Méthodes pour la modélisation des injections de fautes électromagnétiques

Oualid Trabelsi

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2021IPPAT021

Thèse de doctorat

Méthodes pour la modélisation des injections de fautes électromagnétiques

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom Paris

École doctorale n°626 de l'Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat : Réseaux, Informations et Communications

Thèse présentée et soutenue à Paris, le 01/07/2021, par

OUALID TRABELSI

Composition du Jury :

Régis Leveugle Professeur, Université Grenoble Alpes (TIMA)	Président
Jean-Max Dutertre Professeur, École Nationale Supérieure des Mines de Saint-Etienne (SAS)	Rapporteur
Philippe Maurine Professeur associé, Université de Montpellier (LIRMM)	Rapporteur
Lirida Naviner Alves De Barros Professeure, Télécom Paris (LTCI)	Examinatrice
Mathieu Lisart Expert sécurité, STMicroelectronics	Examinateur
Jean-Luc Danger Professeur, Télécom Paris (LTCI)	Directeur de thèse
Laurent Sauvage Maître de conférences, Télécom Paris (LTCI)	Co-directeur de thèse

Résumé

Les attaques par injection de faute représentent une menace considérable pour les systèmes cyber-physiques. Dès lors, la protection contre ces attaques est une nécessité pour assurer un haut niveau de sécurité dans les applications sensibles comme l'internet des objets, les téléphones mobiles ou encore les voitures connectées. Élaborer des protections demande au préalable de bien comprendre les mécanismes d'attaque afin de proposer des contre-mesures efficaces. En matière de méthodes d'injection de faute, celle par interférence électromagnétique s'est vu être une source de perturbation efficace, en étant moins intrusive et avec une configuration à faible coût. Outre l'ajustement des paramètres d'injection, l'efficacité de cette méthode réside dans le choix de la sonde qui génère le rayonnement électromagnétique. L'état de l'art propose déjà des travaux par rapport à la conception et la caractérisation de ce type d'injecteur. Cependant, les résultats correspondant rapportent une différence entre ceux issus de la simulation et ceux à partir des tests expérimentaux.

La première partie de la thèse aborde la question de l'efficacité des sondes magnétiques, en mettant l'accent sur l'implication de leurs propriétés. Afin de comparer les sondes, nous proposons d'observer l'impact des impulsions électromagnétiques au niveau logique, sur des cibles particulières de type FPGA. La caractérisation est aussi établie suivant la variation des paramètres d'injection comme l'amplitude et la polarité de l'impulsion, le nombre d'impulsions ou encore l'instant de l'injection. Ces résultats ont permis de converger sur les paramètres optimaux qui maximisent l'effet des sondes magnétiques.

La caractérisation est par la suite étendue au niveau logiciel sur des cibles de type microcontrôleur. L'objet de la seconde contribution consiste à présenter une démarche d'analyse, basée sur trois méthodes génériques, qui servent à déterminer les vulnérabilités des microcontrôleurs sur les instructions ou les données. Ces méthodes portent sur l'identification des éléments vulnérables au niveau architecture, l'analyse des modèles de faute au niveau bit, et enfin la définition de l'état des fautes, à savoir transitoire ou semi-persistent. Le travail de dresser les modèles de faute, ainsi que le nombre d'instructions ou données impactées, est un jalon important pour la conception de contre-mesures plus robustes.

Concernant ce dernier point, des contre-mesures au niveau instruction ont été proposées contre les modèles de faute logiciels. Actuellement, le mécanisme le plus répandu se résume à appliquer une redondance dans l'exécution du programme à protéger. Toutefois, ce type de contre-mesure est formulé sur l'hypothèse qu'une injection de faute

équivaut un seul saut d'instruction. Vis-à-vis de nos observations, ces contre-mesures basées sur de la duplication au niveau instructions présentent des vulnérabilités, que nous identifions, puis corrigeons.

Abstract

Fault injection attacks represent a considerable threat to cyber-physical systems. Therefore, protection against these attacks is required to ensure a high level of security in sensitive applications such as the Internet of Things, smart devices or connected cars. Developing protection requires a good understanding of the attack mechanisms in order to propose effective countermeasures. In terms of fault injection methods, electromagnetic interference has proven to be an effective source of disruption, being less intrusive and with a low cost setup. Besides the adjustment of the injection parameters, the effectiveness of this attack mean lies in the choice of the probe that generates the electromagnetic radiation. The state of the art already proposes many works related to the design and characterization of this type of injector. However, the corresponding results point out to some difference between those from simulation and those from experimental tests.

The first part of the thesis addresses the question of the efficiency of magnetic probes, with a focus on their properties. In order to compare the probes, we propose to observe the impact of electromagnetic pulses at the logic level, on particular targets such as FPGA. The characterization is also established according to the variation of the injection parameters such as the amplitude and the polarity of the pulse, the number of pulses or the injection time. These results allowed to converge on the optimal parameters that maximize the effect of the magnetic probes.

The characterization is then extended to the architecture level on microcontroller targets. The purpose of the second contribution is to present an analysis approach, based on three generic methods, which are used to determine the vulnerabilities of microcontrollers with respect to instructions or data. These methods concern the identification of vulnerable elements at the architecture level, the analysis of fault models at the bit level, and finally the definition of the temporal fault status, i.e. transient or semi-persistent. Establishing the fault patterns, as well as the number of the impacted instructions or data, is an important milestone for the design of more robust countermeasures.

Regarding the latter, instruction-level countermeasures have been proposed against software fault models. Currently, the most common mechanism is to apply a redundant execution of the program to be protected. However, this type of countermeasure is based on the assumption that a fault injection imply a single instruction skip. With respect to our observations, these countermeasures based on instruction-level duplication present vulnerabilities, which we identify and then correct.

Table des matières

Remerciments

J'aimerais remercier toutes les personnes qui ont contribué de prés ou de loin à ma réussite et à l'accomplissement de ce travail.

Je pense en premier lieu à mes deux mentors et directeurs de thèse, Monsieur Jean-Luc Danger et Monsieur Laurent Sauvage, qui m'ont donné l'opportunité de faire partie de ce projet de thèse et qui m'ont accompagné de par leur disponibilité, valeurs humaines ainsi que la qualité du savoir qu'ils m'ont transmis. Des personnes qui resteront pour moi un exemple à suivre.

Mes remerciements vont aussi à tous les membres du jury à savoir Monsieur Jean-Max Dutertre et Monsieur Philippe Maurine pour avoir accepté d'être rapporteurs de ce travail de thèse, Madame Lirida Naviner, Monsieur Mathieu Lisart et Monsieur Régis Leveugle pour m'avoir fait l'honneur de participer à la soutenance de thèse et de juger mon travail.

J'adresse également mes remerciements à Madame Roselyne Chotin et Monsieur Guénaël Renault, qui avaient bien généreusement accepté d'être les membres de mon jury d'évaluation de mi-thèse.

Cela sans oublier tous les membres du département COMELEC de par son responsable Monsieur Bruno Thedrez, ainsi que ceux que j'ai pu croiser au quotidien Messieurs Yves Mathieu, Chadi Jabbour, Ulrich Kühne et j'en oublie sûrement d'autres. Une mention spéciale à Karim Ben Kalaia et Tarik Graba qui ont répondu toujours présent pour m'aider à résoudre tout genre de points techniques.

J'ai bénéficié d'un corps administratif toujours présent pour m'aider et me soutenir surtout durant les temps difficiles qui ont perturbé le monde. Ceci est l'occasion de leur présenter mes sincères et amicales considérations. Je cite Monsieur Alain Sibille, Mesdames Yvonne Bansimba, Chantal Cadiat et Florence Besnard.

Ensuite, je souhaiterais bien évidemment remercier tous ceux qui, par leur soutien et leur présence, ont fait que ces trois années de thèse restent une aventure marquante que soit sur le plan personnel et professionnel. J'ai eu la chance de vivre l'ambiance Télécom Paris aussi bien aux locaux de Dareau que dans les nouveaux locaux à Palaiseau. Ceci m'a permis de rencontrer des personnes exceptionnelles, à savoir mon conseiller et colloque de bureau Khaled Karray dans un premier temps, et Wei Chang dans un second temps. Les membres de Secure-IC Sofian Takarabt, Meziane, Sebastien Carre, Michael Timbert, Xuan Thuy Ngo, Adrien Facon ainsi que Monsieur Sylvain Guilley pour ses précieux conseils. Mon camarade de laboratoire Thanh Khuat Van avec qui j'ai partagé des bons moments. Merci aussi à tous les autres doctorants que j'ai eu

l'occasion de côtoyer au passage. Une chance d'avoir une telle bonne compagnie.

Un remerciement tout particulier à mon ami le grand Youssef Souissi, celui qui m'a ouvert ses portes et m'a accueilli à bras ouverts lui et sa magnifique petite famille, sa femme Saïda Mouna et ses deux petits anges Sami et Julia. Je te serais reconnaissant à vie mon frère et ta bonté restera un exemple à suivre.

Au cours de toutes ces années, je suis tellement heureux d'avoir des amis qui ont été là dans les bons moments comme les plus dures. Haykel, Mahmoud, Yosra, Nesrine, Mohammed Salama et sa femme Nawel, Mohammed Ali et sa femme Mariem, Mouna, Rym, Mohammed Mehdi, Narjess, Maissa, Akrem, Zouheir et sa femme Zohra, Karim, Adnen et sa femme Mouna, Rania, Fedia, Feriel, Myriam, Afif et sa femme Aroua. Sans oublier Docteur Ilef Turki, à qui je souhaite le plus glorieux des avenirs. Un grand merci à vous tous, vous êtes d'une rareté et des amis en or.

Enfin, ces remerciements ne seraient complets sans mentionner les personnes que j'aime le plus au monde, ma mère Leila Rekik, celle à qui je dois tout et dont je dédie ce travail, que Dieu la garde le plus longtemps possible ; mes soeurs Donia, Lilia et Nadia pour leur soutien continu, la bonne humeur et les heures de bavardage pour me faire sentir moins seul loin de chez nous ; Mes remerciements vont également à mon beau-frère Yassine. Un énorme merci, ma magnifique famille, sans votre soutien inconditionnel je n'y serais jamais arrivé.

Cette consécration je la dédie aussi à l'ami, le frère, mon oncle le Commandant Ali Maher Rekik, qui n'a jamais cessé de croire en moi. Mes tantes et mères Essia et Charifa et mes autres tantes que j'aime infiniment. À tous les autres membres de ma famille, cousins et cousines et particulièrement ma grande soeur Ahlem.

Je dédie aussi ce travail à tous ceux et celles qui nous ont quittés tôt pour un monde meilleur. À mes grands-parents maternels et paternels, j'espère que de là où vous me regardez, vous êtes fière de ce que je suis devenu.

Liste des figures

Liste des tableaux

Glossaire

AES Advanced Encryption Standard.

CCFI Code and Control-Flow Integrity.
CI circuit intégré.
CPU Processeur.

DES Data Encryption Standard.
DPA Differential Power Analysis.

EEFC Enhanced Embedded Flash Controller.
EM Électromagnétique.
EMFI Injection de Faute Electromagnétique.

FAME Fault Aware Microprocessor Extension.
FIA Attaques par Injection de Fautes.
FPGA Réseau de Portes Programmables.

IEM Impulsion Électromagnétique.
IRR Redondance Intra-Instruction.
ISA Instruction Set Architecture.

LLVM Low Level Virtual Machine.

MCU Microcontrôleur.

NVM Mémoire Non-Volatile.

PCB Circuit Imprimé.
PIEM1 Plateforme d'Injection Électromagnétique Agilent.

RO Ring Oscillator.
RSA Rivest Shamir Adleman.

SCA Side Channel Attacks.

SCP Systèmes Cyber-Physiques.

SIMD Single Instruction Multiple Data.

SoC System-sur-Puce.

SPA Simple Power Analysis.

TRNG True Random Number Generator.

WS Wait States.

CHAPITRE 1

Introduction

La sécurité des systèmes modernes, essentiellement numériques, est un enjeu omniprésent vu leur présence dans les taches les plus basiques de la vie quotidienne. Avec la concurrence agressive du marché et la demande incessante du consommateur à plus de nouveautés et de performances, la constante évolution de ces systèmes se fait au prix de leur intégrité. Informations personnelles, données bancaires, projets privés, un flux important de données passent par différents équipements qui sont sujets à diverses attaques par des personnes malveillantes en vue d'avoir accès à ces secrets. Les enjeux sont tellement médiatisés qu'aussi bien une personne n'ayant aucune connaissance en informatique, reconnaît aujourd'hui le sens de cheval de Troie ou d'abréviations tels que malware ou encore ransomeware, dont le dernier en date a touché l'un des plus grands opérateurs d'oléoducs des états-unis. Garantir une protection sans faille de telles informations revient donc à s'assurer de la sécurité de ces équipements.

Plusieurs mécanismes sont mis en œuvre dans le but de rendre le fonctionnement d'un équipement sensible inviolable. Or, les moyens d'attaque se sont à leur tour diversifiés ciblant la couche matérielle ou logicielle suivant les failles détectées. Au niveau matériel, l'injection de fautes, nécessite un accès physique au circuit intégré où les opérations de traitement des informations sont effectuées. Les personnes mal intentionnées chercherons donc à perturber ce fonctionnement en agissant sur les propriétés environnementales telles que la température, ou les paramètres fonctionnels comme la source d'alimentation. Certains utilisent des méthodes intrusives comme dans le cas du faisceau laser. Cette méthode est d'un coté la plus précise à mettre en place, et d'un autre coté peut aller jusqu'à la destruction du circuit.

L'injection de fautes par rayonnement électromagnétique est plus facile à mettre en place et moins intrusive, mais est moins précise que le faisceau laser. Avec la méthode laser, la technique de diminution de l'onde est assez maîtrisée avec des grandeurs capable de stimuler le minimum possible de cellules tout en gardant un niveau de puissance. Ce qui rend cette méthode plus précise pour perturber des circuits au procédé de l'ordre du nanomètre. Coté rayonnement électromagnétique, l'aspect spatial des fautes est lié à l'injecteur dont le rendement n'est pas aussi précis que le laser. La caractérisation du rendement de ces injecteurs, en l'occurrence des sondes magnétiques, est un travail

encore en cours sur les propriétés de fabrication de ces sondes, et les paramètres qui servent à définir l'injection de faute. Ces injections sont présentées sous forme d'impulsion de très courte durée. La relation entre une impulsion électromagnétique et la sonde utilisée pour générer cette impulsion et le point déterminant pour réussir à altérer le fonctionnement d'une cellule logique et générer une faute exploitable.

Les injections de fautes ont pour but de créer un dysfonctionnement dans une ou plusieurs couches d'abstraction d'un système. Ainsi un attaquant va chercher à altérer suffisamment un circuit intégré de sorte à créer à dérèglement dans les opérations en cours de traitement. Dans l'exemple d'une carte bancaire, une attaque pourrait consister à dérouter la fonction de vérification du code pin de la carte de telle sorte qu'un mauvais code soit considéré valide. La précision d'une telle attaque peut se résumer à localiser une position sensible tout en procédant à l'attaque durant le bon instant d'injection qui permet de créer la faute. Ainsi plusieurs paramètres de type spatial, électrique et temporel doivent être étudiés afin de dresser ceux qui vont optimiser la perturbation.

Cependant, les circuits numériques peuvent être protégés face aux injection de fautes. Les protections peuvent se présenter sous différentes formes. Il y a les protections matérielles qui sont intégrés lors de la phase de fabrications des circuits (renforcement du boîtier du circuit, élément cryptographique ...), ou des protections logicielles qui sont assez souvent dans des couches spécifiques dont seul le fabriquant a les moyens d'y accéder. Dans les systèmes les plus complexes on peut même trouver des contre-mesures combinant l'aspect matériel et logiciel pour maximiser la protection.

C'est pour répondre à ces problématiques que cette thèse a été menée dans le cadre du projet ANR CSAFE+ (Circuits sécurisés contre les attaques par injection de fautes électromagnétique avancée) en collaboration avec plusieurs intervenants du secteur industriel et académique. Elle est principalement réalisée au sein du groupe de recherche Secure and Safe Hardware (SSH) de Télécom Paris et dont l'objectif principal est d'élaborer des méthodes pour la caractérisation tout en étudiant l'impact qui en résulte.

Le deuxième chapitre donne une revue de l'état de l'art des différents moyens d'injection de fautes en délivrant des détails sur les attaques par rayonnement électromagnétique. Il est aussi question de rappeler les précédentes caractérisations et les modèles de faute relatifs tout en évoquant les contre-mesures élaborées pour contrer ces menaces.

Le troisième chapitre aborde la caractérisation en proposant des méthodes pour l'évaluation de l'impact des injections électromagnétiques au niveau logique sur des cibles de type FPGA. Une évaluation des sondes magnétiques sera présentée et le résultat sera pris en compte dans les caractérisations suivantes.

Le quatrième chapitre propose une caractérisation au niveau logiciel sur des cibles de type microcontrôleur. Nous y présenterons une démarche d'étude qui regroupe plusieurs méthodes d'analyse pour l'observation des vulnérabilités au niveau architecture, bit et temporel.

Le cinquième chapitre est une application des résultats obtenus dans le précédant

chapitre pour l'évaluation des vulnérabilités des contre-mesures basées sur la redondance d'instructions. Suite aux failles qui y sont observées, nous présenterons des améliorations pour maximiser la robustesse de ces contre-mesures avec à la fin une discussion sur leur niveau de résistance dans le cas des injections multiples dans le temps.

Pour finir, le sixième chapitre vient conclure ces travaux avec un résumé sur les différentes contributions et une discussion sur leur éventuel impact et les possibles perspectives.

CHAPITRE 2

État de l'art

Sommaire

Les travaux de cette thèse se basent sur les effets de la méthode d'attaque par l'injection de fautes avec la caractérisation de ses différentes propriétés tout en discutant les conséquences sur les contre-mesures. Ce chapitre propose une revue de tous ces aspects en commençant par une généralité des méthodes d'attaques et particulièrement ceux au niveau matériel. Les attaques par injection de fautes seront exposées par la suite en considérant différentes techniques. Nous nous focaliserons sur l'injection de faute électromagnétique qui est la principale technique utilisée dans cette thèse. Un aperçu des différents modèles de fautes tant au niveau logique que logiciel sera proposé. En dernier point, nous retraçons les différentes contre-mesures qui sont appliquées pour contrer ces modèles de fautes. Enfin, nous terminons ce chapitre avec une présentation des objectifs de la thèse.

2.1 Généralité sur les attaques matérielles

Le but d'un attaquant est principalement d'extraire des données sensibles. Pour arriver à ce résultat, plusieurs méthodes sont possibles avec différents niveaux de complexité et de succès. Concernant le mode d'attaque, une personne malveillante se trouve entre deux choix : soit attaquer la partie logicielle soit viser la partie matérielle. Le choix sera essentiellement lié aux difficultés d'accès au système protégé. Dans le cas des attaques logicielles, la technique repose sur l'accès logiciel par l'exploitation des failles dans le code. Ici, aucune connaissance du matériel n'est nécessaire. Par-contre, ceci n'est pas le cas si on choisit l'attaque matérielle. En effet, cette méthode requiert un accès direct au composant protégé.

Dans le cas des attaques matérielles, plusieurs scénarios sont avancés :

On trouve d'une part les attaques à caractère invasif, qui sont des attaques qui présentent un effet de modification permanente du composant cible, voir sa destruction. Ce type d'attaque est essentiellement utilisé dans un but de rétro-ingénierie matérielle [1, 2]. La décapsulation de la cible est essentielle et se fait principalement avec des moyens chimiques et mécanique. Une autre technique est le micro-sondage (microprobing) [3] où il est question de me connecter directement aux circuits internes à l'aide de micro-sonde et d'en extraire des données.

Moins destructrices, les attaques semi-invasives nécessitent néanmoins une décapsulation du boîtier de la cible pour permettre un effet maximum sur le silicium du circuit intégré (CI). L'attaque se fait sans connexion directe avec le composant en utilisant le mode par injection de faute [4] ou la lumière [2].

Enfin, les attaques non-invasives qui sont les moins coûteuse (sans préparation de la cible). Ce type d'attaque propose différentes approches : Soit en mode écoute sur les canaux auxilières avec une analyse de temps [5], de consommation [3, 6] our encore électromagntique [7], soit en générant des fautes dans les opérations à travers le signal d'horloge [8], tension d'alimentation, ou même la température [9].

Récemment, un nouveau mode d'attaque dit *Row-Hammer* offre la possibilité à un attaquant d'inverser à distance un ou plusieurs bits dans une mémoire volatile cible [10]. Les attaques récentes exploitent des failles des microarchitectures des processeurs [11, 12, 13].

Ci-après nous proposons quelques détails par rapport à des techniques d'attaque non-invasives.

2.1.1 Les attaques par canaux cachés

Ce type d'attaque non-invasive, appelé aussi attaque passive, se résume à une mesure d'une propriété physique de la cible (consommation électrique, température, émission EM ...). Ces attaques sont particulièrement faciles à mettre en place et prennent avantage des fuites d'informations laissées durant le traitement et opérations sur les données. Plus connu sous le nom Side Channel Attacks (SCA), on retrouve différentes techniques : Le Simple Power Analysis (SPA) où il est question d'une analyse d'activité sur une seule trace pour identifier la valeur des bits d'une clé [5]. Une autre technique est la Differential Power Analysis (DPA) [14] qui utilise la même analyse d'activité en mode statistique sur plusieurs traces pour générer des hypothèses sur une portion de clé secrète.

2.1.2 Les attaques par injection de fautes

Ce mode d'attaque active nécessite une action direct sur les opérations en cours d'un traitement. Le but principal consiste à générer une perturbation momentanée du système et le forcer à adapter un autre comportement qui peut révéler des informations secrètes ou exécuter des opérations supposées non accessibles par un utilisateur normal.

Les techniques et méthodes utilisées sont passées par plusieurs évolutions à commencer par Bonneh *et al.* [15] qui ont introduit cette notion avec une attaque sur l'algorithme de chiffrement asymétrique Rivest Shamir Adleman (RSA) qui exploite des fautes sur le traitement cryptographique. L'article de Biham *et al.* [16] porte sur l'algorithme de chiffrement symétrique Data Encryption Standard (DES) et les attaques proposées par Giraud [17] et celle de Piret et Quisquater [18] sur Advanced Encryption Standard (AES).

Ces attaques ne visent pas que les algorithmes cryptographiques. Le but de l'injection de faute est de créer une irrégularité dans le fonctionnement sans pour autant chercher à viser ces modules :

- Déroutement du fonctionnement nominale (éviter une fonction de test de sécurité)
- Forcer un état
- Modifier un résultat
- etc ...

Ces différentes fautes sont largement exploitées et générées par des injections de faute physiques.

Il existe des fautes dont l'impact est globale et d'autres dont l'effet est locale.

Faute à impact globale : Qu'il s'agisse de la température [19], de la tension d'alimentation [20] ou de l'horloge de fonctionnement [21], un circuit est sensible aux limites minimales et maximales de telles propriétés. Agir sur un ou plusieurs paramètres va entraîner une modification du fonctionnement qui se résume généralement à induire des fautes d'échantillonnage dues à l'accélération ou le ralentissement du calcul, ce qui entraîne un défaut de mémorisation dans un élément logique. Quand une perturbation est globale, sont effet s'étend à tout le système.

Faute à impact locale : Les injections de fautes locales nécessitent une certaine précision spatiale et temporelle puisqu'on cherche à un induire en faute un minimum d'éléments du système durant une très courte période de temps. Moins est le nombre d'éléments qui sont impactés, plus la faute est précise et exploitable.

Ci-après nous donnons des exemples de techniques d'injections de fautes les plus répondues, à savoir les perturbations rapides (*glitch*) sur l'horloge et l'alimentation, celle par injection de faisceau lumineux et enfin la méthode par rayonnement électromagnétique.

2.1.3 Méthode par perturbation de l'horloge ou de la tension d'alimentation

Bien que se sont des paramètres connus pour être globaux, il aussi possible à travers une attaque précise d'utiliser une perturbation sur la tension ou l'horloge pour créer une faute locale. Une variation rapide dans le temps, plus connue sous le nom *glitch* permet de générer ce type d'effet sur un système. Un *Glitch d'horloge* [22, 23] est défini par rapport à sa durée et à l'instant de l'injection de la faute. Alors qu'un *Glitch de tension* [24, 25, 26] est défini par sa durée, son instant d'injection, sa polarité et son amplitude.

On trouve aussi des attaques qui ont l'avantage de pouvoir combiner des injections de faute sur les deux paramètres, *Glitch combiné*, comme présenté par Korak *et al.* [27] dont l'attaque consiste à injecter un *Glitch d'horloge* en profitant d'un effet de réduction de la tension d'alimentation *underpowering*.

L'effet principal généré par ce type d'attaque est une violation du temps de *set up* à l'entrée des registre.

2.1.4 Méthode par rayonnement lumineux

L'injection par rayonnement lumineux focalisé est l'une des méthodes la plus précise de part la localité de la faute. Avec l'évolution des procédés de gravure des CI, cette méthode s'est vu être celle qui permet de suivre cette évolution. La technique consiste à illuminer un transistor et de créer un changement de son état. Outre l'utilisation d'une lumière blanche, comme le cas d'un flash d'une apareil photo [28] pour induire des fautes sur une mémoire SRAM, la méthode la plus répondue est celle par faisceau laser [29].

Il est néanmoins à noter cette technique d'injection de fautes demande une préparation de la cible par la décapsulation de la face à cibler, donc une méthode semi-invasive et assez coûteuse à mettre en place. Ce type d'attaque est réalisée le plus souvent sur la face arrière d'un composant, qui est moins réfléchissante que la face avant, car elle est constituée du substrat silicium et non des couches métalliques.

2.1.5 Méthode par rayonnement électromagnétique

La technique d'attaque par injection électromagnétique, plus connu sous EMFI, se définit par un la génération d'un champ électromagnétique qui va venir se coupler avec le composant. L'outil principale de cette méthode est la sonde magnétique qui se place au plus prés de la surface de la cible pour un meilleur effet. Contrairement à la méthode laser, cette technique ne demande pas une préparation spécifique de la cible. Il est néanmoins nécessaire dans certains cas de décapsuler le boîtier si nous voulons être au plus prés de la surface CI Aussi, de part son principe électromagnétique, le placement de la sonde peut s'effectuer sur la face avant comme sur la face arrière du composant. Cette technique d'attaque est celle qui est utilisée dans cette thèse et est développée dans la section qui suit.

2.2 Injection de faute électromagnétique

Utilisée au préalable comme moyen d'écoute, l'onde électromagnétique est très utilisé pour des attaques de type SCA. Cependant, les propriétés de l'attaque électromagnétique fait qu'elle est devenue un concurrent direct à l'attaque laser de point de vue non intrusivité.

2.2.1 Généralités

Réussir une perturbation EM implique une configuration assez spécifique des différents paramètres. Tout d'abord, on trouve la sonde d'injection avec ses différentes caractéristiques (fig. 2.4) suivi par le paramètre spatial (l'emplacement de la sonde sur la cible). Les

paramètres temporels et électriques (fig. 2.1) comme l'instant d'injection, la polarité, la largeur, l'amplitude, le temps de monté et le nombre d'impulsions définissent le modèle de la perturbation à induire.

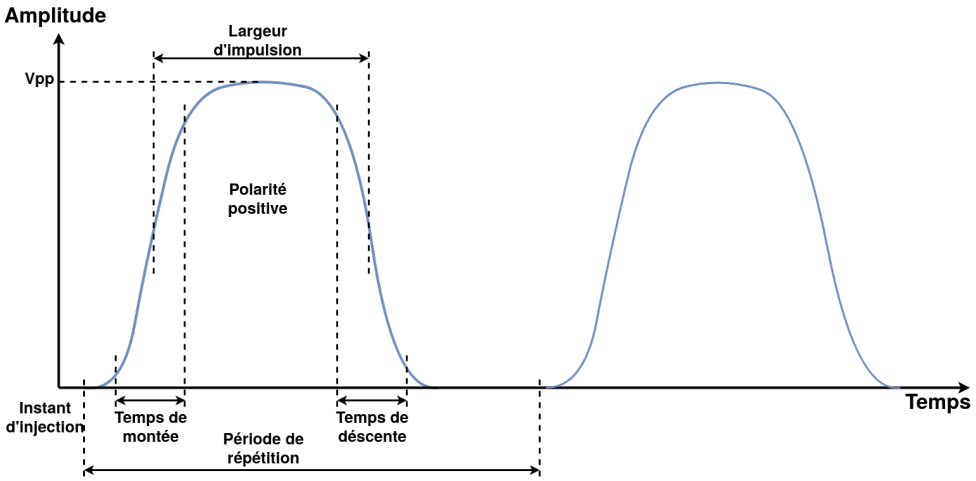


FIGURE 2.1 – Paramètres relatifs à la génération d'impulsions EM.

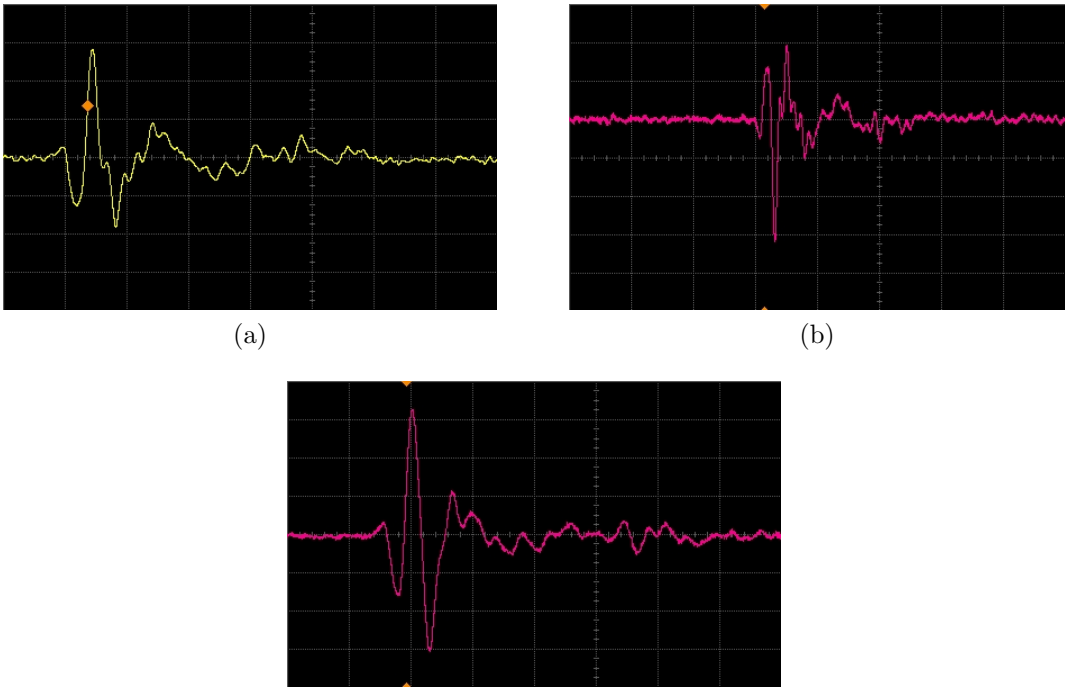
Tous ces paramètres sont important à caractériser puisqu'il vont permettre de définir la configuration optimale qui implique un couplage avec l'élément à perturber.

La première apparition de ce type d'attaque est décrite par Quisquater et Samyde [30]. Les auteurs ont réussi à générer des fautes sur la mémoire RAM en utilisant un fort courant du flash d'un appareil photo couplé avec un solénoïde pour créer un champ EM. La fiabilité de la technique a été par la suite confirmée par Vargas *et al.* [31]. S'ensuivent alors d'autres applications des injections EM avec des ondes en mode harmonique (fig. 2.2c) ou impulsionnel à polarité positive (fig. 2.2a) ou négative (fig. 2.2b).

En appliquant une onde sinusoïdale, on réalise pendant un court moment deux ondes à polarité opposée : une impulsion à polarité positive suivi par une impulsion à polarité négative, ou l'effet inverse. Bayon *et al.* [32] ont utilisé ce type de champ EM harmonique pour perturber le fonctionnement d'un oscillateur en anneaux Ring Oscillator (RO) en altérant sa fréquence, alors que Poucheret *et al.* [33] l'ont utilisé pour corrompre un générateur d'aléa vrai ou True Random Number Generator (TRNG).

Schmidt et Hutter [34] utilisent une onde impulsionnelle, comme dans [30], qui, appliquée sur les rails d'alimentation ou de masse, génère respectivement soit une chute de tension ou une un pic de tension. Dumont *et al.* ont présenté dans [35, 36] une caractérisation de l'effet des EMFI sur les rails d'alimentation et de masse. Par simulation et de manière expérimentale, ils ont montrés que l'injection de fautes durant un moment précis du fonctionnement d'une cible, en l'occurrence durant le front montant de l'horloge, peut provoquer une faute transitoire dû à la non mise à jour (*sampling-fault*) des bascules sous effet de la variation de la tension d'alimentation.

Au vu des techniques d'injection EM utilisées de nos jours, le travail de Dehbaoui *et al.* [37] est considéré comme le premier à présenter des résultats avec l'utilisation d'un générateur d'impulsion et avoir un paramétrage précis de l'impulsion EM.



(a) (b) (c)

FIGURE 2.2 – Formes d'onde EM à la sortie d'une sonde lues sur ligne microruban quand l'onde est (a) une impulsion à polarité positive, (b) impulsion à polarité négative et (c) harmonique.

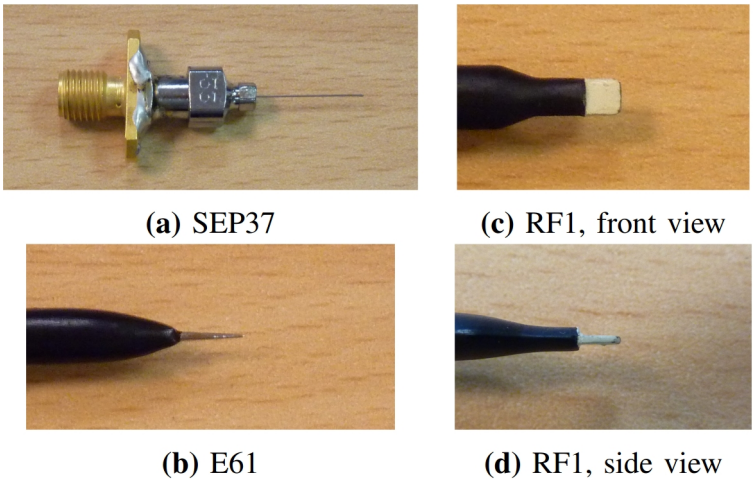
2.2.2 Injecteurs EM

La sonde magnétique est l'injecteur utilisé dans une attaque par injection électromagnétique à l'instar d'un spot laser dans l'attaque par rayonnement lumineux. Les propriétés de l'injecteur entrent en considération dans l'intensité et la forme de l'onde rayonnée. Plusieurs types de sondes à différentes formes et fonctions sont utilisées selon le mode émetteur/récepteur souhaité ou encore le couplage local/global qui va permettre de générer une faute dans l'élément cible.

On trouve les sondes électriques [38] (fig. 2.3) qui d'après leur caractérisation, montrent que des sondes professionnelles spécialement conçues pour l'injection de faute semblent moins efficaces qu'une sonde *fait-maison*.

Les sondes magnétiques sont celles dont la conception est régie par plusieurs propriétés (fig. 2.4). Ce type de sonde est caractérisé par le noyau et les enroulements de spires. Le noyau de la sonde magnétique peut se présenter sous différentes formes et matériau. On le retrouve le plus souvent en ferrite ou simplement de l'air (fig. 2.5). Le diamètre du noyau ϕ_n a un impact direct sur la résolution spatiale puisque plus la largeur du noyau se rapproche du procédé de gravure de la cible plus la résolution spatiale est meilleure. Actuellement, on arrive à avoir des sondes dont l'ordre de grandeur du diamètre atteint les centaines de micromètre. Les spires sont enroulées tout au tour du noyau et sont définis par leur nombre #Tours et le diamètre du fil ϕ_f utilisé pour les former.

Omarouayacheet et Maurine [39] ont utilisé la simulation pour la détermination de



(a) SEP37 **(c)** RF1, front view

(b) E61 **(d)** RF1, side view

FIGURE 2.3 – Exemple de sondes électriques[38] , avec (a) sonde *fait-maison* et (b), (c) et(d) deux sondes commerciales.

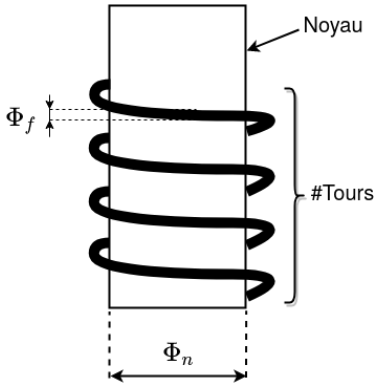


FIGURE 2.4 – Représentation des paramètres d'une sonde magnétique.

propriété qui donnerait le meilleur rendement d'une sonde magnétique. En variant les différents paramètres tel que la largeur du fil des spires ou le diamètre du noyau, les auteurs proposent différentes recommandations comme d'avoir qu'une seule spire pour un meilleur rendement, ou encore une largeur du fil de spire dix fois moins grande que le diamètre du noyau.

Dans la Figure 2.6 Ordas *et al.* [40] présentent des sondes *fait-maison* avec différentes formes de noyau qui sont en ferrite (forme plate, appointée ou oméga). Les auteurs rapportent que la forme oméga génère une meilleur concentration du champ EM et apporte une meilleur résolution spatiale que les sondes à tête plate ou appointée. Un point important ici est que de point de vue expérimental, les résultat de [40] diffèrent des recommandations générées par simulation.

Comme on peut le constater, les sondes peuvent être déterminantes de point de vue impact sur une cible. Nous présenterons dans le Chapter 3 une caractérisation de cinq sondes magnétiques, commerciales et *fait-maison* et nous discuterons de leur influence sur la logique combinatoire de cibles type FPGA.

FIGURE 2.5 – Image au rayon X d'une sonde Langer RF2-B32 à noyau d'air.

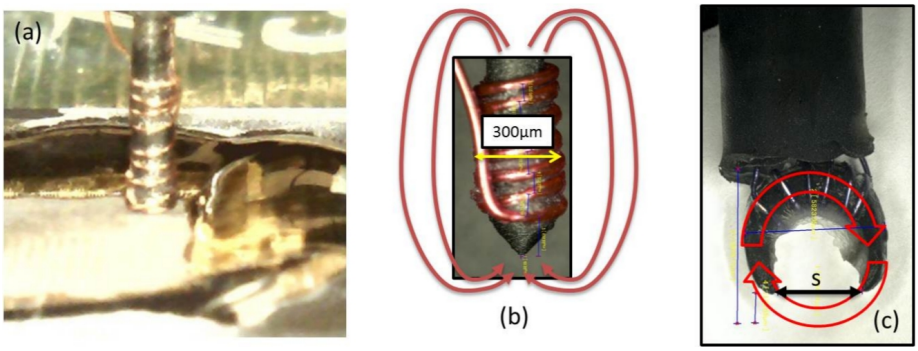


FIGURE 2.6 – (a) Sondes d'injection plate, (b) Sonde d'injection appointée, (c) Sonde d'injection oméga.

2.3 Modélisation des fautes

Pour arriver à une protection optimale contre les injections de faute, il est nécessaire de comprendre les fautes générées par les perturbations dues aux injections de faute. Des articles comme celui de Verbauwhede *et al.* [41] ou Barenghi *et al.* [42] donnent déjà une idée générale sur les modèles de fautes induites à travers les différents types d'attaques matérielles. La caractérisation d'une technique attaque reste primordiale, suite à l'évolution technologique des Systèmes Cyber-Physiques (SCP) et aussi de l'évolution des méthodes d'attaques. Même si beaucoup d'études expérimentales sont bien présentes dans l'état de l'art, l'étude de la relation effet/cible n'est pas aussi bien présentée sur les différents niveaux d'abstraction. Peu de travaux donnent des détails sur l'effet entre le niveau logique ou bit et le niveau exécution logiciel, ce qui est indispensable pour concevoir des contre-mesures au niveau logiciel [43, 44].

Sous effet des Attaques par Injection de Fautes (FIA), et suivant différents niveaux, différents modèles de fautes et effets peuvent être observés [45]. Concernant l'EMFI, des études ont été menées pour présenter les modèles connus à ce jour au niveau logique et logiciel. Ci-après nous présentons ces études et leurs observations sur les deux niveaux en incluant les résultats d'autres techniques d'injection de faute.

2.3.1 Modèles de fautes au niveau logique

L'impact des injections de faute sur les éléments logiques constituent un premiers pas dans la compréhension des phénomènes liés au couplage entre le moyen d'injection et les éléments bas niveau d'une cible. Comme vu précédemment, la perturbation de

l'alimentation ou de la fréquence d'horloge, que se soit avec un effet global ou local, génère un comportement altéré des SCP.

Au plus bas niveau d'une architecture matérielle, les perturbations ont un effet direct sur la logique combinatoire (propagation des signaux) et la logique séquentielle (bascules). Les fautes dites de *timing* sont celles qui sont induites quand il y a interférence avec les contraintes temporelles définies par une technologie. Vargas *et al.* [31] ont donné une première idée sur ce type faute et considéré dans [37, 46] une attaque sur un calcul de chiffrement AES, où l'explication par rapport aux fautes observées sur un ou plusieurs bits conclue à des fautes de *timing*. Dans [47], des fautes au niveau bit ont permis d'altérer le même bit d'un chiffré avec un taux de reproductibilité de 100%.

D'autres fautes dites fautes d'échantillonnages ont été observées comme évoqué dans [40] où une caractérisation de l'effet EMFI sur un circuit dont l'horloge est arrêtée a permis d'observer des fautes au niveau bit (*bit-set, bit-reset*. Les auteurs ont conclu qu'il est ainsi possible d'induire des fautes autres que celles qui altèrent le facteur temporel d'un calcul. Dans [48, 49] les mêmes auteurs présente un autre modèle appelé *sampling fault*, où durant une fenêtre de temps proche du front montant de l'horloge, des EMFI suffisamment fortes peuvent altérer l'échantillonnage des bascules D en induisant un effet sur ses différentes entrées.

Au niveau bit, ces modèles de fautes engendrent les principaux effets connus à savoir le *bit-set* pour le forçage d'un bit à un, le *bit-reset* pour un forçage à zéro, le *bit-flip* avec l'inversion d'un bit à sa valeur opposée ou encore le *no-sampling* en cas de non mise à jour du bit (reste avec sa valeur précédente). On peut déjà imaginer l'effet de chaine qui se créer suite aux fautes sur la logique. D'où l'importance d'avoir une caractérisation au niveau logique pour mieux cerner les possibles modèles de faute qui vont apparaître au niveau logiciel.

2.3.2 Modèles de fautes au niveau logiciel

De nombreuses études ont été menées pour caractériser le comportement d'un microcontrôleur sous FIA, de manière expérimentale. Lorsque la cible est un microcontrôleur COTS (Circuit-Off-The-Shelf) avec un cœur de processeur embarqué, les auteurs suivent une approche de *boîte-noire* car peu d'informations sont fournies sur la structure interne de la cible. Les autres cibles sont des processeurs synthétisables (ou processeurs logiciels) implémentés sur FPGA [50]. Un processeur logiciel offre la possibilité d'optimiser la conception et d'ajouter des protections dans le microcontrôleur.

La perturbation du flux du programme pendant son exécution peut être générée au niveau du flux de contrôle ou du flux de données. Généralement, une première étape consiste à analyser l'exécution normale de l'application logicielle afin d'identifier la partie du programme la plus sensible à la FIA. Ensuite, un signal de déclenchement peut être déterminé pour injecter une faute à un moment précis de l'exécution. Par une approche itérative, une deuxième étape consistera à trouver les paramètres optimaux d'injection qui augmenteront l'apparition d'une faute exploitable.

L'efficacité des EMFI, à perturber le flux de programme des MCU modernes, a déjà été démontrée dans de nombreux articles. Avec leur impact avéré sur les systèmes sécurisés, la caractérisation de leur effet a déjà exposée certains dysfonctionnements

dans le flux de programme. Ainsi, les modèles de faute les plus observés sont le saut d'instruction, la duplication (ou rejeu), le remplacement d'instruction, et la corruption de donnée.

Le modèle le plus commun est le remplacement d'instruction. Une modification dans le flux de contrôle engendre un changement dans l'instruction à exécuter, qui sera donc interprétée différemment par le processeur. Dans des cas bien précis, ce changement peut induire à une non-exécution de l'instruction, ce qui est par la suite assimiler à un saut d'instruction. d'autres cas plus complexes peuvent engendrer un rejeu d'une instruction précédemment exécutée.

Plusieurs travaux sur la caractérisation de ces modèles de faute au niveau logiciel ont été élaborés suivant les techniques d'injection précédemment évoquées. Balasch *et al.* [51] est l'un des premiers articles qui a mis en évidence des modèles de fautes avec une méthode basée sur l'étude au niveau binaire. Son approche est d'analyser l'impact de la perturbation de l'horloge sur l'exécution d'une seule instruction qui est isolée par des instructions de type `NOP`. La cible d'analyse est un MCU 8 bits ATMega163, intégrant un pipeline à deux étages *FETCH* et *EXECUTION*. Grâce à une analyse au niveau logique, les modèles de fautes tels le saut d'instruction, le remplacement ou encore le rejeu, ont été identifiés. Étant donné que les fautes sont observées durant deux cycles successifs, l'auteur est arrivé à la conclusion que l'impact est induit au niveau des deux étages du pipeline.

Korak *et al.* [27] s'est inspiré du précédent travail pour caractériser l'impact de la perturbation de l'horloge sur une cible 8 bits ATMega256 (pipeline à deux étages). Son analyse apporte plus de détails puisque la caractérisation inclut différents types d'instructions (arithmétique, logique, mémoire ...). Aussi, les codes de test ne se limitent pas à une seule instruction, mais à des instructions successives. Des modèles tel que le saut d'instruction, ou encore le remplacement d'instruction ont été observés. L'aspect d'analyse sur plusieurs instructions dans un même code a permis de conclure que l'effet des perturbations est aussi au niveau des deux étages du pipeline. Entre autres, les expérimentations ont mis en lumière la possibilité d'altérer deux instructions successives.

Ce n'est qu'avec Moro *et al.* [52] qu'une première caractérisation en EMFI a été présentée. La cible des tests est un MCU 32 bits qui intègre un cœur ARM Cortex-M3 avec un pipeline à trois étages *FETCH*, *DECODE* et *EXECUTE*. Á noter que la cible en question ne comprend pas un module de cache. L'analyse s'est distinguée dans son approche à varier les paramètres d'injections et à définir la relation entre ces paramètres et la faute générée. Les fautes observées sont principalement des remplacements d'instructions ou la corruption de données. L'auteur a aussi démontré que l'effet engendré par les EMFI sur le MCU sont au niveau du bus de transfert de donnée.

Une autre étude sur la ligne de donnée a été proposée par Menu *et al.* [53] sur une cible différente. Le MCU proposé dans la caractérisation est un Atmel SAM3X8E intégrant un ARM Cortex-M3. L'auteur a suivi une analyse au niveau binaire pour observer des fautes sur le module de *PREFETCH* de donnée. Avec des EMFI sous polarités positives ou négatives, il été possible d'induire différents modèles de faute tel le *bit-set*, *bit-reset*, *bit-flip* ou encore *update fault*. Aussi, la variation des paramètres

spatio-temporels ont mis en évidence le fait d'altérer le chargement de quatre données successives avec une seule injection de faute.

TABLEAU 2.1 – Travaux de caractérisation de modèles de faute suivant différentes méthodes d'injection de faute.

Méthode d'injection	**Ref.**	**Cible**	**Modèle de faute logiciel**
Perturbation d'horloge et/ou de la tension d'alimentation	[51]	8 bits ATmega163	corruption d'instruction
	[27]	8 bits ATxmega256	duplication d'instruction
		32 bits ARM Cortex-M0	saut et duplication d'instruction
	[54]	RISC processor (LEON3)	corruption de donnée
Faisceau laser	[55]	8 bits ATmega328P	corruption d'inst. et de donnée
	[56]	8 bits ATtiny841	saut d'instruction
	[57]	8 bits ATmega328P	saut d'instruction
Injection électromagnétique	[52]	32 bits ARM Cortex-M3	corruption d'inst. et de donnée
	[58]	32 bits ARM Cortex-M4	rejeu d'instruction
	[59]	32 bits ARM Cortex-A9	corruption d'inst. et de donnée
	[60]	8 bits PIC16F687	corruption d'instruction
	[61]	32 bits ARM Cortex-M3	corruption d'instruction
	[62]	64 bits ARM Cortex-A53	corruption d'instruction
		Intel core i3	corruption d'instruction
	[63]	64 bits ARM Cortex-A53	corruption d'instruction
	[53]	32 bits ARM Cortex-M3	corruption de donnée
	[64]	8 bits ATmega328P	saut d'instruction

Sur une cible assez similaire, et avec le même moyen d'injection de faute, l'étude de Rivière et al [58] vient compléter les deux précédents travaux. La cible étant un MCU 32 bits intégrant un cœur ARM Cortex-M4 avec un pipeline à trois étages. La différence avec le MCU proposé par [52], c'est qu'il incorpore un module de cache, qui est l'objet de la caractérisation. Plus précisément, c'est l'effet sur le cache d'instruction qui est présenté. La méthode d'analyse se base sur l'observation des fautes quand le binaire est chargé depuis le cache vers le Processeur (CPU). Il été ainsi possible de repérer un rejeu de quatre instructions successives. Les auteurs expliquent l'effet observé par une non mise à jours de la mémoire tampon du cache.

Avec l'émergence de nouveaux System-sur-Puce (SoC), Trouchkine *et al.* [63] avance une caractérisation sur une cible 64 bits multi-cœurs. Cette dernière est le Broadcom BCM2837, qui intègre quatre cœurs ARM Cortex-A53 (pipeline à huit étages). Une des évolutions avec ce type de cible est que chaque cœur admet un cache de premier niveau (L1) qui lui est dédié. On retrouve aussi un cache de niveau deux (L2) dont l'accès est par contre commun pour tous les cœurs. Bien que l'étude se soit limitée sur un seul cœur, l'auteur propose une analyse au niveau microarchitecture et se focalise sur l'effet des EMFI sur les différents niveaux de cache. Les modèles de fautes observés sont principalement liés au remplacement d'instruction ou la corruption de donnée.

D'autres travaux [54, 55, 56, 57, 60, 61, 62, 64] présentent des caractérisations étendues sur d'autres cibles pour valider les modèles de fautes.

Comme nous pouvons le constater, tous ces travaux (Tableau 2.1) montrent que les modèles de fautes observés sont principalement dues à une modification de l'instruction à exécuter. Avec l'évolution des techniques d'attaques, le nombre d'instructions qui peuvent être altérées a aussi évolué (jusqu'à 300 instructions successives avec l'injection de faute laser [57]). L'analyse au niveau architecture a aussi mis en évidence que, l'évolution technologique des MCU, n'engendre pas forcément une meilleur sécurité dans le traitement des données sensibles. Reste que la plupart des méthodes et analyses

présentées manquent de précision. Un point important à détailler, est la relation entre les paramètres d'injection de faute et les modèles de faute que ce soit au niveau logique ou instruction. Aussi, une question qui reste récurrente et nécessite plus d'étude, est celle par rapport à l'élément de l'architecture qui présente des vulnérabilités.

2.4 Contre-mesures aux injections de fautes

Les SCP sont sujets à des erreurs dues à des actions non intentionnelles (radiations, couplage EM, etc.) qui conduisent à la conception de systèmes tolérants aux fautes [65]. Les contre-mesures associées, qu'elles soient matérielles ou logicielles, ont fourni la base des premiers mécanismes de défense contre les attaques par injection de fautes [66]. Les attaques par injection de fautes sont des attaques intentionnelles menées par un attaquant dans le but d'extraire des informations (par exemple, une clé cryptographique) ou d'obtenir un accès non autorisé (par exemple, contourner un algorithme de vérification du PIN). Depuis lors, de nombreux mécanismes de défense ont été introduits que se soit sur le plan matériel ou logiciel [42]. Ci-après, nous passons en revue différentes propositions et études de ces contre-mesures.

