [Koch 99] et une mise en place d'une telle attaque peut être lue dans [Fouq 03a].

Lors d'une attaque simple en puissance, le but est de trouver des informations sur la clef en utilisant une corrélation entre les données utilisées (messages et clef) et les traces de consommation obtenues. Par l'analyse des courbes de courant, il est facile de noter pour certains algorithmes de chiffrement la répétition de motifs, donnant ainsi des informations sur l'état du circuit à tout instant de son exécution et donc sur les branchements réalisées dans le déroulement de l'algorithme. Ceci peut dans certains cas être très utile car porteur directement d'informations sur la clef.

L'algorithme RSA est basé sur un calcul d'élévation à la puissance. Si elle est réalisée par une multiplication (M) et une élévation au carré (C) ayant toutes les deux un profil de courant différent, il est facile de déterminer la valeur de la clef (l'exposant) en identifiant l'ordre des différentes opérations réalisées pendant le calcul. La Figure 4 montre un extrait d'une trace qui pourrait être obtenue sur une telle implantation : grâce aux motifs différents pour les deux opérations utilisées, on peut dire que la clef est « 100110 » sur cet extrait.

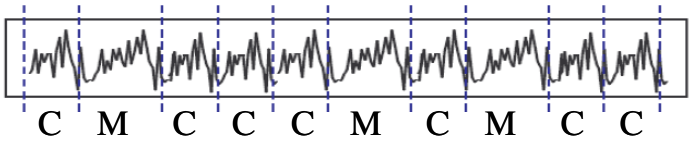

Figure 4 : Attaque en puissance simple sur RSA : extrait d'une trace

Dans d'autres cas, l'identification des motifs n'apporte pas d'information supplémentaire, c'est le cas avec l'algorithme AES, qui ne nous donne que l'indice de la ronde. Cependant, connaître la position dans le flot d'exécution peut être intéressante pour contrôler temporellement une attaque en fautes.

Lors d'une attaque différentielle, le pirate a besoin de l'enregistrement du profil de consommation du circuit pour un grand nombre de messages d'entrée. Soit E l'ensemble des entrées.

La mise en place de l'attaque nécessite le choix d'une fonction de sélection, sous-bloc du circuit attaqué dont la sortie sera utilisée par la suite. Le résultat de cette fonction de sélection doit dépendre de données connues et de la clef (ou d'une portion de la clef). La taille de la clef intervenant dans cette fonction de sélection conditionne en partie l'efficacité de l'attaque ; plus elle est petite, plus il sera rapide de trouver des fragments de clef. Une fois cette fonction de sélection choisie, il est nécessaire de choisir un critère de répartition des courbes en deux sous-ensembles E0 et E1. Cette répartition doit pouvoir mettre en évidence les différences de consommation dans la commutation d'un ou plusieurs bits en sortie de la fonction de sélection. Cette fonction de répartition est en général le poids de Hamming de la sortie de la fonction de sélection ou simplement la valeur d'un de ses bits. Pour chaque valeur possible de la clef en entrée de la fonction de sélection (d'où l'intérêt d'avoir une faible taille), l'ensemble des entrées E est séparé selon la fonction de répartition (ceci ne dépend pas des mesures réalisées). Pour chacune de ces répartitions, on détermine le biais de l'analyse différentielle comme la différence des moyennes des courbes de courants sur les deux sous-ensembles. La clef est déterminée par l'hypothèse ayant généré la courbe de biais présentant les pics les plus élevés. Si aucune courbe ne se distingue des autres, l'attaque a échoué ; ceci peut être causé par un nombre de traces trop faible ou une résistance du circuit à cette attaque. Un exemple de la mise en œuvre du concept d'attaque différentielle est donné dans [Koch 99].

Ce type d'attaque a ensuite été amélioré selon deux angles pour être plus efficace. Le premier type d'amélioration regroupe les attaques différentielles utilisant plusieurs bits de sélection et les attaques par corrélation (CPA) [LE 06]. L'autre type d'amélioration est l'élaboration d'attaques différentielles d'ordre supérieur. Elles sont basées sur le même principe que l'attaque différentielle (d'ordre 1), mais diffèrent par la forme des critères de répartition, qui considèrent plusieurs points durant le calcul de la fonction de sélection. Le but principal de ce nouveau type d'attaque est de contourner certaines contre-mesures. Une réalisation de cette attaque peut être trouvée dans [Mess 00].

Les attaques électromagnétiques sont basées sur l'étude du rayonnement électromagnétique au voisinage proche du circuit. Ce rayonnement électromagnétique est une image de l'activité interne du circuit donc corrélé de la même manière aux données traitées. Tout comme les attaques en puissance, elles peuvent se décliner en attaques simples et différentielles ; ces attaques ont été introduites par [Quis 00]. Contrairement aux attaques en puissance, ces attaques peuvent avoir une résolution de l'ordre de quelques portes en utilisant des sondes adaptées. Un exemple de mise en place d'une telle attaque peut être trouvée dans [Gand 01].

1.2. Méthodes de protection contre ces attaques

Au vu du risque que représentent ces attaques pour la sécurité des applications basées sur de tels circuits, il est nécessaire de mettre au point des mécanismes de protection. Leur but est de tenter de rendre ces attaques impossibles ou au moins très difficiles. Les protections mises au point sont propres aux attaques qu'elles limitent. Elles se repartissent donc en trois groupes.

1.2.1. Protections contre les attaques par sondage

Les attaques par sondage sont basées sur l'ablation de certaines couches de protection du circuit pour accéder aux lignes métalliques voulues.

De ce fait le premier type de protection contre ces attaques va tenter de détecter l'ablation du boîtier. Ceci peut être réalisé par l'ajout de capteurs. L'ablation du boîtier entraîne une augmentation des rayonnements ambiants à la surface du circuit, le boîtier ne jouant plus son rôle d'écran. La détection de cette variation peut être utilisée pour empêcher le fonctionnement du circuit ou le détruire.

Le second type de protection vise à rendre plus difficile la pose de sondes aux bons endroits : pour ce faire il existe des techniques de routage aléatoire, rendant la localisation des pistes plus complexe et des techniques de protection par boucliers en protégeant certaines lignes par des couches supplémentaires dont l'ablation pourrait compromettre l'attaque.

D'autre part, les attaques par sondage écoutent aussi les valeurs sur les lignes sondées, donc l'autre type de protection consiste à brouiller les données qui circulent dans le circuit. Certains proposent l'utilisation de bus chiffrés pour rendre l'interprétation des données circulant dans le circuit plus difficiles.

1.2.2. Protections contre les attaques par fautes

Les protections contre les attaques par fautes sont de nature très variées. Elles ont pour but soit de détecter l'attaque soit de la rendre inefficace, et peuvent être déployées à tous les niveaux entre le matériel et l'application.

Les protections matérielles se répartissent en trois grands groupes : certaines visent à rendre inefficaces les méthodes d'induction de fautes, d'autres essaient de détecter ou de corriger les fautes qui ont été induites et les dernières sont basées sur des modifications de l'algorithme pour rendre l'analyse plus complexe.

Dans le cas de méthodes basées sur la détection de l'attaque, il est nécessaire de prévoir comment le système complet doit réagir. Les réactions les plus utilisées sont la levée d'interruption, le gel de la sortie, la réinitialisation complète de l'unité de calcul ou du système complet, voire même l'effacement total des données critiques. Un bloc décisionnel (matériel ou logiciel) doit se charger d'appliquer la politique choisie. Selon [Yen 00], la détection de l'erreur juste avant la sortie du bloc complet n'offre pas une protection suffisante. Les méthodes de correction de fautes sont plus particulièrement dans une optique de circuits sûrs, devant maintenir leur fonctionnalité malgré la présence de fautes. Dans une optique sécuritaire la correction n'est pas indispensable, pourvu que les fautes soient bien détectées et qu'aucune information compromettante ne soit transmise au pirate.

1.2.2.1. Protections contre l'induction de fautes

La première solution pour se protéger contre les attaques par fautes est de s'assurer que les techniques d'induction de ces fautes soient inefficaces ou plus difficiles à mettre en place. Ces techniques tentent de protéger le circuit dans son intégralité et se répartissent en deux catégories : les protections passives et les protections actives.

Les protections passives ont pour but de rendre l'induction de fautes plus difficile sans savoir s'il y a vraiment une attaque en cours. Elles sont principalement basées sur la mise en place de boucliers autour du circuit. Ces boucliers permettent l'atténuation de l'effet de tous les moyens d'induction optiques ou électromagnétiques. Contourner ces protections demande donc d'endommager le circuit plus profondément ou de réaliser par exemple des attaques en face arrière.

Les protections actives ont pour but de détecter qu'une attaque est en cours. Elles sont basées sur l'utilisation de différents capteurs pour détecter toute modification anormale du milieu d'opération. Ceci concerne aussi bien les conditions environnementales telles que la température ou l'éclairage que l'intégrité des signaux de contrôle et d'alimentation. Si ces paramètres sortent de bornes prédéfinies, une attaque est détectée et un signal envoyé au bloc décisionnel pour appliquer la réaction voulue.

Ces protections peuvent être contournées en utilisant des moyens d'induction plus puissants et précis. La précision d'un tir laser permet de passer à côté des détecteurs de lumière et de générer des fautes. Pour cette raison, il faut ajouter des mécanismes de détection d'erreur au sein du circuit. Cette détection est réalisée en utilisant de la redondance.

1.2.2.2. Détection ou correction par redondance matérielle

Le principe de la redondance matérielle est de réaliser la même opération sur plusieurs copies d'un même bloc de calcul et d'en comparer les résultats. Ce principe a été utilisé pour développer plusieurs schémas de protection.

La duplication simple avec comparaison est basée sur l'utilisation de deux copies en parallèle du même bloc, suivies par la comparaison des deux résultats (cf. Figure 5).

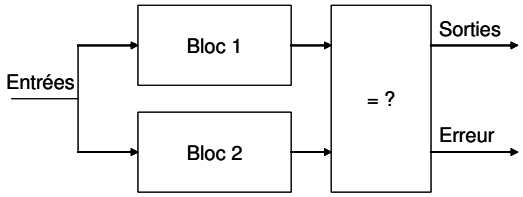

Figure 5 : Duplication simple avec comparaison

La duplication multiple avec comparaison est une extension de la duplication simple à un nombre quelconque de copies du bloc de calcul (cf. Figure 6). La triplication est une des protections les plus utilisées basées sur ce schéma.

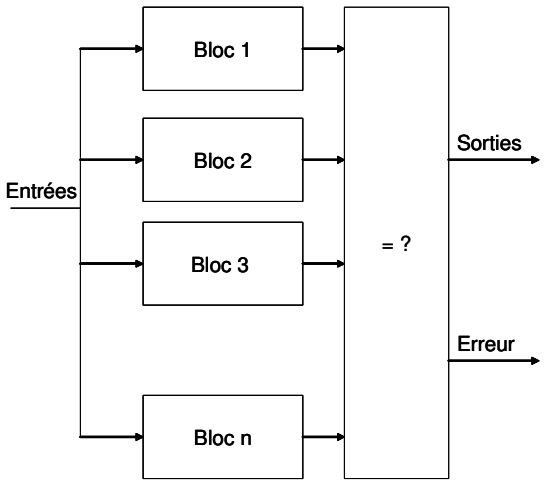

Figure 6 : Duplication multiple avec comparaison

La duplication simple avec redondance complémentaire est une optimisation de la duplication simple en utilisant pour second bloc de calcul la fonction duale du premier. Pour ce faire, il faut donc inverser les entrées et les sorties du second bloc. Cette technique a l'avantage que si la fonction protégée n'est pas autoduale, il sera plus difficile pour un attaquant de générer deux fautes qui aboutiront au même résultat (cf. Figure 7).

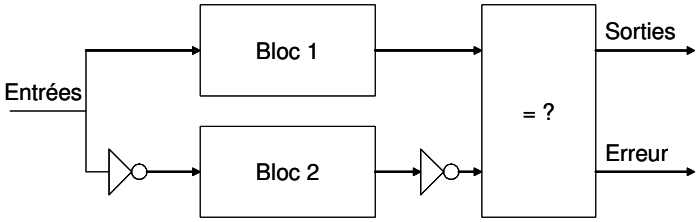

Figure 7 : Duplication simple avec redondance complémentaire

La duplication dynamique est une optimisation de la duplication multiple dans laquelle un bloc qui a donné un résultat erroné (déterminé par vote) ne sera plus utilisé jusqu'à une réinitialisation du dispositif (cf Figure 8).

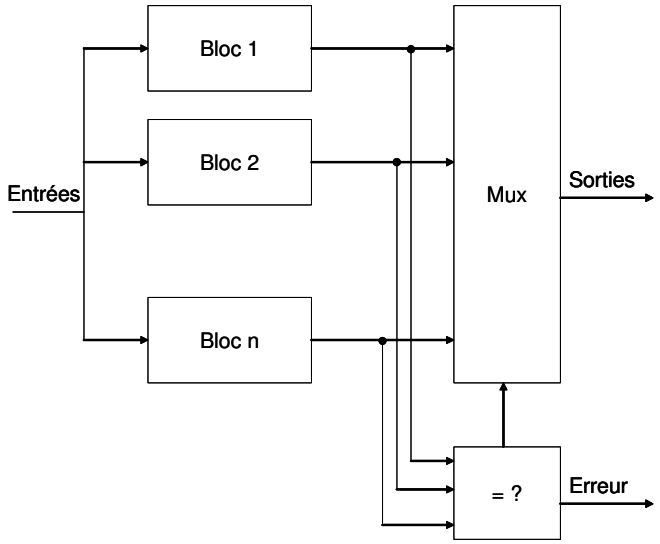

Figure 8 : Duplication dynamique

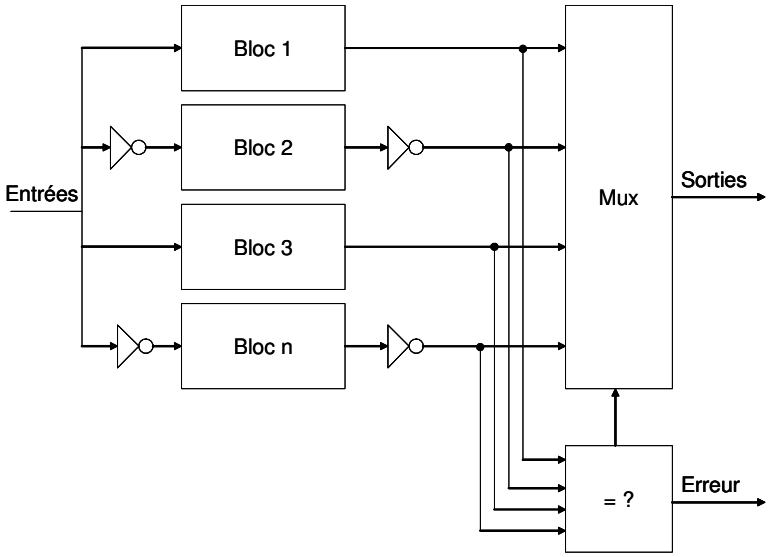

Figure 9 : Duplication hybride

La duplication hybride est un schéma utilisant la duplication dynamique où les différents blocs redondants sont partiellement des blocs duaux avec les inversions d'entrées sorties nécessaires (cf. Figure 9).

La capacité de détection de ces schémas est dépendant du nombre de copies utilisées : avec 2*n+1 blocs, il est possible de détecter 2*n erreurs et de corriger n erreurs. Dans le cas de la duplication simple et de la duplication simple complémentaire, on détecte une seule erreur sans correction possible.

1.2.2.3. Détection ou correction par redondance temporelle

La redondance temporelle est basée sur la réexécution d'un même calcul sur le même bloc matériel et la comparaison des différents résultats obtenus. Ce principe se décline aussi sur plusieurs schémas de protection.

La redondance temporelle simple est basée sur la double exécution d'un calcul sur un même bloc de calcul. Les résultats ainsi obtenus sont donc comparés (cf. Figure 10).

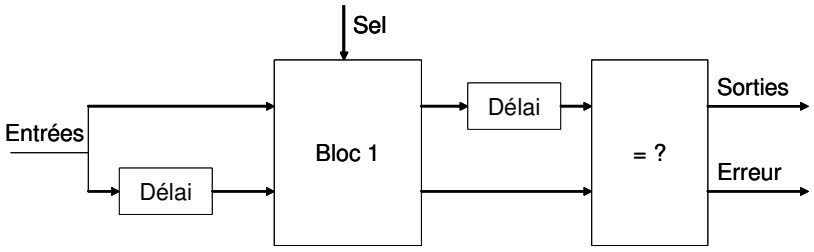

Figure 10 : Redondance temporelle simple

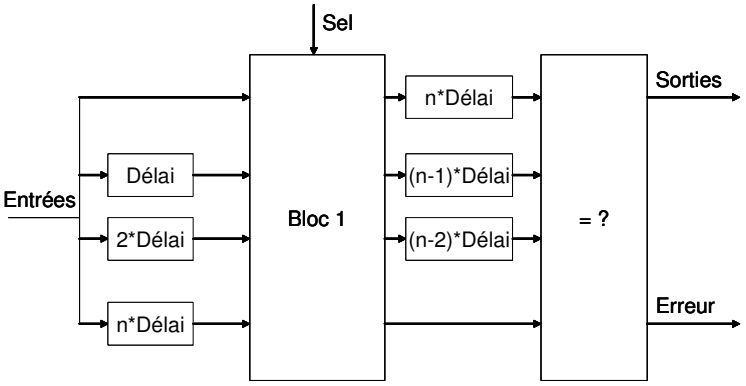

Figure 11 : Redondance temporelle multiple

La redondance temporelle multiple se base sur l'exécution multiple de la même opération sur la même unité de calcul (cf Figure 11).

[Angh 00] présente une méthode utilisant deux instants d'échantillonnage du résultat d'un bloc combinatoire, ceci permet donc de détecter les erreurs de durée inférieure au décalage entre les deux horloges d'échantillonnage sans recalcul complet de la fonction.

La redondance temporelle simple avec opérande inversée utilise le même principe que la redondance simple mais en inversant l'ordre des octets des opérandes (on passe de little endian à big endian et inversement). L'ordre est rétabli avant la comparaison. Il est évident que la fonction ainsi protégée doit avoir certaines propriétés mathématiques permettant cette inversion (cf. Figure 12).

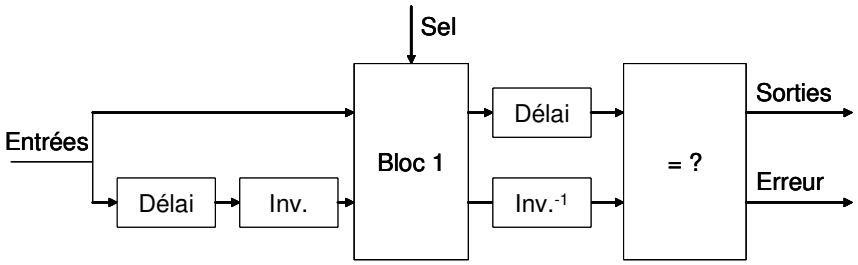

Figure 12 : Redondance temporelle simple avec opérandes inversées

La redondance temporelle simple avec rotation des opérandes utilise certaines propriétés mathématiques de la fonction à protéger permettant l'obtention du même résultat en réalisant une rotation circulaire des entrées et sorties du bloc de calcul (cf. Figure 13).

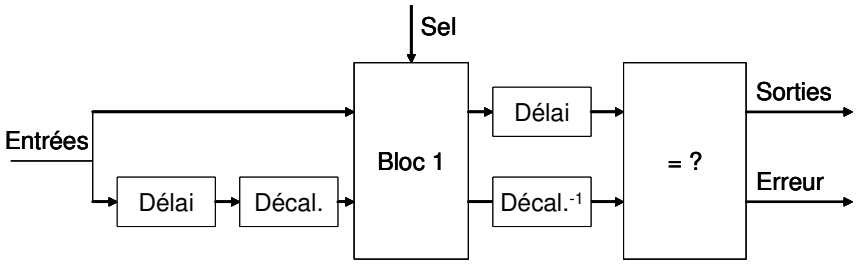

Figure 13 : Redondance temporelle simple avec rotation des opérandes

Enfin, dans le cas de fonctions involutives (étant leur propre inverse), il a été mis en évidence dans [Josh 04] que l'exécution redondante pouvait être remplacée par une ré exécution sur le premier résultat. Ainsi, si aucune erreur ne s'est produite, on obtient en fin de deuxième opération la valeur initiale.

Ces deux grands types de redondance ne sont pas mutuellement exclusifs ; on peut ainsi créer une redondance hybride basée sur l'exécution multiple de la fonction sur plusieurs copies du bloc de calcul. On obtient donc une solution dont le surcoût se répartit sur les deux aspects que sont les performances et la surface (cf. Figure 14).

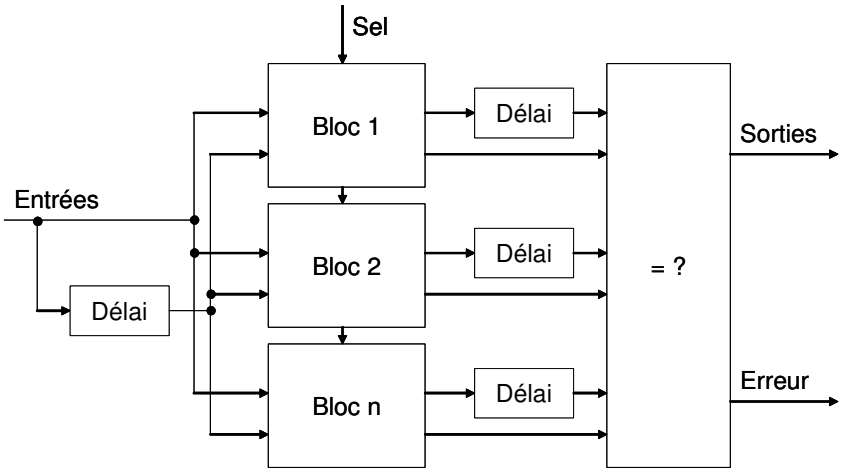

Figure 14 : Redondance hybride

1.2.2.4. Détection ou correction par redondance d'information

Le dernier type de redondance utilisable est la redondance d'information. Cette technique permet, par l'utilisation de codes détecteurs ou correcteurs d'erreur, d'obtenir une certaine protection sans nécessiter d'exécution complète supplémentaire. Ces codes sont répartis en deux grands groupes : les codes séparables et les codes non séparables. Les codes séparables sont basés sur la juxtaposition (parfois avec entrelacement) des bits de données originales et des bits de redondance. Ce type de code favorise le décodage qui est juste une sélection des bits appropriés et donc le type de code majoritairement utilisé.

Ce type de protection est aussi bien adapté aux blocs de calcul qu'aux éléments de mémorisation. Dans ces deux cas, le concept de base est le même : générer le mot de code complet puis vérifier l'intégrité de ce mot. Dans tous les cas, la vérification est gérée par un bloc combinatoire ajouté juste après le bloc protégé. La génération de ce mot code complet varie suivant le type de bloc protégé.

Pour les éléments de mémorisation, il est nécessaire d'accroître la taille des données stockées pour conserver le mot code complet et donc coder la donnée entrante.

Dans le cas d'un code séparable, le mot code est constitué de deux sous mots (les données originales et les bits de redondance) ; la vérification est donc la comparaison entre les bits de redondance stockés et ceux générés à partir des données stockées. Une différence entre ces deux mots indique la présence d'erreurs. Cependant, il n'est pas possible de déterminer leur position : données ou bits de redondance (cf. Figure 15). Certains schémas de conception du bloc de vérification, tels que le double rail, permettent de savoir si les fautes ont eu lieu dans le point mémoire ou dans les arbres de vérification ; cependant cette technique double le coût de la vérification.

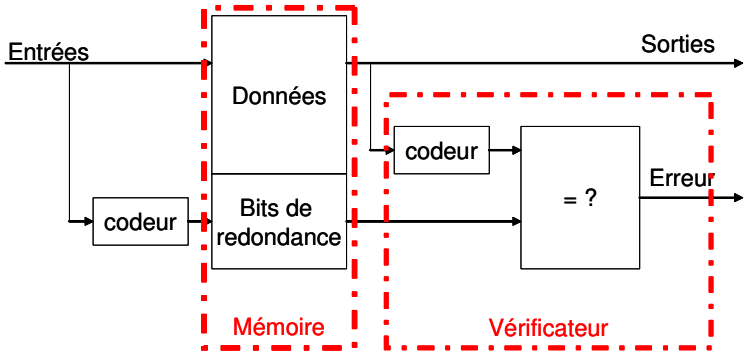

Figure 15 : Implantation d'un code détecteur sur un élément mémoire

Lors de l'application de ce type de protection sur un bloc combinatoire, il faut le modifier de manière à le rendre capable de générer en plus des sorties habituelles les bits de redondance (de préférence en évitant le simple codage de la sortie). Dans le cas d'un code séparable, cette modification du bloc combinatoire peut se limiter à la juxtaposition d'un bloc de prédiction des bits de redondance (cf. Figure 16).

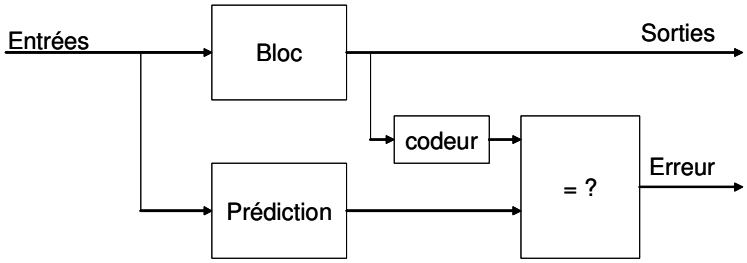

Figure 16 : Implantation d'un code détecteur sur un bloc combinatoire

Le choix du code utilisé est déterminant pour l'efficacité de la protection en terme de capacité de détection, de facilité d'implantation et de coût. Ces codes sont répartis en deux groupes : les codes linéaires et les codes non linéaires.

Un code linéaire de taille n et de dimension k – noté (n, k) – définit un sous-espace vectoriel C de dimension k inclus dans l'espace vectoriel de dimension n $GF(2^n)$. Un codage linéaire est donc un morphisme de groupe entre $GF(2^k)$ et $GF(2^n)$, qui à chaque donnée de $GF(2^k)$ associe un mot code de $GF(2^n)$. Cette application linéaire est définie par une matrice G. L'opération de codage est donc le produit matriciel entre cette matrice G et le vecteur de données à coder. La détection d'erreur est basée sur le calcul d'un syndrome, obtenu par produit matriciel entre un mot de $GF(2^n)$ par la matrice de vérification H déterminée à partir de G. Si ce syndrome est nul, le mot appartient alors au code,

indiquant ainsi l'absence d'erreur détectable. Dans le cas de codes séparables, la relation entre les matrice G et H est simple :

$$G = \begin{bmatrix} I_k \\ A_{n-k,k} \end{bmatrix}, H = \begin{bmatrix} A_{n-k,k} & I_{n-k} \end{bmatrix}.$$

Le code le plus utilisé est le code de parité. Ce code ajoute un bit égal au bit de parité de la somme des bits du mot de données. En pratique, ceci peut être calculé par un simple OU exclusif des bits de données. Divers exemples d'implantation de cette protection peuvent être trouvés chez [Bert 02], [Karr 03a], [Karr 03b] et [Kusn 04]. Dans ce cas, la matrice A est définie par $A = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}$.

Il existe une évolution du code de parité simple utilisant plus de bits de redondance. Ces bits sont les bits de parité des différents sous mots du mot de données original. Dans ce cas, la matrice A a la structure suivante, l'exemple étant donné pour trois bits de redondance :

$$A = \begin{bmatrix} 1...1 & 0...0 & 0...0 \\ 0...0 & 1...1 & 0...0 \\ 0...0 & 0...0 & 1...1 \end{bmatrix}.$$

Le code à parité croisée est une modification de la parité par sous mots qui a pour but d'éloigner le plus possible les bits de données protégés par le même bit de parité. Dans le cas de deux bits de parité, le premier protége les bits de donnée de poids pair, tandis que le second protège ceux de poids impair. Bien que le taux de détection d'erreurs soit identique, ce schéma offre une meilleure protection ; en effet, lors d'une attaque très localisée touchant tout de même plusieurs nœuds, cette méthode favorise leur répartition sur plusieurs sous mots et donc accroît le nombre de bits de protection pouvant détecter les fautes. Une implantation de cette technique est décrite par [Pfla 02], qui utilise une matrice A de la forme :

$$A = \begin{bmatrix} 1 & 0 & \dots & 1 & 0 \\ 0 & 1 & \dots & 0 & 1 \end{bmatrix}.$$

Les codes de Hamming sont très populaires dans une optique de sûreté de fonctionnement car ils permettent la correction d'erreurs simples. Leurs capacités de détection sont plus faibles : ils ne détectent que deux erreurs dans le mot protégé et ne corrigent qu'une erreur. Dans le cas de fautes multiples sur le mot protégé, la détection n'a lieu que si la multiplicité n'est pas un multiple de 3 et la correction est erronée. Ces propriétés rendent donc les codes de Hamming plus adaptés aux systèmes tolérants à faible risque de fautes multiples. La matrice H la plus utilisée est celle constituée de sorte que la $i^{éme}$ colonne corresponde à la représentation binaire du nombre i. Ceci assure que chaque bit de donnée est vérifié par chaque bit de poids apparaissant dans la représentation binaire de son indice. Par exemple, le bit 5 est vérifié par les bits 1 et 4. La matrice H pour le code de Hamming (7,4) serait :

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Une implantation de ce type de code peut être trouvée dans [Lima 00].

Le code de Hsiao [Hsia 70] permet d'obtenir des capacités de détection et de correction similaires aux codes de Hamming. La matrice de vérification de ce code est basée sur son remplissage de poids minimal sans doublon par des vecteurs colonnes de poids impair, tout en conservant la meilleure répartition du poids selon les lignes. Une matrice H pour le code Hsiao (8,4) serait :

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

La seconde grande famille de codes détecteurs d'erreur est celle des codes non linéaires. Comme leur nom l'indique, les opérations de codage, décodage et vérification ne sont plus basées sur des produits matriciels. D'après [Karp 04], les codes linéaires ont le désavantage de présenter une probabilité de détection pas assez uniforme par rapport à la multiplicité d'erreur. Une méthode pour y parvenir est la non-linéarisation de codes linéaires par l'ajout au codage d'une étape non linéaire telle que l'inversion ou l'élévation à une puissance (l'élévation au cube étant présentée dans ce papier). Il existe cependant des codes non linéaires par définition.

Les codes de Berger [Berg 61] et Dong en sont des exemples. Le code de Dong est une amélioration du code de Berger. Ce dernier est basé sur la juxtaposition aux données originales de la représentation binaire de leur poids de Hamming. Le code de Dong ajoute au mot codé par le code de Berger le complément de l'extension ajoutée par le code de Berger. Ceci permet au code de Dong (présenté plus en détails dans [Dong 82]) d'être capable de détecter toutes les fautes unidirectionnelles. Ce code a été utilisé pour protéger un microprocesseur dans [Maam 98].

Un autre type de codes est souvent utilisé pour des opérateurs arithmétiques : les codes arithmétiques. Ils sont basés sur l'ajout aux données de leur résidu modulo une petite valeur. Ceci permet d'obtenir des blocs de prédiction simples, puisqu'ils sont la même opération mais à une échelle plus petite. Plusieurs implantations de ce type de codes peuvent être trouvées dans [Brev 04], [Guer 04], [Joye 01b] et [Walt 00]. Une autre implantation, présentée dans [Mais 05], est basée sur l'utilisation d'un modulo de la forme 2^m-1.

1.2.2.5. Modifications d'algorithmes

Les protections présentées jusqu'ici permettent de détecter des erreurs dans le flot d'exécution de la fonction réalisée par le circuit. Cependant, il est aussi possible de modifier ce flot d'exécution tout en gardant la fonction globale. L'application de ce principe est très dépendante de la fonction réalisée par le bloc matériel à protéger. Les techniques les plus utilisées sont celles dites de masquage de certaines données. Ces techniques ont été utilisées sur des blocs de chiffrement DES et AES par [Tric 02] et [Akka 01], mais aussi sur des blocs de chiffrement RSA par [Gira 05] et [Joye 05] améliorant le célèbre « Shamir's trick ».

1.2.2.6. Protections logicielles

Les systèmes utilisés pour les applications sécurisées comportent des systèmes sur puces complets. Les protections précédemment citées sont basées sur l'augmentation de la robustesse des coprocesseurs de chiffrement. Les processeurs peuvent aussi être protégés au niveau matériel de la même manière. Cependant, il ne faut pas négliger de protéger le code les pilotant. Les contre-mesures logicielles sont utilisées en complément des protections matérielles. Elles permettent d'accroître de manière plus flexible la robustesse du système. Elles visent principalement à protéger le code exécuté sur les processeurs embarqués mais aussi à gérer la prise de décision en cas de détection de faute au niveau matériel. Elles utilisent au niveau logiciel tous les principes énumérés jusqu'ici.

La redondance de variable est une technique basée sur la réexecution d'une opération en utilisant des variables (positions mémoires) différentes. Les différentes variables sont ensuite comparées entre elles pour déterminer s'il y a eu des fautes ou non pendant le calcul. Comme précédemment, le nombre de copies de l'opération détermine les capacités de détection. Contrairement à la redondance matérielle, ici les différentes opérations ne sont pas exécutées tout à fait en même temps du fait de la séquentialité du traitement des instructions par le processeur. Cette technique induit un surcoût en temps et en mémoire.

La redondance d'exécution est la simple transposition au niveau logiciel de la redondance temporelle. La fonction est calculée plusieurs fois de suite en ne stockant que les différents résultats obtenus. Les résultats sont ensuite comparés pour détecter la présence de faute. Dans ce cas, la capacité de détection de fautes dépend de la même manière du nombre d'exécutions réalisées. Le coût est principalement sur le temps d'exécution total.

L'utilisation de codes correcteurs ou détecteurs est aussi possible pour protéger l'exécution d'un programme. Le but ici est de détecter les erreurs présentes dans les structures de données utilisées dans le programme. Le code le plus utilisé dans ce cas est le contrôle de redondance cyclique (CRC).

Certaines méthodes sont spécifiques à la protection logicielle ; c'est le cas de l'exécution aléatoire. Cette protection est basée sur une modification aléatoire de l'ordre d'exécution des instructions. Bien que cette protection ait été initialement conçue pour lutter contre les attaques par canaux cachés, elle a pour effet principal de rendre beaucoup plus difficile la prédiction de l'instruction en cours d'exécution par le processeur et donc la détermination de l'instant d'injection de fautes. Il y a cependant des restrictions quant aux modifications qui peuvent être apportées, le but étant bien sûr de conserver la fonctionnalité. Ceci est le plus efficace si l'attaque requiert plusieurs fautes à des cycles bien précis.

La dernière solution utilisée est l'ajout de petites portions de code de test. Régulièrement, ces petits testeurs sont exécutés et si une erreur est détectée, on incrémente un compteur ; au delà d'une certaine valeur, le système est réinitialisé. Les programmes de test sont très courts et se résument généralement à des lectures/écritures en mémoire, des opérations arithmétiques simples ou toute fonction facile à vérifier.

1.2.3. Protections contre les attaques par canaux cachés

Les protections contre les canaux cachés sont adaptées au type d'attaque qu'elles tentent d'empêcher. On retrouve donc des protections dédiées à chaque canal de fuite. Cependant, certaines mesures ont des effets sur d'autres canaux.

1.2.3.1. Protections contre les attaques temporelles

Pour contrer les attaques temporelles, il est nécessaire de réduire le plus possible la dépendance entre les données manipulées par le circuit, notamment des données critiques, et le temps d'exécution. A tous les niveaux de la conception du système, il faut proscrire l'utilisation directe de la clef pour paramétrer des branchements dans l'exécution. Pour ce faire, il est souvent nécessaire d'ajouter des opérations inutiles dans certaines branches de l'algorithme. Un exemple consisterait dans un calcul d'exponentiation par multiplication et élévation au carré à systématiquement calculer le résultat de la multiplication, indépendamment de la valeur de l'exposant. Une autre technique consiste à rendre le temps d'exécution aléatoire par l'ajout d'opérations factices aléatoirement choisies. Ceci a pour effet la complexification de l'analyse temporelle. Ces techniques sont présentées dans [Koch 96] et [Dhem 98].

1.2.3.2. Protections contre les attaques en puissance

La suppression de la corrélation entre les courbes de courant et les données traitées par un circuit est la base de toute méthode de protection pour les attaques en puissance. Les attaques en puissance simples peuvent être en partie déjouées par les mêmes protections que pour les attaques temporelles. En effet, elles recherchent une corrélation directe entre le profil de consommation du circuit et les données qu'il traite. Aussi, la trace de consommation reflète les différentes opérations réalisées au cours de l'exécution du calcul. Rendre la structure de ce profil de consommation indépendant des données traitées va donc dans le sens recherché. Ainsi, l'ajout d'opérations inutiles pour équilibrer les différentes branches d'exécution et l'exécution aléatoire sont des protections valides aussi contre les attaques en puissance simples. Il reste ainsi toujours possible de se localiser dans l'algorithme en localisant les mêmes motifs ; cependant, il devient impossible de déterminer des informations sur la clef grâce à la présence ou non d'un des motifs.

Ainsi, sur l'exemple du RSA donné plus tôt, l'application de cette protection impliquerait que la multiplication soit systématiquement réalisée, son résultat n'étant pas conservé dans les branches où l'opération est inutile. On obtiendrait ainsi une courbe de courant montrant une alternance régulière de multiplications et d'élévation au carré.

Les attaques en puissance différentielles sont plus complexes à contrer et nécessitent la mise en place de contre-mesures plus adaptées. Elles se répartissent en trois ensembles : les techniques de masquage, l'induction de bruit et le lissage du profil de courant.

Les contre-mesures de masquage sont basées sur la dissimulation des données manipulées pour éviter toute corrélation avec la puissance consommée pendant l'opération. Ces techniques n'influencent pas directement le profil de consommation mais les données qui seront traitées, donc la consommation de la fonction de sélection et l'utilité du calcul de cette fonction. L'opération de masquage consiste à modifier les entrées de la fonction de sélection en fonction d'une valeur aléatoire. Il existe deux sortes de masquages ; les masquages binaires et les masquages arithmétiques. Le masquage binaire est particulièrement adapté aux algorithmes de chiffrement utilisant exclusivement des opérateurs binaires, un tel masquage est défini par le OU exclusif entre l'entrée et la valeur aléatoire. Les masquages arithmétiques sont adaptés aux algorithmes de chiffrement utilisant des opérateurs arithmétiques ; un exemple de tel masquage est la différence entre l'entrée et le masque. Il est ensuite nécessaire en fin d'opération de démasquer le résultat. Un certain nombre de ces techniques de masquage peuvent être contournées par des attaques d'ordre supérieur. [Schr 06] introduit une nouvelle technique de protection : le masquage d'ordre supérieur. Cette technique a elle aussi ses faiblesses ; [Coro 07] et [Biry 07] montrent comment contourner des protections par masquage d'ordre supérieur.

[Akka 01] préconise l'utilisation de techniques de génération de bruit pour renforcer la sécurité. Le principe de cette contre-mesure est l'ajout de bruit dans le composant afin de rendre les mesures de courant inexploitables tout en conservant la fonctionnalité du circuit. Plusieurs méthodes sont disponibles pour ajouter ce bruit : l'utilisation de générateurs aléatoires ou l'introduction de jitter dans les signaux d'horloge ou d'alimentation. Ceci permet de rendre aléatoire l'exécution de certaines fonctions ainsi que les écritures dans les registres. Dans [Mull 01] et [Oswa 01], c'est une modification du profil de courant par déplacement de certains pics de consommation qui est visée. Ceci permet de désynchroniser la commutation de certaines portes permettant ainsi de réduire les pics de courant aux fronts d'horloge notamment et donc le rapport signal sur bruit.

Le dernier type de contre-mesure a pour objectif le lissage des courbes de courant. Il existe deux types de méthodes pour obtenir un tel résultat : réduire les dissymétries de consommation dans les portes logiques ou égaliser la consommation sur le composant. Au niveau portes logiques, l'objectif est de supprimer les dissymétries entre les charges et décharges lors de la transition des entrées et sorties d'une porte logique. [Tiri 02] présente une approche basée sur l'introduction de portes logiques et de bascules de type SABL (Sense Amplifier Based Logic) permettant ainsi la création de cellules dont la consommation est quasi indépendante de leur activité. Ceci vise à garantir dans la porte une consommation indépendante des données traitées. Il est aussi possible de réaliser de telles cellules en utilisant la technique WDDL (Wave Dynamic Differential Logic) pour atteindre cet objectif [Tiri 04b]. L'avantage du WDDL est qu'il se base directement sur les cellules de base des fondeurs. Cependant, ces deux techniques nécessitent une grande attention au placement et routage pour éviter de créer des dissymétries à ce niveau. Une méthode a été proposée pour y remédier dans [Tiri 04]. [Guil 05] propose une autre méthode basée sur le placement et routage d'une netlist sans redondance en laissant de l'espace nécessaire pour entrelacer la « copie » permettant l'uniformisation de la consommation. Cette technique à été optimisée pour limiter les effets de couplage par [Badd 08]. Les techniques de lissage du courant au niveau du circuit global sont basées sur l'ajout entre le circuit et les plots d'alimentation de filtres de courant. Ceci permet selon [Rake 01] de lisser le profil de courant.

1.2.3.3. Protections contre les attaques électromagnétiques

Les protections contre les attaques électromagnétiques sont de deux types. Le premier type de protection vise à réduire le niveau de la puissance rayonnée par les blocs de calcul au niveau du bruit ambiant. Ceci passe notamment par l'ajout de couches métalliques faisant office de boucliers électromagnétiques.

Le second type de protection est basé sur le brouillage des émissions du bloc de calcul attaqué. Le rayonnement électromagnétique au voisinage d'un circuit est une image de son activité (variation de courant dans les différents conducteurs). Ainsi, les techniques limitant l'exploitabilité de la consommation électrique affectent aussi le rayonnement électromagnétique. [Quis 02] présente d'ailleurs les protections contre ces deux types d'attaques comme appartenant à un même groupe.

1.2.3.4. Protections logicielles contre les attaques par canaux cachés

Les protections logicielles contre les attaques par canaux cachés sont basées sur les mêmes principes de corrélation minimale entre le déroulement d'un programme et les clefs de chiffrement utilisées et d'insertion d'aléas lors de l'opération. On retrouve donc l'ajout d'opérations parfois inutiles pour garantir l'équilibre entre les différentes branches, l'exécution aléatoire de certaines instructions.

1.3. Conclusion

Dans notre vie quotidienne, nous voulons de plus en plus que la sécurité des services auxquels nous avons accès soit garantie. Ceci implique l'usage intensif de systèmes sur puces dédiés, ayant la capacité de réaliser toutes les opérations cryptographiques nécessaires à cette tâche. La difficulté pour un utilisateur de retenir des clefs de chiffrement impose aux concepteurs de tels systèmes de stocker ces données confidentielles en leur sein. Ces données secrètes deviennent donc la cible prioritaire de pirates voulant s'octroyer l'accès à ces services.

Les méthodes à leur disposition sont multiples et nombreuses. La plupart d'entre elles ne demandent pas d'investir dans des équipements de pointe et excessivement coûteux. Nous avons vu ici que chaque attaque est rendue plus difficile par une méthode de protection appropriée. Ces protections ne les rendent pas impossibles, et régulièrement les attaques sont affinées pour passer outre les protections existantes, nécessitant ainsi le développement de nouvelles protections.

Les protections ainsi développées sont efficaces contre l'attaque qu'elles ciblent, mais qu'en est-il vis-à-vis des autres types d'attaques ? Se protéger d'un coté n'ouvre-t-il pas de brèche sur un autre front ? C'est l'une des questions qui sera abordée par la suite. Mais avant cela, nous allons étudier plus en détails le type d'erreurs pouvant être induites par l'une des attaques par fautes actuellement les plus efficaces.

Chapitre 2 : Réalisation d'attaques par fautes au laser

Nous avons vu qu'une des méthodes d'induction de fautes les plus utilisées est l'utilisation de faisceaux laser pour illuminer certaines parties du circuit que l'on veut attaquer. Dans ce chapitre nous allons étudier les effets de ces tirs sur, d'une part, un circuit sécurisé contre les fautes réalisé sur une technologie ASIC, et d'autre part, les risques supplémentaires qui peuvent surgir si on choisit d'utiliser une technologie de circuits programmables. Nous allons donc brièvement présenter les deux plates-formes laser utilisées lors de ces campagnes puis analyser les résultats des expériences. Le premier circuit de test est un accélérateur matériel pour la multiplication modulaire protégé contre les fautes ; le second est un circuit programmable de type FPGA programmé par une mémoire SRAM.

2.1. Présentation des plates-formes

2.1.1. Plate-forme ATLAS (Laboratoire IMS)

La plate-forme utilisée lors des campagnes d'injections de fautes par laser au laboratoire IMS de Bordeaux est basée sur l'association de deux instruments : un environnement d'émulation vérifiant le statut du circuit attaqué et un environnement contrôlé d'injection de fautes par laser. La photographie Figure 17 montre le circuit attaqué positionné sur cette plate-forme.

Lors de cette étude, une version optimisée de la plate-forme de test présentée dans [Faur 02] a été utilisée : Testbed for Harsh Environment Studies on Integrated Circuits (THESIC). Elle est basée sur l'utilisation de deux FPGAs. Le premier, nommé FPGA de communication (COM FPGA) implémente un processeur LEON2 ainsi que tous les modules nécessaires pour la communication entre l'ordinateur hôte, via un port Ethernet, et les différentes ressources du système. De plus, il contrôle le courant consommé par le circuit attaqué pour le protéger contre les latch-up. Le second, Chipset FPGA, sert quasi-exclusivement d'interface avec le circuit attaqué : il contrôle notamment sa programmation et l'activation du tir laser.

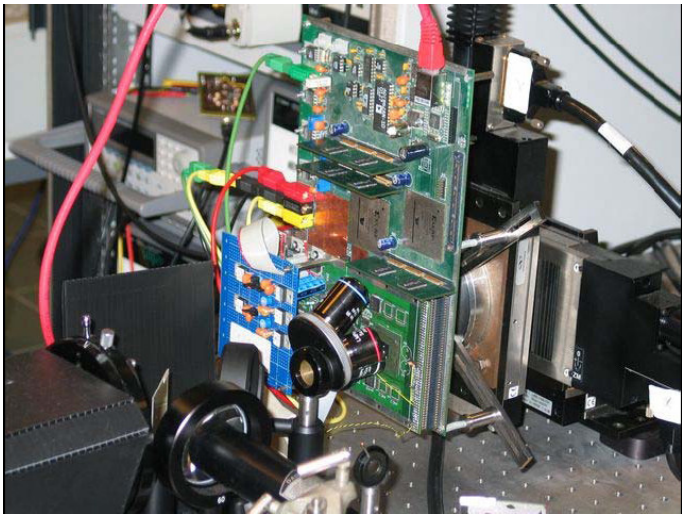

Figure 17 : Circuit attaqué mis en place sur la plate-forme ATLAS

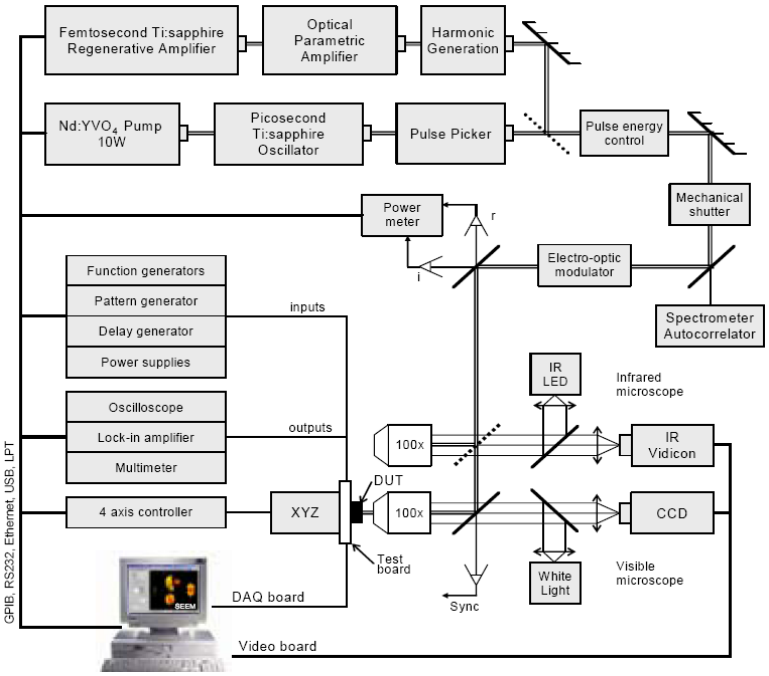

Figure 18 : Architecture de la plate-forme ATLAS (issue de [Poug 06])

La plate-forme ATLAS du laboratoire IMS de l'université de Bordeaux est dédiée au test par laser et à l'analyse des circuits intégrés. Cette plate-forme est présentée dans [Poug 06]. Elle est composée de deux sources laser à impulsions, permettant la mise en place de diverses mesures optiques, et d'un environnement de test et d'émulation de circuits intégrés. La Figure 18 présente l'organisation de cette plate-forme de stimulation photoélectrique par laser (PLS). Pour cette étude la source laser est un oscillateur Ti:Saphir générant des impulsions d'une durée de 1ps à une fréquence de 80 MHz. Un sélecteur d'impulsion permet de sélectionner une seule impulsion pour réaliser un tir

unique. La longueur d'onde est réglable entre 780nm et 1000nm, permettant d'adapter le faisceau à la profondeur de pénétration nécessaire. Le faisceau est focalisé par des objectifs de microscope, permettant ainsi d'ajuster le diamètre du spot illuminant le circuit attaqué, entre 1µm et 20µm. L'énergie de l'impulsion est aussi ajustable jusqu'à une valeur de 1nJ. Le laser est positionné sur le circuit attaqué par une table XYZ ayant un pas de positionnement de 100nm. Un ordinateur contrôle les campagnes d'injection en pilotant la table XYZ et le sélecteur d'impulsions grâce à un logiciel dédié : le SEEM [Poug 06b].

2.1.2. Plate-forme Gemalto

La plate-forme laser Gemalto a été utilisée pour réaliser des attaques laser sur un multiplieur de Montgomery. La Figure 19 présente la mise en place sous le laser du démonstrateur et de la carte de développement le pilotant durant les expérimentations. La plate-forme est constituée de différents éléments ; un ordinateur contrôle à la fois le démonstrateur, une table XY et le laser. Un oscilloscope optionnel permet de vérifier le fonctionnement du circuit, pour s'assurer qu'il reçoit les bonnes commandes et fournisse un résultat. La table XY permet de contrôler de manière précise la position du circuit sous le laser. Le laser utilisé pour ces campagnes d'injection est un laser Yag à impulsions. Les impulsions ont une durée de 6 ns et le faisceau de sortie a une longueur d'onde de 532nm (laser vert). L'énergie du laser est réglable relativement à sa puissance maximale (entre 0 et 100%), le diamètre du faisceau est aussi configurable.

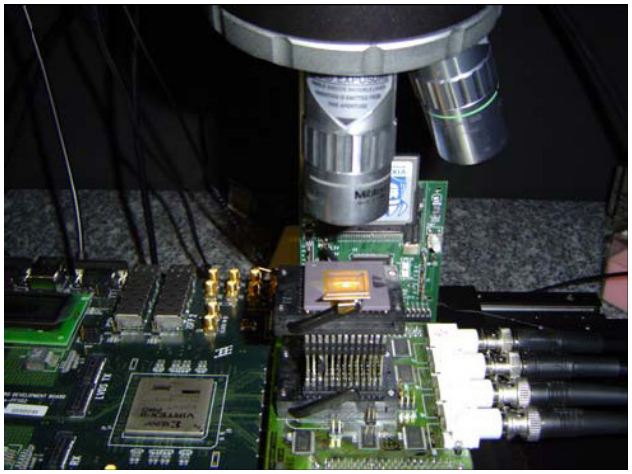

Figure 19 : Circuit à tester en place sur la plate-forme laser Gemalto

Durant les campagnes de tirs, l'énergie du laser est réglée sur 0% ; malgré ce réglage l'énergie transmise au circuit lors du tir est suffisante pour causer en plus de l'induction optique de faute une

élévation de la température de la zone touchée de quelques dizaines de degrés. Ceci peut entraîner l'existence de facteurs multiples de génération de fautes. Ceci n'est pas un problème pour notre étude, puisqu'elle ne vise pas les sources de fautes mais leurs effets.

Les tirs laser ont été réalisés en face avant ; c'est-à-dire que le faisceau doit réussir à influencer les transistors à travers les différentes couches métalliques du circuit qui se comportent comme des écrans.

2.2. Attaque d'un multiplieur modulaire de Montgomery

2.2.1. Présentation du circuit étudié

Le premier circuit qui sera étudié est un cœur de coprocesseur réalisant des multiplications modulaires de Montgomery. Ce cœur peut être utilisé comme accélérateur matériel pour le chiffrement et déchiffrement de messages en utilisant l'algorithme RSA. La multiplication de Montgomery est définie par l'équation suivante : $M(A,B,N) = A * B * R^{-1} \bmod N$ où A, B et N sont des entiers codés sur n bits. A, B et N sont respectivement les multiplieur, multiplicande et modulo ; R est défini par $R = 2^n$. De plus, le multiplieur et le multiplicande sont tous deux inférieurs au modulo, qui doit être impair. Pour notre étude, le circuit considéré permet de réaliser des opérations sur 512 bits.

Le multiplieur est décrit au niveau transfert de registre (RTL) en VHDL. Le cœur de calcul est une matrice systolique de 512 cellules constituées chacune d'additionneurs et de quelques portes logiques. Ce cœur est contrôlé par de la logique séquentielle et est alimenté en entrée par un registre à décalage, permettant la rotation de l'opérande A un cycle sur deux. Les données en entrée et sortie sont stockées dans des registres 512 bits connectés au reste du système par une simple interface AMBA AHB Lite. Le multiplieur de Montgomery peut être utilisé comme un périphérique esclave connecté à un bus AMBA [AMBA]. Les données sont transférées par sous-mots de 32 bits. La Figure 20 résume les différentes phases du calcul. 3*n cycles sont nécessaires au cœur combinatoire pour réaliser l'opération, soit 1536 cycles d'horloge. En incluant les transferts avec le reste du système, l'opération dure 1739 cycles.

Le prototype qui a été conçu et fabriqué pour ces expériences inclue trois versions du multiplieur. La version de référence (Référence) est une version non protégée du circuit, capable uniquement de réaliser des opérations simples. Les deux autres versions possèdent les mêmes protections contre les fautes. La version Protégée1 est le résultat de l'ajout de la protection choisie à la version de référence. La version Protégée2, quant à elle, est le résultat de la sécurisation d'une version

optimisée de la référence. Ces optimisations ont pour but d'accroître ses performances en pipelinant les transferts au niveau système, elle embarque donc des registres supplémentaires.

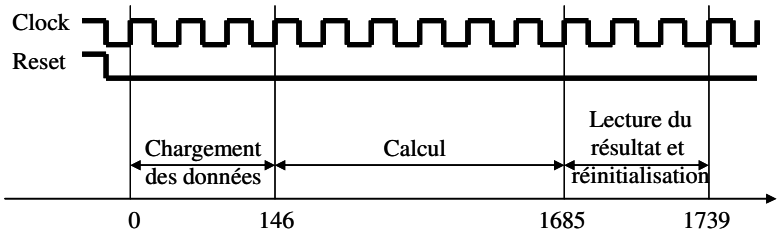

Figure 20 : Chronogramme d'une multiplication de Montgomery

L'ajout de toutes les protections a été réalisé sur la description RTL du circuit. Tous les registres du circuit ont été protégés en utilisant un code de parité et des blocs de vérification en double rail. Pour obtenir un bon compromis entre le coût et les capacités de détection de fautes multiples, un bit de parité est ajouté pour chaque sous-mot de 32 bits. Ainsi, un registre de données de 512 bits est protégé par 16 bits ; les registres de taille inférieure ou les bascules isolées ont été regroupés en mots de 32 bits pour être protégés. Le registre d'état de l'interface AMBA AHB a, quant à elle, été protégée en utilisant un codage en un parmi n. Le cœur de calcul combinatoire à lui aussi été protégé en utilisant un code de parité avec une vérification en double rail. La Figure 21 présente comment cette protection a été implantée.

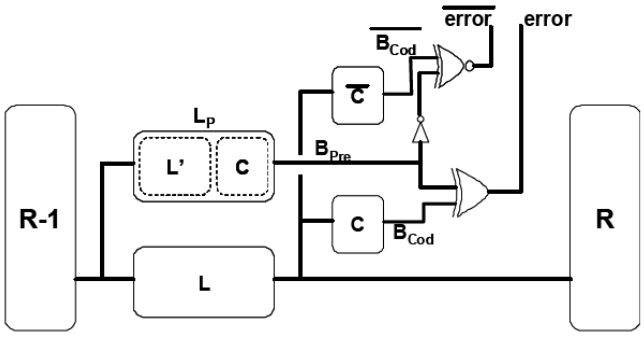

Figure 21 : Principe de prédiction avec vérification en double rail [Port 06]

On retrouve ici le concept présenté précédemment ; cependant, il a été modifié pour ajouter une vérification en double rail. La sortie du bloc de prédiction est comparée au codage de la sortie du bloc protégé par deux vérificateurs duaux générant deux signaux d'erreurs. Ces deux signaux sont complémentaires en l'absence de fautes ou si les fautes ont eu lieu dans le bloc de prédiction ou de calcul original ; si les fautes sont apparues dans un des vérificateurs, on obtient alors une alarme dissymétrique.

Une paire de signaux d'erreur est ainsi générée pour chaque sous-mot protégé. Ces signaux sont ensuite regroupés en fonction du bloc fonctionnel d'où ils proviennent. On crée ainsi 18 alarmes pour la version Protégée1, respectivement 24 pour la version Protégée2. Ces alarmes sont stockées dans deux registres complémentaires Detect0 et Detect1, contenant respectivement uniquement des 0 et des 1 en l'absence de fautes. Du fait de la taille des données transmises par l'interface AMBA AHB, le code d'alarme en sortie est sur 32 bits dont seuls les bits de poids faibles sont significatifs. Les registres de détection conservent leur valeur en cas de détection jusqu'à une réinitialisation du circuit. Un signal d'alarme est généré dès qu'un des bits de ces registres signale la présence de fautes. Tous les blocs de vérification, les codeurs, les registres de détection ainsi que les registres et blocs combinatoires protégés sont implantés dans des entités distinctes du modèle VHDL, permettant ainsi à une synthèse hiérarchique de ne pas éliminer les blocs redondants. La Figure 22 présente l'architecture globale de la version Protégée1.

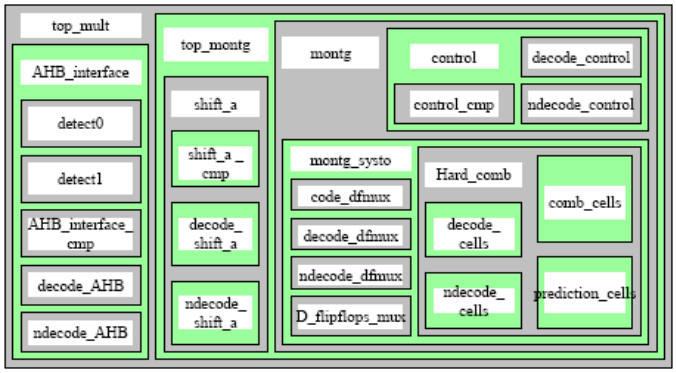

Figure 22 : Architecture globale de la version Protégée1

Les trois versions du multiplieur ont été implantées sur un circuit en utilisant la technologie ST HCMOS 130nm possédant six niveaux de métal. La Figure 23 montre le layout du circuit contenant les trois versions du multiplieur. La seule contrainte donnée lors des étapes de placement et routage est la séparation des registres de détection Detect0 et Detect1, dans le but d'éviter qu'un tir laser ne touche les deux registres simultanément. Le Tableau 1 présente la comparaison en termes de complexité des différentes versions. Le fort surcoût de la version Protégée1 peut s'expliquer par la complexité des blocs de prédiction de parité dans le cœur de calcul, pour lesquelles l'optimisation n'a pas été totale. Le Tableau 2 résume la correspondance entre les différentes alarmes et les blocs fonctionnels auxquels elles sont rattachées.

Tableau 1 : Complexité des différentes versions du multiplieur

Circuit	Surface (mm²)	# cellules	# portes équivalentes
Référence	0,434	26 589	55 751
Protégée1	0,827	46 084	116 796
Protégée2	1,282	59 600	160 849

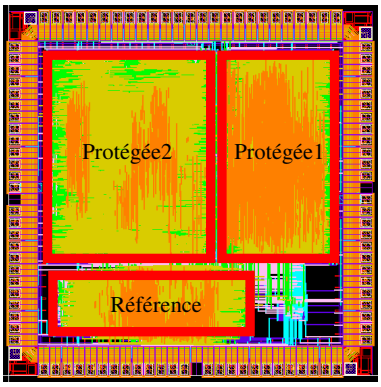

Figure 23 : Layout du circuit

Tableau 2 : Affectation des différentes alarmes aux principaux blocs fonctionnels

Fonction	Vers. Protégée1	Vers. Protégée2
hard_comb	Alarmes 8 à 10	Alarmes 13 à 15
montg_systo	Alarmes 11 à 15	Alarmes 16 à 20
control	Alarme 16	Alarmes 21 et 22
shift_a	Alarme 17	Alarme 23
AHB_interface	Alarmes 0 à 7	Alarmes 0 à 12

En termes de niveau de protection, le but n'était pas de protéger totalement le co-processeur contre un type donné de fautes, mais d'avoir un bon compromis entre le coût et le détection de fautes quelconques. Pour des fautes affectant directement les registres, le taux de détection est de 100% pour les fautes de multiplicité impaire. Pour les fautes de multiplicité paire, le taux de détection dépend de leur répartition au sein des différents sous-mots ; il suffit qu'un seul des sous-mots soit touché par un nombre impair de bits faux pour qu'une alarme soit levée. La probabilité d'erreur dans le cœur combinatoire dépend du nombre de sorties erronées et donc de sa structure logique. Dans cette étude, la synthèse a été réalisée en utilisant un flot standard sans limitation du partage de ressources lors de l'optimisation structurelle. De ce fait, une faute simple sur un nœud interne peut tout aussi bien être détectée ou non en sortie, selon le nombre de bits de sortie que cette faute altère. Contrôler la structure générée ne serait pas plus efficace étant donné qu'un tir laser peut illuminer une large zone touchant ainsi plusieurs cône logiques indépendants. Nous n'avons pas contraint le placement et le routage,

facilitant ainsi l'induction de fautes dans plusieurs éléments simultanément, accroissant ainsi la probabilité que l'un d'eux détecte une erreur.

Dans le cas de notre étude, le circuit ciblé ne présente a priori aucune zone plus sensible que les autres, du point de vue de l'application réalisée. En effet, tout résultat erroné peut être utilisé pour compromettre la sécurité globale de l'application. En conséquence, il a été décidé que la campagne d'injection serait basée sur des balayages complets du circuit tout au long de son calcul. Ceci est en quelque sorte une attaque en boite noire.

Toutes les campagnes de tirs laser sur ce circuit ont été réalisées sur la plate-forme d'injection de Gemalto. Pour conserver des temps d'expérimentation raisonnables, il a fallu réduire le nombre de valeurs possibles ; ainsi, le balayage du circuit sera fait dix fois à des instants uniformément répartis sur la durée d'exécution -entre le cycle 0 et le cycle 1350, donc tous les 150 cycles. De la même manière, les mêmes vecteurs d'entrée A, B et N ont été conservés tout au long de la campagne et la surface de chaque version du circuit a été découpée en plusieurs zones. Le circuit de référence a été divisé en 27 zones (9 positions en X et 3 positions en Y) ; les versions protégées ont, quant à elles, été séparées en 45 zones (5 positions en X et 9 positions en Y) et 63 zones (7 positions en X et 9 positions en Y) pour respectivement les versions Protégée1 et Protégée2. Chaque zone ainsi créée a pour forme un carré de 145 microns de côté, dont le centre correspond à la position du tir.

Pour pouvoir étudier le déterminisme des effets des tirs laser sur le circuit, il a été décidé de réaliser 5 tirs sur chaque position spatio-temporelle précédemment définie. Ceci représente 1350, 2250 et 3150 tirs sur respectivement les versions non protégée, Protégée1 et Protégée2.

2.2.2. Analyse des résultats

Lors de la réalisation des campagnes de tir sur les différentes versions du circuit, on obtient des fichiers de rapport de campagne. Ces fichiers sous un format textuel contiennent toutes les informations sur les tirs laser réalisés : position spatio-temporelle, statut du circuit en fin du temps d'exécution et valeur de sortie ou du registre de détection. Le statut est défini en fonction des effets observés du tir et sert de classification :

- Résultat faux non détecté : le circuit fournit un résultat erroné mais ne signale aucune alarme.
- Résultat faux détecté : le circuit fournit un résultat erroné et au moins une alarme.
- Aucun résultat : le calcul s'est bloqué et aucun résultat n'est retourné.
- Aucun effet : le circuit fournit le résultat attendu sans alarme.

Pour éviter la propagation d'effets latents d'une exécution à une autre, le circuit est réinitialisé entièrement entre deux tirs laser.

La classification des résultats obtenus lors de ces campagnes de tirs est résumée dans le Tableau 3. Il est à noter que, pour les trois versions du circuit, aucun tir n'a pu être classifié comme n'ayant aucun effet. Pour la version de référence, tous les tirs ont engendré soit la production d'un résultat erroné en sortie du circuit, soit le blocage du circuit (et l'absence de résultat). Parmi les 2250 tirs effectués sur la version Protégée1, seulement 6 tirs n'ont pas été détectés et ont généré un résultat faux ; on note l'absence de tirs ayant généré un blocage. Sur la version Protégée2, on remarque que les résultats sont très similaires : on observe 10 tirs non détectés donnant un résultat faux sur 3150 et l'absence de blocage.

Tableau 3 : Classification des effets de tirs sur les trois versions

Version	Résultats Faux non Détecté	Résultats Faux Détecté	Aucun Résultat	Aucun Effet
Référence	89,58 %	0,00 %	10,42 %	0,00 %
Protégée1	0,27 %	99,73 %	0,00 %	0,00 %
Protégée2	0,32 %	99,68 %	0,00 %	0,00 %

Ces chiffres pourraient être interprétés comme de bons résultats. Cependant, le protocole d'attaque utilisé était très simple ; obtenir 6 ou 10 résultats erronés sans levée d'alarme montre que le circuit est encore vulnérable. De plus, si on considère que les tirs laser ont été réalisés avec une énergie « nulle » et en face avant, donc en présence des 6 couches de métal comme écran, cette constatation devient encore plus alarmante. Les résultats obtenus ne seront peut être pas exploitables lors de la cryptanalyse ; il est cependant nécessaire pour le concepteur d'éviter ce genre de vulnérabilités.

Pour tenter de comprendre comment le circuit réagit aux tirs laser, nous allons étudier en détail plusieurs aspects de leurs effets. Nous étudierons dans un premier temps la levée des alarmes du point de vue de la multiplicité d'erreurs ainsi que de la manière dont elles sont levées. Nous analyserons ensuite la sensibilité aux fautes des différents sous-blocs du circuit ainsi mise en évidence et la reproductibilité de ces expériences. Enfin, nous verrons comment l'analyse des informations de placement peut nous aider à comprendre les phénomènes de non détection et dans quelle mesure ces informations nous permettent de prédire a priori la levée des alarmes.

Multiplicité d'erreurs

Les résultats précédents indiquent clairement que des fautes de multiplicité paire ont été générées dans les deux versions protégées du circuit. Une multiplicité d'erreurs paire sur les registres touchés (ici, les sous-mots touchés) ou en sortie du cœur de calcul combinatoire est la seule configuration d'erreurs ne permettant pas la détection. Il nous est impossible de déterminer avec certitude si une faute dans un registre est due à son inversion par le tir ou à la mémorisation d'un aléa dans le cône de logique combinatoire précédant ce registre. De même, le nombre exact de fautes est inconnu et ne peut être déterminé par cette étude. Il a été relativement facile de générer de telles configurations d'erreurs ; ce fait doit être pris en compte lors de l'implantation de protections. Une analyse croisée avec les données de placement et de routage, détaillée plus tard dans cette section, nous donne quelques pistes sur la sensibilité de ces zones.

Durant ces expériences, nous n'avons observé que très peu de tirs pour lesquels une seule alarme a été levée. Ceci confirme bien la présence de fautes multiples suite au tir. Le placement a été réalisé pour maximiser le nombre de blocs illuminés par un même tir et donc accroître la probabilité de détection par au moins l'un d'entre eux. De plus, le rayon du faisceau laser était suffisamment élevé pour illuminer une zone assez large. Il est cependant surprenant de voir le nombre d'alarmes qui peuvent être levées par un seul tir ; la Figure 24 présente ces résultats. Pour la version Protégée1, nous avons pu lors de certains tirs lever 13 alarmes parmi les 18 présentes, pour un nombre moyen sur l'ensemble de la campagne de 5,48 alarmes par tir. De même, pour la version Protégée2, on trouve parfois jusqu'à 15 alarmes levées sur les 24 présentes, pour une moyenne de 6,75 alarmes par tir.