2.4.1 Contre-mesures matérielles

Contre les attaques physiques, et avant de penser sur une refonte interne de la technologie, les constructeurs proposent des améliorations externes liées à la structure de l'enveloppe ou boîtier du CI. L'ajout de couche de substrat, durcissement de boîtier servent comme première ligne de défense face aux perturbations externes. Cela concerne les attaques passives ou actives, dans le sens où on protège contre les perturbations entrantes ou les radiations émises par le CI lui même. Des solutions comme celles proposées par [67] implique l'ajout d'une couche en forme de maillage de pistes métalliques comme une sorte de bouclier passif, ou avec une propriété active comme dans [68] où le but est de rendre des émissions traversantes (comme celles du rayonnement EM) instables.

Il existe aussi des protections qui demandent une modification dans la technologie comme celle de [69] où une modification dans l'aspect structurel de la mémoire SRAM est proposé. Une autre méthode liée à l'évolution de la technologie est le procédé de gravure dont la réduction rend difficile la génération de fautes locales. Actuellement, il n'a que la méthode par faisceau lumineux qui arrive à suivre cette évolution mais avec un coût non négligeable.

Autres méthodes de protection est l'utilisation de capteurs physiques. Ils sont généralement dédiés pour prévenir un moyen d'attaque précis. Que ce soit pour la détection de variation de l'horloge , de la tension, température ou contre le champ EM, le mécanisme du détecteur se déclenche dès qu'il y a une variation par rapport à un niveau prédéfini. Pour les fautes sur la tension, ce type de détecteur est proposé dans [70], alors que [71] et [72] proposent celui dédié aux faisceau lumineux.

Pour les attaques par EMFI, on retrouve plusieurs méthodes de différentes complexités. Dans [73], les auteurs proposent d'intégrer une antenne pour détecter la présence d'une sonde au plus prés de la cible. Sauf que leur technique ne permet pas de détecter les radiations à fort courant. Aussi, dans un environnement où les radiations

sont permanentes (cas spatial) cela engendre forcément des faux positifs en terme de détection. Un brevet [74] a été avancé dans ce sens pour détecter les variations de courant.

En utilisant des détecteurs de délai, Zussa *et al.* [75] ont présenté une protection contre les violations des contraintes temporelles. Le même principe a été utilisé dans [76] et proposé pour détecter le ralentissement ou l'accélération du temps de propagation.

Un autre principe de détection est présenté en [77, 78] qui implique l'utilisation d'une PLL couplé avec un RO. Comme évoqué précédemment, les injections de fautes peuvent déstabiliser le fonctionnement du RO. Quand la sortie de ce dernier est connecté à une PLL, l'effet induit sur le RO se répercute sur la PLL qui déclenche une alarme. L'utilisation d'une telle protection nécessite une utilisation conséquente de ressources. Le même principe est utilisé par [79] en couplant le RO avec un détecteur de phase [80]. Cette proposition a l'avantage d'avoir un taux de détection assez élevé mais en contre partie un taux assez conséquent de faux positif.

Des propositions plus simples à mettre en place connus comme des détecteurs digitaux on été développés pour contrer les EMFI. Un des premiers a été présenté par El Baze *et al.* [81, 82] et se base sur une configuration de bascule D qui servent à détecter des fautes d'échantillonnage. D'après une disposition spécifique entre les bascules, cette configuration est capable de détecter n'importe quelle combinaison d'erreurs : une faute sur une bascule ou plusieurs permet de lever une alarme suite à une comparaison entre leurs valeurs respectives. Une autre détection qui applique la même méthode est proposée dans [83] où seulement deux bascules sont utilisées pour la comparaison finale. Comme ce sont des détecteurs digitaux, plus leur placement couvre de surface du CI, plus le taux de détection est élevé.

Dans cette thèse on ne tiendra pas compte de ce type de contre-mesure et on se limitera aux protections logicielles.

2.4.2 Contre-mesures logiciels

Une des premières propositions de contre-mesure au niveau instruction a été proposée par Barenghi *et al.* [84], elle est basée sur la duplication et même la triplication d'instruction. Une méthode avec calcul de parité a aussi été présentée pour la protection de la ligne de donnée. Pour le cas de la méthode par duplication, cette dernière sert surtout pour la détection de faute si un saut est appliqué sur une seule instruction. Cette fonction de détection est signalée par une comparaison des résultats de la redondance. Par contre, pour le cas de la triplication, elle intègre en plus une partie qui sert à la correction de la faute. Ces contre-mesures résistent bien au saut d'instruction mais sont limitées par le type de l'instruction à protéger. En effet, ils se trouve qu'ils sont applicables principalement pour les instructions dites idempotentes (instruction dont l'exécution multiple n'impacte pas le résultat).

C'est pourquoi Moro *et al.* [85] ont proposé une méthode qui prend en compte les différents types d'instructions. On trouve ainsi trois types d'instructions : idempotentes, non-idempotentes et spécifiques. Pour chaque type, une méthode de redondance est appliquée. Pour les instructions idempotentes, la contre-mesure se résume à une simple

duplication de l'instruction à protéger. Dans le cas d'une instruction non-idempotente, la méthode passe par deux étapes. L'instruction cible est d'abord transformée en sous instructions idempotentes, ensuite, chaque instruction est à son tour dupliquée. Enfin, pour le cas des instructions spécifiques, la méthode est similaire à la précédente, sauf que la décomposition en sous instructions idempotentes est plus complexe. Il est à noter que ces méthodes ne présentent pas de fonction de détection comme celle proposée dans [84].

Comme mentionné précédemment, ces méthodes sont appliquées spécifiquement contre le saut d'instruction. Plus précisément, elles ne tiennent compte que du cas d'un saut induit à travers une seule injection de faute. La plupart des autres contre-mesures proposent des solutions contre le même modèle de faute tout en essayant de minimiser le coût en terme de mémoire et de calcul.

D'ailleurs, ces deux contre-mesures étaient le point de départ pour des améliorations ou de remodelage comme le cas de la contre-mesure appliquée par Barry *et al.* dans [86] en utilisant un Low Level Virtual Machine (LLVM) modifié.Les auteurs ont modifié la précédente contre-mesure en utilisant une nouvelle approche. Le mécanisme adopté est de générer pour toutes les instructions une instruction idempotente équivalente, puis procède au processus de duplication. La méthode introduit également un mécanisme d'ordonnancement des instructions avec une réorganisation de l'ordre d'exécution des instructions modifiées afin de garantir un meilleur temps d'exécution. Avec cette approche, il a été possible de réduire la vitesse d'exécution et le coût du code d'environ la moitié par rapport aux résultats obtenus dans [85].

Dans le [87], Yao et Schaumont présentent une autre contre-mesure contre le saut d'instruction. La contre-mesure est évaluée sous simulateur de CPU LEON3 d'Aeroflex Gaisler. La solution proposée est basée sur l'appel d'une fonction wrapper de détection de faute qui compare la valeur attendue d'un registre cible, suite à une opération, par rapport à sa valeur initiale. Si le résultat de la comparaison est vrai (les deux valeurs sont identiques), cela signifie que l'opération n'a pas été exécutée, ce qui reflète un comportement de saut Les auteurs indiquent que de point de vue ressources, leur approche est moins coûteuse que la méthode de redondance de base.

Patrick *et al.* présentent dans [88] une contre-mesure basée sur Redondance Intra-Instruction (IRR) en utilisant la méthode de découpage de bits. L'évaluation de la contre-mesure est effectuée avec le même simulateur que celui utilisé dans [87]. Trois versions de la contre-mesure sont proposées avec une application à l'algorithme de chiffrement AES. La première version utilise uniquement la méthode IRR, la deuxième version utilise la méthode IRR *pipelinée* et la dernière version utilise la méthode IRR *pipelinée* mélangée. Chacune de ces versions est une réponse à la complexité de l'injection de fautes, des fautes de bits aux fautes de bits multiples. Les auteurs montrent l'efficacité de leurs solutions par l'évaluation des couvertures de fautes par rapport au modèle de faute de saut d'instruction. Par rapport à la version non protégée du logiciel cible, le taux de ressources utilisées (cycles d'exécution, taille de code) n'est pas conséquent.

Lac *et al.* ont discuté dans [89] un nouveau paradigme utilisant Single Instruction Multiple Data (SIMD). La méthode de contre-mesure tire parti des redondances spa-

tiales, notamment contre le modèle de faute de saut d'instruction. L'évaluation a été réalisée sur des microcontrôleurs basés sur ARM Cortex-M3 et ARM Cortex-M4. Les applications testées sont le chiffrement par bloc léger PRIDE et le chiffrement par flux léger TRIVIUM. Le moyen d'injection de fautes utilisé est une rafale d'impulsion EM ciblant le CI de la puce. La vérification des performances de la contre-mesure indique que la charge en ressources dépend soit de l'architecture cible, soit du programme cible. Les auteurs concluent que leur méthode présente une bonne efficacité pour déjouer la méthode d'injection de fautes testée. De plus, ils soulignent qu'elle ne nécessite aucune modification supplémentaire du côté matériel.

Les deux contre-mesures basées sur la redondance de [84, 85] ont été soumises à une évaluation pratique par Moro *et al.* dans [90] en utilisant la technique EMFI. Les tests, suivant la méthode de [85], rapportent que la contre-mesure est **efficace** contre l'injection de fautes sur **une instruction isolée**. En revanche, les auteurs soulignent que cette contre-mesure doit être améliorée car elle ne prend en compte qu'un modèle de faute sur une seule instruction. Les tests de la protection contre la duplication d'instructions de [84] rapportent essentiellement les mêmes observations. Les auteurs concluent que la contre-mesure est efficace pour le jeu d'instructions testé, et proposent de l'étendre aux autres jeux d'instructions.

Une autre évaluation expérimentale a été présenté par Yuce *et al.* dans [43] sur les contre-mesures de [84, 85] Cette fois, le moyen d'injection de faute est le *Glitch d'horloge*. Les résultats des tests diffèrent de ceux présentés dans [90], puisque les contre-mesures **ne sont pas entièrement sécurisées contre une seule injection de glitch**. Notez que ces tests ont été effectués sur une cible différente (un coeur LEON3). Les auteurs suggèrent que le développement de contre-mesures qui ne prennent en compte que le microprocesseur Instruction Set Architecture (ISA) n'est pas suffisant, et qu'il est nécessaire de considérer les aspects micro-architecture de la cible.

La même approche est également discutée par Laurent *et al.* dans [44]. En mode simulation, ils démontrent l'efficacité de l'injection de fautes sur un seul bit contre la contre-mesure de duplication d'instructions détaillée dans [84]. La configuration de la simulation consiste en une version 64 bits de RISC-V. En injectant une faute de *bit-flip* sur les signaux de contrôle du pipeline, les auteurs listent les comportements possibles de l'instruction protégée à travers une analyse faite au niveau de la micro-architecture. La conclusion de l'étude souligne comme dans [43] que la sécurisation du niveau matériel peut améliorer et aider à déjouer plus efficacement les fautes observées et celles qui restent.

2.4.3 Contre-mesures combinées

Outre les contre-mesures au niveau instruction ou algorithmique, certaine contre-mesures tiennent compte de la partie matérielle comme souligné par [43, 44].

Danger *et al.* proposent dans [91] une solution basée sur un élément supplémentaire implémenté dans le matériel appelé Code and Control-Flow Integrity (CCFI). La méthode est présentée comme une contre-mesure générique et n'est pas intrusive puisqu'elle ne nécessite aucune modification du coeur. Ce bloc supplémentaire est composé de deux parties : une première partie est utilisée pour stocker les métadonnées liées au code et aux informations du flux de contrôle, et le second module est nécessaire

pour la vérification de l'intégrité du code et du flux de contrôle. Cette méthode a été testée sur une implémentation de PicoRV32 basée sur RISC-V (pipeline à trois étages). Leur évaluation a montré qu'elle offrait une protection contre les attaques par fautes simulées.

Yuce *et al.* démontrent dans [92] l'efficacité d'une nouvelle contre-mesure basée sur la structure nommée Fault Aware Microprocessor Extension (FAME). L'évaluation a été effectuée en comparant le code protégé avec FAME, sa version non protégée et à une version protégée en utilisant la méthode de duplication des instructions. La contre-mesure présentée est une combinaison de blocs matériels et logiciels. Le côté matériel est composé d'un détecteur matériel (capteur basé sur les délais) pour prévenir les erreurs de synchronisation et d'un bloc de point de contrôle matériel pour assurer le rétablissement de l'état après l'injection d'une erreur. Le côté logiciel représente un bloc qui gère le drapeau d'erreur et exécute la réponse à l'erreur spécifique à l'application. Les auteurs notent qu'il n'est pas nécessaire de modifier le code du programme puisque seules deux instructions supplémentaires sont ajoutées à l'architecture de base.

Toutes les contre-mesures présentées sont principalement destinées à contrer le même modèle de faute qui est le saut d'instruction. Dans chaque proposition, il est toujours question de rendement, taux de ressources et nombre de fautes. Dans cette thèse nous allons nous intéresser particulièrement aux deux contre-mesures de référence [84, 85] pour une étude théorique et puis expérimentale de leur robustesse dans le cas d'une et plusieurs fautes. Aussi, nous proposons d'étendre l'expérimentation à l'étude des fautes multiples dans le temps et de constater les limites de ce type de méthode par redondance d'instruction.

2.5 Objectifs de la thèse

Les attaques matérielles que subisse un système sécurisé sont de plus en plus diverses et complexes. Ce qui implique une veille continue de la part des concepteurs et développeurs de technologies pour contrer ces menaces. Dans cette course entre évolution de performances et intégrité de fonctionnement, il n'est pas toujours évident de garder un certain équilibre. L'état de l'art donnent déjà une idée sur la complexité de cette tache à combiner sûreté de fonctionnement et le coût que ça engendre. L'évolution des procédés de gravure ne semble pas freiner les attaques qui évoluent à leur tour.

Entre contre-mesure matérielle ou logicielle, il n'est pas évident de protéger un niveau sans pour autant négliger un autre. Établir une contre-mesure fiable nécessite au préalable d'avoir une connaissance de tous les effets qu'engendrent les attaques de tous genres, et plus précisément celles par injection de faute. Pour arriver à cette maîtrise, il faut préalablement passer par une phase de caractérisation qui se fait à différentes étapes. De nos jours, l'attaque en faute par injection électromagnétique est une de ces attaques qui présente le meilleur apport entre précision, réussite et coût de mise en place. Le rayonnement électromagnétique en lui même est un de ces effets qui demande une grande attention vu les avantages qu'il offre à produire des attaques passives ou actives. Aujourd'hui, les différentes études essayent de donner le plus de détails sur les effets engendrés par ce type d'attaque mais s'arrêtent à un stade de génération de fautes sans pour autant s'attarder sur les causes.

Une technique comme EMFI admet plusieurs propriétés tant dans ses paramètres que dans de son injecteur. Réussir à produire des fautes exploitables nécessite donc d'analyser l'impact de toutes ces propriétés tout en étudiant le couplage induit avec une cible. C'est déjà le cas avec les études qui ont établi les fautes d'échantillonnage ou de *timing*. Toutefois, ces études manquent de méthodes claires, voir génériques qui apportent une observation détaillée des conséquences du couplage engendré par les injections EM. Entre autres, cette connaissance de l'effet à bas niveau est une information capitale pour permettre l'adaptation des contre-mesures qui se limitent au niveau logiciel pour des questions de coût.

Dans cette thèse, l'objectif est d'avancer des méthodes qui maximisent l'observation de différentes vulnérabilités dues à l'injection de faute électromagnétique. L'application de ces méthodes se fera par la caractérisation des paramètres de l'injection EM avec une étude du couplage de rayonnement EM à différents niveaux des cibles d'études. Cet objectif est traité en trois étapes où nous exposerons nos observations par rapport aux questions qui nous ont amené à formuler des méthodes d'analyse dédiées :

1. Quel est le lien entre les paramètres de EMFI, les propriétés d'une sonde magnétique et l'effet engendré au niveau logique ?
2. Comment identifier les vulnérabilités d'un circuit face aux EMFI à travers des méthodes génériques ?
3. Quel impact peut avoir les fautes multiples sur les contre-mesures logicielles à base de redondance d'instruction ?

CHAPITRE 3

Impact du rayonnement électromagnétique sur FPGA

Sommaire

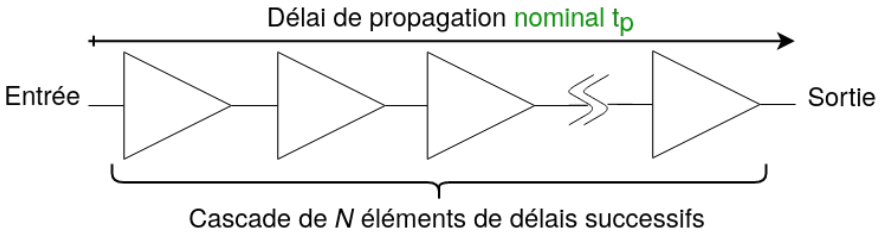
Pour réussir une EMFI, il faut considérer l'efficacité de la plateforme d'injection dans sa capacité à favoriser l'apparition de fautes exploitables sur le CI. Cette efficacité dépend en grande partie des injecteurs, en l'occurrence des sondes magnétiques, mais aussi de tous les autres paramètres qui définissent les impulsions EM. Dans ce chapitre nous proposons une comparaison de sondes d'injection suivant l'observation de l'impact du rayonnement électromagnétique au niveau logique, c'est-à-dire sur les portes logiques d'un CI cible. Pour évaluer l'impact, nous présentons une méthode qui permet de mesurer le délai de propagation d'un chemin logique combinatoire lorsque les différentes propriétés des EMFI sont sujettes à des variations. Notre évaluation comprend plusieurs sondes de différentes propriétés et les expérimentations sont réalisées sur des cibles de type FPGA issues de différents constructeurs.

Les résultats obtenus lors de cette caractérisation ont fait l'objet de deux publications intitulées *Characterization at Logical Level of Magnetic Injection Probes* [93] et *Impact of Intentional Electromagnetic Interference on Pure Combinational Logic* [94].

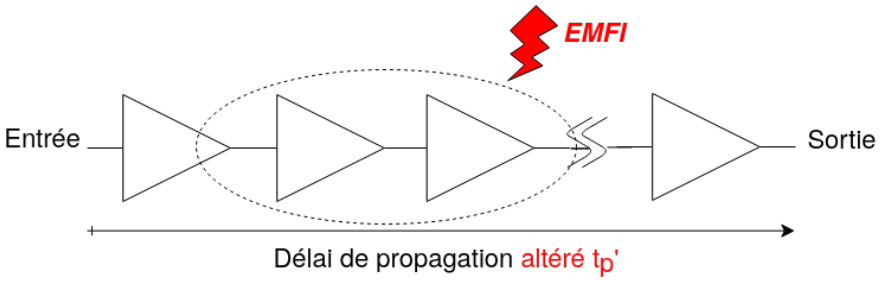
3.1 Analyse théorique de l'impact sur la logique combinatoire

Des fautes dans un CI sont créées soit directement lorsque l'état logique des éléments mémoire est inversé, soit lorsque les temps de propagations des chemins combinatoires ont tellement augmentés que la sortie échantionnée n'est pas correcte [40]. Ces modèles de faute au niveau logique sont alors observés soit sur la logique séquentielle (*bit-set*,

bit-reset ...) soit sur la logique combinatoire (modification du temps de propagation). Dans [40] il été question d'une étude sur l'impact des EMFI sur la logique séquentielle en utilisant une grande chaîne de bascules. Nous proposons pour notre caractérisation l'étude de l'effet des EMFI sur la logique combinatoire moyennant une grande cascade de portes combinatoires formant une chaine à retard.



(a)



(b)

FIGURE 3.1 – Impact d'une injection EM sur une cascade d'éléments de retard.

Dans ce qui suit, le délai de propagation nominal dans une chaine de portes combinatoires est noté t_p (fig. 3.1a). Sous l'effet des EMFI, les perturbations vont induire des variations sur ce délai provoquant un nouveau délai de propagation t'_p (fig. 3.1b). A la position (x, y) d'une sonde d'injection au dessus d'une cible, et en fonction de l'emplacement des portes combinatoire, la variation de l'impact $\Delta t_p(x, y)$ peut être évaluée comme étant la différence entre $t'_p(x, y)$ et t_p eq. (3.1). Ainsi, plusieurs paramètres (position de la sonde, son diamètre, la puissance générée ...) peuvent influencer la valeur de cette variation. Cela va dépendre entre autres du nombre de portes combinatoires de la chaine de retard qui seront impactées.

$$\Delta t_p(x, y) = t'_p(x, y) - t_p \tag{3.1}$$

La Figure 3.2 représente le chronogramme théorique quand une ou plusieurs IEM sont générées durant la propagation d'un signal dans une chaine de retard : Lors d'un fonctionnement normale, à l'instant t_{in}, le signal d'entrée passe à l'état logique haut qui défini le début de la chaine de retard. Le signal arrive à la sortie à l'instant t_{out} et passe à son tour l'état logique haut à la fin du temps de propagation nominale t_p eq. (3.2). Sous effet du rayonnement EM, le signale de sortie est décalé à un autre instant t'_{out} eq. (3.3).

$$t_p = t_{out} - t_{in} \tag{3.2}$$

$$t'_p = t'_{out} - t_{in} \tag{3.3}$$

La variation Δt_p entre ce nouveau instant de sortie et celui du fonctionnement normale peut être interprété sous deux formes :

- $\Delta t_p > 0$: La perturbation EM produit un ralentissement de la propagation.
- $\Delta t_p < 0$: La perturbation EM produit une accélération de la propagation.

Comme mentionné précédemment, la valeur maximale et minimale de cette variation dépend de plusieurs paramètre. Ceci étant, une injection hors la fenêtre de temps délimitée par les instants t_{in} et t_{out} n'a pas d'incidence sur la variation. En effet, générer une IEM alors que la sortie a été déjà mise à jour, ne pourra logiquement pas produire de modification du délai de propagation.

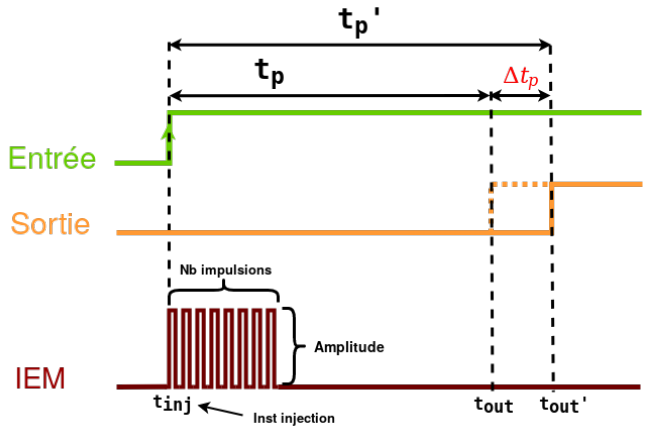


FIGURE 3.2 – Chronogramme de l'effet d'une EMFI sur la propagation d'un signal générant un ralentissement de la propagation.

3.2 Configuration expérimentale

Comme défini précédemment, la chaine de retard est constituée d'éléments combinatoires connectés successivement. Dans le cas des cibles reconfigurables comme les FPGA, une cellule logique est constituée de différents éléments primitifs. comme on cherche à évaluer l'impact sur la logique combinatoire, le choix est porté sur un élément de type *buffer*. En connectant différents *buffers* de différentes cellules de manière à créer une cascade, nous générons alors une chaine de retard qui n'implique que des éléments de la logique combinatoire.

Forcément, pour différents FPGA, le nombre de cellules disponibles est différent d'une technologie à une autre ainsi que leur configuration, leur procédé de gravure ou encore les délais liés aux interconnexions. Cette différence engendre forcément un délai de propagation différent pour chaque technologie. En effet, le délai de propagation d'un seul *buffer* dépend de la technologie de la cible, ce qui signifie que la durée du délai de propagation de toute la cascade est un multiple du nombre de ces *buffers*. Aussi, pour un même nombre de cellules formant la chaine de retard, le délai de propagation nominale sera différent pour chaque technologie. Par souci de généralité, certaines

caractérisations ont donc été réalisées sur trois cartes de développement intégrant des FPGA différents :

- Microsemi Maker Board [95] : Microsemi SmartFusion2 M2S010 [96].
- Sasebo-G [97] : Xilinx Virtex-II Pro XC2VP30 [98].
- Sasebo-W [99] : Xilinx Spartan-6 XC6SLX150 [100].

Le Microsemi SmartFusion2 étant fabriqué avec un procédé de gravure à 65 nm, le Xilinx Virtex-II Pro de procédé 90 nm et le Xilinx Spartan-6 à 45 nm. Pour chaque cible, nous avons programmé un nombre défini de cellules pour la chaine de retard. Les valeurs des délais ainsi que le nombre de cellules programmées pour chaque FPGA sont rapportées dans le Tableau 3.1.

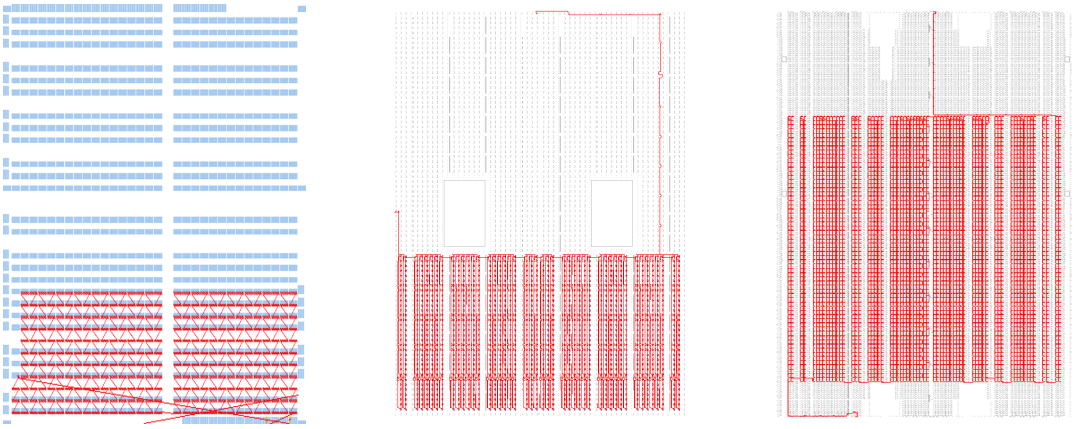
TABLEAU 3.1 – Paramètres de la cascade de *buffers* selon la cible FPGA

FPGA	**Technologie (nm)**	**#Buffers**	t_p **(µs)**
Microsemi SmartFusion2	65	5000	∼0.97
Xilinx Virtex-II Pro	90	5888	∼2.23
Xilinx Spartan-6	45	4096	∼1.78

Lors de la configuration des cibles, nous avons choisi de procéder à un placement manuelle des cellules formant la chaine de retard. Ce choix est motivé par le fait que durant la procédure de placement et routage par defaut, l'outil de développement ne tient pas compte d'un ordre précis ou d'un placement régulier. Comme la caractérisation tient compte du paramètre spacial, il est donc important de connaître l'emplacement de la chaine de retard pour une question de cohérence entre l'impact et la position. Le placement de la chaine (bigDelay) a été groupé sur une partie des cellules logiques disponibles du FPGA comme indiqué dans les plans de masse de la Figure 3.3. Nous pouvons ainsi imaginer des expérimentations dans un second temps, où il est question de générer d'autres plans de masse et de vérifier s'il existe une corrélation dans l'impact entre le placement de la chaîne combinatoire et la position de la sonde d'injection.

Pour effectuer la mesure de délai, un oscilloscope est utilisé dans ce sens. Tout d'abord, nous utilisons un générateur de signal pour fournir sur un port d'entrée/sortie du FPGA le signal à propager. Ce même signal est ainsi observé sur une entrée de l'oscilloscope. Á partir de l'instant t_{in} de son entrée dans le premier *buffer*, le signal va donc mettre un certain temps pour se propager dans la cascade de *buffers* jusqu'à l'instant t_{out} à la sortie du dernier *buffer*. Cette sortie est programmée pour être délivrée sur un autre port E/S du FPGA et observée sur une deuxième entrée de l'oscilloscope comme présenté dans la Figure 3.4.

Une fonction de calcul de délai est configurée dans l'oscilloscope pour évaluer la valeur de $t'_p(x, y)$ qui est calculée pour chaque test. À noter qu'avec le générateur de signal, on configure un second signal TRG_{gen} directement connecté au générateur d'impulsion, et utilisé comme signal de déclenchement pour générer les EMFI. TRG_{gen} est configuré de sorte qu'il est générer bien avant la mise à l'état haut du signale à propager



(a) Microsemi SmartFusion2 (b) Xilinx Virtex-II Pro (c) Xilinx Spartan-6

FIGURE 3.3 – Plan de masse de la cascade de *buffers*, selon la cible FPGA.

de sorte que les EMFI puissent être générées bien avant t_{in} et après t_{out}.

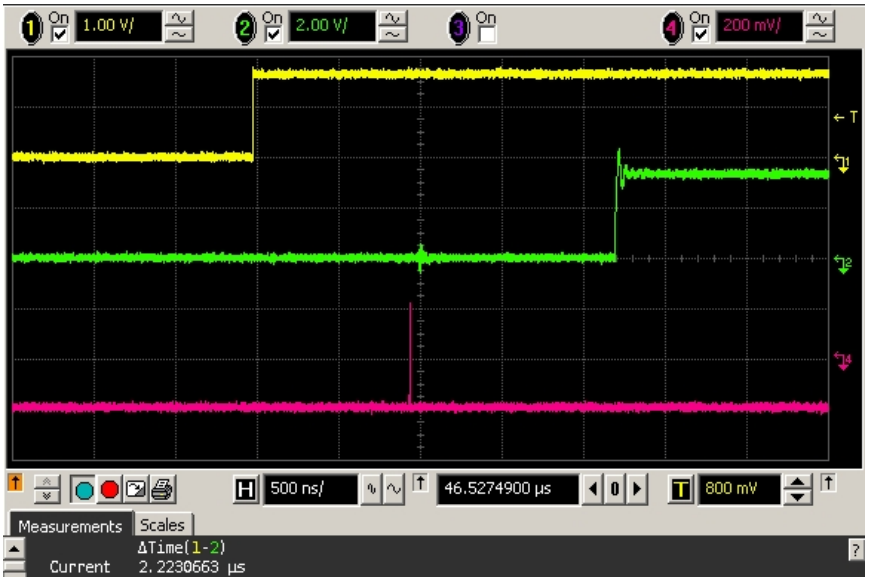


FIGURE 3.4 – Observation des signaux sur oscilloscope montrant le signale d'entrée en couleur jaune, le signal de sortie en vert récupéré depuis le Xilinx Virtex-II Pro et une impulsion EM en magenta générée durant le temps de propagation.

La configuration expérimentale pour tous les tests est représentée dans la Figure 3.5. un générateur d'impulsions capable de générer des impulsions avec une largeur de 1.5 ns est utilisé comme source des impulsions EM. Le choix de ce générateur est motivé par le fait qu'il peut générer plusieurs impulsions successives à une fréquence qui peut atteindre 330 MHz. Pour tous les tests, l'impulsion EM est toujours configurée avec une largeur de 1.5 ns avec un temps de montée et de déclin de 1 ns. Coté puissance, pour une amplitude maximale qu'on peut configurer de 0 dBm, la sortie du générateur est routée vers un amplificateur dont la bande passante s'étale de 10 kHz à 400 MHz et délivrant une puissance de sortie maximale de 260 W. Une sonde magnétique est alors connectée à la sortie de l'amplificateur et peut être déplacée sur la surface du boîtier du FPGA à l'aide d'un système de positionnement à trois axes.

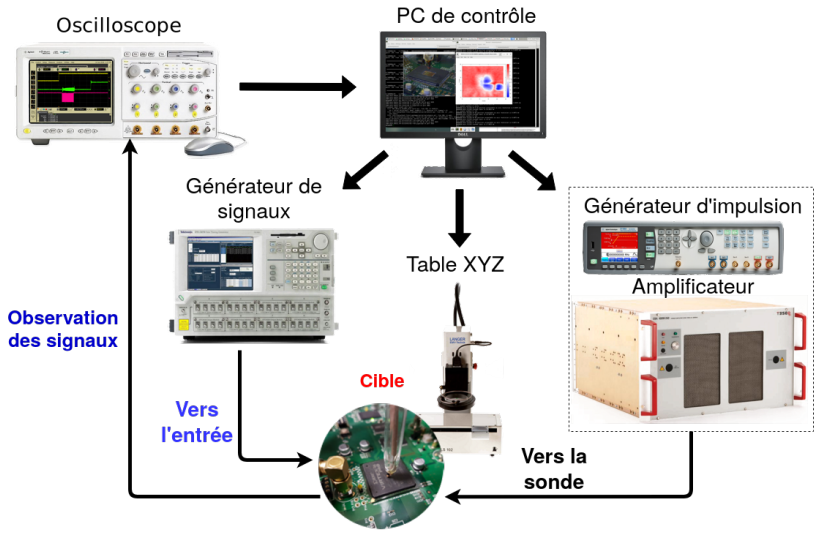


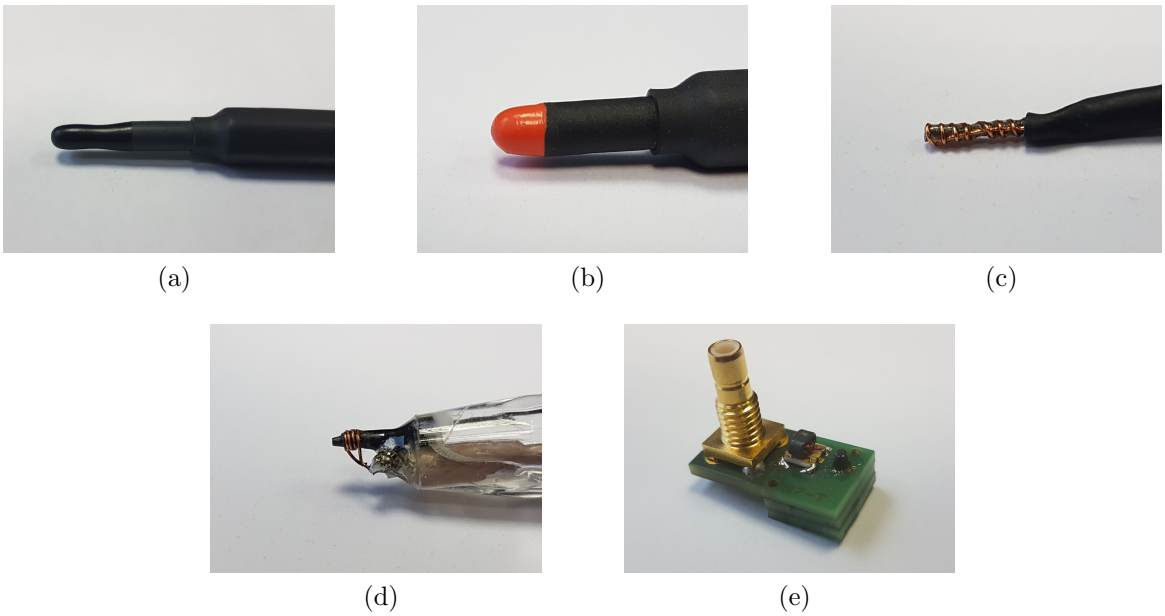
FIGURE 3.5 – Diagramme de la plateforme EMFI avec générateur Keysight pour les expérimentations sur les FPGA.

Différentes caractérisations sont procédés en utilisant cette plateforme pour l'étude de l'effet des paramètres spatial, électrique et temporel. La variation de la position de la sonde, l'amplitude et la polarité de l'impulsion, le nombre d'impulsion ou encore l'instant d'injection sont les principaux paramètres qu'on propose d'évaluer l'influence sur la logique combinatoire. Les résultats de l'effet du paramètre spatial, qui est représenté par la position (x, y) de la sonde au dessus du le boîtier de la cible, sont présentés sous forme de cartographie décrivant le niveau d'impact engendré par lesEMFI en fonction d'un des autres paramètres à évaluer.

Le déroulement détaillé d'une séquence de test est comme suit :

1. **Configuration de la campagne de test :** Réinitialisation de tous les équipements et configuration de la cible.
2. **Exécution sans injection de faute :** 10 itérations de la séquence suivante sont produites :
   - Génération d'un signal à partir du générateur de signaux.
   - Récupération de t_p depuis l'oscilloscope.
   - Calcul de la moyenne t_p.
3. **Exécution avec injection de faute** :
   - configuration des paramètres d'injection (position de la sonde, amplitude et polarité de l'impulsion, instant d'injection et nombre d'impulsion). Pour chaque ensemble de paramètres d'injection fixés, 10 itérations de la séquence suivante sont produites :
     - Génération d'un signal à partir du générateur de signaux. (TRG_{gen} est mis à l'état haut)
     - Récupération de t'_p depuis l'oscilloscope.
   - Calcul de la moyenne $\Delta t_p(x, y)$.
4. **Sauvegarde des résultats**

3.3 Sondes d'injection magnétique



(a) (b) (c)

(d) (e)

FIGURE 3.6 – Photographies des sondes d'injection (a) Langer RF-B 3-2, (b) Langer BS 05DB-h, (c) LIRMM F, (d) Arelis N1 et (e) Arelis S7-T.

Outre l'étude de l'impact des paramètres d'injection énoncée précédemment, nous proposons l'étude du rendement de cinq sondes d'injection magnétiques sur les trois cibles rapportées dans le Tableau 3.1. L'injecteur étant lui même considéré comme paramètre d'injection du fait de ses propriétés. Les analyses issues de l'état de l'art montrent déjà l'importance de ces propriétés dans la génération d'un meilleur rendement. Le parfait exemple est celui de [39] et [40] où les résultats par simulation du premier donnent différentes observations que dans un cas réel dans le deuxième par rapport à la conception d'une sonde pour un meilleur rendement. Les paramètres mécaniques des sondes qui sont évaluées sont présentés dans le Tableau 3.2.

TABLEAU 3.2 – Paramètres techniques des sondes d'injection magnétique

Référence de sonde	Φ_n **(mm)**	Φ_f **(µm)**	**#Tours**	**Noyau**
Langer RF-B 3-2 [101]	4	80	2	Air
Langer BS 05DB-h [102]	2	100	7	Air
LIRMM F [40]	0.75	200	7	Fe
Arelis N1	0.80	150	4	Fe
Arelis S7-T	0.75	200	4	Fe

Langer RF-B 3-2 [101] (fig. 3.6a) est une sonde commerciale, conçue pour la mesure des émissions de Circuit Imprimé (PCB) dans la gamme de fréquences allant de 30 MHz à 3 GHz. Comme il s'agit d'un composant passif, nous pouvons le faire fonctionner de manière inverse, donc en tant que sonde d'injection pour EMFI. La fiche technique de la sonde indique un diamètre de tête Φ_n de 4 mm, et d'après une imagerie par rayons X, il s'agit d'une sonde à noyau d'air constituée de deux spires dont le fil est de diamètre

Φ_f approximatif de 80 µm.

Langer BS 05DB-h [102] (fig. 3.6b) est également une sonde commerciale, mais dédiée à l'immunité de PCB (source de champ magnétique) et conçue pour résister à la haute tension allant jusqu'à 4.4 kV. L'imagerie par rayons X révèle qu'il s'agit également d'une sonde à noyau d'air, conçue avec sept spires d'un fil de diamètre 100 µm et formant une bobine de diamètre 2 mm.

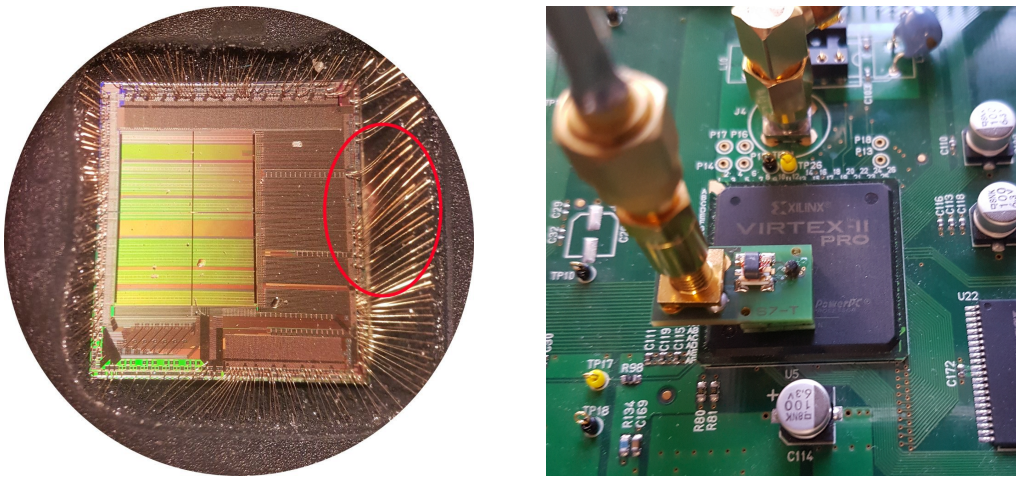
La sonde LIRMM F (fig. 3.6c) est une sonde à tête plate *fait-maison* et décrite dans [40, §2.2]. Elle est composée de sept spires dont le fil est de 200 µm autour d'un noyau de ferrite de 0.75 mm.

Les deux dernières sondes sont des prototypes développés par la société française Arelis, construites à partir d'un noyau de ferrite de la forme d'un cône tronqué circulaire. Le diamètre supérieur est de 1.5 mm alors que le diamètre inférieur est égal de 0.80 mm pour Arelis N1 (fig. 3.6d) et de 0.75 mm pour Arelis S7-T (fig. 3.6e). Arelis N1 est une sonde *fait-maison*, composée de quatre spires avec un fil de 150 µm.

Arelis S7-T (fig. 3.6e) est la version industrielle de Arelis N1. L'enroulement de la bobine est fabriqué en superposant trois couches fine de PCB, avec un chemin de cuivre de largeur 200 µm. De plus, pour améliorer l'adaptation d'impédance, un transformateur RF avec un rapport d'impédance de 2 :1 a été ajouté.

Sachant que les paramètres d'injection sont fixes durant cette partie des expérimentations, l'évaluation nous permet donc de dresser une relation entre les propriétés de conception des sondes par rapport au niveau de l'impact induit.

3.4 Impact des sondes magnétiques



(a) (b)

FIGURE 3.7 – Photo (a) d'un FPGA Microsemi SmartFusion2 décapsulé montrant les fils de bonding autour du circuit et (b) la sonde Arelis S7-T au dessus du FPGA Xilinx Virtex-II Pro.

L'EMFI de par sa technique, permet d'injecter une perturbation sur une cible sui-

vant la position spatiale de la sonde d'injection. Ainsi, la résolution spatiale et la densité de l'impact sont fortement liées aux aspects mécaniques de la sonde d'injection magnétique. Pour valider ce point, nous avons procédé à une analyse comparative de l'impact généré à travers différentes sondes d'injection.

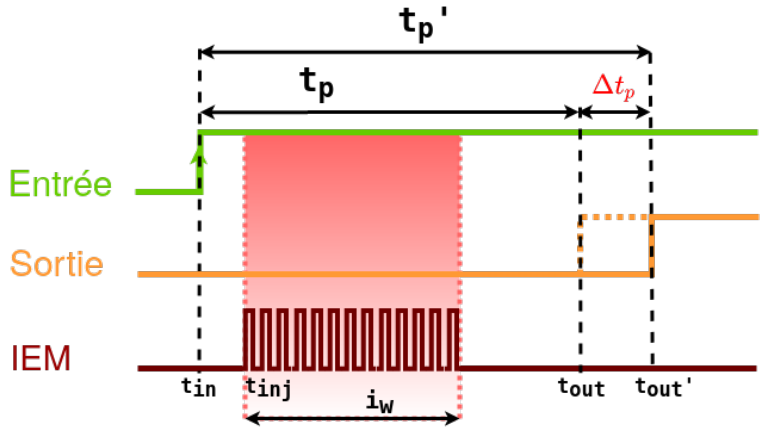


FIGURE 3.8 – Injection de plusieurs impulsions successives durant le temps de propagation d'un signal dans la logique combinatoire.

Les résultats expérimentaux présentés dans cette partie correspondent à la distribution spatiale de Δt_p pour différentes cibles FPGA et pour chaque sonde d'injection. Ces résultats sont sous forme de cartographies générées de façon à mettre en évidence le type d'effet engendré sur la propagation qui est définie par le signe de Δt_p : Si la valeur est positive, on a un effet de ralentissement représenté par la couleur (*en rouge*) et s'il est négative alors le rayonnement à induit un effet d'accélération représenté par la couleur (*en bleu*).

Les boîtiers des cibles sont de différentes dimensions, à savoir 19.4 mm × 19.4 mm pour Microsemi SmartFusion2 (fig. 3.9), 24.0 mm × 24.0 mm pour Xilinx Virtex-II Pro (fig. 3.10) et 18.0 mm × 18.0 mm pour Xilinx Spartan-6 (fig. 3.11).

De point de vue efficacité, la meilleure sonde devrait induire un impact Δt_p maximal dans sa valeur absolu. Plus précisément, l'effet doit être induit sur les portes logiques transportant des données sensibles, car d'autres peuvent appartenir à certains détecteurs EMFI. En d'autres termes, la sonde doit avoir une résolution spatiale élevée.

Comme l'instant de sortie du signal t'_{out} n'est pas connu d'avance sous effet d'un rayonnement EM, nous avons choisis de procéder aux injections durant une fenêtre de temps i_w (fig. 3.8) qui couvre une bonne partie du temps de propagation nominale. Aussi, sachant que ce temps est différent pour chaque cible, plusieurs injection successives sont générées pour couvrir la fenêtre de tir propre à chaque cible : i_w est égale à 375 ns pour Microsemi SmartFusion2, 1200 ns pour Xilinx Virtex-II Pro et 1020 ns pour Xilinx Spartan-6. Les impulsion sont configurées avec une largeur de 1.5 ns et distancées de 3 ns avec une amplitude à 0 dBm et une polarité positive.

Suite au tests sur Microsemi SmartFusion2, la sonde correspondant le mieux à ces critères est LIRMM F (fig. 3.9c), avec un impact positif maximal de 3 ns. Sa distri-

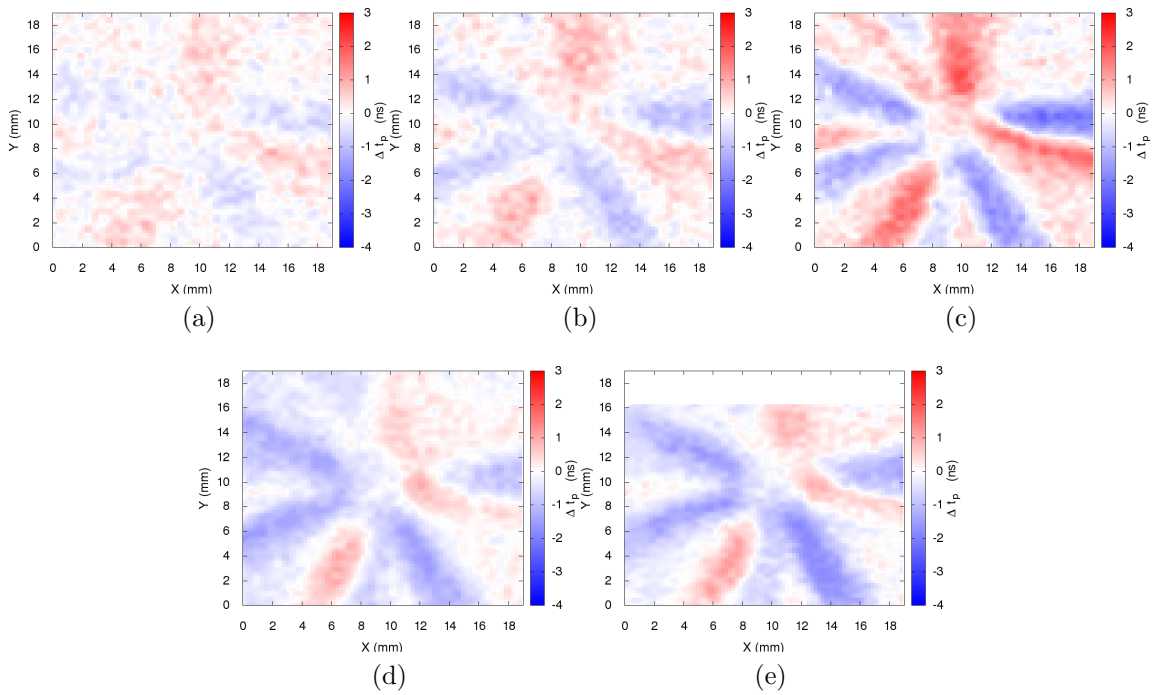


FIGURE 3.9 – Distribution spatiale de Δt_p pour Microsemi SmartFusion2, suivant les sondes d'injection magnétique (a) Langer RF-B 3-2, (b) Langer BS 05DB-h, (c) LIRMM F, (d) Arelis N1 et (e) Arelis S7-T.

bution spatiale contient de grandes zones car la sonde est couplée à des éléments de taille beaucoup plus grande que sa résolution spatiale. Le moyen approprié d'estimer cette dernière consiste à se concentrer sur la netteté de la distribution spatiale, c'est-à-dire sur la largeur de la bordure (généralement de couleur blanche) d'une zone ayant un impact donné sur une autre. Les sondes Arelis (figs. 3.9d and 3.9e) arrivent à la deuxième place, avec une netteté identique mais un impact moindre. Enfin les sondes commerciales de Langer (figs. 3.9a and 3.9b), avec un impact quasiment faible et dont les limites des zones d'impact sont floues.

Ce classement est assez cohérent avec les paramètres mécaniques des sondes (Tableau 3.2) : Les sondes LIRMM F et Arelis offrent une meilleure résolution spatiale par rapport aux sondes Langer grâce à leur noyau en ferrite qui a un diamètre plus petit, une perméabilité magnétique supérieure et, par conséquent, une densité de flux supérieure. Et comme LIRMM F a un enroulement avec presque deux fois plus de tours que celui des sondes Arelis, il a également un meilleur impact.

Quelle que soit la sonde, d'un point de vue macroscopique, toutes les distributions spatiales (fig. 3.9) suivent une forme d'étoile identique : Au centre, l'impact est très faible et tout autour, de manière circulaire, il alterne les zones de ralentissement (*en rouge*) puis d'accélération (*en bleu*). D'un autre côté, les portes logiques sont implémentées dans la puce en silicium du FPGA, qui est situé au centre du paquet.Nous concluons donc que les EMFI n'ont pas d'impact direct sur le coeur de cet FPGA, mais plutôt sur les fils de *bonding* (fig. 3.7a), ou encore ceux des rails d'alimentation.