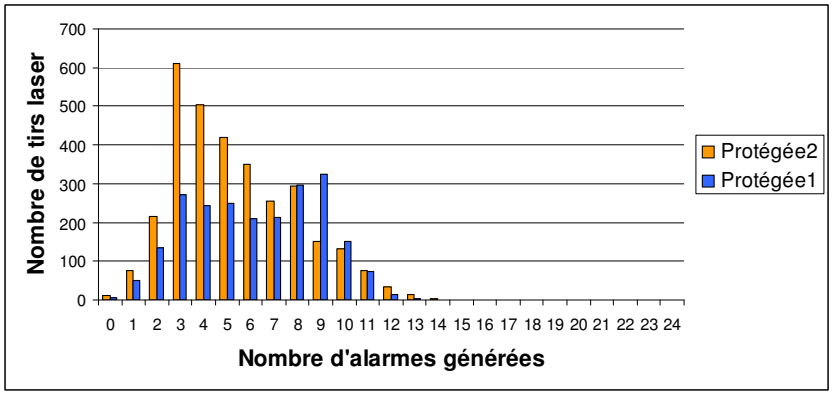

Figure 24 : Répartition des tirs en fonction du nombre d'alarmes levées

La Figure 25 présente comment la répartition des tirs en fonction du nombre d'alarmes levées évolue avec l'instant de tir. On constate sur ce graphe que la forme observée lors de l'étude de la répartition totale est globalement conservée à chaque instant d'injection. On note cependant quelques variations autour de la moyenne pouvant atteindre 25 à 30 %. Ceci tend à montrer une dépendance

faible entre l'instant du tir et ses effets. Ces remarques sont aussi valides pour l'étude de ce phénomène sur la version Protégée2 ; on note cependant une variation plus faible dans ce cas.

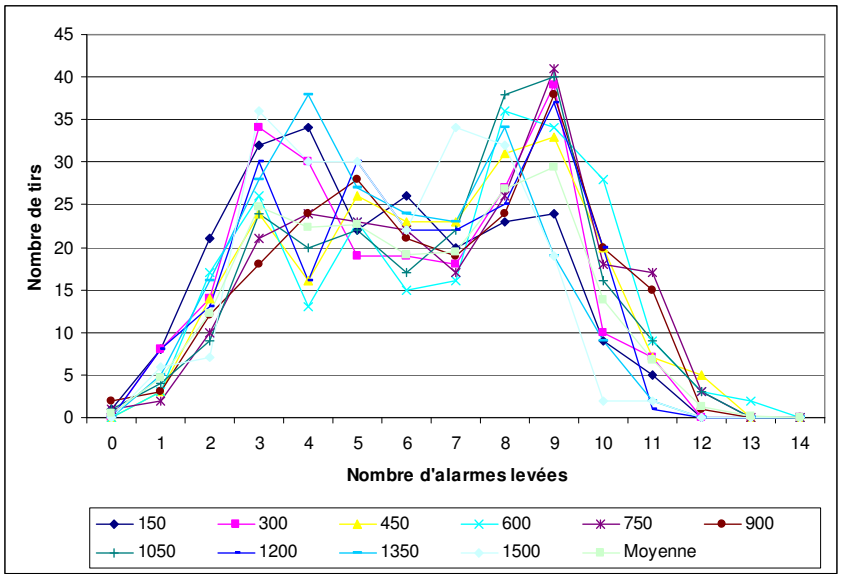

Figure 25 : Evolution temporelle de la répartition des tirs pour la version Protégée1

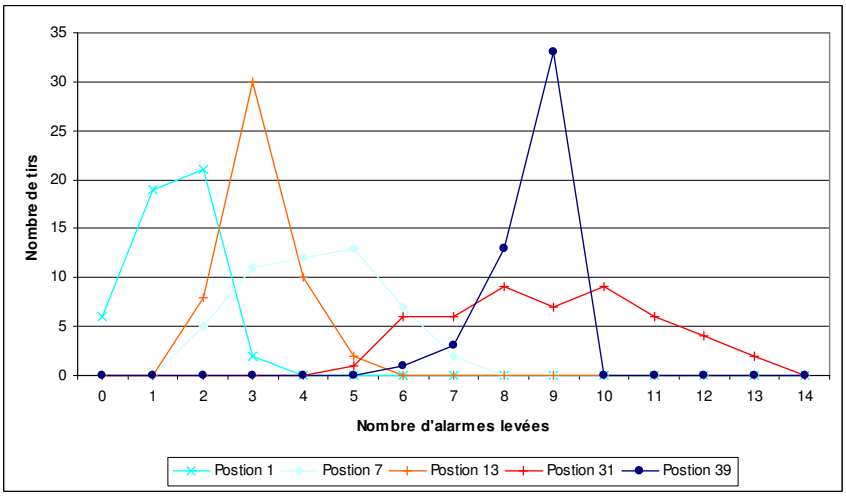

Figure 26 : Influence de la position de tir sur leur répartition pour la version Protégée1

On peut, de la même manière, analyser la répartition des tirs en fonction du nombre d'alarmes levées pour les différentes positions. La Figure 26 présente cette répartition pour quelques positions de tir sur la version Protégée1. On remarque que contrairement à l'instant de tir, la position influe grandement sur la forme de la répartition. Pour chaque position de tir, on trouve une répartition s'approchant d'une loi normale centrée sur une valeur de multiplicité d'erreur. Cependant, les paramètres de cette distribution (la valeur centrale, la hauteur du pic et sa largeur) sont très variables

d'une position de tir à une autre. Les mêmes résultats sont observables lors des campagnes de tirs sur la version Protégée2. Ceci montre une dépendance entre la position du tir et donc les portes sous le laser et les alarmes levées.

Dissymétries dans les registres de détection

Dans la majorité des cas, la levée d'une alarme correspond à la détection d'erreurs dans le bloc fonctionnel. Dans ces cas, on obtient bien des valeurs complémentaires en sortie des blocs de vérification en double rail ; ainsi, les registres Dectect1 et Detect0 sont complémentaires. Ces alarmes symétriques reflètent une activation « normale » de la logique de détection. Cependant, certains tirs ont entraîné la levée d'alarmes dans un seul des deux registres de détection. La Figure 27 montre la répartition des tirs en fonction du nombre d'alarmes levées asymétriquement. Ceci signifie qu'une erreur a eu lieu soit dans l'un des registres de détection, soit sur l'un des deux rails de vérification. Sur la version Protégée1 (respectivement Protégée2), 30,6 % (respectivement 23,3 %) des tirs ont entraîné la génération d'au moins une alarme asymétrique. Dans certains cas, le nombre de ces alarmes peut être très élevé, jusqu'à 7 (respectivement 9) alarmes asymétriques pour la version Protégée1 (respectivement Protégée2) avec une moyenne de 0,48 (respectivement 0,42) alarme asymétrique par tir.

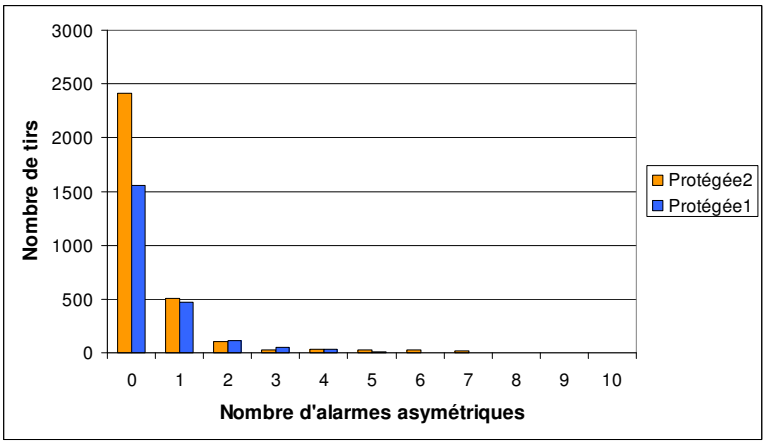

Figure 27 : Répartition des tirs en fonction du nombre d'alarmes asymétriques levées

Sur la version Protégée1, tous les tirs ayant été détectés ont créé au moins une alarme symétrique. Cependant, pour la version Protégée2, un des tirs n'a généré qu'une alarme asymétrique, prouvant ainsi que malgré la faible focalisation du laser, il est possible de n'impacter qu'un petit bloc logique (un des rails d'un des vérificateurs). Ainsi, on ne peut garantir la détection de tous les tirs laser par la simple répartition uniforme des mécanismes de détection sur la surface du circuit. Pour limiter ces risques, la densité des éléments de protection doit être élevée et donc nécessiter un coût important.

Le Tableau 4 présente la répartition des bits de détection des deux versions protégées du circuit entre les différentes configurations d'alarmes sur l'ensemble des tirs. Les différentes configurations possibles sont : « 10 » en cas de non activation, « 01 » pour une activation symétrique et « 00 » et « 11 » pour les activations dissymétriques. On remarque un taux d'activation de 33,09% pour la version Protégée1 (respectivement 22,30% pour la version Protégée2) ; ceci est cohérent avec le nombre moyen d'alarmes levées par tir de 5,48 alarmes parmi 18 (respectivement 6,75 parmi 24). Le faible pourcentage d'alarmes levées de manière asymétrique tend à montrer qu'il est difficile de n'impacter qu'une des branches de vérification. Cependant, si on analyse leur répartition, on constate qu'une part des tirs bien plus importante génère au moins une alarme asymétrique : en effet, ce cas se produit pour 30,60% des tirs sur la version Protégée1 et 23,25% sur la version protégée2. Ceci montre que malgré le faible pourcentage d'activation asymétrique d'une alarme et la faible taille des branches de vérification, il est relativement aisé de causer des alarmes asymétriques, touchant donc ces blocs de vérification. Il est aussi à noter que très peu de tirs n'ont entraîné que la levée d'alarmes asymétriques, puisque ce cas ne s'est produit que pour un tir sur la version Protégée2. Ceci montre que bien qu'il soit facile d'induire des fautes asymétriques, il est très difficile de ne générer que ce type de fautes surtout avec le niveau de focalisation utilisé durant cette étude.

De plus, on peut constater une différence de probabilité d'apparition des deux configurations asymétriques. Dans les deux cas, la probabilité d'activation asymétrique des bits du registre Detect1 est supérieure à celle des bits du registre Detect0. Ceci tend à montrer une sensibilité accrue des bits du registre Detect1. Les campagnes n'incluent pas de tirs assez précis et focalisés sur ces éléments pour permettre de déterminer avec certitude la source de cette sensibilité accrue. Cette différence pourrait très bien être due à des décharges d'énergie plus importantes sur l'un des deux registres.

Tableau 4 : Répartition des bits de détection selon leur configuration

Configuration	« 10 »	« 01 »	« 00 »	« 11 »	Levée	Levée asymétriquement
Protégée1	66,91 %	30,69 %	1,56 %	0,84 %	33,09 %	2,40 %
Protégée2	77,70 %	20,57 %	0,93 %	0,81 %	22,30 %	1,74 %

Sensibilité des différents blocs

La Figure 28 montre le pourcentage de tirs ayant activé chaque alarme pour les 2 versions protégées. On remarque sur ce graphe que les alarmes peuvent être classées en trois groupes en fonction de leur taux d'activation : inférieur à 15%, entre 20% et 60% et au delà de 70%.

Pour les deux versions protégées du circuit, on note la présence d'une seule alarme présentant un taux d'activation supérieur à 70% : alarmes 8 et 13 pour respectivement les versions Protégée1 et

Protégée2. Dans les deux cas, cette alarme indiquant la présence d'erreurs dans un des mots de sortie du bloc combinatoire du cœur de calcul systolique est levée dans près de 95% des cas. On peut noter qu'il s'agit dans les deux cas du même mot, correspondant à une valeur intermédiaire de calcul propagée dans le réseau systolique ; les autres sorties ont un taux beaucoup plus proche du comportement moyen (alarmes 9 et 10, respectivement 14 et 15).

Les alarmes à faible taux d'activation (inférieur à 15%) représentent à peu près un quart des alarmes. Elles protègent pour les deux versions protégées du circuit le bloc de contrôle, le registre à décalage, l'interface AMBA et quelques registres d'entrée et de sortie.

Les autres alarmes, correspondant aux autres éléments protégés, présentent un taux d'activation variable plus centré autour du taux moyen d'activation. Il s'agit de certaines sorties du bloc combinatoire et des registres internes du bloc de calcul et une partie des registres d'entrée et de sortie.

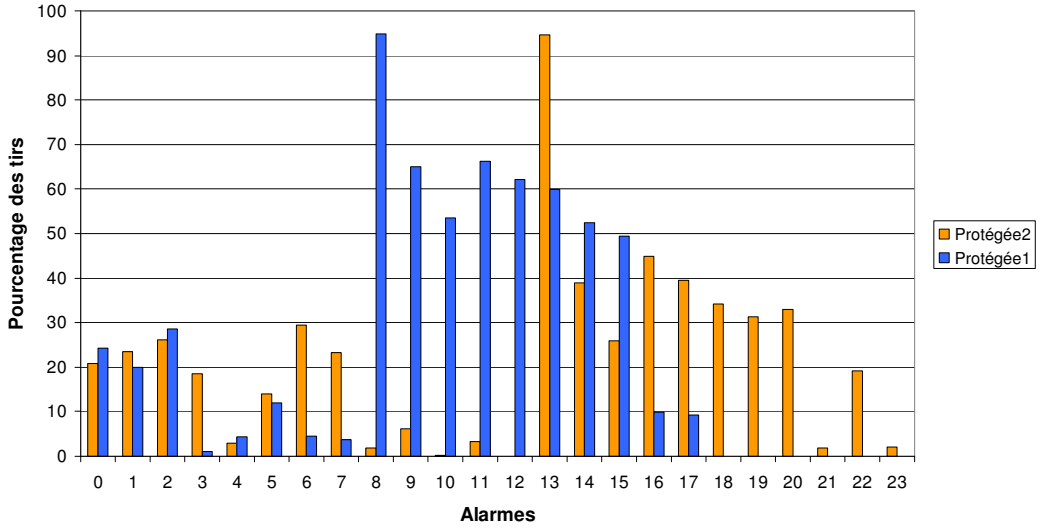

Figure 28 : Taux d'activation des différentes alarmes

Cette classification montre une sensibilité accrue des éléments fonctionnels au cœur de l'algorithme utilisé pour réaliser la fonction globale du circuit. En effet, toutes les alarmes présentant un taux d'activation supérieur à 30% sont liées à des éléments fonctionnels du cœur de calcul du circuit. Le fort taux d'erreur de ces blocs doit être mis en relation avec la surface qu'ils occupent sur le circuit final. Nous allons donc tenter d'étudier le lien entre la surface d'un bloc fonctionnel et son taux de perturbations par les tirs laser. La surface des différents blocs est dans un premier temps estimée par deux critères : le nombre de portes qu'il inclut et la surface du plus petit rectangle incluant toutes les portes du bloc. Ensuite, nous comparerons avec la surface réellement occupée par les portes

construisant chaque bloc. Le Tableau 5 présente la surface occupée par les principaux blocs relativement au circuit complet en utilisant les trois critères établis.

D'après le Tableau 5, le nombre de portes et la surface réelle sont deux critères donnant des taux d'occupation du même ordre de grandeur pour les différents blocs. On note tout de même des variations entre ces deux évaluations pouvant atteindre 10%. Pour les blocs les plus complexes, tels que ceux du cœur systolique, on remarque que le nombre de portes surévalue leur surface. Pour le bloc montg_systo\hard_comb, inclus dans le bloc montg_systo (alarmes 8 à 10 pour la version Protégée1 et 13 à 15 pour Protégée2), ces deux évaluations donnent un taux de 48% et 36% pour la version Protégée1 (respectivement 37% et 25% pour la version Protégée2). Dans le même temps, le graphe précédent donne un taux d'activation moyen pour ce bloc d'environ 65% (respectivement 50%). On observe donc une forte sous-évaluation du taux d'activation des alarmes dans ce bloc par ces deux critères. Cette constatation reste aussi valide si on s'intéresse aux blocs control (alarmes 16 pour Protégée1 et 21 à 22 pour Protégée2) et shift_a (alarme 17 pour Protégée1 et 23 pour Protégée2). On note cependant que la sous-évaluation pour ces deux blocs est moins importante pour la version Protégée2. Si on considère le bloc de calcul complet montg_systo, l'évaluation est beaucoup plus proche du taux d'activation des alarmes réellement observé pour les deux versions. A l'inverse, l'étude de la surface pour le bloc d'interface AHB surévalue totalement le taux d'activation des alarmes pour ce bloc.

Tableau 5 : Evaluation de la surface des principaux blocs en pourcents de la surface totale du circuit

	Protégée1			Protégée2		
	# portes	Plus petit rectangle	Surface réelle	# portes	Plus petit rectangle	Surface réelle
montg_systo\ hard_comb	47,87 %	100 %	36,12 %	36,84 %	100 %	25,23 %
montg_systo	69,90 %	100 %	68,61 %	55,04 %	100%	50,51 %
control	0,25 %	0,79 %	0,28 %	4,74 %	26,66 %	5,82 %
shift_a	1,10 %	7,15 %	1,00 %	0,98 %	4,52 %	0,73 %
AHB_interface	28,06 %	91,37%	29,55 %	38,62 %	91,39 %	42,54 %

Considérons maintenant la surface de ces blocs comme étant la surface du plus petit rectangle contenant toutes les portes de ces blocs. Les coordonnées de ce rectangle sont définies comme les bornes supérieures et inférieures sur l'ensemble des coordonnées de chaque porte. Etant donné l'absence de contrainte lors du placement, cette surface est très largement supérieure à la surface réelle du bloc. Pour tous les blocs, cette méthode surévalue très largement le taux d'activation des alarmes

des différents blocs. On peut cependant noter que les cas d'alarmes ayant un taux d'activation observé anormalement élevé se rapprochent de cette évaluation. Ce critère d'évaluation permet a priori de déterminer quelles positions spatiales sont susceptibles de lever chaque alarme. Cependant, en comparant la zone définie par ce rectangle pour un bloc donné et les positions ayant généré une levée d'alarme sur ce bloc, trois cas se présentent : une forte proportion des tirs recouvrant en partie ce rectangle ont effectivement levé une alarme (exemple : alarme 8 de Protégée1), seules quelques positions dans ce rectangle ont levé l'alarme (exemple : alarme 7 de Protégée1) et certaines positions hors du rectangle activent l'alarme (exemple : alarme 16 de Protégée1). La Figure 29 présente la localisation géographique des tirs levant certaines alarmes de la version Protégée1 ; elle fait aussi apparaître l'ensemble des positions partiellement incluses dans la surface des blocs associés.

De manière générale, les blocs les plus gros sont plus sensibles aux tirs laser. Nous avons vu que la surface d'un bloc n'est pas suffisante pour avoir une idée précise du taux d'activation que ses alarmes vont présenter lors des tirs laser. On note aussi que la répartition géographique des portes constituant un bloc joue aussi sur cette sensibilité. Cependant, il n'est pas possible de dégager de règle générale quant au lien entre cette surface minimale englobant le bloc et son taux d'activation d'alarmes. Nous avons présenté deux cas où ces deux caractéristiques sont très différentes. Dans un de ces deux cas, un bloc est perturbé par des tirs touchant des zones très éloignées d'où sont situées les portes le constituant. Ceci montre une forte complexité des mécanismes de propagation des erreurs.

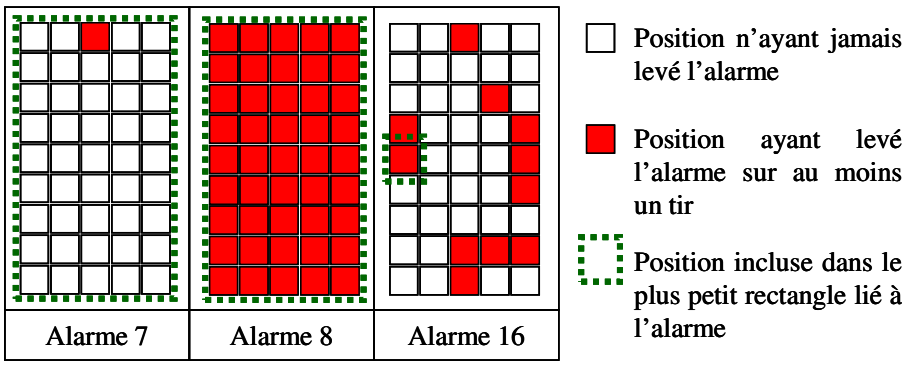

Figure 29 : Répartition géographique d'alarmes de la version Protégée1

Impact de l'instant d'injection

La position spatiale d'un tir définit l'ensemble des portes logiques illuminées, donc donnant potentiellement un résultat erroné. Nous avons vu qu'ainsi les blocs les plus volumineux sont globalement les plus touchés. De la même manière, l'instant d'injection devrait influer sur les alarmes levées, en touchant majoritairement les blocs les plus actifs à cet instant. La Figure 30 montre l'évolution sur la durée de l'exécution de la probabilité d'activation de chaque alarme pour la version Protégée1. Les alarmes se répartissent en trois groupes :

- une alarme ayant une probabilité d'activation élevée durant la première partie du calcul et très faible durant le reste de l'opération (alarme 17 : registres à décalage)
- les alarmes présentant un taux d'activation supérieur en milieu d'opération (alarmes 8 à 10 et 13 à 14 : cœur systolique et certains registres intermédiaires de calcul)
- les autres alarmes ayant un taux d'activation variable autour leur taux moyen pour lesquelles aucun motif particulier ne semble se dessiner (alarmes 0 à 7, 11 à 12 et 15 à 16 : registres d'entrée et de sortie, interface AMBA et d'autres registres intermédiaires de calcul)

Ceci montre que l'hypothèse d'une sensibilité accrue de certains blocs durant leur opération est vérifiée. En effet, le cœur systolique peu actif pendant les phases d'entrée et de sortie est plus sensible pendant la période centrale du déroulement de l'opération (la période de calcul). De même, le registre à décalage alimentant le réseau systolique est plus sensible durant environ les deux premiers tiers du calcul. L'évolution de la sensibilité des autres blocs ne semble pas présenter de motif particulier.

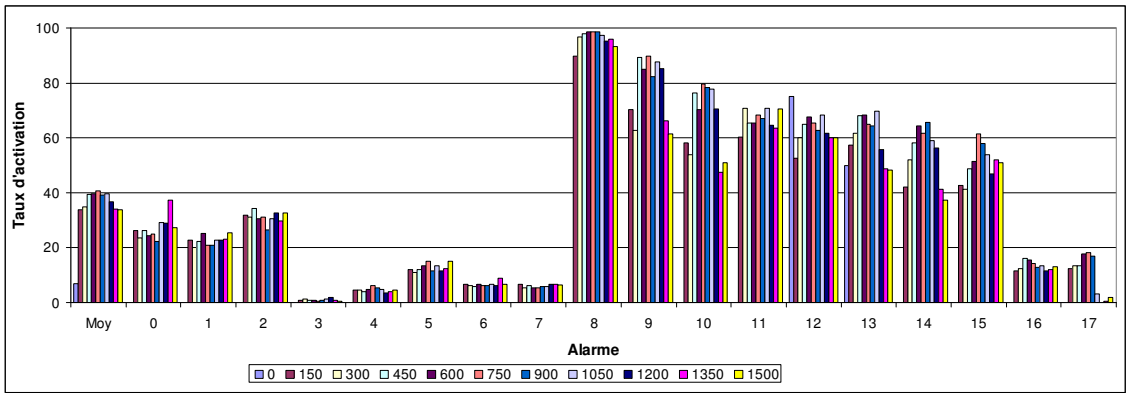

Figure 30 : Evolution temporelle des probabilités d'activation des alarmes pour la version Protégée1 en fonction de l'instant d'injection entre 0 et 1500

Reproductibilité des effets des tirs

Comme indiqué dans la description des campagnes, 5 tirs ont été réalisés pour chaque position spatio-temporelle sur chacune de versions étudiées. En analysant les données obtenues, il est possible d'étudier le déterminisme des effets des tirs laser réalisés. Le Tableau 6 présente un extrait d'un rapport de campagne d'injection ; il présente les résultats obtenus sur deux positions spatio-temporelles distinctes de la version Protégée1. On peut noter que, parmi les 5 tirs sur une même position, on obtient parfois une classification différente suivant le tir considéré : dans l'exemple choisi, un des tirs est non détecté alors que les autres le sont.

Tableau 6 : Extrait d'un rapport de campagne

Position X	Position Y	Cycle	Statut	Detect1	Detect0
1364,0	750,6	150	Alarm	FFFFCFFF	00003000
1364,0	750,6	150	Alarm	FFFFCFFF	00003000
1364,0	750,6	150	Alarm	FFFFEFFF	00001000
1364,0	750,6	150	Alarm	FFFFEFFF	00001000
1364,0	750,6	150	Undetected		
1654,0	895,6	1200	Alarm	FFFE34FF	0000C900
1654,0	895,6	1200	Alarm	FFFFB6FF	00004900
1654,0	895,6	1200	Alarm	FFFE94FF	00006900
1654,0	895,6	1200	Alarm	FFFFFEFF	00000100
1654,0	895,6	1200	Alarm	FFFFB6FF	00004900

De la même manière, plusieurs tirs sur la même position spatio-temporelle peuvent générer des valeurs différentes dans les registres de détection. Les Figure 31 et Figure 32 illustrent ces résultats. La Figure 31 (respectivement Figure 32) montre le nombre moyen d'instants d'injection (respectivement le nombre moyen de positions spatiales) pour lesquels les 5 tirs ont généré 1, 2, 3, 4 ou 5 configurations d'alarmes différentes. Les effets des tirs sont déterministes si une seule configuration d'alarme est détectée.

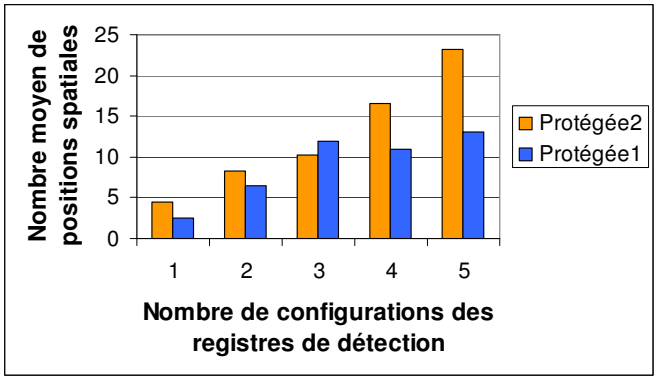

Figure 31 : Répartition des positions de tirs en fonction du nombre de motifs de détection

Ces deux graphes montrent que la levée des alarmes lors des tirs n'est pas déterministe. En effet, le nombre de cas pour lesquels on note la présence d'une seule configuration est très faible. Cette constatation est valable pour les deux versions protégées. De plus, on note que, pour les deux versions, on est majoritairement dans le cas où 5 configurations d'alarmes différentes sont obtenues.

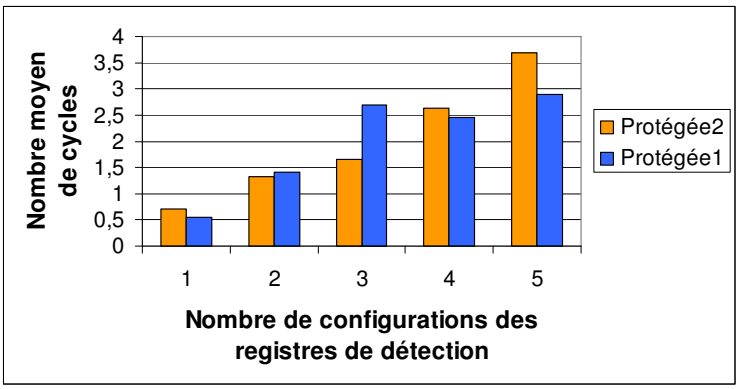

Figure 32 : Répartition des instants de tirs en fonction du nombre de motifs de détection

Le Tableau 7 montre la répartition en pourcents des positions spatio-temporelles en fonction du nombre de configuration d'alarmes générées. Pour la version Protégée1, presque 29% des points d'injections choisis induisent 5 configurations d'alarme différentes, tandis que seulement 6% montrent un caractère déterministe. Pour la version Protégée2, on observe une augmentation du nombre de cas déterministes ; cependant, on note aussi une augmentation du nombre de positions à 5 configurations différentes, confirmant un déterminisme global faible.

Tableau 7 : Répartition globale des tirs par rapport au nombre de configurations d'alarmes générées

Nb. de configurations différentes	1	2	3	4	5
Version Protégée1	5,6 %	14,2 %	26,9 %	25,4 %	28,9 %
Version Protégée2	7,1 %	13,2 %	16,6 %	26,3 %	36,8 %

Ces résultats montrent la complexité des mécanismes de propagation des erreurs induites lors de l'injection de fautes. De plus, les fortes variations dans les effets induits sont à prendre en compte lors de la conception d'un circuit robuste. De ce fait, le postulat de [Karp 04], supposant des fautes identiques sur plusieurs messages d'entrées, n'est pas réaliste lorsque l'on veut modéliser des attaques réelles.

Etude des cas de non détection

En analysant plus en détails les cas où la détection a échoué, on remarque que pour la version Protégée1, les 6 tirs correspondent à une unique position spatiale à des instants répartis sur toute la durée de l'opération. Pour la version Protégée2, ces tirs correspondent à deux positions spatiales non adjacentes à des instants distribués sur toute la durée de l'opération.

L'analyse des zones d'illumination permet d'identifier les sources possibles de la non-détection de ces tirs. Ces zones contiennent des portes des arbres de distribution des signaux de

réinitialisation ; il est donc possible qu'un registre ait été réinitialisé intégralement, incluant bits de données et bits de parité, permettant ainsi la non détection du tir.

De ce fait, lors de l'implantation d'un circuit sécurisé, il serait plus sûr de dissocier les arbres de réinitialisation des bits redondants et des bits de données pour éviter de telles réinitialisations simultanées ou de protéger les arbres eux-mêmes. Ceci aurait bien entendu un coût non négligeable, particulièrement sur les étapes de placement et de routage.

Prédiction des erreurs

L'analyse de la liste des portes illuminées par un tir laser, se trouvant dans la zone a priori éclairée par le faisceau, peut nous permettre de prédire a priori quelles sont les alarmes qui seront levées par ce tir. Une première analyse a été réalisée en ne considérant que les éléments logiques protégés par les mécanismes de détection. Beaucoup d'alarmes étaient levées sans être prédites. La méthode de prédiction a ensuite été modifiée pour prendre en compte les portes logiques ajoutées pour la propagation et l'amplification des signaux globaux (réinitialisation et horloge). Dans ce cas, les registres contrôlés par des éléments de ces arbres touchés par le tir sont aussi marqués pour une présence possible de faute. Les résultats de cette nouvelle méthode sont présentés dans le Tableau 8. En ne tenant compte que d'un type de signal global, les résultats sont quasi identiques au cas où les deux sont considérés.

Tableau 8 : Prédictibilité des alarmes (Protégée1 / Protyégée2)

	Signaux globaux pris en compte		Signaux globaux non pris en compte	
	Prédite	Non prédite	Prédite	Non prédite
Fauté	50,49 % / 30,36 %	0,25 % / 2,38 %	42,96 % / 20,70 %	7,78 % / 12,04 %
Non Fauté	48,02 % / 39,42 %	1,23 % / 27,84 %	23,83 % / 26,59 %	25,43 % / 40,67 %

Ces résultats montrent l'impact non négligeable des arbres d'amplification d'horloge et de réinitialisation. Leur prise en compte permet de grandement réduire le nombre de fautes non prédites. Cependant, on observe une augmentation du nombre d'alarmes signalées comme potentiellement levées mais non levées. Ceci peut s'expliquer par plusieurs éléments : la définition de la zone illuminée par le laser, la répartition de l'énergie du laser dans la zone éclairée et les mécanismes de propagation de fautes transitoires dans un circuit. En effet, la prédiction des alarmes est faite à partir de la liste des portes présentes dans la zone illuminée ; celle-ci est considérée comme un carré dont le coté est le diamètre du faisceau. D'autre part, aucune notion de densité énergétique à l'intérieur du faisceau laser n'a été considérée ; les portes sur les bords de la zone éclairées reçoivent moins

d'énergie. Enfin, une faute transitoire induite sur un signal interne n'entraîne pas systématiquement l'apparition d'une erreur qui sera mémorisée dans un registre protégé.

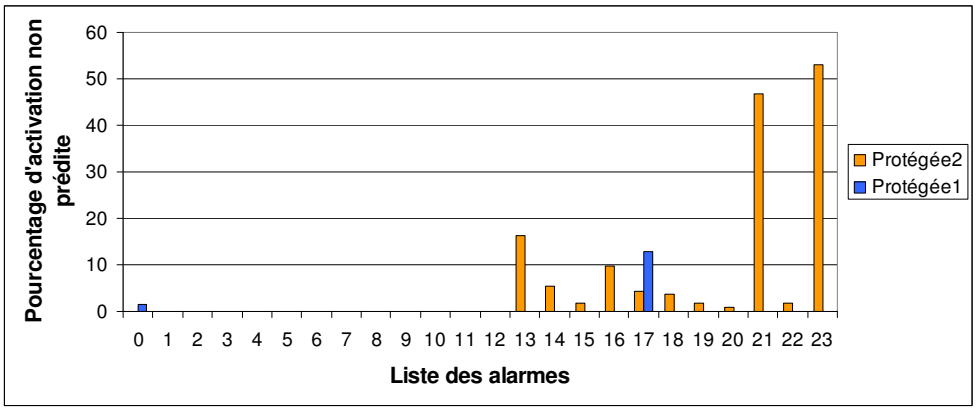

Figure 33 : Répartition des cas d'activation d'alarmes non prédites

La Figure 33 présente la probabilité pour chaque alarme d'être activée sans être prédite. Pour la version Protégée1, on note que seulement deux alarmes sont dans ce cas : l'alarme 0 et l'alarme 17 avec une probabilité respective d'environ 2% et 13%.Ces alarmes correspondent à des éléments liés au décalage de l'entrée A du multiplieur. Pour la version Protégée2, le nombre d'alarmes levées non prédites est plus important : deux alarmes (21 et 23) ont une probabilité d'activation non prédite de l'ordre de 50%, ces alarmes correspondent à une partie du bloc de contrôle et de décalage de l'entrée A. Les autres alarmes dans ce cas, présentant une probabilité plus faible, sont liées à des sorties du bloc de calcul systolique.