Pour la cible Xilinx Virtex-II Pro, l'impact positif maximal est égal à 10 ns et est atteint en utilisant Langer BS 05DB-h (fig. 3.10b). Il est trois fois plus important que

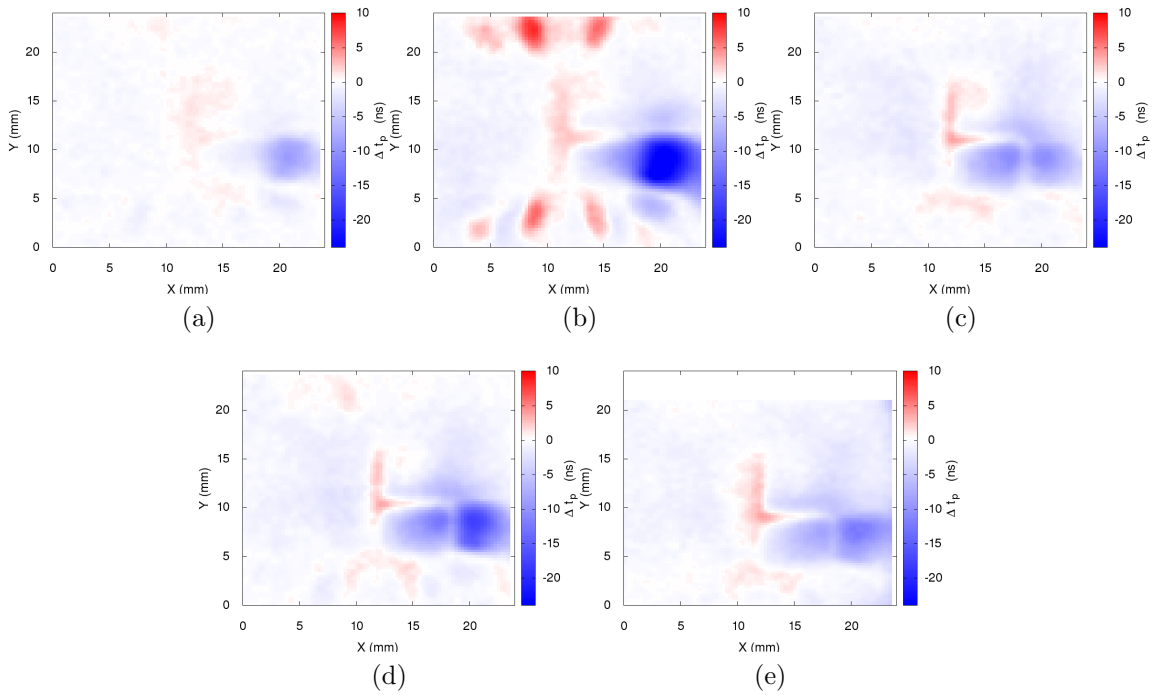
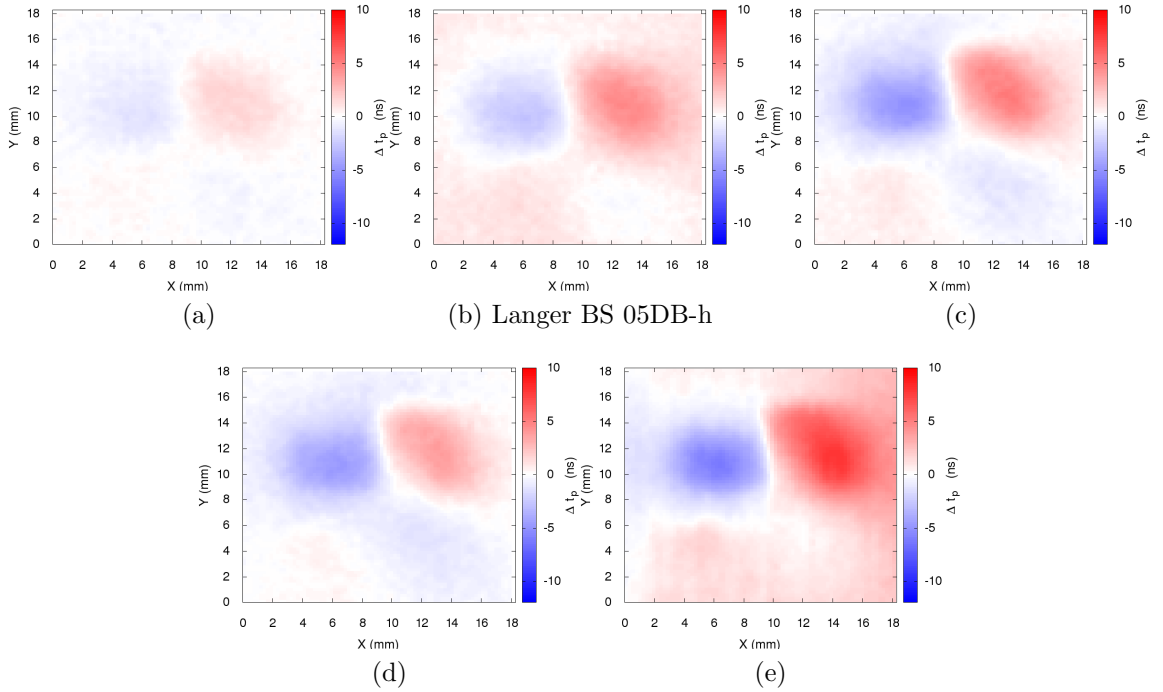


FIGURE 3.10 – Distribution spatiale de Δt_p pour Xilinx Virtex-II Pro, suivant les sondes d'injection magnétique (a) Langer RF-B 3-2, (b) Langer BS 05DB-h, (c) LIRMM F, (d) Arelis N1 et (e) Arelis S7-T.

l'impact sur Microsemi SmartFusion2. Nous supposons que cela est dû à la technologie de gravure qui est de 90 nm pour Xilinx Virtex-II Pro contre 65 nm pour Microsemi SmartFusion2. La méthode de stratification de chaque technologie doit également être prise en compte. Comme précédemment, les zones de ralentissement (*en rouge*) situées en haut et en bas de la distribution sont certainement dues à un couplage avec des fils de *bonding*. Si nous cherchons à avoir un impact uniquement sur le noyau de FPGA, c'est-à-dire au centre, LIRMM F reste la meilleure sonde avec une résolution spatiale plus précise.

Enfin, pour Xilinx Spartan-6, 10 ns d'impact positif maximal est possible avec Arelis S7-T (fig. 3.11e). Néanmoins, les résultats pour LIRMM F, la meilleure sonde durant ces tests, sont très similaires (fig. 3.11c). Les effets d'accélération et de ralentissement sont observés sur une zone centrale du boîtier, où on suppose l'emplacement du silicium du CI. Contrairement aux précédentes caractérisations, il n'y a pas de couplage visible avec les fils de *bonding*. Ils doivent être absents de la conception du boîtier (boîtier BGA [103] qui doit utiliser la technologie dit *flip-chip* [104]).

On note que le test du balayage avec Arelis S7-T n'est pas complet. Cela est dû à la forme géométrique de la sonde qui ne peut accéder à certaines positions faute d'être bloqué par les composants qui sont tout autour du boîtier du FPGA. Sur la fig. 3.7b on retrouve ce cas durant le test sur Xilinx Virtex-II Pro. Bien que les dimensions du boîtier sont assez grandes pour pouvoir réaliser le test sur le maximum de la surface, cela peut certainement être bloquant dans le cas où le boîtier est plus petit.



(a) (b) Langer BS 05DB-h (c)

(d) (e)

FIGURE 3.11 – Distribution spatiale de Δt_p pour Xilinx Spartan-6, suivant les sondes d'injection magnétique (a) Langer RF-B 3-2, (b) Langer BS 05DB-h, (c) LIRMM F, (d) Arelis N1 et (e) Arelis S7-T.

3.5 Impact des paramètres d'injection

Le challenge d'induire une faute dans un circuit repose sur l'ensemble des paramètres et configurations à mettre en place. Une étude de la variation de ces paramètres est donc nécessaire pour déterminer les seuils de sensibilité d'une cible par rapport à ces paramètres et qui amène à engendrer un impact maximal.

Dans cette partie, nous proposons l'étude des paramètres d'injection à savoir, l'instant d'injection, le nombre d'impulsions, leur amplitude et polarité. Les résultats expérimentaux résultant de cette étude correspondent à la distribution spatiale de Δt_p sur la cible Xilinx Virtex-II Pro en utilisant la sonde Arelis N1. Le choix de la cible est motivé par la disponibilité d'une autre carte de remplacement. De même pour la sonde avec la disponibilité d'autres de conception identique.

Pour une position (x, y) de la sonde, le délai de propagation, avec ou sans EMFI, est évalué en tant que moyenne arithmétique de dix mesures. La cible représente une surface carrée de 24.0 mm × 24.0 mm et la cartographie de toutes les positions (x, y) de la sonde est obtenu suivant 40×40 positions à une distance de la surface de FPGA de 50 µm sur l'axe-z.

L'injection de seulement *une* impulsion n'a pas engendré de variation majeure de $\Delta t_p(x, y)$. Nous avons donc reconfiguré les paramètres EMFI, comme présenté au chronogramme de la Figure 3.8, avec un *burst* de 100 impulsions successives distantes de 3 ns et une amplitude de 0 dBm et une polarité positive. L'instant de début de l'injection t_{inj} coïncide avec le front montant de l'entrée t_{in}.

La distribution spatiale de Δt_p suite à l'injection de 100 impulsions est présentée

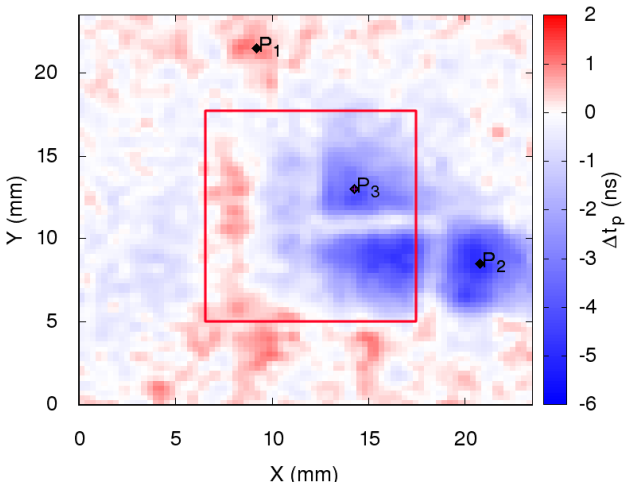


FIGURE 3.12 – Distribution spatiale de Δt_p suite à l'injection de 100 impulsions.

dans la Figure 3.12. Le résultat de la cartographie montre comme vu précédemment que l'impact $\Delta t_p(x, y)$ peut être positif (**ralentissement**) en rouge, ou négatif (**accélération**) avec la couleur bleue. Au moins trois zones d'impact séparées, décrivant un effet d'accélération, peuvent être clairement identifiées par leurs formes délimitées.

Par souci de clarté, trois positions distinctes seront considérées dans la suite de la caractérisation, comme indiqué par la Figure 3.12. Nous allons noter par **P1** la position où le délai de propagation est ralenti à son maximum, et par **P2** une position où ce délai est accéléré à son maximum. **P3** est une position de la zone centrale du FPGA (carré en rouge sur fig. 3.12) qui délimite le possible emplacement du silicium.

3.5.1 Instant d'injection

Figure 3.13 montre l'impact Δt_p lorsqu'un *burst* d'impulsions successives est injectée pendant trois instants t_{inj} différents choisis dans une fenêtre d'injection i_w égale au délai de propagation t_p . La configuration du *burst* reste inchangée avec 100 impulsions successives distantes de 3 ns, tandis que l'amplitude est à −6 dBm.

Quel que soit l'instant d'injection t_{inj}, tous les résultats indiquent un impact positif maximal de 2 ns. Cependant, un impact négatif maximal de −12 ns est observé lorsque l'instant d'injection $t_{inj} = \frac{i_w}{2}$ selon la Figure 3.13b. Il s'agit essentiellement du double de l'accélération générée lorsque $t_{inj} = t_{in}$ (fig. 3.13a). Quand l'injection est générée durant la fin de la fenêtre d'injection (entre $\frac{i_w}{2}$ et t_{out}), la distribution spatiale de l'impact obtenu est différente des valeurs précédentes de t_{inj} et présente des zones moins délimitées (fig. 3.13c).

Pour une analyse plus détaillée de ce comportement, nous concentrons les tests sur les trois positions P1, P2 et P3. La fenêtre d'injection i_w est étendue pour couvrir le temps avant et après le délai de propagation t_p, comme décrit dans la Figure 3.14, avec 3 µs avant t_{in} et 950 ns après t_{out}. L'injection des 100 impulsions est produite suivant la variation de t_{inj} avec un pas configuré à 100 ns.

Sur les trois positions, l'impact maximal est observé lors d'une fenêtre de sensibilité S_w, tel que identifiée dans la Figure 3.15. Cette fenêtre semble avoir la même durée que

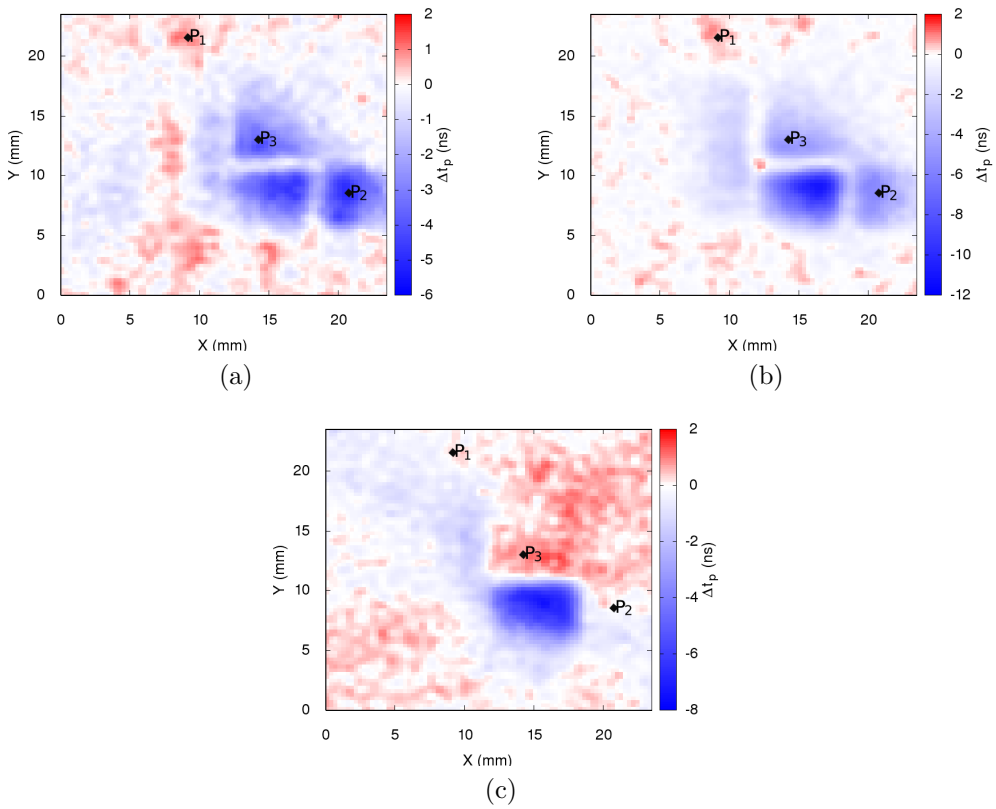


FIGURE 3.13 – Distribution spatiale de Δt_p, avec 100 impulsions injectées (a) au début ($t_{inj} = t_{in}$) du calcul, (b) au milieu ($t_{inj} = \frac{i_w}{2}$) et (c) vers la fin ($\frac{i_w}{2} < t_{inj} < t_{out}$).

le délai de propagation nominale t_p. En dehors de cette fenêtre, l'impact est minimal.

Pour valider cette hypothèse, nous avons implémenté un nouveau design bigDelay double de la chaine de retard avec deux fois le nombre de *buffers* (11 776 de blocs logiques) afin de doubler le délai de propagation. La mesure de ce nouveau délai nominal t_p est de 4.57 µs.

Comme on peut le constater avec la Figure 3.16, les tests effectués aux positions P2 et P3 confirment que la fenêtre d'impact est à son tour doublée et que l'impact suit le même comportement que pour le premier design bigDelay. On notera que ces résultats suivent la logique qu'aucun impact sur la sortie n'est observé avant que l'entrée ne passe à l'état haut logique. De la même manière, les EMFI qui interviennent au delà de la sortie, n'a plus d'impact car la porte logique a déjà mis à jour sa sortie.

3.5.2 Nombre d'impulsions

Suite au précédent résultat, nous avons testé l'effet de la variation du nombre d'impulsions. Figure 3.17 montre l'impact Δt_p pour différents nombres d'impulsions successives injectées (100, 350 et 650 impulsions). Le *burst* est configuré avec des impulsions successives distantes de 3 ns et l'amplitude des impulsions est configurée à −6 dBm.

À partir du résultat des cartographie résultantes, l'impact maximum est observé lors de l'injection de 650 impulsions (fig. 3.17c), avec une valeur maximale de 10 ns comme impact positif et −25 ns pour l'impact négatif. A partir de 350 impulsions (fig. 3.17b),

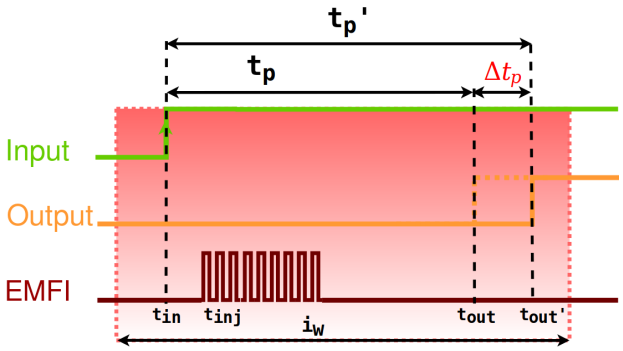


FIGURE 3.14 – Diagramme d'injection d'un *burst* de 100 impulsions durant la fenêtre d'injection $i_w > t_p$.

nous pouvons distinguer l'effet du couplage avec les fils de bonding du FPGA. Nous observons également que la zone centrale de FPGA de fig. 3.17a à fig. 3.17c commute en polarité d'impact lorsque nous atteignons un certain nombre d'impulsions.

Nous avons étudié ce comportement en vérifiant l'impact Δt_p (fig. 3.18) sur les positions P1, P2 et P3 lorsque le nombre d'impulsions varie de 1 à 650 impulsions avec un pas de 5 impulsions.

Il y a une nette évolution linéaire de l'impact avec l'augmentation du nombre d'impulsions pour les positions P1 (fig. 3.18a) et P2 (fig. 3.18b). Pour ces positions, nous observons une courte période de saturation lorsque nous sommes proches du nombre maximal d'impulsions injectées. Cela peut être expliqué du fait qu'avec l'augmentation du nombre d'impulsions le temps on est de plus en plus proche de l'instant de sortie nominale t_{out}. Au-dessus de 600 impulsions, il n'y a plus d'impact de plus induit sur le FPGA. Lorsque la sonde est au-dessus du silicium de FPGA (à la position P3), l'impact suit un comportement différent. La Figure 3.18c montre une augmentation linéaire de l'impact jusqu'à un nombre de 400 d'impulsions, suit alors un effet linéaire opposé de l'impact pour atteindre Δt_p près de 0 ns.

Le même test a été répété utilisant le design bigDelay double (fig. 3.19). Les résultats confirment que nous avons toujours une similitude du comportement de CI sous EMFI. La conclusion majeure de ces résultats est que l'augmentation du nombre d'impulsions ne génère pas le même effet quand la sonde d'injection est positionnée au dessus du silicium de FPGA ou sur les bords de son boîtier.

3.5.3 Amplitude et polarité de l'impulsion

Comme pour les autres paramètres, nous analysons cette fois l'effet de la variation de l'amplitude sur le délai de propagation. Figure 3.20 montre l'impact sur Δt_p lorsque l'amplitude A_{dBm} de la sortie du générateur d'impulsions est configurée à -19 dBm, -12 dBm et -6 dBm. Nous avons fixé le nombre total d'impulsions successives à 650 séparées de 3 ns.

De la Figure 3.20a à la Figure 3.20c, nous pouvons apercevoir une nette évolution de l'impact en terme de valeur et de résolution spatiale. La netteté de la distribution spatiale est également observée avec plus de détails sur les zones touchées par le EMFI,

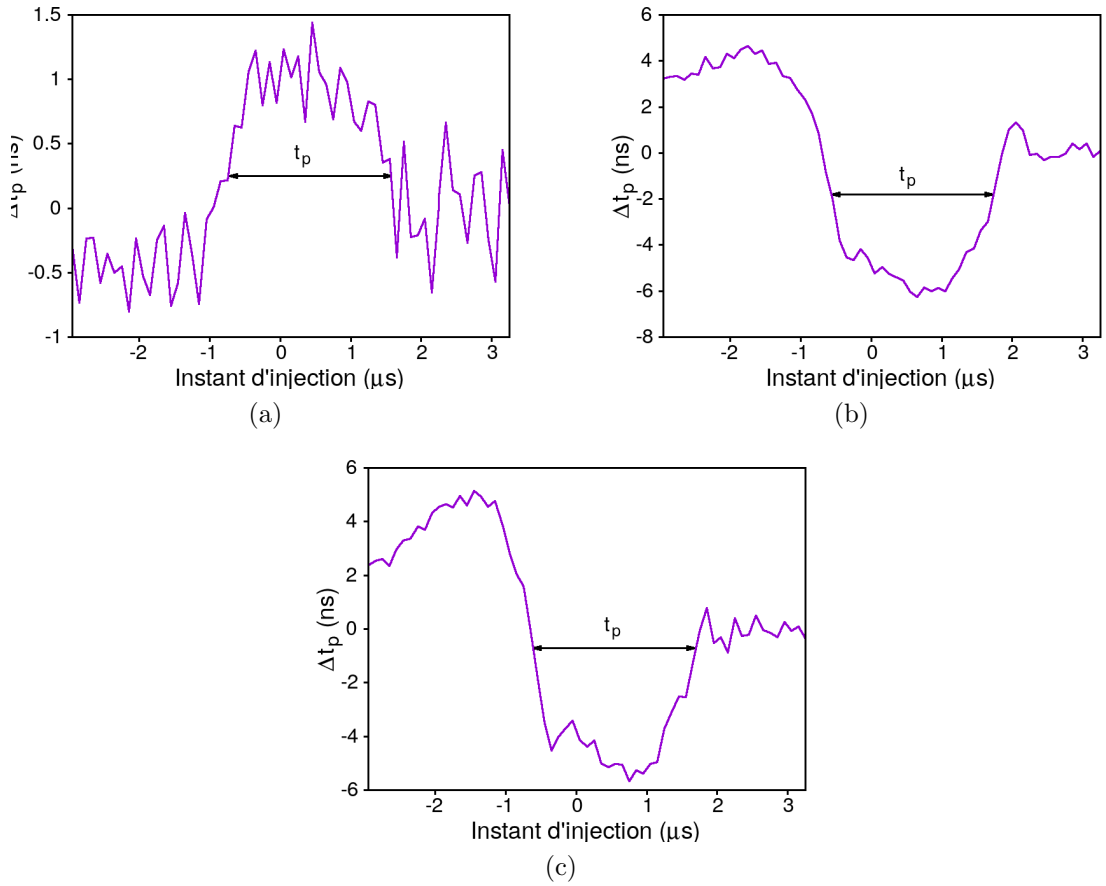


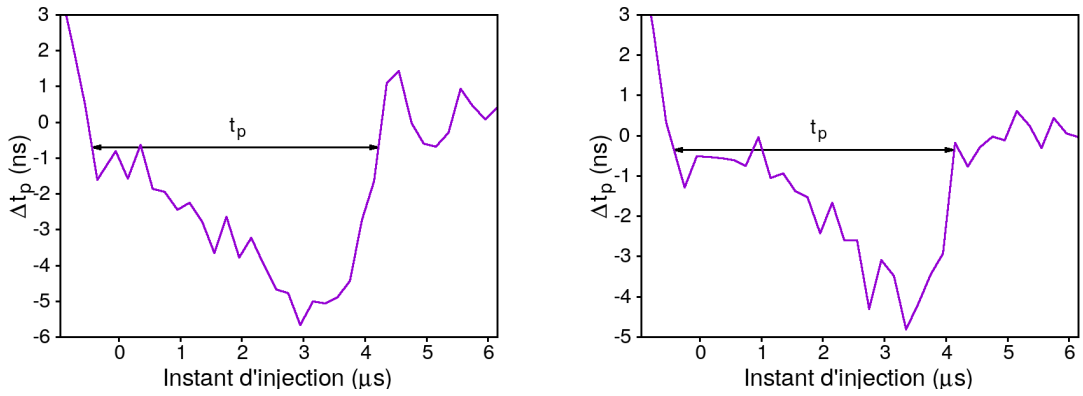
FIGURE 3.15 – Variation de Δt_p pour bigDelay suivant l'instant d'injection t_{inj}, pour les positions (a) P1, (b) P2 et (c) P3.

situé en haut et en bas du boîtier. Cependant, cette distribution est pratiquement la même pour toute les amplitudes testées.

Pour plus de détails, nous analysons la variation de l'amplitude sur les positions sélectionnées P1, P2 et P3 (fig. 3.21). la variation de l'amplitude d'impulsion est configurée entre −19 dBm et 0 dBm avec un pas de 0.25 dBm.

Nous observons que tant que A_{dBm} augmente, t'_p augmente à son tour suivant une fonction linéaire. Il y a même un impact de −6 ns à la valeur minimale de A pour la position P2. A noter qu'il y a une diminution d'impact à partir de −6 dBm to 0 dBm pour les positions P1 (fig. 3.21a) et P2 (fig. 3.21b). Contrairement à ce comportement, celui observé sur la position P3 (fig. 3.21c) suit une augmentation linéaire continue à partir de −19 dBm to 0 dBm. De la même manière que pour les tests avec l'augmentation du nombre d'impulsions, la zone centrale de FPGA indique une sensibilité différente avec l'augmentation de l'amplitude par rapport à celle des autres zones (bordures du boitier).

La dernière caractérisation concerne la polarité de l'impulsion. Figure 3.22 montre l'impact Δt_p dans le cas d'impulsions à polarité positive et négative. Nous avons conservé la configuration des paramètres d'injection avec une amplitude de −6 dBm et un *burst* de 650 impulsions successives séparées de 3 ns. Les deux résultats fig. 3.22a et fig. 3.22b indiquent la même distribution spatiale de l'impact sur les bordures supérieure



(a) (b)

FIGURE 3.16 – Variation de Δt_p pour bigDelay double suivant l'instant d'injection t_{inj}, pour les positions (a) P2 et (b) P3.

et inférieure du boîtier du FPGA. Cela signifie que la commutation de la polarité des impulsions ne semble pas engendrer un impact différent sur les fils de *bonding*. Cependant, certaines zones passent du rouge au bleu et inversement (effet ralentissement vers accélération), en particulier au centre du boîtier où nous supposons l'emplacement de la puce de FPGA. Nous pouvons conclure que la polarité des impulsions peut avoir un impact direct au niveau logique.

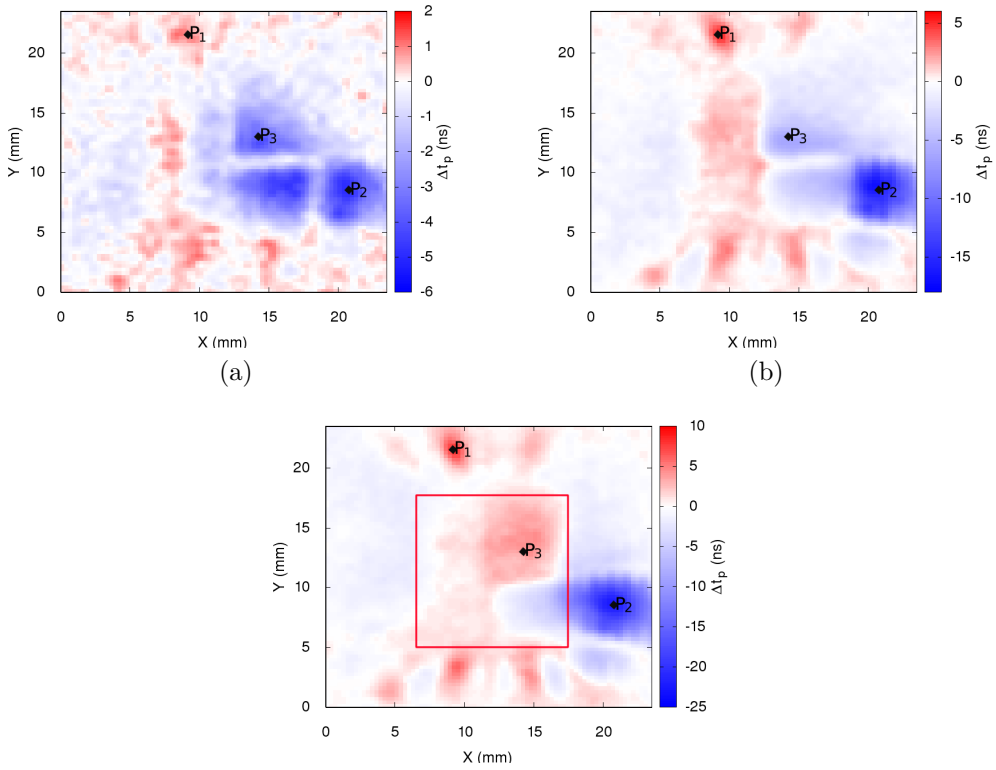


FIGURE 3.17 – Distribution spatiale de Δt_p suite à l'injection de (a) 100, (b) 350 et (c) 650 impulsions.

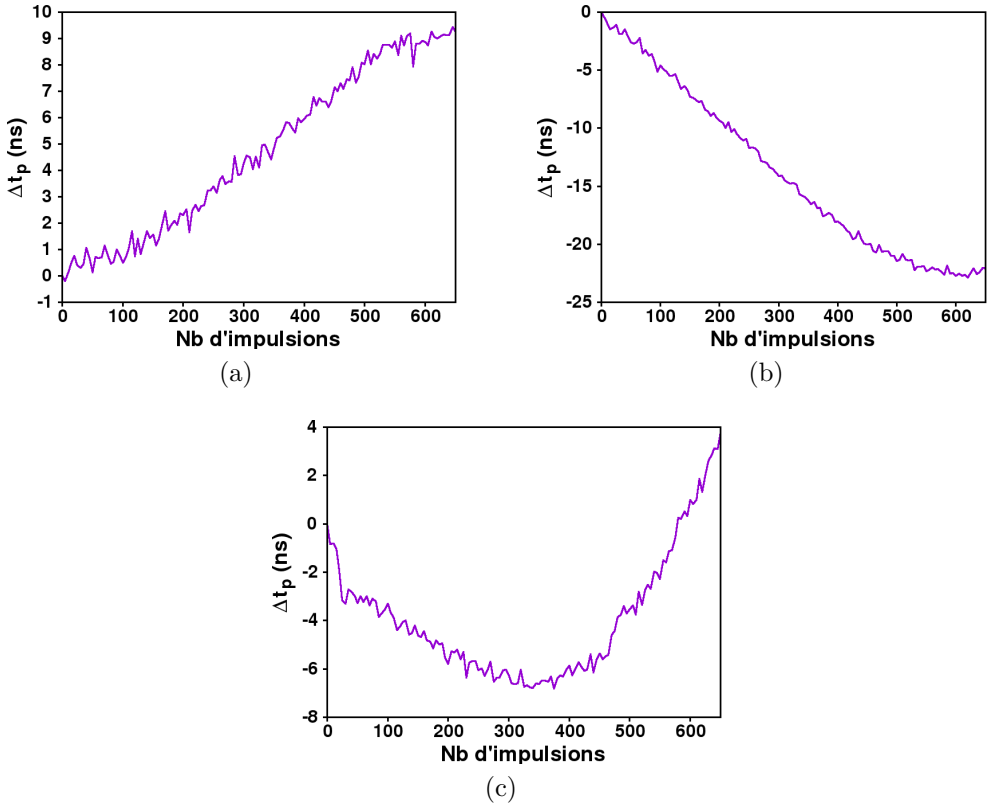


FIGURE 3.18 – Variation de Δt_p en fonction du nombre d'impulsions, sur l'implémentation bigDelay pour les positions (a) P1, (b) P2 et (c) P3.

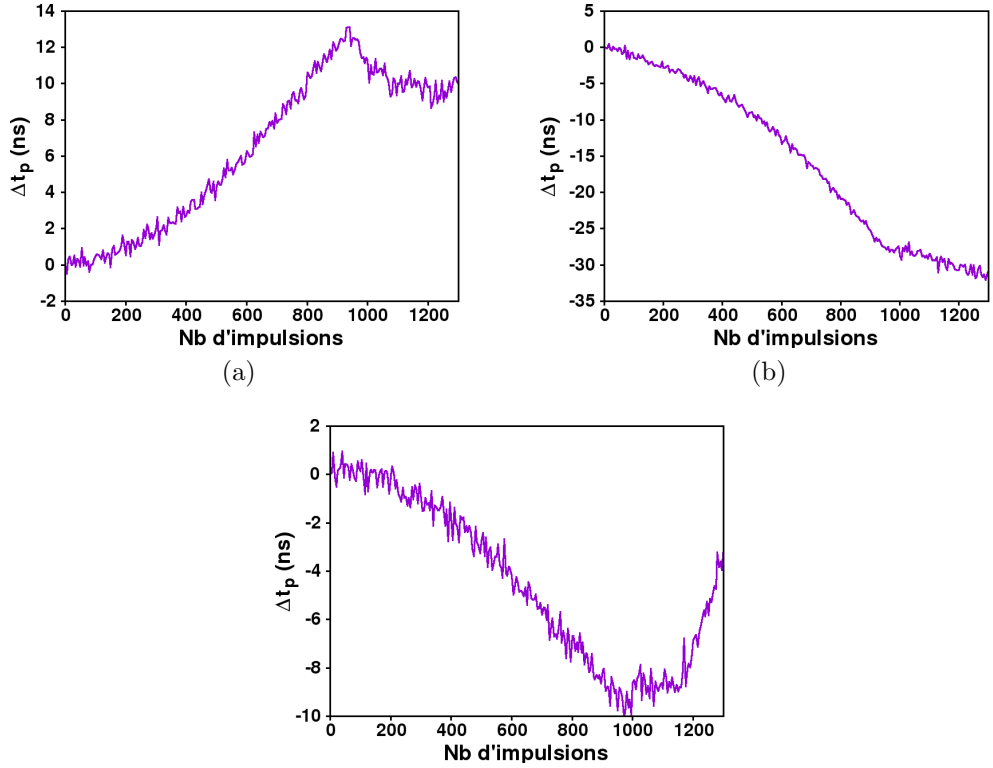


FIGURE 3.19 – Variation de Δt_p en fonction du nombre d'impulsions, sur l'implémentation bigDelay double pour les positions (a) P1, (b) P2 et (c) P3.

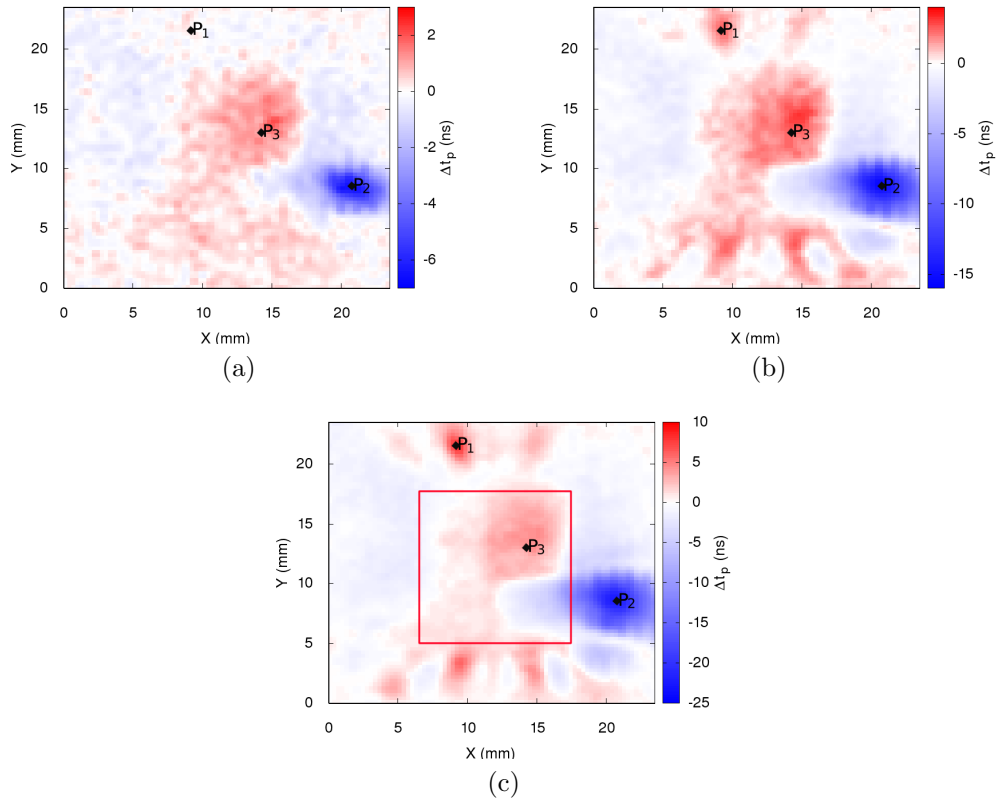


FIGURE 3.20 – Distribution spatiale de Δt_p suite à l'injection de 650 impulsions avec une amplitude configuré à (a) −19 dBm, (b) −12 dBm et (c) −6 dBm.

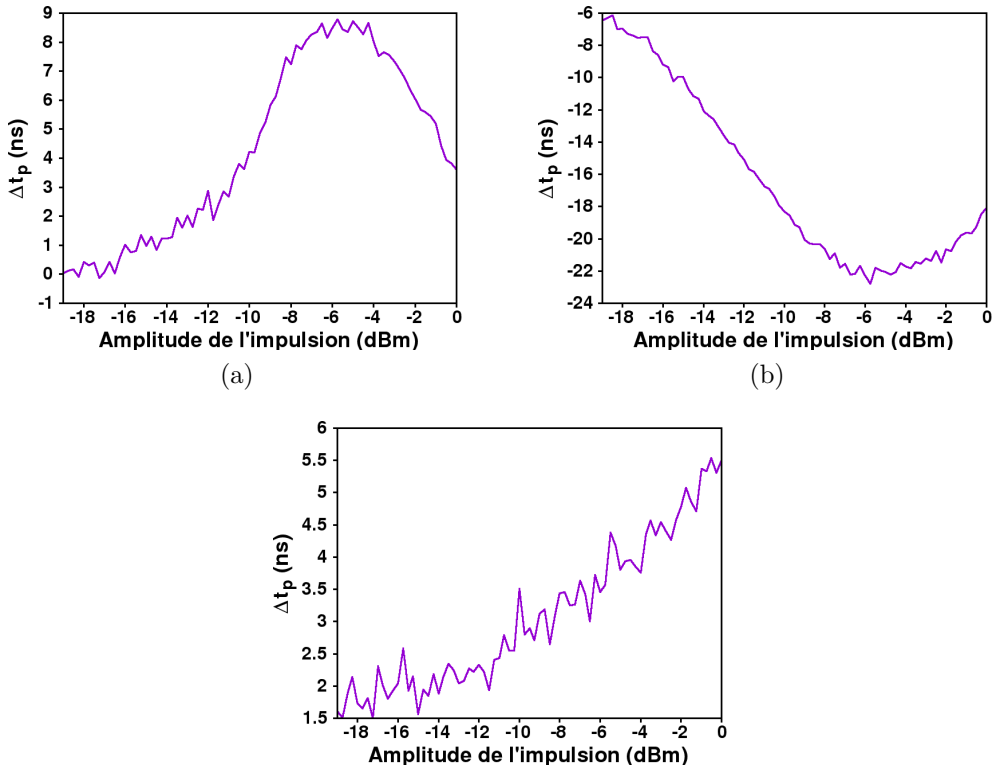


FIGURE 3.21 – Variation de Δt_p en fonction de l'amplitude des impulsions, pour les positions (a) P1, (b) P2 et (c) P3.

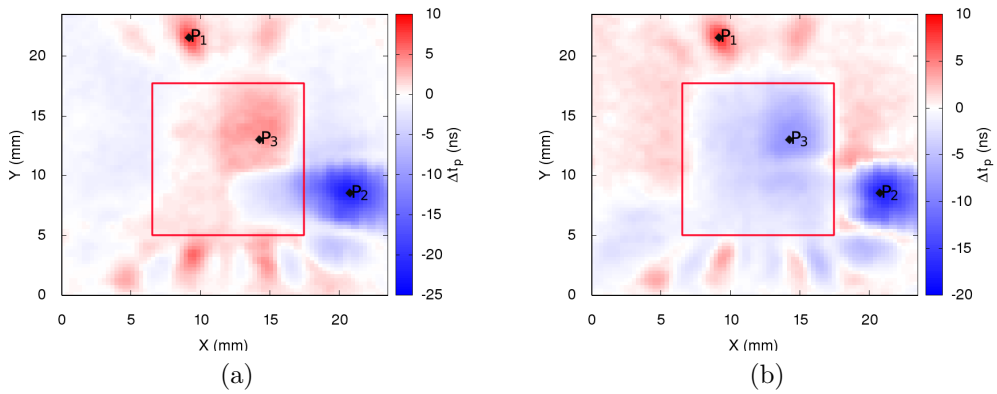


FIGURE 3.22 – Distribution spatiale de Δt_p quand la polarité de l'impulsion est (a) positive et (b) negative.

3.6 Conclusion

A travers une méthode expérimentale, une caractérisation a été réalisé au niveau logique, plus précisément de la logique combinatoire, d'un modèle de faute qui cible le temps de propagation entre cellules logiques d'un FPGA. Une évaluation de sondes magnétique a été présenté sur leur potentiel à créer des fautes à l'intérieur d'un CI par l'augmentation des délais de propagation. En d'autres termes, ceci a permis d'inclure le paramètre sonde dans la caractérisation du couplage électromagnétique.

Les tests ont montrés que des sondes *fait-maison* présentent une meilleur résolution spatial que des sondes commerciales. Le classement des sondes est finalement cohérent avec la théorie de l'électromagnétisme : plus le diamètre de la sonde est proche de la valeur de la technologie de procédé de la cible, plus il y a un gain en résolution dans la distribution spatiale de la faute. les résultats amènent à privilégier les sondes avec un noyau en ferrite, puisque cette propriété permet d'avoir une perméabilité magnétique élevée par rapport aux autres sonde à noyau d'air. Aussi, il s'avère qu'une spire magnétique avec un nombre élevé de tours augmente l'intensité du champ magnétique, et donc l'impact magnétique.

Pour un même FPGA, la distribution spatiale est quasi identique avec l'utilisation des différentes sondes, ce qui signifie que le couplage n'a pas vraiment d'importance. Cependant, les résultats sont différents pour chaque FPGA, principalement dû aux différences de technologies qui implique un couplage avec différents éléments tels que les fils de *bonding*.

La caractérisation a été étendue aux autres paramètres d'injection (instant d'injection, nombre d'impulsion, amplitude et polarité de l'impulsion), dont la variation a permis d'identifier le niveaux à partir du quel un impact sur une cible est effectif. Les résultats montrent qu'il faut plus d'une impulsion (mode *burst*) pour induire un impact significatif sur le délai de propagation de la logique combinatoire. Á noter que cette augmentation du nombre d'impulsions n'impacte pas de la même manière le CI du FPGA et les bords du boîtier (*bonding*). Cependant, la variation de l'intensité du rayonnement généré n'a pas d'incidence sur la distribution spatiale. De manière générale, l'impact suivant la variation des paramètres étudiés est différent selon la positions de la sonde sur le boîtier FPGA.

CHAPITRE 4

Impact du rayonnement électromagnétique sur MCU

Sommaire

Dans ce chapitre, nous proposons une démarche qui permet de caractériser avec précision l'impact EMFI sur les MCUs. Cette démarche est basée sur trois méthodes, dont chacune est dédiée à l'étude d'une vulnérabilité à un niveau particulier. L'application de ces méthodes est validée sur différentes technologies de MCUs. Une partie des résultats qui seront présentés a fait l'objet d'une publication intitulée *Characterization of Electromagnetic Fault Injection on a 32-bit Microcontroller Instruction Buffer* [105].

4.1 Démarche de la caractérisation

Un MCU est constitué d'un ensemble d'éléments prédéfinis à exécuter des fonctions précises. On retrouve dans un MCU tous les éléments d'un mini système, comme une mémoire, un CPU ou encore périphériques (fig. 4.1). La taille de la mémoire est fixe, et est de deux types : volatile et non-volatile. Le CPU est l'élément dont la fonction est d'exécuter des calculs. Il a sa propre microarchitecture et fonctions internes, caractérisée par la largeurs de la donnée manipulée (16 bits, 32 bits ...) ou encore le nombre d'étages de sa chaîne de traitement (pipeline). Dans les architectures modernes, les bus sont divisés en bus d'instruction et bus de donnée.

Le traitement d'une instruction ou une donnée passe par ces différents éléments de l'architecture d'un MCU. Pour la détection de failles exploitables, il est nécessaire de surveiller le fonctionnement de tous ces éléments pouvant être impactés par l'effet d'un rayonnement EM. Cela nécessite d'avoir accès à différentes informations se trouvant

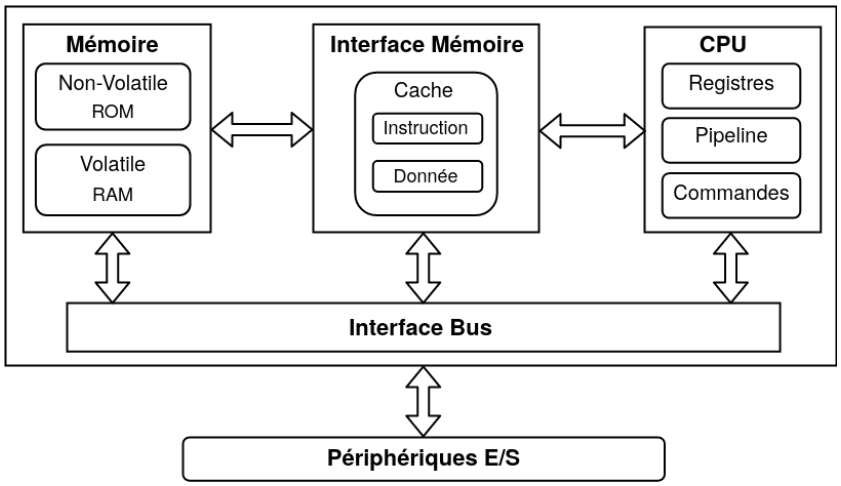


FIGURE 4.1 – Schéma générale de l'architecture d'un microcontrôleur.

dans la mémoire ou dans les registres du CPU. Les méthodes que nous développerons dans la suite se basent entre autre sur l'observation de toute les données relatives à l'état avant et après une injection de faute. Ces observations se font à différents niveaux, sur les instructions comme sur les données, et permettent d'avoir une meilleur traçabilité de la faute, ou encore déterminer ses causes.

4.1.1 Principe des méthodes d'analyse

En se référant à l'état de l'art, peu d'études présentent une analyse avec des informations précises sur l'effet des EMFI sur les MCUs. Aussi, nous avons constaté que les méthodes de caractérisation utilisées manquent de formalisme. Afin d'étudier l'impact des EMFI, nous proposons différentes méthodes génériques, où chacune est dédiée à l'analyse d'une vulnérabilité.

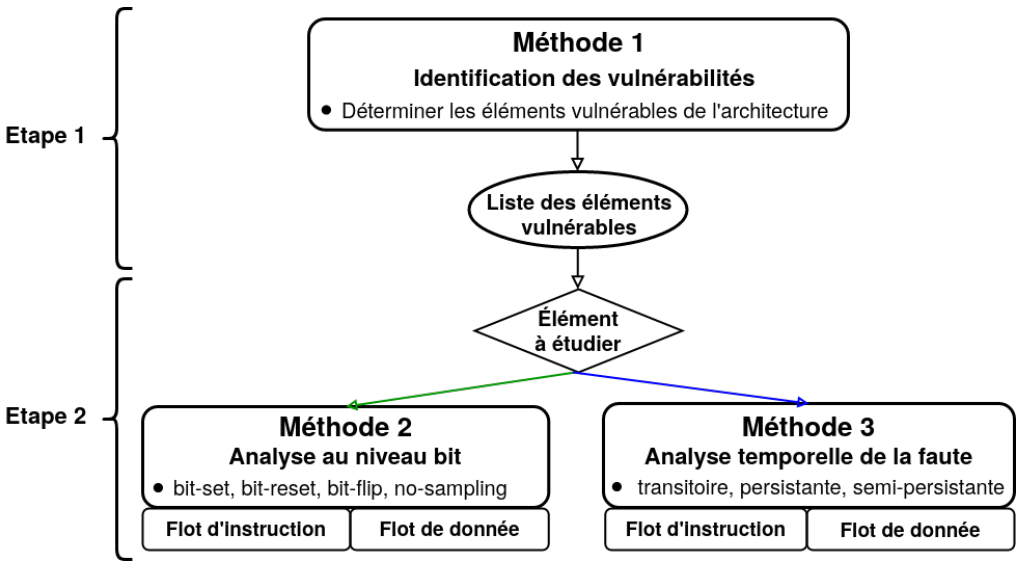


FIGURE 4.2 – Démarche d'analyse pour l'étude des vulnérabilités d'un MCU.

Notre démarche se base sur deux grandes étapes successives et étroitement liées (fig. 4.2). La première étape est définie par une méthode qui permet d'identifier les éléments vulnérables de l'architecture d'un MCU. Cette première étape est importante

puisqu'elle permet d'adapter la configuration et la séquence de test par rapport à un élément de l'architecture défini comme vulnérable. La seconde étape consiste à l'analyse des modèles de faute sur l'élément vulnérable et leur impact dans le temps sur les instructions et les données. Cette étape se base sur deux méthodes : une pour l'analyse des modèles de faute au niveau bit, et la deuxième pour déterminer l'effet temporel de la faute.

Méthode 1 : Identification des vulnérabilités

De manière générale, plusieurs éléments de l'architecture d'un MCU sont sollicités durant une opération. Pendant les cycles d'exécution relatifs à son traitement, une donnée reste sensible à toute perturbation extérieure. Dans cette étape de l'analyse, nous répondons à la question qui porte sur la vulnérabilité du MCU. Plus précisément, il sera question de déterminer les éléments sensibles de l'architecture face aux EMFI.

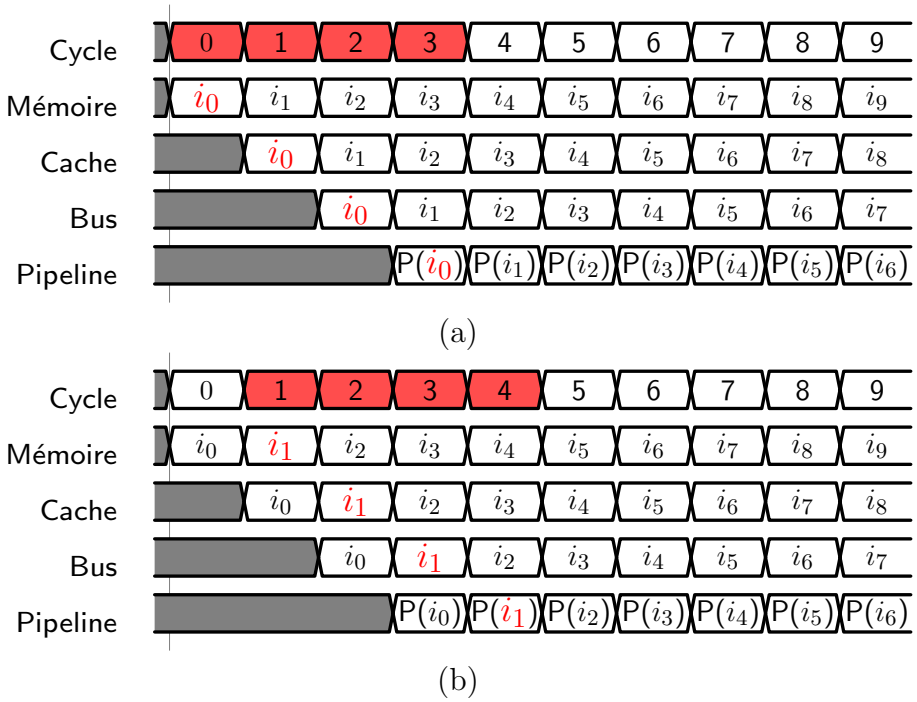


FIGURE 4.3 – Diagramme temporel du traitement d'une instruction dans un MCU avec (a) les cycles de sensibilité pour l'instruction i_0 et (b) pour i_1.