Synthèse des conclusions

Cette étude montre que les mécanismes de propagation des perturbations électriques générées par des attaques par tirs laser sont complexes et souvent non déterministes et ont une forte tendance à entraîner des fautes de multiplicité élevée. On note aussi que malgré la faible taille des mécanismes de vérification, il a été relativement facile de les toucher lors de ces attaques. L'utilisation de vérificateurs en double rail permet d'assurer une sécurité accrue en permettant la détection d'erreur même en présence de fautes dans une des branches de vérification. Bien que la surface des blocs semble être a priori un bon critère d'évaluation de la sensibilité des différentes fonctions d'un circuit, il en est autrement si on veut que cette évaluation a priori soit précise. De la même manière, la sensibilité des éléments les plus utilisés est accrue. Il pourrait donc être utile de mettre un accent supplémentaire sur leur protection. De plus, nous avons vu que les arbres de distribution des signaux d'horloge et de réinitialisation étaient particulièrement sensibles et pouvaient entrainer des cas de non-détection. Il est

donc nécessaire de les protéger et si possible de dissocier ceux pour les éléments de protection de ceux utilisés par le circuit non protégé.

2.3. Attaque par Laser sur un FPGA Xilinx Virtex2 xc2v1000

2.3.1. Présentation du circuit étudié

Le second circuit étudié est un circuit programmable de type FPGA configuré par une mémoire de type SRAM. Plus précisément, il s'agit du Xilinx Virtex II xc2v1000. Toutes les informations techniques sur la structure du circuit et de sa mémoire de configuration sont issues de la documentation technique de celui-ci : [Xili 07]. Un tel dispositif est le résultat de l'imbrication de deux matrices : une mémoire SRAM, contenant le fichier de configuration du circuit (aussi appelé bitstream) et une matrice de portes logiques et d'interconnexions paramétrable. Les valeurs prises par les différents bits de cette mémoire définissent donc la fonction réalisée par la matrice programmable. La matrice programmable est composée de différents éléments : un système de distribution d'horloge, des plots d'entrée/sortie, des blocs de mémoire RAM et une matrice de cellules logiques élémentaires (les CLBs) ; le nombre de ces éléments dépend du composant choisi. De la même manière, le bitstream est divisé en trames (1104 trames de 3392 bits pour le xc2v1000), chaque élément précédemment défini est configuré par un certain nombre de trames. La Figure 34, photographie annotée du circuit, présente la géométrie du FPGA et la position des différents constituants du FPGA.

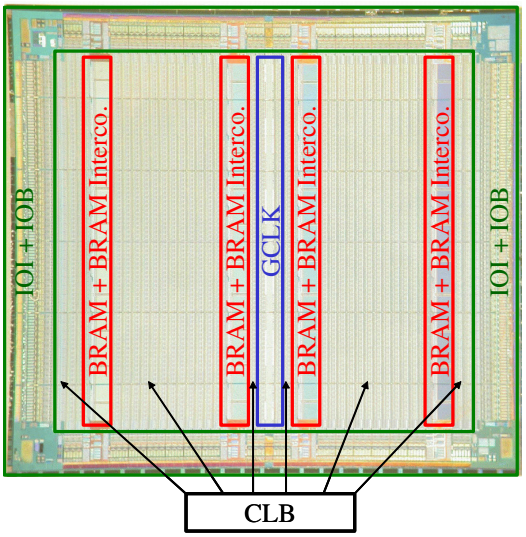

Figure 34 : Photographie du Virtex 2 xc2v1000

On retiendra qu'une colonne de CLB occupe plusieurs trames et qu'une même trame CLB va partiellement configurer tous les CLB de la colonne. Cependant, contrairement à ce que semble suggérer la Figure 34, l'ordre des trames dans le bitstream n'est pas strictement lié à la géométrie du circuit. En effet, les colonnes de RAM qui sont présents au cœur de la matrice de CLB ont leurs bits de configuration en fin de bitstream ; de même que la colonne d'horloge, au centre, voit ses bits en tête de bitstream. Le Tableau 9 indique le nombre de trames de configuration pour chacun des différents blocs.

Tableau 9 : Répartition des trames entre les différents constituants

Type d'élément	IOB	IOI	CLB	BRAM	BRAM Interco.	GCLK
# colonnes	2	2	32	4	4	1
# trames / Col.	4	22	22	64	22	4

La matrice de CLB est constituée de 32 colonnes de 40 CLB. Chacun de ces CLB est identique et contient une grille de routage entre les tuiles voisines et les 4 éléments de logique qui le constituent (chaque élément contient deux bascules D ainsi que des générateurs de fonction de type LUT). Une telle tuile CLB est configurée par un ensemble de 1760 bits : une largeur de 80 bits sur 22 trames. Sur une trame CLB, il reste 192 bits ne configurant pas le CLB lui-même ; ces bits, présents en tête et en fin de trame, configurent les entrées/sorties présentes sur le coté du FPGA, et seront désignés par CLB IO par la suite.

Sur le silicium, la matrice de CLB est entourée par deux couronnes : les IOI et IOB. Le bloc GCLK est au centre de la matrice de CLB, les colonnes de BRAM et leurs interconnexions sont eux aussi répartis dans la matrice de CLB. Cependant, dans le fichier de configuration, les bits correspondant aux différents éléments ne sont pas répartis de la même manière. Le Tableau 10 présente la localisation des différentes trames dans le bitstream.

Tableau 10 : Localisation des différentes trames dans le bitstream

Type d'élément	GCLK	IOB	IOI	CLB	IOI	IOB	BRAM	BRAM Interco.
Indice des trames	1-4	5-8	9-30	31-734	735-756	757-760	761-1016	1017-1104

Le but de cette campagne de tirs laser a été, d'une part, de valider la plate-forme d'injection (développée au sein du groupe de recherche) et sa connexion à la plate-forme laser du laboratoire IMS, et, d'autre part, d'être capable d'identifier les principaux effets des tirs sur le comportement statique

du FPGA. Pour cela, il a été configuré de manière très simple (aucune fonction particulière à réaliser), en utilisant toutes les ressources présentes dans les CLB et en ne connectant pas les mémoires RAM embarquées. Plusieurs configurations ont été utilisées pour varier l'état initial de la configuration de certaines ressources. Pour faciliter l'induction optique de fautes, le boîtier a été retiré et le circuit aminci.

Pour analyser les effets des tirs, il est nécessaire de connaître l'état final du circuit grâce au bitstream qui est relu par THESIC, et de le comparer à l'original.

Un logiciel a été développé au sein de l'équipe utilisant les classes Java JBits 3.0 fournies par Xilinx. Ces classes permettent l'analyse de fichiers de configuration pour les circuits de la famille Virtex II. L'analyse peut être réalisée à partir de fichiers de configuration originaux (.bit) ou relus à partir d'un circuit (.rdb). Cet outil est divisé en deux principales fonctions : l'analyse d'un fichier de configuration et la comparaison entre fichiers pour l'analyse des effets de fautes. Pour plus de détails sur cet outil, se référer à [Poug 07]. Cet outil a de plus été utilisé pour réaliser une ingénierie inverse partielle du fichier de configuration. La fonction des différents bits des CLB a été déterminée de cette manière.

Durant cette étude, une campagne d'environ cinquante balayages avec tirs multiples a été réalisée. Les zones balayées lors de cette étude sont de même surface et ont subi le même nombre de tirs. Le but de cette campagne était de mettre en évidence les effets des fautes sur un FPGA et d'identifier les différents motifs d'erreurs que peuvent générer des fautes.

2.3.2. Analyse des résultats

L'analyse des fichiers de configuration après tir laser sur le circuit montre qu'il y a des modifications de la configuration (donc de la fonction réalisée). Le Tableau 11 présente la répartition moyenne des bits fautés au sein des différents éléments constituant le circuit lors des différentes campagnes réalisées. Ce tableau se divise en trois parties suivant la valeur initiale des bits considérés. Les deux premières lignes représentent toutes les inversions de bits qui ont eu lieu ; les lignes 3-4, les bits qui ont été modifiés de '0' à '1' et les lignes 5-6 pour les inversions de '1' à '0'.

Les tirs laser effectués n'ayant pas pour cibles principales les plots d'entrées sorties, il est normal que le nombre de bits modifiés pour la configuration des tampons d'entrée/sortie ainsi que leurs matrices d'interconnexion ne présente que très peu d'inversion de bits. On remarque aussi que la plus grande partie des erreurs se situe dans les blocs CLB et BRAM, qui sont les constituants occupant la plus grosse partie de la surface du circuit. On remarque aussi que les nombres de '0' et de '1' fautés sont du même ordre de grandeur, avec respectivement 53,09% et 46,91%. Ceci doit cependant être corrélé avec la zone exposée au faisceau laser. Le Tableau 12 montre le nombre moyen de bits de

configuration par CLB pour d'une part le bitstream original et d'autre part pour l'ensemble des bitstreams fautés. Cette densité de répartition permet de déterminer les probabilités de fauter un bit selon sa valeur initiale. On peut ainsi remarquer qu'il est 2,5 fois plus probable de fauter un '1' qu'un '0'. '0' étant la valeur par défaut, ceci tend à signifier par exemple qu'il est plus facile de détruire des connexions que d'en créer.

Tableau 11 : Répartition moyenne des bits fautés

Type d'élément	Total	CLB	CLB IO	GCLK	IOB	IOI	BRAM	BRAM Interco.
# '0' et '1' fautés	137,85	80,95	0,54	0,00	0,03	0,03	50,41	5,87
Pourcentage	100,00	58,72	0,39	0,00	0,02	0,02	36,57	4,26
# '0'fautés	73,18	17,10	0,54	0,00	0,03	0,03	50,41	5,05
Pourcentage	53,09	12,40	0,39	0,00	0,02	0,02	36,57	3,66
# '1' fautés	64,67	63,85	0,00	0,00	0,00	0,00	0,00	0,82
Pourcentage	46,91	46,32	0,00	0,00	0,00	0,00	0,00	0,59

Tableau 12 : Nombre moyen de bits par CLB

Catégorie	Tous	Bits à '1'	Bits à '0'
Bits d'origine	1760	212,80	1547,20
Bits fautés	9,15	2,37	6,78
Probabilité de faute	0,52 %	1,11 %	0,44 %

Le Tableau 13 présente la répartition des bits erronés au sein des CLB. Ces bits sont classés selon 3 groupes dépendant du type de fonction qu'ils configurent : les bits de configuration d'éléments logiques (LUT et mémoire utilisateur), les bits de configuration d'interconnexions et les bits encore non identifiés. Les bits inconnus sont ceux qui ne peuvent être directement accédés en utilisant les classes JBits. Cette correspondance entre bit de configuration et fonction configurée a été rendu possible par la fonction d'analyse de bitstream du logiciel. De plus, au-delà de la simple identification de la fonction configurée par chaque bit, le rôle précis de chacun a été déterminé. On note que, comme on pouvait s'y attendre au vu de la taille des matrices d'interconnexions, que celles-ci sont les principaux éléments touchés au sein des CLB.

Tableau 13 : Répartition des fautes au sein des CLB

Localisation	Total	Logique	Interconnexions	Inconnu
Nombre	80,95	34,49	44,15	2,31
Pourcentage	58,72 %	25,02 %	32,03 %	1,68 %

La plupart des bits de configuration des éléments logiques des CLB servent à définir les fonctions des LUT et des registres. Les bits configurant les bascules ont été localisés dans le bitstream. Les valeurs prisent par ces bascules sont directement stockées dans la mémoire de configuration : chaque bit de mémoire utilisateur correspond à un bit du bitstream. Ainsi, une inversion de bits à ce niveau se manifestera par une erreur (et peut être une défaillance) lors de l'exécution. Les LUTs sont quant à elles paramétrées par leur table de vérité complète. Ainsi, 16 bits configurent une LUT à 4 entrées. Si une inversion de bit intervient, une erreur n'aura lieu à l'exécution que si le mot d'entrée correspondant est utilisé dans le circuit.

Les mécanismes de configuration des interconnexions sont hétérogènes. En effet, le nombre de bits nécessaires varie. La majorité des interconnexions sont définies par 2 bits (90,3%) ; très peu d'entre elles utilisent 3 bits (0,2%). Les 9,5% restant n'utilisent qu'un seul bit de configuration : la valeur de ce bit définit donc l'état de la connexion. Une inversion de ce bit entraînera soit une suppression du lien soit sa création. Pour les connexions définies par plusieurs bits, il existe 6 modifications possibles, dépendant de l'état initial de la connexion. La Figure 35 résume ces différentes modifications. Pour une connexion initialement inexistante, une ou plusieurs inversions de bits peuvent soit ne rien modifier sur la structure de la connexion, soit créer des liens auparavant inexistants. Si la connexion existe initialement, la connexion peut ne pas être affectée par les inversions de bits ; il peut aussi arriver que celle-ci soit simplement supprimée, que de nouveaux liens soient créés en plus du lien de départ ou que la connexion soit modifiée (création de nouveaux lien et suppression du lien initial).

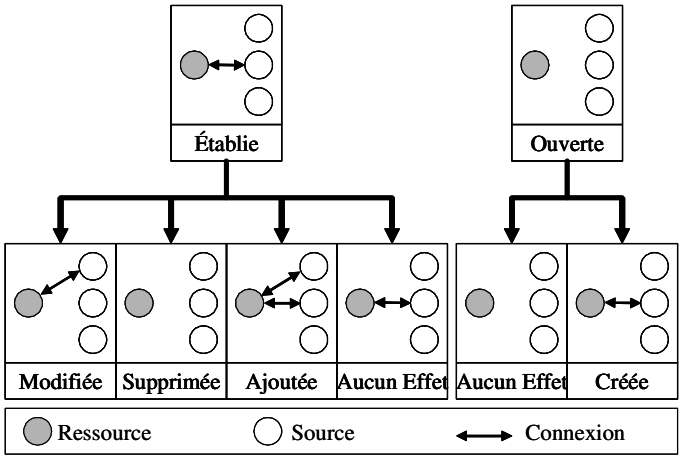

Figure 35 : Différents types de modifications subies par les interconnexions

Le Tableau 14 présente le nombre d'inversions de bits dans les interconnexions pour chaque type, ainsi que le nombre de motifs de modifications observés. Ceci montre que 94% des inversions de bits ont eu lieu sur des connexions configurées par deux bits, pourcentage cohérent avec la proportion

de telles connexions dans le circuit. Il nous a cependant été impossible de générer des fautes dans les très rares connexions définies par 3 bits. La sensibilité d'une connexion dépend grandement du nombre de bits la configurant. Plusieurs connexions sur un bit sont définies par le même bit, ce qui explique le nombre supérieur de motifs de modification par rapport au nombre d'inversion sur ces connexions. Inversement, la complexité des interconnexions à bits multiples permet qu'un nombre d'inversions de bits parfois important soit nécessaire pour créer un motif d'erreur.

Tableau 14 : Répartition des bits fautés dans les interconnexions

# bits de configuration	1	2	3
# inversions de bits	18	2290	0
# motifs de modification	91	1407	0

Le Tableau 15 montre la classification des inversions de bits en fonction du type de motif d'erreur généré sur une interconnexion définie par 2 bits. Il est important de noter que dans de nombreux cas, il est possible de maintenir une connexion établie en présence de fautes. En effet, un peu plus de 50% des modifications ne touchent pas un lien existant (colonnes « Ajoutée » et « Aucun effet »). Dans le cas d'ajout de connexions parasites, il est difficile d'étudier a priori l'impact sur la fonctionnalité du circuit, car cela dépend des connexions existantes dans les CLB voisins, et donc de la fonction globale du circuit.

Tableau 15 : Répartition des bits fautés en fonction des motifs d'erreur générés sur une interconnexion à 2 bits

Etat Initial	Connectée				Non connectée	
Effet sur la connexion	Modifiée	Supprimée	Ajoutée	Aucun effet	Aucun Effet	Créée
# moyen de bits inversés	16,5	30,7	49	0	1613	518
# moyen de motifs	7,1	20,4	29	0	1163,1	187,4
%	0,5 %	1,5 %	2,1 %	0,0 %	82,7 %	13,3 %
# moyen d'inversion de bits / motif	2,3	1,5	1,7	n/a	1,4	3,1

Pour étudier plus en détails les effets d'inversions de bits dans la mémoire de configuration des interconnexions, il est nécessaire d'étudier plus en détails comment celles-ci sont réellement configurées. Pour les connexions à 2 bits, un lien est défini entre une ressource donnée et une des sources/puits atteignable depuis cette ressource. A une ressource donnée est associé un nombre variable de bits de configuration ; chacun de ces bits décrit une partie de la liste des sources atteignables. Chaque source apparaît dans deux de ces listes (3 pour les interconnexions à 3 bits).

Ainsi, pour connecter une ressource A à une source B donnée, il suffit d'activer les deux bits de configuration liés à la ressource A dont les listes de sources atteignables contiennent B. La Figure 36 illustre la structure de l'interconnexion suivante : la ressource OMux9 peut être connectée à 4 sources différentes (XQ0, XQ1, YQ0 et YQ1) ; la connexion de OMux9 à l'une de ces sources est définie par 5 bits nommés de B1 à B5 ayant respectivement pour liste des sources atteignables {XQ0, XQ1}, {YQ0, YQ1}, {XQ0}, {XQ1, YQO} et {YQ1}. Les ressources citées existent dans le circuit, mais la ressource OMux9 peut être connectée à un nombre plus important de sources. En moyenne, les connexions d'une ressource sont configurées par 9,1 bits.

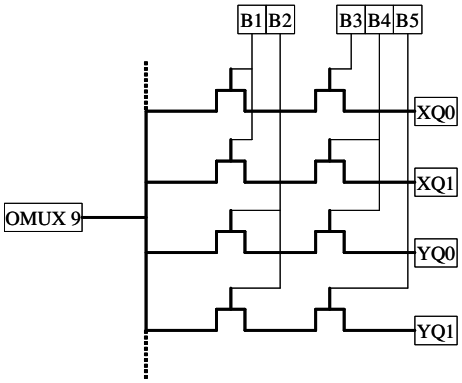

Figure 36 : Exemple d'interconnexion à deux bits

Pour créer une connexion à partir d'un état d'inutilisation d'une ressource, il est donc nécessaire d'inverser les 2 bits correspondant à une source. Cependant, l'intersection de deux listes prises au hasard peut être vide. Ceci implique qu'il faut au moins deux inversions de bits pour créer une connexion. En effet, on note que l'inversion de 3,1 bits en moyenne est nécessaire pour y parvenir (Tableau 15). Ceci explique pourquoi la plupart des inversions n'ont aucun effet sur une connexion initialement non établie, avec un nombre moyen de bits fautés (1,6) inférieur au seuil nécessaire à la création d'un lien.

Si on considère une connexion initialement établie, pour la supprimer, il faut fauter un des deux bits de configuration de ce lien. En fautant un des autres bits de configuration, on peut ajouter un lien supplémentaire. On trouve donc une probabilité d'ajout de connexion supérieure à celle de suppression ; ceci est cohérent avec les résultats obtenus Tableau 15. Cependant, le nombre moyen d'inversions de bits est supérieur pour « Ajoutée » que pour « Supprimée » dû à la présence d'ajouts multiples de connexions. Dans le cas « Modifiée », il faut au moins inverser deux bits : un pour couper le lien existant et un second pour en créer un autre. Le cas « Modifiée » étant la combinaison des autres cas, sa probabilité d'occurrence est inférieure et nécessite un nombre supérieur d'inversions de bits. Le cas « Aucun effet » apparaît quand les bits inversés ne correspondent à aucune des sources présentes dans l'union des listes déjà activées.

La Figure 37 présente le nombre moyen d'inversions de bits par motif d'erreur en fonction du nombre de connexions établies dans les cas « Créée », « Ajoutée » et « Modifiée ». Le nombre moyen d'inversions de bit nécessaire pour créer une connexion est 2, ce qui démontre la très forte probabilité lors de l'activation d'une liste d'obtenir une intersection non vide avec une liste préalablement activée. Ce qui justifie que l'inversion d'un bit suffise en moyenne à ajouter un lien. De la même manière que pour la création, la modification de connexion nécessite une inversion de bit supplémentaire par rapport à un ajout, venant de la nécessité de désactiver un des deux bits existant. Pour chacun des motifs de modification identifiés, le nombre d'inversions de bits nécessaires à la création d'un lien diminue avec le nombre de connexions établies. Ceci est dû à l'augmentation de la probabilité d'activer une liste contenant une source déjà activée quand le nombre de listes activées augmente.

Les motifs de modifications ne sont pas ceux utilisés dans la littérature [Sonz 05] étant donné que le but ici était d'identifier les modifications qui pouvaient survenir sur la configuration d'une ressource. La caractérisation des effets d'un tir laser isolé, l'élaboration d'un modèle de fautes précis et l'étude de la criticité des différents constituants du composant sont des points dont l'étude a continué au sein du groupe. Ainsi, les effets de tirs laser pourront être analysés sur un circuit quelconque. Ceci permettra de connaître l'impact d'une modification ponctuelle sur le circuit complet, et d'utiliser la classification utilisée par [Sonz 05].

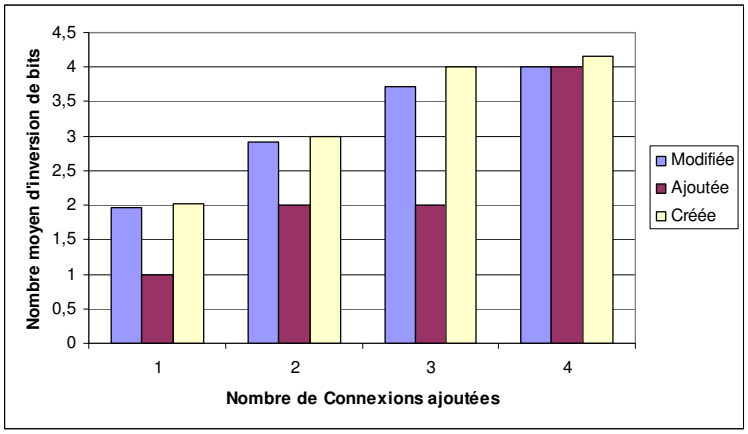

Figure 37 : Nombre moyen de bits fautés nécessaires à la modification de connexions multiples

2.4. Conclusion

Cette étude montre à quel point il est facile d'induire des fautes lors d'une attaque laser. De plus, nous avons vu que même lors d'une approche en boîte noire (méconnaissance a priori des zones les plus critiques du circuit), il est possible de générer des erreurs non détectées, risque majeur potentiel pour une application sécurisée.

Cette étude a notamment montré la nécessité d'utiliser des modèles de fautes à multiplicité élevée durant l'évaluation d'une protection. De plus, malgré la faible taille des blocs de protection, ils peuvent eux aussi être souvent touchés. Il est donc nécessaire de s'assurer que les protections resteront efficaces dans ces cas. Les techniques de double rail semblent être une réponse à ce problème. Enfin, une attention toute particulière doit être apportée sur les arbres d'horloge et de réinitialisation.

Cependant, les mécanismes de propagation des erreurs induites par un tir laser sont complexes. Une étude plus approfondie permettrait de comprendre les variations observées sur le déterminisme des effets d'un tir et l'influence qu'ils peuvent avoir sur des cônes de logique non touchés.

Lors de l'utilisation d'une technologie programmable, de nouveaux problèmes apparaissent. En effet, les fautes induites peuvent se glisser dans la mémoire de configuration du circuit provoquant ainsi des modifications de sa fonctionnalité. L'utilisation d'une telle technologie nécessite d'une part la maximisation du nombre de CLB utilisés et si possible la protection de la mémoire de configuration (seulement réalisable par les concepteurs de circuits programmables). En effet, les modifications des interconnexions observées peuvent entraîner la création de liens parasites avec les ressources au voisinage proche ; la réduction de cette proximité des différentes ressources utilisées peut aider à rendre le circuit plus robuste.

Malgré certains cas de non détection, l'utilisation des protections choisies nous a permis de fortement augmenter la sécurité du circuit en réduisant de manière significative les informations erronées fournies à un attaquant potentiel. Cependant, un pirate utilisera le type d'attaque avec lequel il aura le plus d'informations possible. Les mécanismes de protection mis en place ici, et par extension l'ajout de codes détecteurs, ne pourraient-ils pas être déjoués plus facilement avec des attaques par canaux cachés ? Nous tenterons de répondre à cette question dans les chapitres suivants.

Chapitre 3 : Evaluation de l'impact de codes détecteur/correcteur d'erreurs sur la sensibilité aux attaques par canaux cachés

Nous avons vu précédemment comment certains codes détecteurs d'erreurs, qui sont une des protections les plus largement utilisées contre les fautes, se comportent en présence de faute dans un circuit pour une application sécurisée. Cependant, un pirate possède aussi la carte des attaques par canaux cachés pour attaquer un circuit. Nous allons ici tenter de comprendre quel est l'impact de l'ajout de codes détecteurs ou correcteurs sur la sensibilité d'un circuit aux attaques par canaux cachés. Pour cela, nous allons étudier dans un premier temps l'impact sur les attaques temporelles. Ensuite, nous présenterons deux flots d'analyse pour les attaques en puissance et les résultats obtenus sur le cas simple d'un registre durci suivant différentes méthodes. Comme nous l'avons vu précédemment, la sensibilité aux attaques électromagnétiques découle pour beaucoup de celle vis-à-vis des attaques en puissance, ce point ne sera donc pas couvert ici.

3.1. Sensibilité face aux attaques temporelles

Nous avons vu, dans le chapitre 1, que les attaques temporelles utilisent la corrélation entre les données traitées dans le circuit et le nombre de cycles d'exécution. Pour évaluer l'impact des protections par ajout de codes détecteurs ou correcteurs, il est nécessaire d'étudier leur impact sur le nombre de cycles nécessaires à l'exécution de l'application sécurisée.

Nous savons que l'ajout de telles protections nécessite l'élargissement des registres protégés pour y stocker les bits de redondance, l'ajout des registres de détection stockant les signaux de détection et l'ajout des blocs de prédiction et de vérification. Les deux premières modifications n'ont aucun effet sur le temps d'exécution de l'application, les bascules ainsi ajoutées s'activant en même temps que les celles déjà existantes. L'ajout des blocs de prédiction et de vérification peut, quant à lui, avoir des effets sur ce temps d'exécution. Nous allons dissocier les modifications pour la protection des fonctions combinatoires de celles utilisées pour les registres. Lors de la protection d'un bloc combinatoire, on juxtapose à la fonction originale un bloc de prédiction suivi d'un bloc de vérification.

La vérification doit être terminée durant le cycle d'horloge pour permettre la mémorisation de l'alarme ; ceci peut alors nécessiter une réévaluation de la fréquence d'opération si la fonction ainsi protégée intervient sur le nouveau chemin critique. Au niveau du registre, on observe souvent un impact moins flagrant. En effet, la vérification du registre est souvent plus courte que la fonction qu'il sert à alimenter et le codage de son entrée est souvent réalisé en réutilisant la sortie du codeur utilisé lors de la vérification du bloc combinatoire précédent. Dans le cas contraire, on observe aussi une augmentation du temps de calcul dans le bloc combinatoire précédant le registre pouvant avoir un impact sur la fréquence d'opération. L'ajout de codes détecteurs ou correcteurs ne modifie cependant en rien le nombre de cycles d'horloge nécessaire pour réaliser l'opération.

On peut donc en conclure que l'ajout de codes détecteurs ou correcteurs ne modifie pas la sensibilité aux attaques temporelles.

3.2. Flots d'analyse de la sensibilité aux attaques en puissance

L'étude de la sensibilité d'un circuit face aux attaques en puissance lors de sa conception est basée sur la capacité à réaliser les attaques en l'absence d'un prototype final. Pour ce faire, cette évaluation passe par l'obtention de valeurs de consommation électrique simulées. Les deux flots d'évaluation proposés sont basés sur deux niveaux de description du circuit : le premier est basé sur des simulations sur une description logique au niveau portes, tandis que le second utilise une description au niveau transistors.

Ces deux flots ont été conçus pour s'insérer le plus simplement possible dans le flot de conception classiquement utilisé et seront présentés du niveau le plus haut au niveau le plus bas.

3.2.1. Evaluation niveau portes logiques

La première version du flot d'évaluation de la sensibilité aux attaques en puissance est basée sur l'analyse de valeurs de consommation obtenues lors de simulations de la description du circuit au niveau portes logiques. La Figure 38 présente le graphe de ce flot.

Les premières étapes de ce flot correspondent aux étapes du flot classique de conception auquel il se rattache. Notre point de départ dans cette étude est une description comportementale du circuit écrite dans un langage de description de matériel : dans notre cas, il s'agit de VHDL. L'implantation des protections étudiées a été réalisée à ce niveau de description. L'étape suivante est la synthèse logique sur la technologie cible choisie. Cette étape est réalisée avec Synopsys DesignVision qui est utilisé le plus souvent au sein de l'équipe. La description structurelle au niveau portes logiques du circuit est ensuite exportée au format VHDL.

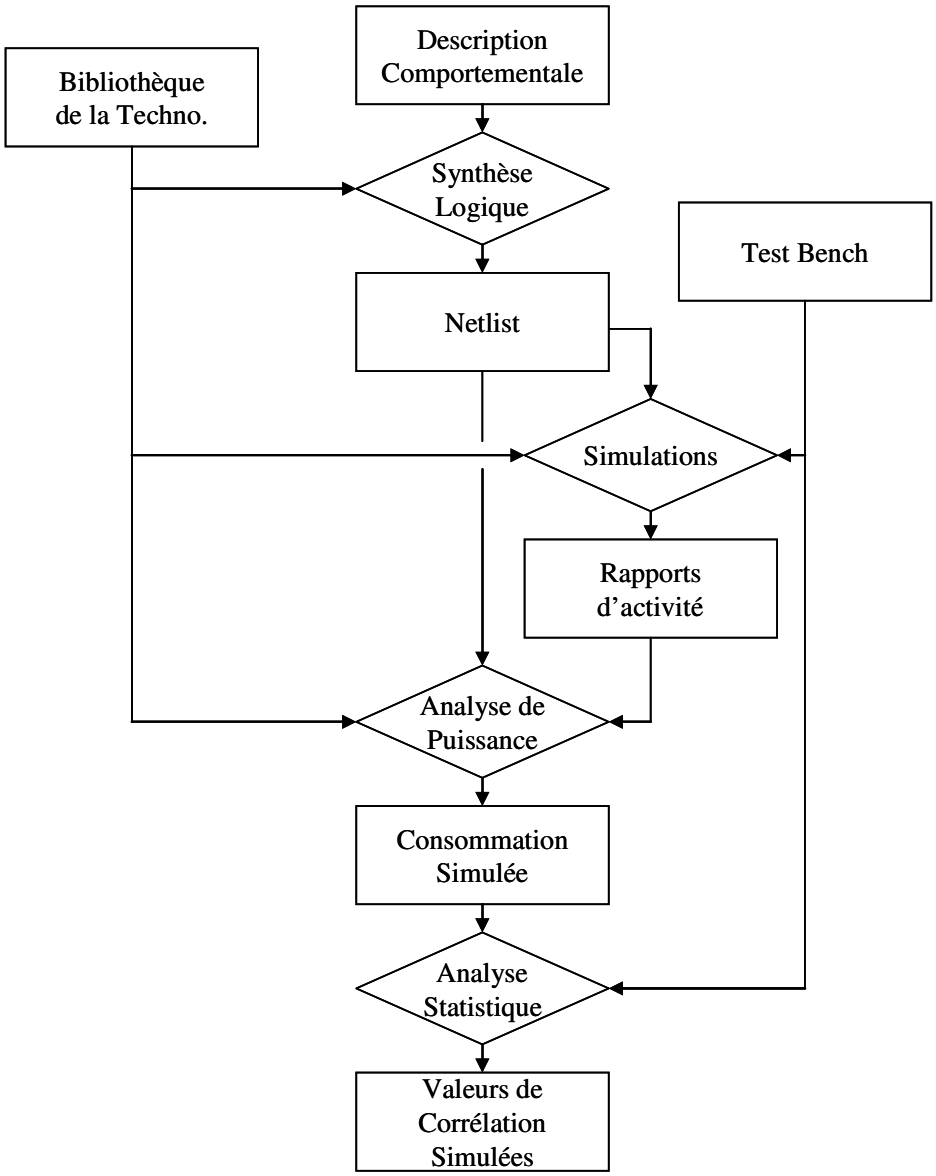

Figure 38 : Flot d'évaluation au niveau portes

La suite du flot regroupe toutes les étapes de l'évaluation de la sensibilité aux attaques en puissance à proprement parler. La première des ces étapes consiste à générer l'ensemble des vecteurs d'entrée voulus pour la campagne de simulations. Pour étudier le lien entre les données traitées et la puissance consommée, il est évident que l'on ne peut pas se limiter à une étude de la consommation moyenne.

Les vecteurs d'entrée ainsi générés sont utilisés via le test bench lors de simulations logiques avec Mentor Graphics Modelsim. Ainsi, pour chaque vecteur d'entrée, l'activité de chaque signal

interne de la description est enregistrée au format VCD. Ces rapports d'activité contiennent, pour tous les instants de commutation de signaux, le signal ayant commuté ainsi que sa nouvelle valeur.

Chacun de ces rapports d'activité interne du circuit est ensuite analysé lors de l'étape d'évaluation de la puissance consommée. Cette étape est basée sur une évaluation de la puissance consommée pour chaque nœud interne du circuit. Ainsi, en croisant ces informations avec l'activité enregistrée, il est possible de connaître la puissance consommée au cours de la période simulée. Cette évaluation de la puissance consommée est réalisée avec l'outil Synopsys PrimePower, plus particulièrement grâce à PowerCompiler qui est le moteur de calcul de puissance qui peut être invoqué indépendamment. On obtient ainsi la consommation de chaque porte logique du circuit sur la durée de simulation.

Toutes ces valeurs de consommation simulées sont ensuite utilisées pour une analyse statistique. Grâce à un ensemble de scripts shell et Matlab, les données sont extraites des rapports d'analyse de puissance et sont exploitées pour déterminer un coefficient de corrélation entre les données traitées et la puissance consommée simulée. Ce calcul est dépendant du circuit étudié et sera explicité plus tard dans cette section.

Ces valeurs de corrélation permettent de mesurer l'impact des différentes protections choisies sur la sensibilité aux attaques en puissance. Ainsi, il est possible de choisir une protection contre les fautes aussi en fonction de son influence sur les canaux cachés.

3.2.2. Evaluation niveau transistors

La deuxième version du flot d'analyse de sensibilité aux attaques en puissance est basée sur une étude de la consommation électrique du circuit sur une description au niveau transistors. Comme la première version, elle est conçue de manière à s'intégrer au mieux dans le flot de conception classiquement utilisé. La Figure 39 montre le graphe de cette version du flot d'étude.

On remarque ici aussi que les premières étapes appartiennent au flot de conception et sont identiques à celles déjà présentées. Cependant, nous sommes obligés d'aller plus loin dans le flot de conception pour être en mesure de réaliser des simulations au niveau transistors. Il est à noter que les étapes de placement et de routage ne sont pas réalisées pour cette évaluation, mais pourraient l'être pour augmenter la précision. L'étape de génération de la description niveau transistors du circuit est réalisée avec des outils de la suite Cadence.

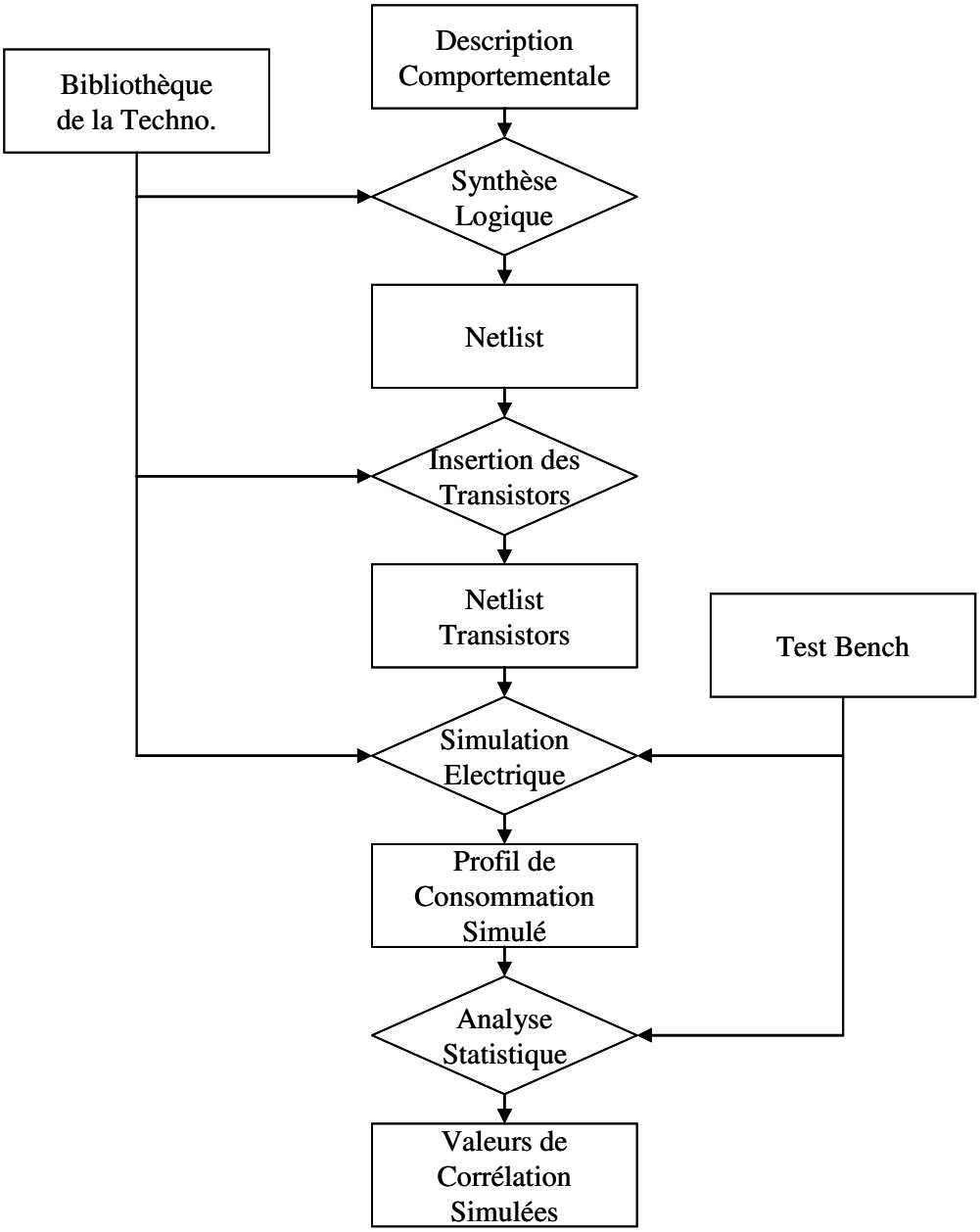

Figure 39 : Flot d'évaluation au niveau transistors

De la même manière que précédemment la génération du test bench inclut aussi celle de tous les vecteurs d'entrée nécessaires. Les simulations électriques sont réalisées grâce au simulateur rapide Nanosim de Synopsys. Ces simulations électriques permettent d'obtenir des traces de courant consommé par le circuit.

Les traces de courant ainsi obtenues sont ensuite converties grâce à un outil de visualisation de courbes dans un format de fichier textuel plus facilement exploitable par des scripts shell ou un programme écrit en langage C. L'outil d'analyse statistique a été écrit dans le cas de ce flot en langage

C, ce qui permet d'avoir des temps de traitement beaucoup moins longs que sous Matlab. Ici aussi le but de ce traitement est d'obtenir des informations de corrélation entre la puissance consommée et les données traitées.

3.3. Analyse niveau portes de registres durcis

Nous avons donc utilisé le flot d'évaluation de sensibilité aux attaques en puissance sur un registre sécurisé de façons différentes en se basant sur des codes détecteurs ou correcteurs. Lors de cette étude, nous avons sélectionné un certain nombre de codes souvent utilisés dans des implantations d'applications sécurisées. Nous allons tout d'abord présenter les codes choisis. Puis, nous étudierons les résultats obtenus lors de l'étude de ces codes.

3.3.1. Présentation des codes étudiés

Les codes choisis se répartissent en trois groupes : des codes détecteurs, des codes correcteurs et des codes détecteurs plus particulièrement utilisés pour les registres d'état. Nous avons aussi sélectionné deux tailles de registres (8 et 16 bits) pour observer l'effet de la taille du registre protégé sur sa sensibilité aux attaques en puissance.

Les codes détecteurs choisis sont d'une part deux codes linéaires et d'autre part deux codes non linéaires. Les codes linéaires utilisés sont la parité simple et la parité double croisée. Ces codes ont été introduits dans le chapitre 1 et sont basés respectivement sur l'ajout d'un bit étant le résultats du XOR de tous les bits de données et sur l'ajout de deux bits étant le résultat du XOR des bits de poids pair d'une part et des bits de poids impair d'autre part (leur matrice de vérification H est donnée dans le chapitre 1). Les deux codes non-linéaires choisis sont le code à parité double complémentaire et le code de Berger. Le code à parité double complémentaire ajoute deux bits au mot de données : l'un est le bit de parité simple et l'autre son complémentaire. Le code de Berger, quant à lui, ajoute au mot de données le codage en binaire du poids de Hamming du mot de données. La Figure 40 montre les schémas de câblage pour les différents codes détecteurs utilisés pour cette étude. Le code de Berger utilise comme boite de base dans sa structure des additionneurs 1 bit.

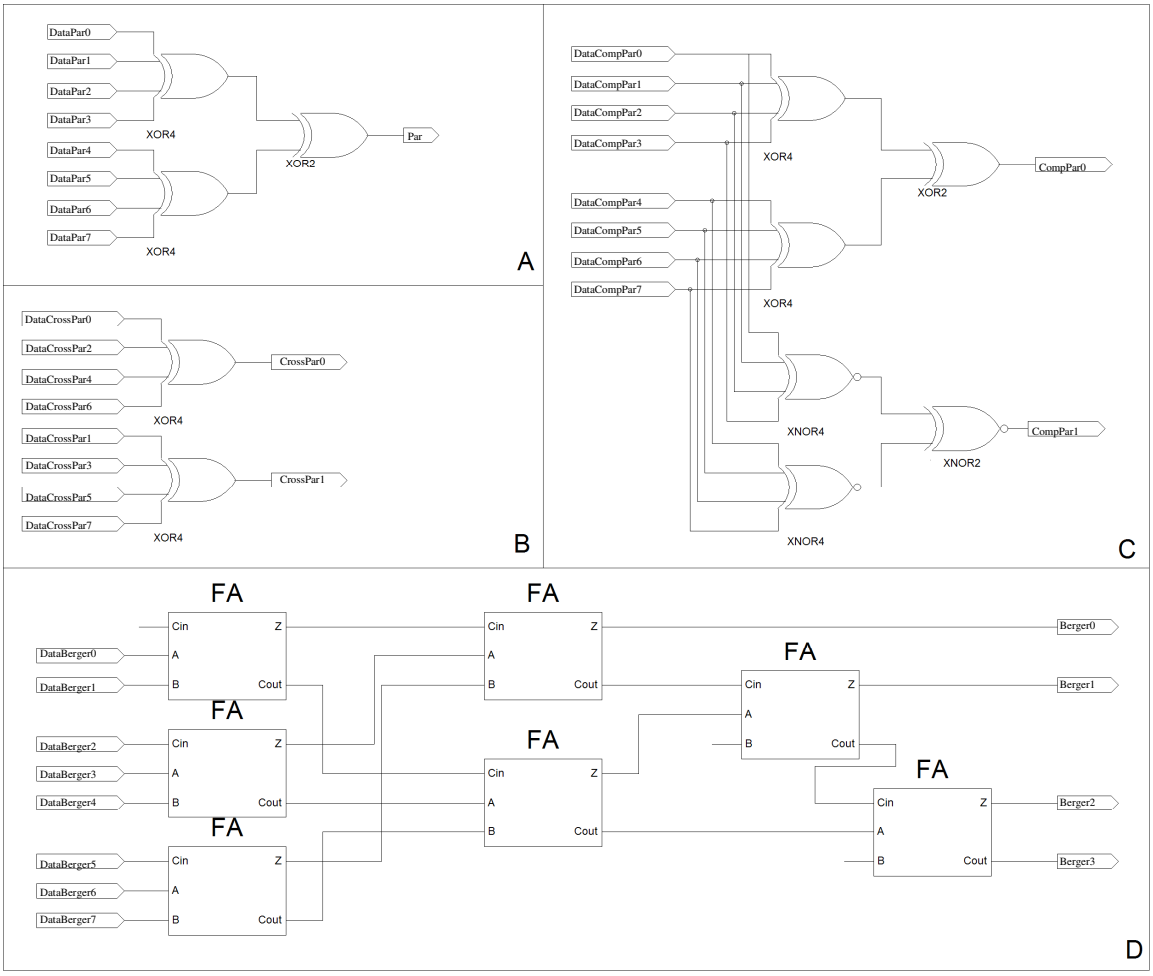

Figure 40 : Schéma de cablage des codeurs des codes détecteurs (A : Parité simple, B : Parité croisée, C : Parité complémentaire et D : Berger)

Les codes correcteurs utilisés sont trois versions d'un code de Hamming de matrices différentes et le code de Hsiao. Ces codes sont définis pour les registres 8 bits par les matrices de vérification suivantes :

$$H_{HammingV1(8)} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_{HammingV2(8)} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_{HammingV3(8)} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_{Hsiao(8)} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

La matrice de vérification H du code HammingV1 est obtenue comme suit : chaque vecteur colonne de cette matrice est la représentation binaire des entiers de 1 à 12. Les puissances de 2 sont regroupées en fin de mot ; ces nombres représentent les bits de vérification ajoutés. Ainsi la version 16 bits de ce code est obtenue de la même manière : les vecteurs colonnes représentent les entiers de 1 à 21.

La matrice de vérification du code HammingV2 est obtenue à partir de la matrice du code précédent. En effet, deux vecteurs colonnes ont été remplacés : les vecteurs 7 et 11 sont devenus 13 et 14. La construction de cette matrice rend impossible son exacte reproduction sur une version 16 bits du code. Ainsi, le code HammingV2 ne sera pas étudié sur 16 bits.

La matrice de vérification du code HammingV3 est obtenue comme suit : les vecteurs colonnes dont l'indice est une puissance de 2 représentent cette puissance de 2 et correspondent aux bits de redondance; les autres vecteurs colonnes, correspondant aux bits de données, représentent l'indice dans le mot original du bit de données occupant l'indice de ce vecteur. Ainsi, le premier bit de données D1 sera en colonne 3 du mot codé et le bit D2 en colonne 5 ; ces deux vecteurs colonnes valent respectivement 1 et 2. La version 16 bits du registre protégé par ce code est construite de la même manière.

La matrice de vérification d'un code de Hsiao est obtenue à partir de la construction décrite dans le chapitre 1. La même technique a été utilisée pour construire la matrice 8 bits présentée ici et la version 16 bits.

La structure logique des codeurs pour les différents codes correcteurs utilisés ici sont constitué de 4 (ou 5 pour Hsiao) arbres de portes XOR dont les entrées sont listés sur chacune des ligne de la matrice H. Par exemple, le premier bit de redondance du code HammingV1 est donné par D0 xor D1 xor D2 xor D6 xor D7.

Nous avons aussi étudié quelques codes principalement utilisés pour les codages de registres d'états. Il s'agit de plusieurs versions de codes en M parmi N. Ces codes utilisent N bascules dont

seulement M valent 1. Ce type de code n'est en général pas utilisé pour des registres de données à cause de son surcoût en nombre de bascules très élevé. Un registre d'état a en général besoin de peu de valeurs et les fonctions de transition sont en général moins complexes avec un tel codage qu'avec un codage binaire classique. La protection de ce type de registres ne nécessite avec un tel codage que l'ajout de vérificateurs pour s'assurer du nombre exact de 1 dans le registre.

Les différentes versions du registre ont été synthétisées en utilisant la bibliothèque C35_CoreLib de technologie AMS.

Grâce au flot mis en place, nous avons pu obtenir pour différentes transitions de données au sein des différentes versions des registres la consommation des différentes portes logiques les constituant. Nous avons donc divisé le registre en trois sous parties fonctionnellement distinctes : le codeur, le banc de bascules et les vérificateurs, incluant la logique de correction pour les codes correcteurs. L'analyse statistique consiste dans ce cas en deux étapes : le calcul de la puissance totale consommée dans chacun des trois sous-blocs et le calcul du coefficient de corrélation entre cette puissance et le poids de Hamming de la différence entre le mot initialement dans le registre et celui chargé pendant la simulation. Le coefficient de corrélation est calculé grâce à la formule $Corr = \frac{\sum (Hi - \overline{H})^2 * \sum (Pi - \overline{P})^2}{\sum (Hi - \overline{H}) * (Pi - P)}$, où *Hi* et *Pi* représentent respectivement le poids de Hamming et la consommation pour le vecteur *i* (i parcourrant l'ensemble des vecteurs utilisés). Pour chacun des codes testés, nous avons ainsi obtenu quatre valeurs de corrélation (une par sous-bloc et une pour l'ensemble). Ce calcul de corrélation est impossible pour les registres codés en M parmi N du fait de l'absence de variation du poids du mot contenu dans le registre ; leur étude sera présentée plus tard. Avant cela, nous allons d'abord comparer les différents codes détecteurs choisis, puis nous étudierons les codes correcteurs. De plus, nous avons voulu étudier l'influence de la géométrie des données chargées sur la consommation du registre durci. Pour ce faire, nous avons déterminé la corrélation entre les valeurs de consommation obtenues et une mesure de cette géométrie : la fraction des bits à 1 du mot de différence présents dans sa moitié de poids fort.

3.3.2. Comparaison des codes détecteurs

L'étude des résultats obtenus regroupe plusieurs aspects. Nous allons dans un premier temps étudier le rapport entre le poids de Hamming des données et la consommation du registre. Ensuite, nous verrons comment la géométrie des données chargées influe sur la consommation du registre

durci. Nous verrons ensuite que la manière de connecter le banc de bascules aux arbres de codage et de vérification peut avoir des conséquences non négligeables. Enfin, nous étudierons l'impact du passage à des données sur 16 bits sur les conclusions précédentes.

Influence du poids de Hamming des données chargées dans un registre 8 bits durci sur sa consommation

Le Tableau 16 présente les valeurs de corrélation ainsi obtenues pour les différents codes détecteurs étudiés pour la protection du registre 8 bits. Il présente aussi le surcoût en surface de leur implantation ainsi que leur probabilité de détecter une erreur de multiplicité quelconque. Le surcoût en surface inclut le coût double des vérificateurs double rail. Les valeurs de corrélation présentées dans ce tableau sont issues de simulations exhaustives avec une valeur initiale nulle (le registre est bien sûr initialisé avec la valeur du mot nul codé et non que des 0). Avec d'autres valeurs initiales, les valeurs obtenues sont du même ordre de grandeur et permettent de tirer les mêmes conclusions. Dans ce qui suit, le poids de Hamming du mot chargé est aussi le poids de Hamming de la différence entre le mot initialement dans le registre et celui chargé.

Tableau 16 : Comparaison des codes détecteurs sur un registre 8 bits

Code	Surcoût en surface	Probabilité de détection	Corrélation			
			Codeur	Bascules	Vérificateur	Global
Aucun	+ 0 %	0 %	n/d	1	n/d	1
Parité Simple	+ 120 %	50 %	0,995	0,927	0,955	0,977
Parité Double Croisée	+ 159 %	74 %	0,998	0,799	0,909	0,952
Parité Complémentaire	+ 230 %	75 %	0,012	0,334	0,005	0,056
Berger	+ 358 %	93 %	0,989	0,999	0,993	0,994

Trois groupes de codes semblent ressortir du Tableau 16. Le premier est constitué du cas «Aucun code » qui présente une corrélation de 1, indiquant une dépendance linéaire entre la consommation du registre et le poids de Hamming du mot chargé. Le second groupe regroupe les codes à parité simple, à parité double croisée et de Berger ; ils présentent tous trois une très forte corrélation, indiquant une dépendance presque linéaire. Enfin, le code à parité complémentaire présente quant à lui une très faible corrélation entre les données chargées et la puissance consommée, présentant ainsi une consommation quasi indépendante des données. La Figure 41 présente la puissance consommée (normalisée par rapport à la consommation du chargement de la valeur hexadécimale 0x00) lors du chargement des seize plus petites valeurs pour les différents codes étudiés. Ces valeurs sont triées par poids de Hamming croissant.

On note sur ce graphe la présence de paliers proportionnels au poids de Hamming du mot chargé pour toutes les courbes sauf pour la parité complémentaire. Ceci est le signe d'une tendance à une forte corrélation entre la puissance consommée et les poids de Hamming du mot chargé. Cependant, on note que pour les registres protégés ces paliers ne sont pas réguliers, illustrant ainsi une corrélation strictement inférieure à 1. Ces résultats sont étroitement liés à la structure des registres ainsi durcis.

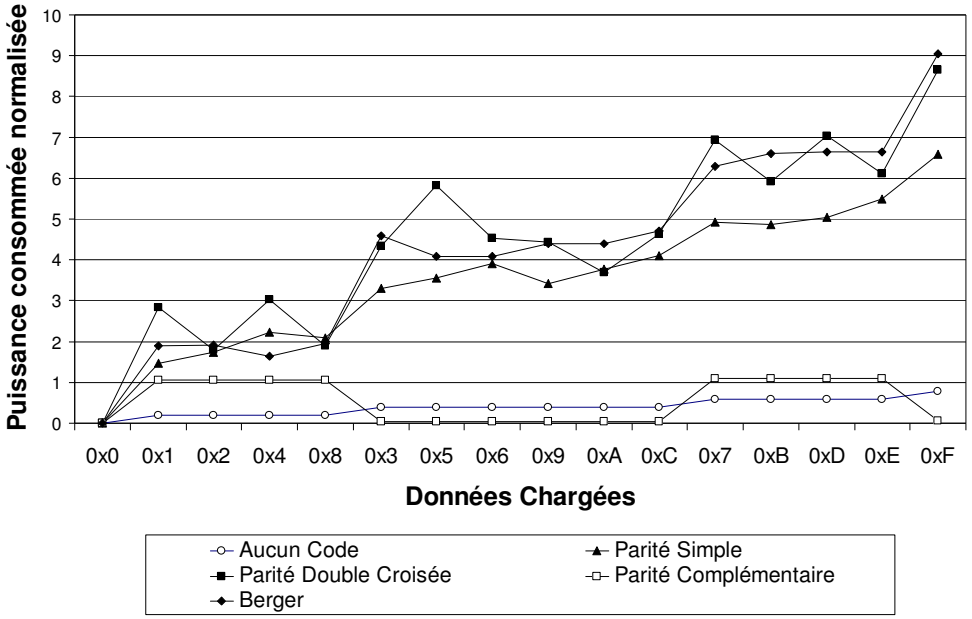

Figure 41 : Puissance consommée par un registre 8 bits durci

Le registre protégé par le code à parité complémentaire présente globalement une faible corrélation entre sa consommation et les données qu'il stocke. Cependant, on observe une assez grande différence entre la corrélation au niveau du banc de bascules et la corrélation dans les codeurs et vérificateurs. Ceci s'explique par la structure du registre durci : le banc de registre stocke toujours de manière non modifiée les données entrantes, cependant, l'ajout des deux bascules stockant des valeurs complémentaires diminue la corrélation à ce niveau. De manière inverse, les codeurs et vérificateurs sont constitués de deux arbres duaux de XOR et XNOR créant simultanément deux transitions opposées (si la sortie de l'arbre XOR passe de 0 à 1, celle de l'arbre XNOR avec les mêmes entrées passera de 1 à 0, et inversement). Ainsi, leur consommation est beaucoup moins dépendante des données. Bien que cette dépendance soit très faible, la consommation des parties combinatoires du registre durci n'est pas constante. En effet, on note la présence de deux valeurs pour les entrées montrées sur la Figure 41. Dans l'implantation réalisée, les arbres de XOR ont été synthétisés par le

XOR à deux entrées (XOR2) des résultats de deux XOR à quatre entrées (XOR4) connectés respectivement aux quatre bits de poids forts et aux quatre bits de poids faible. Ainsi, pour les valeurs de 0 à 15, on peut soit noter une absence de commutation soit la commutation d'un des deux XOR4 et du XOR2. Le nombre de valeurs observé dépend de la taille du registre et de la structure des arbres de XOR et de XNOR, plus particulièrement du nombre de nœuds internes dans les arbres.

Les deux autres codes basés sur la parité sont vraiment proches l'un de l'autre au niveau de la corrélation. Durant le codage, ils présentent une corrélation très proche de 1 ; le code à parité simple est légèrement en dessous. Cette différence peut être expliquée par la présence du XOR2 final dans l'implantation du code à parité simple qui crée un bruit supplémentaire sur la consommation des parties combinatoires du registre durci. Au niveau du banc de bascules constituant le registre, la différence entre ces deux codes s'accroît et s'inverse : la présence d'un bit « bruiteur » de plus dans l'implantation du code à parité double croisée lui donne une corrélation plus faible. Au niveau global, la différence au niveau du registre est prépondérante par rapport à celle observée au niveau des codeurs et vérificateurs. Ainsi, le code à parité double croisée semble être la meilleure de ces deux options.

Le code de Berger bien qu'ayant un taux de détection bien plus élevé présente une très forte corrélation entre sa consommation et les données traitées. Sa définition (juxtaposition aux données de leur poids de Hamming) explique tout naturellement cette forte corrélation. En effet, son implantation est basée sur un arbre d'additionneurs pour le calcul des bits de redondance. Cette structure est donc propice au lien direct entre le poids des données entrantes et le nombre de cellules d'addition activées. Ces arbres d'addition sont utilisés dans les codeurs ainsi que les vérificateurs, justifiant ainsi la forte corrélation à ces niveaux. La corrélation au niveau du registre est directement liée à la corrélation obtenue théoriquement entre le poids de Hamming des données entrantes et celui des données réellement stockées.

Influence de la géométrie des données chargées dans un registre 8 bits durci sur sa consommation

Revenons un peu sur le code à parité simple. Dans ce cas, nous avons vu que la consommation des différents mots de même poids de Hamming différait. Considérons plus particulièrement les mots de poids 1 ayant leur moitié de poids fort nulle, on remarque qu'au niveau de la structure logique ces mots sont équivalents. En effet, dans ces quatre cas, un seul bit commute en entrée d'une des deux portes XOR4. Ceci montre que les différentes entrées d'une porte XOR4 ne sont pas toutes équivalentes. De ce fait, la consommation reflète en partie la géométrie du mot chargé dans le registre.

Au vu de ces différences, nous allons tenter de comparer les différents codes choisis par rapport à la corrélation qui existe entre la consommation du registre durci par ce code et une

quantification de la géométrie du mot chargé. Pour ce faire, nous avons considéré le pourcentage des bits valant 1 situés dans la moitié de poids fort du mot de données ; ceci quantifie le déséquilibre entre le poids de Hamming des deux moitiés du mot de données. Les résultats obtenus avec ce critère sont présentés dans le Tableau 17. Au vu des chiffres apportés par ce tableau, les trois codes à base de parité semblent se comporter de la même manière. Le code de Berger quant à lui présente un comportement différent. Ceci peut tout naturellement s'expliquer par la structure relativement proche des trois premiers codes basée sur des arbres de XOR ou XNOR, tandis que le code de Berger est basé sur des additionneurs.

Tableau 17 : Influence de la géométrie sur les codes détecteurs appliqués à un registre 8 bits

Code	Corrélation			
	Codeur	Bascules	Vérificateur	Global
Parité Simple	0,0685	0,1374	0,0771	0,0472
Parité Double Croisée	0,0672	0,1358	0,0755	0,0462
Parité Complémentaire	0,0672	0,1375	0,0747	0,0464
Berger	0,0673	0,0127	0,0827	0,0793

Influence de la connexion des codeurs et des vérificateurs aux données originales et codées

Nous avons vu précédemment que certains paramètres structurels du registre durci peuvent influer sur la corrélation entre sa consommation et les données qu'il traite. Nous allons ici nous concentrer d'abord sur l'influence du choix de la connexion du mot non codé sur l'arbre de codage. La Figure 41 permet de mettre en évidence que le registre durci ne consomme pas la même quantité de courant, pour un code donné, sur toutes les entrées de même poids de Hamming, et ce même si les mots considérés activent exactement les mêmes portes. La Figure 42 montre les courbes de consommation de trois versions d'un registre protégé par parité simple sur les seize valeurs de poids faible. Ces trois versions diffèrent de la manière dont les arbres de codage et de vérification ont été alimentés par les données à coder ou vérifier. Dans cette étude, on dénote la version originale par « 01234567 » ; ceci représente l'ordre des bits de données lors de leur connexion aux différents arbres. Ainsi, la version « 02461357 » est le résultat de la permutation (1 2 4)o(3 5 6) et « 32107654 » est le résultat de la permutation (0 3)o(1 2)o(4 7)o(5 6).

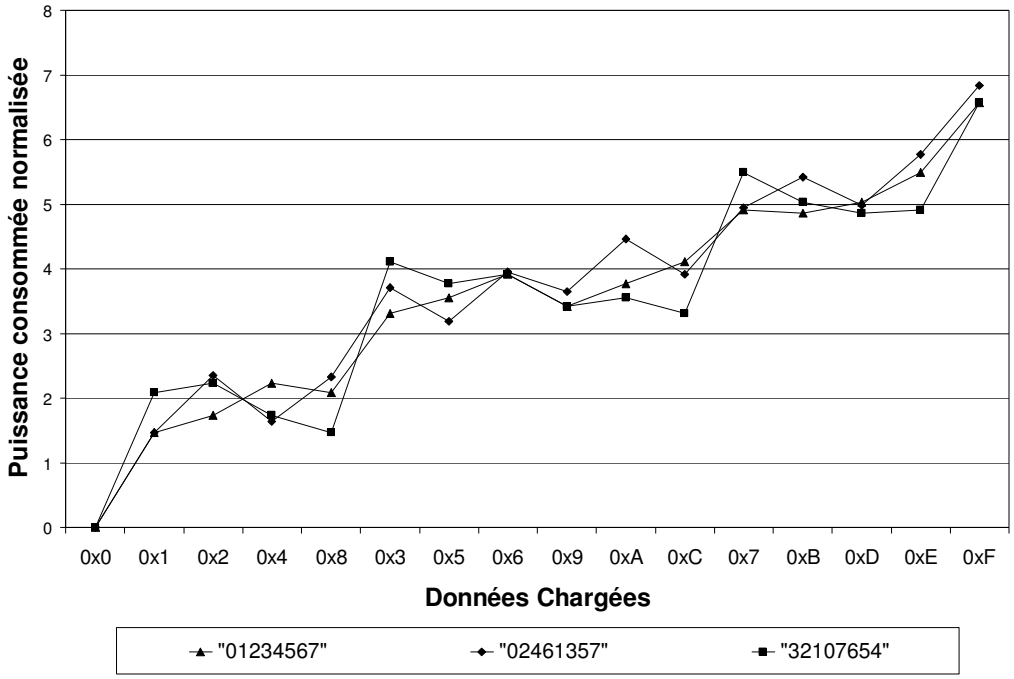

Figure 42 : Influence de la connexion en entrée des codeurs et vérificateurs

Ce graphe montre que la permutation des entrées des arbres de codage et de vérification a bien un effet sur la consommation pour les différents vecteurs d'entrée. Cependant, on note que les valeurs de consommation simulées en présence de ces permutations ne sont pas nouvelles : elles correspondent aux valeurs permutées de la version d'origine. Ceci est très visible si on compare les courbes « 01234567 » et « 32107654 » car la permutation effectuée entre ces deux versions ne fait pas intervenir de valeurs non représentées. La définition du coefficient de corrélation fait que ces permutations n'ont pas d'effet sur la corrélation entre la consommation du registre et les données qu'il stocke. De la même manière, les différences de consommation entre les différentes entrées de même poids de Hamming ne sont que permutées. Il pourrait donc être intéressant d'utiliser des permutations différentes dans les différents codeurs et vérificateur pour tenter de masquer ces différences.

Influence du poids de Hamming des données chargées dans un registre 16 bits durci sur sa consommation

Nous allons maintenant analyser l'effet de la taille du registre sécurisé sur ces valeurs de corrélation. Le Tableau 18 présente les valeurs de corrélation entre la consommation des différentes versions du registre 16 bits sécurisé et le poids de Hamming des données chargées.

Tableau 18 : Comparaison des codes détecteurs sur un registre 16 bits

Code	Surcoût en surface	Corrélation			
		Codeur	Bascules	Vérificateur	Global
Aucun	+ 0 %	n/d	1	n/d	1
Parité Simple	+ 105 %	0,965	0,945	0,949	0,970
Parité Double Croisée	+ 127 %	0,995	0,819	0,926	0,956
Parité Complémentaire	+ 207 %	0,030	0,537	0,024	0,162
Berger	+ 1727 %	0,849	0,970	0,929	0,935

D'un point de vue théorique, il est possible de prédire l'évolution de ces valeurs de corrélation au niveau des bascules du registre durci. Ceci est en relation directe avec le nombre de bits de redondance ajouté par chaque code. Pour les trois codes basés sur la parité, la fraction que représente les bits de redondance tend asymptotiquement vers O(1/n), tandis que pour le code de Berger, ce ratio est asymptotiquement équivalent à un O(Log n/n). Cependant, le calcul théorique de la corrélation entre le poids de Hamming des mots de données et celui des mots codés est de 0,9644 pour le registre à huit bits et de 0,9510 pour le registre à seize bits. De ce point de vue théorique, la corrélation du code de Berger devrait diminuer avec l'augmentation de taille du registre. La structure des codeurs et vérificateurs des différents codes étudiés ne change pas lors de cette mise à une échelle plus grande ; ainsi, leur valeur de corrélation devrait rester sensiblement identique. Les chiffres du Tableau 18 vont dans ce sens. De plus, la comparaison des différents codes reste la même avec cette nouvelle taille. Les codes à parité simple et double croisée sont à peu près équivalents avec un léger avantage pour la parité simple. Tous restent plus intéressants que le code de Berger dans l'optique de la corrélation entre données et consommation. Mais tous trois restent de très loin inférieurs au code à parité complémentaire sur cet aspect de l'étude.

Influence de la géométrie des données chargées dans un registre 16 bits durci sur sa consommation

L'étude de l'évolution de la corrélation entre la consommation du registre et la géométrie des données traitées quand la taille du registre augmente est présentée dans le Tableau 19. On remarque que les commentaires sur l'augmentation de la corrélation avec la taille du registre restent majoritairement valides. Cependant, le code de Berger présente une augmentation des cette corrélation dans des proportions plus grandes. Le code à parité complémentaire voit sa corrélation très fortement diminuer.

Tableau 19 : Influence de la géométrie sur les codes détecteurs appliqués à un registre 16 bits

Code	Corrélation			
	Codeur	Bascules	Vérificateur	Global
Parité Simple	0,0385	0,1140	0,1095	0,0795
Parité Double Croisée	0,0103	0,1898	0,1315	0,0925
Parité Complémentaire	0,0288	0,0002	0,0228	0,0185
Berger	0,4504	0,1378	0,3098	0,2985

3.3.3. Comparaison des codes correcteurs

Après avoir vu comment évoluait la corrélation entre la consommation et les données stockées dans un registre capable de détecter des fautes, nous allons maintenant étudier comment ce dernier se comporte lorsqu'on y implante des capacités de correction. La présentation des résultats de l'étude des codes correcteurs choisis suit le même principe que celle des codes détecteurs. Cependant, nous étudierons tout d'abord le lien entre le poids de Hamming des données et la consommation, sur un registre 8 bits dans un premier temps et sur un registre 16 bits dans un second temps. Enfin, nous étudierons les effets de la géométrie des données sur la consommation.

Influence du poids de Hamming des données chargées dans un registre 8 bits durci sur sa consommation

Les quatre codes choisis sont capables de détecter toutes les erreurs doubles et de corriger les erreurs simples. En présence de multiplicités de fautes plus élevées, la détection peut échouer et la correction donne un résultat erroné. Le Tableau 20 présente les chiffres de corrélation obtenus lors de l'implantation de ces quatre codes sur un registre 8 bits. Comme précédemment, les probabilité de détection et correction sont données pour des fautes de multiplicité quelconque.

Tableau 20 : Comparaison des codes correcteurs sur un registre 8 bits

Code	Surcoût en surface	Proba. de détection	Proba. de correction	Corrélation			
				Codeur	Bascules	Vérificateur	Global
Aucun	+ 0 %	0 %	0 %	n/d	1	n/d	1
HammingV1	+ 301 %	93,8 %	0,366 %	0,716	0,358	0,048	0,070
HammingV2	+ 307 %	93,8 %	0,366 %	0,695	0,247	0,041	0,041
HammingV3	+ 246 %	93,8 %	0,366 %	0,855	0,211	0,149	0,149
Hsiao (13,8)	+ 358 %	96,9 %	0,379 %	0,472	0,209	0,036	0,036

Comme pour les codes détecteurs, la présence du code entraîne une diminution de la corrélation entre la puissance consommée par le registre durci et le poids de Hamming des données qu'on y charge. Cependant, les taux de corrélation sont beaucoup plus faibles. On peut aussi noter que les modifications de la matrice de vérification H qui différencient les différentes versions de Hamming peuvent avoir un impact important sur les valeurs de corrélation obtenues : la version 3 présente une corrélation globale et dans les blocs combinatoires supérieure aux autres versions. On remarque que la corrélation globale est très fortement influencée par la corrélation au niveau du vérificateur. Ceci peut s'expliquer par l'importance de ce bloc dans l'implantation de tels codes. En effet, le bloc de vérification inclut comme précédemment une détection d'erreur en double rail ainsi que les mécanismes de correction des erreurs. Le code de Hsiao n'apporte pas de gain réel par rapport à la version HammingV2 (version présentant les meilleures caractéristique vis-à-vis de la corrélation) : sa corrélation et son taux de détection sont légèrement meilleurs mais pour un coût 50% plus élevé. Globalement, le code HammingV2 est la meilleure de ces quatre options. En terme de détection, il rivalise avec le code de parité complémentaire au niveau de la corrélation (sauf au niveau du codeur) et obtient de meilleurs résultats au niveau de la détection ; cependant, son coût est très supérieur.