De point de vue fonctionnel, nous proposons dans la Figure 4.3 une hypothèse sur le diagramme temporel du traitement d'instructions relatif à l'architecture générale d'un MCU (fig. 4.1). Avec un traitement séquentiel des instructions, on suppose qu'il y a au moins un cycle de délai pour avoir un transfert stable entre les éléments. Á partir de ce diagramme, nous pouvons établir les cycles d'exécution correspondant. C'est le cas par exemple des instructions i_0 et i_1, avec respectivement leurs cycles de traitement correspondant $[C_0; C_3]$ (fig. 4.3a) et $[C_1; C_4]$ (fig. 4.3b). Sous l'effet des EMFI, et durant un même cycle, plusieurs instructions peuvent être perturbées (quatre instructions durant C_3). D'autre part, les fautes qui seront remontées sur un ou plusieurs cycles d'exécution, révèlent ainsi les cycles de vulnérabilité liés à une instruction cible. Le croisement entre l'instruction et le ou les cycles remontant des fautes, permet alors d'identifier le ou les éléments de l'architecture qui sont vulnérables (faute sur i_0 pen-

dant C_3 signifie une vulnérabilité dans le pipeline).

```
/*  Seq_REF  */          /*   Seq_1   */          /*    Seq_2   */          /*    Seq_N   */
/* 0 Décalage */         /* 1 Décalage */         /* 2 Décalages */        /* N Décalages */

0 inst_cible             0 NOP.W                  0 NOP                    N-7 NOP
1 NOP                    1 inst_cible             1 NOP                    N-6 NOP
2 NOP                    2 NOP                    2 inst_cible             N-5 NOP
3 NOP                    3 NOP                    3 NOP                    N-4 NOP
4 NOP                    4 NOP                    4 NOP                    N-3 NOP
5 NOP                    5 NOP                    5 NOP                    N-2 NOP
6 NOP                    6 NOP                    6 NOP                    N-1 NOP
7 NOP                    7 NOP                    7 NOP                    N-0 inst_cible
```

FIGURE 4.4 – Génération de séquences de test basées sur le décalage de l'instruction cible.

Par ailleurs, ce comportement nous amène à formuler l'hypothèse que si on décale l'instruction cible dans une séquence de test, les cycles de vulnérabilité correspondant seront aussi décalés dans le temps. Pour identifier l'élément vulnérable de l'architecture sous l'effet des EMFI, notre méthode se base principalement sur l'analyse de l'impact durant plusieurs cycles d'exécution. Plus précisément, cette analyse est appliquée sur une série de séquences de test, générées suivant un décalage progressifs d'une instruction cible (fig. 4.4). Nous proposons de limiter la séquence de test à une seule instruction entourée d'instructions de *No Operation* `NOP`. Cette démarche permet de simplifier l'identification de l'élément vulnérable puisqu'il n'y aura qu'une seule opération à étudier. Le nombre de séquences à générer est égale au nombre `N` des cycles de sensibilité, qui est donc égal au nombre d'éléments traitant l'instruction. Á partir d'une séquence de référence, l'instruction est décalée progressivement jusqu'à obtenir une séquence à `N` décalages. L'analyse du croisement entre les cycles de vulnérabilité pour chaque séquence, permet d'identifier les éléments vulnérables à l'effet des EMFI. Notre méthode diffère ainsi des caractérisations de l'état de l'art, puisqu'elle ne se limite pas à un cycle d'exécution précis, mais propose d'analyser l'effet EMFI durant plusieurs cycles d'exécution.

Á la fin de cette première étape de la caractérisation, il est possible de lister les éléments vulnérables de l'architecture d'un MCU, mais aussi de déterminer la relation entre l'opération en faute et le cycle d'exécution correspondant. Une fois cette liste identifiée, la deuxième étape de la caractérisation est l'étude de l'impact sur un élément vulnérable à la fois. La compréhension du fonctionnement de l'élément à étudier s'avère pertinente pour la mise en place d'un plan de test adéquat. En effet, pour maximiser l'observation des fautes, une séquence de test devra répondre aux critères de fonctionnement de l'élément. Au lieu d'utiliser le même code de test pour toutes les analyses, nous proposons d'adapter les codes de sorte à pouvoir identifier un effet précis. Aussi, différents codes de test seront utilisés pour l'analyse sur les instructions et sur les données.

La deuxième étape de la démarche repose sur deux méthodes qui peuvent être distinctes de point de vue application : une méthode pour l'analyse des fautes au niveau bit, et une méthode afin d'étudier le mode temporelle de la faute.

Méthode 2 : Analyse au niveau bit

Après l'identification des éléments vulnérables, et avant une analyse de faute au niveau logiciel, il est important de déterminer les causes des fautes engendrées. Cela passe par l'analyse de l'effet des EMFIs au niveau bit. Quatre modèles de faute sont largement répondus à ce niveau. On retrouve le modèle par forçage à un (*bit-set*), le forçage à zéro (*bit-reset*), l'inversion du bit (*bit-flip*) ou encore la non mise à jour du bit (*no-sampling*).

Les instructions sont définies par leur code opération, dit opcode. Ce dernier est constitué de plusieurs champs, tels que le type d'opération ou encore les opérandes (registre source, registre de destination, valeur). La modification d'un ou plusieurs bits de l'opcode d'une instruction, aura un impact sur son interprétation par le CPU. Deux cas peuvent alors être remontés par rapport à cette modification. Le premier est que l'opcode modifié est interprété en tant que opcode non valide. Dans ce cas, soit il y a une levée d'alarme, avec ou sans interruption de l'exécution du programme, soit l'instruction altérée est ignorée. Dans ce cas précis, on dit que l'instruction a été remplacée par une instruction `NOP`. De point de vue logiciel, ceci est interprété comme un saut d'instruction. La deuxième interprétation possible de l'opcode altéré, est qu'il soit un opcode valide. Ce type de modification est vue au niveau logiciel comme un remplacement d'instruction.

Pour étudier les possibles modifications au niveau bit, notre méthode s'appuie sur la définition de séquences de test, chacune spécifique à un modèle de faute. La génération de ces séquences se fait par le choix d'instructions adéquates, telles que la représentation binaire de l'opcode qui répond aux spécifications du modèle de faute à identifier.

Pour observer le modèle de faute *bit-set* ou *bit-reset*, une seule condition est à vérifier sur l'opcode altéré pour chaque modèle :

- modèle *bit-set* : altérer un opcode dont la valeur binaire est tout à `0`, signifie qu'un ou plusieurs bits ont été forcés à `1`.
- modèle *bit-reset* : altérer un opcode dont la valeur binaire est tout à `1`, signifie qu'un ou plusieurs bits ont été forcés à `0`.

Pour un modèle comme pour l'autre, dire qu'il y a un possible effet de *bit-flip* est aussi valable. Sauf que pour observer le modèle *bit-flip*, les deux conditions énoncées (forçage à `0` et à `1`) doivent être validées. En revanche, pour le modèle *no-sampling*, son observation requière la connaissance de la valeur précédente du *buffer* dédié au chargement de l'opcode. Le Tableau 4.1 résume les cas où l'observation de ces modèles est possible suivant les différentes possibilités de la valeur précédente, attendue et altérée d'un bit de l'opcode. Pour confirmer ces modèles de fautes, il faut vérifier qu'ils répondent à toutes les conditions des cas où ils sont observables.

Sur la ligne de donnée, le même principe est utilisé. La forme binaire de la donnée cible devra répondre aux conditions du modèle de faute à observer. Cette étapes de l'analyse des modèles sera d'autant importante, qu'elle permet de mieux adapter les séquences de test pour une analyse plus aboutie des fautes au niveau logiciel.

TABLEAU 4.1 – Conditions d'observation des modèles de faute *bit-set*, *bit-reset*, *bit-flip* et *no-sampling* en se basant sur la valeur précédente, attendue et altérée d'un bit

bit_{N-1}	bit_N	$bit_{N'}$	**bit-set**	**bit-reset**	**bit-flip**	**no-sampling**
0	0	0	–	–	–	–
0	0	1	✓	–	✓	–
0	1	0	–	✓	✓	✓
0	1	1	–	–	–	–
1	0	0	–	–	–	–
1	0	1	✓	–	✓	✓
1	1	0	–	✓	✓	–
1	1	1	–	–	–	–

Méthode 3 : Analyse temporelle de la faute

Dans un programme, une instruction ou donnée est définie par son adresse mémoire. Dans le cas d'une fonction boucle par exemple, le programme peut être amené à solliciter plusieurs fois cette même instruction ou donnée. Sous EMFI, il est donc important d'évaluer dans le temps l'impact d'une faute sur une opération, quand cette dernière est ré-exécutée. Dans le cas d'une utilisation récurrente d'une même instruction ou donnée, nous proposons une méthode qui aide à identifier la nature de la faute : transitoire, persistante ou semi-persistante.Néanmoins, il est bien important de prendre en considération la condition que la récurrence à étudier est principalement liée à l'adresse et non à la forme de l'instruction ou la donnée.

Pour l'analyse sur la ligne d'instruction, une étude de cas se résume à appeler au moins deux fois la même instruction. Notre observation se portera sur la comparaison des résultats du premier et du second appel, quand une injection de faute est générée durant le premier appel de l'instruction cible :

- si le résultat de la première exécution est en faute, et le résultat de la seconde est correcte, ceci indique une faute transitoire.
- si les résultats de la première et de la seconde exécution sont en faute, ceci indique que la corruption est persistante.

Ce même principe est appliqué pour l'analyse sur les données, avec la comparaison du résultat de deux chargements successifs de la même donnée (chargée à partir de la même adresse mémoire).

Étant donné le placement des éléments constituant un MCU, l'analyse inclut aussi l'identification des zones sensibles du CI de la cible. Cela nécessite une étude de l'impact du paramètre spatial sur toute la surface du CI, tout en variant les paramètres temporels et électriques liés à l'impulsion EM. Ainsi, pour chaque configuration des paramètres d'injection, on dresse la relation entre ces paramètres et l'élément qui la cible du rayonnement EM. Identifier l'élément vulnérable pour chaque position de la sonde d'injection, ainsi que le modèle et la nature temporelle de faute, servira à dresser les vulnérabilités matérielles et logicielles. Ces résultats seront indispensables pour la conception de contre-mesures robustes.

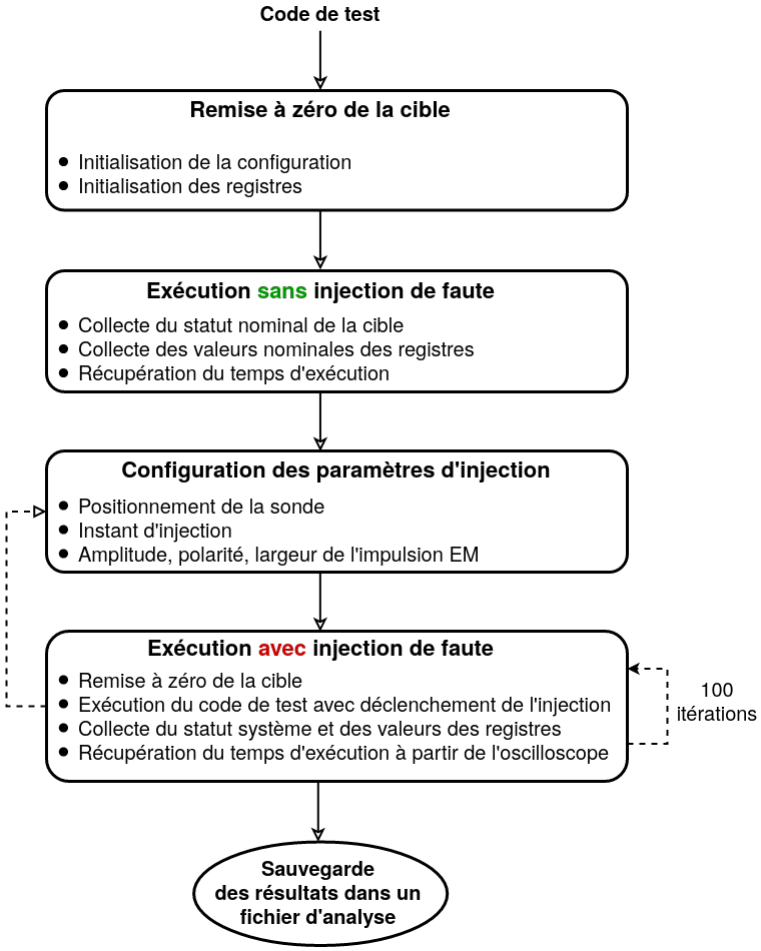


FIGURE 4.5 – Diagramme définissant un test EMFI sur un MCU.

Tout au long des expérimentations, nous tenons à différencier les effets EMFI sur la logique, de ceux qui perturbent les lignes d'horloge (effet *glitch*). Ce point consiste à prévoir un mécanisme pour vérifier le temps d'exécution de la séquence cible. Dans certains MCU, des fonctions de contrôle d'exécution, comme par exemple le compteur de cycle d'exécution, peuvent être utilisés dans ce sens. Il est en tout cas nécessaire de disposer d'un signal pour observer la durée d'exécution du programme sur un oscilloscope. Nous définissons ΔT_{exe} comme étant la différence entre le temps d'exécution nominale T_{exe} et le temps d'exécution post-injection T'_{exe}.

- $\Delta T_{exe} < 0$: indique un allongement du temps d'exécution suite à l'injection de faute.
- $\Delta T_{exe} > 0$: indique une réduction du temps d'exécution suite à l'injection de faute.

Pour une étude précise des modèles de fautes, l'analyse ne doit pas porter seulement sur une observation directe du résultat d'une opération, mais aussi sur l'état de tous les types de registre (à usage libre ou système), ainsi que l'état des interruptions ou le contenu de la mémoire.

Pour un élément vulnérable défini, un code de test est élaboré selon les besoins d'une des trois méthodes énoncées précédemment. La Figure 4.5 résume l'exécution

d'un test EMFI :

1. **Remise à zéro du MCU** : Cette étape sert à réinitialiser la cible à son état initiale (réinitialisation des registres d'état) ainsi qu'à sa reconfiguration.
2. **Exécution sans injection de faute** : Le but de cette exécution est de générer l'état de référence pour un fonctionnement nominale. L'étape est suivi par la collecte des données système ainsi que celles liées à tous les registres. Le temps d'exécution nominal de la séquence cible est récupéré à partir de l'oscilloscope.
3. **Configuration des paramètres d'injection** : Selon la campagne de test, la variation des paramètres d'injection est à configurer :
   - position de la sonde
   - amplitude et polarité de l'impulsion
   - instant d'injection
4. **Exécution avec injection de faute** : Pour chaque ensemble de paramètres d'injection, 100 itérations de la séquence suivante sont produites :
   - remise à zéro de la cible.
   - exécution du code de test avec déclenchement de l'injection de faute.
   - collecte des états des registres ainsi que ceux liés au système.
   - récupération du temps d'exécution post-injection.
5. Sauvegarde des tous les résultats dans un fichier pour analyse

Nous proposons d'appliquer nos méthodes d'analyse sur plusieurs cibles (différents constructeurs) afin d'appuyer l'aspect générique de notre démarche. Dans un même souci de généralité, nous présenterons les résultats obtenus avec l'utilisation de deux générateurs d'impulsions différents.

4.1.2 Classification des résultats

Plusieurs types de comportement peuvent être observés sous l'effet EMFI. On propose de regrouper ces comportements et de les définir en classe. Selon le résultat d'un test, quatre classes sont identifiées et illustrées par l'organigramme de la Figure 4.6. Quand aucune faute n'est observée, le résultat est marqué comme *valide*. Cet état revient à dire que le résultat final est correct et qu'aucune erreur sur les registres ou drapeau système n'est remontée. Si le test se solde par une erreur matérielle, le résultat est classé comme *erreur système*. Ce type d'erreur est reconnu suite à la levée d'interruptions système, l'observation d'une remise à zéro de ce dernier ou son gel. La cause d'une erreur système peut être identifiée en analysant la valeur du registre d'état. La détection d'une faute exploitable est observée seulement si un ou plusieurs registres à usage libre du MCU présentent une valeur autre que celle attendue. Le test est alors marqué comme *faute sur registres*. Dans ce cas, aucune erreur ne doit être remontée sur les registres système. Durant les campagnes de tests, il arrive d'avoir une quatrième classe de résultat, où la valeur d'un ou plusieurs registres est altérée, et en même temps une levée d'un drapeau d'opération. Cette levée de drapeau est essentiellement due au fait qu'une opération peut mettre à jour un drapeau suivant le résultat de cette dernière (résultat nul, résultat négatif ...). Cette classe de résultat *fautes sur registres avec drapeau* est utile, pour identifier durant les tests, un modèle de faute de type modification d'instruction.

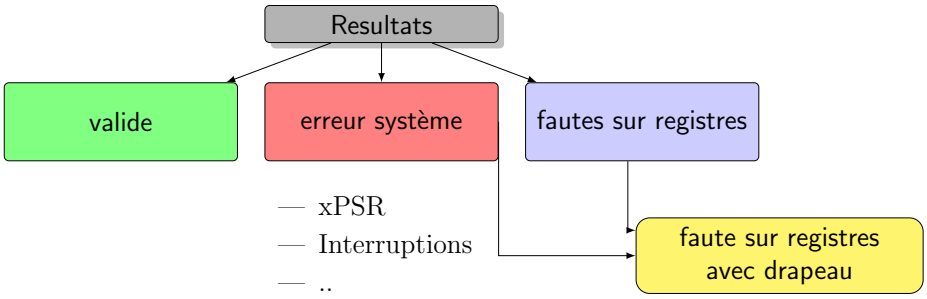


FIGURE 4.6 – Classification des résultats

4.2 Cibles d'étude

Pour étudier l'impact EMFI sur les MCUs, nous avons choisi dans un premier temps de mener notre analyse sur une cible assez répandue sur le marché : La carte de développement STM32F4DISCOVERY [106], qui intègre le MCU STM32F407VG [107], dont le choix a été motivé par la disponibilité d'une version décapsulée du boîtier de ce MCU. Cela permet à la sonde d'injection de se positionner au plus près du silicium, et grâce au placement visuel, mieux identifier les éléments de l'architecture.

Dans un deuxième temps, nous avons étendu nos tests sur d'autres cibles dans le but de généraliser la démarche d'analyse. Ainsi, les expérimentations ont été menées sur deux autres cartes, le Atmel ATSAM4C-EK [108] et le Arduino MKR ZERO [109], intégrant respectivement les MCUs SAMC4C16C [110] et SAMD21G18A [111].

4.2.1 Architecture de la cible STM32F407VG

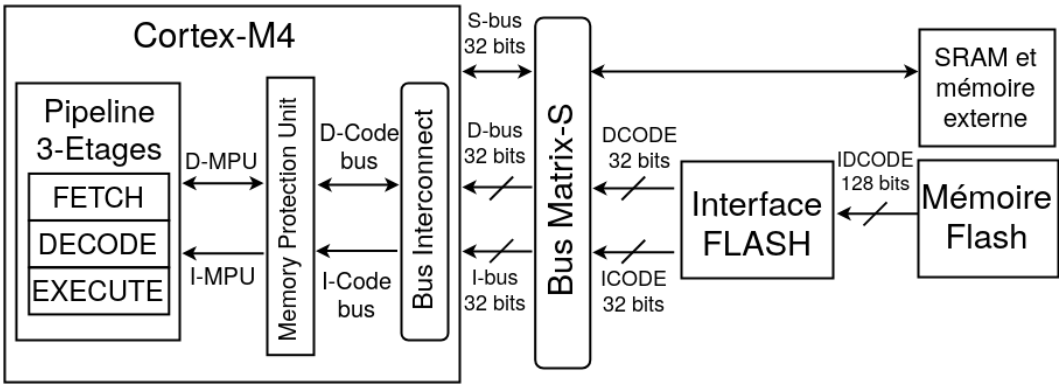


FIGURE 4.7 – Bloc diagramme de l'architecture système du STM32F407.

Le STM32F407VG est un MCU mono-cœur, basé sur un CPU 32 bits ARM Cortex-M4. Ce dernier implémente le jeu d'instruction ARMv7-M avec le support des instructions en encodage 32 bits et 16 bits. Le cœur repose sur une architecture *Harvard*, un pipeline à trois étages (*FETCH*, *DECODE* et *EXECUTE*), et propose seize registres 32 bits. Les treize premiers registres (`R0` à `R12`) sont pour un usage libre. Les trois derniers (`R13`, `R14` et `R15`) offrent un usage système de plus, défini par le registre de pile *SP*, le registre de lien *LR* et le registre du compteur programme *PC*.

De point de vue microarchitecture, de la mémoire vers le CPU et vice-versa, l'acheminement d'une donnée passe par différents éléments (fig. 4.7). Hormis l'interface mémoire, ces éléments assurent le transfert sur deux lignes à 32 bits séparées (ligne d'instruction et ligne de donnée). Une donnée peut être chargée soit depuis la Mémoire Non-Volatile (NVM) *Flash*, ou bien depuis la mémoire volatile interne ou externe *SRAM*. Quand elle est chargée depuis la *Flash*, elle passe par un bus de 128 bits vers une interface mémoire. Dans le cas de lecture d'une ligne d'instruction, les 128 bits dédiés peuvent contenir soit quatre instructions 32 bits, et jusqu'à huit instructions en 16 bits. L'interface mémoire traite ainsi la donnée demandée suivant son type (instruction, constante ou variable). Elle est par la suite délivrée à travers le bus adéquat (d'instruction ou de donnée). L'interface intègre aussi des fonctions d'optimisation, qui sont utiles pour certains programmes à des fins de performance :

- un cache *PREFETCH* d'une seule ligne 128 bits, utilisé seulement sur la ligne d'instruction.
- une fonction cache pour la ligne d'instruction, pouvant charger 64 lignes de 128 bits.
- une fonction cache pour la ligne de donnée, pouvant charger jusqu'à 8 lignes de 128 bits.

La fréquence d'horloge du CPU peut atteindre 168 MHz. Pour une fréquence et une tension d'alimentation donnée, la stabilité de fonctionnement du système dépend du paramètre WS qui défini le nombres de cycles de latence. En effet, la configuration de ce paramètre est requise afin d'ajuster le temps nécessaire pour une opération de lecture d'un flot d'instruction ou de donnée depuis la *Flash*.

Wait states (WS) (LATENCY)	HCLK (MHz)			
	Voltage range 2.7 V - 3.6 V	Voltage range 2.4 V - 2.7 V	Voltage range 2.1 V - 2.4 V	Voltage range 1.8 V - 2.1 V Prefetch OFF
0 WS (1 CPU cycle)	0 < HCLK ≤30	0 < HCLK ≤24	0 < HCLK ≤22	0 < HCLK ≤20
1 WS (2 CPU cycles)	30 < HCLK ≤60	24 < HCLK ≤48	22 < HCLK ≤44	20 <HCLK ≤40
2 WS (3 CPU cycles)	60 < HCLK ≤90	48 < HCLK ≤72	44 < HCLK ≤66	40 < HCLK ≤60
3 WS (4 CPU cycles)	90 < HCLK ≤120	72 < HCLK ≤96	66 < HCLK ≤88	60 < HCLK ≤80
4 WS (5 CPU cycles)	120 < HCLK ≤150	96 < HCLK ≤120	88 < HCLK ≤110	80 < HCLK ≤100
5 WS (6 CPU cycles)	150 < HCLK ≤168	120 < HCLK ≤144	110 < HCLK ≤132	100 < HCLK ≤120
6 WS (7 CPU cycles)		144 < HCLK ≤168	132 < HCLK ≤154	120 < HCLK ≤140
7 WS (8 CPU cycles)			154 < HCLK ≤168	140 < HCLK ≤160

FIGURE 4.8 – Table de correspondance du STM32F407 [107, Tab.10], entre nombre de latence (WS) et tension d'alimentation du MCU, pour garantir un fonctionnement stable pour une fréquence d'horloge donnée.

Nous présentons dans la Figure 4.9 une hypothèse du diagramme temporel d'une séquence d'instructions $[i_0; i_{11}]$. L'exécution d'une seule instruction nécessite un cycle d'horloge. Avec un nombre de latence WS configuré à zéro, la lecture d'un bloc d'instruction 128 bits se fait durant un seul cycle d'exécution. Aussi, nous avançons l'hypothèse qu'il existe un cycle de décalage du transfert entre chaque élément de la microarchitecture. Dans la Figure 4.9 est illustrée une opération de lecture d'une ligne d'instruction 128 bits depuis la *Flash* qui s'effectue tous les quatre cycles (C_0, C_4 et C_8).

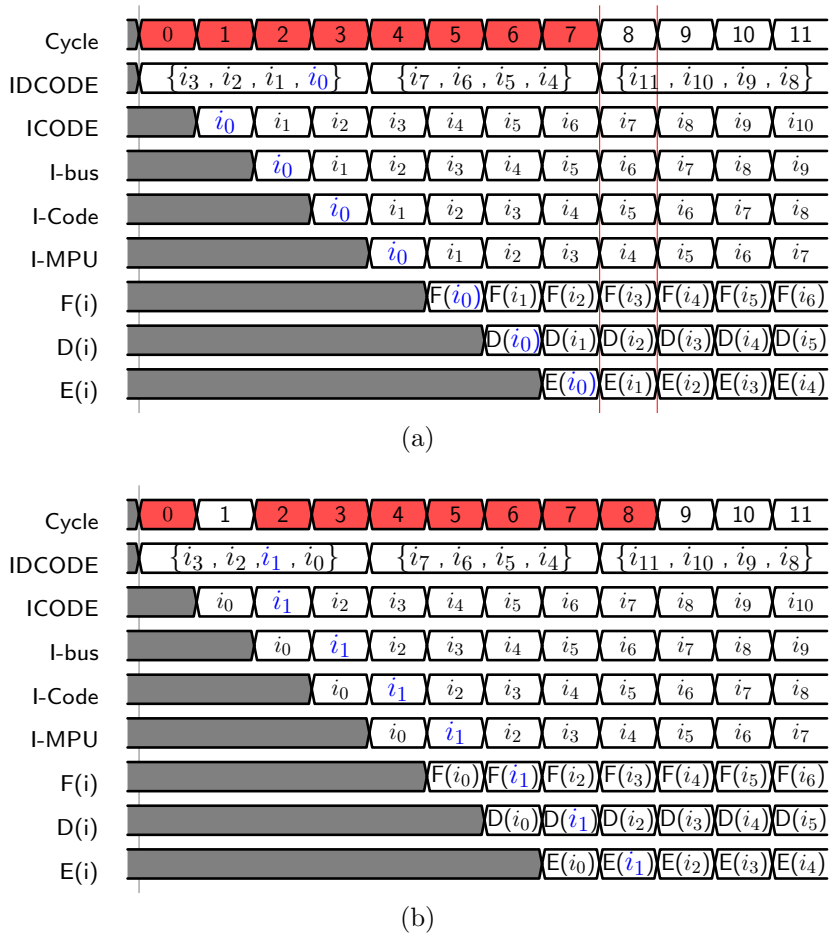


FIGURE 4.9 – Chronogramme d'une opération de lecture 128 bits depuis la *Flash*.

Durant le cycle C_0, une ligne d'instruction composée des quatre instructions 32 bits $\{i_0,i_1,i_2,i_3\}$ est chargée à partir de la mémoire par le biais du bus dédié IDCODE. Pendant les cycles suivants, instruction par instruction passe par les différents bus 32 bits (ICODE, I-bus, I-Code et I-MPU) jusqu'au pipeline. De la mémoire jusqu'à son exécution, une instruction donnée (exemple de i_0), est sensible à une corruption durant huit cycles ($[C_0; C_7]$). Nous pouvons souligner la possibilité de corrompre jusqu'à onze instructions successives ($[i_1; i_{11}]$) si une faute est générée durant le même cycle C_8. Le cas de corruption de plusieurs instructions successives a été discuté par Yuce et al. [54, fig.4] dans une analyse centrée sur les fautes dans le pipeline. Les résultats correspondant ont montré qu'une injection de faute pendant un cycle pourrait altérer autant d'instructions que le nombre d'étages du pipeline.

Á partir de notre hypothèse sur le fonctionnement, on se propose d'étudier les vulnérabilités du STM32F407VG suivant les 3 méthodes décrites précédemment. L'étude est établie à travers une analyse sur les instructions ainsi que sur les données.

4.2.2 Configuration de la plateforme de test

La Figure 4.10 présente la plateforme expérimentale P_{Av} pour la caractérisation sur notre cible. Pour les premières expérimentations, nous avons choisis d'utiliser la sonde LIRMM F [40] comme injecteur connecté à la sortie du générateur d'impulsion *Avtech*. Le choix de la sonde est motivé suite aux résultats de performance observés

dans Chapitre 3, alors que celui du générateur par rapport à la puissance maximum qu'il peut générer (jusqu'à 400 V).

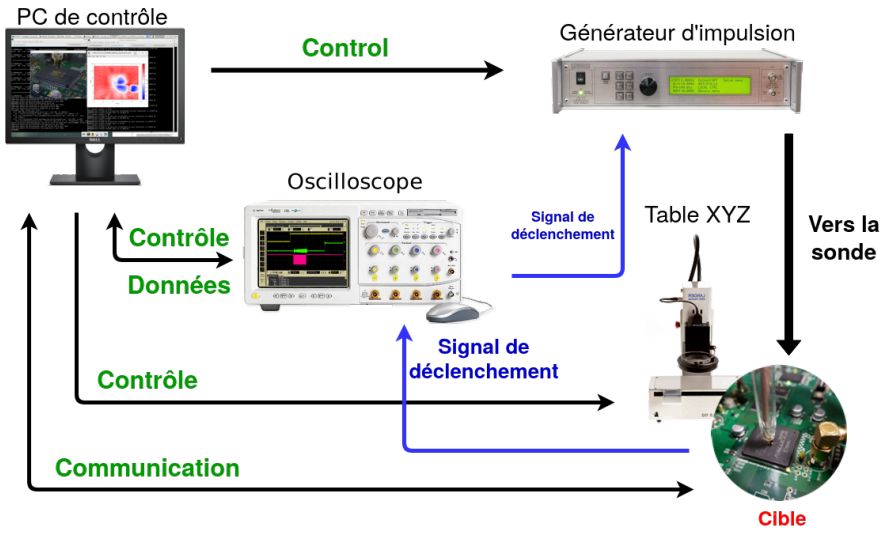


FIGURE 4.10 – Diagramme de la plateforme EMFI avec générateur Avtech pour les expérimentations sur les MCU.

Pour tous les tests, le MCU est configuré de la même manière, où seul la séquence cible est différente :

- Deux ports d'entrée-sortie sont utilisés comme signaux de déclenchement. Le premier TRG_{gen} est relié à un oscilloscope et redirigé vers le générateur d'impulsion pour déclencher l'injection de faute. Le deuxième TRG_{exe}, relié aussi à l'oscilloscope, sert comme signal d'observation du temps d'exécution de la séquence de test. Une directive est intégrée au code de test pour la mise à l'état haut de ce signal, dès le début de la séquence cible, et son passage à zéro à la fin.
- Pour simplifier l'analyse, le cache d'instruction et de donnée ainsi que la fonction *PREFETCH* auront leurs propres fonctions dans le code de test. Il sera ainsi possible de contrôler l'activation/désactivation de ces fonctions selon le besoin du test à effectuer.
- La fréquence d'horloge est configurée à 24 MHz, avec un nombre de WS égale à zéro. Selon certains cas d'analyse, le nombre de cycles de latence sera modifié manuellement.
- Seuls les registres à usage libre $[R0, R9]$ sont utilisés dans les séquences cible. Tous ces registres sont initialisés à des valeurs différentes et autre qu'une valeur nulle. Les autres registres $R10$, $R11$ et $R12$ sont utilisés pour les fonctions de configuration.

Le déroulement détaillé d'un code de test est comme suit :

1. initialisation du MCU.
2. configuration de l'horloge et du nombre de latence WS.
3. configuration de la fonctionnalité du cache.
4. initialisation des registres $[R0, R9]$ à une valeur connue.

5. mise à l'état haut du signal TRG_{exe} pour le début de la séquence.
6. mise à l'état haut et puis bas du signal TRG_{gen} pour déclencher l'injection de faute.
7. exécution de la séquence cible.
8. mise à l'état bas du signal TRG_{exe} pour la fin de la séquence.
9. l'exécution du programme s'arrête quand le PC atteint l'adresse désignée comme point d'arrêt (*breakpoint*).

Concernant les paramètres liés à l'impulsion EM, seule la largeur de l'impulsion est fixée à 6 ns avec un temps de monté de 2.5 ns. Cette largeur est le minimum qu'on peut configurer sur le générateur *Avtech*. Le reste des paramètres à configurer pour générer l'impulsion sont établis comme suit :

- L'amplitude de l'impulsion est à définir pour chaque test. Soit à une valeur fixe, soit en mode variation avec un pas fixe.
- L'instant d'injection de l'impulsion EM est à définir selon l'analyse. La variation de ce paramètre est effectuée avec un pas fixe pour couvrir, soit plusieurs cycles d'exécution, soit une fenêtre d'injection réduite.
- L'effet des deux polarités positive et négative de l'impulsion sera testé et défini au début de chaque expérimentation.
- Une contrainte matérielle du générateur d'impulsion fait qu'il existe un délai entre l'instant de la réception du signal TRG_{gen} par le générateur et la génération de l'impulsion EM. Pour pallier ce délai, des instructions `NOP` sont ajoutées dans la séquence de test.

4.3 Identification des éléments vulnérables de l'architecture

Les travaux de caractérisation précédents ont déjà fait état de vulnérabilité quant à certaines parties de l'architecture des MCU, notamment les bus de donnée [52], le cache d'instruction [58], le mécanisme de préchargement de données *Prefetch* [53]. Notre étude de vulnérabilité prend en considération l'analyse de l'effet quand des instructions ou des données sont traitées. Le diagramme théorique de la Figure 4.9 donne une première idée, sur un ensemble d'éléments de notre cible, qui peuvent être perturbés sous l'effet d'un rayonnement EM.

La première étape de notre démarche d'analyse concerne l'identification des vulnérabilités au niveau architecture. En se basant sur la méthode présentée précédemment, nous proposons une séquence de test de référence (fig. 4.11a) définie par une seule instruction 32 bits (`ADD.W R7,R7,#0x1`). Cette dernière est entourée de nombreuses instructions de type `NOP`. Cela assure qu'aucune autre opération n'est traitée par le CPU durant le test, et permet d'isoler l'effet sur l'instruction cible uniquement.

Sous l'effet des EMFI, un balayage de la sonde avec un pas de 220 µm est effectué sur un boîtier non décapsulé du MCU. La zone de balayage est une zone centrale du boîtier couvrant 4 mm par 4 mm sur les axes X et Y. Durant le test, le générateur d'impulsion est configuré pour générer une impulsion à amplitude fixe de 230 V. La variation de

```
/*  Code_REF  */
/* 0 Décalage */
0 ADD.W R7,R7,#0x1
1 NOP.W
2 NOP.W
3 NOP.W
4 NOP.W
5 NOP.W
6 NOP.W
7 NOP.W
```

(a)

```
/*   Code_1   */
/* 1 Décalage */
0 NOP.W
1 ADD.W R7,R7,#0x1
2 NOP.W
3 NOP.W
4 NOP.W
5 NOP.W
6 NOP.W
7 NOP.W
```

(b)

```
/*    Code_2   */
/* 2 Décalages */
0 NOP.W
1 NOP.W
2 ADD.W R7,R7,#0x1
3 NOP.W
4 NOP.W
5 NOP.W
6 NOP.W
7 NOP.W
```

(c)

```
/*    Code_7   */
/* 7 Décalages */
0 NOP.W
1 NOP.W
2 NOP.W
3 NOP.W
4 NOP.W
5 NOP.W
6 NOP.W
7 ADD.W R7,R7,#0x1
```

(d)

FIGURE 4.11 – Séquences de test pour la méthode d'identification des éléments vulnérables, en appliquant un décalage de l'instruction cible par (b) un, (c) deux et (d) sept décalages à partir de (a) la séquence de référence.

l'instant d'injection se fait avec un pas de 1 ns, couvrant une fenêtre d'injection égale à huit cycles d'exécution. Cette fenêtre est déterminée à partir de la fenêtre du signal TRG_{exe}.

L'expérimentation avec le code de test $Code_{REF}$ a permis de relever des erreurs sur la valeur du registre `R7` (fig. 4.13). La Figure 4.12 donne le résultat du balayage sur l'axe X et Y avec les positions où ces fautes sont observables. On peut clairement identifier que la distribution spatiale de l'impact est concentrée sur une zone délimitée du CI. Ces erreurs n'ont été observées que durant une même courte fenêtre de temps pour toutes les positions, qu'on associera au cycle de référence C_0.

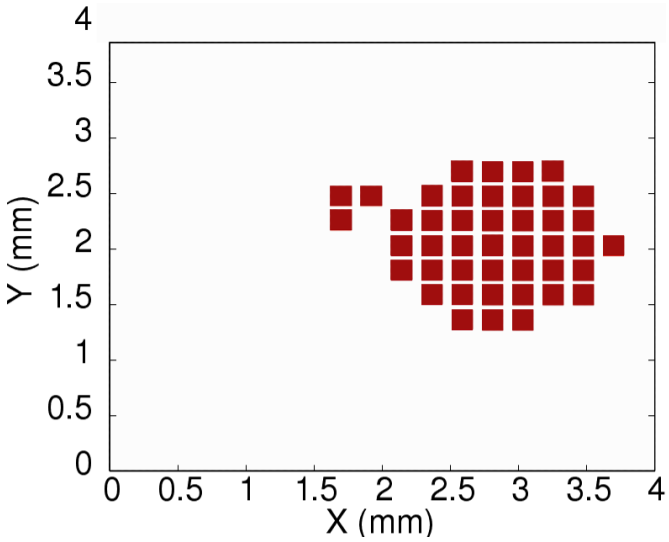


FIGURE 4.12 – Positions de la sonde d'injection où les fautes sur la valeur du registre R7 sont observées.

Suite à ce premier résultat, nous avons émis l'hypothèse que les EMFIs induisent un impact sur un seul élément du fonctionnement (un des étages du pipeline, un bus ...). Á partir de la séquence de test de référence Code_REF, nous avons généré huit codes où la position de l'instruction `ADD` est décalée de façon incrémentale de un à huit décalages fig. 4.11. Sur toutes les séquences testées, la valeur de `R7` n'a été altérée que pour les codes avec quatre et huit décalages fig. 4.13. Cela signifie que l'impact (corruption de la valeur de `R7`) est toujours observé pendant un seul cycle pour un même test.

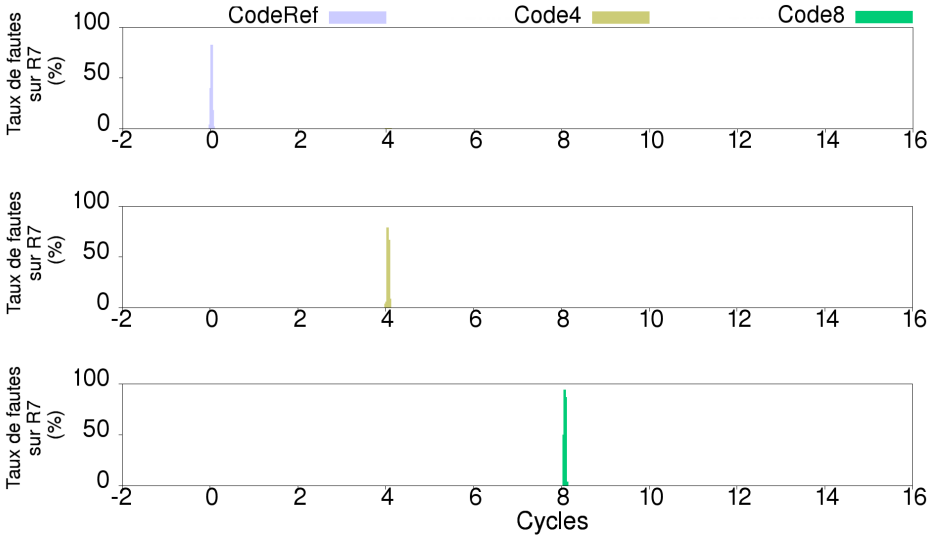


FIGURE 4.13 – Les fautes sur le résultat du registre R7 sont observées durant les cycles C_0, C_4 et C_8, respectivement pour les séquences de tests Code_REF, Code_4 et Code_8.

En combinant ces résultats, nous en déduisons que l'effet des EMFIs est induit tous les quatre décalages d'instructions (équivaut quatre cycle d'exécution). Ce comportement est similaire au processus de chargement de la ligne d'instruction (quatre instructions 32 bits) depuis la *Flash* vers la mémoire tampon de l'interface *Flash*. En effet, notre configuration de la cible (zéro WS, quatre cycles pour l'exécution de la ligne d'instruction) fait que le chargement s'effectue tous les quatre cycles.

Pour confirmer ce point, nous avons répété l'expérience avec des fréquences d'horloge différentes, ce qui implique différentes valeurs de WS. Nous avons observé que les fautes sur le registre `R7` sont remontées durant des cycles avec un intervalle $d_{C_{fault}}$ qui est différent pour chaque fréquence d'horloge. $d_{C_{fault}}$ peut être défini comme l'addition de la valeur configurée des WS, et du nombre de cycles nécessaires pour exécuter chaque instruction de la ligne 128 bits.

L'interface mémoire propose une fonctionnalité qui permet de définir un autre mode de chargement, à savoir le mode *Prefetch*. L'activation de cette option élimine le recours aux cycles de latence WS utiles pour une lecture stable de données depuis la *Flash*. Á partir de l'exemple défini en [107, Fig.5], nous proposons dans la Figure 4.14 le diagramme temporel théorique de fonctionnement avec et sans la fonction *Prefetch*. Quand ce dernier est désactivé, le chargement des blocs de ligne 128 bits (bloc0, bloc1, ...) se fait normalement durant les cycles de latence (dans notre exemple un seul WS). Un bloc chargé est stable durant le cycle suivant sur le buffer $I-CACHE\,[0]$, qui est le buffer d'instruction principale. Quand le *Prefetch* est activé, et vue qu'il y a un préchargement de la ligne d'instruction suivante, un deuxième buffer $I-CACHE\,[1]$ est alors nécessaire pour stocker cette ligne 128 bits préchargée.

Á l'aide d'un code simple, nous avons analysé l'impact des EMFIs sur le chargement d'instructions en ayant la fonction *Prefetch* activée. Pour observer le déclenchement de la fonction de *Prefetch*, la fonction d'activation de cette dernière est effectuée cinq cycles avant le chargement du premier bloc d'instruction de notre séquence de test. Aussi, la mesure à l'oscilloscope du signal d'exécution permettra de confirmer la réduction du

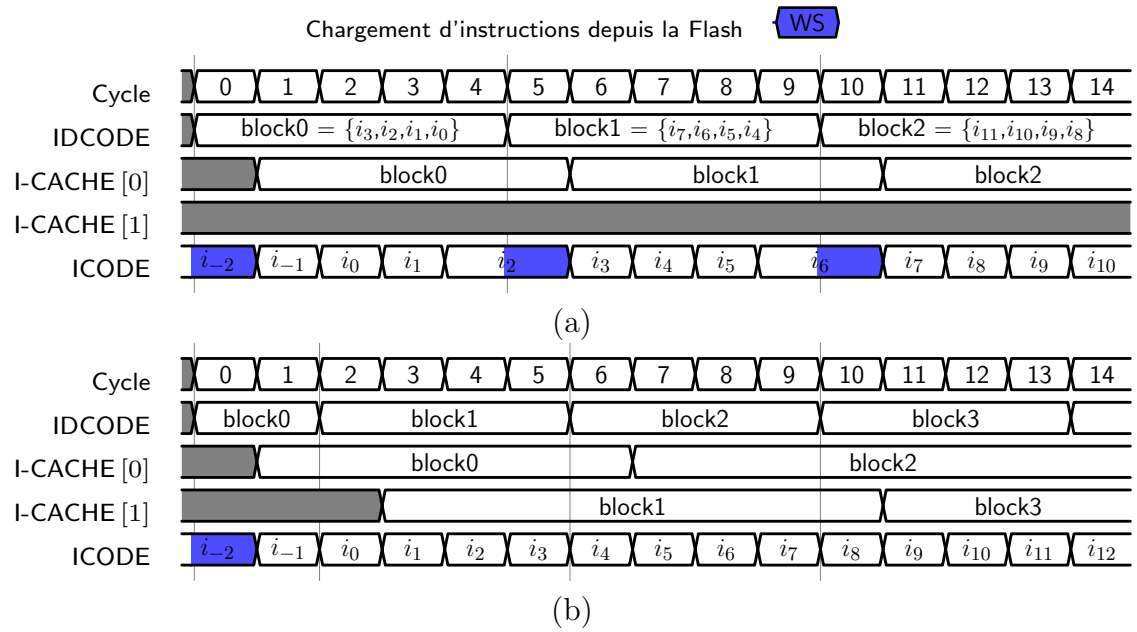


FIGURE 4.14 – Principe de fonctionnement théorique d'un code séquentiel avec la fonction de *Prefetch* (a) désactivée et (b) activée.

temps d'exécution suite à l'activation de la fonction.

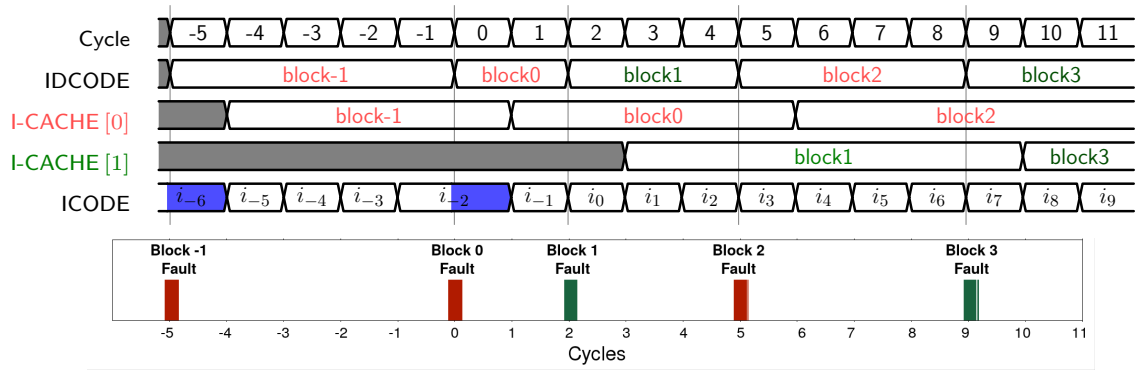
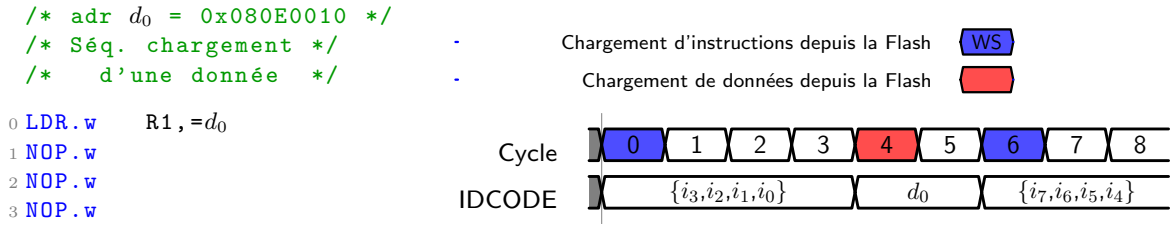


FIGURE 4.15 – L'analyse de fautes avec la fonction *Prefetch* activée montre une corruption du buffer dédié $I-CACHE$ [1].

En analysant les résultats du test d'injection de fautes, nous avons observé des erreurs durant plusieurs cycles, identifiés comme étant les cycles de chargement (fig. 4.15). Cela a permis d'identifier les cycles de sensibilité et définir avec précision la ligne d'instruction impactée. Le premier préchargement d'une ligne d'instruction est effectué durant C_2, c-à-d deux cycles après le chargement du précédent. Á partir de C_5, le MCU fonctionne en mode sans WS avec le chargement d'une ligne d'instructions tous les quatre cycles. Générer des fautes sur le buffer d'instruction principale I-CACHE[0] étant déjà établi, ce test démontre que les injections EM peuvent aussi induire des fautes sur le buffer de *Prefetch* I-CACHE[1].

La même méthode d'analyse a été suivie pour l'étude de l'impact sur une ligne de donnée. Seules les opération de type *LOAD* et *STORE* peuvent interagir avec les données. Comme pour l'analyse précédente sur la ligne d'instruction, nous avons choisi un code de référence simple (fig. 4.16a) d'une seule instruction 32 bits (`LDR.w R1,=d0`). L'opération consiste donc à charger dans le registre `R1` l'adresse de la constante `d0`. Pour

différencier le cycle de chargement d'instruction à une autre opération, nous avons effectué le test en configurant le cycle de latence WS à un.



(a) (b)

FIGURE 4.16 – Séquence de test (a) utilisée pour l'analyse de l'effet EMFI sur l'opération de chargement d'une donnée 32 bits avec (b) le diagramme temporelle du test correspondant.

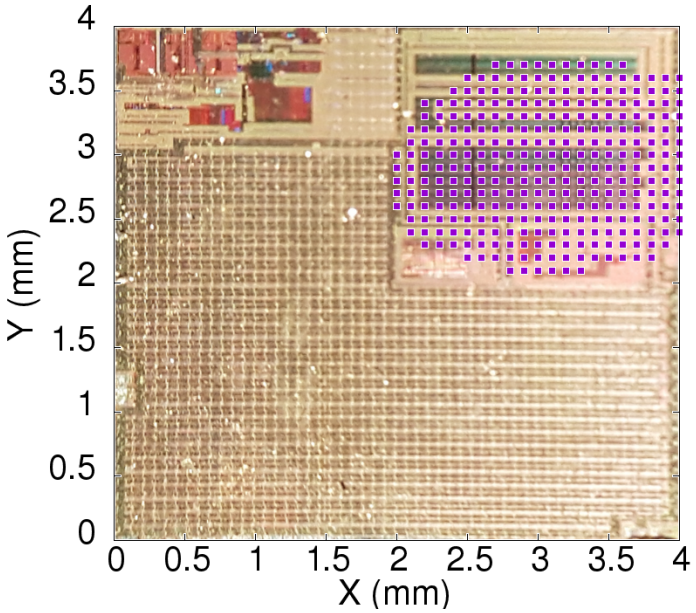


FIGURE 4.17 – STM32F4 décapsulé avec les positions de la sonde d'injection, où des fautes sont observées sur les instructions et les données.

Le résultat du balayage avec un pas de 100 µm sur une cible décapsulée, donne les positions où des fautes sont observées (fig. 4.17). La zone de sensibilité pour perturber le chargement d'une ligne de donnée 128 bit, s'est révélée identique à celle identifiée durant l'analyse sur la ligne d'instruction. Aussi, cette zone d'impact est plus large que celle observée sur un boîtier non-décapsulé. Cela est dû au fait que la sonde d'injection est positionnée au plus prêt du silicium du CI suivant l'axe Z. La Figure 4.16b présente le résultat d'identification des cycles de vulnérabilité sous forme de diagramme temporel. En plus d'une vulnérabilité durant le cycle du chargement d'une ligne d'instruction (durant C_0), un autre cycle (C_4) a été identifié où seule la valeur de la donnée chargée est altérée. Nous supposons donc que c'est durant ce cycle que l'opération de chargement d'une donnée depuis la mémoire *Flash* est effectuée.

On définit pour la suite, les fenêtres d'injection ΔTi_{fault} et ΔTd_{fault} en tant que fenêtres de temps où les chargements respectifs des lignes 128 bits d'instruction et de donnée peuvent être perturbé. Au niveau architecture, pour les instructions comme pour les données, seule l'interface mémoire semble être vulnérable aux EMFIs. La prochaine étape de la démarche consiste à étudier plus en détails cette vulnérabilité, avec une analyse au niveau bit des fautes observées, ainsi que l'étude de leur impact d'un point de vue temporel avec les méthodes 2 et 3.

4.4 Analyse des fautes au niveau bit

Afin de modéliser les fautes au niveau bit, nous proposons une méthode d'analyse élaborée sous forme de séquences de test spécifiques à chacun des modèles de faute discutés précédemment. Avec le résultat de la première étape, nous avons identifié que les fautes sont générées durant les cycles de chargement des instructions ou des données depuis la mémoire NVM. Pour les résultats qui vont suivre, nous présentons l'analyse de l'effet quand les injections de faute sont générées durant les fenêtres de temps ΔTi_{fault} pour la ligne d'instruction et ΔTd_{fault} pour les analyses sur la ligne de donnée.