Influence du poids de Hamming des données chargées dans un registre 16 bits durci sur sa consommation

Le Tableau 21 présente les valeurs de corrélation obtenues sur les versions 16 bits des registres durcis. On peut noter une forte augmentation de la corrélation globale obtenue pour les trois codes étudiés sur 16 bits. Pour la version HammingV1, ce phénomène est moins important. De ce fait, cette version devient plus intéressante que la version HammingV3.

Tableau 21 : Comparaison des codes correcteurs sur un registre 16 bits

Code	Surcoût en surface	Corrélation			
		Codeur	Bascules	Vérificateur	Global
Aucun	+ 0 %	n/d	1	n/d	1
Hamming V1	+ 289 %	0,110	0,284	0,137	0,153
Hamming V2	n/d	n/d	n/d	n/d	n/d
Hamming V3	+ 245 %	0,224	0,286	0,342	0,342
Hsiao (22,16)	n/a	0,234	0,009	0,395	0,379

Influence de la géométrie des données chargées dans un registre durci sur sa consommation

Le Tableau 22 montre l'influence de la géométrie du mot de données sur la corrélation entre consommation et données traitées. La géométrie du mot a été quantifiée comme précédemment par la

fraction des bits à 1 présents dans la moitié de poids forts du mot de données. Les conclusions obtenues précédemment se confirment aussi sur ces quelques codes correcteurs : sur 8 bits, ils semblent tous équivalents, mais lors du passage sur 16 bits la corrélation augmente. Cette augmentation est beaucoup plus forte pour la version HammingV3, rendant cette version du code la moins intéressante sur ce critère.

Tableau 22 : Influence de la géométrie sur les codes correcteurs

Code	Corrélation (8 Bits / 16 Bits)			
	Codeur	Bascules	Vérificateur	Global
Hamming V1	0,0674 / 0,0347	0,1356 / 0,0762	0,0766 / 0,0923	0,0470 / 0,0951
Hamming V2	0,0679 / n/d	0,1359 / n/d	0,0763 / n/d	0,0473 / n/d
Hamming V3	0,0669 / 0,0899	0,1351 / 0,1721	0,0765 / 0,1239	0,0470 / 0,1147
Hsiao	0,0690 / 0,1687	0,1378 / 0,0001	0,0757 / 0,0659	0,0462 / 0,0630

3.3.4. Comparaison des codes M parmi N

Pour les registres codés avec un codage en M parmi N, l'ensemble des poids de Hamming des mots chargés présente une variance nulle (car ils sont tous identiques) ; de ce fait, le calcul du coefficient de corrélation est impossible. Pour réussir à quantifier les variations de la consommation électrique des différentes versions du registre, nous avons décidé de considérer la variance relative de la consommation (quotient de la variance de la consommation et de sa valeur moyenne). Le Tableau 23 présente les valeurs ainsi obtenues pour les codes 1,2 et 3 parmi 8 ou 16. Une faible valeur reflète une consommation moins dépendante de la donnée. On note que le codage en 1 parmi N présente de meilleures caractéristiques globales. On observe aussi une différence notable entre les deux composantes de la consommation : la partie registre a une consommation bien plus uniforme.

Tableau 23 : Comparaison de codes M parmi N

Code	Variance relative (N = 8)			Variance relative (N = 16)		
	Registre	Vérificateur	Globale	Registre	Vérificateur	Globale
1 parmi N	0,012	0,532	0,056	0,003	0,260	0,112
2 parmi N	0,050	0,276	0,197	0,099	0,300	0,213
3 parmi N	0,042	0,227	0,164	0,039	0,209	0,155

3.3.5. Synthèse des conclusions

Nous avons donc vu que les différents codes correcteurs et détecteurs étudiés présentent différentes caractéristiques de corrélation entre les données et la consommation. Nous avons pu isoler un code détecteur ayant des valeurs de corrélation très intéressantes du point de vu attaques en puissance, car il présente une très faible dépendance entre les données traitées et la consommation du registre ; il s'agit du code à parité complémentaire.

Nous avons aussi vu que les codes correcteurs ont des comportements très proches des codes détecteurs étudiés précédemment. Certains d'entre eux présentent des caractéristiques de corrélation très proche de la parité complémentaire (HammingV2 sur 8 bits) et présentent des taux de détection bien plus élevés (un gain de près de 20% de détection dans ce cas). Ces codes correcteurs pourraient être une bonne option même en l'absence de contrainte de correction pour peu que l'ont puisse payer le surcoût supplémentaire.

Nous avons aussi vu que la manière de connecter les codeurs et les vérificateurs aux données entrantes ou au registre interne peut influer sur la consommation du circuit. Les varier pourrait permettre une diminution de la corrélation et un meilleur comportement vis-à-vis des attaques en puissance.

Lors de la protection de registres d'états, une des techniques est de conserver un codage en M parmi et N ; la vérification se faisant par comptage du nombre de 1. Nous avons montré que le code en 1 parmi N semble être la meilleure option.

3.4. Analyse niveau transistor de registres durcis

Nous avons utilisé le second flot d'analyse pour obtenir des résultats plus proche du silicium et voir si les conclusions obtenues au niveau portes logiques étaient toujours valides. Cette section va donc présenter les résultats obtenus lors de cette étude ainsi qu'une comparaison avec l'étude précédente.

3.4.1. Présentation des cas d'étude

Lors de cette étude, nous avons utilisé les mêmes versions du registre que lors de l'étude présentée précédemment. Les différentes versions du registre ont été synthétisées en utilisant deux bibliothèques technologiques différentes.

Nous étudierons tout d'abord les résultats au niveau transistors des versions utilisant la bibliothèque C35_CORELIB de technologie AMS. Ceci permettra de comparer les résultats obtenus lors de l'étude précédente avec la même technologie cible.

Pour étudier l'influence de cette dernière, nous avons aussi synthétisé les versions à 8 bits du registre sur la bibliothèque HCMOS9 de technologie ST Microelectronics. Les résultats ainsi obtenus seront présentés plus tard dans cette section.

Lors des simulations au niveau transistors, nous avons pu obtenir seulement un profil de consommation du registre complet (bascules, codeur et vérificateurs) lors du chargement de données. Ainsi, bien qu'il nous soit possible de connaître la consommation instantanée du registre, nous ne pouvons pas séparer la contribution de chaque sous-bloc. L'analyse des valeurs de consommation nous a tout de même permis d'établir différentes valeurs de corrélation : la corrélation totale sur la trace enregistrée, ainsi que les corrélations instantanées minimale et maximale.

3.4.2. Comparaison des codes détecteurs

L'étude des résultats obtenus regroupe plusieurs aspects. Nous allons dans un premier temps étudier le rapport entre le poids de Hamming des données et la consommation du registre. Ensuite, nous verrons comment la géométrie et la taille des données chargées influent sur la consommation du registre durci.

Influence du poids de Hamming des données chargées dans un registre 8 bits durci sur sa consommation

Le Tableau 24 présente les valeurs de corrélation entre le poids de Hamming des données chargées dans un registre 8 bits durci et sa consommation, obtenues lors de l'analyse au niveau transistors des codes détecteurs choisis.

Tableau 24 : Comparaison des codes détecteurs sur un registre 8 bits au niveau transistors

Code	Corrélation		
	Instantanée Minimale	Instantanée Maximale	Totale
Parité Simple	0,0003	0,9990	0,9348
Parité Double Croisée	0,1123	0,9989	0,9198
Parité Complémentaire	0,0001	0,9974	0,7805
Berger	0,0104	0,9769	0,8587

Comme précédemment, on peut noter que les valeurs de corrélation obtenues par les codes à parité simple et à parité double croisée sont très élevées. Le code à parité complémentaire présente une corrélation très inférieure aux autres, bien que celle-ci soit beaucoup plus élevée que l'évaluation faite au niveau portes. On note aussi une amélioration du code de Berger lors du changement de niveau d'abstraction pour les simulations. En effet, sa valeur de corrélation est plus faible et située entre les deux groupes précédemment cités.

La corrélation instantanée passe pour tous les codes par un maximum très proche de un. Ce critère ne permet pas de les discriminer. Cependant, si on observe la valeur minimale de cette corrélation instantanée, certaines différences peuvent apparaître. Bien que cette valeur semble globalement très faible, le code à parité double croisée présente un minimum de corrélation instantanée bien supérieur aux autres codes. Ceci le rend donc moins intéressant que le code à parité simple par rapport à cet aspect. Le code à parité complémentaire est à nouveau celui ayant la plus petite valeur.

Influence du poids de Hamming des données chargées dans un registre 16 bits durci sur sa consommation

Le Tableau 25 présente les valeurs de corrélation entre le poids de Hamming des données chargées dans un registre 16 bits durci et sa consommation. Au niveau portes logiques, on pouvait observer une augmentation de la corrélation de tous les codes et une conservation de leurs différences, lors du passage d'un registre 8 bits à un registre 16 bits.

Tableau 25 : Comparaison des codes détecteurs sur un registre 16 bits au niveau transistors

Code	Corrélation		
	Instantanée Minimale	Instantanée Maximale	Totale
Parité Simple	0,2526	0,9986	0,9816
Parité Double Croisée	0,4305	0,9972	0,9749
Parité Complémentaire	0,2266	0,9974	0,9581
Berger	0,2180	0,9981	0,9747

Les résultats au niveau transistors montrent également une augmentation de la corrélation et les différences entre les codes sont beaucoup moins marquées que sur les versions 8 bits de ces registres. On note cependant que le code à parité complémentaire reste le plus intéressant. Comme noté sur 8 bits, les codes à parité simple et à parité double croisée semblent assez proches l'un de l'autre au niveau de la corrélation de la puissance totale ; cependant, leur corrélation instantanée minimale les sépare toujours.

Influence de la géométrie des données chargées dans un registre durci sur sa consommation

Le Tableau 26 présente les valeurs de corrélation obtenues lors de l'analyse au niveau transistors de l'impact de la géométrie des données chargées dans un registre durci.

Comme au niveau portes, la corrélation entre la consommation et la géométrie du mot est globalement assez faible sur 8 bits. On observe cependant certaines différences entre les codes étudiés : le code de Berger se révèle être le meilleur suivi, par les codes à parité simple et double croisée. A ce niveau, le code à parité double complémentaire est le moins performant. Lors du passage à un registre de 16 bits de large, l'augmentation observée au niveau portes est dans ce cas beaucoup plus prononcée et masque quasiment les différences entre les codes. Cependant, la corrélation instantanée minimale du registre 16 bits permet de les départager : le code de Berger est ici aussi le plus intéressant, suivi par le code à parité complémentaire et le code à parité simple.

Tableau 26 : Influence de la géométrie sur les codes détecteurs appliqués à un registre au niveau transistors

Code	Corrélation (8 bits / 16 bits)		
	Instantanée Minimale	Instantanée Maximale	Totale
Parité Simple	0,0000 / 0,2503	0,1432 / 0,9339	0,0178 / 0,8174
Parité Double Croisée	0,0000 / 0,3192	0,3154 / 0,9887	0,0340 / 0,7977
Parité Complémentaire	0,0000 / 0,2178	0,3428 / 0,9338	0,1266 / 0,8111
Berger	0,0000 / 0,1096	0,2109 / 0,9339	0,0003 / 0,8071

3.4.3. Comparaison des codes correcteurs

Cette étude au niveau transistors a aussi été utilisée pour comparer les différents codes correcteurs choisis. Les résultats ainsi obtenus vont maintenant être discutés.

Influence du poids de Hamming des données chargées dans un registre 8 bits durci sur sa consommation

Le Tableau 27 présente les valeurs de corrélation entre le poids de Hamming des données chargées dans un registre 8 bits durci et sa consommation.

Les quatre codes correcteurs ici étudiés semblent se regrouper selon deux ensembles : le code HammingV3 et les autres. En effet, ce code présente des caractéristiques de corrélation inférieures aux trois autres codes qui sont très proches les uns des autres, suggérant une plus faible influence de la structure de la matrice de vérification. On remarquera aussi que les valeurs de corrélation obtenues sont ici encore largement supérieures à celles obtenues lors de l'analyse au niveau portes logiques. Au

niveau portes, la version HammingV2 semblait être la plus intéressante ; elle est désormais dans la moyenne.

Tableau 27 : Comparaison des codes correcteurs sur un registre 8 bits au niveau transistors

Code	Corrélation		
	Instantanée Minimale	Instantanée Maximale	Totale
HammingV1	0,6502	0,9982	0,9910
HammingV2	0,6564	0,9979	0,9914
HammingV3	0,0668	0,9506	0,8819
Hsiao	0,6825	0,9988	0,9936

Influence du poids de Hamming des données chargées dans un registre 16 bits durci sur sa consommation

Le Tableau 28 présente les valeurs de corrélation entre le poids de Hamming des données chargées dans un registre 16 bits durci et sa consommation, qui ont été obtenues lors de l'analyse au niveau transistors des codes correcteurs choisis.

Tableau 28 : Comparaison des codes correcteurs sur un registre 16 bits au niveau transistors

Code	Corrélation		
	Instantanée Minimale	Instantanée Maximale	Totale
HammingV1	0,6346	0,9958	0,9793
HammingV2	n/d	n/d	n/d
HammingV3	0,6551	0,9944	0,9711
Hsiao	0,6383	0,9684	0,9643

Lors du passage d'une taille de registre de 8 à 16 bits, on remarque ici deux choses. Les trois codes correcteurs étudiés ont des comportements très similaires. Contrairement à ce qui a été constaté dans les autres cas, on note ici une légère diminution des valeurs de corrélation.

Influence de la géométrie des données chargées dans un registre durci sur sa consommation

Le Tableau 29 présente les valeurs de corrélation obtenues lors de l'analyse au niveau transistors de l'impact de la géométrie des données chargées dans un registre ayant des capacités de correction.

Tableau 29 : Influence de la géométrie sur les codes correcteurs au niveau transistors

Code	Corrélation (8 bits / 16 bits)		
	Instantanée Minimale	Instantanée Maximale	Totale
HammingV1	0,0000 / 0,5337	0,2254 / 0,9345	0,0188 / 0,8118
HammingV2	0,0000 / n/d	0,2412 / n/d	0,0499 / n/d
HammingV3	0,0000 / 0,6411	0,3950 / 0,9709	0,0109 / 0,7882
Hsiao	0,0000 / 0,6936	0,2905 / 0,9347	0,0116 / 0,7620

Les conclusions obtenues lors de l'étude au niveau portes logiques restent ici valides. En effet, les valeurs de corrélation obtenues pour les différents codes correcteurs étudiés sont du même ordre de grandeur. Seule la version HammingV2 semble moins performante sur 8 bits. Lors du passage à un registre de taille plus grande, les valeurs de corrélation augmentent fortement restant dans la tendance observée pour les codes détecteurs. On remarque cependant que les trois codes étudiés sur 16 bits sont à peu près équivalents en terme de corrélation.

3.4.4. Influence de la technologie

Le Tableau 30 présente les valeurs de corrélation entre le poids de Hamming des données chargées dans un registre 8 bits durci et sa consommation en utilisant la technologie ST.

Tableau 30 : Comparaison des codes détecteurs sur un registre 8 bits avec la technologie ST

Code	Corrélation		
	Instantanée Minimale	Instantanée Maximale	Totale
Parité Simple	0,0005	0,9927	0,6688
Parité Double Croisée	0,1123	0,9989	0,9198
Parité Complémentaire	0,0001	0,9987	0,3705
Berger	0,2977	0,9865	0,4253

Les quatre codes détecteurs étudiés se distinguent les uns des autres au niveau de la corrélation globale. On note une conservation des tendances fortes : la corrélation du code à parité double croisée reste très forte tandis que celle du code à parité complémentaire reste faible. Les deux autres codes présentent des valeurs de corrélation intermédiaires. Ainsi, le code à parité simple qui jusque là avait des propriétés très proches du code à parité double croisée s'en distingue sur cette technologie.

3.4.5. Synthèse des conclusions

Cette étude au niveau transistors confirme certaines des conclusions de l'analyse au niveau portes logiques. En effet, les différents codes étudiés présentent globalement les mêmes caractéristiques qualitatives que celles observées au niveau portes logiques et ce même si les valeurs de corrélation obtenues sont parfois assez éloignées.

Ainsi, le code à parité complémentaire qui présentait au niveau portes logiques les caractéristiques les plus intéressantes d'un point de vue attaque en puissance conserve globalement sa position : en effet, il reste le code ayant les meilleures caractéristiques dans la plupart des cas étudiés. Cependant, les fortes variations observées sur les valeurs de corrélation lors du changement de niveau d'abstraction ont aussi fortement réduit son avance sur les autres et l'ont parfois même rendu moins performant comme pour l'influence de la géométrie des données. Ce changement de niveau d'abstraction apporte peut-être une précision plus pointue ainsi qu'une meilleure fiabilité des résultats ; elle demande cependant de descendre plus loin dans le flot de conception. Ainsi, il parait judicieux d'utiliser le niveau le plus haut pour éliminer les moins performants et de départager les restants avec un niveau d'abstraction plus bas.

Nous avions aussi vu que les codes correcteurs avaient des comportements très proches des codes détecteurs étudiés. Certains d'entre eux présentaient des caractéristiques de corrélation très proches de la parité complémentaire et des taux de détection bien plus élevés. Certains d'entre eux vont jusqu'à avoir de meilleures valeurs de corrélation au niveau transistors. Ainsi certains codes correcteurs deviennent de réelles alternatives aux codes détecteurs à la fois en terme de capacité de détection que d'influence sur les canaux cachés.

Nous avons aussi noté que le choix de la technologie cible influe énormément sur les valeurs de corrélation simulées lors de cette étude. Ceci rend d'autant plus difficile la quantification de manière absolue de l'impact des différents codes sur la sensibilité aux attaques en puissance.

Chapitre 4 : Evaluation de la robustesse de quelques primitives de chiffrement

Nous avons étudié précédemment l'influence de certains codes détecteur ou correcteur sur la sensibilité d'un registre aux attaques en puissance, en étudiant la corrélation entre la puissance consommée lors du chargement de données et une quantification de certains aspects de celles-ci. Nous en avons déduit que l'ajout des codes détecteurs ou correcteurs avait tendance à réduire cette corrélation et donc à diminuer la sensibilité aux attaques en puissance. Cette conclusion reste-t-elle vraie lorsqu'on essaie d'attaquer des primitives cryptographiques? Nous tenterons de répondre à cette question dans ce chapitre, en étudiant aussi l'impact des protections utilisées vis-à-vis des attaques par fautes en elles-mêmes. Pour ce faire, nous étudierons ici une S-Box AES et des coprocesseurs de chiffrement AES et RSA. La différence avec l'étude précédente est la présence de la logique de calcul (combinatoire) qui peut être prépondérante.

4.1. Etude de S-Box AES durcies

Le premier cas d'étude regroupe différentes versions d'une boîte de substitution (S-Box) AES. Cette section se divise en trois parties : dans un premier temps, nous décrirons brièvement l'algorithme AES et présenterons les différentes versions utilisées ; ensuite, nous étudierons leur sensibilité d'abord face aux attaques par fautes, puis face aux attaques en puissance.

4.1.1. Présentation des S-Box durcies

4.1.1.1. Algorithme AES

L'algorithme AES (Advanced Encryption Stardard), aussi connu sous le nom de Rijndael (du nom de ses deux inventeurs Daemen et Rijmen), a depuis quelques années remplacé l'algorithme DES (Data Encryption Standard) dans de nombreuses applications sécurisées. L'AES est un algorithme symétrique : la même clef est donc utilisée pour les opérations de chiffrement et de déchiffrement.

Les données sont chiffrées par bloc de taille fixe de 128 bits, soit 16 octets. La clef utilisée peut être de plusieurs tailles 128, 192 ou 256 bits. Pour les besoins de l'algorithme, le bloc de données

est représenté sous la forme d'une matrice 4x4 d'octets. Cette matrice, nommée état, sera modifiée par toutes les transformations réalisées lors du chiffrement ou du déchiffrement. La définition des lignes et des colonnes sur lesquelles agissent certaines de ces transformations est basée sur cette représentation matricielle de l'état. Les opérations de chiffrement et de déchiffrement sont découpées en rondes, ensemble de transformations répétées successivement lors du déroulement de l'algorithme. Le nombre d'itérations de ces rondes dépend de la taille de clef utilisée : 10, 12 et 14 fois pour respectivement 128, 192 et 256 bits. Le pseudo code de la Figure 43 présente la succession de ces transformations.

```
01 – KeyExpansion
02 – Initial Round
03 –          AddRoundKey
04 – For NbRound-1
05 –          SubBytes
06 –          ShiftRows
07 –          MixColumns
08 –          AddRoundKey
09 – Final Round
10 –          SubBytes
11 –          ShiftRows
12 –          AddRoundKey
```

Figure 43 : Algorithme AES

L'étape « KeyExpansion » permet à partir de la clef de chiffrement initiale de générer les clefs utilisées durant les différentes rondes. Chaque clef de ronde ainsi générée a le même format que le bloc de chiffrement, d'une taille de 128 bits organisés en une matrice 4x4 d'octets.

L'étape « AddRoundKey » consiste en l'ajout de la clef à l'état courant. Chaque octet de l'état suivant est le résultat du XOR bit à bit des octets correspondants de l'état courant et de la clef de ronde.

L'étape « SubBytes » est une transformation non linéaire qui modifie chaque octet de l'état courant de manière indépendante. Chaque octet de l'état suivant est ainsi le résultat d'une transformation affine de l'inverse multiplicatif de l'octet correspondant de l'état courant, présentée en Figure 44 où [x0, …, x7] représente l'inverse multiplicatif de l'octet transformé. La table regroupant l'ensemble des valeurs possibles de cette fonction est appelée S-Box.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} x0 \\ x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 44 : Transformation affine de « SubBytes »

L'étape « ShiftRows » est une permutation circulaire des éléments de chaque ligne de l'état courant. Le décalage se fait par octet, le nombre d'octets par lequel une ligne est décalée dépend de l'indice de la ligne. Ainsi, la première ligne n'est pas modifiée, les suivantes sont décalées respectivement de 1, 2 et 3 octets.

L'étape « MixColumn » est la multiplication de chaque colonne de l'état courant par un polynôme donné.

En observant la Figure 43, on remarque que chaque substitution « SubBytes » est précédée par un ajout de clef de ronde « AddRoundKey ». Nous avons vu dans le chapitre 1 qu'une attaque de type DPA sur une implantation d'un algorithme de chiffrement nécessitait la définition d'une fonction de sélection (fonction simple dont le résultat dépend d'entrées connues et d'un fragment de la clef recherchée). La succession de ces deux étapes constitue une excellente fonction de sélection. En effet, l'attaque d'une implantation de l'algorithme AES se déroule comme suit. Dans un premier temps, on sélectionne un octet de l'état et donc le bloc « AddRoundKey » + « SubBytes » correspondant. Ce bloc sélectionné est ensuite attaqué par DPA permettant ainsi d'obtenir un octet de la clef de ronde. Cette opération est répétée sur les autres blocs et rondes pour obtenir assez d'informations pour régénérer la clef initiale. C'est pour cette raison que nous avons sélectionné comme premier cas d'étude d'une primitive cryptographique cette fonction de sélection : le bloc « AddRoundKey » + « SubBytes ».

4.1.1.2. Implantation et protections matérielles choisies

Parmi différentes techniques possibles pour implanter notre S-Box, nous avons choisi une méthode simple d'expression sous forme de somme de produits des fonctions de génération des huit bits de sortie, l'optimisation en terme de portes logiques étant laissée à l'outil de synthèse. L'ajout de

la clef est fait trivialement par un XOR entre les deux entrées du bloc avant le bloc de substitution, comme le montre la Figure 45.

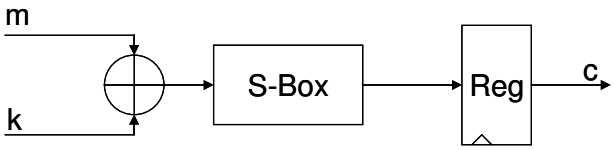

Figure 45 : Schéma du bloc implanté

Nous avons ensuite créé plusieurs versions de ce bloc, incluant chacune un code correcteur ou détecteur différent. Ces différentes versions peuvent être regroupées en deux ensembles : celles sans protection de la logique combinatoire et celles avec cette protection. L'ajout du code a été réalisé par modification de la description comportementale du bloc en VHDL. Pour assurer la conservation des sous-blocs de redondance matérielle, nous avons adopté une structure hiérarchique qui est conservée lors de la synthèse. La Figure 46, respectivement Figure 47, présente la structure hiérarchique des blocs durcis par un code détecteur, respectivement correcteur. Les blocs de vérification « Chk » sont implantés pour générer des signaux d'erreur double rail. Les blocs de correction, de prédiction et de vérification en pointillés sur les schémas ne sont présents que pour les versions où le bloc combinatoire est lui aussi protégé.

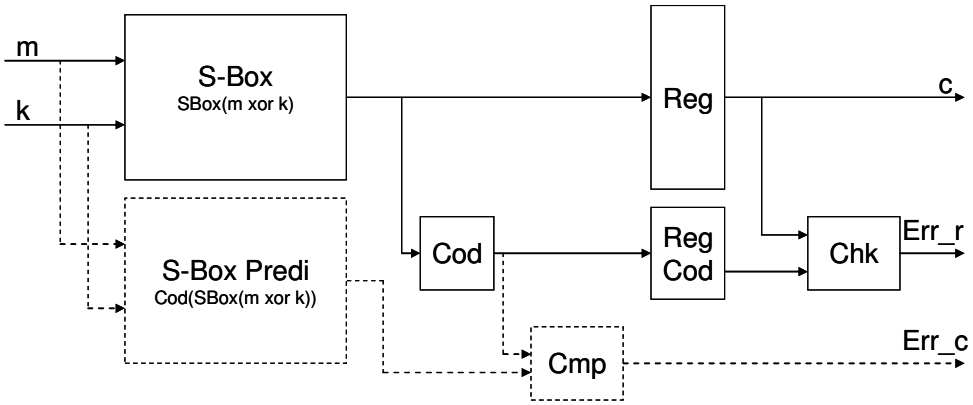

Figure 46 : Schéma des blocs durcis par code détecteur

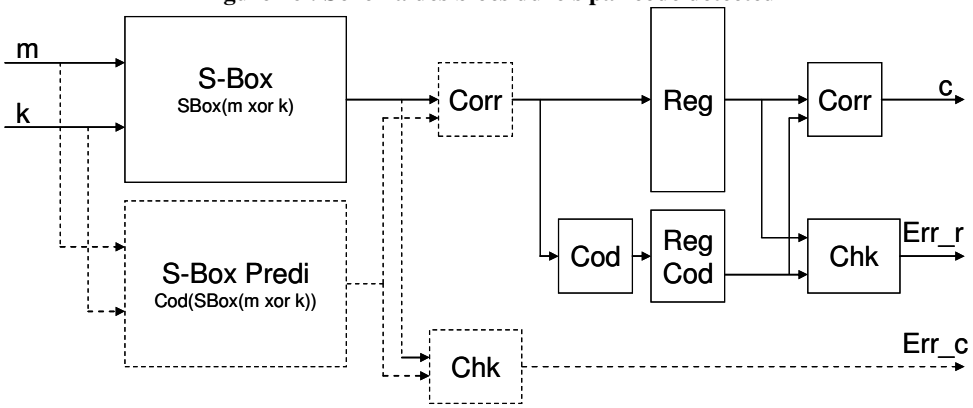

Figure 47 : Schéma des blocs durcis par code correcteur

Les codes utilisés pour cette étude sont les mêmes que ceux utilisés pour l'étude précédente, à savoir : le code à parité simple, le code à parité double croisée, le code à parité double complémentaire, le code de Berger, ainsi que les codes correcteurs Hamming1, Hamming2, Hamming3 et Hsiao. Le Tableau 31 présente les coûts d'implantation de ces protections sur cette S-Box.

Tableau 31 : Surface occupée par les différentes versions de la S-Box (en % de la Référence)

Code	Logique comb. Protégée	Logique comb. Non protégée
Aucun Code (Référence)	100 %	100 %
Parité Simple	200 %	103 %
Parité Double Croisée	204 %	105 %
Parité Double Complémentaire	206 %	108 %
Berger	235 %	122 %
Hamming1	225 %	116 %
Hamming2	224 %	116 %
Hamming3	218 %	113 %
Hsiao	235 %	120 %

4.1.2. Sensibilité aux attaques par fautes

Lors de l'implantation des codes correcteurs et détecteurs étudiés, nous avons protégé suivant les versions soit le registre stockant le résultat, soit ce même registre et la logique combinatoire de calcul. Le deuxième type d'implantation permet de détecter des erreurs dans les deux parties du circuit.

Nous avons vu précédemment que les probabilités de détection des différents codes sont variées et sont récapitulées dans le Tableau 32. Les valeurs indiquées correspondent à la probabilité de détecter des fautes de multiplicité quelconque de répartition uniforme.

Tableau 32 : Probabilité de détection des codes correcteurs et détecteurs étudiés

Code Correcteur	Hamming1	Hamming2	Hamming3	Hsiao
Proba. Détection	94 %	94 %	94 %	97 %
Code Détecteur	Parité Simple	Parité Double Croisée	Parité Double Complémentaire	Berger
Proba. Détection	50 %	74 %	75 %	93 %

De la même manière, les protections implantées au niveau de la partie combinatoire permettent de détecter dans les mêmes proportions les inversions de multiplicité quelconque parmi les bits du résultat.

4.1.3. Sensibilité aux attaques en puissance

Pour évaluer l'impact des protections ajoutées sur la sensibilité aux attaques en puissance, nous avons réalisé une attaque DPA sur chacune des versions de la S-Box. Pour ce faire, nous avons réutilisé le flot de simulation au niveau transistors présenté dans le chapitre précédent et remplacé la dernière étape de calcul de corrélation par une analyse DPA.

Cette étude de sensibilité se sépare en deux temps. Dans le premier temps, nous avons tenté d'attaquer par DPA chaque version protégée de la S-Box en se basant sur des simulations exhaustives au niveau transistors. Ensuite, nous avons tenté d'affiner cette étude en essayant de déterminer le nombre minimal de traces nécessaires pour que l'attaque DPA réussisse pour chacune des versions de la S-Box.

Nous avons donc développé un programme en C capable d'analyser les traces de simulations obtenues et de réaliser l'attaque DPA à partir de celles-ci. Le partitionnement de l'ensemble des traces lors de l'attaque DPA est défini par un paramètre de l'algorithme. Celui-ci, s'il est positif et inférieur ou égal à l'indice du bit de poids fort du mot protégé, indique le bit utilisé pour le partitionnement ; dans les autres cas, le partitionnement se fait sur le poids de Hamming des données codées. Dans le cas d'une version protégée de la S-Box, il est ainsi possible de sélectionner un des bits de redondance. Nous avons donc réalisé ces attaques sur les différentes versions de la S-Box avec tous les paramétrages possibles pour différentes clefs. Dans tous les cas, un vote parmi les octets de clefs donnés par ces attaques donne toujours la clef utilisée lors de l'obtention des traces attaquées.

Le Tableau 33 présente le pourcentage des paramétrages d'attaque DPA ayant donné la bonne clef (l'octet complet) pour chaque version de la S-Box, ainsi que la hauteur relative moyenne du second pic DPA. Ces chiffres montrent que la puissance consommée par la logique de protection utilisée lors de l'implantation de codes correcteurs et détecteurs n'empêche pas l'attaque DPA de trouver la clef utilisée par la S-Box. En effet, on peut noter des pourcentages d'attaques réussies très élevés, supérieurs à 79% pour toutes les versions attaquées sauf une, la S_Box partiellement protégée avec le code de Hsiao, et très majoritairement supérieurs à 95%. On note de plus des taux de réussite supérieurs lors de la protection complète de la S-Box. Ceci tend à indiquer une augmentation de la sensibilité face à la DPA lors de l'ajout de protections sur les éléments combinatoires. La hauteur relative du second pic DPA permet de départager plus finement les codes des taux de réussite

d'attaques équivalents. Cependant, on remarque que le pic secondaire exhibé lors de l'attaque DPA voit son amplitude relative augmenter sur les versions protégées. Ceci reflète donc une diminution de la sensibilité face à l'attaque DPA des versions protégées par codes. On note que d'une manière générale les codes correcteurs sont globalement plus performants que les codes détecteurs, exception faite du Hamming3. Le code détecteur de Berger fait exception en s'octroyant la meilleure place dans ce classement. Etrangement, le code à parité double complémentaire, qui était le meilleur lors de l'étude précédente, apporte ici à peine plus que les deux autres codes détecteurs à base de parité. Ceci peut s'expliquer par la contribution plus faible des arbres de codage et vérification dans la consommation globale du circuit, cette tendance s'accentuant lors de la presque duplication du bloc combinatoire nécessaire à sa protection. De plus, l'attaque DPA a tendance à supprimer les contributions identiques dans la consommation du circuit, ce que ne fait pas l'analyse de corrélation.

Tableau 33 : Efficacité de l'attaque DPA sur les différentes S-Box

	% paramètres trouvant la clef	Hauteur relative du 2nd pic DPA	% paramètres trouvant la clef (Logique Protégée)	Hauteur relative du 2nd pic DPA (Logique Protégée)
Aucun code	100 %	45 %	100 %	45 %
Parité simple	100 %	52 %	100 %	46 %
Parité croisée	100 %	55 %	100 %	51 %
Parité complém.	100 %	52 %	100 %	47 %
Berger	79 %	62 %	86 %	58 %
Hamming1	92 %	64 %	95 %	60 %
Hamming2	79 %	63 %	91 %	64 %
Hamming3	100 %	66 %	100 %	59 %
Hsiao	54 %	68 %	94 %	72 %

Le Tableau 34 présente les taux de réussite des attaques DPA en fonction du type de bit sélectionné par leur paramétrage ; lors d'une répartition selon le poids de Hamming, l'attaque donnait toujours la bonne clef. La Figure 48 illustre graphiquement ce taux de réussite en fonction du type de partitionnement effectué. Cette analyse montre que d'une manière générale les attaques ne considérant pas les bits de redondance fonctionnent sur les versions protégées et restent très efficaces ; elles fonctionnent dans tous les cas pour tous les codes sauf deux : le code de Berger avec un taux de réussite quand même très élevé et le code de Hsiao où l'attaque échoue beaucoup plus souvent pour la version sans protection de la logique combinatoire. On remarque que les attaques sur les bits de redondance réussissent dans une très grande partie des cas ; on retrouve aussi une sensibilité

supérieure à l'attaque en cas de protection de la partie combinatoire. Ceci montre que les bits de redondance peuvent eux aussi être ciblés par l'attaque DPA et que l'exactitude des informations obtenues grâce à eux est fortement probable, bien que quelque peu inférieure. Il a aussi été remarqué que la valeur moyenne de la hauteur du second pic DPA ne varie que très faiblement en fonction du type de bit attaqué.

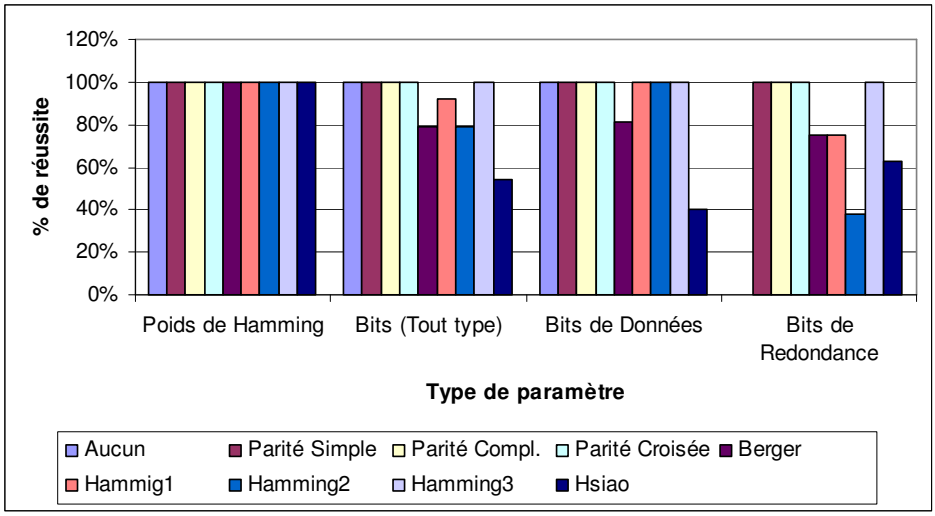

Figure 48 : Taux de réussite des attaques en fonction du type de répartition (Logique non protégée)

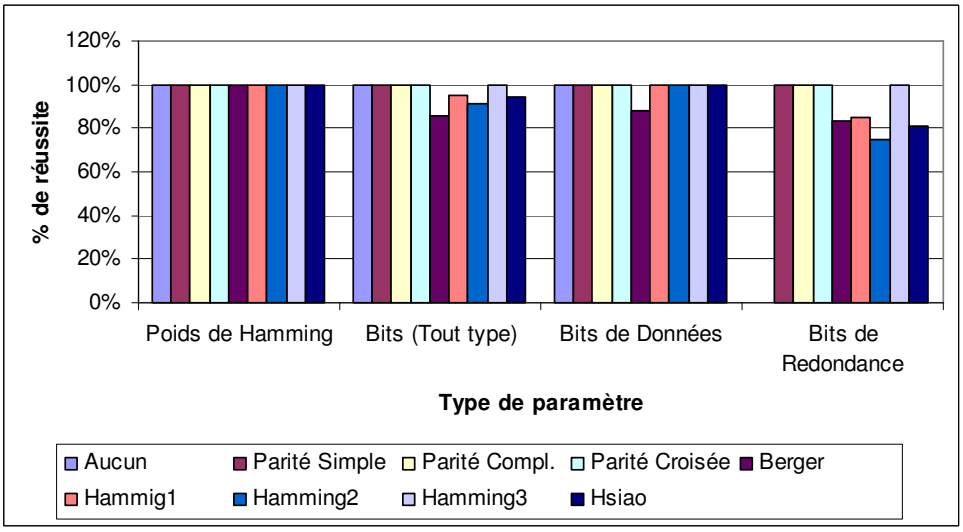

Figure 49 : de réussite des attaques en fonction du type de répartition (Logique protégée)

Nous avons donc ensuite tenté de déterminer le nombre minimum de traces nécessaires pour que l'attaque DPA réussisse dans 90% des cas. Pour ce faire, nous avons donc modifié le programme d'attaque DPA pour qu'il prenne en paramètres les traces à utiliser. Par exécutions successives de l'attaque avec des paramètres différents, nous avons pu obtenir les valeurs listées dans le Tableau 35. Ces valeurs sont approximées à la dizaine la plus proche.

Tableau 34 : Influence du type de bit ciblé lors de l'attaque DPA sur son efficacité

Code	% paramètres trouvant la clef - Bits de données	% paramètres trouvant la clef - Bits redondants	% paramètres trouvant la clef - Bit de données (Logique Protégée)	% paramètres trouvant la clef - Bits redondants (Logique Protégée)
Aucun code	100 %	n / d	100 %	n / d
Parité simple	100 %	100 %	100 %	100 %
Parité croisée	100 %	100 %	100 %	100 %
Parité complém.	100 %	100 %	100 %	100 %
Berger	81 %	75 %	88 %	83 %
Hamming1	100 %	75 %	100 %	85 %
Hamming2	100 %	38 %	100 %	75 %
Hamming3	100 %	100 %	100 %	100 %
Hsiao	40 %	63 %	100 %	81 %

Le nombre minimum approximatif de traces nécessaires pour que l'attaque réussisse dans 90% des cas permet de quantifier l'impact de chacun des codes étudiés sur la sensibilité à l'attaque DPA. Les valeurs « n / d » indiquent l'indisponibilité de cette valeur car même avec les 256 traces le taux de réussite des attaques est inférieur à 90%. Ces valeurs permettent donc de créer un classement des codes les plus intéressants vis-à-vis des attaques en puissance. Une valeur élevée indique une sensibilité plus faible face à l'attaque DPA. Le classement qui ressort en étudiant ce critère est le même que celui obtenu précédemment en analysant la hauteur relative du second pic DPA.

Tableau 35 : Nombre de traces minimum nécessaires à l'attaque DPA sur S-Box

Code	Logique non protégée	Logique Protégée
Aucun Code	80	80
Parité Simple	90	90
Parité Double Croisée	150	100
Parité Double Complémentaire	110	100
Berger	n / d	n / d
Hamming1	210	200
Hamming2	n / d	250
Hamming3	210	220
Hsiao	n / d	250

Cette étude sur différentes versions protégées par des codes correcteurs et détecteurs d'une S-Box AES montre que, bien que les bits de redondance soient autant attaquables que les bits de données, leur ajout diminue la sensibilité globale aux attaques DPA. On pourra aussi noter les coûts beaucoup plus faibles lors de la protection d'une fonction plus complexe que lors de la protection du registre étudié précédemment. De plus, on remarque que le caractère différentiel de l'attaque DPA plus particulièrement analysé dans cette section modifie l'ordre obtenu lors de l'étude de la corrélation au niveau d'un registre.

4.2. Etude d'un co-processeur AES

Nous avons donc testé notre flot d'analyse DPA sur deux versions d'un co-processeur AES. Nous allons brièvement présenter l'implantation utilisée par ces deux circuits. Ensuite, nous évaluerons leur sensibilité aux différents types d'attaques.

4.2.1. Présentation du co-processeur

Le co-processeur AES utilisé dans cette étude est un circuit disponible au sein de l'équipe. [Mais 08] présente plus précisément les détails d'implantation de l'algorithme sur ce circuit.

Le circuit est décomposé en trois blocs principaux : le bloc de contrôle, le bloc de génération des clefs de ronde et le chemin de données. Le chemin de données contient 16 cellules contenant chacune un des octets de l'état, ainsi que 4 S-Box. La Figure 50 présente la structure du chemin de données. Dans ce circuit, les S-Box sont conçues sur une architecture pipeline à deux étages utilisant des opérations dans $GF((2^4)^2)$. La réduction du nombre de S-Box est une optimisation du coût d'implantation du circuit. L'entrée des S-Box est connectée aux cellules de la dernière ligne. Lors d'une ronde, les lignes se décalent successivement vers le bas pour entrer dans les S-Box. Après 5 cycles d'horloge la dernière ligne a été substituée ; au $6^{ème}$, l'état se retrouve à nouveau entièrement dans les cellules de données et a subi les transformations « ShiftRows » et « MixColumn ».

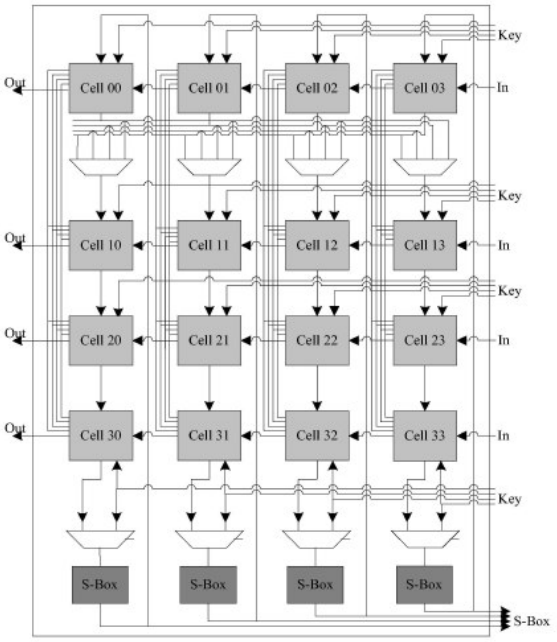

Figure 50: Architecture du chemin de données [Mang 03]

Deux versions de ce circuit ont été protégées par des mécanismes de détection d'erreurs. Ces versions incluent respectivement un code à parité simple et un code à parité double complémentaire. D'une manière générale, un bit (ou paire de bits) a été ajouté à chaque octet des données et de la clef entrante ainsi qu'à chaque octet de l'état. Les bits de redondance sont mis à jour opération après opération par une méthode de prédiction et sont constamment vérifiés. Les autres blocs ont eux aussi été protégés en utilisant ces codes détecteurs.

4.2.2. Sensibilité aux attaques

L'étude de l'impact de ces protections sur la sécurité de ce circuit vis-à-vis des attaques par fautes a été réalisée dans [Mais 07]. Elle ne sera pas détaillée ici. Nous allons en revanche nous concentrer sur la DPA.

Pour évaluer la sensibilité des deux versions de ce co-processeur face aux attaques par canaux cachés, nous avons apporté quelques modifications au flot précédent. Les premières étapes de simulation au niveau transistors ont été réalisées de la même manière. Cependant, l'étape d'analyse DPA a été adaptée à la structure du nouveau circuit ; il s'agit de la prise en compte de l'indice de l'octet de clef attaqué et de l'itération sur les différentes valeurs possibles de cet indice.

L'analyse a été séparée en deux parties. La première est basée sur l'attaque DPA d'un des octets de la clef en utilisant des simulations exhaustives pour la S-Box ciblée. La seconde est l'attaque de la clef complète grâce à des messages choisis aléatoirement.

Dans un premier temps, nous avons attaqué une des S-Box du circuit complet. Pour ce faire nous avons utilisé les traces obtenues lors du chiffrement des 256 messages contenant des 0 sur les octets non liés à la S-Box sélectionnée. En utilisant les traces ainsi obtenues, nous avons réalisé les attaques DPA avec les différents paramètres de partitionnement possibles. Contrairement aux résultats obtenus sur les S-Box seules dans la première partie de ce chapitre, le taux de réussite n'est plus de 100%. Le Tableau 36 présente les valeurs trouvées dans ce cas. Cependant, un vote parmi les octets de clefs trouvés permet toujours de faire ressortir la bonne clef. Ceci est valable que le vote soit fait sur la valeur complète de l'octet trouvé ou sur chacun de ses bits indépendamment. On remarque que, comme précédemment, la S-Box attaquée la plus sensible à l'attaque DPA est celle de la version non protégée contre les fautes.

Tableau 36 : Efficacité de l'attaque DPA sur les deux versions de l'AES

	% paramètres trouvant la clef	Hauteur relative moyenne du 2nd pic DPA
AES non protégé	89 %	63 %
AES protégé	64 %	75 %

La seconde partie de cette étude de l'AES est basée sur des traces de courant obtenues lors de simulations avec des messages à chiffrer choisis aléatoirement. Successivement, toutes les S-Box de la première ronde ont été attaquées avec les différents paramètres de partitionnement possibles. Un vote a ensuite été réalisé pour déterminer chaque octet de la clef. Deux méthodes de vote ont été utilisées : le vote à l'octet majoritaire et le vote bit par bit. Le nombre de bits justes ainsi retrouvés avec 768 traces est présenté dans le Tableau 37, sous forme d'une fraction de la taille de la clef. Dans ce cas, on remarque que l'ajout de la protection a aussi réduit le nombre de bits de clef trouvés.

Tableau 37 : Fraction des bits de la clef obtenus lors de l'attaque DPA

	Vote par Octet	Vote par Bit
AES non protégé	25 %	58 %
AES protégé	25 %	52 %

Dans [Rega 07] et [Rega 08], différentes versions d'un co-processeur AES sont attaquées. Certaines de ces versions incluent des protections contre les fautes : des codes détecteurs à parité simple ou à base de résidu modulo 3 ou 7. Les résultats obtenus lors d'une attaque de type CPA (attaque en puissance par corrélation) en présence de bruit blanc montrent plusieurs choses. Une

attaque classique, sans hypothèse sur la protection, permet de trouver la clef. Avec la connaissance du type de protection utilisé, les bits de redondance sont attaquables de la même manière et les pics CPA qu'ils génèrent ont une amplitude similaire. L'ajout du code détecteur augmente le nombre de bits de clef trouvé par trace, ou diminue le nombre de traces nécessaires à l'obtention de la clef, rendant ainsi le circuit plus sensible s'il est protégé.

Les conclusions obtenues ici sont partiellement opposées à celles obtenues par [Rega 07] et [Rega 08]. En effet, l'ajout de codes détecteurs n'empêche pas l'attaque de fonctionner. Les bits de redondance peuvent eux aussi être attaqués de la même manière. Cependant, à propos de l'impact de ces protections sur la sensibilité aux attaques en puissance, les conclusions s'opposent. Les deux différences majeures entre ces études peuvent être la source de cette divergence : la nature de l'attaque (DPA vs. CPA) et l'absence ou non de bruit lors des simulations. Ce point reste à confirmer lors d'études futures

4.3. Etude d'un co-processeur RSA

Nous avons aussi analysé l'impact de certaines protections sur la sensibilité aux attaques par fautes et en puissance d'un bloc de chiffrement RSA. Nous allons donc décrire le circuit étudié avant de voir comment il réagit face à ces deux types d'attaques.

4.3.1. Présentation du co-processeur

4.3.1.1. Algorithme RSA

L'algorithme RSA, du nom de ses auteurs Rivest, Shamir et Adleman, est un algorithme de chiffrement asymétrique basé sur l'utilisation de clefs publiques et privées constituées toutes deux d'un couple d'entiers. Cet algorithme exploite certaines propriétés de l'arithmétique modulaire.

Les clefs sont générées de la manière suivante. Pour commencer, il faut déterminer deux nombres premiers (de préférence de même grande taille) notés P et Q. Ces deux entiers restent secrets, connus au plus du propriétaire de la clef associée. On effectue leur produit afin d'obtenir l'entier N. On note φ, la fonction qui à un entier X associe $\varphi(X)$, le nombre d'entiers inférieurs à X et premiers avec X. On choisit ensuite un entier E premier avec $\varphi(N)$ et inférieur à N. Enfin, on détermine D l'inverse multiplicatif de E modulo $\varphi(N)$. On appelle le couple (E, N) la clé publique et le couple (D, N) la clé privée. Il va sans dire que la clé privée ne doit pas être connue par d'autres personnes que son propriétaire.

Le chiffrement est réalisé par l'élévation du message *M* à la puissance *e* modulo *n*, en utilisant la clef publique du destinataire. Le destinataire déchiffre en utilisant sa clef privée en calculant la même élévation à la puissance.

4.3.1.2. Implantation matérielle de l'algorithme RSA choisie

Un circuit permettant le chiffrement et déchiffrement RSA est un opérateur arithmétique permettant le calcul modulaire d'élévation à la puissance. L'implantation choisie pour réaliser l'élévation à la puissance est basée sur un algorithme utilisant alternativement une multiplication et une élévation au carré pour construire le résultat, toutes deux utilisant une multiplication modulaire de Montgomery. La Figure 51 présente le pseudo code décrivant ainsi l'algorithme utilisé.

```
0 -   Int MontExp(M,E,N,2^2nmodN)
1 -   P = MontMult(M,2^2nmodN,N)
2 -   R = MontMult(1,2^2nmodN,N)
3 -   For i = 0 to n-1
4 -         Temp = MontMult(P,R,N)
5 -         If E(i) Then R = Temp
6 -         P = MontMult(P,R,N)
7 -   End For
8 -   S = MontMult(R,1,N);
9 -   Return S;
```

Figure 51 : Algorithme d'élévation à la puissance

La multiplication de Montgomery a été implantée en utilisant une structure pipelinée basée sur le découpage des opérandes en sous-mots qui seront traités par les cellules élémentaires du pipeline. La Figure 52 présente cette structure.

Les données étant transmises en série mot à mot, les registres de stockage des opérandes Y et N sont des registres à décalage de *w* bits. L'opérande X est fournie bit par bit à chaque cellule donc stockable dans un registre à décalage de *pplp* bits, où *pplp* est le nombre de cellules élémentaires. Considérons *t* comme étant la taille des opérandes et *w* celle des sous mots. On définit ainsi $pple = \left\lceil \frac{t+1}{w} \right\rceil$; *pple* représentant le nombre de mots nécessaires au calcul. Le bit supplémentaire est indispensable, étant donné qu'une variable interne à l'algorithme sera comprise entre 0 et 2*M-1 (ou M est le modulo). On décompose les opérandes comme suit (Y et M en sous mot et X bit à bit):

$M=(M^{(pple-1)},\ldots,M^{(1)},M^{(0)})$

$Y=(Y^{(pple-1)},\ldots,Y^{(1)},Y^{(0)})$

$X=(x_{t-1},\ldots,x_1,x_0)$

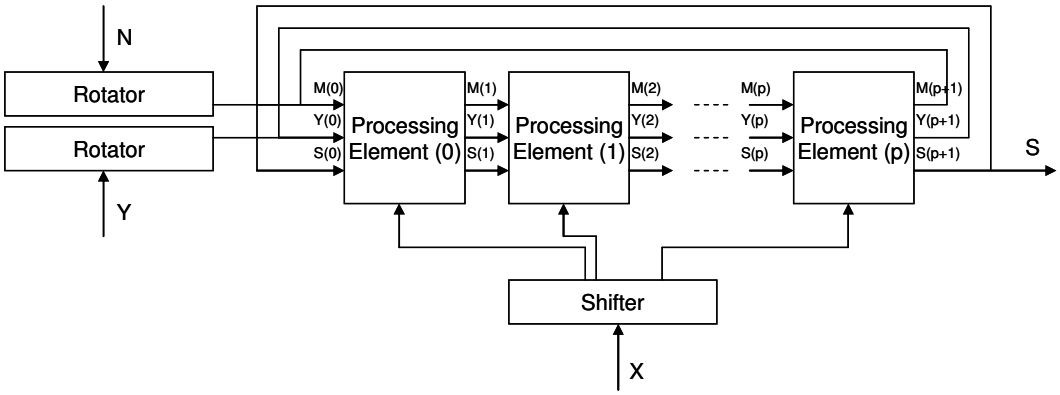

Figure 52 : Architecture pipeline de la multiplication de Montgomery [Koç 99]

Dans l'algorithme en pseudo code de la Figure 53, on notera (A,B) la concaténation de A et B, $A_{i..j}$ le sous mot de A allant du bit *i* au bit *j* et $A^{(k)}{}_j$ le bit *j* du sous mot *k* de A.

```
0 -   Int MWR2MM(X,Y,M)
1 -   S=0 ;
2 -   For i = 0 to t-1
3 -         (C,S(0))=xi*Y(0)+S(0);
4 -         if S(0)0 = 1 then
5 -               (C,S(0))=(C,S(0))+M(0);
6 -               for j = 1 to pple-1
7 -                     (C,S(j))=C+xi*Y(j)+M(j)+S(j);
8 -                     S(j-1)=(S(j)0,S(j-1)w-1..1)
9 -               S(e-1)=(C,S(e-1)w-1..1)
10-         else
11-               for j = 1 to pple-1
12-                     (C,S(j))=C+xi*Y(j)+S(j);
13-                     S(j-1)=(S(j)0,S(j-1)w-1..1)
14-               S(e-1)=(C,S(e-1)w-1..1)
15-   Return S;
```

Figure 53 : Algorithme de la multiplication de Montgomery

Cet algorithme calcule une somme partielle pour chaque bit de X en analysant les mots de Y et M. Les opérations arithmétiques sont effectuées sur *w* bits et ne dépendent pas de la taille totale des opérandes. La seule chose qui change est le nombre d'itérations dans chaque boucle.

Les dépendances entre les opérations effectuées dans les boucles indicées par *j* limitent leur parallélisation, à cause de la retenue C (cf. graphe de dépendance en Figure 54).

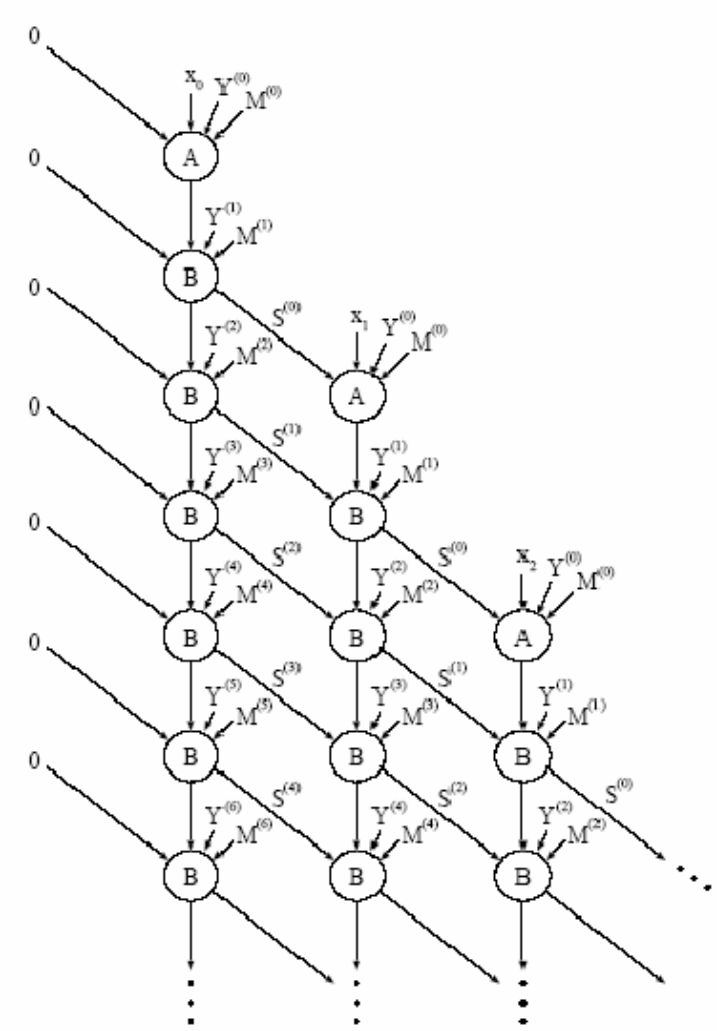

Figure 54 : Graphe de dépendance dans l'algorithme de Montgomery [Koç 99]

Chaque noeud sur ce graphe représente une action élémentaire de l'algorithme et est étiqueté en fonction de l'action effectuée. L'action A regroupe les lignes 3 à 5 : le test du bit de poids faible de *S* et l'addition de *S*, *xiY* et *M* si besoin. L'action B regroupe les lignes de 7 à 9. Ce graphe montre un fort degré de parallélisme. Dans l'architecture pipeline, chaque colonne est calculée par une cellule élémentaire et les données circulent d'une cellule à l'autre.

Etant donné que le *j*-ième mot de *Y* et *M* est utilisé pour déterminer le *j-1*-ième mot de *S*, il faut mettre un 0 en entrée lors du calcul du *e-1*-ième mot de *S*. Les tâches A et B sont effectuées par le même matériel capable de mémoriser la parité de $S^{(0)}$.

Pour obtenir une parallélisation maximale, il faut utiliser *p* cellules élémentaires avec *p* défini comme suit :

$$pplp = \left\lceil \frac{pple + 1}{2} \right\rceil$$

L'opération de chiffrement ou de déchiffrement RSA peut être optimisée en performance, si les deux facteurs premiers du modulo sont connus. Pour des raisons évidentes de secret de ces facteurs, cette optimisation n'est possible que dans le cas du déchiffrement ou de la signature. Elle est basée sur le théorème du reste chinois (CRT) permettant de reconstituer un nombre entier à partir des restes lors de sa division par rapport à différents entiers premiers. Dans le cas du RSA, nous allons calculer le résultat modulo *p* et *q* (facteurs premiers du modula de la clef). Ceci peut être réalisé par exponentiation sur la taille de ces facteurs. Ensuite, le théorème des restes chinois nous apprend comment calculer la valeur finale recherchée. La Figure 55 résume ce calcul.

$$Soit\ Dp = D \bmod P - 1$$
$$Soit\ Dq = D \bmod Q - 1$$
$$Soit\ Cp = (C \bmod P)^{Dp} \bmod P$$
$$Soit\ Cq = (C \bmod Q)^{Dq} \bmod Q$$
$$\begin{cases} M = Cp + ((Cq - Cp) * (P^{-1} \bmod Q)) * P & Si\ Cq < Cp \\ M = Cq + ((Cp - Cq) * (Q^{-1} \bmod P)) * Q & Si\ Cp < Cq \end{cases}$$

Figure 55 : Déchiffrement RSA par les restes chinois

Un sous-bloc de déchiffrement CRT a donc été ajouté de manière paramétrable au circuit. Ce paramétrage permet en réalité de créer trois versions non protégées du circuit de calcul RSA : la première n'inclut que le bloc d'exponentiation modulaire, la seconde version embarque l'optimisation pour le déchiffrement utilisant les restes chinois, la dernière version est plus limitée puisqu'elle n'embarque que le bloc CRT et ne permet donc pas de chiffrer.

4.3.1.3. Protections implantées

Ce circuit a été protégé contre les fautes par l'ajout paramétrable de deux protections de deux types différents. La première protection ajoutée est la détection d'erreurs à plusieurs niveaux

intermédiaires du calcul par un code à parité complémentaire. La seconde technique utilisée est une technique de redondance temporelle au niveau de l'exponentiation.

L'ajout du code à parité double complémentaire a été réalisé sur la description comportementale en VHDL du circuit. Pendant cette opération, chaque niveau hiérarchique a été découpé en deux sous-blocs : l'un séquentiel et l'autre combinatoire. Ainsi, chacun de ces sous-blocs a pu être protégé indépendamment et selon sa nature. Un soin a été apporté pour ne pas dupliquer les codeurs identiques entre ces deux sous-blocs : la sortie du bloc combinatoire est codée une seule fois. Les bits de redondance ainsi générés sont utilisés à la fois pour vérifier l'absence de fautes dans le bloc combinatoire et pour protéger les bits du registre. La Figure 56 montre comment un niveau hiérarchique dont les deux sous-blocs ont été protégés a été modifié. Sur un même niveau hiérarchique, tous les registres ont été regroupés pour permettre la répartition de leurs bits en sous-mot de taille fixe ; en effet, un paramètre structurel définit le nombre de bits de données à protéger par une paire de bits de redondance. Les blocs clefs du circuit ont été protégés avec cette technique : il s'agit des cellules élémentaires du pipeline de multiplication ainsi que l'additionneur/soustracteur utilisé dans le bloc CRT. Les autres blocs ont seulement été protégés au niveau de leurs registres.

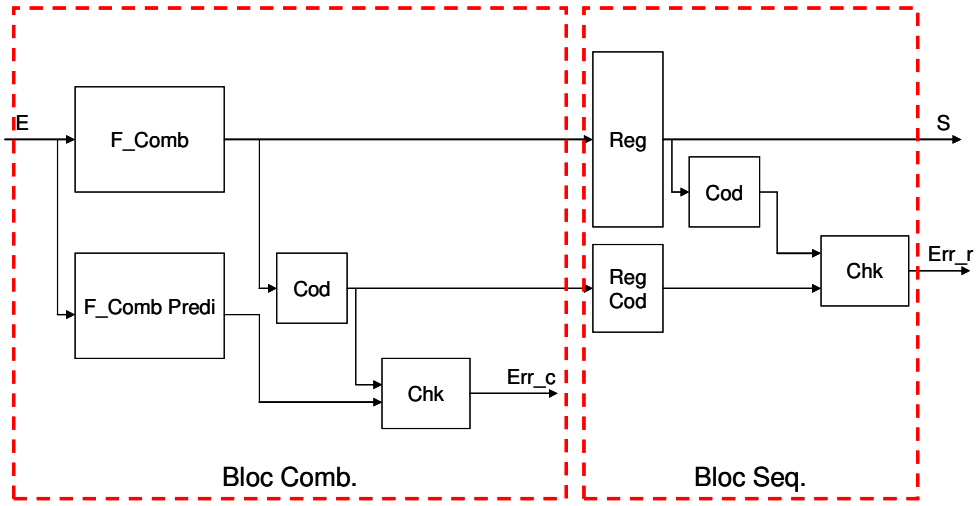

Figure 56 : Protection des deux sous-blocs d'un même niveau hiérarchique

A chaque niveau hiérarchique, les signaux d'erreurs retournés vers le niveau supérieur ont deux origines. La première partie de ces signaux provient des niveaux hiérarchiques inférieurs. Les autres sont la ou les deux paires de signaux en double rail générés au niveau considéré. Ce nombre de paires dépend de la protection ou non du sous-bloc combinatoire. Ces signaux d'erreur correspondent à l'unification des signaux d'erreurs générés pour chaque sous-mot. Cette méthode permet en fonction de la valeur du signal d'erreur global de savoir à quels niveaux hiérarchiques et dans quels sous-blocs

il y a eu des erreurs détectées. Ne pas unifier les signaux générés aux différents niveaux permettrait une localisation plus précise des erreurs mais coûterait un nombre de bits bien plus important. Le Tableau 38 résume les coûts de l'implantation de cette technique de protection. Les fréquences de fonctionnement plus faibles et insensibles à l'ajout de la protection viennent de la trop faible optimisation du chemin critique dans le bloc de division nécessaire au calcul de *C mod P* et *C mod Q* lors de l'utilisation des restes chinois ; de plus, ce bloc de division n'a pas été protégé.

Tableau 38 : Coût d'implantation du code à parité complémentaire sur le RSA

		Mode		
		MontExp	CRT + MontExp	CRT
Surface	Non Protégé	100 %	236 %	200 %
	Protégé	217 % (+117 %)	466 % (+97 %)	418 % (+109 %)
Fréquence	Non Protégé	100%	38 %	38 %
	Protégé	78 %	38 %	38 %

La seconde protection ajoutée est un mécanisme de redondance temporelle basé sur l'exécution redondante de l'opération de chiffrement ou déchiffrement la moins coûteuse en temps. En fonction de la configuration de la structure du circuit l'opération redondante peut changer. Le Tableau 39 présente la correspondance entre le couple (structure, opération initiale) et l'opération redondante. L'opération redondante est soit la même que l'opération initiale soit son inverse. Le choix de cette opération est fait de manière à minimiser le temps de calcul global tout en permettant le plus possible la détection de fautes persistant sur la durée des deux opérations. L'implantation de cette protection a elle aussi été réalisée par modification de la description comportementale en VHDL du circuit. Ces modifications sont beaucoup moins intrusives que l'implantation de codes détecteurs, vu qu'elles ne nécessitent que l'ajout d'une machine à état au niveau hiérarchique le plus haut pour contrôler l'opération redondante et la vérification de la conformité des résultats obtenus. L'automate ainsi ajouté diffère légèrement d'une configuration du circuit à une autre, notamment au niveau des commandes émises vers les blocs de calcul et les valeurs stockées. Le Tableau 40 présente les coûts de l'implantation de cette technique de protection sur le circuit.

Tableau 39 : Choix de l'opération redondante

Mode	Chiffrement	Déchiffrement
CRT	n / d	Déchiffrement
CRT + MontExp	Déchiffrement CRT	Déchiffrement CRT
MontExp	Déchiffrement	Chiffrement

Tableau 40 : Coût de l'implantation de la redondance temporelle sur le RSA

Mode		MontExp		CRT + MontExp		CRT
Surface	Non Protégé	100 %		236 %		200 %
	Protégé	115 % (+ 15 %)		255 % (+ 7 %)		218 % (+ 9 %)
Opération		Chiffr.	Déchiffr.	Chiffr.	Déchiffr.	Déchiffr.
Coût sur le temps d'exécution		+ 99 %	+ 94 %	+ 70 %	+ 96 %	+ 96 %

4.3.2. Sensibilité aux attaques par fautes

Nous allons étudier l'impact de chacune des protections implantées sur la sensibilité aux attaques par faute de manière indépendante, le paramétrage de ces protections n'ayant pas été prévu pour les superposer.

Nous avions vu précédemment que le code à parité complémentaire appliqué à un registre induisait une probabilité de détection de 75%. En effet, cette probabilité est définie par le quotient du nombre de mots n'appartenant pas au code sur le nombre total de mots possibles. Nous obtenions ainsi la probabilité indiquée ($(2^{n+2} – 2^n)/ 2^{n+2} = 75\%$). Par extension, la probabilité de détection d'un erreur de multiplicité quelconque sur l'ensemble des bits protégés par sous-mots de taille k est définie par ($(2^{n+2n/k} – 2^n) / (2^{n+2n/k})$) qui vaut $1 - 1/2^{n+2n/k}$, qui donne bien 75% dans le cas où les données ne sont pas divisées (n=k). Le paramètre k de taille des sous-mots permet donc d'adapter le coût en fonction du taux de détection recherché.

Pour évaluer l'impact de la redondance temporelle sur la sensibilité du circuit protégé face aux attaques par fautes, mettons nous du point de vue d'un attaquant. Son but est de créer une ou plusieurs fautes permettant d'obtenir un résultat faux. Dans le meilleur des cas, il lui suffit d'induire deux fois les mêmes erreurs dans le circuit sur les deux copies de la même opération. Dans un cas plus général, il doit être capable de générer dans les deux opérations des fautes lui permettant d'obtenir d'une part un résultat faux mais surtout d'avoir d'autre part le résultat en fin de seconde opération attendu pour la comparaison. Etant donné le faible déterminisme de l'induction de fautes par laser, vouloir reproduire exactement la même erreur deux fois de suite semble très difficile. Dans le cas où l'opération redondante est l'inverse, l'attaque est encore plus difficile, car elle nécessite de déterminer avec précision quelles fautes sont nécessaires et d'être sûr de garantir leur induction correcte.