4.4.1 Impact sur la ligne d'instruction

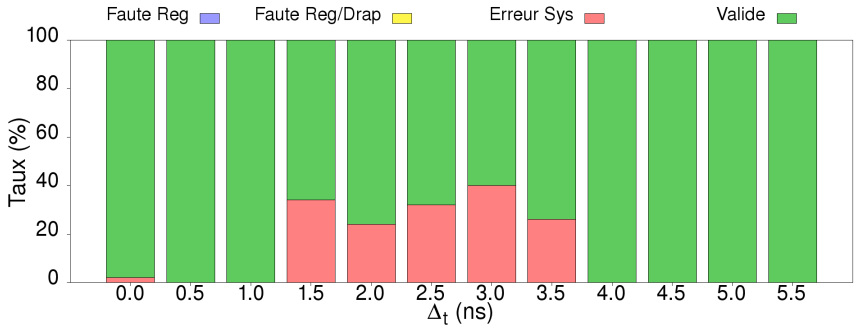
Au niveau instruction, et sous l'effet des perturbations EM, quatre modèles de faute sont largement observés dans l'état de l'art. Le plus discuté est le saut d'instruction (*instruction skip*). On trouve aussi le remplacement d'instruction, la duplication (ou rejeu) d'instruction et la corruption de données. D'autre part, la cause principale qui induit ces modèles est principalement une modification de l'opcode de l'instruction cible. Cette modification se fait au niveau bit par une corruption d'un ou plusieurs bits de l'opcode sous l'effet des injections de fautes. L'étude qui suit est effectuée sur une position fixe de la sonde et avec une impulsion configurée à une amplitude de 230 V. Un test de balayage sera par la suite effectué pour généraliser les résultats.

Modèle bit-set

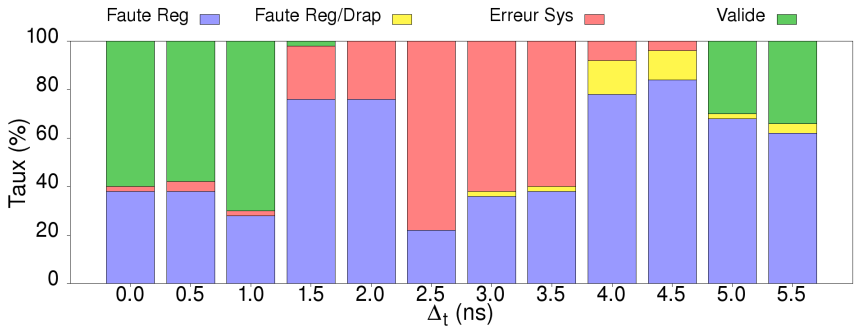
Un effet de *bit-set* implique qu'un bit est forcé à avoir la valeur `1`. L'observation d'un tel effet n'est possible seulement si la valeur attendue du bit est égale à `0`. De ce fait, remplir cette condition lors du chargement d'une ligne d'instruction 128 bits, revient à charger 128 bits tous à la valeur `0`.

D'après la documentation ISA du ARMv7 [112], il n'y a pas d'instruction 32 bits dont l'opcode est tout à zéro. Par contre, il existe une instruction 16 bits qui répond à notre critère. L'instruction en question est `MOVS R0,R0` dont l'opcode est `0000`. Par conséquent, afin d'avoir un chargement d'une ligne d'instruction tout à zéro, huit instructions successives 16 bits `MOVS R0,R0` seront nécessaire (fig. 4.20a). Au cours d'un test EMFI, l'observation d'une faute sur la valeur d'un registre (classe de résultat *fautes sur registres*) nous indiquera qu'il y a eu modification d'une instruction de la séquence. Cela revient à dire qu'un ou plusieurs bits du opcode `0000` ont été forcés à un (ou inversés *bit-flip*).

Hors, d'après les résultats du test fig. 4.18a, aucune faute n'a été observée appartenant à la classe *faute sur registres*. Seules des fautes de la classe *erreur systèmes* ont été remontées. En utilisant cette configuration expérimentale, ce résultat élimine la possibilité que le l'impact EMFI induit un effet de *bit-set*, voir même un effet de *bit-flip* au niveau bit.



(a)



(b)

FIGURE 4.18 – Taux d'occurrences des différentes classes de résultats pour les séquences de test (a) *bit-set* et (b) *bit-reset*.

Modèle bit-reset

En utilisant le même principe de la précédente méthode, nous avons défini la séquence de test qui sert à vérifier un possible effet de *bit-reset*. Contrairement à l'effet *bit-set*, celui du *bit-reset* n'est observable que seulement si la valeur attendue d'un bit est à la valeur `1`. On peut donc penser à avoir une séquence où la valeur binaire de l'opcode de l'instruction cible est tout à `1`. Malheureusement, un tel opcode n'existe pas ni en encodage 32 bits ni en 16 bits.

```
/* opcode   |   instruction */
7FFF            ldrb r7, [r7, #0x1f]
CFFF            ldm r7, {r0, r1, r2, r3, r4, r5, r6, r7}
DFFF            svc #0xff
3FFF            subs r7, #0xff  -> choix pour séquence bit-reset
5FFF            ldrsh r7, [r7, r7]
6FFF            ldr r7, [r7, #0x7c]
9FFF            ldr r7, [sp, #0x3fc]
```

FIGURE 4.19 – Liste des instructions ARMv7 16 bits dont l'opcode contient le plus de nombre de bits à 1.

La Figure 4.19 liste les instructions disponibles en ISA ARMv7 où l'opcode d'instructions 16 bits contiennent le plus de bits à `1`. Pour mieux identifier l'effet du *bit-reset*, notre choix s'est porté sur l'instruction `SUBS R7, #0xFF` (opcode `3FFF`). La séquence de test se compose donc de huit instructions 16 bits `SUBS R7, #0xFF` (fig. 4.20b). Le résultat de la Figure 4.18b montre l'observation de fautes de la classe *faute sur registres* avec un taux de reproductibilité de faute à $\approx 80\,\%$. Ce résultat indique qu'au moins, une instruction sur huit qui composent la ligne 128 bits, a été remplacée par une autre et correctement interprétée par le CPU. Un tel comportement n'est possible que si au moins un bit de l'opcode `3FFF` a été mis à zéro. Étant donné qu'aucune faute exploitable n'a été observée durant le précédent test, nous pouvons affirmer que les injections EM induisent un effet de *bit-reset* sur la ligne d'instruction 128 bits. Nous pouvons aussi affirmer que le modèle de faute *bit-flip* n'est pas effectif sous l'effet des EMFI.

```
/* séquence bit-set */
op   | instruction
/* 8 x 16 bits */

0 0000   MOVS R0,R0
1 0000   MOVS R0,R0
2 0000   MOVS R0,R0
3 0000   MOVS R0,R0
4 0000   MOVS R0,R0
5 0000   MOVS R0,R0
6 0000   MOVS R0,R0
7 0000   MOVS R0,R0
```

(a)

```
/* séquence bit-reset */
op   | instruction
/* 8 x 16 bits */

0 3FFF   SUBS R7,#0xFF
1 3FFF   SUBS R7,#0xFF
2 3FFF   SUBS R7,#0xFF
3 3FFF   SUBS R7,#0xFF
4 3FFF   SUBS R7,#0xFF
5 3FFF   SUBS R7,#0xFF
6 3FFF   SUBS R7,#0xFF
7 3FFF   SUBS R7,#0xFF
```

(b)

```
/*              séquence no-sampling              */

/*    (0->1)   */
op   | instruction
/* bloc0 8 x 16 bits */

0 0000   MOVS R0,R0

/* bloc1 8 x 16 bits */

8 3FFF   SUBS R7,#0xFF
```

(c)

```
/*    (1->0)   */
op   | instruction
/* bloc0 8 x 16 bits */

0 3FFF   SUBS R7,#0xFF

/* bloc1 8 x 16 bits */

8 0000   MOVS R0,R0
```

(d)

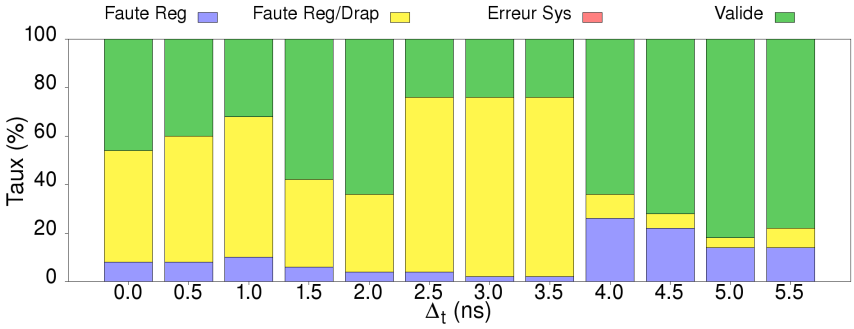
FIGURE 4.20 – Séquences de test utilisées pour identifier l'effet EMFI au niveau bit : (a) *bit-set*, (b) *bit-reset*, (c) *no-sampling* (tout à 0 vers tout à 1) et (d) *no-sampling* (tout à 1 vers tout à 0).

Modèle no-sampling

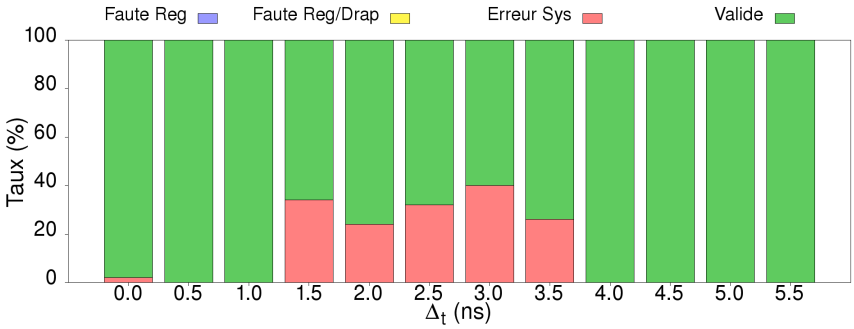
Quand un bit n'est pas mis à jour avec la bonne valeur, le modèle est défini en tant que faute de *no-sampling*. D'après le Tableau 4.1, nous avons défini deux séquences de test qui répondent aux conditions pour l'observation de ce modèle de faute. La première représente le chargement de deux blocs opposés de point de vue du contenu. Le premier bloc d'instruction est composé d'une ligne 128 bits tout à `0` (huit instructions successives `MOVS R0,R0`). Quant au second, la majorité des bits sont à `1` (huit instructions successives `SUBS R7,#0xFF`) fig. 4.20c. La deuxième séquence est l'inverse de la première, avec un premier bloc de huit instructions successives `SUBS R7,#0xFF`, suivi par huit instructions `MOVS R0,R0` fig. 4.20d.

Pour ces deux séquences, l'injection de faute est effectuée durant le cycle de chargement du second bloc. Cela nous permettra de valider une possible non mise à jour d'un bit de la ligne d'instruction. Rappelons que le modèle *no-sampling* n'est effectif que seulement s'il est observable sur les deux séquences de tests. Ce qui n'est pas le cas d'après les résultats de l'expérimentation (fig. 4.21). Une corruption lors du chargement du bloc d'instructions cible n'est observable que pour la séquence où les 128 bits du bloc sont composés par huit instruction `SUBS R7,#0xFF`) (fig. 4.21a). Ce résultat confirme celui du test sur le modèle *bit-reset* à savoir que ce dernier est le seul modèle de faute observé au niveau bit. Il est à noter que ces tests ont été appliqués avec une impulsion

avec polarité positive et négative. Pour les deux polarités, le même effet *bit-reset* est observé.



(a)



(b)

FIGURE 4.21 – Taux d'occurrences des différentes classes de résultats pour les séquences de tests *no-sampling* (a) tout à 0 vers tout à 1 et (a) tout à 1 vers tout à 0.

4.4.2 Impact sur la ligne de donnée

Menu et al. [53] ont présenté une caractérisation de l'impact des injection EM sur la ligne de donnée. Plus précisément, il est question de l'impact sur le *buffer* de *Prefetch* de donnée d'un MCU 32 bits ATSAM3X8. Au niveau bit, les modèles de faute *bit-set* et *bit-reset* ont été observés, en utilisant une impulsion en polarité positive ou négative. L'observation d'un modèle comme l'autre dépend principalement de l'instant d'injection de l'impulsion EM.

Pour identifier le modèle de faute au niveau logique sur une ligne de donnée, nous avons appliqué la même méthode utilisée pour l'étude de l'impact sur la ligne d'instruction. Pour identifier un effet de *bit-set* sur la ligne de donnée, notre méthode consiste à charger un mot de 128 bits dont tous les bits sont à la valeur `0`. L'analyse de la corruption d'une donnée est plus simple que pour les instructions, puisqu'il suffit de vérifier si la valeur de la donnée chargée comporte un ou plusieurs bits à `1`. Pour arriver à contrôler la valeur chargée d'un mot 128 bits, nous avons analysé l'effet EMFI selon deux cas de chargement. Le premier quand le chargement concerne des données se trouvant dans des adresses mémoire successives, et le deuxième cas quand les données sont des adresses mémoire non-successives.

Pour le premier cas, la séquence de test comprend le chargement de quatre données à adresses successives de la mémoire (d_0,d_1,d_2,d_3), vers un registre correspondant (R_1,R_2,R_3,R_4).

Comme expliqué précédemment, l'instruction `LDR.w Ri,=dj` charge dans R_i l'adresse de la donnée d_j. Par identification des cycles de vulnérabilité, le résultat du test montre qu'un seul cycle (C_4) suffit à charger quatre données à adresses successives fig. 4.22. En effet, durant ce cycle, les registres ont été chargés avec des valeurs (adresses) (d_0',d_1',d_2',d_3') autres que celles demandées. Ce qui rejoint le résultat de [53], à savoir que lorsque une donnée est chargée depuis la *Flash*, un mot de 128 bits est en réalité chargé en un seul cycle d'exécution. Ce mot équivaut à quatre données de 32 bit chargées à partir de l'adresse de la première donnée demandée par le programme.

```
/* adr d0 = 0x080E0010 */
/* adr d1 = 0x080E0014 */
/* adr d2 = 0x080E0018 */
/* adr d3 = 0x080E001C */
/* Séq. chargement */
/*   de données    */
/*   successives   */
0 LDR.w   R1,=d0
1 LDR.w   R2,=d1
2 LDR.w   R3,=d2
3 LDR.w   R4,=d3
```

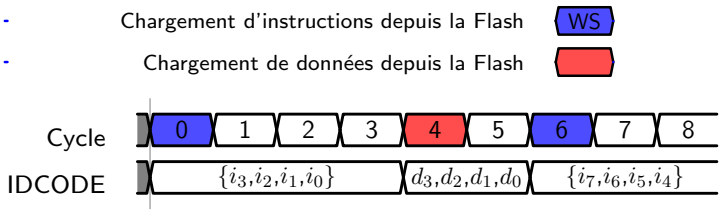


FIGURE 4.22 – Un seul cycle (C_4) est nécessaire pour charger quatre données à adresses successives dans la mémoire.

Le second test est réalisé avec une séquence de chargement de quatre données 32 bits à adresses non-successives dans la mémoire (d_0,d_4,d_8,d_{12}). Le résultat du test fig. 4.23 vient valider la dernière constatation puisque l'opération nécessitera quatre cycles (C_4,C_6,C_8 et C_{10}) pour charger chacune des données. Ce résultat nous donne plus de détails quant à la manière dont le MCU gère les opération liées au traitement de données.

```
/* adr d0 = 0x080E0010 */
/* adr d4 = 0x080E0020 */
/* adr d8 = 0x080E0030 */
/* adr d12 = 0x080E0040 */
/* Séq. chargement */
/*   de données    */
/* non-successives */
0 LDR.w   R1,=d0
1 LDR.w   R2,=d4
2 LDR.w   R3,=d8
3 LDR.w   R4,=d12
```

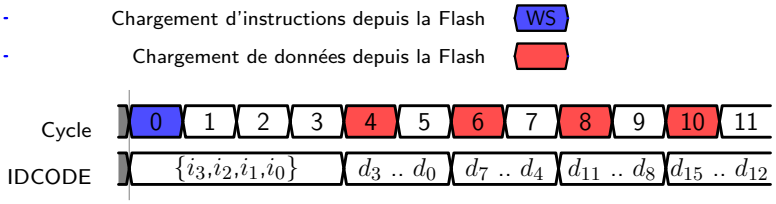


FIGURE 4.23 – Quatre cycles (C_4,C_6,C_8 et C_{10}) sont nécessaires pour charger quatre données à partir des adresses mémoire non-successives.

Pour l'analyse au niveau bit, notre choix de séquence de test est finalement similaire à celle de [53]. Elle se résume à une seule instruction `LDM` qui effectue un chargement multiple de données vers un jeu de registre (`R1`,`R2`,`R3` et `R4`). Quatre données 32 bits (d_1,d_2,d_3,d_4) sont initialisées à la valeur `00000000`, et leurs chargements s'effectue par

incrémentation d'adresse à partir de celle de référence (fig. 4.24).

```
/*   Seq Trig Haut   */
..
/*Chargement adr_ref*/
LDR.w   R0,=d1_addr
/* Inst. NOP */
..
```

```
/* bloc 0 */
0 LDM.w   R0!,{R1-R4}
1 NOP.w
2 NOP.w
3 NOP.w
```

```
/*   Inst. NOP   */
..
/* Seq Trig Bas */
..
```

FIGURE 4.24 – Séquence de test basée sur le chargement multiple de données successives vers un jeu de registres à partir d'une adresse de référence.

Le test d'injection de faute est réalisé sur une position fixe de la sonde durant la fenêtre d'injection ΔTd_{fault}. Les différents test ont montré un taux de reproductibilité élevé des fautes de la classe *fautes sur registres*. Ce résultat signifie que au moins un bit des 128 bits du flot de donnée a été altéré sous l'effet EMFI en induisant un effet de *bit-set*. Pour le cas des chargements de donnée, on se retrouve avec un modèle de *bit-set*, contrairement au modèle sur la ligne d'instruction. L'effet est confirmé par le test de chargement d'un mot de 128 bits tout à `1`, où les quatre données 32 bits (d_1,d_2,d_3,d_4) sont initialisées à la valeur `FFFFFFFF`. Le résultat du test pour l'effet *bit-reset* ne donne aucune faute sur les données chargées.

Pour le modèle de faute *no-sampling*, la séquence de test dédiée (fig. 4.25) se base sur deux instructions `LDM`, dont la deuxième charge une donnée d_6 à partir d'une adresse non-successive à la précédente donnée d_1. Sachant qu'un chargement d'une seule donnée équivaut un mot de 128 bits de données successives (à partir de l'adresse de la donnée cible), ceci implique d'initialiser les données précédentes soit toutes à `1` soit toutes à `0`.

Le premier cas testé est quand les valeurs des données (d_1,d_2,d_3,d_4) sont toutes égales à `FFFFFFFF`, alors que celles des données (d_6,d_7,d_8,d_9) est `00000000`. Le deuxième cas est l'inverse du premier avec (d_1,d_2,d_3,d_4) toutes égales à `00000000`, et celles des données (d_6,d_7,d_8,d_9) est `FFFFFFFF`. Les adresses des données d_1 et d_6 sont chargées respectivement dans les registres `R0` et `R5`. L'injection de faute est générée durant la fenêtre ΔTd_{fault} relatif au chargement de la donnée d_6.

```
/*   Seq Trig Haut   */
/*Chargement adr_ref*/
LDR.w   R0,=d1
LDR.w   R5,=d6
..
..
```

```
/* bloc 0 */
0 LDM.w   R0!,{R1-R4}
1 LDM.w   R5!,{R6-R9}
2 NOP.w
3 NOP.w
```

```
/*   Inst. NOP   */
..
/* Seq Trig Bas */
..
```

FIGURE 4.25 – Séquence de test basée sur deux opérations de chargement successives pour le test du modèle de faute *no-sampling*.

Les résultats de ces deux tests montrent que les fautes sur les données ne sont observées que pour le premier cas. Cela confirme que sur la ligne de donnée, le rayonnement EM induit un effet de *bit-set* durant le chargement d'un mot de 128 bits composé de quatre données 32 bits à adresses successives. Aussi, sur notre cible, on n'est pas arrivé à avoir un résultat similaire à [53] en inversant la polarité de l'impulsion. Le même effet de

bit-set est observé avec une impulsion à polarité positive ou négative. Cette observation peut être expliquée du fait que les deux MCUs présentent une architecture différente (différent constructeurs), et que les équipements des plateformes EM diffèrent.

4.5 Impact du paramètre spatial sur la distribution des fautes

Nous présentons dans ce qui suit une analyse détaillées de l'impact des injections EM sur l'interface mémoire. Les résultats précédent présentent l'impact des EMFI dans la corruption de l'opération de lecture depuis la *Flash* vers l'interface mémoire. Plus précisément, nous avons démontré la corruption d'une ligne 128 bits d'instruction ou de donnée lors de leur chargement respectif. Pour les instructions, le bloc de 128 bits peut comporter au moins quatre instructions 32 bits et jusqu'à huit instructions 16 bits. Pour la ligne de donnée, un mot de 128 bits est chargé même si l'opération traite qu'une seule donnée.

Ce type de corruption est d'autant plus dangereux si un attaquant arrive à cibler précisément une ou plusieurs instructions ou données du bloc chargé. Á travers une caractérisation de l'impact par variations des paramètres d'injection, nous proposons une analyse qui montre l'effet du paramètre spatial dans le contrôle d'une telle propriété de l'attaque. L'analyse est effectuée sur le flot d'instruction ainsi que sur le flot de donnée.

4.5.1 Distribution spatiale des fautes

Le résultat précédent présente l'effet du rayonnement EM sur un bloc 128 bits composée de quatre instructions 32 bits, et dont une seule instruction représente une opération. Nous présentons dans un premier temps d'étendre l'analyse de l'impact sur un bloc composé de quatre instructions 32 bits de type `ADD`. Dans un deuxième temps, nous porterons l'analyse sur une ligne d'instruction de huit instructions 16 bits du même type. Afin d'avoir une caractérisation complète, nous proposons dans ce test sur une cible non-décapsulé, une variation des paramètres d'injection EM, à savoir l'amplitude et l'instant d'injection de l'impulsion, ou encore la position de la sonde. On pourra alors dresser un premier lien entre les paramètres d'injections et l'emplacement des instructions altérées au sein du bloc 128 bits d'instructions.

Concernant le paramètre spatial, le balayage est effectué sur les coordonnées X et Y (une zone centrée du boîtier couvrant 4 mm par 4 mm) avec un pas de la sonde d'injection de 220 µm. Pour chaque position P_i de la sonde, nous procédons à la variation de l'amplitude V_{pulse} de l'impulsion de 200 V à 400 V avec un pas de 3 V. Pour chaque V_{pulse}, la variation de l'instant d'injection T_{pulse} est configurée avec un pas de 0.5 ns. Cette variation est programmée durant une fenêtre de temps liée au cycle de chargement de la ligne d'instruction cible ΔTi_{fault}. Pour chaque configuration donnée (P_i,V_{pulse},T_{pulse}), nous procédons à 100 itérations pour calculer le taux de reproductibilité.

En ce qui concerne la séquence de test, nous avons ciblé le chargement de quatre instructions 32 bits de type `ADD.w Rx,Rx,#0x1` ($x \in [1, 4]$). Avec l'analyse de la valeur

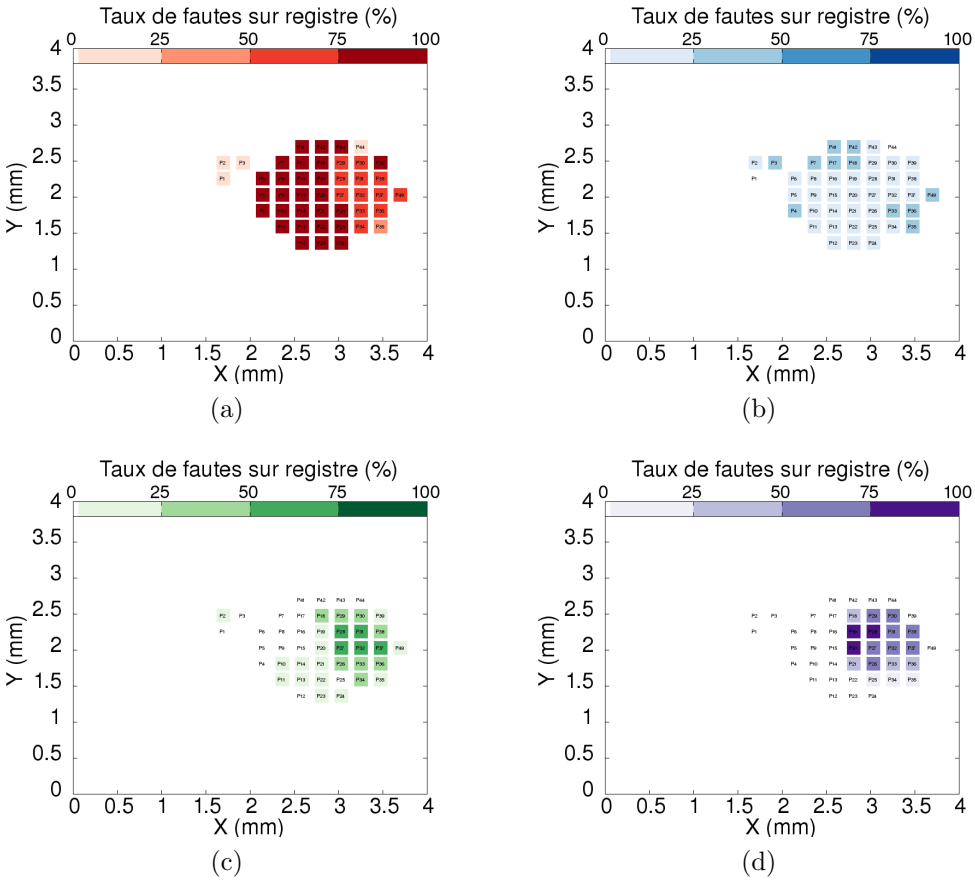


FIGURE 4.26 – Distribution spatiale des fautes sur les registres au cours du chargement d'un flot de quatre instructions 32 bits, quand les fautes sont observées sur (a) une seule, (b) deux, (c) trois et (d) tous les quatre instructions.

de chaque registre correspondant (`R1`,`R2`,`R3` et `R4`), on peut facilement identifier la corruption d'une instruction et son emplacement dans le bloc 128 bits fig. 4.27a.

La Figure 4.26 donne le résultat du balayage de la cible en se basant sur le nombre d'instructions en fautes dans le bloc chargé. Suite à la variation de V_{pulse} et T_{pulse}, nous mettons en évidence pour chaque position sensible P_i ($i \in [1, 44]$), le taux de reproductibilité maximum obtenu pour la classe *faute sur registre*. Nous ne prenons en compte que les fautes affectant les registres concernés par le code de test (`R1`,`R2`,`R3` et `R4`). La plupart des modèles de fautes observés sont soit un effet de saut d'instruction, soit un modèle de substitution d'opcode (corruption de l'opération ou l'opérande). Altérer une seule instruction de la séquence (fig. 4.26a) est la faute la plus répandue avec un taux élevé sur toute la zone de sensibilité de la cible. Même observation si deux instructions-sont altérées, mais avec un moindre taux (fig. 4.26b). Avoir trois (fig. 4.26c) ou quatre (fig. 4.26d) instructions en faute est plutôt concentré sur certaines positions de la zone de sensibilité avec un taux supérieur à 50 %.

Il est clair que l'impact sur la ligne 128 bits chargée dans le buffer d'instruction dépend fortement de la position de la sonde, avec un positionnement qui permet de choisir le nombre d'instructions à cibler. Un réglage de l'amplitude de l'impulsion et de l'instant d'injection est alors essentiel pour augmenter la reproductibilité de la faute.

```
0 ADD.w  R1,R1,#0x1
1 ADD.w  R2,R2,#0x1
2 ADD.w  R3,R3,#0x1
3 ADD.w  R4,R4,#0x1
```

(a)

```
0 ADDS  R1,#0x1
1 ADDS  R1,#0x2
2 ADDS  R1,#0x4
3 ADDS  R1,#0x8
4 ADDS  R1,#0x10
5 ADDS  R1,#0x20
6 ADDS  R1,#0x40
7 ADDS  R1,#0x80
```

(b)

FIGURE 4.27 – Séquences de test lors du balayage du boîtier de la cible, quand le bloc d'instruction est composé d'instructions (a) 32 bits et (b) 16 bits.

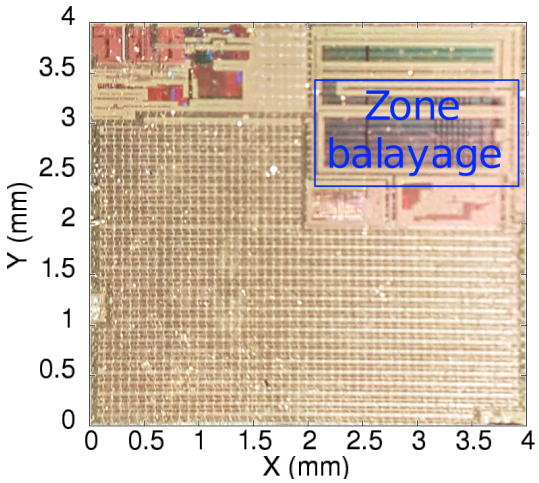


FIGURE 4.28 – Zone de sensibilité identifiée comme zone d'impact sur le chargement de la ligne d'instruction ou de donnée.

Pour la ligne de donnée, le même test a été réalisé avec la séquence de la Figure 4.24 pour le chargement de quatre données à adresses successives dans la mémoire (d_0,d_1,d_2,d_3). Le balayage de la sonde d'injection est effectué avec un pas de 100 µm au dessus de la zone de sensibilité délimité sur la Figure 4.28. Le test a été effectué sur une cible décapsulée avec une amplitude d'impulsion de 90 V. La variation de l'instant d'injection est configurée durant la fenêtre de chargement ΔTd_{fault}. Étant donné que pour la ligne de donnée le modèle de faute observé est *bit-set*, les données à charger sont initialisées à la valeur `00000000`. Cela permet de mieux observer l'impact du rayonnement EM sur le mot de 128 bits chargé.

La Figure 4.29 donne une idée sur la distribution spatiale des fautes dans le cas où une faute de *bit-set* est observé sur une, deux ou quatre données. Une région plus concentrée se dessine dans le cas où les injections EM impactent le chargement de toute la ligne 128 bits de données Figure 4.29c (impact sur quatre données). De même que pour la ligne d'instruction, Nous pouvons conclure que le paramètre spatial joue un rôle important pour cibler un nombre précis de données au sein d'un mot 128 bits. Nous proposons dans l'expérimentation qui suit, une analyse plus détaillée de l'impact de la position de sonde sur l'évolution de la mise à `1` des bits de la ligne de donnée.

En identifiant les coordonnées (Px_{max},Py_{max}) de la position qui remonte le taux le plus élevé de fautes sur les quatre données, on applique alors un balayage linéaire sur

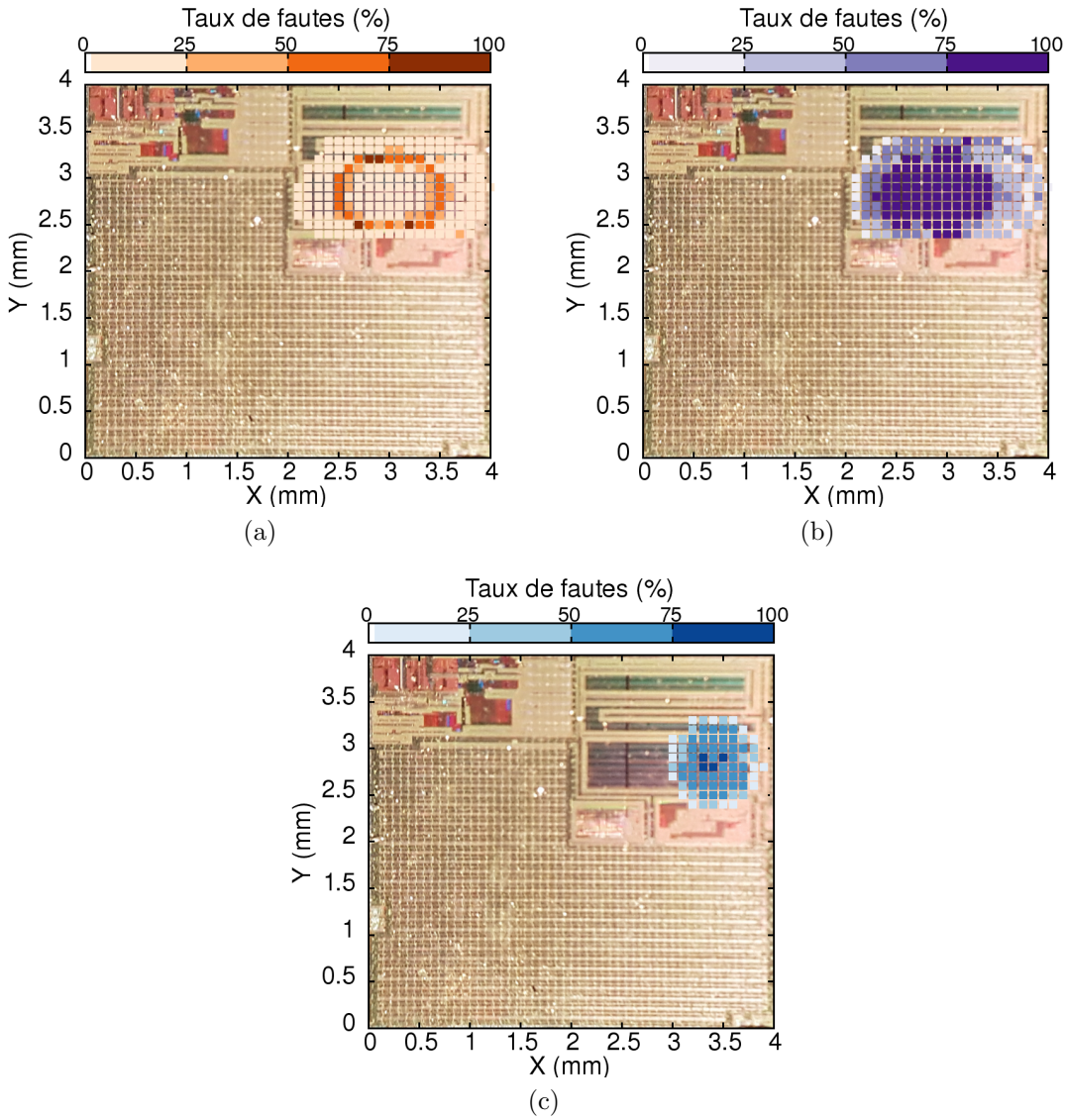
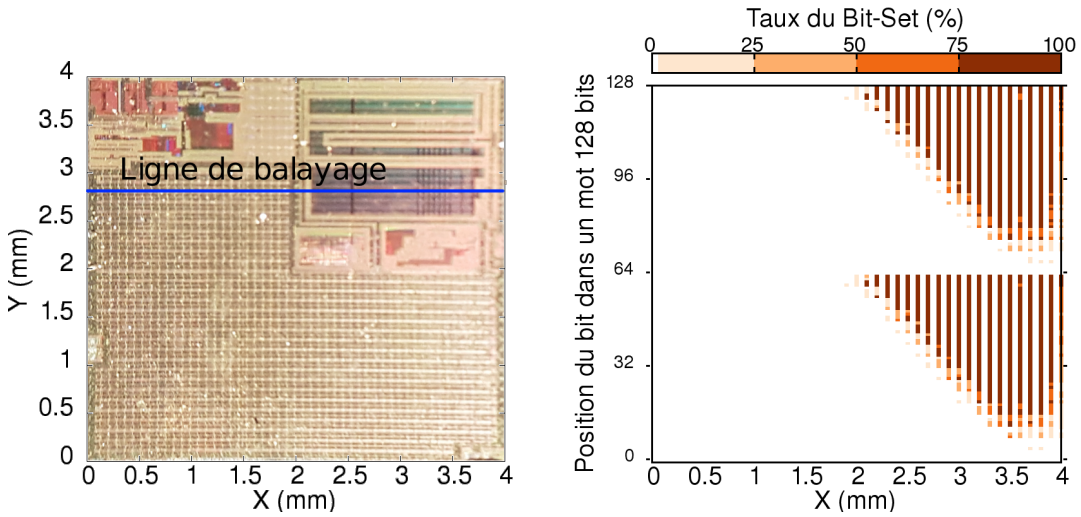


FIGURE 4.29 – Distribution spatiale des fautes lors de chargements de données lorsque une faute est observée sur (a) une seule, (b) deux et (c) quatre données 32 bits.

l'axe X. L'axe Y est fixé à la valeur de Py_{max} 2.8 mm (fig. 4.30a). Cette expérimentation est similaire à celle effectuée dans [53]. Le Tableau 4.2 montre une claire évolution de la mise à `1` des bits de la ligne de donnée en suivant une démarche bien précise. Avec la variation de la position de la sonde sur l'axe X, c'est la donnée d_2 qui est d'abord impactée suivi de d_4. Leurs valeurs en 32 bits sont forcés à `1` progressivement, à partir du bit du poids le plus fort, jusqu'à une position de Px ($2.8\,\text{mm} < Px < 2.9\,\text{mm}$) où la totalité des bits de la valeur binaire des deux données d_2 et d_4 sont forcés à `1` (`FFFFFFFF`).

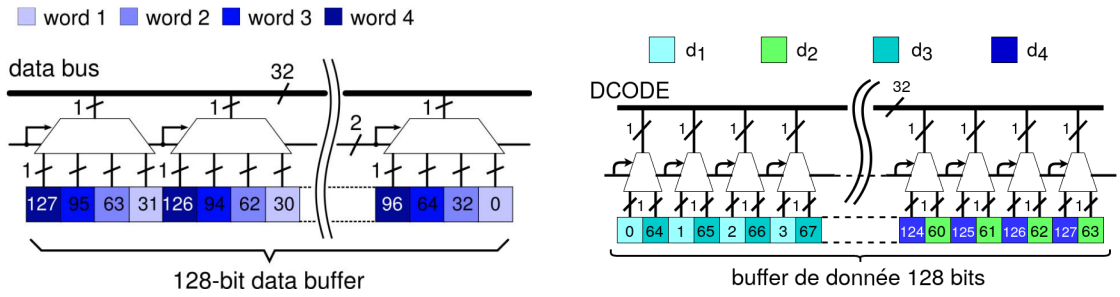
Á partir de la position Px, les 64 bits restant de la ligne de donnée (32 bits pour d_1 et 32 bits pour d_3) sont à leurs tours forcés à `1`. La Figure 4.30b donne une représentation de la position des bits altérés d'une ligne de donnée 128 bits en fonction de la position de la sonde sur l'axe X. Ce comportement nous amène à proposer une hypothèse sur la gestion du chargement d'une ligne de donnée.



(a) (b)

FIGURE 4.30 – Définition de la ligne de balayage (a) sur l'axe X avec (b) la répartition des fautes de type *bit-set* dans une ligne de donnée 128 bits en fonction de la position de la sonde.

Hypothèse sur l'architecture :
Le précédent résultat permet d'identifier la manière dont l'interface mémoire gère la lecture d'une ligne de donnée 128 bit et son écriture sur le *buffer* dédié. En effet, il est clair d'après le comportement observé, que les données ne sont pas chargées dans l'ordre de leurs adresses respectives, mais plutôt deux par deux (64 bits par 64 bits), en commençant par la deuxième et quatrième adresse, suivi par la première et la troisième (fig. 4.31b). L'hypothèse qu'on propose sur la gestion de donnée dans un STM32F407, diffère de peu de celle qui est proposée dans [53] pour le ATSAM3X8 (fig. 4.31a).



(a) (b)

FIGURE 4.31 – L'hypothèse de chargement d'une ligne de donnée 128 bits (a) pour ATSAM3X8 [53] montre le chargement à la fois du même bit des quatre données 32 bits, alors que (b) pour STM32F407 montre le chargement du même bit de deux données par deux (64 bits par 64 bits).

4.5.2 Distribution de fautes dans un bloc 128 bits

Dans le cas d'impact sur plus d'une instruction, nous pouvons observer différentes possibilités en termes de nombre et de distance au sein d'un bloc 128 bits d'instruction. À partir des résultats de fig. 4.26, nous avons analysé la reproductibilité maximale ob-

TABLEAU 4.2 – Évolution de l'effet de *bit-set* sur les données chargées en fonction de la position de la sonde sur l'axe X.

X (mm)	Ligne de Données 128 bits							
	d3		d2		d1		d0	
	127	96	95	64	63	32	31	0
1.9	80000000		00000000		00000000		00000000	
2.0	F0000000		00000000		F0000000		00000000	
2.1	C0000000		00000000		C0000000		00000000	
2.2	F8000000		00000000		F8000000		00000000	
2.3	FF800000		00000000		FF000000		00000000	
2.4	FFF80000		00000000		FFF00000		00000000	
2.5	FFFE0000		00000000		FFFC0000		00000000	
2.6	FFFFF000		00000000		FFFFE000		00000000	
2.7	FFFFFE00		00000000		FFFFFE00		00000000	
2.8	FFFFFFF0		00000000		FFFFFFE0		00000000	
2.9	FFFFFFFF		F8000000		FFFFFFFF		F0000000	
3.0	FFFFFFFF		E0000000		FFFFFFFF		80000000	
3.1	FFFFFFFF		F8000000		FFFFFFFF		F0000000	
3.2	FFFFFFFF		FFFF0000		FFFFFFFF		FFFE0000	
3.3	FFFFFFFF		FFF80000		FFFFFFFF		FFF00000	
3.4	FFFFFFFF		FFFFF800		FFFFFFFF		FFFFF000	
3.5	FFFFFFFF		FFFFFC00		FFFFFFFF		FFFFFC00	
3.6	FFFFFFFF		FFFFFE00		FFFFFFFF		FFFFFC00	
3.7	FFFFFFFF		FFFFE000		FFFFFFFF		FFFFC000	
3.8	FFFFFFFF		FFFF8000		FFFFFFFF		FFFF0000	
3.9	FFFFFFFF		FFFF8000		FFFFFFFF		FFFF0000	

tenue pour avoir une faute sur toutes les combinaisons d'instructions possibles. Nous donnons dans fig. 4.32 les résultats des taux pour corrompre une, deux, trois ou les quatre instructions 32 bits de la ligne d'instruction. Ici encore, nous représentons l'occurrence lorsque seuls les registres ciblés sont impactés. Aussi, nous comparons ces taux avec le cas où le registre `R0` figure parmi les registres dont la valeur a été altérée.

Il y a une forte probabilité d'avoir une faute sur la première et la troisième instruction du bloc ciblé de la séquence de test. Par contre, on observe une faible probabilité d'avoir une faute sur la quatrième, et beaucoup moins sur la deuxième instruction. Toutes les combinaisons de deux instructions peuvent être en faute en même temps, sauf pour la combinaison d'instructions (i_1,i_4). Seules les deux combinaisons de trois instructions (i_1,i_3,i_4) et (i_2,i_3,i_4) peuvent être impactées. La plupart de ces fautes ont un taux encore plus élevé si l'on prend en considération les résultats lorsque la valeur de `R0` est corrompue. En raison du modèle de faute *bit-reset*, le forçage à `0` des bits correspondant à l'opérande du registre de destination d'un opcode, est la cause principale de ce cas de faute.

Pour approfondir notre analyse, nous présentons les résultats issus du cas où la séquence de test est composée d'instructions 16 bits fig. 4.27b. Nous ciblant alors le processus de chargement de huit instructions formant le bloc d'instruction 128 bits. La corruption d'une instruction est déterminée par la valeur du résultat du registre `R1`. Par exemple, suite à un test EMFI, si le valeur finale de `R1` est `0xF7`, cela signifie que la faute produit un saut de l'instruction i_4 `ADDS R1,#0x08`). La Figure 4.33 présente le taux de reproductibilité, du modèle de faute saut d'instruction, pour une seule ou une combinaison d'instructions successives. Lorsqu'on considère une seule instruction altérée, La première et la cinquième instruction 16 bits sont les plus en faute avec respectivement un taux supérieur à 90 % et 50 %.

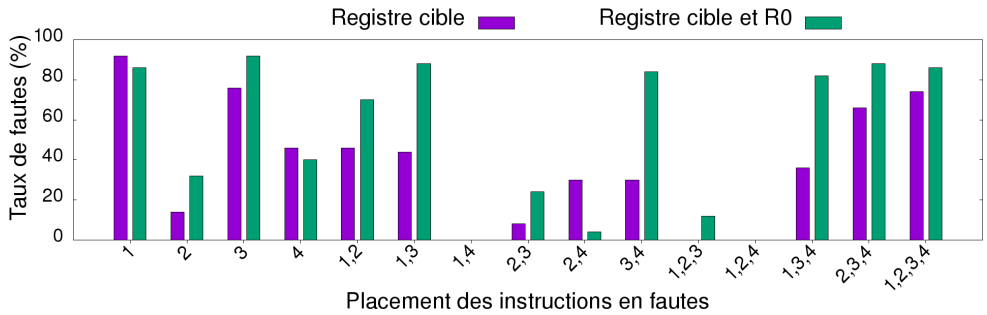


FIGURE 4.32 – Taux de reproductibilité maximum pour altérer les combinaisons d'instruction, avec une comparaison entre le taux des fautes sur les registres cibles Rj ($j \in [1,4]$) et quand R0 est parmi les registres en faute.

Pour les combinaisons de deux à quatre instructions successives, le taux d'observation du modèle de faute est inférieure à 40 %, sauf pour la combinaison d'instructions (1,2) où le taux est supérieur à 60 %. La situation est encore une fois différente lorsque l'on prend en considération les fautes sur la valeur du registre `R0`. On observe alors une augmentation du taux de faute. Cela conforte notre hypothèse sur la façon dont le modèle de faute au niveau logique est lié à la probabilité d'impacter une instruction. En effet, dans le cas du *bit-reset*, il y a tendance à mettre à zéro les bits des opérandes de l'opcode. Cela implique de voir le registre `R0` se retrouver le plus souvent en tant que registre de destination de l'instruction en faute. On pourrait ainsi supposer que dans le cas d'un *bit-set*, le registre `R7` sera le plus impacté pour une instruction 16 bits, et le registre `R12` dans le cas d'une instruction 32 bits.

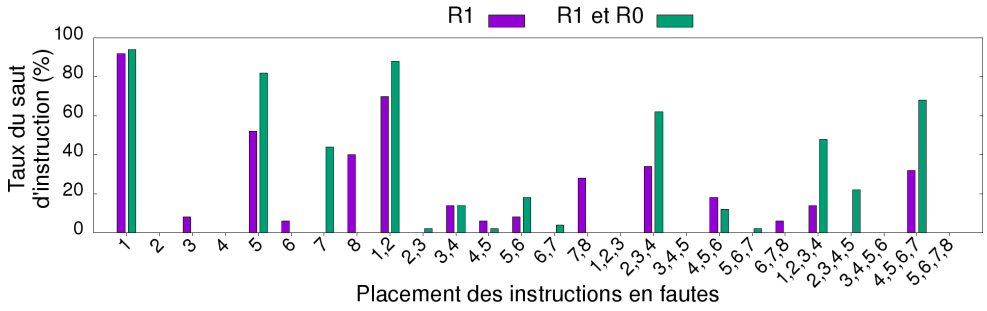


FIGURE 4.33 – Taux de reproductibilité maximum pour le cas du saut d'une ou plusieurs instructions, avec une comparaison entre le taux des fautes sur les registres cibles Rj ($j \in [1,4]$) et quand R0 est parmi les registres en faute.

Dans le cas de faute sur les données, la Figure 4.34 dresse le taux de fautes sur les combinaisons possibles de données chargées. Dans le cas où une seule donnée est altérée, d_2 est la seule à être en faute avec un taux 80 %. On trouve aussi des erreurs sur d_4 mais avec un taux inférieur à 10 %. Aucune erreur n'est remontée pour la donnée d_1 ou d_3. Dans le cas où deux données sont erronées, seul le couple (d_2,d_4) est reporté en faute avec un taux qui atteint les 100 % Seules les combinaisons de trois données (d_1,d_2,d_4) et (d_2,d_3,d_4) pouvant être en faute, avec un taux respectif de moins de 5 % et moins de 40 %. Il est aussi possible d'avoir toute la ligne de donnée 128 bits en faute avec un taux supérieur à 80 %.

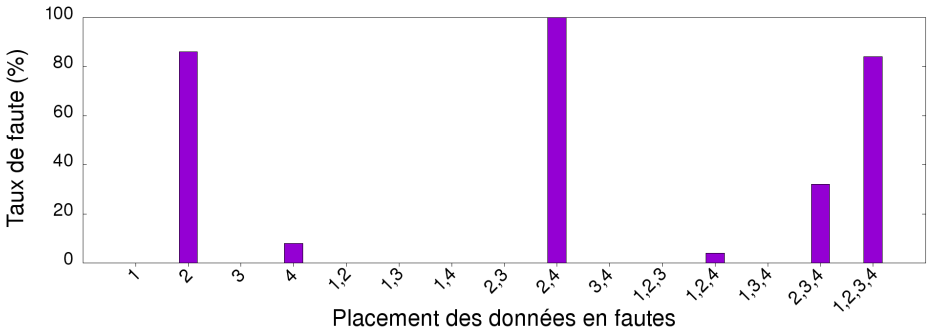


FIGURE 4.34 – Distribution des fautes de *bit-set* sur le chargement d'une ligne de donnée, avec un taux de faute supérieur à 80 % dans le cas d'une seule donnée (d_2), deux données (d_2,d_4) et les quatre données (d_1,d_2,d_3,d_4).

4.6 Analyse de l'impact temporel des fautes

En appliquant la première méthode de la seconde étape de notre démarche d'analyse, nous avons établi que le modèle de faute au niveau bit diffère entre la ligne d'instruction et la ligne de donnée, avec respectivement des fautes en *bit-reset* et *bit-set*. L'analyse s'est limité jusqu'à maintenant à l'observation des fautes durant le chargement d'un seul bloc 128 bits. Dans une séquence de programme plus étendue, il est donc important d'évaluer les conséquences de l'injection de ces fautes, et plus précisément le cas où une même instruction ou donnée altérée est traitée plus d'une fois dans un programme. Dans le cas d'une utilisation récurrente d'une même instruction ou donnée, notre cible offre des fonctions d'optimisation afin d'augmenter les performances du traitement de point de vue temps d'exécution. Dans notre analyse, nous proposons d'étudier l'effet du rayonnement EM sur l'ensemble de ces fonctions (cache d'instruction et cache de donnée). l'étude est présenté sous forme d'une comparaison entre les deux cas activation/désactivation de ces fonctions. En adaptant les séquences de test, nous appliquerons la méthode d'analyse discutée dans section 4.1 pour étudier l'évolution d'une faute sur les instructions et les données.

4.6.1 Impact sur le cache d'instruction

Le cache d'instruction est utile dans le cas où des fonctions sont utilisées plus d'une fois (cas de fonction en loop). Les instructions en cache sont rapidement délivrées au CPU, en cas de multiple demande, sans avoir à les recharger de nouveau depuis la mémoire, ce qui implique des cycles de latence. Il est important de signaler le point sur la récurrence dans la demande d'une instructions. En effet, dans le cas d'un programme totalement séquentiel (aucune instruction n'est traitée plus d'une fois), l'activation du cache d'instruction n'apporte aucun gain au temps d'exécution. Dans ce cas, le recours à un chargement depuis la *Flash* sera toujours effectif. Hors, dans des travaux proposant une analyse de l'impact sur le cache d'instruction [58, 62], les codes de test proposés sont principalement des codes avec le même type d'instruction appelée successivement (sans rappel).

Nous avons de notre côté testé l'impact EMFI sur une séquence qui charge successivement deux blocs d'instructions ayant les mêmes instructions 32 bits, tout en configurant le nombre de WS à un. Premièrement, nous avons observé que le temps

d'exécution, avec ou sans activation du cache, est le même. L'analyse des fautes sur les cycles d'injections corrobore ce résultat, vu que des corruptions de chargement sont reportés pour le premier et le second bloc avec une distance de cinq cycles. Cela confirme donc qu'une séquence de test à instructions séquentielles, n'est pas adaptée pour l'analyse de l'impact sur le cache d'instruction. En effet, comme une instruction est définie par son adresse dans un programme, bénéficier de l'utilité du cache d'instruction revient à rappeler la même instruction (même adresse) au moins une fois durant l'exécution.

Nous proposons un diagramme temporel théorique d'une séquence qui fait appel deux fois à une fonction *bloc_Fct*. Avec le cache désactivé, le chargement de chaque bloc de quatre instructions 32 bits se fait durant les cycles de latence correspondants. Dans notre exemple, *bloc_Fct* (i_8,i_9,i_{10},i_{11}) est chargé deux fois, respectivement durant les cycles C_5 et C_{15}. Avec l'activation du cache, le premier appel de *bloc_Fct* fait que les instructions correspondant sont chargées dans le cache. Lors du second appel, *bloc_Fct* n'est plus chargée de nouveau depuis la *Flash*, mais directement délivré depuis le cache. Contrairement au cas avec le cache désactivé, cela évite donc d'avoir un cycle de latence dans C_{15}. Dans ce cas de figure, le gain en terme de temps d'exécution est égale à un cycle.

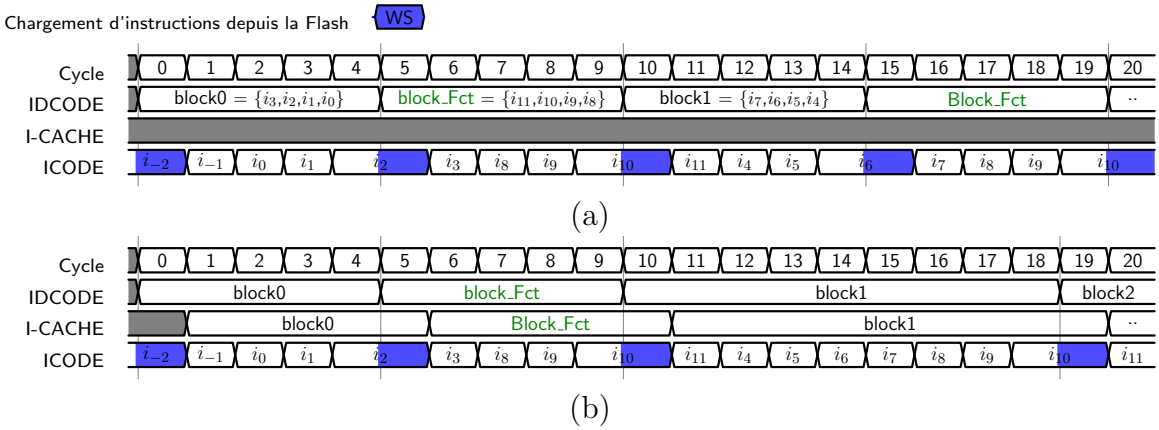


FIGURE 4.35 – Principe de fonctionnement théorique d'un code d'appel de fonction avec le cache d'instruction en mode (a) désactivé et (b) activé.

Sous effet EMFI, et avec l'activation de la fonction cache d'instruction, nous pouvons supposer deux cas d'impact sur la ligne d'instructions. Soit un impact durant le chargement depuis la *Flash* vers le cache (faute d'écriture), ou bien un impact durant le chargement depuis le cache vers le CPU (faute de lecture). Pour vérifier cet impact, notre méthode propose d'appeler deux fois une même fonction (bloc d'instructions). L'injection de fautes durant le cycle du premier chargement nous permettra de vérifier l'évolution de la faute durant l'exécution :

- si le résultat de la première exécution est en faute, alors que le résultat de la seconde est correcte, ceci indique une faute durant la lecture depuis le cache.
- si les résultats de la première et de la seconde exécution est en faute, ceci indique que la corruption est induite durant l'écriture sur le cache.

La séquence de test se résume alors aux étapes suivantes :

1. premier appel de la fonction à tester (l'injection de faute est effectué durant ce premier chargement de la fonction dans le cache).

2. sauvegarde du résultat de la fonction dans un jeu de registre
3. second appel de la fonction
4. sauvegarde du résultat de la seconde exécution dans un autre jeu de registre

La comparaison des deux résultats de l'exécution nous permettra ainsi de déduire si l'impact est durant l'écriture ou la lecture du cache. La Figure 4.36 présente le code de la séquence de test pour l'analyse de l'impact sur le cache d'instruction. La fonction à tester *Fct* est composé de trois opération d'addition, chacune sur un registre défini (`R1, R2, R3`). Deux jeux de registres (`R4, R5, R6`) et (`R7, R8, R9`) sont utilisés pour la sauvegarde respective du résultat de la premier appel (i_0) et du second appel (i_4) de la fonction cible.

```
/*  Trig Haut   */      /* Bloc 0 */     /* Bloc 1 */     /*   Fct   */        /* Inst. NOP */
..                    0 BL.w   Fct     4 BL.w   Fct     8 ADD.w  R1,R1,#0x1    ..
/* Cfg. I-Cache */     1 MOV.w  R4,R1   5 MOV.w  R7,R1   9 ADD.w  R2,R2,#0x2   /* Trig Bas  */
..                    2 MOV.w  R5,R2   6 MOV.w  R8,R2  10 ADD.w  R3,R3,#0x4    ..
/*  Inst. NOP   */     3 MOV.w  R6,R3   7 MOV.w  R9,R3  11 NOP
..                                                     12 BX     LR
```

FIGURE 4.36 – Séquence de test pour l'analyse de l'impact EMFI sur le cache d'instruction.

Sous l'effet des EMFIs, l'analyse des deux jeux de registres de sauvegarde (Tableau 4.3) montre des corruptions d'exécution durant le premier et le second appel de *Fct*. En se basant sur la valeur altérée de `R1` durant le premier appel (`0x40811000` au lieu de `0x00011001`), il est clair que les perturbations EM ont induit un remplacement de l'opération de l'instruction. Cette valeur est aussi le résultat de `R1` pour le second appel, ce qui suggère que l'instruction i_8 (`ADD.w R1,R1,#0x1`) a été remplacée par une instruction idempotente i'_8 dont l'exécution récursive ne change pas le résultat de `R1`.

Une même faute est aussi remonté pour le registre `R3` suite aux deux appels de *Fct*, mais se caractérise plutôt par un saut d'instruction. Il est à noter qu'un test avec la variation de l'instant d'injection durant plusieurs cycles a permis d'identifier un seul cycle qui produit ces fautes (cycle du premier chargement de *Fct*). Du fait qu'on utilise un code qui implique un chargement d'instructions depuis le cache vers le CPU durant le second appel de *Fct*, ce test permet d'observer une corruption du cache. La faute intervient durant le premier chargement de l'instruction cible, et résulte d'une écriture corrompue dans le cache. Tant que le contenu du cache n'a pas été mis à jours, tout rappel de cette instruction reproduira une corruption du résultat, ainsi on est devant un effet de faute **semi-permanente**.

TABLEAU 4.3 – Les résultats du premier et second appel de la fonction *Fct* sont en faute et implique une corruption durant l'écriture sur le cache d'instruction.

Registre	**Valeur Initiale**	1^{er} **Appel**			2^{nd} **Appel**		
		Attendu	Résultat	Modèle	Attendu	Résultat	Modèle
R1	0x00011000	0x00011001	0x40811000	Rempl.	0x40811001	0x40811000	Rempl.
R2	0x00033000	0x00033002	0x00033002	–	0x00033004	0x00033004	–
R3	0x00077000	0x00077004	0x00077000	Saut	0x00077004	0x00077000	Saut

Dans le cas où le cache n'est pas activé, le second chargement de la fonction *Fct* ne se

fait plus depuis le cache, mais depuis la *Flash*. Cela implique qu'une faute durant le premier appel de *Fct* n'a pas d'incidence sur la validité du bloc d'instruction chargé durant le second appel. Dans ce cas, la faute sur l'instruction est définie comme **transitoire**.

4.6.2 Impact sur le cache de donnée

L'utilité du cache de donnée est le même que celui du cache d'instruction, à savoir un gain en temps d'exécution lors de multiple chargement d'une même donnée. Théoriquement, dans le cas d'un cache inactif, si dans une séquence de test il y a deux instructions successives pour le chargement d'une même donnée, on devra observer deux cycles de chargement. Dans le cas de l'activation du cache, un seul cycle de chargement de la donnée est observé, puisque pour le second chargement, la donnée est délivrée depuis le cache. Sauf que le résultat d'un test expérimental n'aboutit pas à ce résultat théorique. En effet, nous avons observé pour une telle séquence de test un seul cycle de chargement de donnée avec ou sans activation du cache. La mesure du temps d'exécution de la séquence donne le même délai et confirme ce comportement. Nous supposons donc que tant que le *buffer* de donnée n'est pas mis à jour, la donnée existante est directement transmise au CPU, si elle est redemandée. Ce cas implique que l'impact sur le cache de donnée ne peut être observé avec ce modèle de séquence de test. Aussi, ce cas de fonctionnement implique une faute **semi-permanente** sur le flot de donnée. Dans les autres cas il sera **transitoire**. Le principe de fonctionnement qui suit propose un exemple de séquence qui met en évidence l'impact sur du cache de donnée.

Nous proposons dans la Figure 4.37 le fonctionnement théorique du système durant un chargement de donnée avec et sans activation du cache de donnée. En cas d'un fonctionnement sans la fonction cache, les cycles C_4, C_6 et C_8 représentent trois opérations de chargement de données 32 bits. Chaque opération charge en réalité 128 bits de données à partir de l'adresse de la donnée demandée. Durant les cycles C_4 et C_8, il y a chargement de la même donnée d_0 (même adresse mémoire). Le chargement effectué durant C_6 est celui d'une donnée d_4 non successive à d_0. Avec ce type de chargement alterné, l'activation de la fonction cache prend son sens. Le deuxième chargement de d_0 n'est plus effectué depuis la *Flash* puisque la donnée est directement communiquée depuis le cache vers le CPU. Le gain en temps d'exécution est ici d'un seul cycle qui est égal au nombre de rappel de la donnée d_0 (un seul rappel).

Au cours d'un test EMFI, la même question s'est posée pour le cache de donnée, à savoir, à quel moment une faute dans le cache est induite et quel est sont effet dans le temps ? La séquence de test pour l'analyse de l'impact sur le cache de donnée s'inspire du fonctionnement théorique expliqué précédemment. La séquence répond aux conditions énoncées pour valider l'utilisation du cache de donnée, à savoir, le rappel d'une même donnée (même adresse mémoire), en utilisant un chargement d'une donnée intermédiaire pour inciter la mise à jour du *buffer* de donnée.

La même méthode appliquée en cas d'activation du cache d'instruction est ainsi suivie, avec une injection de fautes durant le cycle du premier chargement de la donnée cible. Notre méthode d'analyse se résume par la séquence suivante :

1. premier chargement de la donnée cible dans un registre (l'injection de faute est

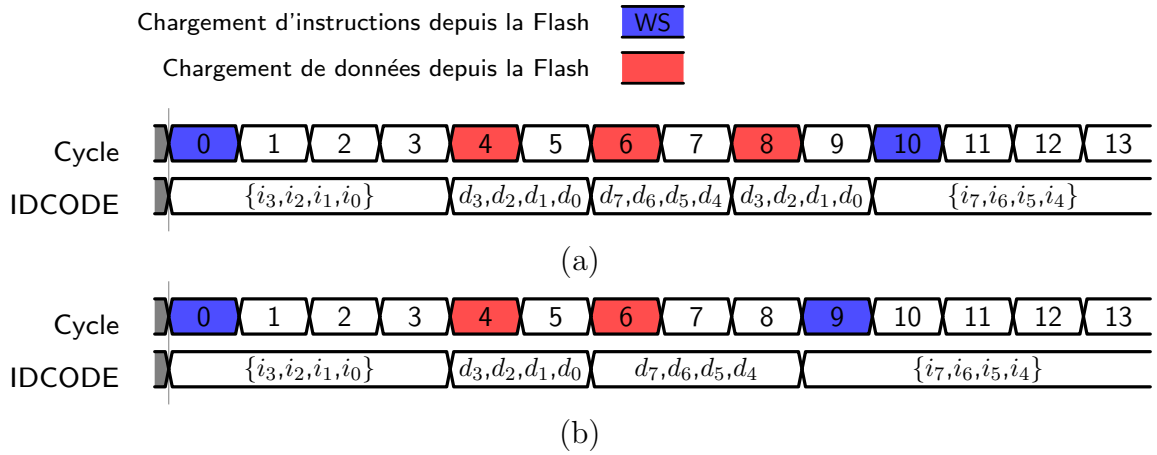


FIGURE 4.37 – Principe de fonctionnement théorique d'un code de chargement de données non successives avec le cache de donnée en mode (a) désactivé et (b) activé.

effectué durant ce premier chargement).

2. chargement d'une donnée intermédiaire dans un second registre
3. second chargement de la donnée cible dans un troisième registre

La Figure 4.38 présente le code de test utilisé pour l'analyse de l'impact EMFI sur le cache de donnée. Les deux chargements de la même donnée d_0 s'effectuent respectivement avec l'instruction i_0 pour le chargement de la donnée sur `R1`, et avec i_2 pour un chargement sur `R3`. Les valeurs des deux registres `R1` et `R3` sont analysées pour identifier l'évolution de la faute sur la ligne de donnée. La mesure du signal du temps d'exécution confirme une différence entre le mode activé et désactivé du cache (ΔTRG_{exe} est égale à un cycle). Lors du chargement de d_0, 128 bits de données sont chargés à la fois à partir de l'adresse de d_0. Les 32 bits relatif à d_0 se trouvent ainsi aux bits du poids le plus faible. Puisqu'on cherche à corrompre la valeur de d_0, la sonde d'injection est placée dans une position qui induit des fautes sur les bits correspondant (Tableau 4.2).

```
/* Seg Trig Haut */
..
/* Cfg. D-Cache */
..
/*   Inst. NOP   */
..
```

```
/* bloc 0 */
0 LDR.w   R1,=d0
1 LDR.w   R2,=d4
2 LDR.w   R3,=d0
3 NOP.w
```

```
/*   Inst. NOP   */
..
/* Seq Trig Bas */
..
```

FIGURE 4.38 – Séquence de test pour l'analyse de l'impact EMFI sur le cache de donnée.

L'analyse des valeurs des registres de sauvegarde (`R1` et `R3`) montre que si une donnée est altérée durant le premier chargement (durant l'injection de faute), elle l'est aussi pour le second chargement Tableau 4.4. Cela confirme donc que l'effet est induit durant l'écriture de la donnée sur le cache. Comme pour le cache d'instruction, cette corruption est **semi-permanente**. En effet, tant que l'adresse du cache relatif à cette donnée n'a pas été mise à jour, tout rappel de la donnée aboutira à un chargement d'une valeur corrompue. Nous notons aussi que comme pour le test sur le cache d'instruction, la

variation de l'instant d'injection n'a pas permis d'induire une faute durant le second chargement de d_0 (qui est effectué depuis le cache), ce qui confirme le principe théorique de la Figure 4.37.

TABLEAU 4.4 – La faute sur le registre R3 (second chargement de la donnée d_0) implique une corruption durant l'écriture de la donnée sur le cache.

Registre	Valeur Initiale	Attendu	Résultat
R1	0x00011000	0x080E0420	0xFF0E0420
R2	0x00033000	0x080E0430	0x080E0430
R3	0x00077000	0x080E0420	0xFF0E0420

4.7 Généralisation de la démarche d'analyse

Les différentes méthodes d'analyse appliquées précédemment ont permis de mettre en évidence l'effet des EMFI sur la stabilité de fonctionnement du MCU STM32F4. Cependant, et afin d'appuyer l'aspect générique de notre démarche d'analyse, nous présentons dans ce qui suit les résultats obtenus à travers l'application de ces méthodes sur un autre MCU, le Atmel SAM4C16C.

4.7.1 Architecture de la cible SAM4C16

Le second MCU qu'on propose d'étudier est le Atmel SAM4C16C. Il se distingue principalement par le fait qu'il intègre deux coeurs 32 bits ARM Cortex-M4. De point de vue architecture, chaque coeur admet un accès privilégié (qui n'implique pas de retard) à une zone mémoire spécifique (fig. 4.39). Un programme à exécuter est placé dans la NVM *Flash* s'il est dédié au coeur principale (CM4P0), alors qu'il sera placé dans la mémoire *SRAM1* s'il est dédié au coeur secondaire (CM4P1). Aussi, ce MCU associe à chaque coeur une fonction cache pour les instructions et les données qui est gérée par un contrôleur dédié. D'autres fonctions d'optimisation de performance sont proposées par un module propriétaire Enhanced Embedded Flash Controller (EEFC), utiles pour définir la méthode d'accès aux instructions et données :

- Configuration du mode d'accès (128 bits ou 64 bits).
- Optimisation de la lecture du flot d'instruction sur deux *buffer*.
- Optimisation de la lecture du flot de donnée sur un *buffer*.
- Optimisation par détection de boucle dans le code.

Le recours aux cycles de latence WS est aussi nécessaire pour ce MCU suivant la fréquence d'horloge du système. Á noter que pour une configuration avec un à trois WS, un cycle de plus est ajouté durant le chargement depuis la mémoire.

En se référant à [110, Fig.22-3], nous présentons dans la Figure 4.40 une hypothèse sur le chronogramme de chargement d'une séquence d'instructions 32 bits $[i_0; i_{11}]$ pour un code exécuté depuis CM4P0. Le principe de traitement d'instruction sur ce MCU est assez similaire à celui du STM32F4. Avec le nombre de WS configuré à zéro, la lecture d'un bloc d'instruction 128 bits se fait durant un seul cycle d'exécution. Nous pouvons

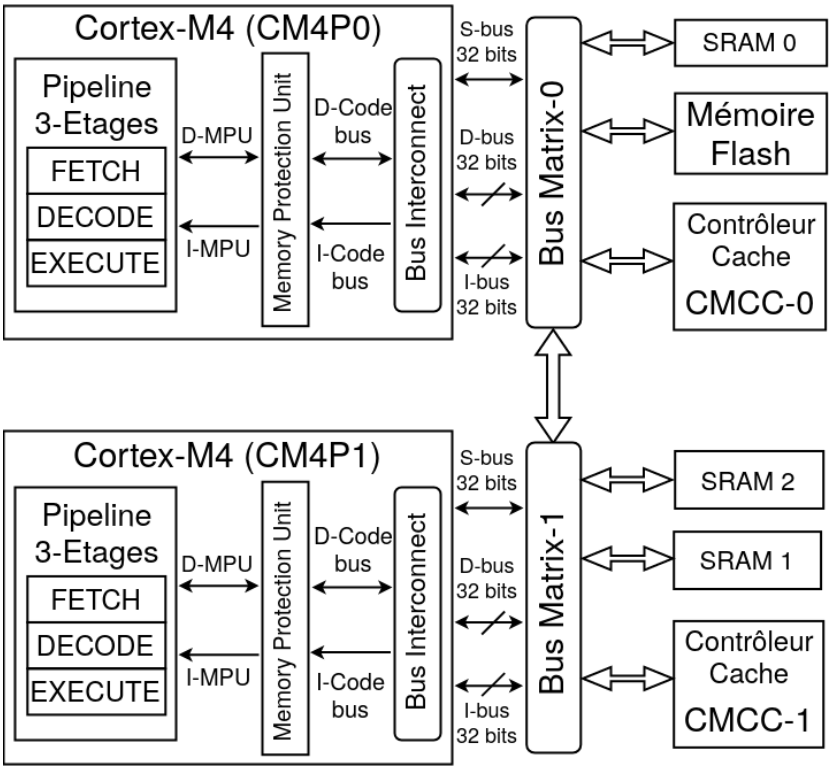


FIGURE 4.39 – Architecture du Atmel SAM4C16.

noter que le chargement de ce bloc se fait en deux temps : Le premier est durant l'accès à la mémoire, tout en délivrant dans I-bus le premier opcode à 32 bits (i_0). Dans un second temps, le bloc de 128 bits est stocké dans le Buffer [0] avec un cycle de décalage pour délivrer les opcodes restant de la séquence au CPU. Cette configuration permet d'illustrer l'opération de lecture d'un flot d'instruction 128 bits depuis la mémoire *Flash* qui s'effectue tous les quatre cycles (C_0, C_4 et C_8).

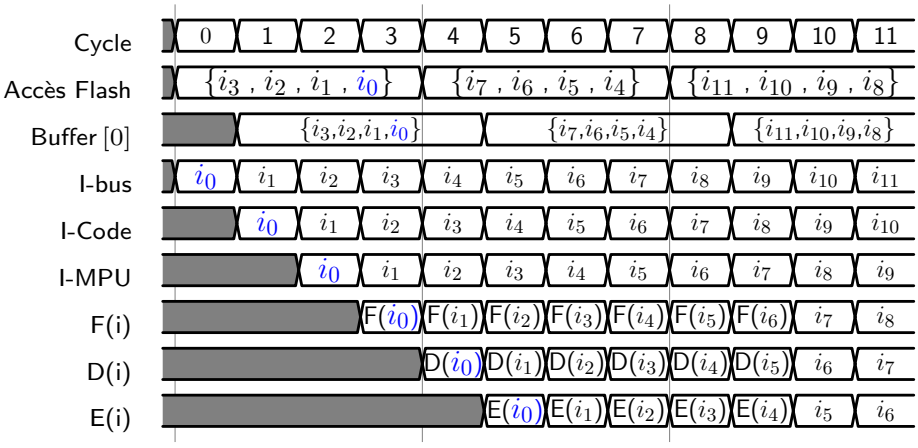


FIGURE 4.40 – Hypothèse sur le diagramme temporel du chargement d'instructions sur le MCU SAM4C16 sans option d'optimisation et avec un nombre de WS égale à zéro.

Les résultats relatifs à la caractérisation avec cette cible sont menés dans le cas où seulement le cœur CM4P0 est actif.

4.7.2 Comparaison des résultats de caractérisation entre SAM4C16 et STM32F4

Sur le SAM4C, nous avons suivi la même démarche d'analyse appliquée sur le STM32F4. Les différents résultats issus des deux caractérisations sont présentés dans le Tableau 4.5.

TABLEAU 4.5 – Comparatif entre les résultats de caractérisation sur STM32F4 et SAM4C16.

Caractérisation		**STM32F4**	**SAM4C**
Élement vulnérable		interface mémoire	interface mémoire
Modèle niveau bit	instruction donnée	*bit-reset* *bit-set*	*bit-reset* *bit-reset*
Effet temporel	cache désactivé cache activé	transitoire semi-permanent	transitoire semi-permanent

Éléments vulnérables de l'architecture :

L'étude de vulnérabilité au niveau architecture est la première étape de notre démarche d'analyse. En utilisant la méthode 1 (fig. 4.11), nous avons analysé l'impact EMFI sur plusieurs cycles d'exécution pour chaque code. Durant ce test, la fréquence de fonctionnement de la cible est configurée à 16 MHz, ce qui implique un nombre de WS égale à zéro. Cette analyse a été effectuée tout en procédant à un balayage de la sonde d'injection sur une zone central du boîtier de la cible (6 mm par 6 mm) et étendu dans un second temps à toute la surface du boîtier(13.5 mm par 13.5 mm). La variation de l'amplitude est configurée entre 100 V et 310 V avec un pas de 10 V. Le résultat du balayage est présenté dans la Figure 4.41 avec les positions qui reportent des fautes de la classe *faute sur registres*.

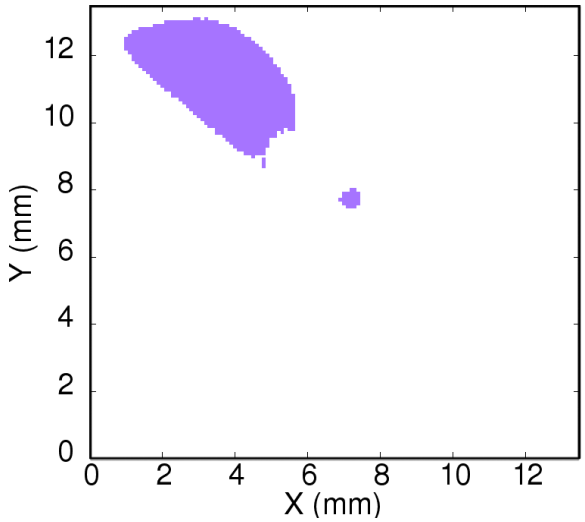


FIGURE 4.41 – Positions de vulnérabilité suite au balayage de la totalité du boîtier du SAM4C avec une amplitude d'impulsion à 155 V.

Pour chaque code testé, les fautes sur le registre `R7` sont observées durant une courte fenêtre de temps pendant un seul cycle. En combinant les résultats, les cycles de vulnérabilité sont par période de quatre cycles. Ce résultat rappel celui pour le STM32F4, qui par l'hypothèse de fonctionnement de la Figure 4.40 nous amène à identifier le EEFC comme étant l'élément vulnérable du SAM4C16. Pour confirmer ce point, nous avons analysé les cycles de vulnérabilité quand la fonction d'optimisation de la lecture du flot d'instruction est activée, et en ayant [110, Fig.22-4] comme référentiel par rapport au fonctionnement.

Pour observer cette fonctionnalité qui est équivalente à la fonction *Prefetch* sur le STM32F4, le nombre de WS a été configuré à un. Aussi, l'activation de la fonction est déclenchée juste avant le chargement du *bloc0*. La Figure 4.42 présente les cycles de vulnérabilité observés pour une séquence de test avec plusieurs instructions successives de type `ADD.w R7,R7,#1`. Ce test est réalisé sur une position fixe de la sonde et avec une amplitude d'impulsion à 310 V. Suivant les deux cas, avec et sans l'option d'optimisation, le fonctionnement suit la logique définie par le constructeur, à savoir l'observation du début d'alternance entre buffer [0] et buffer [1] lors de l'opération de chargement des blocs d'instruction.

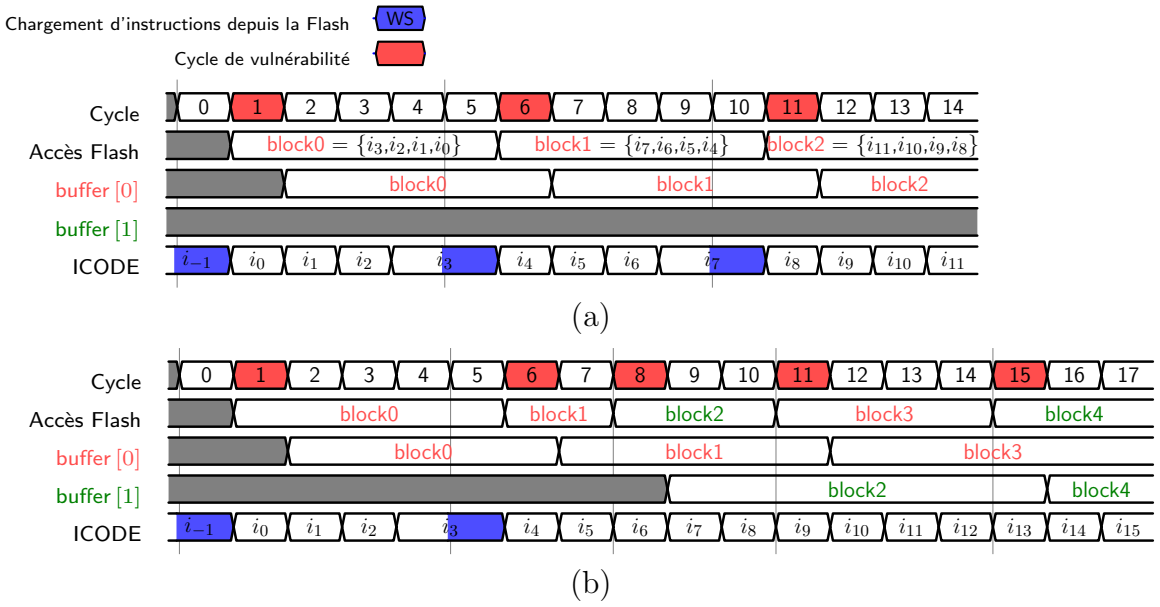


FIGURE 4.42 – Identification des cycles de vulnérabilité avec la fonction d'optimisation de lecture sur le flot d'instruction (a) désactivée et (b) activée.

Un troisième test vient appuyer nos constatations, avec la modification du mode de lecture en 64 bits au lieu de 128 bits. Le résultat de ce test montre que les cycles de vulnérabilité sont observés tous les deux cycles dans le cas où aucune optimisation n'est activée et un nombre de WS configuré à zéro.

Pour le flot de donnée, et comme pour le STM32F4, nous observons un cycle de vulnérabilité de plus propre au chargement des données. Aussi, nous retrouvons les mêmes mécanisme de chargement par rapport aux données à adresses successives ou non. Ces observations sont en quelque sorte sans surprise puisque les deux cibles

intègrent le même type de cœur ARM.

Sur le plan temporel, nous retrouvons un effet similaire quant à l'évolution d'une faute dans un programme. En effet, les EMFI affectent le chargement depuis la mémoire vers le cache avec des faute semi-permanentes tant que l'adresse de la donnée ou de l'instruction cible dans le cache n'a pas été mise à jour. Pour le SAM4C, il faut néanmoins au préalable reconfigurer le fichier de lien pour que le code (instructions et données) soit affecter à une zone mémoire de la *Flash* qui peut bénéficier de la fonction cache.

En appliquant notre première méthode d'analyse, nous avons identifié que la principale vulnérabilité du SAM4C16 se situe au niveau du contrôleur de mémoire EEFC. De point de vue vulnérabilité sur l'architecture, ce résultat est similaire à celui du STM32F4.

Modèles de faute au niveau bit :

L'élément vulnérable de l'architecture étant identifié, nous avons appliqué la deuxième méthode qui permet de dresser les modèles de faute au niveau bit. Pour cela, nous avons étudié l'effet EMFI suivant les différentes séquences de test définies pour l'analyse sur la ligne d'instruction et la ligne de donnée.

Sur le flot d'instruction, nous avons analysé l'impact EMFI durant l'exécution des trois séquences de la Figure 4.20. Des fautes sont observées seulement quand le bloc d'instruction chargé est composé d'opcodes dont la plupart des bits de leur valeur binaire sont à `1` (fig. 4.20b). On retrouve ici une similitude avec le le STM32F4, puisque seul le modèle *bit-reset* est observé sur les instructions.

Les séquences de test de la Figure 4.24 et Figure 4.25 sont utilisées pour déterminer le modèle de faute sur le flot de donnée. Le choix de la valeur à charger (`FFFFFFFF` ou `00000000`) est définie pour chaque test afin d'observer un modèle de faute précis.

Quand on cible le cycle de chargement des données, les EMFI ne sont effectives que seulement si la valeur binaire des données chargées est tout à `1` (`FFFFFFFF`). Ce résultat signifie qu'au niveau bit, les EMFI engendrent un effet de *bit-reset* sur le flot de donnée. On constate ici une différence avec le modèle *bit-set* observé sur le STM32F4. Une des hypothèse sur la cause de cette différence peut s'expliquer par le fait que les deux cibles sont issues de constructeurs différents, ce qui implique un procédé de fabrication différent.

Influence de la variation de l'amplitude de l'impulsion EM :

Durant les expérimentations précédentes, les différents paramètres EMFI sont sujets à des variations pour déterminer tous les possibles effets engendrés par les injections. Pour le cas du STM32F4, c'est principalement le paramètre spatial qui est déterminant sur le nombre d'instruction en faute dans un flot d'instruction ou encore la distribution des fautes en *bit-set* sur la valeur d'une donnée. En ce qui concerne la cible SAM4C16, les différents résultats montrent que c'est plutôt le choix de la valeur de l'amplitude de l'impulsion générée qui favorise l'observation d'un type de faute.

En effet, il est possible d'identifier certains seuils d'amplitude où on observe un changement dans la distribution des fautes dans un flot d'instruction 128 bits. En fixant la position de la sonde dans une position de sensibilité au centre du boîtier, nous avons varié l'amplitude de l'impulsion de 150 V à 300 V avec un pas de 10 V et analysé l'impact sur le code de la Figure 4.27a.

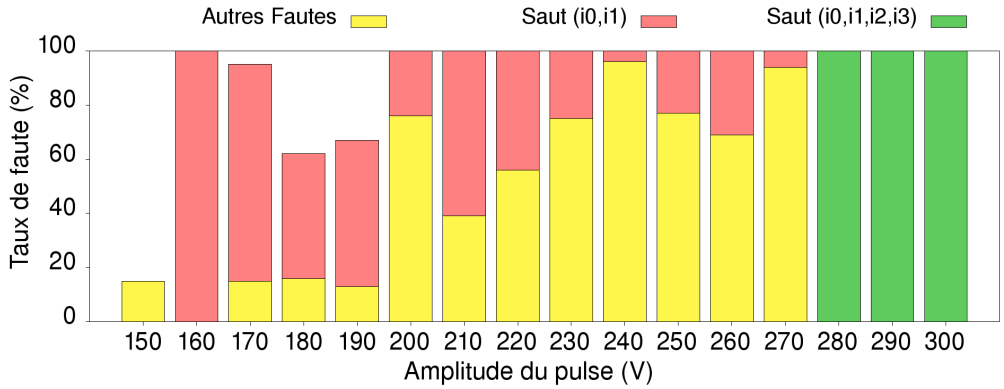


FIGURE 4.43 – Effet de la variation de l'amplitude de l'impulsion EM sur le SAM4C pour produire le saut de deux ou quatre instructions dans un même flot d'instruction 128 bits.

Pour le cas du modèle de faute saut d'instruction, la Figure 4.43 montrent l'effet de l'amplitude sur le saut d'un nombre défini d'instructions. Le même test sur d'autres positions donne le même résultat à savoir que les EMFI engendrent soit le saut de deux instructions (i_0,i_1), soit le saut des quatre instructions du flot 128 bit d'instruction et cela en fonction de l'amplitude d'impulsion configurée.

Sur le flot de donnée, l'effet de la variation de l'amplitude est aussi présent surtout par rapport à la façon dont EEFC gère le chargement des données. En reprenant la séquence de test de la Figure 4.24 comprenant le chargement de quatre données à adresses successives dans la mémoire, nous initialisons cette fois ces données avec la valeur `FFFFFFFF` afin de mieux observer l'effet *bit-reset*. En ciblant le cycle de chargement relatif aux données, la variation de l'amplitude nous donne une idée sur le mécanisme de chargement d'un mot de 128 bits par l'interface mémoire du SAM4C.

Le Tableau 4.6 donne l'évolution de l'effet *bit-reset* sur chacune des données 32 bits chargées en fonction de la variation de l'amplitude de 120 V à 270 V avec un pas de 2 V. Il est clair d'après ce résultat que l'augmentation de l'amplitude force plus de bits à `0`, avec à 252 V on obtient une remise à zéro totale de l'ensemble des bits du mot 128 bits chargé. Contrairement à l'analyse sur le STM32F4, il nous est difficile dans le cas du SAM4C (fig. 4.44) d'établir un schéma précis de la façon dont le mécanisme de chargement des données fonctionne.

4.7.3 Caractérisation sur différentes plateformes EMFI

Avec la disponibilité de deux plateformes d'injection EM P_{Ke} et P_{Av} (utilisant respectivement les générateurs d'impulsion *Keysight* et *Avtech*), nous nous sommes posé la question sur une éventuelle différence dans le résultat de la caractérisation.

TABLEAU 4.6 – Évolution de l'effet de *bit-reset* sur les données chargées en fonction de l'amplitude de l'impulsion.

Amplitude (V)	**Ligne de Données 128 bits**							
	d3		d2		d1		d0	
	127	96	95	64	63	32	31	0
120	FBFFFFFF		FBFFFFFF		FFFFFFFF		FFFFFFFF	
128	FBFFFFFF		FBFFFBFF		FFFFFFFF		FFFFFFFF	
136	FBFFFF7F		FBFFFBFF		FFFFEFFF		FFFFFFEF	
144	FBFFFF5F		EBFFF2FF		FBFFE77F		FBFFFF4F	
152	FBFFEF7F		E9FFF2F7		FBBFA77F		F9FFFD4F	
160	F1FF437F		E96F42D7		E3FF807F		F19FF44F	
168	31FF037F		C06F4207		E1FE803F		F19FF44B	
176	20FF006F		402F4607		C1FA003B		609FF403	
184	20FF006F		000F4005		8178003A		401FF003	
192	00BF000F		04070005		0078001A		001FA003	
200	00BD000D		00070001		0078001A		000E8003	
208	00BC080D		00070001		0070001A		000E8003	
216	003C000D		00070001		00500010		000E0003	
222	00380008		00030001		00500000		00060003	
230	00280008		00030001		00500000		00060001	
238	00280000		00030001		00000000		00060001	
246	00000000		00000000		00000000		00040001	
254	00000000		00000000		00000000		00000000	
262	00000000		00000000		00000000		00000000	
270	00000000		00000000		00000000		00000000	

Le diagramme des deux plateformes est représenté dans la Figure 4.45 et Figure 4.10 respectivement pour P_{Ke} et P_{Av}.

Les tests sont réalisés en utilisant la même cible STM32F4. Pour ne pas apporter plusieurs changements qui rendraient la comparaison difficile à établir, la seule différence entre les deux plateformes est le générateur d'impulsion. De point de vue IEM, le générateur *Keysight* a été configuré de la même manière que le *Avtech* à savoir une largeur d'impulsion à 6 ns et un temps de monté de 2.5 ns . L'amplitude de l'impulsion est configurée à 0 dBm et qui est par la suite amplifié à l'aide d'un amplificateur qui peut atteindre jusqu'à 260 W de puissance.

La Figure 4.46 donne un aperçu sur les formes d'onde EM récupérées sur un oscilloscope quand la sonde LIRMM F [40] est placée sur une ligne microruban. La première observation est par rapport à la forme d'onde : Les rebonds sont assez réguliers (fig. 4.46b) à la sortie du *Avtech*, alors qu'ils sont plutôt irréguliers (fig. 4.46a) en utilisant le *Keysight*. Nous supposons que cette différence vient principalement de l'adaptation et la réponse de l'amplificateur dont l'entrée est connectée à la sortie du générateur *Keysight*. Aussi, avec le maximum d'amplitude configuré sur le *Keysight* (0 dBm), l'amplitude récupérée sur le microruban est égale à celle à la sortie du *Avtech* quand il est configuré à 100 V. Sachant que tous les autres équipements et paramètres restent inchangés, l'onde EM est pratiquement le seul point de différence entre les deux plateformes.

Le Tableau 4.7 présente un comparatif entre les résultats présentés précédemment sur le STM32F4 avec P_{Av} et ceux issus des tests avec P_{Ke}. Avec ce dernier, nous avons obtenu la même zone d'impact de point de vue distribution spatial des fautes. Nous n'avons pas réussi à perturber un autre élément autre que l'interface mémoire, ce qui confirme qu'on induit des fautes sur l'élément le plus vulnérable de l'architecture de cette cible.

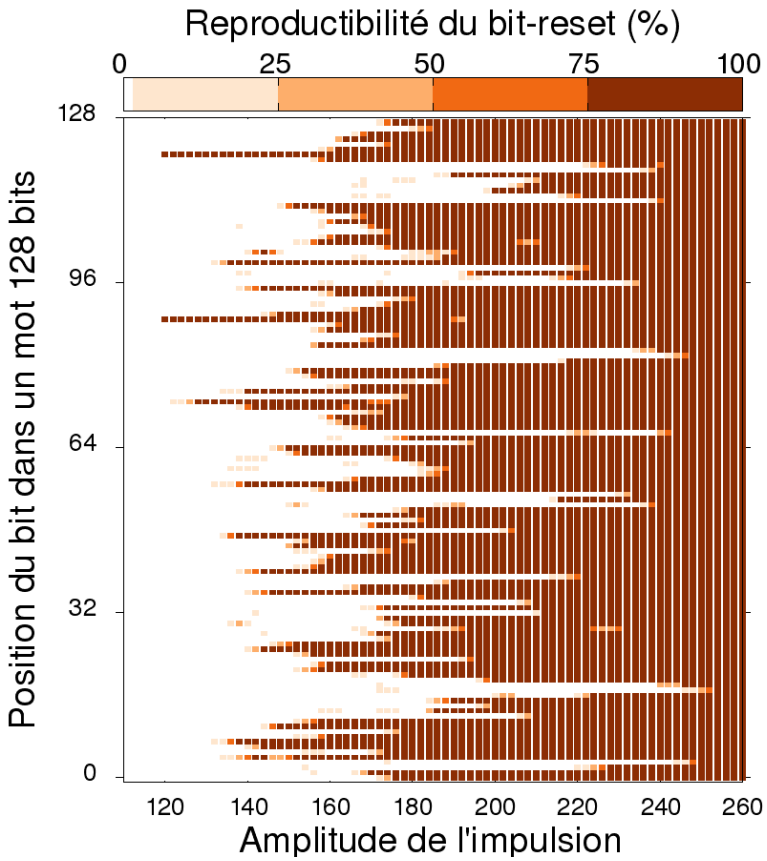


FIGURE 4.44 – Répartition des fautes en *bit-reset* dans un buffer de donnée 128 bits sur le SAM4C en fonction de l'amplitude de l'impulsion.

Pour l'analyse au niveau bit, nous avons observé une différence dans l'impact sur les instructions. Alors qu'avec le P_{Av} nous avons identifié un effet de *bit-reset*, l'utilisation du P_{Ke} a permit d'obtenir le modèle *bit-set* comme effet sur le flot d'instruction. Puisque sur une même position nous pouvons avoir deux modèles différents en changeant de générateur d'impulsion, ce résultat confirme que l'observation d'un modèle de faute n'est pas une question de puissance seulement, mais que la forme d'onde de l'impulsion est aussi un paramètre à prendre en considération. Contrairement au flot d'instruction, les deux plateformes reportent le même modèle *bit-set* au niveau bit sur le flot de donnée. L'effet temporel reste aussi équivalent entre P_{Ke} et P_{Av} puisque les fautes sont toujours créées durant l'opération d'écriture depuis la mémoire vers le cache.

TABLEAU 4.7 – Comparaison des résultats de caractérisation sur STM32F4 en utilisant P_{Ke} et P_{Av}.

Caractérisation		Plateforme EMFI P_{Av}	Plateforme EMFI P_{Ke}
Élement vulnérable		interface mémoire	interface mémoire
Modèle niveau bit	instruction	*bit-reset*	*bit-set*
	donnée	*bit-set*	*bit-set*
Effet temporel	cache désactivé	transitoire	transitoire
	cache activé	semi-permanent	semi-permanent

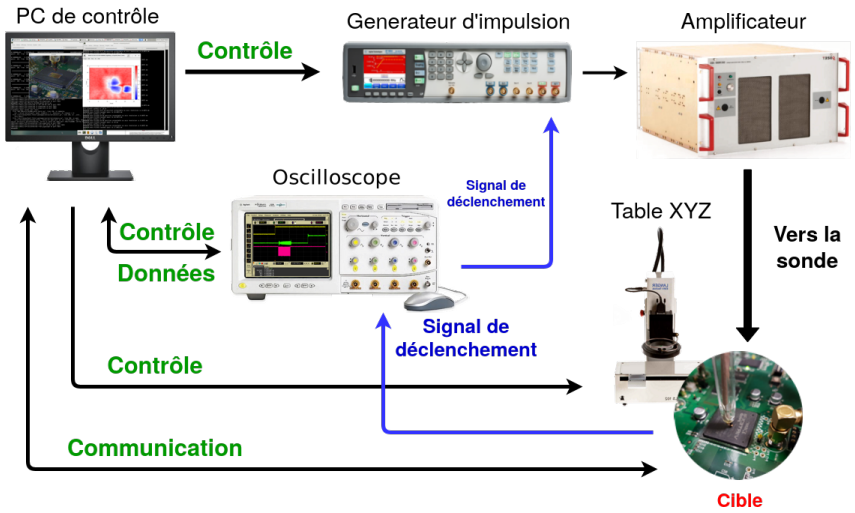
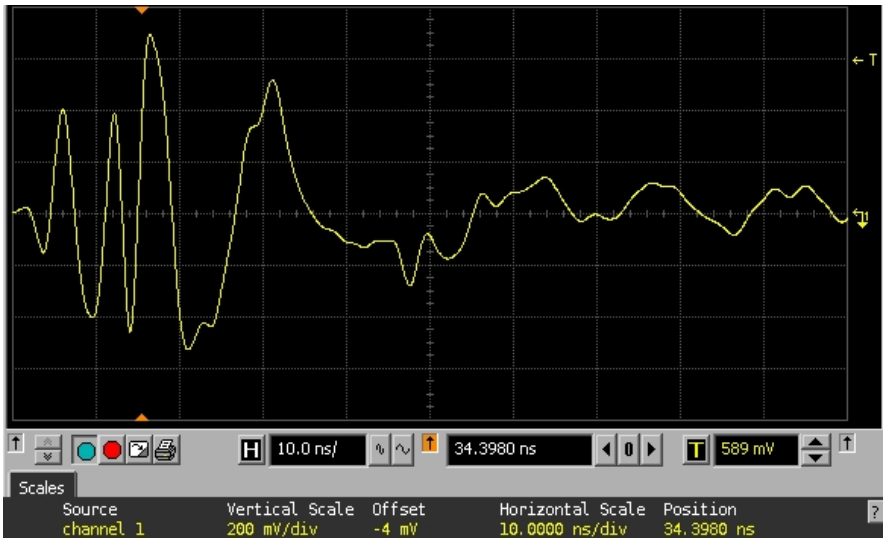
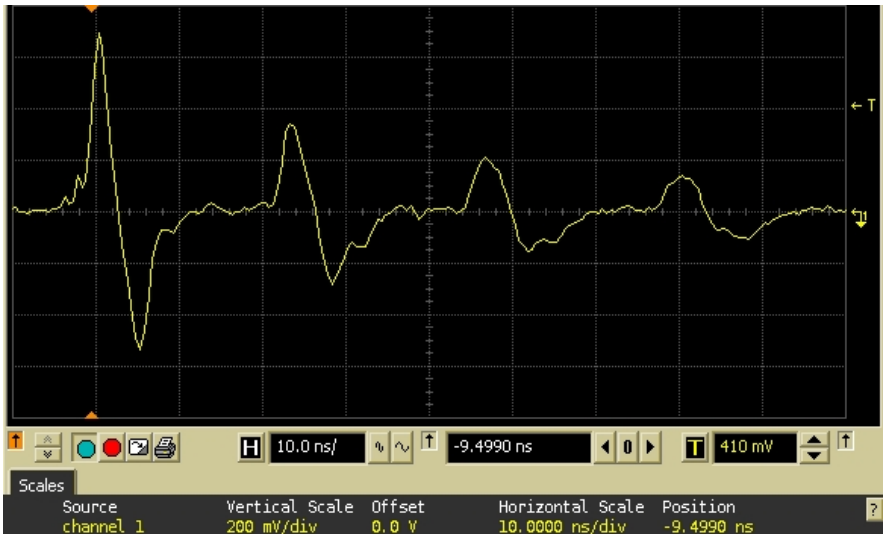


FIGURE 4.45 – Diagramme de la plateforme EMFI avec générateur Keysight pour les expérimentations sur les MCU.

La caractérisation en utilisant une autre plateforme EMFI valide en elle même notre démarche d'analyse, puisqu'avec une même méthode, en l'occurrence la méthode 2, nous avons pu observer deux résultats différents sur une même cible. Un possible contrôle de la forme d'onde d'une IEM s'avère intéressant à étudier dans le sens où on peut choisir d'induire le modèle *bit-set* ou *bit-reset* selon le résultat souhaité.



(a)



(b)

FIGURE 4.46 – Forme d'onde d'une impulsion EM mesurée sur ligne microruban générée par (a) P_{Ke} et (b) P_{Av} .

4.8 Conclusion

Faisant suite à l'analyse au niveau logique du précédent chapitre, une démarche de caractérisation au niveau logiciel a été présentée permettant de mettre en évidence l'impact des EMFI sur des cibles de type MCU et de dresser les différentes vulnérabilités. Contenu du manque d'approche clair dans l'état de l'art, l'établissement de méthodes dédiées à l'étude et l'observation d'un effet précis se révèle nécessaire pour mieux comprendre la propagation des fautes ou encore leurs causes. Formulées sous deux étapes successives, trois méthodes génériques sont élaborées pour l'étude des vulnérabilités. Les expérimentations sont alors effectuées sur deux différents MCU de différents constructeurs pour appuyer l'aspect générique de la démarche.

Avec la première méthode il été possible de distinguer avec certitude l'élément sensible de l'architecture qui s'avère être pour les deux MCU le contrôler mémoire. Outre l'analyse au niveau architecture, la méthode apporte plus détails quant au fonction-

nement des MCU de part leur traitement du flot d'instruction ou de donnée. Cela a permis de mieux appréhender les modèles de faute avec une meilleur adaptation de la configuration des tests de caractérisation.

Compte tenu du fait que les modèles de faute au niveau instruction ou la corruption de données sont principalement dues à une modification au niveau bit, la deuxième méthode propose d'identifier le modèle induit par les EMFI à ce niveau. Pour une analyse sur les deux flots, la méthode se base sur l'adaptation des séquences de test avec un choix adéquat d'opcode ou la valeur de donnée de sorte à maximiser l'observation des modèles comme le *bit-set*, *bit-reset*, *bit-flip* ou encore le *no-sampling*.. La précision de la méthode est démontrée puisque sur chaque cible, différents effets sont observés au niveau bit en exécutant une même séquence de test.

La troisième méthode vise à définir la propagation de la faute durant l'exécution d'un programme. Elle s'appuie sur différentes conditions sur les instructions et les données afin de définir une séquence de test dédiées et établir l'effet temporel des fautes. Ces fautes se trouvent être transitoires quand le buffer d'instruction ou de donnée est mis continuellement à jour ou quand la fonction de cache est désactivé. Cependant, elles sont semi-permanentes quand le cache est activé et tant que son contenu n'a pas été mis à jour.

Tout au long de l'application de ces méthodes, la variation des paramètres d'injection est essentielle pour dresser tous les possibles effets dus aux EMFI. L'effet de ces variations sur des séquences de test adaptées, a permit de déterminer le nombre d'instructions ou de donnée en faute et leur distribution dans leur flot respectif. Ces tests ont aussi révélés l'utilité d'étudier le maximum de paramètres durant la caractérisation de l'effet des EMFI. C'est ainsi que pour la cible STM32F4, le paramètre spatial est celui qui est déterminant dans la distribution des fautes dans la ligne d'instruction ou de donnée, alors que pour la seconde cible SAM4C, c'est plutôt l'amplitude de l'impulsion qui est le paramètre déterminant.

Enfin, une comparaison des résultats de caractérisation effectuée sur deux plateformes avec différents générateurs d'impulsion vient souligner l'importance d'avoir des méthodes génériques et dédiées à l'étude des vulnérabilités. Les résultats ont montrés que sur une même cible, il est possible d'obtenir deux effets opposés au niveau bit même à faible puissance. La différence entre les deux plateformes réside dans la forme d'onde de l'impulsion EM qui en ressort comme un autre paramètre déterminant dans l'observation des modèles de faute.

Sur les instructions comme sur les données, il est donc envisageable en ayant ces différents détails sur les modèles de faute de cibler de façon précise une partie d'un code et de le corrompre de manière contrôlée. Vu les résultats de cette caractérisation au niveau logiciel, une question précise se dégage à savoir l'impact des modèles de faute identifiés sur la robustesse des contre-mesures logicielles. C'est le sujet du prochain chapitre qui aborde ce point de manière théorique dans un premier temps, suivi par une analyse expérimentale pour consolider nos observations.

CHAPITRE 5

Vulnérabilités des contre-mesures logicielles

Sommaire

Au niveau logiciel, les précédents résultats montrent l'efficacité d'une seule EMFI à engendrer des fautes durant l'opération de chargement depuis la NVM. Jusqu'à quatre instructions 32 bits ou huit instructions 16 bits peuvent être altérée durant le chargement d'un bloc d'instructions 128 bits. De même, la corruption est effective durant le chargement d'un bloc de 128 bits de donnée équivaut quatre données 32 bits chargées depuis des adresses mémoire successives. Les fautes sur les deux flots se sont même avérées semi-permanentes dans le cas d'utilisation de la fonction cache.