Il ne reste qu'à essayer d'attaquer les mécanismes ajoutés pour détecter les fautes induites. Pour ce faire, le pirate doit réussir à modifier la sortie du comparateur. En effet, si le pirate arrive à garantir que le comparateur indique toujours une égalité, il pourra alors obtenir des résultats fautés.

4.3.3. Sensibilité aux attaques en puissance

Le flot d'attaque DPA par simulations au niveau transistors a été utilisé pour attaquer deux versions du co-processeur RSA. Les deux versions attaquées sont la version non protégée sans CRT et la version avec le code à parité complémentaire. Comme pour l'AES, l'étape finale de partitionnement et calcul du biais à dû être adaptée à cet algorithme.

La fonction de sélection choisie pour cet algorithme est la valeur de la variable R après un certain nombre d'itérations dans la boucle principale de l'algorithme. Plusieurs paramètres interviennent dans la définition de cette fonction de sélection : le nombre de bits de clef déjà connus, leur valeur et le nombre de bits de clef à déterminer. Le nombre de bits de clef connus débute à 1 et augmente de la valeur de l'autre paramètre dès que des bits sont trouvés, jusqu'à ce que la clef complète soit trouvée. Le paramètre de sélection du mode de partitionnement est toujours présent.

Lors de l'application de cette attaque DPA au circuit, nous avons utilisé de nombreuses combinaisons des différents paramètres d'attaque sans succès. Nous n'avons pas pu exhiber les clefs de chiffrement utilisées lors de l'obtention des traces. Ceci tend à montrer une certaine robustesse de l'implantation réalisée. Lors de l'attaque de la version protégée par parité complémentaire, les mêmes techniques ont été utilisées nous permettant d'obtenir le même résultat : l'attaque a échoué. Ceci montre que le code à parité complémentaire n'a pas introduit de faille de sécurité dans l'implantation du RSA.

4.3. Conclusion

Dans ce chapitre, nous avons appliqué notre flot d'étude de sensibilité aux attaques en puissance par simulations au niveau transistors sur des primitives cryptographiques. Ceci nous a permis d'une part de montrer son efficacité sur des circuits plus complexes et, d'autre part, d'étudier l'impact des différents codes choisis sur la sensibilité du circuit aux attaques.

Nous avons tout d'abord vu que l'ajout de ces codes détecteurs ou correcteurs ne bloque pas les attaques en puissance. De plus, avec la bonne hypothèse sur le code choisi, les bits redondant sont attaquables dans la même mesure que les bits de données. Ceci suggère donc que lors de la protection globale du circuit, les bits de redondance doivent être protégés contre les attaques par canaux cachés.

De plus, les différents codes ont une influence variable sur la sensibilité aux attaques en puissance. De manière générale, l'ajout de ces codes n'augmente pas cette sensibilité. En effet, le nombre de traces nécessaires pour obtenir les clefs est plus élevé en présence de code. On remarque

aussi que les codes correcteurs ont globalement de meilleures caractéristiques que les codes détecteurs. Seul le code de Berger fait exception.

Ces résultats sont en grande partie conformes avec les conclusions faites lors de l'étude du registre durci. Seule la position de certains codes dans le classement a changé, notamment l'inversion du code de Berger et du code à parité double complémentaire. Cette différence est due d'une part à la réduction de la part des codeurs et vérificateurs dans la consommation totale et d'autre part au caractère différentiel de l'attaque DPA.

Conclusion et Perspectives

La conception d'applications sécurisées nécessite plus que la simple implantation d'algorithmes de chiffrement et/ou d'authentification dans les dispositifs utilisés. L'évolution des technologies de semi-conducteurs, notamment la réduction de leurs dimensions, favorise les perturbations externes du circuit. Ces perturbations d'origines multiples, naturelles ou intentionnelles, peuvent avoir des conséquences variables sur les circuits conçus dans ce cadre. La corruption des données traitées est une des plus dangereuses conséquences pour des applications sécurisées. L'étude des techniques de piratage d'implantations d'algorithmes de chiffrement nous permet de comprendre les enjeux nécessitant l'établissement de protections contre ces phénomènes. Ainsi, ces circuits embarquent des mécanismes de détection et/ou de correction de fautes.

Cependant cette étude montre aussi l'existence d'autres techniques d'attaque beaucoup plus difficilement détectables et dont la protection nécessite des mécanismes totalement différents. Pour analyser l'influence de certaines protections contre les fautes sur la sensibilité d'un circuit aux attaques par observation, nous avons mis au point une méthode d'évaluation de cette sensibilité utilisable lors de la conception d'un circuit sécurisé. Le choix de réaliser cette analyse pendant la conception du circuit permet la réalisation d'ajustements des protections mises en place contre les fautes. Pour avoir une précision suffisante, cette étude est basée sur des simulations d'une description intermédiaire du circuit, entre la synthèse logique et le placement et routage.

La méthodologie mise en place a été conçue de manière à être facilement utilisable avec une technologie cible et un circuit évalué quelconques. L'utilisation de technologies différentes ne nécessite que d'y avoir accès, tandis que le changement de circuit peut demander une certaine adaptation de l'étape d'analyse des traces de simulation. Cette adaptation n'est que la transposition dans le logiciel de la fonction calculée par le circuit évalué.

Nous avons appliqué cette méthodologie sur différents cas d'études : différentes versions d'un registre sécurisé, un boite de substitution AES, un co-processeur AES et un co-processeur RSA. Dans tous ces cas, nous avons montré que l'ajout de codes correcteurs ou détecteurs sur ces circuits, en plus de les avoir doté de capacités de détection ou de correction, n'a pas accru la sensibilité du circuit aux attaques en puissance. De plus, dans les trois premiers cas, l'ajout de ces protections a permis de réduire partiellement cette sensibilité aux attaques en puissance.

De plus, lors de campagnes de tirs laser sur deux circuits, nous avons pu avoir une vision plus précise des effets de ces tirs. Nous avons pu notamment observer le caractère fortement non déterministe des erreurs ainsi générées, ainsi que leur forte multiplicité. De plus, nous avons vu que les technologies de type FPGA sont susceptibles d'ajouter le risque supplémentaire de sa reconfiguration partielle.

Ce travail de thèse offre plusieurs perspectives de recherche.

Nous avons ici analysé l'impact de protections contre les fautes sur la sensibilité d'un circuit face aux attaques par canaux cachés. Il serait aussi intéressant d'étudier l'influence que des protections contre les attaques par observation peuvent avoir sur la sensibilité aux attaques par fautes.

Nous avons comparé l'impact de quelques codes correcteurs et détecteurs sur la sensibilité aux attaques en puissance d'un circuit. Il pourrait être intéressant de mettre en place un modèle mathématique permettant à partir de la matrice de vérification d'un code linéaire de déterminer son influence sur la consommation du circuit, puis de l'étendre à des codes non linéaire ou plus globalement à toute protection contre les fautes. Il serait donc ainsi plus facile de transposer cette étude à différentes protections.

Nous avons vu d'une part que les protections contre les deux principaux types d'attaques sont de nature très différente. D'autre part, les techniques de protection contre les fautes étudiées ici se sont aussi montrées bénéfiques face aux canaux cachés. Serait-il possible de concevoir un type de protection unique, efficace contre toutes les attaques ?

Enfin, l'étude des effets des tirs laser sur FPGA réalisée lors de cette thèse montre les risques de ce type de technologie pour des applications sécurisées. Une compréhension plus approfondie des phénomènes mis en jeu lors de l'attaque de ce type de composants est donc nécessaire pour permettre de les caractériser plus précisément. Ceci permettrait, entre autres, de développer des contre-mesures particulières pour l'utilisation de ce type de circuit.

Bibliographie

[Akka 01] M.L. Akkar, C. Giraud, “An implementation of DES and AES, Secure Against Some Attacks”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp. 309-318, 2001

[AMBA] http://www.arm.com/products/solutions/AMBA_Spec.html

[Angh 00] L. Anghel, M. Nicolaidis, “Cost Reduction and Evaluation of a Temporary Faults Detecting Technique”, Proc. of Design, Automation and Test in Europe (DATE ’00), pp. 591-597, 2000

[Badd 08] K. Baddam, M. Zwolinski, « Divided Backend Duplication Methodology for Balanced Dual Rail Routing », Workshop on Cryptographic Hardware and Embedded Systems (CHES 2008), LNCS 5154, pp. 396-410, 2008

[Berg 61] J. M. Berger, “A note on an error detection code for asymmetric channels”, Information and Control, vol 4, pp. 68–73, 1961

[Bert 02] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, “A parity code based fault detection for an implementation of the Advanced Encryption Standard”, IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002), IEEE Computer Society Press, pp. 51-59, 2002

[Bert 03] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, V. Piuri, ”Concurrent fault detection in a hardware implementation of the RC5 encryption algorithm”, IEEE International Conference on Application-Specific Systems, Architectures, and Processors, (ASAP'03), 2003, pp. 423-432

[Biha 93] E. Biham, A. Shamir, “Differencial cryptanalysis of the Data Encryption Standard”, Springer-Verlag, 1993

[Biha 97] E. Biham and A. Shamir, “Differential Fault Analysis of Secret Key Cryptosystems”, Proceedings of Advances in Cryptology - CRYPTO ’97, LNCS 1294, pp. 513-525, 1997

[Biry 07] A. Biryukov, D. Khovratovich, « Two New Techniques of Side-Channel Cryptanalysis », Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS 4727, pp. 195-208, 2007

[Bone 97] D. Boneh, R. DeMillo, R. Lipton, "On importance of checking cryptographic protocols for faults", Advances in cryptology – Eurocryp'97, LNCS 1233, Springer-Verlag, pp. 37-51, 1997

[Brev 04] L. Breveglieri, I. Koren, P. Maistri, "Detecting faults in integer and finite field arithmetic operations for cryptography", Workshop on fault diagnosis and tolerance in cryptography (FDTC 2004), Supplemental volume of the 2004 International Conference on Dependable Systems and Networks (DSN), pp. 361-367, 2004

[Cath 03] J. Cathalo, F. Koeune, J.-J. Quisquater, "A New Type of Timing Attack: Application to GPS", Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS 2779, Springer-Verlag, pp. 291-303, 2003

[Coro 07] J.-S. Coron, E. Prouff, M. Rivain, « Side Channel Cryptanalysis of a Higher Order Masking Scheme », Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), LNCS 4727, pp. 28-44, 2007

[Dhem 98] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestre, J.-J. Quisquater, J.-L. Willems, "A practical implementation of the timing attack", Proc. CARDIS 1998, LNCS 1820, Springer-Verlag, pp. 167-182, 1998

[Dong 82] H. Dong, "Modified Berger codes for the detection of unidirectionnal errors", 12th Int Symposium on Fault-Tolerant Computing, pp 317-320, 1982

[Faur 02] F. Faure, P. Peronnard, R. Velazco, R. Ecoffet, "THESIC+, a flexible system for SEE testing", Proc. of RADECS 2002 Workshop, pp. 231-234, 2002.

[Fouq 03a] P.-A. Fouque, G. Martinet, G. Poupard, "Attacking Unbalanced RSA-CRT Using SPA", Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS 2779, Springer-Verlag, pp. 2-16, 2003

[Gand 01] K. Gandolfi, C. Mourtel, F. Olivier, "Electromagnetic Analysis: Concrete Results", Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp. 251-272, 2001

[Gira 05] C. Giraud., "Fault-resistant RSA Implementation", Fault Diagnosis and Tolerance in Cryptography (FDTC 2005), pp. 142-151, 2005

[Guer 04] S. Gueron, ”Fault detection mechanisms for smartcards performing modular exponentiation”, Workshop on fault diagnosis and tolerance in cryptography, Florence, Italy, June 30, 2004, Supplemental volume of the 2004 International Conference on Dependable Systems and Networks (DSN), 2004, pp. 368-372

[Guil 05] S. Guilley, P. Hoogvorst, Y. Mathieu and R. Pacalet. The "backend duplication" method. Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), LNCS 2005

[Habi 92] D.H. Habing, “The use of Lasers to simulate radiation-induced transients in semiconductors devices and circuits”, IEEE Transaction On Nuclear Science, vol. 39, pp. 1647-1653, 1992

[Hsia 70] M. Y. Hsiao, “A Class of Optimal Minimum Odd-weight-column SEC-DED Codes”, IBM Journal of Research and Developpment, volume 14, number 4, Coding Theory and Application, pp. 395,1970

[Josh 04] N. Joshi, K. Wu, R. Karri, “Concurrent Error Detection Schemes for Involution Ciphers”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), LNCS 3156, Springer-Verlag, pp. 400-412, 2004

[Joye 01a] M. Joye, C. Tymen, “Protections against Differential Analysis for Elliptic Curve Cryptography: An Algebraic Approach”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp. 377-390, 2001

[Joye 05] M. Joye and M. Ciet., “Practical Fault Countermeasures for Chinese Remainder Based RSA”, Fault Diagnosis and Tolerance in Cryptography (FDTC 2005), pp. 12-131, 2005

[Karp 04] M. Karpovsky, K. J. Kulikowski, A. Taubin, “Robust protection against fault-injection attacks of smart cards implementing the Advanced Encryption Standard”, International Conference on Dependable Systems and Networks (DSN 2004), pp. 93-101, 2004

[Karr 03a] R. Karri, G. Kuznetsov, M. Goessel, “Parity-Based Concurrent Error Detection of Substitution-Permutation Network Block Ciphers”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS 2779, Springer-Verlag, pp. 113-124, 2003

[Karr 03b] R. Karri, G. Kuznetsov, M. Goessel, ”Parity-based concurrent error detection in symmetric block ciphers”, International Test Conference (ITC 2003), pp. 919-926, 2003

[Koç 99] Ç. K. Koç, C. Paar, “A Scalable Architecture for Montgomery Multiplication”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 1999), LNCS 1717, Springer-Verlag, pp. 94-108, 1999

[Koch 96] P. Kocher, “Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems”, Advances in Cryptology - CRYPTO '96, LNCS 1109, Springer-Verlag, pp. 104-113, 1996

[Koch 99] P. Kocher, J. Jaffe, B. Jub, “Differential power analysis”, Proc. of Advances in Cryptology (CRYPTO '99), LNCS 1666, Springer-Verlag, pp. 388-397, 1999

[Komm 99] O. Kommerling and M. Kuhn, “Design Principles for Tamper Resistant Smartcard Processors”, Proceedings of the USENIX Workshop on Smartcard Technology, pp. 9-20, 1999

[Kusn 04] G. Kusnezov, R. Karri, M. Goessel, “Error detection by parity modification for the 128-bit serpent encryption algorithm”, Workshop on fault diagnosis and tolerance in cryptography (FDTC 2004), Supplemental volume of the 2004 International Conference on Dependable Systems and Networks (DSN 2004), pp. 375-380, 2004

[LE 06] T.-H. Le, J. Cledière, C. Canovas, B. Robisson, C. Serviere, J.-L. Lacoume, « A proposition for correlation power analysis enhancement », Workshop on Cryptographic Hardware and Embedded Systems (CHES 2006), LNCS 4249, Springer-Verlag, pp. 174-186, 2006

[Lima 00] F. Lima, E. Costa, L. Carro, M. Lubaszewski, R. Reis, S. Rezgui, R. Velazco, “Designingand Testing a Radiation hardened 8051-like Micro-Controller”, 3rd Military and Aerospace Applications of Programmable Devices and TechnologiesInternational Conference, 2000

[Maam 98] A. Maamar, G. Russell, « A 32-Bit Risc Processor with Concurrent Error Detection », 24 th. EUROMICRO Conference Volume 1 (EUROMICRO'98), pp. 10461-10467, 1998

[Mais 05] P. Maistri, L. Breveglieri and I. Koren, “Incorporating Error Detection in a RSA Architecture”, Fault Diagnosis and Tolerance in Cryptography (FDTC 2005), pp. 132-141, 2005

[Mais 07] P. Maistri, P. Vanhauwaert, R. Leveugle; “Evaluation of Register-Level Protection Techniques for the Advanced Encryption Standard by Multi-Level Fault Injections”, Defect an Fault Tolerance in VLSI Systems (DFT 2005); pp. 499-507, 2005

[Mais 08] Paolo Maistri, Régis Leveugle, “Double-Data-Rate Computation as a Countermeasure against Fault Analysis”, November 2008, Computers IEEE Transactions on, vol57, pp 1528-1539

[Mang 03] S. Mangard, M. Aigner, S. Domonukus, « A hightly regular and scalable AES hardware architecture », IEEE Transaction on Computers, vol.52, issue 4, pp. 483-491, 2003

[Mess 00] Thomas S. Messerges, “Using Second-Order Power Analysis to Attack DPA Resistant Software”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), LNCS 1965, Springer-Verlag, pp. 238-251, 2000

[Mull 01] D. May, H. L. Muller and N. P. Smart, “Random Register Renaming to Foil DPA”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp. 28-38, 2001

[Oswa 01] E. Oswald, M. Aigner, “Randomized Addition-Subtraction Chains as a Countermeasure against Power Attacks”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2001), LNCS 2162, Springer-Verlag, pp. 39-50, 2001

[Otto 05] M. Otto, J. Bloemer and J.P. Seifert, “Sign Change Fault Attacks On Elliptic Curve Cryptosytems”, Fault Diagnosis and Tolerance in Cryptography (FDTC 2005), pp. 25-40, 2005

[Pfla 02] M. Pflanz, K. Walther, C. Galke and H. T. Vierhaus, “On-Line Detection and Correction in Storage Elements with Cross-Parity Check”, Proceedings of the 8th IEEE International On-Line Testing Workshop (IOLTW’02), pp. 69-73, 2002

[Port 06] M. Portolan, "Conception d'un système embarqué sûr et sécurisé", Doctorat de l'INPG, spécialité "Micro et Nano Electronique", 6 Décembre 2006

[Poug 06] V. Pouget, D. Wan, P. Jaulent, A. Douin, D. Lewis, P. Fouillat, “Recent developments for SEE testing at the ATLAS laser facility”, Proc. of 15th Single-Event Effects Symposium, 2006.

[Poug 06b] V. Pouget, P. Fouillat, D. Lewis, “Using the SEEM software for SET testing and analysis”, *Radiation effects in embedded systems*, to be published by Springer, 2006.

[Poug 07] V. Pouget ; A. Douin, D. Lewis, P. Fouillat, G. Foucard, P. Peronnard, V. Maingot, J.B. Ferron, L. Angel, R. Leveugle, R. Velazco, "Tools and methodology development for pulsed laser fault injection in SRAM-based FPGAs"8th Latin-American Test Workshop (LATW), March 12-14, 2007

[Quis 00] J.J. Quisquater, D. Samyde, “A nex tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods”, presented during a rump session in EUROCRYP’00, 2000

[Quis 02] J.J.Quisquater, F.Koeune, “Side Channel Attacks”, CRYPTREC Report 1047, 2002. (available at http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1047_Side_Channel_report.pdf)

[Rake 01] P. Rakers, T. Collins, “Secure Contactless SmartCard ASIC with DPA Protection”, IEEE Journal of Solid-State Circuit, vol. 36, pp. 559-565, 2001

[Rega 07] F. Regazzoni, T. Eisenbarth, J. Grosschlädl, L. Breveglieri, PP. Ienne, I. Koren, C. Paar, “Power Attacks Resistance of Cryptographic S-boxes with added Error Detection Circuits”, Defect and Fault Tolerance in VLSI Systems (DFT 2007), pp 508-516, 2007

[Rega 08] F. Regazzoni, T. Eisenbarth, L. Breveglieri, PP. Ienne, I. Koren, “Can knowledge regardeing the presence of countermeasures against fault attacks simplify power attacks on cryptographic devices?”, Defect and Fault Tolerance in VLSI Systems (DFT 2008), pp 202-210, 2008

[Schi 00] Werner Schindler, “A Timing Attack against RSA with the Chinese Remainder Theorem”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), LNCS 1965, pp. 109-124, 2000

[Schr 06] Kai Schramm, Christof Paar: Higher Order Masking of the AES. CT-RSA 2006: pp.208-225, 2006

[Skor 02] S.P. Skorobogatov, R.J. Anderson, "Optical Fault Induction Attacks", Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), LNCS 2523, Springer-Verlag, pp. 31-48, 2002

[Sonz 05] Sonza Reorda M, Sterpone L, Violente M. Efficient estimation of SEU effects in SRAM-based FPGAs. In: Proceedings of international online testing symposium (IOLTS); 2005. p. 54–9

[Tiri 02] K. Tiri, M. Akmal, I. Verbauwhede, "A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards", European Solid-State Circuits Conference (ESSCIRC 2002), pp. 403- 406, 2002

[Tiri 04] K. Tiri, I. Verbauwhede, ”Place and route for secure standard cell design”, CARDIS'04: Sixth Smart Card Research and Advanced Application IFIP Conferences, 18th IFIP World Computing Congress, pp. 143-158, 2004

[Tiri 04b] K. Tiri and I. Verbauwhede. “A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”, Proc. of DATE’04, pp. 246–251, 2004

[Tric 02] E. Trichina, D. de Seta and L. Germani, “Simplified Adaptive Multiplicative Masking for AES”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), LNCS 2523, Springer-Verlag, pp. 187-197, 2002

[Walt 00] Colin D. Walter, “Data Integrity in Hardware for Modular Arithmetic”, Workshop on Cryptographic Hardware and Embedded Systems (CHES 2000), LNCS 1965, pp. 204-215, 2000

[Yen 00] S.-M. Yen, M. Joye, “Checking before output may not be enough against fault-based cryptanalysis”, IEEE Trans. on Computers 49, pp. 967-970, 2000

[Xili 07] Xilinx, “Virtex-II Complete Data Sheet”, available on the xilinx website at the adress http://www.xilinx.com/support/documentation/virtex-ii.htm

Bibliographie de l'auteur

Revues internationales

[RI.1] V. Maingot, J.B. Ferron, R. Leveugle, V. Pouget, A. Douin, "Configuration Errors Analysis in SRAM-Based FPGAs: Software Tool and Practical Results", Microelectronics reliability, Elsevier (ISSN: 0026-2714), vol. 47, num. 9-11, pp. 1836-1840, 2007, présenté à la conférence ESREF 2007

Conférences Internationales avec actes

[CI.1] R. Leveugle, V. Maingot, "Error On the Use of Information Redundancy When Designing Secure Chips", Proceedings of Workshop on Design and Diagnostics of Electronic Circuits and Systems 2006 (DDECS'06), pp. 141-142, 2006

[CI.2] V. Maingot, R. Leveugle, "Error Detection Code Efficiency for Secure Chips", Proceedings of the IEEE International Conference on Electronics, Circuits, and Systems 2006 (ICECS 2006), pp. 561-564, 2006

[CI.3] V. Pouget ; A. Douin, D. Lewis, P. Fouillat, G. Foucard, P. Peronnard, V. Maingot, J.B. Ferron, L. Angel, R. Leveugle, R. Velazco, "Tools and Methodology Development for Pulsed Laser Fault Injection in SRAM-Based FPGAs", Latin-America Test Workshop (LATW), 2007

[CI.4] V. Maingot, R. Leveugle, "On the Use of Error Correcting Codes in Secured Circuits", Proceedings of the Latin-America Test Workshop 2007 (LATW 2007), pp. ,2007

[CI.5] R. Leveugle, A. Ammari, V. Maingot, E. Teyssou, P. Moitrel, C. Mourtel, N. Feyt, J.B. Rigaud, A. Tria, "Experimental Evaluation of Protections Against Laser-induced Faults and Consequences on Fault Modeling", Proceedings of the conference on Design, Automation and Test in Europe (DATE 2007), pp. 1587-1592, 2007

[CI.6] V. Maingot, R. Leveugle, “On the Use of Error Correcting and Detecting Codes in Secured Circuits”, Proceedings of the PHD Research In Microelectronics and Electronics Conference (PRIME 2007), pp. 245-248, 2007 (Best Paper Award : Gold Leaf)

[CI.7] V. Maingot, R. Leveugle, “Analysis of Laser-Based Attack Effects on a Synchronous Circuit”, Proceedings of the International Design and Test Workshop 2007 (IDT’07), pp. 99-104, 2007

Conférences Nationales

[CN.1] V. Maingot, R. Leveugle, “Error Redondance d’Information : une Sécurité Suffisante?”, JNRDM 2006

[CN.2] V. Maingot, “De l’utilisation de codes détecteurs dans les circuits sécurisés”, Colloque GRD SoC-SiP, 2007

Colloques sans actes

[C.1] V. Maingot, J.B. Ferron, G. Canivet, R. Leveugle, “Fault Attacks on SRAM-based FPGAs: Analysis of Laser-induced Faults in a Virtex II”, USE-IT Workshop, 2007

Annexe 1 : Exemple d'attaque en puissance différentielle sur une S-Box

Lors de l'attaque en puissance différentielle réalisée sur la S-Box AES, nous avons procédé comme détaillé dans cette annexe.

Tout d'abord, nous avons générée en suivant le flot de conception traditionnellement utilisé dans l'équipe pour générer en générer la description au niveau transistors. Cette étape a été réalisant en utilisant successivement les outils Synopsys Design Vision pour réaliser la synthèse logique et les outils de la suite Cadence pour générer la liste des transistors.

Cette description est ensuite utilisée par Synopsys Nanosim en conjonction avec un fichier de données de configuration pour le simulateur ainsi que le fichier de stimuli. Un script shell Unix permet de générer automatiquement les traces de courant consommé pas le circuit pour l'ensemble des entrées possibles. La Figure 57 présente le tracé du courant consommé de la S-Box alimentée avec une même clef et des messages différents.

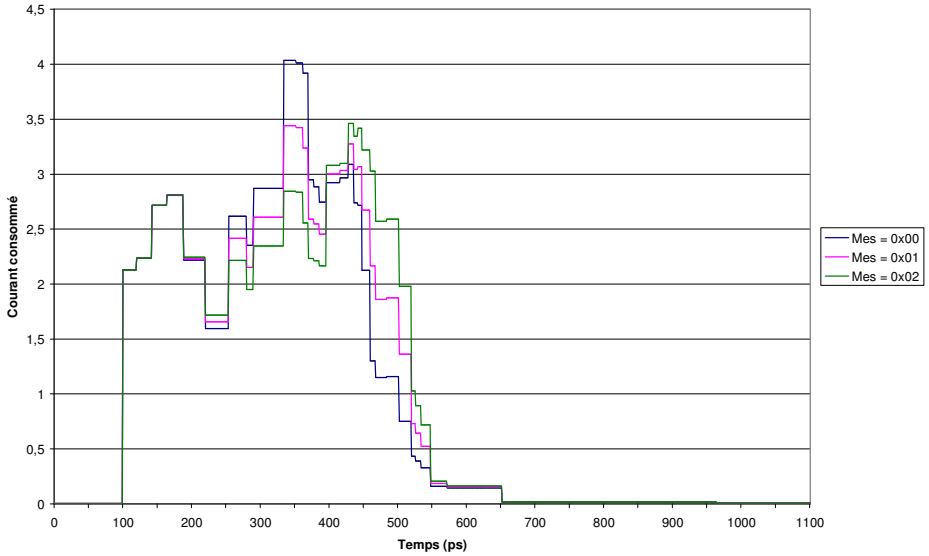

Figure 57 : Quelques profils de consommation de la S-Box

On remarque aisément sur ce graphe les différences de consommation pour les différents messages d'entrées. Ces différences dans les profils de consommation sont exploitées par l'attaque en puissance différentielle. Dans la suite de cette annexe, la fonction de sélection pour l'attaque en puissance différentielle est le premier bit de la sortie de la S-Box ; ainsi, la fonction de répartition est la valeur de ce bit.

L'étape suivante consiste à répéter pour l'ensemble des clefs possibles un certain nombre d'opérations. Tout d'abord, on détermine pour chaque message d'entrée quelle est la valeur du premier bit de sortie de la S-Box. S'il vaut 1, on ajoute la trace de courant pour ce message à la trace somme correspondant aux messages ayant généré un 1. S'il vaut 0, on ajoute la trace à la trace somme correspondant aux messages ayant généré un 0. Une fois tous les profils de courant ainsi répartis dans les deux groupes, il ne reste qu'a divisé chaque point de ces sommes par le nombre de traces pour en obtenir la moyenne. C'est deux courbes moyennes sont ensuite soustraites l'une à l'autre pour obtenir la courbe de biais correspondant à l'hypothèse de clef en cours. La présente l'ensemble des 256 courbes de biais obtenues lors de l'attaque de la S-Box.

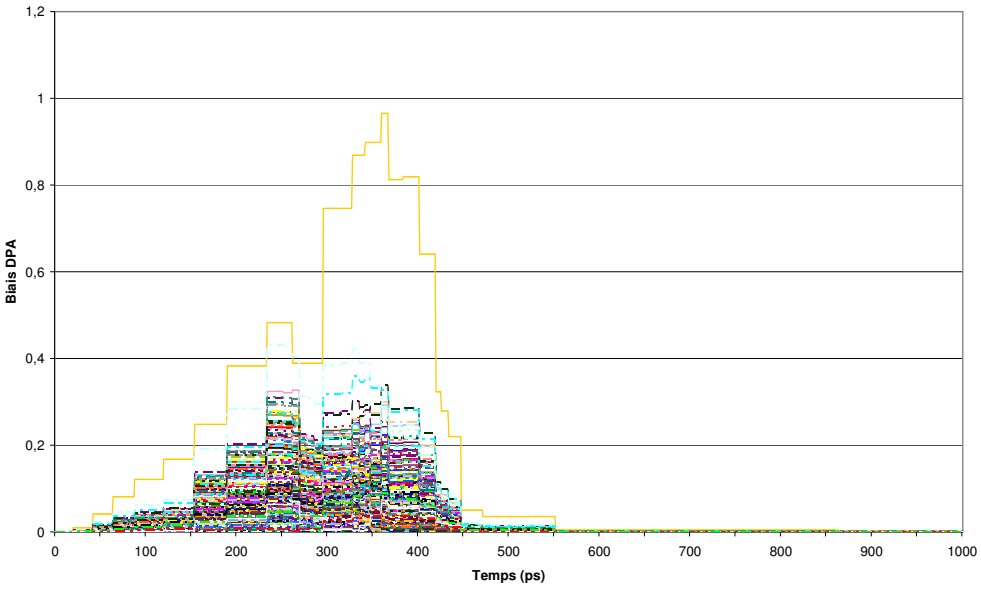

Figure 58 : Courbes de biais

La dernière étape consiste à parcourir l'ensemble de ces courbes de biais pour déterminer celle qui présente le plus haut maximum. Si l'attaque réussit, la clef réellement utilisée est celle correspondant à l'hypothèse dont la courbe présente ce maximum. La Figure 59 représente la courbe des maxima des courbes de bais. On note sur ce graphe un pic pour K=19, qui était bien la clef utilisée lors des simulations utilisées pour cet exemple.

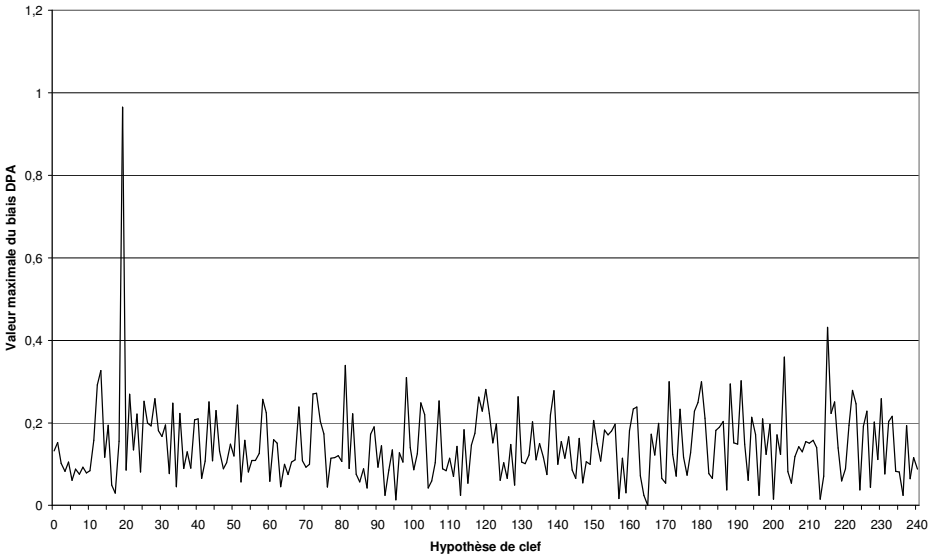

Figure 59 : Courbe des maxima de biais

RESUME

L'évolution des besoins en sécurité des applications grand public a entraîné la multiplication du nombre de systèmes sur puces doués de capacités de chiffrement. En parallèle, l'évolution des techniques de cryptanalyse permet d'attaquer les implantations des méthodes de chiffrement utilisées dans ces applications.

Cette thèse porte sur le développement d'une méthodologie permettant l'évaluation de la robustesse apportée par des protections intégrées dans le circuit. Cette évaluation est basée d'une part sur l'utilisation de plates-formes laser pour étudier les types de fautes induits dans un prototype de circuit sécurisé ; et d'autre part, sur l'utilisation d'une méthode basée sur des simulations pendant la phase de conception pour comparer l'influence sur les canaux cachés de protections contre les fautes.

Cette méthodologie a été utilisée dans un premier temps sur le cas simple d'un registre protégé par redondance d'information, puis sur des primitives cryptographiques telles qu'une S-Box AES et des co-processeurs AES et RSA. Ces deux études ont montré que l'ajout de capacités de détection ou de correction améliore la robustesse du circuit face aux différentes attaques.

MOTS CLEFS

Circuits sécurisés, attaques par fautes, attaques par canaux cachés, redondance d'information

TITLE

SECURE DESIGN AGAINST FAULT ATTACKS AND SIDE-CHANNEL ATTACKS

ABSTRACT

The increasing need for security in our every day life has resulted in the repeated use of multiple secure devices with cryptographic capabilities. Meanwhile, the cryptanalysis has made many advances allowing hackers to attack these secure devices.

This dissertation discusses a methodology to evaluate the robustness provided by counter-measures implemented in such devices. This analysis is based, on one hand, on the use of laser platforms to study the types of faults induced in a circuit prototype; and on another hand, on the use of a method based on simulations at design time to compare the impact on side-channels of protections against fault attacks.

This methodology has been experimented firstly on the simple case of a register protected by information redundancy, then on cryptographic primitives such as an AES S-Box and AES and RSA co-processors. Both of these studies have shown an increase of the robustness of the device against both types of attacks when adding detection or correction capabilities.

KEYWORDS

Secure circuits, fault attacks, side-channel attacks, information redundancy

ISBN : 978-2-84813-134-4