L'importance de ces résultats repose sur l'efficacité d'une EMFI à cibler de une à plusieurs instructions ou données avec une seule injection de faute. Ce modèle de faute met à défaut l'hypothèse formée par les contre-mesures qu'on retrouve dans l'état de l'art. La majorité de ces contre-mesures ne prennent en considération que les injections de faute de premier ordre avec un effet que sur une seule instruction ou donnée.

Dans ce chapitre, nous proposons une analyse fonctionnelle de la robustesse des contre-mesures basées sur la redondance d'instruction, et particulièrement celles proposées par [84] et [85]. L'efficacité de ces contre-mesures est discutée en tenant compte des modèles de faute obtenu dans le Chapitre 4. Dans notre analyse, nous évoquerons seulement le chemin de d'attaque le plus court (nombre minimum d'instructions à altérer). Aussi, bien que ces contre-mesures sont présentées pour protéger principalement contre les saut d'instruction, nous détaillerons aussi l'impact sur le flot de donnée. Compte tenu des failles observées, nous proposerons des améliorations pour garantir

une protection face au cas de plusieurs sauts d'instructions ou une corruption de plus d'une donnée.

5.1 Protection par duplication d'instruction

5.1.1 Analyse de résistance

La méthode de protection par duplication proposée dans [84] intègre un mécanisme de redondance associé à celui de détection de faute. Pour protéger une instruction, la contre-mesure nécessite quatre instructions comme présenté dans l'exemple fig. 5.1a.

Les instructions i_0 et i_1 représentent la duplication de l'instruction à protéger `LDR.w R4,[R7]`. Ici, la redondance se traduit par une deuxième instruction avec un registre de destination différent `LDR.w R12,[R7]`. Le résultat de la comparaison dans i_2, de la valeur des deux registres de destination `R4` et `R12`, sert à identifier un saut de i_0 ou i_1. En cas de valeur non nulle de ce résultat, l'instruction i_3 redirige le programme vers la routine *erreur* pour signaler la détection de la faute.

Cette méthode est bien robuste dans le cas d'un seul saut. En effet, le saut de i_0 ou i_1 est détecté avec i_2, et aboutira à une redirection du programme avec i_3. Mais comme notre modèle de faute offre la possibilité d'altérer plus d'une instruction, la robustesse de cette contre-mesure est discutable.

```
/* inst. cible */
LDR.w R4,[R7]
/* Duplication */
---- bloc0 ----
0 LDR.w R4,[R7]
1 LDR.w R12,[R7]
2 CMP.w R12,R4
3 BNE.w <erreur>
```

(a)

```
/* 1ere Amélioration */
---- bloc0 ----
0 inst. libre
1 inst. libre
2 inst. libre
3 LDR.w R4,[R7]
---- bloc1 ----
4 LDR.w R12,[R7]
5 CMP.w R12,R4
6 BNE.w <erreur>
7 inst. libre
```

(b)

```
/* 2eme Amélioration */
---- bloc0 ----
0 inst. libre
1 inst. libre
2 LDR.w R4,[R7]
3 LDR.w R12,[Rx]
---- bloc1 ----
4 LDR.w R12,[R7]
5 CMP.w R12,R4
6 BNE.w <erreur>
7 inst. libre
```

(c)

FIGURE 5.1 – Instruction cible (a) avec la contre-mesures par duplication [84], et les propositions d'améliorations (b) sur la ligne d'instruction et (c) sur la ligne de donnée.

Dans notre exemple, le *bloc0* (bloc d'instruction de 128 bits) représente les quatre instructions formant la contre-mesure. En cas d'injection de faute durant le cycle de chargement du flot d'instruction, différents chemins d'attaque peuvent se présenter. Une des fautes exploitables sur les instructions, est de produire un saut des deux instructions (i_0,i_2). Le saut de l'instruction à protéger (non mise à jour de `R4`) et le saut du mécanisme de détection (pas de fonction de comparaison) fait que l'erreur ne sera pas détectée et évitera une redirection vers la routine *erreur*. Un autre chemin plus simple, est de produire le saut de toutes les instructions du *bloc0* (i_0,i_1,i_2 et i_3)

En ce qui concerne les fautes exploitables sur les données, le chemin de faute considéré est de corrompre le premier chargement de `[R7]` en i_0. Les résultats de la caractérisation ont montré que si des instructions successives chargent la même donnée, cette opération se fait en un seul cycle d'exécution. Avec notre modèle de faute sur

la ligne de donnée, cela implique que `R4` et `R12` seront chargés avec la même valeur erronée. En conséquence, le résultat de la comparaison de `R4` et `R12` est valide, mais la faute est non détectée.

Ces deux cas de figure de faute montrent bien les problèmes de robustesse de ce type de contre-mesure face aux fautes à sauts multiples,induits par une seule injection de faute durant un cycle d'exécution. Dans [85], une évaluation a déjà pointée ces problématiques. Il est néanmoins possible de rendre cette contre-mesure efficace contre ces modèles de faute.

5.1.2 Proposition d'améliorations

En tenant compte de nos modèles de fautes sur cette contre-mesure par duplication, de potentielles failles sur les instructions et sur les données ont été relevées. Plus précisément, notre analyse montre que la transformation requise par la contre-mesure rend les fautes possibles. Ainsi, lorsque les instructions redondantes se trouvent dans le même flot d'instruction de 128 bits, il est facile de contourner la protection. Aussi, notre modèle de faute sur les données implique qu'une seule faute durant le chargement de la première donnée impact de la même manière l'opération redondante. Nous proposons dans ce qui suit, une amélioration à deux étapes destinée à rendre cette contre-mesure plus sûre sur les instructions et les données.

La première amélioration concerne la protection du mécanisme de redondance. Notre méthode se base essentiellement sur la répartition de la contre-mesure sur plusieurs blocs (équivaut plusieurs chargements). Ainsi, on évite l'alignement des instructions sur le même bloc 128 bits à charger. Dans le cas d'une attaque de premier ordre, cette nouvelle configuration de la contre-mesure oblige un attaquant à cibler un seul bloc d'instruction. Comme résultat, le saut d'une ou plusieurs instructions d'un même bloc n'affectera pas les fonctionnalités de la contre-mesure. En effet, étant donné que l'instruction à protéger et sa redondante se trouvent dans deux blocs différents, la protection reste fonctionnelle.

Pour le cas de la contre-mesure par duplication [84], la nouvelle décomposition en plusieurs blocs de chargement est représentée dans la Figure 5.1b. Avec une seule injection de faute, les sauts ne sont possible que sur un seul flot d'instruction :

- un ou plusieurs sauts dans *bloc0* → valeur de `R4` erronée mais détectée dans *bloc1*.
- un ou plusieurs sauts dans *bloc1* → valeur de `R4` reste correcte.

Ainsi, quel que soit le nombre de saut dans un bloc, deux résultats sont possibles, à savoir la détection de la faute ou un résultat correcte. La décomposition en blocs de la contre-mesure renforce donc la robustesse face à la corruption du chargement d'une ligne d'instruction.

Notre deuxième amélioration consiste à ajouter un mécanisme qui prend en considération la corruption de donnée. L'analyse de la robustesse a montrée qu'injecter une faute durant le premier chargement de la donnée à protéger affecte de la même manière le chargement redondant. Pour éviter cette propagation de la faute, nous proposons d'insérer un chargement intermédiaire qui éliminera la propagation. Le but de ce chargement est de mettre à jour le *buffer* de donnée et inciter l'interface mémoire à charger de

nouveau la donnée depuis la mémoire. Une condition est néanmoins liée à la donnée intermédiaire, est que cette dernière n'est pas incluse dans le premier chargement 128 bits de données. Ce principe du chargement de donnée intermédiaire est le même principe discuté précédemment dans l'analyse temporelle des fautes sur le flot de donnée. Pour cette contre-mesure par duplication, nous proposons d'inclure un chargement d'une donnée intermédiaire `[Rx]` (fig. 5.1c). La mise à jour du *buffer* de donnée, entre le premier (i_2) et le second (i_4) chargement de `[R7]`, évite d'avoir une même corruption de donnée sur `R4` et `R12`.

Cette méthode par répartition en plusieurs blocs, engendre bien évidement un surcoût dans le programme. Dans nos propositions d'améliorations, ce surcoût est défini comme étant des placements d'instructions libres (`inst. libre`). Nous pouvons imaginer une modification du compilateur binaire, de manière à réorganiser le programme en plaçant d'autres instructions dans ces adresses libres (similaire à [86]). Cela nous permet d'avoir des améliorations qui apportent une meilleure robustesse à la contre-mesure, sans engendrer un surcoût considérable. Aussi, l'amélioration sur le flot de donnée n'est requise que si l'instruction à protéger admet une opération sur les données.

5.1.3 Étude expérimentale de la protection par duplication

Nous proposons dans cette partie de tester expérimentalement l'efficacité de la contre-mesure avec et sans amélioration. Les tests sont menés sur le MCU SAM4C16 et en utilisant la plateforme P_{Av}. Les résultats issues de la caractérisation sur ce MCU ont établi l'effet des EMFI à engendrer des sauts d'instructions suivant un réglage de l'amplitude de l'impulsion. Pour ces tests, l'amplitude de l'impulsion est configuré à 310 V, ce qui permet d'avoir le saut de quatre instructions d'un même bloc 128 bits. Á cette amplitude, tous les bits d'un flot de donnée sont mis à zéro.

Les deux codes protégés de la Figure 5.1a et la fig. 5.1c, respectivement la contre-mesure par duplication de [84] sans et avec notre amélioration, sont analysés sous effet des EMFI. Pour les deux codes, nous ciblons le cycle de chargement du *bloc0* ainsi que le cycle du premier chargement de la donnée `[R7]`. La fonction *erreur* se résume à assigner la valeur `0x5A5A5A5A` au registre `R9` dans le cas d'une détection de la faute. Pour l'amélioration sur la protection du flot de donnée, le chargement intermédiaire se fait avec la donnée `[R6]` (`0xF0F11F0F`) qui se trouve dans une adresse mémoire non successive à `[R7]`. Le Tableau 5.1 présente une comparaison des résultats du test EMFI sur les deux codes.

Pour la contre-mesure sans amélioration, l'injection de faute durant le chargement du bloc d'instruction *bloc0* provoque le saut des quatre instructions de la protection : saut de l'instruction à protéger i_0, sa duplication i_1, l'instruction de comparaison i_2 et la fonction de branche sur routine *erreur* i_3. Le registre `R4` n'est pas mis à jours avec la bonne valeur et reste donc avec sa valeur initiale. En l'absence de la fonction de détection, cette faute ne sera pas pas signalée. Une injection de faute durant le chargement de la donnée `[R7]` fait que les deux registres de destination `R4` et `R12` sont chargés avec la même valeur erronée `0x00000000`. Ici encore cette faute n'est pas détectée puisque le résultat de la fonction de détection est correcte (caleur de `R4` est bien égale à la valeur de `R12`).

TABLEAU 5.1 – Effet des sauts multiples sur la contre-mesure par duplication de [84] avec et sans améliorations.

				Injection de faute	
				Chargement bloc0	**1^{er} chargement** $[R7]$
Duplication	**Registre**	**Valeur Initiale**	**Attendu**	**Résultat**	**Résultat**
sans amélioration	R4	0x0FF00000	0xF010010F	0x0FF00000	0x00000000
	R12	0x03300000	0xF010010F	0x03300000	0x00000000
	R9 (erreur)	0x00000000	0x00000000	0x00000000	0x00000000
			Faute	(non détecté)	(non détecté)
avec amélioration	R4	0x0FF00000	0xF010010F	0x0FF00000	0x00000000
	R12	0x03300000	0xF010010F	0xF010010F	0xF010010F
	R9 (erreur)	0x00000000	0x00000000	0x5A5A5A5A	0x5A5A5A5A
			Faute	détectée	détectée

La contre-mesure améliorée évite bien la non détection de ces fautes. En effet, une injection de faute durant le chargement du *bloc0* induit une faute que sur le registre `R4`, mais le *bloc1* reste fonctionnel et la faute est bien détectée avec la mise à jour de `R9` avec la valeur `0x5A5A5A5A`. Aussi, une faute durant le premier chargement de $[R7]$ sur `R4` a pour effet le signalement de la faute suite au résultat de la comparaison entre `R4` et `R12`.

Le résultat de cette expérimentation montre d'une part l'inefficacité d'une telle contre-mesure dans une application réelle dans le cas des sauts multiples avec une seule injection de faute. D'autre part, l'amélioration qu'on propose rend cette contre-mesure 100% sûre contre ce modèle de faute, et apporte aussi une robustesse sur la protection du flot de donnée.

5.2 Protection par triplication d'instruction

La contre-mesure par triplication proposée dans [84] suit le même principe de celui pour la duplication. Sauf que dans cette méthode, l'instruction cible est répliquée trois fois, en utilisant donc trois registres de destinations différents pour la sauvegarde du résultat. Aussi, en plus de la fonction de détection, elle intègre une fonction de correction de faute. La protection de l'instruction `LDR.w R4,[R7]` avec cette méthode est représentée dans la Figure 5.2a. Comme on peut le constater, et dans le cas d'instructions 32 bits et un bus de chargement à 128 bits, cette méthode est répartie sur pas moins de quatre blocs d'instructions. Le *bloc0* assure la fonction de triplication, les *bloc1* et *bloc2* la détection de faute, et le *bloc3* la correction. Le programme est redirigé vers la routine *erreur* seulement si les trois registres de destination se retrouvent avec des valeurs différentes. Cela équivaut le cas de saut d'au moins deux instructions des trois utilisées pour la triplication (i_1, i_2 et i_3). La correction de la valeur de `R4` est assuré avec (i_{12},i_{13}) seulement si un saut de l'instruction à protéger (i_3) est détecté. Ainsi, cette méthode propose une correction de la faute dans le cas d'un seul saut (instruction cible), et le signalement de la faute si plus.

Comme mentionné précédemment, cette méthode de protection par triplication est déjà répartie sur plusieurs blocs d'instruction, ce qui correspond déjà à notre méthode d'amélioration. Cependant, un chemin d'attaque précis met à mal cette contre-mesure. Le saut de tout le *bloc0* provoque les états suivants :

```
/* inst. cible */
LDR.w R4,[R7]
/* Triplication */
---- bloc0 ----
0 EOR.w R12,R12,R12
1 LDR.w R10,[R7]
2 LDR.w R0,[R7]
3 LDR.w R4,[R7]
---- bloc1 ----
4 CMP.w R4,R10
5 EOREQ.w R12,R12,#1
6 CMP.w R4,R0
7 EOREQ R12,R12,#2
---- bloc2 ----
8 CMP.w R10,R0
9 EOREQ.w R12,R12,#4
10 CMP.w R12,#0
11 BEQ.w <erreur>
---- bloc3 ----
12 CMP.w R12,#4
13 MOVEQ.w R4,R0
```

(a)

```
/* 1er Amélioration */
---- bloc0 ----
0 inst. libre
1 inst. libre
2 LDR.w R10,[R7]
3 LDR.w R0,[R7]
---- bloc1 ----
4 EOR.w R12,R12,R12
5 LDR.w R4,[R7]
6 CMP.w R4,R10
7 EOREQ.w R12,R12,#1
---- bloc2 ----
8 CMP.w R4,R0
9 EOREQ.w R12,R12,#2
10 CMP.w R10,R0
11 EOREQ.w R12,R12,#4
---- bloc3 ----
12 CMP.w R12,#0
13 BEQ.w <erreur>
14 CMP.w R12,#4
15 MOVEQ.w R4,R0
```

(b)

```
/* 2eme Amélioration */
---- bloc0 ----
0 LDR.w R10,[R7]
1 LDR.w R4,[Rx]
2 LDR.w R0,[R7]
3 LDR.w R4,[Rx]
---- bloc1 ----
4 EOR.w R12,R12,R12
5 LDR.w R4,[R7]
6 CMP.w R4,R10
7 EOREQ.w R12,R12,#1
---- bloc2 ----
8 CMP.w R4,R0
9 EOREQ.w R12,R12,#2
10 CMP.w R10,R0
11 EOREQ.w R12,R12,#4
---- bloc3 ----
12 CMP.w R12,#0
13 BEQ.w <erreur>
14 CMP.w R12,#4
15 MOVEQ.w R4,R0
```

(c)

FIGURE 5.2 – Instruction cible (a) avec la contre-mesures par triplication [84] et la propositions d'amélioration (b) sur les instructions (c) sur les données.

— les trois registres de destination restent avec leurs valeurs initiales.
→ le résultat de toutes les fonctions de test est négatif

— le registre contrôle `R12` n'est pas initialisé à zéro au début de la séquence.
→ la détection de faute avec i_{10} n'est plus garantie

L'effet de cette faute est que `R4` se retrouve avec une valeur erronée qui est ni détectée ni corrigée par la contre-mesure. Aussi, si nous ciblons le cycle de chargement de donnée, on se retrouve avec la même faille rencontrée dans la méthode par duplication. Une injection de faute durant le chargement de `[R7]` fait que les trois registres de destination seront chargés avec la même donnée erronée.

Pour rendre ces fautes détectables par la contre-mesure, on propose une amélioration en deux étapes (fig. 5.2b). Premièrement, il faut garantir que le registre de contrôle et les instructions redondantes ne se retrouvent pas dans un même flot d'instruction. Cette condition permet :

— Si registre de contrôle en faute → l'instruction cible est protégée

— Si l'instruction cible ou ses redondantes sont en faute → le registre de contrôle est remis à zéro et la fonction de détection effective

Aussi, pour renforcer la protection, l'instruction à protéger et celle qui servira à corriger la faute ne doivent pas figurer dans le même flot. Dans notre exemple, cela garantira d'avoir une donnée dans `R4` qui n'est pas égale à celle de `R0` en cas de faute.

Deuxièmement, et dans le cas où on cherche à protéger le flot de donnée, on propose la même méthode avancée dans la précédente contre-mesure qui consiste à ajouter un chargement de donnée intermédiaire. Ce principe élimine donc une propagation de la

faute sur les données et oblige un attaquant à ne pouvoir cibler qu'un seul chargement dans le cas d'une attaque de premier ordre. Contrairement à la contre-mesure par duplication, et comme la redondance est par triplication, on aura besoin de deux chargements intermédiaires qui seront placés entre chaque chargement de `[R7]`. Avec le deuxième chargement intermédiaire, on assure d'exécuter une nouvelle opération de chargement depuis la mémoire à chaque fois qu'une donnée est demandée.

Étude expérimentale de la protection par triplication : Nous proposons ici de tester l'effet des EMFI sur la contre-mesure par triplication avec et sans application des améliorations. Le test est réalisé aussi sur le MCU SAM4C16, en appliquant une amplitude d'impulsion de 310 V. Pour le code de la Figure 5.2a (sans améliorations), nous donnons les résultats quand on cible le chargement du *bloc0* ainsi que le chargement de la première donnée `[R7]` durant l'exécution de i_1. Pour le code de la Figure 5.2c (avec améliorations), nous présentons en plus le résultat si on cible le chargement du *bloc1*. Dans le cas de l'amélioration sur les données, nous testons aussi l'effet de faute sur le deuxième chargement de `[R7]` (i_2). La routine *erreur* est la même que précédemment, à savoir assigner la valeur `0x5A5A5A5A` au registre `R9`. Les chargements intermédiaires sont appliqués en utilisant la donnée `[R6]` dans les instructions i_1 et i_3.

TABLEAU 5.2 – Résultats du test EMFI sur une séquence protégée avec la contre-mesure par triplication de [84] avec et sans améliorations.

				Injection de faute durant le chargement		
				bloc0	1^{er} *[R7]*	**bloc1**
Triplication	**Registre**	**Valeur Initiale**	**Attendue**	**Résultat**	**Résultat**	**Résultat**
sans amélioration	R4	0x0FF00000	0xF010010F	0x0FF00000	0x00000000	–
	R10	0x03300000	0xF010010F	0x03300000	0x00000000	–
	R0	0x07700000	0xF010010F	0x07700000	0x00000000	–
	R12	0xFFFFFFFF	0x00000007	0xFFFFFFF8	0x00000007	–
	R9 (erreur)	0x00000000	0x00000000	0x00000000	0x00000000	–
			Faute	(non détectée)	(non détectée)	–
avec amélioration	R4	0x0FF00000	0xF010010F	0xF010010F	0xF010010F	0xF010010F
	R10	0x03300000	0xF010010F	0x03300000	0x00000000	0xF010010F
	R0	0x07700000	0xF010010F	0x07700000	0xF010010F	0xF010010F
	R12	0xFFFFFFFF	0x00000007	0x00000000	0x00000006	0x00000004
	R9 (erreur)	0x00000000	0x00000000	0x5A5A5A5A	0x00000000	0x00000000
			Faute	détectée	pas de faute	corrigée

Le résultat du test sur la contre-mesure sans amélioration (Tableau 5.2) prouve encore fois l'incapacité de la protection à détecter les sauts multiples (saut de toutes les instructions du *bloc0*) . Plus précisément, comme le registre de contrôle `R12` n'a pas été initialisé avant sa mise à jour, le résultat de la comparaison en i_{10} ne redirige pas vers la routine *erreur* en cas de faute sur les trois registres de destination. D'un autre coté, la faute durant le premier chargement de `[R7]` fait que tous les trois registres de destination `R4, R0 et R10` se trouvent avec la même valeur `0x00000000`, dû à l'effet *bit-reset* sur le flot de donnée. Par conséquent, dans ce cas aussi le mécanisme de détection ne signale pas l'erreur, et le registre `R4` reste avec une valeur corrompue dans la suite du programme.

Ces failles sont bien corrigées avec la version améliorée de la contre-mesure. Le saut du *bloc0* provoque la détection de faute même si `R4` est bien chargé avec la bonne valeur dans le *bloc1*. Avec le saut du *bloc1*, `R4` se retrouve avec la valeur de la donnée

intermédiaire `0xF0F11F0F` (i_3), qui est par la suite corrigée dans i_{15} puisque les registres de redondance `R0 et R10` sont chargés avec la bonne valeur.

Le test sur le flot de donnée reflète aussi l'efficacité de notre amélioration. Injecter une faute durant le premier ou le second chargement de `[R7]` fait que `R4` se trouvera toujours avec la bonne valeur `0xF010010F`. Une faute durant le troisième chargement de `[R7]` (i_5) aura pour conséquence la correction de la faute sur `R4` avec i_{15}. Notre amélioration sur la contre-mesure par triplication ajoute donc une protection de plus sur le flot de donnée et rend la protection plus sûre sur le flot d'instruction.

Aussi, ces deux précédentes contre-mesures ne sont pas applicables pour tous les types d'instructions. Elle sont principalement utilisées pour protéger des instructions idempotentes. Nous analysons dans la suite l'impact de nos modèles de faute sur des contre-mesures appliquées sur les différents types d'instructions.

5.3 Contre-mesures par duplication résiliente

Moro et al. proposent dans [85] une extension aux précédentes contre-mesures en tenant compte de tous types d'instructions. Quand une instruction est idempotente, la contre-mesure consiste à appliquer une redondance simple (exécuter deux fois la même instruction). Ainsi, cela diffère de la méthode de [84] puisqu'il n'y a pas d'utilisation d'un registre intermédiaire pour la sauvegarde du résultat de l'opération dupliquée. Aussi, la méthode de [85] n'intègre aucune fonction de détection ou de correction de faute et est exclusivement destinée à contrer un seul saut d'instruction.

5.3.1 Cas des instructions non-idempotentes

Dans le cas des instructions non-idempotentes, la méthode se résume à décomposer l'instruction cible en instructions idempotentes tout en les dupliquant. Dans la fig. 5.3a, nous présentons l'exemple de l'instruction non-idempotente `LDR.w R1,[R1]` et sa protection avec la méthode de [85]. Cette instruction est décomposée en deux instructions idempotentes `MOV.w R4,R1` et `LDR.w R1,[R4]`, qui sont elles-mêmes par la suite dupliquées. La contre-mesure est donc composée de quatre instructions, formant un bloc d'instruction qui nécessite un seul cycle de chargement.Contenu de la précédente analyse, ce type de configuration n'est donc pas robuste contre le modèle de faute observé lors de notre caractérisation.

La non présence d'une fonction de détection de saut dans cette contre-mesure, fait que les combinaisons de faute sont plus nombreuses dans le cas de plusieurs sauts. De point de vue instruction, le chemin d'attaque le plus court est d'avoir un saut des instructions redondantes (i_0,i_1) ou (i_2,i_3). Dans le cas de chargement de donnée, on se retrouve devant la même faille sur les données, discutée pour les contre-mesures de [84]. Sauf que cette méthode de protection ne peut être améliorée pour prendre en compte les fautes sur les données, sans l'ajout d'un mécanisme de détection. Les fautes sur les données (injection de faute durant les cycles de chargement de donnée) ne seront pas pris en considération dans la suite.

Proposition d'améliorations : Le principe reste le même pour une meilleur robustesse de cette protection. La protection de chaque instruction est nécessaire afin

```
/* Non-idempotente */
LDR.w R1,[R1]
/* Remplacement */
---- bloc0 ----
0 MOV.w R4,R1
1 MOV.w R4,R1
2 LDR.w R1,[R4]
3 LDR.w R1,[R4]
```

(a)

```
/* Amélioration */
---- bloc0 ----
0 MOV.w R4,R1
1 inst. libre
2 inst. libre
3 inst. libre
---- bloc1 ----
4 MOV.w R4,R1
5 inst. libre
6 inst. libre
7 inst. libre
---- bloc2 ----
8 LDR.w R1,[R4]
9 inst. libre
10 inst. libre
11 inst. libre
---- bloc3 ----
12 LDR.w R1,[R4]
13 inst. libre
14 inst. libre
15 inst. libre
```

(b)

FIGURE 5.3 – Protection d'intruction non-idempotente avec (a) la contre-mesures [85] et (b) en utilisant la version améliorée pour les instructions.

de garantir la fonctionnalité de la contre-mesure. Chaque instruction doit donc se retrouver dans un flot de chargement distinct (fig. 5.3b). Cette configuration offre une robustesse totale contre le saut de n'importe quelle instruction, moyennant une seule injection de faute. En effet, le saut de n'importe quel bloc de la séquence ne génère aucune conséquence sur le résultat, qui sera toujours correcte. Notons encore que, injecter une faute durant le dernier chargement de la donnée `[R4]` ($i_1 3$) provoque une corruption sur le résultat de `R1`.

Analyse expérimentale : Nous présentons dans le Tableau 5.3 le résultat du test des deux codes de la Figure 5.3a et Figure 5.3b sous effets des EMFI. C'est sans surprise que le saut de tout le bloc d'instruction de la contre-mesure non améliorée rend cette protection inefficace. En revanche, aucune faute sur la version améliorée n'est effective puisque le saut d'un seul bloc n'a pas de conséquence sur la redondance.

TABLEAU 5.3 – Résultats du test EMFI sur une séquence protégée avec la contre-mesure pour instructions non-idempotentes de [85] avec et sans améliorations.

				Injection de faute durant le chargement		
				bloc0	**bloc1**	**bloc2**
Non-Idempotente	**Registre**	**Valeur Initiale**	**Attendue**	**Résultat**	**Résultat**	**Résultat**
sans amélioration	R4 R1	0x0FF00000 0x01000360	0x01000360 0xF010010F	0x0FF00000 0x01000360	– –	– –
			Faute	(non protégée)	–	–
avec amélioration	R4 R1	0x0FF00000 0x01000360	0x01000360 0xF010010F	0x01000360 0xF010010F	0x01000360 0xF010010F	0x01000360 0xF010010F
			Faute	protégée	protégée	protégée

5.3.2 Cas des instructions spécifiques

Il existe des instructions dont la décomposition en instructions idempotentes nécessite plusieurs étapes. C'est le cas par exemple des instructions de branchement avec retenu de l'adresse de retour `BL`. Dans la Figure 5.4a, nous présentons un exemple de protection de cette instruction comme proposé dans [85].

La décomposition de cette instruction comprend trois instructions idempotentes, avec en plus, l'ajout d'une référence pour l'adresse de retour. Avec cette configuration en deux blocs d'instruction, le saut d'une des trois duplications (i_0,i_1), (i_2,i_3) ou (i_4,i_5) est suffisant pour générer une faute exploitable.

```
/* Specifique */
BL.w <fonction>
/* Remplacement */
---- bloc0 ----
0 ADR.w Ry,<retour label>
1 ADR.w Ry,<retour label>
2 ADD.w lr,Ry,#1
3 ADD.w lr,Ry,#1
---- bloc1 ----
4 B.w <fonction>
5 B.w <fonction>
6 retour label
```

(a)

```
/* Amélioration */
---- bloc0 ----
0 ADR.w Ry,<retour label>
1 inst. libre
2 inst. libre
3 inst. libre
---- bloc1 ----
4 ADR.w Ry,<retour label>
5 inst. libre
6 inst. libre
7 inst. libre
---- bloc2 ----
8 ADD.w lr,Ry,#1
9 inst. libre
10 inst. libre
11 inst. libre
---- bloc3 ----
12 ADD.w lr,Ry,#1
13 inst. libre
14 inst. libre
15 inst. libre
---- bloc4 ----
16 B.w <fonction>
17 inst. libre
18 inst. libre
19 inst. libre
---- bloc5 ----
20 B.w <fonction>
21 retour label
```

(b)

FIGURE 5.4 – Instruction cible avec (a) la contre-mesures pour les instructions spécifiques [85] et (b) la propositions d'amélioration sur la ligne d'instruction.

Une amélioration par placement spécifique (décomposition en blocs d'instruction) reste le seul moyen pour rendre cette contre-mesure robuste contre les sauts multiples. Avec cette amélioration, et comme pour la précédente contre-mesure, le saut d'un seul bloc d'instruction n'a aucun impact sur l'intégrité de la protection.

Ces contre-mesures résilientes proposées par [85] sont dédiées exclusivement à la protection du flot d'instruction contre le modèle de faute saut d'instruction. Nos propositions d'amélioration apportent plus de robustesse contre le cas de plusieurs sauts

engendrés par une seule injection de faute lors de l'opération de chargement d'un flot. Toutefois, rendre ce type de protection plus sûre demande un surcoût qui n'est pas toujours évident à éviter. Aussi, contrairement aux contre-mesures de [84], appliquer ce modèle de protection n'est pas recommandé pour les opérations traitant des données.

5.4 EMFI avancées : injections de fautes multiples dans le temps

Les contre-mesures à redondance d'instruction sont à la base bien robustes contre un seul saut d'instruction moyennant une seule injection de faute. Cependant, la précédente analyse a montrée les limites de ce type de protection contre les sauts multiples dans un même flot d'instruction. Nous avons proposé des améliorations qui rendent ces contre-mesures 100% robustes face à ce modèle de faute sur la ligne d'instruction, et pour certains sur la ligne de donnée. L'amélioration consiste à réorganiser et répartir les instructions sur plusieurs chargements pour rendre les attaques de premier ordre inefficaces à contrer la protection. D'un autre coté, les performances actuelles des équipements d'injection de faute font qu'il est maintenant possible de générer plusieurs injections successives et induire plusieurs fautes.

Dans [57], Dutertre et al. évoquent déjà cette possibilité en utilisant l'injection laser. Leurs tests ont permis d'évaluer à trois cent le nombre de sauts d'instructions successives sous certaines conditions d'injection de faute. Ce résultat a été possible en procédant à la variation de la durée de l'impulsion laser. Avec un test expérimental sur un code de vérification de PIN, ils ont aussi mis en évidence l'effet des injections multiples à générer des sauts ciblés d'instructions. Á ce jour et dans le cas EM, il n'y a pas de caractérisation de fait sur de telles possibilités. Le modèle de faute de notre caractérisation permet de cibler un chargement de flot d'instruction représentant au plus quatre instructions 32 bits et jusqu'à huit instructions 16 bits. Ce modèle d'attaque présente donc un risque considérable aux contre-mesures étudiées même sous leurs formes améliorées.

Nous présentons dans la suite une étude expérimentale qui a permis d'évaluer la génération de EMFI multiples dans le temps pour produire plusieurs sauts d'instructions successifs.

5.4.1 Configuration expérimentale

Comme nous l'avons déjà établi, altérer plusieurs instructions ou données revient à injecter des fautes durant leurs cycles de chargement respectifs. La première méthode de notre démarche d'analyse de vulnérabilité sur les MCU permet d'identifier ces cycles de faute. Nous pouvons alors supposer que cibler des chargements successifs aura comme conséquence la corruption de toute une séquence d'un programme. Plus précisément, induire plusieurs sauts multiples d'instruction (remplacement par un `NOP`)), peut être assimilé à un effacement de la séquence cible.

Durant les campagnes de tests, la génération d'une IEM se fait à travers un seul signal de déclenchement configuré à un instant précis d'une séquence à tester. Produire plusieurs instants de déclenchement successifs et dans un temps assez court reste

compliqué à réaliser, et c'est surtout une limitation matérielle. En revanche, une des fonctionnalités du générateur d'impulsion de P_{Ke} est de pouvoir générer des impulsions successives (*burst*) à partir d'un seul instant de déclenchement. Ces impulsions sont alors produites à une fréquence de répétition fixe et avec les mêmes propriétés (amplitude, largeur de l'impulsion ...). Ce principe est le même qui est utilisé dans les expérimentations du Chapitre 3.

Pour rester dans l'esprit de généralité dans l'application de nos méthodologies, la caractérisation est réalisée sur une autre cible, le Arduino MKR ZERO [109]. Cette cible est dotée d'un MCU Atmel SAMD21G18A [111], dont les éléments de base de l'architecture (fig. 5.5) sont assez similaires aux MCU étudiés. Une différence par rapport aux autres cibles est que le SAMD21 implémente le jeu d'instruction ARMv6-M avec principalement le support d'instruction à encodage 16 bits et un nombre réduit à encodage 32 bits.

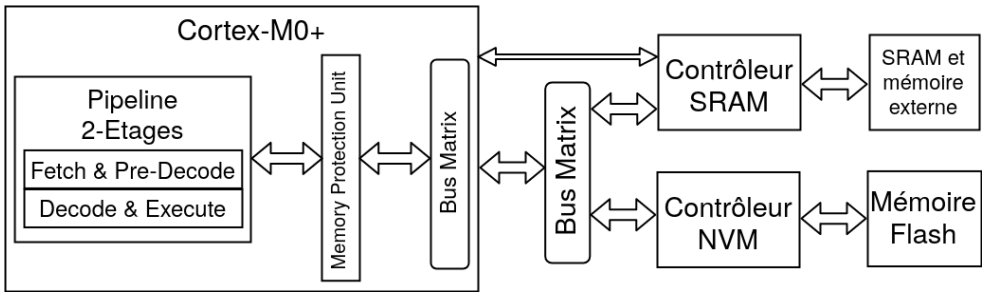


FIGURE 5.5 – Architecture du Atmel SAMD21.

Comme pour le STM32F4, nous avons appliqué nos méthodes d'analyse des vulnérabilités sur ce MCU qui est configuré avec une fréquence d'exécution de 8 MHz (source d'oscillateur interne). Cette analyse est effectuée en utilisant le Plateforme d'Injection Électromagnétique Agilent (PIEM1) avec la sonde LIRMM F, et sous effet des EMFI avec des impulsions à 6 ns de largeur et une amplitude à 0 dBm. Les résultats de la caractérisation sont comme suit :

- Vulnérabilité au niveau architecture :
  → l'interface mémoire
- Identification de deux modes de chargement :
  - Chargement alterné sur deux buffers 32 bits (deux instructions 16 bits par buffer) quand le cache est désactivé.
  - Chargement sur un buffer 64 bits (quatre instructions 16 bits) quand le cache est activé.
- Modèle de fautes au niveau bit sur le flot d'instruction :
  → *bit-reset*
- Impact temporel des fautes :
  - transitoire si cache désactivé.
  - semi-permanente si cache activé.

D'autres résultats issues de la caractérisation ont fait l'objet d'une publication et sont présentés dans [113]. Comme on peut le constater, le nombre d'instructions altérées durant un cycle de chargement diffère selon le mode de la fonction cache. Ce point sera pris en considération durant nos tests pour définir la relation entre le nombre d'impulsions et nombre d'instructions en fautes.

5.4.2 Méthodologie pour l'analyse de l'effet des injections multiples

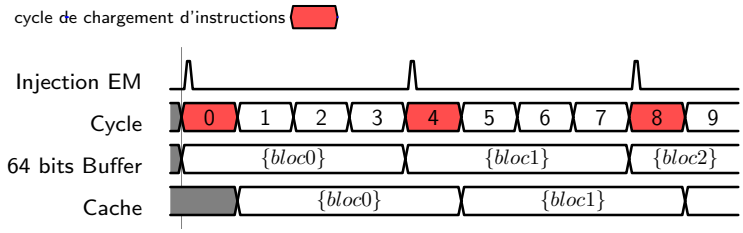
Notre démarche d'expérimentation pour générer plusieurs sauts de chargements successifs est basée sur les points suivants :

- Définition de la séquence de test.
- Déterminer la position de la sonde et l'instant d'injection.
- Variation du nombre et de la fréquence de génération des injections de faute.

Séquence de test : Le résultat finale du code à tester doit nous permettre de quantifier avec précision le nombre d'instructions en fautes. Pour arriver à identifier ce nombre, nous avons opté pour une séquence de type compteur. Ce dernier se résume à une séquence assembleur d'instructions successives de type `ADD Rx,\#1`. Cette instruction nécessite un cycle d'exécution, ce qui nous permet d'avoir une séquence avec des cycles de chargement de flot d'instruction à période fixe. La valeur du résultat finale du compteur (valeur finale dans `Rx`) sans effet EMFI, est définie en tant que valeur nominale $R_{compt}(nominale)$. Á la fin de l'exécution d'une séquence de test, mais cette fois sous effet EMFI, la valeur de `Rx` est égale à une valeur $R_{compt}(test)$, équivaut le nombre d'instructions qui sont exécutées. Selon le nombre de chargement ciblé, nous pouvons alors facilement identifier le nombre d'instruction en faute Nb_{saut} en vérifiant la valeur de différence entre le $R_{compt}(nominale)$ et $R_{compt}(test)$ eq. (5.2).

```
/* 320 instructions */
/*     16 bits      */
/*   cache activé   */
#### bloc 0 ####
0 ADD R1,#1
1 ADD R1,#1
2 ADD R1,#1
3 ADD R1,#1
...
...
#### bloc 80 ####
316 ADD R1,#1
317 ADD R1,#1
318 ADD R1,#1
319 ADD R1,#1
```

(a)



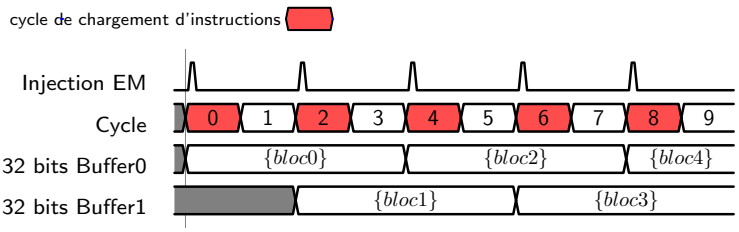
(b)

FIGURE 5.6 – Séquence de test (a) avec cache activé pour l'analyse des sauts multiples d'instructions 16 bits et (b) le diagramme temporel du test correspondant.

Position de la sonde et instant d'injection : Au cours de la caractérisation, un balayage au dessus du boîtier de la cible est effectué pour identifier les positions sensibles. Suit alors une analyse pour déterminer la position qui permet d'avoir un taux de reproductibilité de faute le plus élevé, et plus précisément la position où le modèle de faute saut d'instruction peut être observé (fig. 5.8). Pour notre est sur le SAMD21, le balayage de la sonde est effectué sur une surface de 3 mm x 3 mm avec un pas de 100 µm. En même temps, une variation de l'instant d'injection est effectué pour déterminer l'instant ou une fenêtre d'injection qui maximise le taux d'observation du modèle de faute. Afin d'être cohérent avec la séquence de test, c'est le premier chargement du code

```
/* 320 instructions */
/*     16 bits      */
/* cache désactivé  */
#### bloc 0 ####
0 ADD R1,#1
1 ADD R1,#1
...
...
#### bloc 160 ####
318 ADD R1,#1
319 ADD R1,#1
```

(a)



(b)

FIGURE 5.7 – Séquence de test (a) avec cache désactivé pour l'analyse des sauts multiples d'instructions 16 bits et (b) le diagramme temporel du test correspondant.

compteur qui doit être ciblé. Une fois la fenêtre d'injection identifiée (généralement de l'ordre de quelques nanosecondes), un instant d'injection est fixé pour ensuite faire varier le nombre et la fréquence des IEM.

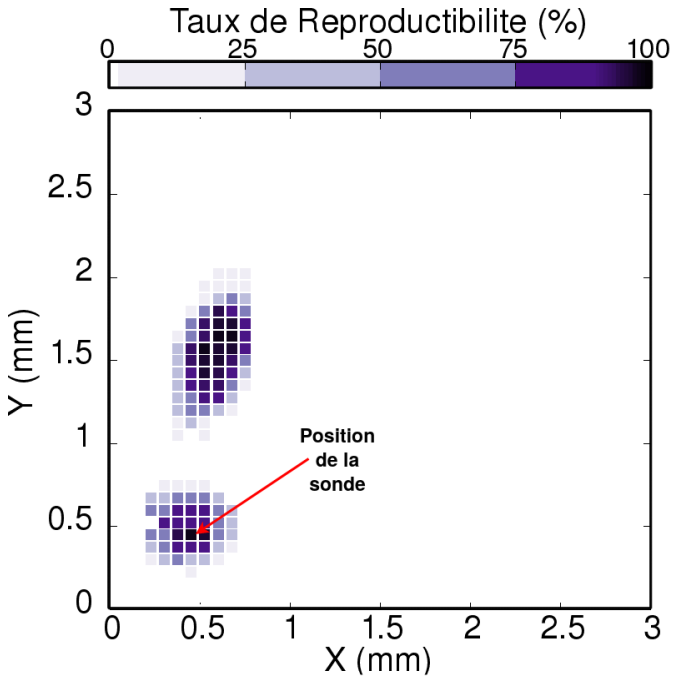


FIGURE 5.8 – Identification de la position de la sonde qui génère le saut d'instruction sur Atmel SAMD 21.

Nombre et fréquence des injections de faute : Vu qu'on cherche à altérer des chargements successifs du flot d'instruction, les injections de faute doivent alors coïncider avec les cycles relatifs à ces chargements. La fréquence des injections F_{pulse} est alors équivalente à la fréquence des chargements. Selon l'architecture et le mode de chargement d'une cible, on peut au préalable estimer la fréquence de ces chargements. Si nous prenons l'exemple de la cible SAMD21, deux modes de chargement d'instruction sont possibles. Quand le cache est activé, le flot d'instruction est de 64 bits, ce qui représente un bloc de quatre instructions 16 bits (fig. 5.6). Dans le cas où le cache est désactivé, le mode de chargement est de 32 bits, ce qui revient à avoir un bloc de deux instructions de 16 bits (fig. 5.7). On peut alors définir la fréquence de répétition

des injections F_{pulse} selon le mode de chargement. Cette fréquence est à peu près égale à une fréquence moyenne F_{moy} qui représente la valeur de la fréquence d'exécution de la cible F_{sys} diviser par le nombre d'instruction dans un bloc I_{bloc} eq. (5.1).

$$F_{pulse} \simeq F_{moy} = \frac{F_{sys}}{I_{bloc}} \tag{5.1}$$

Pour palier l'éventuel problème de précision de F_{sys}, une variation de F_{pulse} est configurée sur une plage de fréquences de façon à couvrir des fréquences inférieures et supérieures à la fréquence d'injection moyenne F_{moy}. Cette variation est produite pour chaque nombre d'impulsion Nb_{pulse}. Ce nombre est progressivement augmenté avec un pas fixe, ce qui permet d'examiner à chaque fois le résultat du compteur et être précis dans le nombre d'instructions successives qui ont subis un saut sous EMFI. Pour une valeur de Nb_{pulse} et F_{pulse} définie, l'injection de faute est concluante quand Nb_{saut} satisfait la condition de l'équation eq. (5.2).

$$Nb_{saut} = R_{compt}(nominale) - R_{compt}(test) = Nb_{pulse} \times I_{bloc} \tag{5.2}$$

La valeur de F_{pulse} qui permet de satisfaire cette condition pour tout nombre Nb_{pulse}, peut être considérée comme la fréquence moyenne réelle des opérations de chargement depuis la mémoire par rapport à notre code compteur.

5.4.3 Évaluation de la reproductibilité des injections multiples dans le temps

L'efficacité des EMFI multiples à produire des fautes successives de type saut d'instruction, est analysée selon les deux modes de chargement (relatifs à l'état du cache). Avec les différentes variations des paramètres du test (nombre d'impulsion, fréquence de l'impulsion et instant d'injection), nous présenterons seulement les résultats qui ont permis d'avoir le meilleur taux de reproductibilité des fautes. La séquence de test du compteur comporte 320 instructions successives `ADD R1,\#1`. Pour tous les tests, en fixant l'instant de l'injection, la variation de Nb_{pulse} est effectuée avec un pas de deux impulsions. Cent itérations par configuration sont effectuées pour établir le taux de reproductibilité.

Cas où I_{bloc} est égale à 4 (cache activé) : Avec un nombre d'instruction par bloc égale à quatre, nous pouvons représenter la séquence de test sous quatre-vingt flots d'instruction 64 bits successifs (fig. 5.6a). Cette configuration implique qu'il faut générer un nombre maximale de Nb_{pulse} égale à quatre-vingt afin d'altérer tous les chargements de la séquence du code compteur. La variation de F_{pulse} pour tous les nombres de Nb_{pulse} (jusqu'à 80) a permis d'évaluer la valeur moyenne réelle de la fréquence de chargement à 1.9936 MHz.

La fig. 5.9 donne l'évolution du taux de reproductibilité pour produire $Nb_{pulse} \times 4$ sauts d'instruction avec F_{pulse} égale à 1.9936 MHz. La première constatation est que ce taux décroît proportionnellement avec l'augmentation de Nb_{pulse}. Vingt IEM successives permettent de générer le saut de pas moins de 80 instructions avec un taux de reproductibilité supérieur ou égale à 80% (Tableau 5.4). Trente-cinq IEM permettent d'avoir le saut de 140 instructions successives à un taux supérieur à 50%. Pour le saut de 320 instructions de la séquence, le taux de reproductibilité atteint 30% en générant

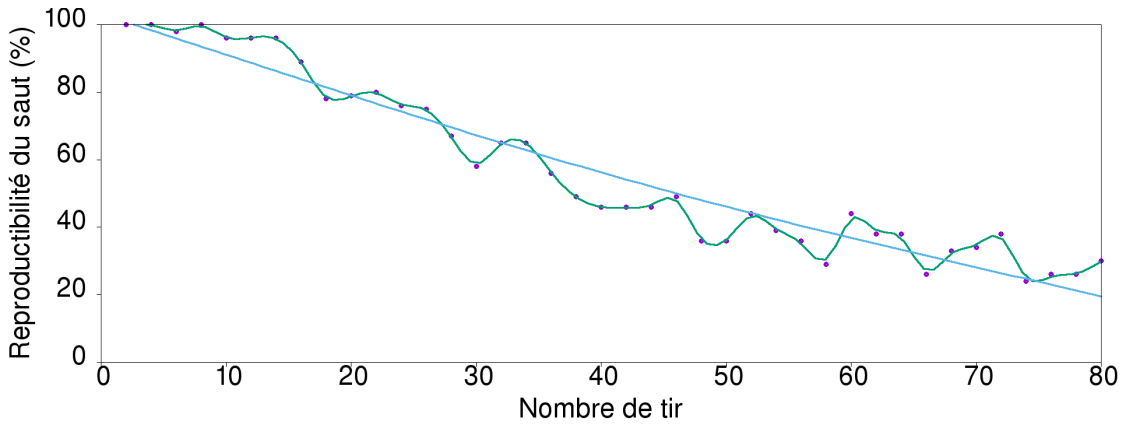


FIGURE 5.9 – Évolution du taux de reproductibilité du saut de Nb_{pulse} x 4 instructions en fonction du nombre Nb_{pulse} de IEM.

quatre-vingt IEM.

TABLEAU 5.4 – Taux de reproductibilité du nombre de saut d'instruction par rapport au nombre d'impulsion dans le cas de chargement de quatre instructions (cache activé).

Nb d'impulsion	2	10	20	30	40	50	60	70	80
Nb de saut	8	40	80	120	160	200	240	280	320
Taux (%)	100	96	79	58	46	36	44	34	30

Cas où I_{bloc} est égale à 2 (cache désactivé) : Quand le nombre d'instruction par bloc est égale à deux, 160 IEM, équivaut 160 chargements, sont nécessaires pour produire le saut des 320 instructions de la séquence de test. Pour cette configuration, la valeur moyenne réelle de F_{pulse} a été évaluée à 3.984 MHz.

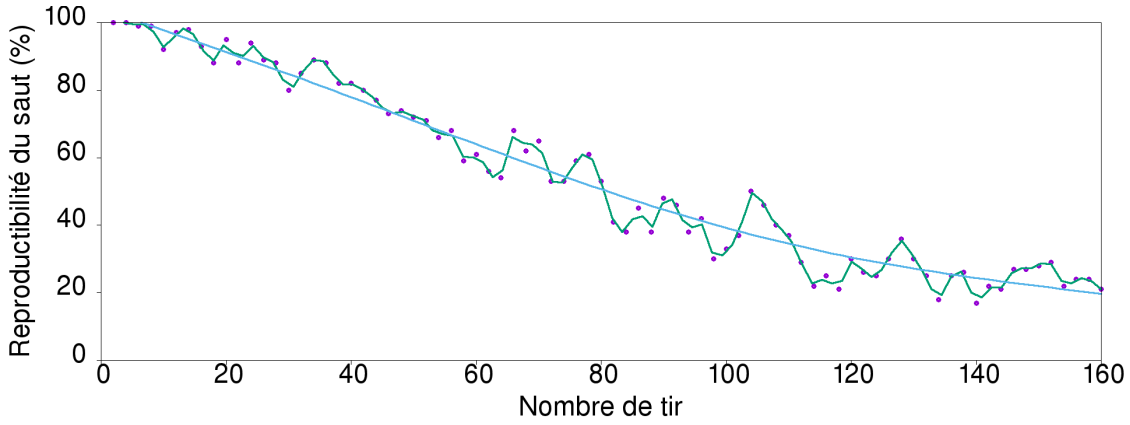


FIGURE 5.10 – Évolution du taux de reproductibilité du saut de Nb_{pulse} x 2 instructions en fonction du nombre Nb_{pulse} de IEM.

Les résultats de la Figure 5.10 montrent aussi une décroissance du taux de reproductibilité suivant l'augmentation du nombre d'impulsion. Sous l'effet de 40 IEM, le taux reste supérieur à 80% (Tableau 5.5), ce qui est pratiquement le double dans le

cas précédent pour le même nombre d'impulsion. Ce taux est supérieur à 50% quand on avoisine les 80 IEM. Au finale, 160 IEM permettent de produire le saut des 320 instructions de la séquence avec un taux de 21%.

TABLEAU 5.5 – Taux de reproductibilité du nombre de saut d'instruction par rapport au nombre d'impulsion dans le cas de chargement de deux instructions (cache désactivé).

Nb d'impulsion	2	20	40	60	80	100	120	140	160
Nb de saut	4	40	80	120	160	200	240	280	320
Taux (%)	100	95	82	61	53	33	30	17	21

La tendance décroissante du taux pour les deux modes de chargement peut être expliquée par le fait que le MCU fonctionne avec une horloge à oscillateur interne qui est moins précis. Surtout si on compare Tableau 5.4 et Tableau 5.5, on remarque que pour un même nombre de saut d'instruction, le taux est quasi-identique.

Bien qu'on n'arrive pas à égaler le taux obtenu avec l'injection de faute laser dans [57], nous pouvons néanmoins avancer l'efficacité des EMFI à produire des sauts successifs en utilisant un seul signal de déclenchement. Á noter que l'expérimentation est faite en se basant sur une séquence dont les chargements depuis la mémoire sont à période constante, ce qui permet d'avoir des IEM à fréquence constante. Par conséquent, et contenu de ces résultats, nous pouvons déjà supposer l'impact de ce modèle d'injection de fautes successifs sur les contres-mesures étudiées précédemment, aussi bien sur leur forme améliorée. Nous proposons dans la suite de réévaluer expérimentalement ces contre-mesures vis à vis le modèle de faute par injection de fautes multiples dans le temps.

5.5 Contre-attaque avec EMFI multiples sur les contre-mesures par redondance

Suite au résultat précédent, il est clair qu'induire plus d'une faute, avec un effet multiple dans le temps, va à l'encontre de l'hypothèse de base des contre-mesures par redondance. Par ailleurs, les améliorations pour la robustesse que nous avons proposé pour [84, 85] reposent essentiellement sur la répartition de la protection sur des blocs de chargement multiples. En couplant l'effet d'une injection de faute sur le chargement de tout un bloc d'instruction avec la génération d'injections multiples, nous pouvons théoriquement envisager la possibilité de cibler plusieurs chargements précis d'un programme. Nous proposons donc de réévaluer les protections améliorées de la Figure 5.1b et Figure 5.3b avec une implémentation sur la cible SAMD21 et en appliquant une attaque par injection de fautes multiples. Pour garder l'intégrité du code des protections améliorées, les tests sont exécutés en mode cache activé (chargement d'un bloc de quatre instructions). Au lieu d'utiliser des instructions 32 bits, nous adaptons les codes pour utiliser des instructions 16 bits ce qui demande un changement mineur dans le choix des registres de destination. Une IEM garde les mêmes propriétés que celles utilisées pour l'évaluation des injections multiples, à savoir 6 ns de largeur et une amplitude à 0 dBm.

```
/* Contre-mesure Améliorée */
/* méthode par duplication */
---- bloc0 ----
0 inst. libre
1 inst. libre
2 inst. libre
3 LDR R4,[R1]
---- bloc1 ----
4 LDR R2,[R1]
5 CMP R2,R4
6 BNE <erreur>
7 inst. libre
```

(a)

```
/*    Contre-mesure Améliorée    */
/* instructions non-idempotentes */
---- bloc0 ----
0 MOV R4,R1
1 inst. libre
2 inst. libre
3 inst. libre
---- bloc1 ----
4 MOV R4,R1
5 inst. libre
6 inst. libre
7 inst. libre
1 ---- bloc2 ----
10 LDR R1,[R4]
13 inst. libre
16 inst. libre
19 inst. libre
---- bloc3 ----
14 LDR R1,[R4]
17 inst. libre
20 inst. libre
23 inst. libre
```

(b)

FIGURE 5.11 – Codes des contre-mesures avec encodage 16 bits avec (a) contre-mesure amélioré de [84] avec méthode de duplication et (b) contre-mesures améliorée de [85] pour les instruction non-idempotentes.

Notre amélioration de la Figure 5.1b se résume à répartir la contre-mesure par duplication de [84] sur deux blocs de chargement pour garantir l'application de la redondance en cas de faute sur un seul chargement. Comme nous proposons de tester l'effet des injections multiples, notre test consiste à cibler ces deux chargements. L'étude des cycles de vulnérabilité (cycles de chargement) a permis de calculer la période entre le chargement du *bloc0* et *bloc1*. Vu qu'on applique seulement deux injections successives la précision de F_{pulse} n'est pas requise. La routine *erreur* est la même utilisée précédemment pour le SAM4C, sauf que pour le SAMD21 elle consiste à assigner la valeur `0x5A5A5A5A` au registre `R3`. Le Tableau 5.6 donne le résultat du test de la contre-mesure améliorée contre deux injections de faute successives ciblant le chargement du *bloc0* et *bloc1*.

TABLEAU 5.6 – Effet des injections multiples sur la version améliorée de la contre-mesure par duplication de [84].

			Injections de faute multiples Chargement bloc0 et bloc1
Registre	**Valeur Initiale**	**Attendu**	**Résultat**
R4	0xF03FF300	0xF0777000	0xF03FF300
R2	0xF0F00F00	0xF0777000	0xF0F00F00
R3 (erreur)	0xF01FF100	0xF01FF100	0xF01FF100
		Faute	(non détecté)

La conséquence la plus importante du saut des deux blocs est que la fonction de détection n'est plus effective, d'autant que les registres de destination de la redondance restent avec leur valeur initiale, donc différentes. L'exemple suivant montre le résultat

quand on cible différents chargements.

La contre-mesure amélioré de [85] pour le cas des instructions non-idempotentes est répartie sur quatre blocs d'instruction. Chaque bloc intègre une instruction de la contre-mesure initiale pour protéger la redondance. Nous proposons avec ce test de générer deux injections successives ciblant dans un premier temps les chargements de (*bloc0*,*bloc1*) et dans un deuxième temps ceux (*bloc2*,*bloc3*).

TABLEAU 5.7 – Effet des injections multiples sur la version améliorée de la contre-mesure pour instructions non-idempotentes de [85].

			Injections de faute multiples durant les chargements	
			bloc0 et bloc1	**bloc2 et bloc3**
Registre	**Valeur Initiale**	**Attendue**	**Résultat**	**Résultat**
R4	0x000002BC	0x000002B0	0x000002BC	0x000002B0
R1	0x000002B0	0xF0777000	0xF03FF300	0x000002B0
		Faute	(non protégée)	(non protégée)

Comme on peut le constater d'après le résultat du test (Tableau 5.7), cibler plus d'un chargement revient à corrompre la protection. L'effet du saut multiple des chargements peut être assimilé à un remplacement d'une portion d'un code par des `NOP`. Dans notre étude sur les contre-mesures par redondance, cela revient à contourner une partie voir toute la fonction de protection. Nos amélioration renforce bien la robustesse de ces contre-mesures contre le saut d'un seul chargement, mais restent inefficaces si plus d'un chargement est ciblé.

5.6 Conclusion

Les contre-mesures par redondance d'instruction ont fait l'objet de plusieurs évaluations pour déterminer leurs résistances contre le modèle de faute saut d'instruction. Il revient à dire que ces contre-mesures sont bien robustes principalement pour protéger le flot d'instruction contre un seul saut d'instruction. Le modèle de faute issue des résultats de la caractérisation dans le Chapitre 4 permet par contre de générer plusieurs fautes, en l'occurrence plusieurs sauts d'instructions d'un même flot. Dans ce chapitre, ce modèle de faute a été pris en considération pour évaluer les vulnérabilités des contre-mesures à base de redondance d'instruction.

En prenant comme contre-mesures de références celles présentées par [84, 85], une étude théorique a été présentée sur les failles de ces protections par rapport aux sauts multiples dans un même flot d'instruction. Cette analyse a permis de mettre en évidence la non-robustesse de l'implémentation de ces contre-mesures même dans le cas où une fonction de détection ou de correction est utilisée. Sur les protections étudiées de [84], des failles ont aussi été identifiées au niveau du flot de donnée, alors que pour les protections de [85], ils ne tiennent pas compte de ce type de faute. Cependant, des propositions d'améliorations de ces protections ont été avancées pour une meilleure robustesse au niveau du flot d'instruction, et aussi pour le flot de donnée sur les contre-mesures de [84] uniquement. Puisque les fautes sont induites principalement durant le chargement d'un flot, la méthode d'amélioration consiste à répartir la redondance sur plusieurs flots, en l'occurrence plusieurs chargements. Au niveau des données, l'ajout

d'une opération intermédiaire sur une donnée non-successive à celle à protéger permet de contourner l'effet semi-persistent de la faute si le *buffer* de donnée n'est pas mis à jour.

Une analyse expérimentale de ces protections sous effets des EMFI a permis de confirmer l'analyse théorique de ces contre-mesures tant sur leur forme initiale qu'améliorée. Les résultats soulignent la robustesse des versions améliorées contre le saut multiple lors d'un seul chargement d'une ligne d'instruction pour les protections de [85] et sur les deux flots d'instruction et de donnée pour [84]. Étant donnée que ces améliorations peuvent ajouter un surcoût dans l'utilisation de l'espace mémoire, une idée de modification du compilateur binaire a été proposé pour éviter ce surcoût.

En se référant à ces propositions d'améliorations pour contrer le saut d'un seul flot de chargement, l'analyse a été étendue à l'éventuel cas des sauts multiples de chargements et leurs impacts sur les protections améliorées. Dans un premier temps, une caractérisation de l'effet des EMFI multiples dans le temps a été détaillée et comparée à celle effectuée dans [57] avec des injections de faute laser. Bien que le taux de reproductibilité des fautes diminue avec l'augmentation du nombre d'instructions ciblées, le résultat de cette caractérisation soulève le risque que peut poser le rayonnement EM à engendrer un saut d'un nombre important d'instructions successives. Une amélioration du taux peut être envisagée, comme l'utilisation d'un oscillateur externe comme source d'horloge pour une meilleure stabilité de la fréquence d'exécution, ou encore créer plusieurs instants de déclenchement ....

Ce modèle de faute a été expérimentalement testé sur les contre-mesures sous leurs formes améliorées. Les résultats ont montré que ces protections par redondance d'instruction sont totalement inefficaces face aux sauts multiples dans le temps. La capacité de pouvoir cibler plus d'un flot d'instruction en EMFI, évaluée et confirmée par ces expérimentations, rend la conception de contre-mesures au niveau instruction assez complexe à mettre en place.

CHAPITRE 6

Conclusions

Face aux menaces grandissantes qui visent les systèmes sécurisés, il est de plus en plus important de prévoir les contre-mesures adéquates et qui tiennent compte des différentes formes de perturbations externes. Pour arriver à un tel niveau de protection, il est essentiel d'avoir une connaissance la plus complète de l'effet qu'engendrent ces perturbations, et plus particulièrement par rapport aux attaques par injections de faute. L'objectif de cette thèse tend à avancer des méthodes qui contribuent à une définition détaillée des vulnérabilités des circuits intégrés face à la menace des attaques, précisément ceux par rayonnement électromagnétique, et cela à plusieurs niveaux d'abstraction. Les différents résultats permettront d'établir le plus grand nombre de modèles de faute afin d'en tenir compte pour la conception de protection plus robuste. Pour toutes les caractérisations effectuées dans ce travail, nous avons considéré l'utilisation de différentes configurations entre cibles de tests, paramètres d'injection, injecteurs, et même plateforme d'injection électromagnétique, et cela, dans un souci de généralité et aussi pouvoir apporter plus de détails sur les effets du rayonnement électromagnétique.

Dans un premier temps, nous nous sommes intéressés sur l'effet qu'engendrent les injections de faute électromagnétiques au niveau logique dans un circuit programmable. Dans cette première caractérisation, une attention particulière est portée sur le type d'injecteur, en l'occurrence une sonde magnétique, et comment les propriétés qui définissent sa conception contribuent dans l'efficacité d'une injection électromagnétique. Différentes sondes, faites maison et commerciales sont testées sur différentes cibles de type FPGA afin d'apporter une comparaison exhaustive. Les différentes expérimentations révèlent que plus le diamètre de la sonde est proche de la valeur de la technologie de procédé de la cible, plus on peut être précis dans la définition de la résolution spatiale des fautes. Cette étude des sondes d'injection nous a aussi amené à privilégier les sondes avec noyau en ferrite du fait que leur impact est considérablement élevé par rapport aux autres sondes à noyau d'air. L'évaluation de l'impact est effectuée au niveau de la logique combinatoire, avec une méthode centrée sur la mesure du temps de propagation entre cellules logiques d'un FPGA sous effet des injections de faute électromagnétiques. D'autre part, l'effet des paramètres d'injection comme l'amplitude et la polarité de l'impulsion électromagnétiques, ou encore le nombre d'impulsions et l'instant d'injection, est étudié dans le but de comprendre comment leur configuration

permet d'optimiser l'observation de l'impact.

Nous avons par la suite étendu la caractérisation au niveau logiciel sur différentes cibles de type microcontrôleur. Vu le manque d'informations sur un procédé clair dans l'état de l'art, il nous semblait important d'avancer une démarche précise qui aiderait à définir avec précision les types de vulnérabilités à ce niveau. Dans le quatrième chapitre, l'accent a été donc mis sur la validation d'une démarche d'analyse qui est formulée sur deux étapes successives. Dans la première étape, nous avons cherché à identifier les éléments vulnérables des microcontrôleurs au niveau architecture. Une méthode (méthode 1) a été présenté dans ce sens et qui est basé sur l'observation de l'effet du rayonnement électromagnétique durant plusieurs cycles de fonctionnement. L'application de cette méthode a permis d'identifier l'interface mémoire comme étant l'élément vulnérable pour tous les MCU testés. Le résultat de cette première étape, sert comme point de départ pour la suite de la caractérisation avec sa deuxième étape.

Ce résultat est le point de départ de la seconde étape, qui est définie par deux méthodes (méthode 2 et méthode 3), où chacune peut être utilisée séparément. L'élément vulnérable étant identifié, différentes adaptations des séquences de tests sont établis suivant qu'on cible le fonctionnement de l'élément sur le flot d'instruction et de donnée. Ces adaptations sont faites suivant un choix précis des opcodes des instructions formant la séquence ou encore la valeur binaire des données chargées. Avec la méthode 2, il été possible d'identifier le modèle de faute au niveau bit sur les différents microcontrôleurs testés. Fort de constater qu'avec une même configuration d'injection EM (même plateforme), deux cibles de différents constructeurs ne réagissent pas de la même façon au rayonnement EM qui génère des modèles opposés à savoir *bit-set* et *bit-reset*. D'un autre côté, nous avons observé qu'une même cible ne réagit pas de la même façon sur deux plateformes différentes, en l'occurrence à la forme d'onde générée par différents générateurs d'impulsion.

La troisième méthode qui vient compléter notre démarche permet quant à elle de déterminer l'impact temporel des fautes engendrées. Sous effet des EMFI, nous sommes parvenus à définir de manière expérimentale que les fautes induites peuvent soit avoir l'aspect transitoire soit semi-persistent. Cet effet temporel des fautes est directement lié au mode de fonctionnement défini par l'interface mémoire du MCU cible. Aussi, à l'aide de séquences bien adaptées pour ce type de caractérisation, nous avons pu confirmer que les fautes engendrées par EMFI se produisent au cours du chargement depuis la mémoire vers les buffers dédiés aux instructions ou données ou vers la mémoire cache. Tant qu'un buffer mémoire comportant une instruction ou lune donnée n'a pas été mise à jour, la faute est évaluée en tant que semi-permanente. Cet effet est principalement observé quand la fonction cache est utilisée dans l'exécution du programme. Cependant, la faute aura un effet transitoire avec la mise à jour du buffer, ce qui est le cas durant un fonctionnement sans cache vue que durant une exécution sans optimisation, la mise à jour d'un buffer se fait en continue à chaque demande de chargement depuis la mémoire.

Durant cette caractérisation, d'autres effets des paramètres d'injection ont été observés où le paramètre spatial est déterminant pour une cible alors que c'est plutôt l'amplitude de l'impulsion qui l'est pour une autre. Cette variation des paramètres est dans un sens la clé qui a permis de mettre en évidence les fautes sur les instructions et les données de point de vue nombre et répartition dans leurs flots respectifs. Avec

les différents résultats issus de l'application de nos différentes méthodes, il est possible de cibler une portion de code d'un programme et de prédire l'éventuel impact au niveau instruction, simplement en fixant les paramètres d'injection à une configuration précise. La perturbation qui est générée permet d'altérer quatre instructions 32 bits et jusqu'à huit instructions 16 bits d'un même flot 128 bits. Ce type de corruption a aussi été possible sur le flot de donnée, que se soit par le caractère temporel des fautes ou le fait d'induire un effet de *bit-set* ou *bit-reset* sur un nombre défini de donnée 32 bits ou la totalité d'un mot 128 bits (quatre données 32 bits) et cela avec une précision à 100%.

Avec cette éventualité d'attaque contrôlée en main, on s'est tourné vers le but essentiel d'une telle caractérisation qui est de permettre la conception de contre-mesures capables de contrer différents modèles de fautes. Pour cela, dans le cinquième chapitre nous avons effectué une évaluation de contre-mesures logicielles existantes qui sont principalement basées sur la redondance d'instruction. Cette évaluation théorique dans un premier temps a mis en avant l'effet les fautes multiples dans un même flot, identifié dans le chapitre précédent, sur la robustesse de ces contre-mesures. Plus précisément, c'est la capacité à induire des sauts multiples qui est discuté dans cette étude de vulnérabilité, sachant qu'à la base ces contre-mesures sont supposées résister à un seul saut, voir jusqu'à trois pour une contre-mesure précise. L'évaluation expérimentale a confirmé les déductions théoriques à savoir que ces contre-mesures ne sont pas robuste face aux fautes de type saut multiple. C'est dans ce sens que nous avons avancé des améliorations qui prennent en compte ce type de fautes multiples et dont l'efficacité a été confirmée expérimentalement, rendant ces contre-mesures, sous leur forme améliorée, totalement résistants aux sauts multiples générés dans un même flot d'instruction. Pour certaines contre-mesures, nous avons même proposé des améliorations sur la ligne de donnée sans apporter un changement drastique à l'intégrité de la protection.

Cependant, et en ayant comme point de départ le résultat du chapitre quatre, nous avons poussé notre analyse sur le cas où il serait possible d'induire plusieurs fautes sur plusieurs chargements de flot d'instruction. Une démarche de caractérisation a été présentée dans ce sens pour évaluer l'effet des injections de faute multiples dans le temps. Nos tests ont permis de montrer qu'avec des EMFI on peut réaliser des fautes successives sur des chargements successifs, comme pour le cas des injections en mode laser. Avec nos expérimentations, nous avons pu générer le saut de 320 instructions successives avec un taux de reproductibilité entre 20% et 30% selon le mode de chargement du flot d'instruction. Ce résultat, qui à notre connaissance n'a pas été encore mentionné dans l'état de l'art avec comme moyen d'injection le rayonnement électromagnétique, implique entre autres de revoir notre précédente évaluation des contre-mesures surtout par rapport aux améliorations qui ont été apportées. C'est sans surprise que lors des nouvelles expérimentations et en générant des sauts multiples dans le temps (dans notre cas le saut de deux chargements de flot d'instruction), il été possible de contrer nos améliorations rendant encore une fois ces protections inefficaces.

Au final, les contre-mesures par redondance d'instruction de l'état de l'art sont bien robustes contre un seul saut d'instruction dans un même flot, elles le sont aussi pour les sauts multiples moyennant des améliorations, mais se trouvent totalement inefficaces contre les EMFI avancées à fautes multiples dans le temps.

Discussion et perspectives

Les résultats évoqués précédemment montrent l'intérêt d'une caractérisation méthodologique pour établir les différentes vulnérabilités d'un circuit. L'analyse de l'influence des paramètres qui définissent les EMFI offre des informations précieuses aux contre-mesures afin de tenir compte et anticiper les différentes corruptions.

Au niveau logique, notre étude s'est limitée à l'analyse de la partie combinatoire sur différentes configurations de cibles FPGA. Il est donc important de continuer cette analyse en caractérisant d'autres paramètres d'injection comme la largeur d'une impulsion EM. La variation de ce paramètre a déjà permis de montrer un évident contrôle dans le choix du modèle de faute à induire [113]. Á noter que nos caractérisations ont été faites suivant un placement précis des portes logiques, ce qui rend intéressant de vérifier le comportement des cibles suivant un placement différent. Une telle analyse aide à déterminer la zone occupée par les cellules logiques sans avoir à décapsuler le boîtier d'une cible, et donc être précis de point de vue spatiale. Une suite possible sera alors l'intégration de protection par détecteur et de vérifier la cohérence par rapport à l'impact spatial.

Notre caractérisation des sondes magnétiques a contribué à donner un début d'idée sur les propriétés à privilégier dans le choix d'une sonde pour l'injection de faute. Il est cependant important d'étendre la caractérisation à d'autres sondes surtout de point de vue aspect mécanique, qui comme avancé dans [40] améliore la précision spatiale qui est déterminante pour générer des fautes exploitables. Á noter qu'on s'est limité à une caractérisation impulsionnelle de la forme d'onde et qu'il est nécessaire de considérer les ondes harmoniques qui ont montré leur efficacité à provoquer des fautes d'échantillonnage.

Notre démarche d'analyse sur les MCU apporte beaucoup de détails sur les vulnérabilités à différents niveaux. La fonction cache reste une des fonctions dont il faut analyser davantage, surtout qu'avec des MCU modernes multi-coeurs, on trouve différents niveaux de cache et donc une possibilité d'étendre notre méthode à prendre en compte plusieurs niveaux et surtout déterminer celui en faute. Aussi, on peut penser à appliquer nos méthodes sur ces cibles multi-coeurs et de caractériser l'effet EMFI surtout d'un point de vue spatial. Le but serait de déterminer si avec cette configuration de cible une seule sonde est capable d'altérer le fonctionnement sur les deux coeurs à partir d'une même position. Le cas contraire, deux positions différentes, impliquera la nécessité d'en avoir deux.

Le saut d'instruction reste le modèle de faute le plus dangereux puisqu'il permet de contourner un bon nombre d'instructions, voir tout un bout de code. Ceci étant, on a pu constater que les fautes sur le flot de donnée sont très négligées dans les contre-mesures alors qu'elles sont plus simples à générer que des fautes sur les instructions. Notre analyse a montré que mettre à `1` ou à `0` un ou plusieurs bits d'un opcode n'est pas forcément interprété comme un `NOP` par le CPU si de plus l'opcode altéré n'est pas valide. Ce n'est pas le cas avec une donnée puisqu'il n'y a aucun processus de validation de la part du CPU sur l'intégrité de sa valeur. D'un MCU à un autre, on a vu que l'effet diffère, surtout de point de vue sensibilité à un paramètre d'injection.

Que ce soit sur les instructions ou les données, l'évidence des fautes multiples a été démontré sur différentes cibles ce qui donne déjà un bon aspect de généralité de la méthode qu'on a adoptée. Sauf que ce type de faute laisse beaucoup de questions sur le manque d'efficacité des contres-mesures au niveau instructions. Une proposition de perspective sera de voir s'il y a une possible amélioration de ces contre-mesures par rapport à ce mode de faute, ou si comme évoqué précédemment, une solution avec une architecture multi-coeurs serait la plus simple à mettre en place.

Liste des Publications

Communication dans un congrès :

- Oualid Trabelsi, Jean-Luc Danger and Laurent Sauvage. Characterization at Logical Level of Magnetic Injection Probes. *Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility*, 2019.
- Oualid Trabelsi, Jean-Luc Danger and Laurent Sauvage. Impact of Intentional Electromagnetic Interference on Pure Combinational Logic. *International Symposium on Electromagnetic Compatibility*, 2019.
- Oualid Trabelsi, Jean-Luc Danger and Laurent Sauvage. Characterization of Electromagnetic Fault Injection on a 32-bit Microcontroller Instruction Buffer. *Asian Hardware Oriented Security and Trust Symposium*, 2020.
- Van Thanh Khuat, Oualid Trabelsi, Jean-Luc Danger and Laurent Sauvage. Multiple and Reproducible Fault Models on Micro-controller using Electromagnetic Fault Injection. *Joint International Symposium on Electromagnetic Compatibility, Signal & Power Integrity, and EMC EUROPE*, 2021.

Bibliographie

[1] O. Kömmerling et M-G. Kuhn. "Design Principles for Tamper-Resistant Smartcard Processors". In : *Proceedings of the USENIX Workshop on Smartcard Technology on USENIX Workshop on Smartcard Technology*. 1999 (cf. p. 6).

[2] S. Skorobogatov. "Semi-invasive attacks-A new approach to hardware security analysis". In : (2005) (cf. p. 6).

[3] M. Witteman. "Advances in smartcard security". In : *Information Security Bulletin* (2002) (cf. p. 6).

[4] T. Ordas et al. "Near-Field Mapping System to Scan in Time Domain the Magnetic Emissions of Integrated Circuits". In : *18th International Workshop on Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation - Volume 5349*. 2008, p. 229–236 (cf. p. 6).

[5] P-C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In : *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*. 1996, p. 104–113 (cf. p. 6).

[6] P. Kocher et al. "Introduction to differential power analysis". In : *Journal of Cryptographic Engineering* (2011) (cf. p. 6).

[7] J-J. Quisquater et D. Samyde. "ElectroMagnetic Analysis (EMA) : Measures and Counter-Measures for Smart Cards". In : *Proceedings of the International Conference on Research in Smart Cards : Smart Card Programming and Security*. 2001, p. 200–210 (cf. p. 6).

[8] S. Endo et al. "An on-chip glitchy-clock generator for testing fault injection attacks". In : *Journal of Cryptographic Engineering* (2011), p. 265–270 (cf. p. 6).

[9] M. Hutter et J-M. Schmidt. "The Temperature Side-Channel and Heating Fault Attacks". In : 2013. doi : 10.1007/978-3-319-08302-5_15 (cf. p. 6).

[10] Y. Kim et al. "Flipping bits in memory without accessing them : An experimental study of DRAM disturbance errors". In : *ACM SIGARCH Computer Architecture News*. 2014, p. 361–372 (cf. p. 6).

[11] V. Van Der Veen et al. "Drammer : Deterministic Rowhammer attacks on mobile platforms". In : *Proceedings of the ACM Conference on Computer and Communications Security*. 2016, p. 1675–1689 (cf. p. 6).

[12] P. Frigo et al. “Grand Pwning Unit : Accelerating Microarchitectural Attacks with the GPU”. In : *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, p. 195–210 (cf. p. 6).

[13] Z. Zhang et al. “Triggering Rowhammer Hardware Faults on ARM : A Revisit”. In : *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, ASHES*. 2018, p. 24–33 (cf. p. 6).

[14] P. Kocher, J. Jaffe et B. Jun. “Differential power analysis”. In : *19th Annual International Cryptology Conference, Advances in Cryptology, CRYPTO 99*. 1999. doi : 10.1007/3-540-48405-1_25 (cf. p. 6).

[15] D. Boneh, R-A. DeMillo et R-J Lipton. “On the Importance of Checking Cryptographic Protocols for Faults”. In : *Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques*. 1997, p. 37–51 (cf. p. 7).

[16] E. Biham et A. Shamir. “Differential Fault Analysis of Secret Key Cryptosystems”. In : *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*. 1997, p. 513–525 (cf. p. 7).

[17] C. Giraud. “DFA on AES”. In : *Proceedings of the 4th International Conference on Advanced Encryption Standard*. 2004, p. 27–41 (cf. p. 7).

[18] G. Piret et J-J Quisquater. “A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD”. In : *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop*. 2003, p. 77–88. doi : 10.1007/978-3-540-45238-6_7 (cf. p. 7).

[19] M. Soucarros et al. “Influence of the temperature on true random number generators”. In : *Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2011, p. 24–27. doi : 10.1109/HST.2011.5954990 (cf. p. 7).

[20] A. Barenghi et al. “A fault induction technique based on voltage underfeeding with application to attacks against AES and RSA”. In : *J. Syst. Softw.* (2013), p. 1864–1878. doi : 10.1016/j.jss.2013.02.021 (cf. p. 7).

[21] D. Zhi-bo, Y. Chen et A-D. Chen. “The Impact of the Clock Frequency on the Power Analysis Attacks”. In : *2011 International Conference on Internet Technology and Applications*. 2011, p. 1–4. doi : 10.1109/ITAP.2011.6006291 (cf. p. 7).

[22] T. Fukunaga et J. Takahashi. “Practical Fault Attack on a Cryptographic LSI with ISO/IEC 18033-3 Block Ciphers”. In : *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2009, p. 84–92. doi : 10.1109/FDTC.2009.34 (cf. p. 7).

[23] M. Kuhn et O. Kömmerling. “Physical security of smartcards”. In : *Inf. Secur. Tech. Rep.* (1999), p. 28–41. doi : 10.1016/S0167-4048(99)80012-0 (cf. p. 7).

[24] C. Bozzato, R. Focardi et F. Palmarini. “Shaping the Glitch : Optimizing Voltage Fault Injection Attacks”. In : *IACR Trans. Cryptogr. Hardw. Embed. Syst.* (2019), p. 199–224 (cf. p. 7).

[25] K. Tobich et al. “Voltage Spikes on the Substrate to Obtain Timing Faults”. In : *2013 Euromicro Conference on Digital System Design*. 2013, p. 483–486. doi : 10.1109/DSD.2013.146 (cf. p. 7).

[26] Jörn-Marc SCHMIDT et Christoph HERBST. "A Practical Fault Attack on Square and Multiply". In : *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2008, p. 53–58. DOI : 10.1109/FDTC.2008.10 (cf. p. 7).

[27] T. KORAK et M. HOEFLER. "On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms". In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2014, p. 8–17. DOI : 10.1109/FDTC.2014.11 (cf. p. 8, 14, 15).

[28] S-P. SKOROBOGATOV et R-J. ANDERSON. "Optical Fault Induction Attacks". In : *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop*. 2002. DOI : 10.1007/3-540-36400-5_2 (cf. p. 8).

[29] C. ROSCIANL et al. "Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells". In : *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2013, p. 89–98. DOI : 10.1109/FDTC.2013.17 (cf. p. 8).

[30] J-J. QUISQUATER et D. SAMYDE. "Eddy current for magnetic analysis with active sensor". In : (2002) (cf. p. 9).

[31] F. VARGAS et al. "On the proposition of an EMI-based fault injection approach". In : *11th IEEE International On-Line Testing Symposium*. 2005, p. 207–208. DOI : 10.1109/IOLTS.2005.47 (cf. p. 9, 13).

[32] P. BAYON et al. "Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator". In : *Constructive Side-Channel Analysis and Secure Design - Third International Workshop (COSADE)*. 2012, p. 151–166. DOI : 10.1007/978-3-642-29912-4_12 (cf. p. 9).

[33] F. POUCHERET et al. "Local and Direct EM Injection of Power Into CMOS Integrated Circuits". In : *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2011, p. 100–104. DOI : 10.1109/FDTC.2011.18 (cf. p. 9).

[34] J-M. SCHMIDT et M. HUTTER. "Optical and em fault-attacks on crt-based rsa : Concrete results". In : (2007) (cf. p. 9).

[35] M. DUMONT, M. LISART et P. MAURINE. "Electromagnetic Fault Injection : How Faults Occur". In : *2019 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2019, p. 9–16. DOI : 10.1109/FDTC.2019.00010 (cf. p. 9).

[36] M. DUMONT, P. MAURINE et M. LISART. "Modeling of Electromagnetic Fault Injection". In : *2019 12th International Workshop on the Electromagnetic Compatibility of Integrated Circuits (EMC Compo)*. 2019, p. 246–248. DOI : 10.1109/EMCCompo.2019.8919964 (cf. p. 9).

[37] A. DEHBAOUI et al. "Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES". In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2012, p. 7–15. DOI : 10.1109/FDTC.2012.15 (cf. p. 9, 13).

[38] L. SAUVAGE. "Electric probes for fault injection attack". In : *Asia-Pacific Symposium on Electromagnetic Compatibility (APEMC)*. 2013, p. 1–4 (cf. p. 10, 11).

[39] R. OMAROUAYACHE et al. "Magnetic microprobe design for EM fault attack". In : *International Symposium on Electromagnetic Compatibility*. 2013, p. 949–954 (cf. p. 10, 29).

[40] S. Ordas et al. “Evidence of a Larger EM-Induced Fault Model”. In : *Smart Card Research and Advanced Applications - 13th International Conference, (CARDIS)*. 2014, p. 245–259. doi : 10.1007/978-3-319-16763-3_15 (cf. p. 11, 13, 23, 24, 29, 30, 55, 86, 114).

[41] I. Verbauwhede, D. Karaklajic et J-M Schmidt. “The Fault Attack Jungle - A Classification Model to Guide You”. In : *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. 2011, p. 3–8. doi : 10.1109/FDTC.2011.13 (cf. p. 12).

[42] A. Barenghi et al. “Fault Injection Attacks on Cryptographic Devices : Theory, Practice, and Countermeasures”. In : *Proceedings of the IEEE* (2012), p. 3056–3076. doi : 10.1109/JPROC.2012.2188769 (cf. p. 12, 16).

[43] B. Yuce et al. “Software Fault Resistance is Futile : Effective Single-Glitch Attacks”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2016, p. 47–58. doi : 10.1109/FDTC.2016.21 (cf. p. 12, 19).

[44] J. Laurent et al. “On the Importance of Analysing Microarchitecture for Accurate Software Fault Models”. In : *21st Euromicro Conference on Digital System Design (DSD)*. 2018, p. 561–564. doi : 10.1109/DSD.2018.00097 (cf. p. 12, 19).

[45] P. Maistri et al. “Electromagnetic analysis and fault injection onto secure circuits”. In : *22nd International Conference on Very Large Scale Integration (VLSI-SoC)*. 2014, p. 1–6. doi : 10.1109/VLSI-SoC.2014.7004182 (cf. p. 12).

[46] A. Dehbaoui et al. “Investigation of near-field pulsed EMI at IC level”. In : *Asia-Pacific Symposium on Electromagnetic Compatibility (APEMC)*. 2013, p. 1–4. doi : 10.1109/APEMC.2013.7360621 (cf. p. 13).

[47] A. Dehbaoui et al. “Injection of transient faults using electromagnetic pulses -Practical results on a cryptographic system”. In : *IACR Cryptol. ePrint Arch.* (2012) (cf. p. 13).

[48] S. Ordas, L. Guillaume-Sage et P. Maurine. “EM Injection : Fault Model and Locality”. In : *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2015, p. 3–13. doi : 10.1109/FDTC.2015.9 (cf. p. 13).

[49] S. Ordas, L. Guillaume-Sage et P. Maurine. “Electromagnetic fault injection : the curse of flip-flops”. In : *J. Cryptogr. Eng.* (2017), p. 183–197. doi : 10.1007/s13389-016-0128-3 (cf. p. 13).

[50] M. Ghodrati et al. “Inducing Local Timing Fault Through EM Injection”. In : *55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 2018, p. 1–6. doi : 10.1109/DAC.2018.8465836 (cf. p. 13).

[51] J. Balasch, B. Gierlichs et I. Verbauwhede. “An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2011, p. 105–114. doi : 10.1109/FDTC.2011.9 (cf. p. 14, 15).

[52] N. Moro et al. “Electromagnetic Fault Injection : Towards a Fault Model on a 32-bit Microcontroller”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2013, p. 77–88 (cf. p. 14, 15, 57).

[53] A. Menu et al. “Precise Spatio-Temporal Electromagnetic Fault Injections on Data Transfers”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2019, p. 1–8 (cf. p. 14, 15, 57, 65–67, 71, 72).

[54] B. Yuce, N. F. Ghalaty et P. Schaumont. “Improving Fault Attacks on Embedded Software Using RISC Pipeline Characterization”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2015, p. 97–108. doi : 10.1109/FDTC.2015.16 (cf. p. 15, 55).

[55] J. Breier et D. Jap. “Testing feasibility of back-side laser fault injection on a microcontroller”. In : *Proceedings of the 10th Workshop on Embedded Systems Security, WESS*. 2015 (cf. p. 15).

[56] M-S. Kelly, K. Mayes et J-F. Walker. “Characterising a CPU fault attack model via run-time data analysis”. In : *Proceedings of the 2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST*. 2017 (cf. p. 15).

[57] J-M. Dutertre et al. “Experimental Analysis of the Laser-Induced Instruction Skip Fault Model”. In : *Secure IT Systems - 24th Nordic Conference, NordSec*. 2019 (cf. p. 15, 101, 107, 110).

[58] L. Rivière et al. “High precision fault injections on the instruction cache of ARMv7-M architectures”. In : *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2015, p. 62–67 (cf. p. 15, 57, 75).

[59] J. Proy et al. “A First ISA-Level Characterization of EM Pulse Effects on Superscalar Microarchitectures : A Secure Software Perspective”. In : *Proceedings of the 14th International Conference on Availability, Reliability and Security, ARES*. 2019, 7 :1–7 :10. doi : 10.1145/3339252.3339253 (cf. p. 15).

[60] H. Liao et C. Gebotys. “Methodology for EM Fault Injection : Charge-based Fault Model”. In : *Design, Automation Test in Europe Conference Exhibition (DATE)*. 2019, p. 256–259. doi : 10.23919/DATE.2019.8715150 (cf. p. 15).

[61] L. Claudepierre et P. Besnier. “Microcontroller Sensitivity to Fault-Injection Induced by Near-Field Electromagnetic Interference”. In : *Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*. 2019, p. 673–676. doi : 10.23919/EMCTokyo.2019.8893701 (cf. p. 15).

[62] T. Trouchkine, G. Bouffard et J. Clédière. “Fault Injection Characterization on Modern CPUs : From the ISA to the Micro-Architecture”. In : 2019, p. 123–138 (cf. p. 15, 75).

[63] T. Trouchkine et al. “Electromagnetic fault injection against a System-on-Chip, toward new micro-architectural fault models”. In : *CoRR* (2019). url : arxiv.org/abs/1910.11566 (cf. p. 15).

[64] A. Menu et al. “Experimental Analysis of the Electromagnetic Instruction Skip Fault Model”. In : *15th Design & Technology of Integrated Systems in Nanoscale Era, DTIS*. 2020. doi : 10.1109/DTIS48698.2020.9081261 (cf. p. 15).

[65] I. Koren et C-M. Krishna. *Fault-Tolerant Systems*. 2007 (cf. p. 16).

[66] H. Bar-El et al. “The Sorcerer’s Apprentice Guide to Fault Attacks”. In : *Proceedings of the IEEE* (2006), p. 370–382 (cf. p. 16).

[67] P. Laackmann et H. Taddiken. “Apparatus for protecting an integrated circuit formed in a substrate and method for protecting the circuit against reverse engineering”. 2000 (cf. p. 16).

[68] J-M. Cioranesco et al. “Cryptographically secure shields”. In : *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2014, p. 25–31. doi : 10.1109/HST.2014.6855563 (cf. p. 16).

[69] A. Sarafianos et al. “Robustness improvement of an SRAM cell against laser-induced fault injection”. In : *2013 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*. 2013, p. 149–154. doi : 10.1109/DFT.2013.6653598 (cf. p. 16).

[70] A-G. Yanci, S. Pickles et T. Arslan. “Characterization of a Voltage Glitch Attack Detector for Secure Devices”. In : *2009 Symposium on Bio-inspired Learning and Intelligent Systems for Security*. 2009, p. 91–96. doi : 10.1109/BLISS.2009.18 (cf. p. 16).

[71] N-A. Anagnostopoulos. *Optical fault injection attacks in smart card chips and an evaluation of countermeasures against them*. 2014 (cf. p. 16).

[72] A. Sarafianos. “Injection de fautes par impulsion laser dans des circuits sécurisés”. In : 2013 (cf. p. 16).

[73] H. Naofumi et al. “EM Attack Is Non-invasive ? - Design Methodology and Validity Verification of EM Attack Sensor”. In : *Cryptographic Hardware and Embedded Systems - CHES - 16th International Workshop*. 2014, p. 1–16. doi : 10.1007/978-3-662-44709-3_1 (cf. p. 16).

[74] T. Ordas et al. “Integrated circuit protection method, and corresponding integrated circuit”. 2016 (cf. p. 17).

[75] L. Zussa et al. “Efficiency of a glitch detector against electromagnetic fault injection”. In : *Design, Automation Test in Europe Conference Exhibition (DATE)*. 2014, p. 1–6. doi : 10.7873/DATE.2014.216 (cf. p. 17).

[76] D. Fujimoto et al. “Detection of IEMI fault injection using voltage monitor constructed with fully digital circuit”. In : *IEEE International Symposium on Electromagnetic Compatibility and IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC)*. 2018, p. 753–755. doi : 10.1109/ISEMC.2018.8393882 (cf. p. 17).

[77] N. Miura et al. “PLL to the rescue : a novel EM fault countermeasure”. In : *Proceedings of the 53rd Annual Design Automation Conference, DAC*. 2016, 90 :1–90 :6. doi : 10.1145/2897937.2898065 (cf. p. 17).

[78] N. Miura et S. Bhasin. “Attack sensing against EM leakage and injection”. In : *International SoC Design Conference (ISOCC)*. 2016, p. 201–202. doi : 10.1109/ISOCC.2016.7799857 (cf. p. 17).

[79] J. Breier, S. Bhasin et W. He. “An electromagnetic fault injection sensor using Hogge phase-detector”. In : *18th International Symposium on Quality Electronic Design (ISQED)*. 2017, p. 307–312. doi : 10.1109/ISQED.2017.7918333 (cf. p. 17).

[80] C-R. Hogge. “A self correcting clock recovery circuit”. In : *IEEE Transactions on Electron Devices* (1985), p. 2704–2706. doi : 10.1109/T-ED.1985.22402 (cf. p. 17).

[81] D. El-Baze, J. Rigaud et P. Maurine. “An Embedded Digital Sensor against EM and BB Fault Injection”. In : *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. 2016, p. 78–86. doi : 10.1109/FDTC.2016.14 (cf. p. 17).

[82] D. El-Baze, J. Rigaud et P. Maurine. “A fully-digital EM pulse detector”. In : *Design, Automation Test in Europe Conference Exhibition (DATE)*. 2016, p. 439–444 (cf. p. 17).

[83] C. Deshpande et al. “Employing dual-complementary flip-flops to detect EMFI attacks”. In : *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 2017, p. 109–114. doi : 10.1109/AsianHOST.2017.8354004 (cf. p. 17).

[84] A. Barenghi et al. “Countermeasures against fault attacks on software implemented AES : effectiveness and cost”. In : *Proceedings of the 5th Workshop on Embedded Systems Security, WESS 2010*. 2010 (cf. p. 17–20, 91–98, 101, 107–110).

[85] N. Moro et al. “Formal verification of a software countermeasure against instruction skip attacks”. In : *Journal of Cryptographic Engineering* (2014) (cf. p. 17–20, 91, 93, 98–100, 107–110).

[86] T. Barry, D. Couroussé et B. Robisson. “Compilation of a Countermeasure Against Instruction-Skip Fault Attacks”. In : *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems*. 2016, p. 1–6. doi : 10.1145/2858930.2858931 (cf. p. 18, 94).

[87] Y. Yao et P. Schaumont. “A Low-cost Function Call Protection Mechanism Against Instruction Skip Fault Attacks”. In : *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*. 2018, p. 55–64. doi : 10.1145/3266444.3266453 (cf. p. 18).

[88] C. Patrick et al. “Lightweight Fault Attack Resistance in Software Using Intra-instruction Redundancy”. In : *Selected Areas in Cryptography - SAC- 23rd International Conference*. 2016, p. 231–244. doi : 10.1007/978-3-319-69453-5_13 (cf. p. 18).

[89] B. Lac et al. “Thwarting Fault Attacks against Lightweight Cryptography using SIMD Instructions”. In : *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2018, p. 1–5. doi : 10.1109/ISCAS.2018.8351693 (cf. p. 18).

[90] N. Moro et al. “Experimental evaluation of two software countermeasures against fault attacks”. In : *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. 2014, p. 112–117. doi : 10.1109/HST.2014.6855580 (cf. p. 19).

[91] J-L. Danger et al. “CCFI-Cache : A Transparent and Flexible Hardware Protection for Code and Control-Flow Integrity”. In : *21st Euromicro Conference on Digital System Design (DSD)*. 2018, p. 529–536. doi : 10.1109/DSD.2018.00093 (cf. p. 19).

[92] B. Yuce et al. “A Secure Exception Mode for Fault-Attack-Resistant Processing”. In : 2019, p. 388–401. doi : 10.1109/TDSC.2018.2823767 (cf. p. 20).

[93] O. Trabelsi, L. Sauvage et J-L. Danger. “Characterization at Logical Level of Magnetic Injection Probes”. In : *2019 Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC)*. 2019, p. 625–628. doi : 10.23919/EMCTokyo.2019.8893692 (cf. p. 23).

[94] O. Trabelsi, L. Sauvage et J-L. Danger. “Impact of Intentional Electromagnetic Interference on Pure Combinational Logic”. In : *International Symposium on Electromagnetic Compatibility (EMC EUROPE)*. 2019, p. 398–403. doi : 10.1109/EMCEurope.2019.8871909 (cf. p. 23).

[95] Microsemi. *Microsemi Maker Board, “SmartFusion2 SoC FPGA”*. https://www.microsemi.com/existing-parts/parts/144012 (cf. p. 26).

[96] Microsemi. *IGLOO2 FPGA and SmartFusion2 SoC FPGA, DS0128”*. https://www.microsemi.com/document-portal/doc_download/132042-ds0128-igloo2-and-smartfusion2-datasheet (cf. p. 26).

[97] Side-channel Attack Standard Evaluation Board SASEBO. *SASEBO-G*. http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-g.html (cf. p. 26).

[98] Xilinx. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs :Complete Data Sheet, DS083*. https://www.xilinx.com/support/documentation/data_sheets/ds083.pdf (cf. p. 26).

[99] Side-channel Attack Standard Evaluation Board SASEBO. *SASEBO-W*. http://satoh.cs.uec.ac.jp/SASEBO/en/board/sasebo-w.html (cf. p. 26).

[100] Xilinx. *Spartan-6 FPGA Data Sheet :DC and Switching Characteristics, DS162*. https://www.xilinx.com/support/documentation/data_sheets/ds162.pdf (cf. p. 26).

[101] Langer EMV-Technik. *RF-B 3-2 H-Field Probe 30 MHz up to 3 GHz*. https://www.langer-emv.com/en/product/rf-passive-30-mhz-3-ghz/35/rf2-set-near-field-probes-30-mhz-up-to-3-ghz/272/rf-b-3-2-h-field-probe-30-mhz-up-to-3-ghz/15 (cf. p. 29).

[102] Langer EMV-Technik. *BS 05DB-h Magnetic Field Source*. https://www.langer-emv.com/en/product/accessory-eft-burst-generators-iec-61000-4-4/15/h2-set-field-sources/874/bs-05db-h-magnetic-field-source/60 (cf. p. 29, 30).

[103] Xilinx. *Spartan-6 FPGA Packaging and Pinouts, Product Specification, UG385*. https://www.xilinx.com/support/documentation/user_guides/ug385.pdf (cf. p. 33).

[104] Xilinx. *Xilinx Advanced Packaging, Chip Scale Packages - Quad Flat No-Lead - Plastic BGAs - Cavity-Down BGAs - Flip-Chip BGAs - Flip-Chip CCGAs - Pb-Free, PN0010951*. https://www.xilinx.com/publications/prod_mktg/pn0010951.pdf (cf. p. 33).

[105] O. Trabelsi, L. Sauvage et JL. Danger. “Characterization of Electromagnetic Fault Injection on 32-bit Microcontroller Instruction Buffer”. In : *IEEE Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. 2020 (cf. p. 45).

[106] STMicroelectronics. *UM1472 User manual Discovery kit with STM32F407VG MCU*. https://www.st.com/resource/en/user_manual/dm00039084-discovery-kit-with-stm32f407vg-mcu-stmicroelectronics.pdf. 2020 (cf. p. 53).

[107] STMicroelectronics. *RM0090 Reference manual advanced Arm-based 32-bit MCUs, Rev 18*. https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf. 2019 (cf. p. 53, 54, 59).

[108] Atmel. *ARM-based Embedded MCUs ATSAM4C-EK USER GUIDE, 11251A*. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel_11251_SmartEnergy_ATSAM4C-EK-User_Guide_SAM4C8-SAM4C16_User-Guide.pdf. 2016 (cf. p. 53).

[109] Arduino. *Arduino MKR ZERO*. https://store.arduino.cc/arduino-mkr-zero-i2s-bus-sd-for-sound-music-digital-audio-data. 2020 (cf. p. 53, 102).

[110] Atmel. *SAM4C Series SMART ARM-based Flash MCU DATASHEET, 11102G*. https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11102-32-bit-Cortex-M4-Microcontroller-SAM4C32-SAM4C16-SAM4C8-SAM4C4_Datasheet.pdf. 2016 (cf. p. 53, 80, 83).

[111] Atmel. *SAM D21/DA1 Family, DS40001882F*. https://ww1.microchip.com/downloads/en/DeviceDoc/SAM_D21_DA1_Family_DataSheet_DS40001882F.pdf. 2020 (cf. p. 53, 102).

[112] ARM. *ARMv7-M Architecture Reference Manual, E.d.* https://developer.arm.com/documentation/ddi0403/latest/. 2018 (cf. p. 62).

[113] VT Khuat et al. "Multiple and Reproducible Fault Models on Micro-controller using Electromagnetic Fault Injection". In : *Joint International Symposium on Electromagnetic Compatibility, Signal & Power Integrity, and EMC EUROPE*. 2021 (cf. p. 102, 114).

ECOLE
DOCTORALE

Titre : Méthodes pour la modélisation des injections de fautes électromagnétiques

Mots clés : Électromagnétique, injection de faute, contre-mesure, sécurité

Résumé : Les attaques par injection de faute représentent une menace considérable pour les systèmes cyber-physiques. Dès lors, la protection contre ces attaques est une nécessité pour assurer un haut niveau de sécurité dans les applications sensibles comme l'internet des objets, les téléphones mobiles ou encore les voitures connectées. En matière de méthodes d'injection de faute, celle par interférence électromagnétique s'est vu être une source de perturbation efficace, en étant moins intrusive et avec une configuration à faible coût. Outre l'ajustement des paramètres d'injection, l'efficacité de cette méthode réside dans le choix de la sonde qui génère le rayonnement électromagnétique.
La première partie de la thèse aborde la question de l'efficacité des sondes magnétiques, en mettant l'accent sur l'implication de leurs propriétés. Afin de comparer les sondes, nous proposons d'observer l'impact des impulsions électromagnétiques au niveau logique, sur des cibles particulières de type FPGA. La caractérisation est aussi établie suivant la variation des paramètres d'injection comme l'amplitude et la polarité de l'impulsion, le nombre d'impulsions ou encore l'instant de l'injection.
L'étude est par la suite étendue au niveau logiciel sur des cibles de type microcontrôleur. L'objet de la seconde contribution consiste à présenter une démarche d'analyse, basée sur trois méthodes génériques, qui servent à déterminer les vulnérabilités des microcontrôleurs sur les instructions et les données. Ces méthodes portent sur l'identification des éléments vulnérables au niveau architecture, l'analyse des modèles de faute au niveau bit, et enfin la définition de l'état des fautes, à savoir transitoire ou semi-persistent. Le travail de dresser les modèles de faute, ainsi que le nombre d'instructions ou données impactées, est un jalon important pour la conception de contre-mesures plus robustes.
Les résultats de cette caractérisation nous ont permis d'évaluer des contre-mesures au niveau instruction, dont le mécanisme le plus répandu se résume à appliquer une redondance dans l'exécution du programme à protéger. Ce type de contre-mesure est formulé sur l'hypothèse qu'une injection de faute équivaut un seul saut d'instruction. Vis-à-vis de nos observations, ces contre-mesures basées sur de la duplication au niveau instructions présentent des vulnérabilités, que nous identifions, puis corrigeons.

Title : Methods for modeling of electromagnetic fault injection

Keywords : Electromagnetic, fault injection, counter-measure, security

Abstract : Fault injection attacks represent a considerable threat to cyber-physical systems. Therefore, protection against these attacks is required to ensure a high level of security in sensitive applications such as the Internet of Things, smart devices or connected cars. In terms of fault injection methods, electromagnetic interference has proven to be an effective source of disruption, being less intrusive and with a low cost setup. Besides the adjustment of the injection parameters, the effectiveness of this attack mean lies in the choice of the probe that generates the electromagnetic radiation.
The first part of the thesis addresses the question of the efficiency of magnetic probes, with a focus on their properties. In order to compare the probes, we propose to observe the impact of electromagnetic pulses at the logic level, on particular targets such as FPGA. The characterization is also established according to the variation of the injection parameters such as the amplitude and the polarity of the pulse, the number of pulses or the injection time.
The characterization is then extended to the software level on microcontroller targets. The purpose of the second contribution is to present an analysis approach, based on three generic methods, which are used to determine the vulnerabilities of microcontrollers with respect to instructions or data. These methods concern the identification of vulnerable elements at the architecture level, the analysis of fault models at the bit level and finally the definition of the temporal fault status, i.e. transient or semi-persistent. Establishing the fault patterns, as well as the number of the impacted instructions or data, is an important milestone for the design of more robust countermeasures.
The latter results allowed us to evaluate countermeasures at the instruction level, of which the most common mechanism is to apply redundancy in the execution of the program to be protected. However, this type of countermeasure is based on the assumption that a fault injection imply a single instruction skip. With respect to our observations, these countermeasures based on instruction-level duplication present vulnerabilities, which we identify and then correct.

Institut Polytechnique de Paris
91120 Palaiseau, France